Milestone 3: Join Optimization

Chunchun Xu, Chenying Liu

March 17, 2017

1 XQuery Rewriter

Our xquery rewriter handles multi-way joins: transform a unoptimised flower XQuery into a join-based XQuery.

1.1 Main Idea

Our implementation rewrits the XQuery from the XQuery parse tree generated by Antlr. We detect different parts to join in the original for clause, every document node represents another partition to join. And detect the join keys in the where clause since the join conditions are proposed in the where clause. And in this part, we do not consider let clause.

1.2 Implementation

Mainly, we use three data structures below to implement the main idea:
$LinkedHashMap < String, LinkedHashMap < String, String >> varMap$ stores variables and the corresponding part they belong to.
$ArrayList < LinkedHashMap < String, String >> joinParts$; stores the root variables and the corresponding documents
$ArrayList < String > wheres$; stores the join conditions. To keep the input order, we use LinkedHashMap. With **XQuery parse tree**, we can have access to the for clause, where clause and return clause correspondingly. During the process traversing the for clause, every time we meet a variable, we either create a new join part if the corresponding path is a absolute path, or stores it into the existing part according to the relative path. After this step, if there is less than two part to be joined than this query cannot be rewritten.
Then, we can get the join conditions which are equal pairs by split the where clause by "and". In this part we only consider variable equal to constant(keep in the flower XQuery) and variable equal to variable(join).
To handle multi-way joins, we recursively match join parts with conditions and rewrite the the XQuery to **deep left joins**. Inside each loop, we go through the join conditions. The two first distinguished parts involved in the unmatched conditions are dealt. Rewrite them

into the join format with two XQuery and 2 corresponding join key list. After that, merge the two parts into one by merging the joinParts. Keep rewrite the XQuery util there is only one part left. In the meantime, we have to rewrite the return clause, change the tag name, add "$/*$" after variable. We use *java.util.regex* to find the patterns such as match variable pattern "*var*".

2 Join Implementation

The join is to add elements together by getting together two different table, thus we can have a larger table

2.1 Main Idea

The join can also combine with the flwr. In the flwr, the join can be used as a part of this combine. In the for loop, everytime, we get the element, we can use append for the arraylist and put element inside the arraylist and then return. The arraylist can then be a tool to join. Thus the join can be written with different way. The join can also be independent way to use, because the XML can be divided into left and right branch of tree. The join hash is a good way to combine two table. We can start with the samll size table to do hashing, then find the same element to put into the big hashmap

2.2 Implementation

The join function is for put every query together. In the g4 file, we put join clause as join (xq , xq , attlist , attlist). First, we can get the size after visit the contet of joincontext. Then, we can compare the size of left and right branch. We can think the two branch as two tables. Later, we use arraylist, which include the hashmap in it. According to the visit, the listsize is 1521, however, the context is null. In order to get the real value, we can use getText(). In the for loop, in the range of 1 to 1521, we can get value for each, then for xqvisit, each time we get the key, so key shows the speaker of each query. In the hashmap, we put key inside the map. Is the map contains certain key, we put key inside the array list. Is the map does not contain the key, we put key inside the map. In the XML structure, the element can be divided into two branch, left tree and right tree. The left tree can be assigned as the smaller size. Then we can iterate the left tree to do hashing. If the map contains key, the arraylist have to store the list of key. If the map does not contain the key, put key inside a arraylist and put temp inside the key. Then we can get a set of key. We also do the same thing on the right side,however, the right tree should be a part of total tree after iterate. The value we put in is the string of node. In the process of handling each node if the left and right, we read element and put element in the new arraylist, thus the element can be read the the output can be like the sample.