제 26 회 한국정보처리학회 추계학술발표대회 논문집 제 13 권 제 2 호 (2006. 11)

범위 술어에 대한 문자열 선택도 추정 구현

김재명*, 이미영**, 이상원*
*성균관대학교 정보통신공학부
**알티베이스 개발본부
e-mail: jam02@skku.edu

Implementation of String Selectivity Estimation for Range-based Predicate

Jae-myung Kim*, Mi-young Lee**, Sang-won Lee*
*School of ICE, Sungkyunkwan University
**ALTIBASE Research Center

요 약

범위 술어에 대한 문자열 선택도 추정은 해당 문자열 범위를 숫자 표현으로 변환 해야 하는 어려움이 있다. 하지만 문자열을 숫자 표현으로 변환할 경우 각각의 바이트에 대한 모든 경우의 수를 모두 고려해야 한다. 따라서 변환 시 문자열 뒷부분에 대한 정보를 고려할 수 없는 문제가 발생한다. 최근 연구되고 있는 부분 문자열에 대한 선택도 추정 방식을 적용할 경우 통계정보와 추정에 대한 연산이 증가되는 단점이 있다. 따라서 이는 범위 술어에 대한 추정만을 위해 사용하기에는 적합하지 않다. 따라서 이 논문에서는 B+ Tree 인덱스의 제한적인 통계정보만을 가지고 범위 술어에 대한 문자열 선택도를 추정하는 방법으로 알티베이스에 구현하였다.

1. 서론

문자열 애트리뷰트(attribute)에 대한 선택 술어(selection predicate)는 관계형 데이터베이스에서 널리 쓰이고 있다. 특히 특정 범위에 대한 선택의 경우 숫자 혹은 날짜형 애트리뷰트에 대해서는 해당 애트리뷰트가 의미하는 총 범위를 고려할 수 있지만, 문자열에 대해서 적용하기는 어려움이 있다.

이 논문에서는 기존의 선택도 추정 루틴을 그대로 유지하면서 최소한의 통계정보만을 이용하여 문자열에 애트리뷰트에 대해 더 나은 선택도를 추정하는 방법을 제안하고 알티베이스에 구현하였다.

문자열 애트리뷰트에 대한 상대적인 범위를 구하려면, 문자열이 정렬되는 순서를 고려한 숫자 표현으로 변환이 필요하다. 보통 문자열 앞부분의 문자열의 아스키 값을 숫자로 변환하는데, 해당 애트리뷰트에 숫자를 의미하는 문자열 데이터가 포함되어 있는 경우 선택도 추정의 정확도가 많이 떨어진다. 하지만 실제로 문자열 애트리뷰트에 숫자를 의미하는 데이터를 저장하고자 하는 요구사항은 빈번한 것이어서 이를 개선하기 위한 구현 사례 또한 소개한다.

이 논문의 구성에 대해 간단히 소개하면, 2 장에서는 기존의 범위 술어에 대한 선택도 추정알고리즘과 부분 문자열과 관련된 술어에 대한 선택도 추정알고리즘에 대해 알아본다. 3 장에서는 해당 문자열 애트리뷰트의 의미를 고려하여 문자열을 숫자 표현으로 변환하는 방법과 한계, 단일 술어에 대한 선택도 추정 방식 그리고 복수의 술어가 오는 경우와 호스트 상수 항에 호스트 변수가 오는 경우의 처리에 대해 간단히 소개한다. 마지막으로 4 장에서 구현 결과에 대한 검증 방법을 소개하면서 마무리한다.

2. 관련 연구

2.1. 문자열 선택도 추정의 한계

문자열 애트리뷰트 범위 값에 대한 선택도를 추정하기 위해 일반적으로 사용하는 방식은 문자열을 숫자 표현으로 변환하는 것이다.

위에서 설명한 바와 같이 기존의 방식의 문제점은 문자열 숫자 표현의 크기가 한정되어 있어 긴 문자열의 순서를 표현하지 못한다는 것이다. 예를 들어 문자열의 앞부분이 많이 중복되는 경우 그에 대한 선택도를 제대로 추정하지 못한다. 또 다른 문제점은 부

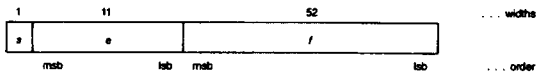
분 문자열에 대한 선택도를 추정하지 못한다는 것인데, 그에 대한 부분은 이 논문에서 다루지 않는다.

```
(1) Extract a maximum of 32 bytes from
    the attribute.
(2) Extract the first 15 bytes from the
    32, padding with zeros at the right if
    necessary.
(3) Convert the 15 bytes hex number to
    decimal and round to 15 significant
    figures.
```

(그림 1) 문자열의 숫자 표현 (오라클)

현은 분수형태를 표현하거나 소수점을 표현할 수 있는 실수형 데이터 타입이어야 한다. 또한 일반적으로 C 로 구현된다고 가정하면 C 의 내장형 데이터 타입은 double 형을 이용하게 되는데, 실제로 floating point 형의 mantissa 부분에서 표현할 수 있는 경우의 수를 고려해야 한다[9]. exponent 값을 이용하여 실수 값의 크기를 조정할 수 있으나 근본적으로 값의 정확도는 mantissa 가 결정하기 때문이다. 그와 같은 이유로 오라클®에서는 그림 1 과 같은 방법으로 문자열에 대한 범위 값의 선택도를 추정한다. [2]

간단히 정리하여 위의 과정은 15 자리의 10 진수가 표현할 수 있는 범위의 값만 숫자표현으로 변경되는 것이다. 10 진수 15 자리가 표현하는 가장 큰 값은 0x38D7EA4C67FFF (=999,999,999,999,999)과 같다. 결론적으로 64 비트 실수형 타입이 표현할 수 있는 52 비트(16 진수 13 자리)의 가수(mantissa)를 고려하고 있다.



(그림 3) IEEE Standard 754 – Double Format

2.2. 부분 문자열에 대한 선택도 추정

최근에는 서픽스 트리(suffix tree) [6], [7] 혹은 q-gram 테이블[8]을 이용하여 부분 문자열에 대한(예: il LIKE '%str%') 선택도를 추정하려는 연구도 진행중이다. 서픽스 트리는 부분 문자열을 인덱싱하는 대표적인 자료구조이다. 서픽스 트리를 구성하고 해당하는 부분 문자열에 대한 발생빈도를 유지하여 해당 부분 문자열의 빈도를 찾는다. Q-gram 또한 이와 비슷한 방식으로 각각의 부분 문자열의 길이에 따른 발생빈도를 저장하고 각각의 부분 문자열을 조합하는 과정을 거쳐 원래의 부분 문자열에 대한 선택도를 추정한다. 하지만 이러한 방식은 처음에 언급한 바와 같이 부분 문자열의 선택도를 추정하는 방식으로 경우에 따라 원본데이터보다 통계정보가 커지는 문제점이 있다. 따라서 이는 범위검색에 사용하기에는 통계정보의 크기가 너무 무겁다. 오히려 기존의 숫자 표현으로 변환한 값에 히스토그램을 적용하여 데이터 분포에 대한 정보를 구축하여 범위 검색에 문자열을 숫자 표현으로 변환하려면 해당 숫자 표

```
(1) Extract a maximum 15 bytes from the
    attribute.
(2) Read min, max, and boundary values
    and determine the attribute type.
(3) If type is numeric, convert 15 bytes
    numeric string to double value.
(4) Otherwise, extract 52 bits and
    convert to double value
```

(그림 2) 문자열의 숫자 표현 (알티베이스)

활용하는 것이 적합하다.

3. 디자인 및 구현

3.1. 문자열의 숫자 표현

이 논문에서는 문자열 애트리뷰트가 숫자 의미의 문자열이 담고 있는 경우에는 해당하는 숫자로 변환되도록 고려하였다. 실제로 문자열 애트리뷰트에 DATE 타입의 데이터를 담는 경우가 있는데, 예를 들어 생산라인의 데이터베이스 시스템에서 기계적인 동작 오류로 인하여 DATE 타입에서 입력이 불가능한 값(예: 42 시와 같은)이 나오는 경우가 있다. 따라서 일반적인 DATE 타입으로 데이터를 저장하지 못하고, CHAR(16) 타입으로 데이터를 저장하고 있다. 이러한 상황에서 기존의 방법으로 선택도를 추정 할 경우 문자열이라 인식하고 선택도를 추정하기 때문에 DATE 타입으로 데이터를 저장하는 경우와 비교하여 정확하지 않는 선택도를 추정하게 된다. – 보통 문자열의 경우 앞부분 6 바이트에 대한 값이 숫자로 변환되는데, 날짜형 데이터를 고려하여 일(day) 단위 혹은 시간(hour)단위의 범위를 검색하는 경우의 값을 숫자로 변환되지 못한다. -- 이러한 문제를 해결하기 위해 문자열 애트리뷰트에 숫자 의미를 담은 데이터가 담긴 경우에 해당 문자열이 의미하는 경우 해당 문자열이 의미하는 숫자로 변환하는 방식을 적용하였다.

문자열에 포함하는 의미는 정규 표현식으로 쉽게 찾을 수 있다. 문자열의 의미는 질의의 의미(semantic)를 검사하는 단계에서 사용되는 방법과 동일한 과정으로 검사할 수 있다. 예를 들어 10 진수의 경우 [0-9]*로 표현이 가능하고, 16 진수는 [0-9A-F,a-f]*로 표현할 수 있다. 실수의 경우 또한 표현이 가능하다. 하지만 문자열의 의미를 파악할 때 주의할 점은 원래 문자열이 의미하는 문자열의 순서를 유지해야 한다는 것이다. 예를 들어 '1000'라는 문자열과 '200'이라는 문자열이 있을 경우 그 숫자가 의미하는 값으로는 1000 이 더욱 큰 수이지만, 실제로는 상대적 위치를 고려하여 '200'이 더 큰 수로 인식되어야 한다. 실수의 경우 소수점을 의미하는 '.'문자가 문자열 상의 같은 위치에 존재하는 경우에만 숫자로 변환할 수 있다. 이러한 문제점을 고려하여 구현에서는 10 진수와 16 진수를 의미하는 경우에만 해당 문자열

의 의미에 근접한 문자열로 변환하였다. 문자열에 대한 선택도를 추정하는 방법은 그림 2 와 같다.

실제로 문자열의 종류를 판단하는 과정에서는 최대, 최소 그리고 범위를 나타내는 상수만을 가지고 판단을 하는데, 최대, 최소값은 각각의 바운드를 의미하므로 값의 판단의 정확도가 높다. 설령 범위내의 숫자가 아닌 값이 포함되어 있더라도 최대, 최소 범위의 문자열이므로 선택도 추정에는 큰 영향을 미치지 않는다.

3.2. 타입의 고려

구현을 적용한 문자열 타입은 CHAR(n)과 VARCHAR(n) 타입이다. 두 타입을 거의 동일하나 실제 값을 저장하는 구조에서 약간의 차이를 가진다. CHAR(n) 타입의 경우 항상 길이 n 만큼의 공간을 할당하여 저장한다. 반면 VARCHAR(n)의 경우 길이 n 을 넘지 않는 문자열에 대해 가변적인 길이로 값을 저장한다. 따라서 CHAR(n)타입의 경우 실제로 저장되는 값 이외의 문자를 공백을 채우게 된다. 만약 문자열 길이보다 작은 숫자 문자열이 저장되는 경우 일반 문자열로 인식하게 된다. 따라서 인위적으로 문제를 해결하기 위해서는 문자열 길이에 맞게 '0'으로 채워주어야 한다.

3.3. 단일 술어에 대한 선택도 추정

문자열의 숫자표현에 따라 선택도가 결정되는 술어는 기본 술어(<, >, <=, >=)와 BETWEEN, LIKE 등의 술어이다. 반대로 =, !=, IS NULL, IN 과 같은 술어들은 타입에 무관하게 선택도를 추정할 수 있는 술어에 포함된다. 따라서 이 논문에서는 대소를 비교하는 기본 술어(<, >, <=, >=)와 BETWEEN, LIKE 에 대한 선택도 추정에 대해서만 고려한다.

<표 1> 단일 술어에 대한 선택도 추정

Predicate	Case	Estimation Rule
<, >	I1 < x x > I1	(x-MIN) / (MAX-MIN)
	I1 > x X < I1	(MAX-x) / (MAX-MIN)
<=, >=	I1 <= x x >= I1	(x-MIN) / (MAX-MIN) + 1/CARD
	I1 >= x x <= I1	(MAX-x) / (MAX-MIN) + 1/CARD
BETWEEN	I1 BETWEEN x AND y	(y-x) / (MAX-MIN) + 2/CARD
NOT BETWEEN	I1 NOT BETWEEN x AND y	1 - ((y-x) / (MAX-MIN) + 2/CARD)
LIKE	I1 LIKE x%	(y-x) / (MAX-MIN) + 1/CARD
NOT LIKE	I1 NOT LIKE x%	1 - ((y-x) / (MAX-MIN) + 1/CARD)

기본 술어와 BETWEEN 의 경우 표 1 에 기술된 것과 같은 방식으로 값을 추정할 수 있다.

3.4. 다중 술어에 선택도 추정

마지막으로 단일 술어가 모여 범위를 형성하는 경우의 선택도는 일반적으로 사용하는 것과 같이 위의 표 3 과 같이 경계의 종류에 따라 4 가지로 분류하여 추정하였다.

<표 3> 다중 술어에 대한 선택도 추정

Case	Example	Estimation Rule
open, open	i1 > x AND i1 < y	(y-x) / (MAX-MIN)
open, closed	i1 > x AND i1 <= y	(y-x) / (MAX-MIN) + 1/CARD
Closed, open	i1 >= x AND i1 < y	(y-x) / (MAX-MIN) + 1/CARD
Closed, closed	i1 >= x AND i1 <= y	(y-x) / (MAX-MIN) + 2/CARD

3.5. 문자열 질의의 변환

위의 표 1 에서 보는 바와 같이 LIKE 의 경우 문자열의 앞부분에 고정된 값이 존재하는 경우에 범위 값으로 변환이 가능하다.

<표 2> 문자열 질의의 변환

Original Query	Transformed Query
I1 BETWEEN x AND y	I1 >= x AND I2 <= y
I1 LIKE 'str'	I1 = 'str'
I1 LIKE 'str%'	I1 >= 'str' AND I1 < 'sts'

BETWEEN 과 LIKE 술어 처리하는 방법은 두 가지가 있다. 첫째는 기본 술어 형태로 질의를 변환하는 방법이다. 이 방법은 논리 최적화(Logical Optimization) 단계에서 처리하는 경우에 속한다. 두 번째 방법은 질의를 변환하지 않고, 해당 술어에 대한 선택도를 각각 추정하는 방법이다. 질의를 변환하는 방법은 표 2 에 기술되어 있다.

일반적으로 질의를 변환하는 이유는 변환된 질의를 처리할 때 더 좋은 성능을 보이기 때문이다. 위의 경우 질의를 변환할 경우 기존의 선택도 추정 (기본 술어에 대한 선택도 추정)을 수정하지 않고, 같은 방법으로 선택도를 추정할 수 있다. 하지만 실제의 구현에서 LIKE 술어에 아래와 같은 두 가지 문제가 있어 두 번째 방법을 선택하였다.

첫 번째 문제는 호스트 변수에 대한 고려이다. 알티베이스 질의 처리기는 상수 항에 호스트변수가 오는 경우 실행(Execution) 직전의 단계에서 선택도를 다시 계산하여 실행 계획(Execution Plan)을 최종 조율하는 단계가 있다. 그러나 LIKE 의 경우 문자열 패턴에 상수가 아닌 호스트변수가 오는 경우 패턴의 형태를 미리 알 수 없기 때문에 표 2 에 나오는 변환을 적용할 수 없다. 따라서 LIKE 에 대한 질의는 이를 변환하지 않고 패턴의 상수를 분석하여 적절한 선택도를 계산하는 방식을 적용하였다.

두 번째 문제는 지역 언어 지원(NLS: National Language Support)에 따른 범위 문제이다. 한글의 경우 NLS 의 선택에 같은 질의라도 다른 결과를 가지는 경우가 있다. 예를 들어 한글 초성에 대한 인코딩은 일단 한글 문자들 보다 앞쪽에 서로 모여있다. 따

라서 단순히 한글 초성만 가지고 쿼리를 날리는 경우 결과가 다르게 나올 수 있으며, 각각의 NLS 설정에 따라 인덱스의 정렬순서가 다르게 설정되도록 디자인 되었다. 이러한 순서를 고려하며 쿼리를 변환할 수 있으나, NLS 값과 쿼리의 변환은 서로 종속성을 가지지 않는 방향으로 가는 것이 적합하다 판단하였다.

결론적으로 기본술어, BETWEEN, LIKE 의 경우를 나누어 선택도를 추정하였지만, 범위 값에 대한 계산 과정에서는 모두 그림 2 의 알고리즘이 적용되어 숫자 값의 경우 선택도 추정을 위해 사용되는 문자열의 길이를 확장할 수 있는 방법이 적용되었다.

4. 검증

구현한 결과를 검증하기 위해 사전에 구축된 소프트웨어 테스트 자동화 프레임워크[10]를 이용하였다. 테스트에는 크게 성능을 측정하기 위한 테스트와 기능을 검증하기 위한 테스트가 있다. 성능의 경우 구현 범위가 크지 않아 원하는 대로 작동한다면 특정 문자열에 대해 더 정확한 범위의 선택도를 추정할 수 있다.

기능 테스트는 사전에 구축된 테스트 케이스를 통과한 뒤 각각의 술어에 맞는 테스트 케이스를 작성하였다.기본 술어에 대해서는 10 진수 의미를 가지는 경우, 16 진수 의미를 가지는 경우, 닫혀있고 열려있는 각각의 경우, 통합 선택도, 범위 검색 및 경계 값의 초과하는 상황에 대한 처리(예를 들어 상한 값이 애트리뷰트 최대값보다 커지는 경우)에 대하 테스트를 수행하였다. 그 밖에 BETWEEN 및 LIKE 의 경우에는 일치하는 패턴(exact match), 전위 패턴(prefix match) 및 일반 패턴에 대해 검사를 수행하였고, 기본 술어와 같이 각각의 경우에 대한 테스트를 수행하였다.

5. 결론

범위 술어에 대한 문자열 선택도 추정은 해당 문자열 범위를 숫자 표현으로 변환 해야 하는 어려움이 있었다. 하지만 문자열을 숫자 표현으로 변환할 경우 각각의 바이트에 대한 모든 경우의 수를 모두 고려해야 한다. 따라서 변환 시 문자열 뒷부분에 대한 정보를 고려할 수 없는 문제가 발생한다. 최근 연구되고 있는 부분 문자열에 대한 선택도 추정 방식을 적용할 경우 통계정보와 추정에 대한 연산이 증가되는 단점이 있다. 따라서 이는 범위 술어에 대한 추정만을 위해 사용하기에는 적합하지 않다. 따라서 이 논문에서는 B+ Tree 인덱스의 제한적인 통계정보만을 가지고 범위 술어에 대한 문자열 선택도를 추정하는 방법으로 알티베이스에 구현하였다.

구현 과정에서 LIKE 술어의 경우 호스트 변수과 지역 언어 지원의 문제 등을 고려하여 질의 변환을 수행하지 않고, 선택도를 추정하는 방식을 선택하였다.

하지만 여전히 문자열을 숫자 표현을 숫자로 가정하고 히스토그램을 적용하는 경우에는 숫자를 의미하는 문자열의 경우에도 일반문자열 범위의 선택도로 추정해야 하는 어려움이 있다.

Acknowledgment

본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었음. IITA-2006-(C1090-0603-0046)

연구 기간 내내 물심양면으로 도움을 주신 개발본부장님을 비롯한 모든 분들과 개발 2 팀 선배님들께 감사의 말을 전합니다.

참고문헌

[1] ALTIBASE 4 User's Manual: Administrator's Manual (http://data.altibase.com/pdf/a4/4.1.3.Admin.pdf)

[2] Cost-Based Oracle Fundamentals, Jonathan Lewis, Apress 2006 (ISBN : 1590596366).

[3] Ioannidis, Y. E. and Poosala, V. "Balancing histogram optimality and practicality for query result size estimation," In Proceedings of the ACM SIGMOD 1995.

[3] Matias, Y., Vitter, J. S., and Wang, M. "Wavelet-based histograms for selectivity estimation", In Proceedings of the SIGMOD pp 448-459., 1998.

[4] Lipton, R. J., Naughton, J. F., and Schneider, D. A. "Practical selectivity estimation through adaptive sampling", In Proceedings of the ACM SIGMOD 1990.

[5] Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., and Price, T. G. "Access path selection in a relational database management system", In Proceedings of the ACM SIGMOD 1979.

[6] P. Krishnan, J. S. Vitter, and B. R. Iyer, "Estimating alphanumeric selectivity in the presence of wildcards", In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, pages 282—293, 1996.

[7] H. V. Jagadish, O. Kapitskaia, R. T. Ng, and D. Srivastava, "One dimensional and multi-dimensional substring selectivity estimation", The VLDB Journal (2000) 9, pages 214—230, 2000.

[8] Surajit Chaudhuri and Venkatesh Ganti and Luis Gravano, "Selectivity Estimation for String Predicates: Overcoming the Underestimation Proble", Proceedings of the 20t ICDE, page 227, 2004.

[9] IEEE Standards Committee 754, "IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Standard 754-1985.

[10] C. Rankin, "The Software Testing Automation Framework", IBM SYSTEMS JOURNAL 2002, VOL 41, PART 1 126-139