**Report of Milestone 3**

**Team #17**

**Rewriting**

There are basically five steps for rewriting.

**Step 1:** Classify all variables defined in for-clause into multiple tables. All variables in the same table are related. The way to judge whether two variables are related is to see whether these two variables appear in the same context. In the example below, $ta and $a are related because $ta and $a both appear in the second sentence.

for $a in doc("input")/book,

$ta in $a/title

**Step 2:** Based on the conditions in the where-clause, judge which two tables are joined (var1 eq var2) and which table needs selection (var eq constant). We use HashMap to record.

If table 1 and table 2 are joined on the condition that “$a eq $b and $a2 eq $b2”, then the key is a string “1-2”, in which the smaller number is at the front. The value is a list of array. Each array is of length of two with each element recording a variable like, string[]{“$a”, “$b”} and string[]{“$a2”, “$b2”}. Here the hash table used is named joinBetweenTables.

If table 3 needs selection that “$c eq “John””. The key is “3”. The value is a list of string array containing string[] {“$c”, “John”}. Here the hash table used is named joinOnConst.

**Step 3:** Rewrite each table in the format as below, which will appear in the join-clause in the final rewritten query. The tables that need selection (we have recorded the names of them in joinOnConst hash map) have an additional where-clause.

for $b1 in doc("input")/book,

$aj in $b1/author/first/text(),

where $aj eq "John"

return <tuple>

<b1>{$b1}</b1>

<aj> {$aj} </aj>

</tuple>

**Step 4:** Rewrite join-clause. Here we define a cluster as tables associated with each other. First, we identify all clusters using union-find algorithm and the knowledge about which two tables are joined (recorded in joinBetweenTables hash map). There is no any relation between any two clusters. Then we recursively write join-clause about joint tables. Finally, we write Cartesian product among clusters by leaving [], [] empty.

**Step 5:** Rewrite return-clause. In the return-clause, there are two types of nodes, variable and node starting with <tagname>. For the second case, a recursion for rewriting is needed.

**Hash Join**

To enable Antlr to identify join-clause, we add the rule for join-clause in grammars.

For joining on empty variable lists, we simply return the Cartesian product of the two table entries, and make the concatenation of nodes as new records. These records as a list then become our return result.

For joining conditions being on non-empty variable lists, we will take the following steps, besides doing preparations for record-keeping such as remembering the current context.

**Step 1:** Create a HashMap

We pick from the two joint tables the one with fewer result tuples and hash its tuples based on the variables mentioned within “[ ], [ ]”. We use TransformerFactory to transform joint variable nodes into string (if there are multiple joint variables, concatenate all strings into one) and use the hashcode of this string as the key in hashmap. The value of the hashmap is a list of Node, each node is the tuple of the smaller table. Tuples that have the same values in joint variables are put into the same list.

**Step 2:** Search for matching tuples

We loop over the bigger table, for each of the records we look for matching records in the smaller table. We use the same method to get a hash key for each of the return tuple of the smaller table. If the key exists in the hash map, fetch the list of nodes on this key from the hash map. Skip to next tuple otherwise.

**Step 3:** Create a new combined tuple

For each of node in the matching list, we concatenate its children with the children of the current examined tuple in the smaller table, and wrap it as a new tuple. Then put it into the result list.

**Problem Encountered**

There is ambiguity for grammar “xq , xq” and “xq/rp” making Antlr wrongly parse the query after rewriting. So put “xq , xq” rule at the front of “xq/rp” rule so that the former has greater priority.