CSI 410. Database Systems I – Spring 2021

Programming Assignment II

The total grade for this assignment is 100 points. The deadline for this assignment is **11:59 PM, May 10, 2021**. *Submissions after this deadline will not be accepted.* Students are required to enter the UAlbany Blackboard system and then upload a .zip file (in the form of [first name]_[last name].zip) that contains the Eclipse project directory and a short document describing:

- any missing or incomplete elements of the code
- any changes made to the original API
- the amount of time spent for this assignment
- suggestions or comments if any

No submission of the above document will lead to a *loss of 5 grade points.* Also, please add *comments* in your code. Code with (almost) no comments may result in a loss of 1 grade point per method to implement.

This programming assignment focuses on implementing two relational operators (selection and aggregation). You first need to run Eclipse on your machine and import the "`hdb_query`" project (see Appendix A). This assignment requires the "`hdb_data`" project that you worked on in the past. You should be able to do this assignment as long as `Tuple(RelationSchema, Object...)` in `Tuple.java` and `attributeIndex(String)` and `attributeType(int)` in `RelatonSchema.java` are correctly implemented (other parts in "`hdb_data`" are unlikely to affect this project). Please generate a Javadoc API document and then take a look at the document as well as the source code to familiarize yourself with this assignment. For this assignment, we have provided you with a set of classes (in particular, see `ProjectionOperator` which will help you understand how a relational operator can be implemented as well as `TupleArrayReader`, `SelectionOperator`, `AggregageOperator`, and `Aggregator` that you need to complete by adding more code). Your code will be graded by running a set of unit tests and then examining your code. Passing unit tests does not necessarily guarantee that your implementation is correct and efficient. Please strive to write correct and efficient code. If you have questions, please contact the TA(s) or the instructor. The remainder of this document describes the components that you need to implement.

Part 1. Operators and `TupleArrayReader` (30 points)

In this assignment, all of the relational operators support the following four methods (see `Operator.java`):

- `hasNext()`: determines whether or not the operator has the next output `Tuple`.
- `next()`: returns the next output `Tuple`.
- `rewind()`: rewinds the operator in order to retrieve all of the output `Tuple`s again.
- `outputSchema()`: returns the output schema of the operator.

It should be noted that the `Operator` interface inherits the `hasNext()` and `next()` methods from the standard Java `Iterator` interface.

In this part, you need to complete the code in TupleArrayReader.java. Each TupleArrayReader outputs, given an array of Tuples, the Tuples in that array. The methods to complete are hasNext(), next(), and rewind(). Implemet these methods using the currentIndex member variable which is to remember the index of the next Tuple to retrieve/return during each iteration over the Tuples. When all of the above methods are implemented correctly, your code will pass the unit tests in TupleArrayReaderTest.java.

Part 2. Selection (50 points)

In this part, you need to complete the code in SelectionOperator.java. For this task, it might be helpful to understand the implementation of ProjectionOperator. Each SelectionOperator outputs, given a series of Tuples from another operator, the Tuples that satisfy a predicate given to the SelectionOperator when it was constructed. In other words, it filters out all Tuples that do not match its predicate. Each SelectionOperator has its own ExpressionEvaluator which can evaluate an expression (i.e., the predicate) for each input Tuple. Given ExpressionEvaluator evaluator and an input Tuple t, (evaluator.evaluate(t) == Boolean.TRUE) indicates that t satisfies the predicate that evaluator uses (i.e., the SelectionOperator must output t). The SelectionOperator can also be viewed as an iterator over all of the Tuples that it outputs. Your implementation should not keep all of the output Tuples in the memory since it may be infeasible in practical situations (i.e., the memory may not be able to preserve all of the output Tuples when a large number of the Tuples are output). Instead, the SelectionOperator (which is also viewed as an iterator) needs to find, whenever the hasMext() and next() methods are called, the next input Tuple that satisfies the predicate (on-demand, pull-based pipeline; see Section 15.7 in the textbook).

The constructor/methods to complete are as follows:

- SelectionOperator(Operator input, String predicate): constructs a SelectionOperator. This constructor needs to create an ExpressionEvaluator for evaluating the predicate on each input Tuple and may do some additional work (depending on your implementation) to support the hasNext() and next() methods.
- outputSchema(): returns the output schema of the SelectionOperator. This output schema is the same as the input schema of the SelectionOperator. Consider using a method or a member variable provided by a super-type of SelectionOperator to get the input schema of the SelectionOperator.
- hasNext(): determines whether or not the SelectionOperator has the next output Tuple.
- next(): returns the next output Tuple.
- rewind(): rewinds the operator in order to retrieve all of the output Tuples again.

When all of the above methods are implemented correctly, your code will pass the unit tests in SelectionOperatorTest.

Part 3. Aggregation: AggregateOperator.java (5 points)

In this part, you need to complete AggregateOperator.java. An Aggregator groups all Tuples (by user-specified attributes) from an input Operator and outputs, for each group of Tuples, a Tuple that represents/summarizes that group of Tuples (e.g., for each location code, the minimum temperature). An AggregateOperator is an operator that uses an Aggregator and supports the basic iterator capabilities (hasNext() and next() methods).

This part requires implementing only the following method:

- createOutputSchema(AggregateFunction[] aggregateFunctions): constructs and then returns the output schema of the AggregageOperator. The output schema consists of the grouping attributes (e.g., Location) and additional attributes from AggregateFunctions (e.g., Maximum(Temperature)). This method needs to construct a new RelationSchema which requires an array storing the names of the attributes and another array storing the types of these attributes.

  For the i-th grouping attribute, the name of that attribute can be obtained from groupingAttributeNames[i]. Given that attribute name, the type of the attribute can be obtained by using inputSchema.attributeIndex(String) and inputSchema.attributeType(int). For the i-th AggregateFunction, the name of that AggregateFunction can be obtained from aggregateFunctions[i].toString(). The type of that AggregateFunction can be obtained from aggregateFunctions[i].valueType().

When the above method is implemented correctly, your code will pass the unit tests in AggregateOperatorTest while showing the following output:

```
input schema: {ID=java.lang.Integer, Location=java.lang.Integer, Temperature=java.lang.Double}
grouping attributes: [Location]
aggregate functions: [Minimum(Temperature)]
output schema: {Location=java.lang.Integer, Minimum(Temperature)=java.lang.Double}
```

The above output shows a situation where (i) the input schema has attributes ID, Location, and Temperature, (ii) the grouping attribute is Location, (iii) the aggregate function is Minimum(Temperature), and thus (iv) the output schema has attributes Location and Minimum(Temperature).

Part 4. Aggregation: Aggregator.java (10 points)

As mentioned in Part 3, an Aggregator groups all Tuples (by user-specified attributes) from an input Operator and outputs, for each group of Tuples, a Tuple that represents/summarizes that group of Tuples (e.g., for each location code, the minimum temperature). In this part, you need to implement the following costructor and method in Aggregator.java:

- Aggregator(Operator input, RelationSchema outputSchema, String[] groupingAttributeNames, Class<?>[] aggregateFunctionTypes, String[] aggregationAttributeNames): constructs an Aggregator. Given an input tuple t, the Aggregator needs to extract the values of the grouping attributes (e.g., Location value 0) and then finds the AggregateFunctions for that combination of grouping values (e.g., Minimum and Maximum that have been applied to the Temperature attribute from all Tuples whose Location value is 0). Then, the Aggregator needs to update all of these AggregateFunctions based on tuple t (e.g., update the minimum value if the Temperature value of t is smaller than the previous minimum). The above implementation approach requires space linear in the number of distinct groups. For the purposes of this assignment, you do not need to worry about the situations where there are too many groups to fit into the memory.

- iterator(): returns an iterator over the output Tuples of the Aggregator. The details of these output Tuples are explained above (refer to the output schema of the AggregageOperator).

When the above constructor and method are implemented correctly, your code will pass the unit tests in AggregatorTest.

Appendix A. Importing a Java Project

1. Start Eclipse. If Eclipse runs for the first time, it asks the user to choose the workspace location. You may use the default location.

2. In the menu bar, choose “File” and then “Import”. Next, select “General” and “Existing Projects into Workspace”. Then, click the “Browse” button and select the “`hdb_query.zip`” file contained in this assignment package.

3. Once the project is imported, you can choose `TupleArrayReaderTest.java`, `SelectionOperatorTest.java`, textttAggregateOperatorTest.java, or `AggregatorTest.java` and then run the program.