DBMS 에서의 정규표현식 검색기능 구현

윤기태* 김성탄* 이상원*
*성균관대학교 정보통신공학부
e-mail : kardiell@skku.edu

Implementation of Regular Expression Searching in DBMS

Yun Gi-tae*, Kim Sung-Tan*, Lee Sang-Won*
*Dept. of Electrical and Computer Engineering, Sungkyunkwan University

요 약

DBMS 에서 사용되는 SQL 의 표준으로는 검색에 관련해서 LIKE 만을 명시하고 있다. LIKE 는 2 종류의 와일드 카드 문자를 사용한다. 하지만 두 가지만으로는 사용자의 다양한 검색 요구에 응하기 어렵다. 그 해결방법으로 LIKE 를 보완할만한 기능을 가진 정규표현식 검색을 제안하는 바, 이를 DBMS 에 추가적으로 구현하는데 있어 고려해야 할 사항을 정리한다.

1. 서론

데이터베이스 관리 시스템(이하 DBMS) 에서는 데이터를 검색하기 위해 LIKE 라는 SQL 문법을 사용한다. 보다 기능적인 검색을 하기 위해 문자를 나타내는 '_' 와 문자열을 나타내는 '%' 두가지 와일드 카드를 사용한다. 반면에 정규표현식은 몇 가지 특수 문자와 규칙을 사용해서 LIKE 보다 강력한 표현이 가능하다. 예를 들어 LIKE 를 사용한 10 줄의 쿼리 문을 정규표현식을 사용하여 한 줄로 표현할 수 있다. 이와 같은 이유로 사용자의 요구에 따라 대형 DBMS vendor 는 정규표현식 검색 기능을 지원하고 있다. 표준이 지정되지 않은 이유로 제품마다 제각기 다른 형태로 표현되고 있는 것이 사용하기에 불편한 점이다. 그러나 국산 DBMS 는 아직 정규표현식 검색을 지원하고 있지 않으며 이는 외산 DBMS 가 선점하고 있는 시장에서는 약점이 될 수 있다. 정규표현식을 사용한 기존의 쿼리를 재활용 할 수 없기 때문에 전부 LIKE 로 바꾸어 새로 쓰는 비용이 적지 않기 때문에다. 따라서 본 논문에서는 국산 DBMS 에서 필요한 정규표현식 처리 기능을 구현할 때 고려 해야 할 점을 기술한다.

2. 정규표현식을 처리할 때 고려해야 할 사항

2.1 정규표현식 에서 지원해야 할 특수문자의 범위

정규표현식 문법에도 여러 종류가 있다. 그 중에서 표준으로 지정된 IEEE POSIX Basic Regular Expression, IEEE POSIX Extended Regular Expression(표 1) 이 가장 많이 사용되고 있다.

표 1 : POSIX Regular Expression

Operator	Description
POSIX Basic Regular Expression	
₩	Escape 문자
*	0 번 이상의 반복
^	String 의 시작. [] 안에선 NOT 의 의미
$	String 의 끝
.	NULL 의 제외한 Any character
[]	Bracket 안의 문자 중 한가지만 맞으면 매칭 [0-9]처럼 범위를 지정하는 0-9 까지 중 한가지 만 맞으면 매칭된다.
()	Grouping expression
{m}	m 번 반복
{m,}	적어도 m 번 반복
{m,n}	m 번 이상 n 번 이하
₩n	Backreference. 매칭된 Subexpression('()' 에 매칭된 스트링)을 버퍼에 저장해서 활용 한다. N 은 나타난 괄호의 순서번호로 매겨진다.
[..]	Specifies one collation element, and be a multicharacter element
[:cc:]	Charater classes 에 속한 character 매칭
[==]	Specifies equivalence classes
POSIX Extended Regular Expression	
+	1 번 이상의 반복
?	0 또는 1 번의 반복
|	Alternation operator

2.2 정규표현식 에서 지원해야 할 함수의 종류

정규표현식을 활용한 검색에서 해당 레코드가 조건에 해당하는지 여부를 구분한다면 condition 으로 구현되어야 한다. 그 외 부가기능으로 패턴 치환, 패턴 위치 등을 지원하는 기능은 함수로 구현되어야 한다. Condition 은 함수와 달리 not 연산자가 적용 될 수 있다.

본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었음 (NIPA-2009-(C1090-0902-0046))

2.3 문자 인코딩 고려

서로 다른 길이의 character set 을 구분 할 수 있어야 한다. 한글(2byte)과 알파벳(1byte)이 섞여 있더라도, 정규표현식의 문법이 문자 단위로 적용되어야 한다. 또한 레코드의 인코딩과 정규표현식의 인코딩이 다른 경우는 유니코드와 같이 각국의 언어를 표현할 수 있는 character code 로 일치시켜야 문자비교가 가능할 것이다.

2.4 정규표현식을 유의미한 단위(토큰) 으로 구분

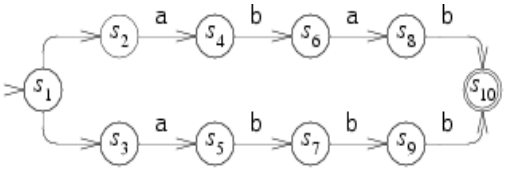
a, [a], Wa 는 다르게 표현되었지만 모두 같은 값을 나타내는 표현이다. 정규식 처리기에서 이를 모두 '알파벳 소문자 a' 라는 하나의 의미단위로 표현하고 사용하는 것이 필요하다.

2.5 토큰의 형태

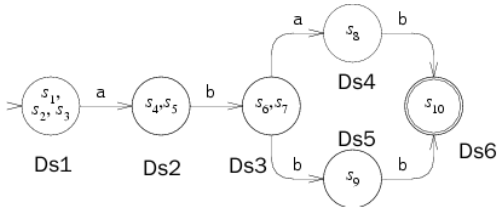
하나의 토큰이 1 개의 문자 값을 갖는 것이 아니라 MIN/MAX 두 개의 값을 갖게 하면 오토마타 머신(automata machine) 에서 상태(state) 수를 크게 줄일 수 있다. 예를 들어 [0-9]라는 표현을 1 개의 문자 값으로 나타내려면 [0123456789] 10 개의 상태가 필요하다. 하지만 두 개의 값으로 표현하면 MIN=0, MAX=9 으로 표현하면 1 개의 상태크기는 커지지만 전체 상태의 수가 줄어든다. 메모리도 절약하고 탐색시간도 줄이는 효과가 있다.

2.6 사용하는 오토마타 머신

기본적으로 그림 1 의 NFA 와 그림 2 의 DFA 두 종류가 있다. NFA 는 구성 비용이 적게 들지만 검사 비용이 많이 든다. DFA 는 NFA 로 부터 다시 만들어지기 때문에 초기 구성 비용이 들지만 검사할 때는 훨씬 빠르다. 하나의 정규표현식으로 다수의 레코드를 검사한다면 DFA 를 사용하는 것이 바람직하다.



(그림 1) NFA - abab|abbb



(그림 2) DFA - abab|abbb

2.7 인덱스의 사용

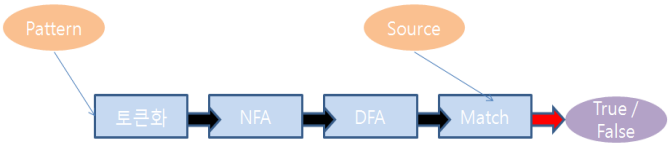
결론부터 말해 정규표현식에서는 인덱스를 사용하지 않는다. LIKE 는 패턴의 앞이 상수 문자라면 인덱스를 활용하여 Key Range 를 계산, 불필요한 레코드 읽기 회수를 줄일 수가 있다. 하지만 정규표현식 검색은 인덱스를 활용하려면 여러 가지 제약이 따른다. 시작 부분을 고정시키는 특수문자 '^' 가 사용돼야 하고 시작부분이 '|' 혹은 '[]' 에 관련 되어 있지 않아야 인덱스를 이용할 수 있다. 이는 정규표현식 중에 아주 제한적인 경우이기 때문에 인덱스를 사용하지 않아도 정규표현식 검색 성능에는 영향이 없다.

2.8 메모리 누수 관리

일반적인 경우 하나의 정규표현식으로 다수의 레코드를 검색할 때 한번 구성도니 DFA 를 재활용하므로 메모리 문제가 없다. 그러나 정규표현식이 레코드에 포함되어 있거나 bind variable 로 주어지면 매번 비교할 때 마다 DFA 가 생성되며 이 DFA 는 한번 탐색되고 사용되지 않게 된다. 따라서 레코드가 많은 경우에는 메모리 부족을 일으킬 수 있다. 따라서 이 경우에 함수 종료시 동적 메모리를 해제하는 코드를 따로 추가할 필요가 있다.

3. 정규표현식을 처리기의 구성

위와 같은 점들을 고려하여 정규표현식을 처리하는 시스템을 구현할 때 그림 3 과 같은 구성이 필요하다. 주어진 Pattern(정규표현식)을 의미 있는 단위인 토큰으 로 변환하여 이를 다시 해석하여 NFA 로 구성한다. NFA 를 사용하여터 e-closure 로 묶인 DFA 를 만들어 낸 후 source(레코드)가 주어지면 완성된 DFA 에 대입하여 true/false 를 결정한다. 이후에 주어지는 레코드에 대해서는 이미 완성된 DFA 를 사용하여 결과를 도출한다.



(그림 3) 정규표현식 처리기 구성

4. 결론

DBMS 에서는 데이터를 검색하기 위해 LIKE 라는 SQL 문법을 사용한다. 하지만 정규표현식을 사용한면 LIKE 를 사용할 때 보다 간결한 표현을 사용하여 복잡한 검색을 할 수 있다. 따라서 기존 DBMS 에 정규표현식을 처리하는 기능의 구현이 필요하며 이를 위해서는 본문에서 언급한 고려해야 할 점이 있음을 알 수 있다. 향후 연구로 본 논문에서 구성한 기능을 실제로 구현하고 성능평가를 수행하겠다.

참고문헌

[1] Jeffrey E.F. Friedl "Mastering Regular Expressions"
[2] Peter Linsley "Introducing Oracle Regular Expressions"
[3] Russ Cox "http://swtch.com/~rsc/regexp/"