Task documentation XQR: XML Query in Python 3 for IPP 2016/2017
Name and surname: Juraj Ondrej Dúbrava
Login: xdubra03

XQR script is made for simulating SQL SELECT command in XML file. User defines SELECT query from command line as one of the arguments of the script or specifies input file with query to be used. Output of the script is XML file with XML elements which were selected using user´s query.

SELECT query handling

SELECT query has to be syntactically and semantically correct. Firstly, we have to get tokens from query. This part is implemented as a finite automata in get_next_token() function. Get_next token() uses getchar() function to get character from query string. Token itself is represented as a class named Token, where informations about type and value are stored. If lexical analysis fails, unrecognized token was found in query and script returns lexical error. Second part of query processing is syntactic analysis. It is implemented as recursive descend according to context-free grammar for query. Each rule checks syntactic construction for specific part of query. While syntactic analysis is running, each part of SELECT query is stored to dictionary query_items, where parts of query are stored for further processing. If syntactic analysis passed, query is correct and script can continue with searching for required elements from XML file.

SELECT elements from input

After successful lexical and syntactic analysis, script performs applying SELECT query in XML file. For handling XML file from input, I used minidom XML library. First part of query processing involves searching for first occurrence of FROM element in input file. FROM element could be of 4 different types, ROOT element of XML file, element, element's attribute or only attribute. Each type of element is processed individually and this process also contains processing WHERE condition. Searching for elements is implemented with function getElementsByTagName() which returns list of all element with entered name. If no element was found, it means that FROM element does not exist and output correct_elems list, where all correct elements are stored, is empty. If FROM element was found, script continues with searching for all SELECT elements. List of all occurrences is always stored in roots variable. In case that list of SELECT elements is not empty, for each element is performed processing of WHERE condition. Absence of WHERE condition in query means that correct_elems list contains all SELECT elements.

If WHERE condition is in query, script searches for element from WHERE condition. Non empty list of WHERE elements means that script can continue with evaluating condition using check_condition() function. If result of condition is true, currently processed SELECT element is added to the correct_elems output list. Evaluating NOT condition is implemented in check_condition for each operation. For non existing WHERE element is NOT condition evaluated with not_condition() function. While getting value from WHERE element, script is checking whether element is TEXT_NODE or if it is ELEMENT_NODE and no text value is inside. Searching for element.attribute or attribute in FROM and WHERE element is similar to searching for an element. Script uses function hasAttribute() to find element with certain attribute and getAttribute() to get value from an attribute.

SELECT elements output

For creating output XML file is used correct_elems list containing all elements from SELECT query or all element which suited WHERE condition. Before creating output, script performs check for LIMIT element in query. If this element is in query, only limit_num of elements from correct_elems list is written to the output file. Creating output also contains control for arguments for generating XML header and adding root element from option value. For writing XML element is used function toxml(), which represents elements structure with all subelements.