CS44800 Project 3
Relational Operators
Spring 2019

Due: **April 1, 2019, 11:59PM**
Total Points: **9 points**

Learning Objectives

1. Understand how Relational Algebra operators are implemented inside of a database system
2. Understand algorithms for executing different Relational Algebra operators

Background

When executing an SQL query, a DBMS transforms the query into an equivalent Relational Algebra expression. Each Relational Algebra is implemented through an *iterator interface*. This interface consists of the following functions:

- open()
- close()
- hasNext()
- getNext()

Relational Algebra operators are then chained together to form a Query Tree. A query is then evaluated by starting with the root operator of the tree and calling hasNext() and getNext() to recursively request the next tuple from its child Iterators until all rows have been exhausted.

This results in an implementation where the results of a query are evaluated lazily and in a non-blocking fashion. Any given Iterator only requests results from its child Iterators when necessary and in most cases does not need to compute its entire set of results before generating results for the Iterators higher up the query tree – tuples simply "flow" up the query tree as they are evaluated. This *pipelined* approach results in querys that can start generating results immediately, and do not require any storage space for intermediate results.

In this project, you will be implementing Relational Algebra using this Iterator interface, creating the functionality necessary to actually execute any arbitrary query in Minibase.

You can find the API for the Minibase classes you will be using here. Please familiarize yourself with the API before starting this project. In particular, the classes you will need to use are the classes in the *relop* package, HeapFile and HeapScan in the *heap* package, HashIndex, HashScan and BucketScan in the *index* package, and AttrOperator and AttrType in the *global* package.

Part 1: Scan Operators - From Records to Tuples

All Relational Algebra operators you will be implementing in this project will be subclasses of the Iterator abstract class. The Iterator class provides a *schema* field (of protected visibility) that can store the schema of any output tuple from that Iterator, as well as defines the following methods that must be implemented:

- public void restart()
- public boolean isOpen()
- public void close()
- public boolean hasNext()
- public Tuple getNext()

The constructor for each Iterator is equivalent to the isOpen() function of the Iterator interface.

Note: you do not have to implement the explain() method

Your first task is to construct the initial scan operators. These are Iterators that read through an entire relation or index and report one Tuple at a time. Minibase already has methods defined to scan through and retrieve records in raw byte format from the HeapFile and HashIndex classes, but we need to create Iterator versions of these scans that will act as wrappers for these scans and will return Tuples with defined schemas as opposed to the raw byte information.

Specifically, we require the following Scan operators:

- **FileScan**
  A *HeapScan* that returns *Tuples* instead of byte[] 's
- **KeyScan**
  A *HashScan* that returns *Tuples* instead of *RID*s
- **IndexScan**
  A *BucketScan* that returns *Tuples* instead of *RID*s

The FileScan implementation is already provided to you. You will need to implement similar scan operators for the KeyScan and IndexScan.
Some useful hints and tips:

- Each constructor should initialize the inherited field *schema*. (i.e. it's given as a parameter to these three iterators)
- Don't be surprised by how little code these classes require
- *HashScan* scans the hash index for records having a given search key. *BucketScan* scans the whole hash index. Those classes only return RIDs. You have to build wrapper classes KeyScan and IndexScan that return Tuple.

Part 2: Primitive Operators

Now that you have the basic leaf nodes of most query trees, you can make some more interesting iterators. Your next task is to implement the three fundamental operations of relational algebra:

Selection

Filters the output of another Iterator on a set of Predicates. When there are multiple Predicates, they are assumed to all be ORed together (AND is implemented through chaining separate Selection operators together)

Projection

Performs a projection on a Tuple by removing columns from the output of another Iterator. This will change the Schema of all Tuples output by the Projection. Note: you should not remove duplicates

HashJoin

The code for "Nested Loops Join" is provided for you in SimpleJoin.java. You are required to implement a **Hash-Join**. You may consider to study the code in SimpleJoin.java as an example of how to implement a Join operation.

A Hash-Join consists of two stages: first, a HashTable is constructed on one of the input relations. Second, this HashTable is probed using tuples from the second relation. Luckily, you do not have to implement most of this from scratch – you can make use of the IndexScan within Minibase to perform this.

For the first phase of a Hash-Join, you need to transform the input Iterators into an IndexScan with the join column as the key. **Note: You cannot simply cast the child Iterators of the HashJoin as an IndexScan. You must construct a new temporary HashIndex and HeapFile populated with the results of the child Iterator and create an IndexScan.**

Once the child Iterators have been transformed into IndexScans with the join column as the search key, you can the iterate over the IndexScans to determine matching results. An IndexScan iterates over an index bucket-by-bucket. Note that the contents of a bucket may be in any arbitrary order and may not all be matches, even if all entries have hashed to the same bucket. Therefore, you will have to track matching tuples within a bucket as you iterate through. To facilitate this, you are provided with a class HashTableDup. This class acts as a HashTable, but allows for multiple identical keys to exist.

Some useful hints and tips:

- For Relational Algebra operators that may not always find results (i.e. Selection and HashJoin), you need to be careful about exactly how you are determining the result of the hasNext() method call. Unlike with the basic scans, you cannot simply just return the result of the child iterator's hasNext() method. You may check the SimpleJoin.java class for an example of how to properly handle this.
- The join column specification for HashJoin is based on the schemas of the individual child Iterators *before* the join. For example, a HashJoin constructor call with "(…,0,2)" would mean the join columns are column 0 of the left Iterator and column 2 of the right Iterator.
- Implementing the HashJoin will likely be the most complicated part of the project. Please budget your time accordingly.

Running Minibase

As with the previous project, you may run the test cases using the provided Makefile.

This project has been tested and run through the terminal on the university Linux machines. We cannot provide support for using different IDEs or running the program on different platforms, but some posts have been made on Piazza regarding setting up IDEs to run the Minibase projects that you can reference.

What to Turn in

1. A .zip archive of the relop folder.

These files should be submitted on Blackboard.