CSCE-608 Database Systems

Spring 2023

Instructor: Dr. Jianer Chen
Office: PETR 428
Phone: 845-4259
Email: `chen@cse.tamu.edu`
Office Hours: TR 2:15 pm–3:45 pm

Teaching Assistant: Ya-Ru Yang
Office: PETR 445
Phone: (979) 969-0095
Email: `yaruyang@tamu.edu`
Office Hours: MW 2:00 pm–3:30 pm

COURSE PROJECT II

(Due April 27, 2023)

Overview. In this project, you will implement a number of key algorithms that have been widely used in databsae management systems, which include: (1) implementation of B+trees; and (2) implementation of hash-based algorithms for relational algebraic operations.

The project is an individual project. Each student should work on her/his own project.

1 B+ Trees

This part of the project includes an implementation of B+trees, whose order is given as a parameter. The B+trees should support search, range search, insertion, and deletion operations. In addition, you also need to implement procedures that generate a large collection of records, and that construct dense B+trees and sparse B+trees when a collection of records is given.

Part 1. Data Generation. Assume that each of your records only contains a search key with no other attributes. A search key is an integer between 100,000 and 200,000. Your first procedure is to generate 10,000 *different* records.

Part 2. Building B+trees. For a given collection C of records, you should implement 2 procedures that build B+trees for the collection C, one builds a dense B+tree (i.e., a B+tree whose no-root nodes are as full as possible), and the other builds a sparse B+tree (i.e., a B+tree whose nodes are as sparse as possible).

Part 3. Operations on B+trees. Now you implement the search, range search, insertion, and deletion operations on B+trees.

Part 4. Experiments. Test your implementations, with the following steps:

(a) using your implementation in Part 1 to generate a collection of 10,000 records;
(b) using your implementation in Part 2 to build 2 B+trees of order 13, one dense and one sparse, and 2 B+trees of order 24, one dense and one sparse;
(c) testing your implementation in Part 3 for B+tree operations. For the B+trees constructed in step (b),
  (c1) apply 2 randomly generated insertion operations on each of the dense trees;
  (c2) apply 2 randomly generated deletion operations on each of the sparse trees;
  (c3) apply 5 additional randomly generated insertion/deletion operations on each of the dense and sparse trees;
  (c4) apply 5 randomly generated search operations on each of the dense and sparse trees.

For each test, your procedure should print out all the tree nodes involved in the operation. If the node is changed, print out the nodes before and after the operation.

2 Join based on Hashing

This part of the project implements the two-pass join algorithm based on hashing. Assume that you have two relations $R(A, B)$ and $S(B, C)$, and you want to compute their natural join $R(A, B) \bowtie S(B, C)$ using the two-pass join algorithm based on hashing.

Assume that each block can hold upto 8 tuples of the relations, and that we have a virtual main memory of 15 blocks and a virtual disk whose size is unlimited. You can pick any data structures for the virtual main memory and virtual disk. However, the join operation can only be performed when the tuples are in the virtual main memory.

Part 1. Data Generation. Generate a relation $S(B, C)$ of 5,000 tuples, where B is the key attribute and C can be of any type. The values of the attribute B are random integers between 10,000 and 50,000. The relation $S(B, C)$ is stored in the virtual disk.

Part 2. Virtual Disk I/O. Write two procedures that simulate disk I/O's, in which one reads from the virtual disk to the virtual main memory and the other writes from the virtual main memory to the virtual disk. The read/write should be in blocks.

Part 3. Hash Function. Pick a good hashing function that maps the B-values of the relations to a proper range for your algorithm.

Part 4. Join Algorithm. Implement the two-pass natural join operation based on hashing. Your algorithm must use the procedures in Part 2 to transform data between the virtual disk and virtual main memory, and can process data only when the data are in the virtual main memory. Note that this part also includes the implementation of a one-pass natural join algorithm.

Part 5. Experiment. Test your implementation.

5.1 Generate a relation $R(A, B)$ of 1,000 tuples, in which the values of the attribute B are randomly picked from that in the relation $S(B, C)$ (duplicates are allowed), and the attribute A can be of any type. Call your algorithm in Part 4 to compute the natural join $R(A, B) \bowtie S(B, C)$. Count the number of disk I/O's used by the algorithm. Randomly pick 20 B-values, and print out all tuples in the join $R(A, B) \bowtie S(B, C)$ whose B-values are the picked values.

5.2 Generate a relation $R(A, B)$ of 1,200 tuples, in which the values of the attribute B are randomly picked from integers between 20,000 and 30,000, but not necessarily form the B-values in the relation $S(B, C)$ (again duplicates are allowed). The attribute A can be of any type. Call your algorithm in Part 4 to compute the natural join $R(A, B) \bowtie S(B, C)$. Count the number of disk I/O's used by the algorithm. Print out all tuples in the join $R(A, B) \bowtie S(B, C)$.

3 Report

Write a report of at least five pages, single-spaced, to explain your implementations, summarize and discuss your experiment results, report difficulties you encountered during the project and how you got over them, and share the experience you gained from the project.

4 Grading Policy

- Data generations: 10%,
- Algorithm implementations: 40%
- Experiment summary and discussion: 30%
- Project report: 20%