CENG351 – Data Management and File Structures

Programming Assignment 2

BLOCK NESTED LOOP (BNL) JOIN

Due Date: **6th of January**

1 Introduction

In this homework you will simulate join operation of the relational database management systems (RDBMS). What you will do is to join two relations which do not fit into memory on a single attribute and create a merged relation with qualified tuples (records). At the end of this homework you will be familiar with nested loop join operation and how to improve performance of it by making use of memory blocks. Furthermore, you will get your hands dirty by reading from binary files and extract information byte by byte. You will also write to binary files in the scope of this homework.

2 BNL Join

BNL Join is a modified version of the Nested Loop Join algorithm. While NL Join algorithm reads entire inner relation for each tuple of the outer relation, BNL Join algorithm reads it for each block of the outer relation. Hence, it decreases number of reads made for the inner relation dramatically.

```
foreach tuple r in R
  foreach tuple s in S
    if r.x = s.x
      output <r,s>
```

Figure 1: Nested Loop Join algorithm

```
foreach block B_r of R
  for each page P_s of S
    if r == s where r ∈ B_r, s ∈ P_s
      output <r,s>
```

Figure 2: Block Nested Loop algorithm

Let relation R has M pages, S has N pages and each page contains p tuples. When Nested Loop algorithm is applied, the operation will involve $M+N*(p*M)$ disc I/Os (Relation R will be read only once. Relation S will be read $p*M$ times. Reading entire relations costs M and N I/Os, respectively.). When Blocked Nested Loop algorithm is applied, the operation will involve $M+N(\frac{M}{B-2})$ disc I/Os where B is the size of the memory in terms of number of pages (One page is used to hold pages of relation S and one page is used for the output buffer.). For detailed information about NL and BNL algorithms you can refer to course notes.

3 Tasks

You will implement a tool which joins two relations which are stored on binary files on a selected attribute by using BNL algorithm and writes the result to a binary file again.

3.1 Execution

Your program will be executed with 5 arguments which are name of the file which stores outer relation, name of the file which stores inner relation, name of the result file, memory size in terms of number of pages and the name of the attribute that join operation will be done on, respectively.

Your programs will be executed with memory limitation to simulate the join operation properly. So do not try to read all tuples into memory. You should use exactly given size of memory to store pages of relations. Do not worry, you will be provided slightly more memory than the given size for use of internal calculations.

3.2 Join

You will use Block Nested Loop algorithm for the join operation. Size of the memory block to be used in terms of number of pages will be given as command line argument. You will use one page size buffer for inner relation, one page size buffer for output and remaining memory for the outer relation.

Order of the join operation is crucial for evaluation. Assume that B_i is the *i*th block of the outer relation, $P(R)_i$ is the *i*th page of outer and $P(S)_i$ is the *i*th page of the inner relation, $T(R_k)_i$ is the *i*th tuple of the *k*th page of the outer and $T(S_k)_i$ is the *i*th tuple of the *k*th page of the inner relation. Join operation will be done in the following order:

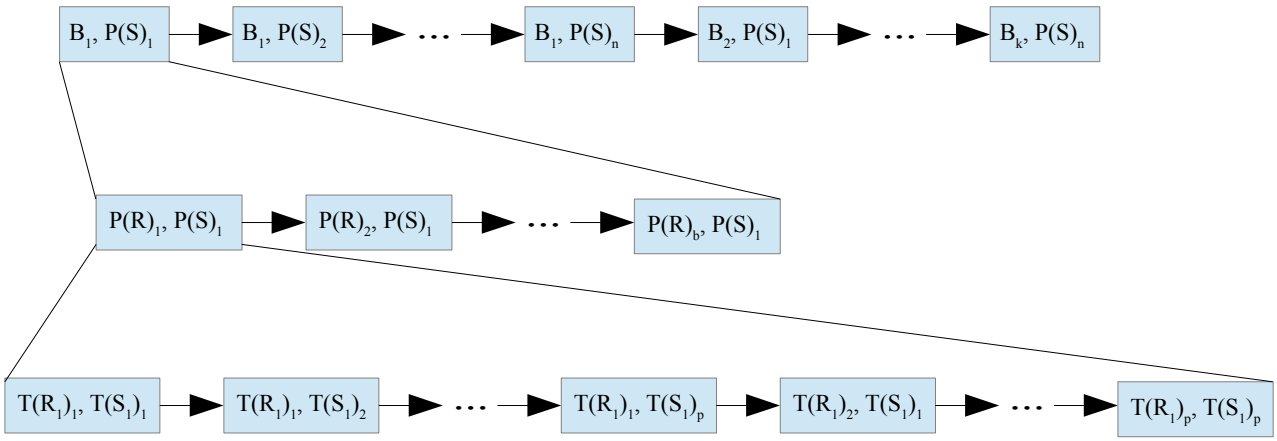

Figure 3: Order of the join operation

where outer relation has *k* blocks, inner relation has *n* pages Block for outer relation can hold *b* pages and page size is *p*.

Normally, when a block is read to the memory hash indexes are built on the joined attribute for fast comparisons. But, in our case, because orders of the pairs are crucial, do not create any hash index and do not sort tuples on the memory.

Relations may have different number and types of attributes. Attribute name for join condition to be applied on will also be given as command line argument. Both relation will have a attribute with that name and the same type. All information related to attribute names and types is given in the header which is included in binary relation file.

3.3 Writing Results

You will write tuples satisfying the join condition into a binary result file. This file will not have any header, it will only contain qualified tuples consecutively. For example, r_1 and s_1 are the tuples satisfying the join condition from inner and outer relations respectively, then you should write r_1s_1 to the result file with no extra data between tuples. If r_2 and s_2 are the next qualified tuples, then output file should contain $r_1s_1r_2s_2$ and so on...

Size of the tuples in the inner and the outer relations may be different. You should write them to the result file in binary form as they are stored in relation file. Order of the pairs in the result file is crucial. Therefore, you should follow the order given above while joining the relations.

You should make use of output buffer while writing results into the result file. In other words, in order to decrease seek time, you should write the results into the buffer first. After the buffer is full, you will flush it into the result file and reuse it.

3.4 Logs

You will inform me during the join process by printing logs to the standard output. Every time you read a block of pages from the outer relation you will print "*Pages .. - .. read*" (without quotation marks). For example, assume your program is executed with 50 pages of memory, after the first read from the outer relation you should print "Pages 1 – 48 read" (Because 48 pages of memory will be available for the outer relation). Be careful about that total number of pages in the outer relation may not be multiple of the available memory size.

After you finish joining that block of the outer relation, before reading the next block, you should print the number of compared records and total number of joined pairs in that block in the format *".. compared .. joined"*. For example, assume there are *p* records in the first block read from the outer relation and when you join them with the inner relation *q* pairs satisfy the join condition. Then you should print *"p compared q joined"*. Be careful about that *q* may be greater than *p*.

4 File Structures

All files will be binary files with little endian byte ordering. Both inner and outer relation files will have the following structure:

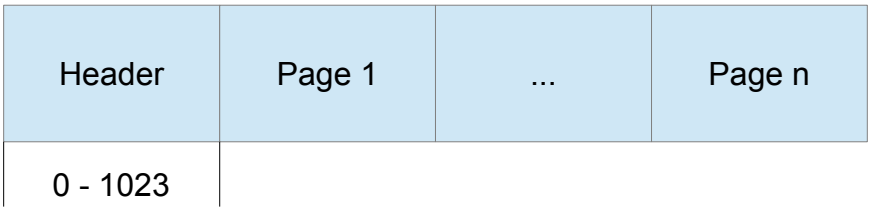

Figure 4: General structure of binary relation files

Relation files starts with a 1024-byte header which contains necessary information about the relation. Header is structured as:

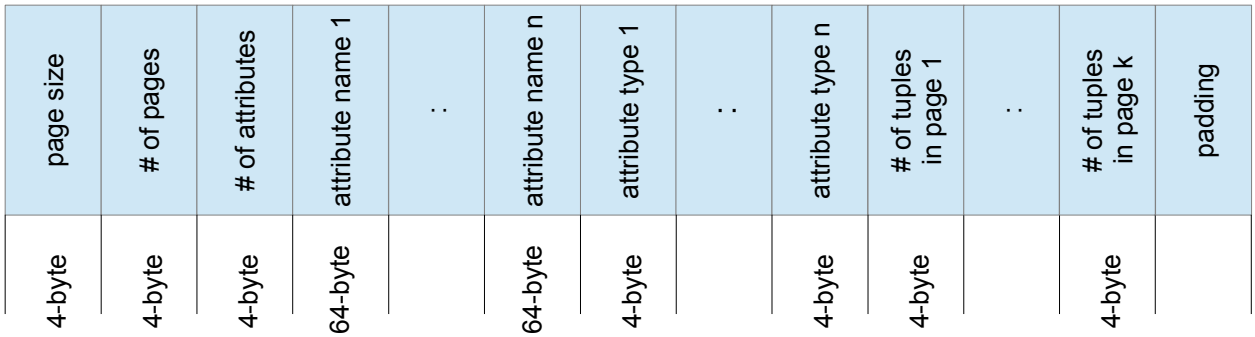

Figure 5: Structure of the Header

- *page size*, *# of pages*, *# of attributes* and *# of records in page i* fields contain 4-byte integer.
- attribute *name* fields contain null-ended strings (max 64-byte) which are name of the attributes of the tuples consecutively.
- *attribute type* fields contain type of the attributes of the records consecutively. Each type field consists of two 2-byte integers (short). First integer contains type information, second one contains length information. There are three possible types which are 4-byte integer, 4-byte floating-point number(float) and variable length string. Type values are 1, 2 and 3 for integer, floating-point and string types, respectively. If the type is string then length of the

string is stored in the second integer. Otherwise, the second integer stores meaningless data. Be careful about that you should read *attribute type* field as two separate integers because byte ordering is little endian. If you read them as 4-byte integer and then split you encounter wrong values.

- *# of tuples in page i* fields contain number of tuples in each page, consecutively. These number may vary from page to page because of deleted tuples.
- Pages consist of consecutive tuples without any delimiters and extra character between them. Because size of tuples in a relation can be calculated using size of attributes, you can extract tuples in a page easily. There will be '&' character at the end of tuples in a page. Because of deleted tuples, every page may not be full of tuples.
- Tuples consist of consecutive attributes without any delimiters and extra character between them. Because size of the attributes are given in the header each attribute of a tuple can be extracted easily.

Result file will consist of consecutive pair of tuples. Each pair consists of one tuple from the outer relation and one tuple from the inner relation, respectively. There will be no delimiter or extra character between tuples or pairs.

5 Specifications

- You will write your codes in Java. You are NOT allowed to use third party libraries. So, do NOT submit any jar file with your homework.
- Evaluation will be done on Inek machines. Please test your homework on inek machines before submitting. Different Java versions may cause your homework not compile.
- You will create the result file with the exact name given. Do not add any extension.
- Please use flush after any output (standard output or file output) for buffers to be flushed.
- I will provide 10% more memory than one given as argument to be used for internal calculations.
- Page size will have the same value for both relations for proper execution.
- Page indices start at 1. In other words, 'page 1' is the first page (not 'page 0').
- Size of the relation files will have the value $1024 + k * pagesize$ where k is a positive integer.
- All comparisons should be done case-sensitive
- COW is the official platform for updates and clarification. I mean, you are responsible for the updates announced on COW.

- Everything you submitted must be your own work. Do not copy any code piece from each other or any third party source (internet included).
- Blackbox test method will be used for evaluation. Which means, I will not inspect your codes for any mistake. Test script will evaluate your homeworks. So, you should obey I/O specifications strictly for correct evaluation.

6 Submittion

You will submit all of your *.class* files and a *build.xml* file in compressed file named "hw2.tar.gz".

Your homeworks will be evaluated using the following command chain:

$ tar xvf hw2.tar.gz

$ ant init

$ ant compile

$ ant run -Dor=<file1> -Dir=<file2> -Drf=(file3) -Dms=<memory size> -Dan=<attribute>

where <file1>, <file2> and <file3> are the file names for the outer relation, the inner relation and the result file, respectively. <memory_size> is the size of the block in terms of number of pages and <attribute> is the name of the attribute on which BNL join will be applied.

Note that if hw2.tar.gz extracts *build.xml* into a directory, commands given above will fail to execute. An example "build.xml" file which is compatible with the commands given above can be found in *Appendices* part. Check the link in the *Useful Links* part for detailed information about *ant project*.

Kolay gelsin:)

Emres

Appendices

A Example *build.xml* File

```
<project name="ceng351-phw2"  basedir=".">
    <property name="src.dir" value="src"/>
    <property name="build.dir" value="build"/>

    <target name="init">
      <mkdir dir="${build.dir}" />
```

```
    </target>
    <target name="clean" >
       <delete dir="${build.dir}"/>
    </target>

    <target name="compile" depends="init">
      <echo message="Compiling...." />
      <javac srcdir="${src.dir}" destdir="${build.dir}"  />
    </target>

    <target name="run">
      <java fork="yes"
            className="ceng351.phw2.ExternalSort">
            <arg line="${filename}" />
            <arg line="${memory_size}" />
            <arg line="${k}" />
      </java>
    </target>
</project>
```

B Useful Links

- Java Tutorial: http://docs.oracle.com/javase/tutorial/
- Ant Project: http://ant.apache.org
- Binary File Operations with Java: http://www.javapractices.com/topic/TopicAction.do?Id=245