CS740 Project

Fall 2001

v1.0

Abstract

The goal of the project is to develop an extensible single-user (at this point) relational database system in JAVA.

The project is organized into three *assignments*. The first two assignments are common to everyone in the class, for the last assignment you may choose a topic from the list of suggested topics, all of which will be discussed in the class.

Assignment 1: Files, Indices, and Iterators

Using the following two `Interfaces`:

```
public interface Iterator {
    public void  open(Tuple parameters);
    public Tuple fetch();
    public void  close();
}

public class Tuple {
    public Tuple(int n) {...};       // new tuple of size n
    public int size() {...};
    public Object get(int index) {...};
    public void   set(int index, Object value) {...};
}
```

develop the following *Access Methods*; each access method consists of a *reader* that implements the `Iterator` interface (the iterator allows retrieving data from a disk-based file), and a *writer* that consumes the `Iterator` interface and produces the appropriate file[1].

[1] If a file with the same name exists, it is to be overwritten.

Text File: Access to standard test files in Unix.

Reader. Given a file name (`String`) provides an `Iterator` that produces a sequence of binary tuples $[i, s]$ where s is a `String` representation of a line in the input file and i is an `Integer` denoting the position of the line in the input (in bytes from the start of the file). The set of lines read is determined by a binary *parameter* tuple supplied to the `open(Tuple t)` method; it contains two `Integer` values denoting the offsets of the first and last lines, respectively, returned by the iterator (a `null` as a parameter reads the whole file).

```
class TextFileReader implements Iterator {
    public TextFileReader(String name, ...)
}
```

Writer. Given a file name and an instance of an `Iterator`, it creates a *text* file containing the string representation of the records produced by the iterator. This method will also be used to report the results of queries (by directing the output to the terminal).

```
class TextFileWriter {
    public TextFileWriter(String name,Iterator input, ...)
}
```

Extra Bonus: the constructors of the reader/writer use an additional argument to specify the inner structure of fields in the records stored in the file (e.g., by defining various *tags*). In this case, the reader produces general tuples of the form $[i, s_1, \ldots, s_k]$ where $s_1, \ldots, s_k$ are strings representing the values of the individual fields.

B+ Tree Index. Similarly to the **Text File**, the reader and writer retrieve and store sets of records in a B+ tree structure. The reader should support value lookups and range lookups using parameters passed to the `open(Tuple t)` method (similarly to the Text File).

Your implementation of the B+ tree writer may rely on the assumption that the iterator given as its input produced records in increasing order of the search values (search keys). However, in this case you must also implement an

```
Iterator Sort(Iterator input, Tuple sortOrder, ...)
```

implementation of the `Iterator` interface[2] that allows sorting sets of tuples generated by the `input` iterator.

[2] This is the recommended implementation path as you'll need the `Sort` for query processing too.

Preview of Future Assignments

Here is a preview of how the above pieces will be used in the rest of the project.

Assignment 2: Relational Algebra and Query Processor

The goal of the second part of the project:

1. implement *sufficient number* of relational algebra operators based on the same `Iterator` interface (here we use the file I/O from part 1.).

2. implement a *query processor*—an user interface that reads (text representation of) a set of assignment statements of an expression trees specifying a physical plan for query to file representation of relations of the form

$$R := E(R_1, \ldots, R_k).$$

   The query processor verifies the validity of the sequence and executes it creating new relations (files) and/or printing results.

As everyone is using the same `Iterator` interface, you may *trade* th individual relational operations, as long as the authors of the code are acknowledged.

Assignment 3: Topics

The last part of the assignment may vary among the groups; here are several of the topics I'd like to see at least one group to take[3]:

- a XML front end: XML data can be stored in relations using an *interval encoding* and then manipulated by relational DBMS. This also includes a XML-specific query optimizer.

- materialized views: the system so far does not implement updates. As the intermediate relations (and thus all indices) are specified as views, this sub-project's goal is to extend the interface to handle incremental change.

- Advanced data structures: phase 1 of this project defines only two ways of storing relations—text files and B+ trees. There are many other disk-resident data structures that speed up data retrieval (in particular, for the XML specific project, a 2-dimensional range queries are essential). The goal here is to implement additional disk-resident data structures including a proper buffer manager (skipped in phase 1).

In each case, the result of the third phase should be built on top (bottom) of the system from phase 2. In addition to the actual code base you'll be also asked to document your work (i.e., write a short paper about what you've done).

[3] More to come; if you have an idea, talk to me.