Dokumentace úlohy XQR: XML query v Pythonu do IPP 2011/2012
Jméno a příjmení: Petr Dvořáček
Login: xdvora0n
E-mail: xdvora0n@stud.fit.vutbr.cz

1 Úvod

Tento dokument pojednává o impelemntaci projektu do předmětu IPP. Myšlenkou této úlohy bylo vytvořit skript, který provádí vyhodnocení zadaného dotazu, který je podobný pseudo-příkazu SELECT jazyka SQL, nad vstupem ve formátu XML. Výstupem je XML obsahující elementy splňující požadavky dané dotazem.

V tomto projektu jsem použil následující moduly: `sys` pro prácí se vstupem a výstupem, `re` kvůli regulárním výrazům a `xml.etree.ElementTree` pro práci s XML souborem.

2 Hlavní funkce

Při spuštění programu se zavolá funkce `main`, ve které se následně zpracovávají parametry programu.

2.1 Řídicí proměnné

Jako řídicí proměnnou používám `flags`, ve které každý bit značí svou činnost definovanou ve třídě `CTRL`. Uložil jsem to právě do třídy, protože Python nepodporuje výčtový typ `enum`. Jednotlivé řídicí bity přidávám pomocí bitové operace *or*. Chci-li zjistit zda je nějaký řídicí bit nastaven, použiji bitovou operaci *and*. Takový to zápis má výhodu v tom, že je jednak čitelný a jednak zabírá méně místa v paměti.

2.2 Zpracování argumentů a volání metod

Zpracování argumentů programu probíhá ve smyčcce `for actArg in sys.argv:` za pomocí regulárních výrazů. Zde kontroluji zda byly parametry zadány vícekrát a zda jsou kompaktibilní. Pak volám metodu `setQuery()` třídy `Sql`, která si nastaví do vlastností objektu řidící proměnnou `flags` a řetězec s dotazem. Potom zavolám metodu `sql.parse()`, která uloží do vlastností analyzovaný dotaz. Následuje analýza XML dokumentu na základě SQL dotazu. Zakončeno je to tiskem XML cílovehé dokumentu. Použití modulu `getopt` mi přišlo zbytečné.

2.3 Funkce pro zpracování chyb

V případě chyby se volá funkce `errExit()`. Jejím prvním parametrem je řetězec, který se tiskne na `stderr`. Druhý je návratová hodnota programu, která je uložená ve třídě `ECODE`. Python nemá výčtový typ.

3 Analýza SQL dotazu

SQL dotaz musí vyhovovat následujícímu regulárnímu výrazu. Trošku se o tom rozepíši, jelikož z mých osobních zkušeností se dobře píší ale špatně čtou.

Reg. výraz	**Popis**
`ELEMENT = '(\w+|\w+\.\w+|\.\w+)'`	Element je buď slovo, nebo slovo s tečkou uprostřed, nebo s tečkou na začátku.
`G1 = '^\s*SELECT\s+(\w+)'`	Gramatika začíná klíčovým slovem SELECT následováno nějakým elementem.
`G2 = '(\s+LIMIT\s+\d+)?'`	Nepovinné klíčové slovo LIMIT následováno číslem.
`G3 = '\s+FROM\s+' + ELEMENT`	Povinné klíčové slovo FROM následováno elementem.
`G4 = '(\s+WHERE\s+.*?)?'`	Nepovinné klíčové slovo WHERE následováno takřka čímkoli. Není to ale **hladové** vyhledávání, tudíž **„nesežere“** případné ORDER BY.
`G5 = '(\s+ORDER\s+BY\s+'+ ELEMENT +'\s+(ASC|DESC))?\s*$'`	Nepovinné klíčové slovo ORDER BY, následováno elementem podle kterého se řadí buď vzestupně, nebo sestupně.
`G = G1 + G2 + G3 + G4 + G5`	Výsledná gramatika

Tím bych měl část analýzy SQL dotazu. Chybí jen analyzovat jeho podmínku, o tu se stará třída `ExpParse`. Inicializuji ji a zavolám její metodu `parse(string)`, které předám řetězec s podmínkou SQL dotazu. Výsledkem je podmínka v postfixové notaci – `rightParse`. Veškeré elementy, pravý rozbor a čísla si uložím do vlastností objektů. Návratovou hodnotou je proměnná `flags`, ve které jsou zaznamenány nepovinné hodnoty.

4 Analýza podmínky SQL dotazu

Pro zpracování výrazů jsem využil precedenční analýzu. Nejdříve spustím lexikalní analýzu výrazu, ve které ukládám tokeny do fronty `queue`. Tu pak používám v precedenční analýze, která využívá následující tabulku na rozhodnutí další operace:

	`not`	`and`	`or`	`(`	`)`	`id`	`$`
`not`	S	R	R	S	R	S	R
`and`	S	R	R	S	R	S	R
`or`	S	S	R	S	R	S	R
`(`	S	S	S	S	SS	S	
`)`	R	R	R		R		R
`id`	R	R	R		R		R
`$`	S	S	S	S		S	OK

S – Shift; R – Reduce; SS – Specialní shift; OK – vše v pořádku; Prázdno – Chyba

Levý sloupec – nejvrchnější terminál na zásobníku; První řádek – vstupní token.

Po úspěšném proběhnutí precedenční analýzy vracím pravý rozbor.

5 Analýza XML souboru

O analýzu XML souboru se stará modul `xml.etree.ElementTree` a mnou vytvořená třída `Xml`.

Nejdříve použiji jeho funkci `parse()`, která mi vytvoří ze souboru strom. Naleznu kořenový element (FROM). V něm pak hledám položky (SELECT). Pak jednotlivé položky vybírám podle toho, zda vyhovují podmínce (WHERE). Nakonec je seřadím lexiograficky pomocí klíčové funkce `sorted()` (ORDER BY). Výsledek je pak uložen v seznamu `Xml.result`.

Tisk hodnot ze seznamu `Xml.result` se provádí buď do souboru, nebo na standardní výstup. Záleží na spuštění programu. Byla-li zadána hodnota LIMIT n pak se vybere prvních n položek.

6 Závěr

Toto byl můj první program, ve kterém jsem použil objektově orientované programování. Snažil jsem se rozdělit celý porblém na podproblémy – analýza SQL; precedenční analýza; samotná analýza XML souboru. Podle toho jsem vytvořil i třídy, které následně celý kód dost zpřehlednily.