Assignment #2

Join Algorithm

DBMS

2021045796

Kimdogyeom

김도겸

0. Set

1) 과제 제출물의 파일 주소를 수정하였습니다. 채점 시 확인 부탁드립니다.

ex)

-output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/case1/output1.csv");

-block[0].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/case1/name_age/" + to_string(current_block[0]) + ".csv");

-block[1].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/case1/name_salary/" + to_string(current_block[1]) + ".csv");

2) Case 2와 Case 3에서 Hash Join을 사용합니다. 따라서 Case 2를 실행하고 Case 3를 실행하기 전에 buckets 폴더를 초기화시켜주어야 합니다.

1. Case 1

1) Choosing Algorithm

Case 1의 첫 번째 relation의 attribute는 name과 age로 구성되어있습니다.

두 번째 relation의 attribute는 name과 salary로 구성되어있습니다.

두 relation의 공통 속성으로 name이 존재하고, 두 tuple 모두 정렬되어있으므로 Merge-join Algorithm을 사용하여 Join을 진행하려고 합니다. Nested-loop join에 비해 반복 횟수를 확연하게 감소시킬 수 있으며, 정렬이 되어있으므로 굳이 Hash Join을 사용해야 할 이유가 없습니다.

2) Trouble Shooting

-How to check Pointer

Merge-join Algorithm을 사용하기 위해서는 현재 진행 정도에 대해 Pointer를 이용하여 표시해야합니다. 이때 주어진 current_block 배열을 이용하여 첫 번째 relation의 tuple과 두 번째 relation의 tuple의 순서를 정리하여 이를 대체하였습니다.

-다음 csv 불러오기

block[0]에 첫 번째 relation을 담은 csv를, block[1]에 두 번째 relation을 담은 csv를 불러왔습니다. 각각의 join 연산 이후 다음 줄을 불러오는데, "!block[0].eof() && !block[1].eof()"을 만족하는 동안 작업을 반복하며, 해당 조건이 깨졌을 때 block을 닫고 다음 csv를 호출하였습니다.

-Merge-join Algorithm 구현

```
while(!block[0].eof() &&!block[1].eof()) {
        temp0.set_name_age(buffer[0]);
        temp1.set_name_salary(buffer[1]);
        //cout <<"check2" <<endl;
        if(temp0.name==temp1.name) {
```

```
        output<<make_tuple(temp0.name,temp0.age,temp1.salary);
        cout<<temp0.name<<" "<<temp0.age<<" "<<temp1.salary<<endl;
        getline(block[1],buffer[1]);
        //cout <<"a : " <<a <<endl;
        //a++;
      }
      elseif(temp0.name<temp1.name) {
        getline(block[0],buffer[0]);
        //cout <<"check3" <<endl;
      }
      else {
        getline(block[1],buffer[1]);
        //cout <<"check4" <<endl;
      }
    }
    //cout <<"check5" <<endl;
    block[0].close();
    block[1].close();
    //cout <<"check6" <<endl;
```

temp0에 첫 번째 relation의 name과 age를, temp1에 두 번째 relation의 name과 salary의 class를 지정하였습니다. temp0.name이 temp1.name과 같을 때는 output1.csv에 temp0.name, temp0.age, temp1.salary를 입력하고 두 번째 relation을 담당하는 block[1]의 다음 tuple을 불러왔습니다. temp0.name < temp1.name인 경우에는 첫 번째 relation을 담당하는 block[0]의 다음 tuple을 불러오고, 나머지 경우에는 block[1]의 다음 tuple을 불러왔습니다.

-종료 시점

current_block에 작업 횟수를 저장하여 current_block[0]과 current_block[1]이 둘 다 1000을 넘어갈 때 작업을 종료하였습니다.

3) Open 횟수

-Relation이 정렬되어 있어서 2001번의 open이 발생할 것이라고 생각합니다.

2. Case 2

1) Choosing Algorithm

-Case 2의 첫 번째 relation의 attribute는 name과 age로 구성되어있습니다.

두 번째 relation의 attribute는 name과 salary로 구성되어있습니다.

두 relation의 공통 속성으로 name이 존재하지만, name을 기준으로 정렬되어있지 않아 Case 1과 같은 방식으로 해결하기 어려울 것 같습니다. 때문에 Nested-loop join Algorithm과 Hash join Algorithm을 고민하였고, Open횟수를 줄이기 위해 Hash join

Algorithm을 사용하였습니다.

2) Trouble Shooting

-Hash Memory Data 만들기

저는 해쉬 분할을 진행하는 과정에서 이름의 첫 글자가 a로 시작하는지 b로 시작하는지... 등등에 따라서 분류하였습니다. map과 같은 기능을 사용할 수 없었기 때문에 분류하여 Hash memory로 넣을 때 학생들의 이름을 기준으로 분류할 수 밖에 없었습니다. 때문에 a로 시작하는 학생들은 첫 번째 relation의 경우에는 a.csv에, 두 번째 relation의 경우에는 aa.csv에 입력하였습니다. 이러한 방식으로 a.csv와 aa.csv만 비교하면 되도록 만들었습니다.

-반복문 문제

a.csv와 aa.csv, b.csv와 같은 방식으로 만들다 보니 작업을 진행하면서 반복문으로 csv 파일을 호출할 수 없었습니다. 때문에 a.csv와 aa.csv를 비교하는 작업부터 j.csv와 jj.csv를 비교하는 작업을 반복문 없이 수행합니다.

-Hash Memory Data Code

```
for(inti=0; i<1000; i++) {

  block[0].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/case2/name_age/"+to_string(i) +".csv");
    openNum++;
    if(block[0].fail())
    {
      cout<<"file opening fail.\n";
    }
    for(intj=0; j<10; j++) {
      getline(block[0],buffer[0]);
      temp0.set_name_age(buffer[0]);
      buffer[1] =temp0.name.substr(0,1);

   output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buckets/"+buffer[1] +".csv",ios::app);
      openNum++;
      output<<temp0.name<<','<<temp0.age<<'\n';
      output.close();
    }
    //cout <<"first file " <<i <<" is done.\n";
    block[0].close();
  }
  for(inti=0; i<1000; i++) {
```

```
  block[0].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/case2/name_salary/"+to_string(i) +".csv");
    openNum++;
    if(block[0].fail())
    {
      cout<<"file opening fail.\n";
    }
    for(intj=0; j<10; j++) {
      getline(block[0],buffer[0]);
      temp1.set_name_salary(buffer[0]);
      buffer[1] =temp1.name.substr(0,1);

    output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buckets/"+buffer[1] +buffer[1] +".csv",ios::app);
      openNum++;
      output<<temp1.name<<','<<temp1.salary<<'\n';
      output.close();
    }
    //cout <<"second file " <<i <<" is done.\n";
    block[0].close();
  }
```

하나의 relation에 1000개의 csv 파일이 존재합니다. 하나의 csv 파일에는 10개의 tuple이 들어있으므로 1000번의 반복문 안에 10번의 반복문을 입력하여 각각의 tuple의 name의 첫 자리에 따라 해당하는 csv 파일을 open하여 값을 입력해주었습니다.

relation이 두 개이므로 해당 작업을 2번 반복하였습니다.

해당 과정에서 22000번의 open이 실행되었습니다.

-Join 구현

```
//a.csv

 block[0].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buckets/a.csv");
  openNum++;
  if(block[0].fail())
  {
    cout<<"first block file opening fail.\n";
  }
  for(inti=0; i<1000; i++) {
    getline(block[0],buffer[0]);
```

```
    temp0.set_name_age(buffer[0]);

  block[1].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buckets/aa.csv");
    openNum++;
    if(block[1].fail())
    {
      cout<<"second block file opening fail.\n";
    }
    while(true) {
      getline(block[1],buffer[1]);
      temp1.set_name_salary(buffer[1]);
      if(temp0.name==temp1.name) {
        output<<make_tuple(temp0.name,temp0.age,temp1.salary);
        //cout <<make_tuple(temp0.name, temp0.age, temp1.salary);
        block[1].close();
        break;
      }
    }
  }
  block[0].close();
```

Partition된 memory를 바탕으로 Join을 진행해주었습니다. a.csv와 aa.csv에 들어있는 사람은 일치하지만, 같은 방식으로 정렬되어있지 않으므로 temp0.name == temp1.name이 될 때 까지 getline을 진행해주었습니다. 위의 코드는 a.csv와 aa.csv의 Join 과정이며, 해당 작업을 j.csv와 jj.csv의 Join을 진행할 때 까지 반복합니다.

a.csv를 여는 횟수 1번, aa.csv를 여는 횟수 1000번, 총 1001번의 open을 사용합니다. a부터 j까지 작업을 진행하므로 해당 작업에서 발생하는 open의 횟수는 10010번입니다.

-종료 시점

j.csv와 jj.csv의 Join이 완료되는 시점에 종료합니다.

3) Open 횟수

-Partition 과정에서 22000번, Join 과정에서 10010번, output2가 2번 Open 함수를 사용하여 총 32012번의 Open이 발생합니다.

3. Case 3

1) Choosing Algorithm

-Case 3은 총 3개의 relation이 존재합니다. 학생의 이름과 학번을 가진 name_number와 학생의 이름과 1학기 성적을 가진 name_grade1, 학생의 이름과 2학기 성적을 가진 name_grade2가 있습니다. 2학기에 1학기에 비해 성적이 오른 과목이 2개 이상인 학생을 찾

아야하고, 모든 data가 무작위로 저장되어있기 때문에 Hash Join을 이용하기로 하였습니다. 성적 향상이 발생한 학생을 temp.csv에 저장하고, name_number도 hash partition 해준 뒤 Join해주었습니다.

2) Trouble Shooting

-name_grade1과 name_grade2는 Case 2 작업을 할 때와 같은 방식으로 Partition 해주었습니다. a.csv와 aa.csv를 Join하여 temp0.subject > temp1.subject를 만족하는 경우가 2개 이상일 경우에 temp.csv에 저장해주었습니다. 해당 작업을 j.csv와 jj.csv를 Join할 때 까지 반복해주었습니다.

```
for(inti=0; i<1000; i++) {

  block[0].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/case3/name_grade1/"+to_string(i) +".csv");
    openNum++;
    if(block[0].fail())
    {
      cout<<"file opening fail.\n";
    }
    for(intj=0; j<10; j++) {
      getline(block[0],buffer[0]);
      temp0.set_grade(buffer[0]);
      buffer[1] =temp0.student_name.substr(0,1);

    output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buckets/"+buffer[1] +".csv",ios::app);
      openNum++;
      output<<buffer[0] <<"\n";
      output.close();
    }

    block[0].close();
  }
```

-아래의 내용은 a.csv와 aa.csv를 비교하는 과정입니다.

```
output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buckets/temp.csv");
  openNum++;
  //a.csv

 block[0].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buckets
```

```
/a.csv");
   openNum++;
   if(block[0].fail())
   {
    cout<<"first block file opening fail.\n";
   }
   for(inti=0; i<1000; i++) {
    getline(block[0],buffer[0]);
    temp0.set_grade(buffer[0]);

  block[1].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/bucket
s/aa.csv");
    openNum++;
    if(block[1].fail())
    {
     cout<<"second block file opening fail.\n";
    }
    while(true) {
     getline(block[1],buffer[1]);
     temp1.set_grade(buffer[1]);
     if(temp0.student_name==temp1.student_name) {
      intcheck=0;
      if(temp0.korean>temp1.korean) check++;
      if(temp0.math>temp1.math) check++;
      if(temp0.english>temp1.english) check++;
      if(temp0.science>temp1.science) check++;
      if(temp0.social>temp1.social) check++;
      if(temp0.history>temp1.history) check++;
      if(check>=2) {
       output<<buffer[0] <<"\n";
       total++;
       //cout <<buffer[0] <<"\n";
      }
      block[1].close();
      break;
     }
    }
   }
   block[0].close();
```

-위의 작업 이후 name_number relation도 Partition 해주었습니다. a로 시작하는 학생을

buckets에 1.csv로, j로 시작하는 학생을 buckets에 10.csv로 저장하였습니다.

```
for(inti=0; i<1000; i++) {

  block[0].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/case3/name_number/"+to_string(i) +".csv");
    openNum++;
    if(block[0].fail())
    {
     cout<<"file opening fail.\n";
    }
    for(intj=0; j<10; j++) {
     getline(block[0],buffer[0]);
     temp2.set_number(buffer[0]);
     buffer[1] =temp2.student_name.substr(0,1);
     if(buffer[1] =="a") {

     output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buckets/1.csv",ios::app);
       openNum++;
       output<<temp2.student_name<<','<<temp2.student_number<<'\n';
       output.close();
     }
     elseif(buffer[1] =="b") {

     output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buckets/2.csv",ios::app);
       openNum++;
       output<<temp2.student_name<<','<<temp2.student_number<<'\n';
       output.close();
     }
     elseif(buffer[1] =="c") {

     output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buckets/3.csv",ios::app);
       openNum++;
       output<<temp2.student_name<<','<<temp2.student_number<<'\n';
       output.close();
     }
     elseif(buffer[1] =="d") {
```

```
        output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buck
ets/4.csv",ios::app);
          openNum++;
          output<<temp2.student_name<<','<<temp2.student_number<<'\n';
          output.close();
        }
        elseif(buffer[1] =="e") {

        output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buck
ets/5.csv",ios::app);
          openNum++;
          output<<temp2.student_name<<','<<temp2.student_number<<'\n';
          output.close();
        }
        elseif(buffer[1] =="f") {

        output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buck
ets/6.csv",ios::app);
          openNum++;
          output<<temp2.student_name<<','<<temp2.student_number<<'\n';
          output.close();
        }
        elseif(buffer[1] =="g") {

        output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buck
ets/7.csv",ios::app);
          openNum++;
          output<<temp2.student_name<<','<<temp2.student_number<<'\n';
          output.close();
        }
        elseif(buffer[1] =="h") {

        output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buck
ets/8.csv",ios::app);
          openNum++;
          output<<temp2.student_name<<','<<temp2.student_number<<'\n';
          output.close();
        }
        elseif(buffer[1] =="i") {
```

```
      output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buck
ets/9.csv",ios::app);
        openNum++;
        output<<temp2.student_name<<','<<temp2.student_number<<'\n';
        output.close();
      }
      elseif(buffer[1] =="j") {

      output.open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buck
ets/10.csv",ios::app);
        openNum++;
        output<<temp2.student_name<<','<<temp2.student_number<<'\n';
        output.close();
      }
    }
    //cout <<"first file " <<i <<" is done.\n";
    block[0].close();
  }
```

-반복문으로 1.csv부터 10.csv까지 temp.csv와 join하여 해당하는 학생을 선별하였습니다. temp.csv에 존재하는 학생을 output3.csv에 추가하였습니다.

```
for(inti=1; i<=10; i++) {


  block[0].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/bucket
s/"+to_string(i) +".csv");
    openNum++;
    if(block[0].fail())
    {
      cout<<"first file opening fail.\n";
    }
    for(intj=0; j<1000; j++) {
      getline(block[0],buffer[0]);
      temp2.set_number(buffer[0]);

    block[1].open("C:/Users/admin/Language/C_Lang/2022_DBMS_ASS2/ass2/buck
ets/temp.csv");
      openNum++;
      if(block[1].fail())
      {
        cout<<"second file opening fail.\n";
```

```
    }
    for(intk=0; k<total; k++) {
     getline(block[1],buffer[1]);
     temp0.set_grade(buffer[1]);
     //buffer[1] = student_name
     if(temp0.student_name==temp2.student_name) {
      output<<temp2.student_name<<','<<temp2.student_number<<'\n';
      //cout <<temp2.student_name <<',' <<temp2.student_number <<'\n';
      break;
     }
    }
    block[1].close();
   }

  block[0].close();
  }
```

3) Open 횟수

-Case 3의 조건을 만족하는 학생을 8330명 찾을 수 있었습니다.

-Open의 발생 횟수는 총 53023번이며, Hash Partition 과정에서 33000번, 두 grade relation을 join하는 과정에서 10010번, grade의 비교 자료를 저장해둔 temp.csv와 name_number를 hash partition해준 자료를 join하는 과정에서 10010번, output 호출 과정에서 3번 발생하였습니다.

-위의 방식대로 했을 경우 Nested-loop Join 방식으로 진행되어 10010번의 Open 함수 호출이 발생합니다. inner relation과 outer relation을 바꾸어 진행했다면 8331번의 Open 함수 호출로 변경할 수 있지 않았을까하는 아쉬움도 남습니다.

4. etc

1) Open 함수를 최소로 호출하는 방법

-처음에는 기준에 따라서 Partition을 해주는 방법이 가장 효율적이라고 생각하였습니다. 조건을 세밀화 할수록 연산 속도가 빨라질 것이라는 생각 때문이었습니다. 하지만 과제를 끝내고 생각해보니 Open 함수를 최소로 호출하는 것을 목표로 한다면 조건 없이 하나의 Hash Memory에 적용하여 무식한 반복문을 돌리는 것이 Open 함수를 최소로 호출하는데 더 효과적이지 않을까하는 생각이 들었습니다.