University of Edinburgh
School of Informatics
INF11011 - Advanced Databases (Spring 2019)

Programming Assignment 2

Due: **Tuesday, 26 March 2019 at 11:59pm**

IMPORTANT:

- **Plagiarism:** Every student has to work **individually** on this assignment.

  All of the code for this project must be your own. You may not copy source code from other students or other sources that you find on the web. You may not host your code on a public code repository. Plagiarism **will not** be tolerated.

  See the University's page on Academic Misconduct for additional information.

Project: Joins

For this assignment, you are expected to implement two standard join algorithms and evaluate their relative performance. Their description is given in the lecture material and can also be found in the Ramakrishnan & Gehrke textbook.

Background

Let us quickly review some of the Minibase components you will need for this assignment. In Minibase, a relation is implemented as a heap file, which is a collection of records. Records can be inserted or deleted from a heap file, and each record is uniquely identified by a record id. A scan is the interface used to access records in a heap file, one by one.

In this assignment, you will need the methods provided in the following two classes:

- `HeapFile`: implements a heap file
- `Scan`: scan interface to a heap file

The Join Algorithms

1. **Tuple-At-A-Time Nested Loops Join**

   Start with this one, it is the simplest (and also least efficient) join algorithm.

2. **Block Nested Loop Join**

   Since Minibase does not provide page-by-page access into a relation stored in a heap file (i.e., you can only retrieve records one at a time), you will simulate a "block" with an array storing the records. The parameter B represents the size of the array storing the outer relation expressed in the number of tuples of the outer relation.

   The pseudocode for this join is:

   ```
   For each block b in R
     For each tuple s in S
       For each tuple r in b
         If Match r with s is successful then
           Insert (r,s) into the result relation
   ```

   Compare the performance of the "Block nested loop" join for various block sizes.

Simplifications

- You can assume that all records are of fixed length.
- You can assume that we only perform joins on integer fields.

Programming Assignment 2 continues. . .

Performance Comparison

- Compare the relative performance of the two join algorithms. Record the times taken for each algorithm to run, and the number of page misses.
- Study the effect of the buffer pool size on the algorithms by changing the buffer pool size (`NUM_OF_BUF_PAGES` in *main.cpp*).
- Study the effect of the relation size on the algorithms by changing the number of records in the given `Employee` and `Project` relations (`NUM_OF_REC_IN_R` and `NUM_OF_REC_IN_S` in *join.h*).
- Study the effect of swapping the outer and inner relations.
- Submit a report containing documentation, tables, and graphs of these statistics, together with an analysis of the results. Note that your analysis must be thorough, and cover a wide range of buffer pool sizes and relation sizes. The report should not have more than 5 pages.

Collecting Statistics

You should let your algorithm run several times and report the average (wall-clock, not user) running time for each result reported. For every configuration and every join algorithm, print out the average running time and the number of buffer pool page misses.

Source Code

You are given an archive which contains the following files:

- *join.cpp*, *join.h*: utility functions useful for writing join algorithms.
- *relation.cpp*, *relation.h*: functions to create test relations.
- *blockjoin.cpp*, *tuplejoin.cpp*: each file contains a skeleton for implementing a particular join method.
- *main.cpp*: main program.
- *include*: subdirectory of all *.h* files needed.

You should write your code in *blockjoin.cpp*, *tuplejoin.cpp*, and *main.cpp*. The functions in *join.cpp* and *relation.cpp* will be useful for writing your join methods and for debugging. The method `SortFile` in *join.cpp* is particularly useful as an example of how to use `HeapFile`, `Scan`, `BTreeFile`, and `BTreeFileScan` (the last two are not used in this project).

Programming Assignment 2 continues...

Hints

- Study the `Scan` class. Learn how to access records in a heap file. Study the `SortFile` method in *join.cpp*.
- Use a small numbers of records (by changing the constants in *join.h*) for testing.
- A rough estimate: each join algorithm should be around 100-150 lines of code.
- You are free to modify anything in the main function, but do not change the functions that create relations so that we can compare your results with our own.

Compiling and Running Your Code

The file `Joins.tgz` contains the skeleton code and libraries necessary for this assignment. The libraries are precompiled for the DICE environment, thus we expect that you develop and test your solution on a DICE machine. The grading of this assignment will also be done on a DICE machine.

Assuming your working directory is `~/`, copy the file `Joins.tgz` into your working directory and unzip it using the command `tar -xvzf Joins.tgz`. This will result in a folder `~/Joins/` containing the source code and makefile of the project. Now run `make` to compile the source code; run `make clean` to delete the binary and any compiled object files. The resulting executable file is `./bin/joins`. You should be able to compile and run the code, but the execution of the code will only create random records for two relations without executing the join (you have to implement the joins and invoke them in the main class).

Differently from the previous assignment, there are not test cases to be passed this time. You are required to modify the *main.cpp* file in order to invoke the two different join algorithms, collect statistics during each run and print these statistics at the end of each run. You are required to run each algorithm several times and report the average of all the running times.

What to Turn In

You should submit the same set of files given to you at the beginning of this assignment, plus any additional header and source files you have created for this assignment. These files should be zipped up into a file named `Joins-FirstName-LastName.tgz` (or `.zip`) using the command `tar -cvzf Joins-FirstName-LastName.tgz Joins`. These files should be organized in the same directory structure as well. Your report should be located in the `Joins` directory.

Upload your solution (i.e., tgz/zip file) using the DICE submission system:
`submit adbs cw2 Joins-FirstName-LastName.tgz`.

Marking: Each join algorithm carries 35% of the mark, while the report carries 30% of the mark. We will test your solution for correctness using different input relations and also validate your experimental results (we will not be looking for identical numbers but close enough).

Make sure you start early! Good luck!