INFO-H-417: Database Systems Architecture – Lab 2

Lecturer: Stijn Vansummeren
Teaching-Assistant: Stefan Eppe
`http://cs.ulb.ac.be/public/teaching/infoh417`

Academic Year 2013–2014

Setup

In this second lab, we focus on the physical layout of B-Tree and on performance improvement that can be achieved by using a B-Tree index to access a relation. In this first part we first setup our database prototype. Then, to establish a baseline for comparison, we check the number of blocks that must be read from the disk when querying a relation through a simple scan of all records.

1. Clone the database implementation from the `http://wit-projects.ulb.ac.be/rhodecode/INFO-H-417/Labs/Lab-2` git repository.
2. Compile this project from eclipse and run it. It should indicate that it created a database, with a relation named `Foo`.
3. In the `IOTest` class, add calls to the `printStatistics` function before and after scanning the `Foo` relation. How many blocks are read when `NUMRECORDS` is 50, 1000, and 10000?

Index Initialization

We will now add a B-Tree Index to our relation in the database, and describe the layout of the B-Tree on disk.

1. Add a B-Tree index (using `BTIndexFactory`) on the `Foo` relation, in the `createDatabase` function. The B-Tree will use (A, B) as its schema (and will hence be dense for the relation).
2. **Pen and paper exercise** Consider the leaves of the B-Tree. Is there an ordering of the keys inside these leaves? How many keys do you expect to find in the leaves in total, if the number of records in the relation is 10000?
3. **Pen and paper exercise** In our B-Tree, inner nodes have a header of 4 bytes, and each (key, block address) pair is the size of the key plus 4 bytes. The leaf nodes also use 4 bytes for their header, but stores each (key, record address) on the size of the key + 8 bytes. How many leaf blocks do we need to store 10000 records? What is the height of the B-Tree in this circumstance?

Querying the Index

In this second part, we will use the index to answer a few queries. We will also improve the implementation by chaining leaf blocks, to enable range queries (e.g. finding the records such that $A = 8$ and $B >= 5$). When statistics are to be computed, we will only consider the situation where 10000 blocks are loaded.

1. Using the index, fetch the record for which $A = 5$ and $B = 2$. Note that since keys inside the B-Tree are represented as `Record` objects, the `createRecord` function can be used to create the query key for the index.

   Check the number of disk reads and of pin requests for this operation. Explain this result by means of the result from exercise 3 of part I (index initialization).

   Compare with the disk reads and of pin requests when scanning the relation.

2. Describe how the `BTIndexIterator` class uses the chaining of leaf blocks to provide ranged queries.

3. Looking at the code of the `BTIndexBlockLeaf` class, determine which operations need to be aware of block chaining. Update the code accordingly.

4. Using the index, fetch the records for which $A = 8$ and $B >= 5$. How does this compare with scanning the relation?

Querying the index (continued)

In this section, we try a variety of query operations to understand the limits of the B-Tree index. For the purpose of this exercise, you can assume that the B-values range from 0 to 1000, while the A-values range from 0 to 9

1. Explain how you can perform the following queries on the B-Tree.

   - Find the records such that $A = 2$ and $B >= 4$.
   - Find the records such that $A < 3$.
   - Find the records such that $A = 3$ and $B < 4$.
   - Find the records such that $B = 3$.
   - Find the records such that $B < 4$.
   - Find the records such that $A > 3$ and $B > 7$.

2. **Supplemental** Write the code implementing the previous queries, and compare the performance with scanning the relation.