연산자 LIKE의 새로운 한글 탐색 패턴
(A New Korean Search Pattern of the Operator LIKE)

박 성 철 † 노 은 향 †† 박 영 철 ††† 박 종 철 ††††
(Sung Chul Park) (Eun Hyang Roh) (Young Chul Park) (Jong Cheol Park)

요 약 데이타베이스 언어인 SQL의 연산자 LIKE는 문자열을 탐색하기 위한 연산자로서 문자열 양식을 설정함으로써 그에 부합하는 칼럼값들을 식별할 수 있게 한다. 표음문자인 한글의 각 음절은 초성과 중성으로 구성되거나 초성, 중성, 그리고 종성으로 구성된다. 본 논문은 연산자 LIKE의 한글 음절의 탐색 양식으로서 한글 음절로 표현되는 기존 양식에 추가하여 한글의 초성과 중성에 기반한 새로운 양식을 제안한다. 제안하는 한글 탐색 양식은 특정 초성을 가지는 한글 음절들, 특정 중성을 가지는 한글 음절들, 또는 특정 초성과 중성을 가지는 한글 음절들을 탐색할 수 있게 한다. 제안하는 한글 탐색 양식을 SQL의 기존 연산자들로 표현하는 것은 실질적으로 많은 불편을 수반하며 DBMS의 문자 집합에 따라 응용 프로그램의 호환성 문제를 초래할 수 있다. 본 논문은 제안하는 한글 탐색 양식을 고려한 연산자 LIKE의 수행 알고리즘을 한글과 한자에 대한 정보 교환용 부호계의 국가 표준인 KS X 1001로 표현된 문자들을 기반으로 제시한다.

키워드 : SQL, LIKE, 문자열 양식, KS X 1001, 한글

Abstract The operator LIKE of the database language SQL is a string pattern search operator. By providing the string pattern, the operator can identify column values that match with the string pattern. As a phonetic symbol, each Korean syllable is composed either of a leading sound and a medial sound or of a leading sound, a medial sound, and a trailing sound. As a search pattern of Korean syllables of the operator LIKE, in addition to the traditional Korean search pattern, this paper proposes a new search pattern that is based on leading sounds and medial sounds of Korean. With the new Korean search pattern, Korean syllables having specific leading sounds, specific medial sounds, or both specific leading sounds and medial sounds can be found. Formulating predicates equivalent with the new Korean search pattern by way of existing SQL operators is cumbersome and might cause the portability problem of applications depending on the underlying character set of the DBMS. This paper presents algorithms for the execution of the operator LIKE considering the new Korean search pattern based on the characters that are represented in KS X 1001, which is a Korean standard code for information interchange of Korean and Chinese.

Key words : SQL, LIKE, string pattern, KS X 1001, Korean

1. 서 론

SQL의 연산자 LIKE는 문자열을 탐색하기 위한 연산자이다. 연산자 LIKE는 문자열 양식을 통해 그 문자열 양식에 부합(符合, match)하는 칼럼값을 가진 레코드들을 식별할 수 있게 한다. SQL 표준[1,2]은 일반 문자들과 예약된 문자(reserved character)인 '%'와 '_'를 연산자 LIKE의 문자열 양식의 양식 문자들로 허용한다. 예약된 문자인 '%'와 '_'는 각각 임의의 수효의 임의의 문자들 그리고 하나의 임의의 문자와 부합한다. SQL 표준은 이들 두 예약된 문자들을 일반 문자로 처리하기 위하여 예약된 문자 직전에 둘 수 있는 탈출문자(escape character)를 선언할 수 있게 한다.

연산자 LIKE를 영어에 적용할 경우, 그 문자열 양식은 영어의 알파벳(alphabet) 문자들을 기반으로 구성된

· 본 논문은 지방 기술혁신사업의 일환으로 지원하는 차세대선도산업기술 연구개발사업의 연구결과임
† 정 회 원 : 경북대학교 전자전기컴퓨터 교수
scmh@paran.com
†† 정 회 원 : 한국신발피혁연구소 생산시스템연구팀 연구원
soovia@nate.com
††† 종신회원 : 경북대학교 전자전기컴퓨터 교수
ycparkknu@hanmail.net
†††† 정 회 원 : (주)퓨전소프트 R&D3그룹 사원
iaminjoy@fusionsoft.co.kr
논문접수 : 2006년 12월 1일
심사완료 : 2007년 3월 16일

다. 예를 들어, 영어의 알파벳 'P'로 시작하는 문자열을 문자열 양식 'P%'로, 'Pa'로 시작하는 문자열을 'Pa%'로, 그리고 두 번째 문자가 'a'인 문자열을 '_a%'로 탐색할 수 있다. 연산자 LIKE를 한글에 적용하는 경우, 그 문자열 양식은 한글의 문자들을 기반으로 구성된다. 예를 들어, 한글 음절(音節, syllable) '박'으로 시작하는 문자열을 찾을 경우에는 문자열 양식을 '박%'로, 한글 자음(子音, consonant) 'ㅂ'을 독립된 문자로 가지는 문자열을 찾을 경우에는 '%ㅂ%'로 설정하면 된다. 그러나, SQL 표준은 한글 자음 'ㅂ'을 초성(初聲, leading sound)으로 가지는 음절들, 한글 자음 'ㅂ'과 모음(母音, vowel) 'ㅏ'를 각각 초성과 중성(中聲, medial sound)으로 가지는 음절들, 그리고 한글 모음 'ㅏ'를 중성으로 가지는 음절들을 찾을 수 있는 연산자 LIKE의 탐색 양식을 제공하지 않고 있다. 이는 연산자 LIKE를 이용한 한글 문자열의 탐색이 한글 음절의 특성을 충분히 활용하지 못하는 것임을 의미한다. 본 논문은 연산자 LIKE의 문자열 양식에서 영어 문자와 한글 문자를 탐색하기 위한 탐색 양식을 각각 영어 탐색 양식과 한글 탐색 양식이라 하며, 한글 음절을 탐색하기 위한 탐색 양식을 특별히 한글 음절 탐색 양식이라 한다.

표음문자(表音文字)인 한글의 각 음절은 초성과 중성으로 구성되거나 초성, 중성, 그리고 종성(終聲, trailing sound)으로 구성된다. 현대 한글은 자음 30자('ㄱ', 'ㄲ', 'ㄳ', 'ㄴ', 'ㄵ', 'ㄶ', 'ㄷ', 'ㄸ', 'ㄹ', 'ㄺ', 'ㄻ', 'ㄼ', 'ㄽ', 'ㄾ', 'ㄿ', 'ㅀ', 'ㅁ', 'ㅂ', 'ㅃ', 'ㅄ', 'ㅅ', 'ㅆ', 'ㅇ', 'ㅈ', 'ㅉ', 'ㅊ', 'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ')와 모음 21자('ㅏ', 'ㅐ', 'ㅑ', 'ㅒ', 'ㅓ', 'ㅔ', 'ㅕ', 'ㅖ', 'ㅗ', 'ㅘ', 'ㅙ', 'ㅚ', 'ㅛ', 'ㅜ', 'ㅝ', 'ㅞ', 'ㅟ', 'ㅠ', 'ㅡ', 'ㅢ', 'ㅣ')를 가진다. 한글의 자음들만이 한글 음절의 초성과 종성으로 사용될 수 있으며 초성으로 사용되는 자음들을 초성자음이라 하고 종성으로 사용되는 자음들을 종성자음이라 한다. 한글의 자음들 30자 중에서 19자('ㄱ', 'ㄲ', 'ㄴ', 'ㄷ', 'ㄸ', 'ㄹ', 'ㅁ', 'ㅂ', 'ㅃ', 'ㅅ', 'ㅆ', 'ㅇ', 'ㅈ', 'ㅉ', 'ㅊ', 'ㅋ', 'ㅌ', 'ㅍ', 'ㅎ')만이 초성자음들로 사용되며 27자(전체 자음들 중 'ㄸ', 'ㅃ', 'ㅉ'을 제외)가 종성자음들로 사용될 수 있다. 한글의 모음들만이 한글 음절의 중성으로 사용되며 모든 한글 모음들이 한글 음절의 중성으로 사용될 수 있다.

본 논문은 LIKE 연산에서 기존 한글 음절 탐색 양식에 추가하여 초성과 중성을 기반으로 하는 새로운 한글 음절 탐색 양식을 제안한다. 제안하는 한글 음절 탐색 양식은 특정 초성을 가지는 한글 음절들, 특정 중성을 가지는 한글 음절들, 그리고 특정 초성과 중성을 가지는 한글 음절들을 탐색할 수 있게 한다. 본 논문은 제안하는 새로운 한글 음절 탐색 양식을 초성과 중성 기반 한글 음절 탐색 양식이라 한다. 본 논문은 초성과 중성 기반 한글 음절 탐색 양식을 고려한 연산자 LIKE의 수행 알고리즘을 한글과 한자에 대한 정보 교환용 부호계의 국가 표준인 KS X 1001[3]의 한글 완성형으로 표현된 문자들을 기반으로 제시한다. KS X 1001 부호계의 한글 완성형에는 현대 한글의 자음들과 모음들 그리고 음절들이 사전적 순서에 따라 정렬되어 있으며 이들 각각에게 2-바이트(byte)의 부호값(codeword)이 설정되어 있다.[1] 본 논문은 한글의 자음들과 모음들이 정보 처리용 건반 배열의 국가 표준인 KS X 5002[6]를 따라 정렬됨을 가정한다. 즉, 컴퓨터 자판에서 초성자음과 종성자음이 구분되지 않으며 이들은 한글 음절을 완성하는 적절한 오토마타(automata)에 의해 구분됨을 가정한다. 본 논문은 한글 고어(古語)를 고려하지 않는다.

참고문헌 [7]의 "한글 두음문자열 검색방법 및 장치"는 KS X 1001 부호계의 한글 조합형을 기반으로 특정 초성들을 가지는 한글 문자들을 탐색하기 위하여 한글의 초성자음들로만 구성된 문자열 양식(예를 들어, "ㅅㅈㄷㅇ")을 제안하며 그 문자열 양식에 부합하는 문자열들(예를 들어, "세종대왕", "사장대우", "사전동의" 등)을 식별하는 방법을 제시한다.[2] 이 방법은 한글의 초성자음들로만 구성된 문자열 양식의 각 초성자음에 대해 바이트 단위의 마스크(mask)를 '1[그 초성자음의 5-비트 표현]00'으로 설정하고 그를 비교할 문자열의 각 문자와 비교한다. 예를 들어, 문자열 양식 "ㅅㅈㄷㅇ"의 초성자음들에 대한 바이트 단위의 마스크들은 각각 '10101100', '10111000', '10010100', '10110100'로 설정된다. 앞의 마스크들에서 밑줄 친 부분은 각 초성자음의 5-비트 표현을 나타낸다. 또한, 비교할 문자열의 부호계가 KS X 1001 부호계의 한글 조합형이 아닌 경우에는 부호계 변환 프로그램을 이용하여 비교할 문자열을 KS X 1001 부호계의 한글 조합형으로 변환한 후, 그 문자열의 문자들을 문자열 양식의 초성자음들에 대한 바이트 단위의 마스크들과 비교한다.

참고문헌 [7]은 문자열 양식을 SQL로 표현하는 방법을 제시하지 않으며, KS X 1001 부호계의 한글 조합형을 기반으로 하며, 문자열 양식은 초성자음들만으로 구성되며, 부호계 변환 프로그램에 대한 구체적 사항들을 제시하지 않고 있으며, 그 비용에 대한 어떠한 언급도

1) KS X 1001 부호계의 부호값은 유니코드(Unicode)[4]의 부호점(code point)과 동일한 의미이다[5].

2) KS X 1001 부호계의 한글 조합형은 하나의 한글 음절을 2-바이트로 표현하며 하나의 한글 음절의 초성, 중성, 그리고 종성을 각각 5-비트(bit)로 표현한다. 2-바이트의 각 비트 순서를 〈F, [E, D, C, B, A], [9, 8, 7, 6, 5], [4, 3, 2, 1, 0]〉로 표현할 경우, 하나의 한글 음절을 구성하는 초성, 중성, 종성의 값은 〈1, [초성], [중성],[종성]〉으로 표현된다. 예를 들어, 음절 '박'은 〈1, [초성자음ㅂ], [중성 ㅏ], [종성자음ㄱ]〉으로서 〈1, [0, 1, 0, 0, 1], [0, 0, 0, 1, 1], [0, 0, 0, 1, 0]〉으로 표현된다.

없다. 참고문헌 [7]을 제외하고는 본 논문과 관련된 내용을 가진 다른 문헌들을 찾지 못하였다. 본 논문은 초성과 중성 기반 한글 음절 탐색 양식을 연산자 LIKE의 새로운 문자열 양식으로 제안하며 그 수행 알고리즘을 KS X 1001의 한글 완성형을 기반으로 제시하고 문자열 양식은 초성자음뿐만 아니라, 중성모음, 그리고 초성과 중성의 음절을 기반으로 표현된다.

특정 초성, 특정 초성과 중성, 또는 특정 중성을 가진 한글 음절들의 탐색을 SQL 표준의 기존 연산자들을 이용하여 표현하는 방법과 정규 표현식(regular expression) 또는 연산자 LIKE의 확장된 문법을 이용하여 표현하는 방법을 고려할 수 있다.

예제 1. "직원 테이블에서 주소의 첫 글자의 초성이 'ㅂ'인 사원들을 인출하라".

KS X 1001 부호계의 한글 완성형에서 'ㅂ'을 초성으로 가지는 음절들은 음절 '바'와 음절 '빛' 사이에 위치하며 이들은 '빠' 보다 그 값이 작다. 따라서, SQL 표준의 기존 연산자들을 이용하여 작성하면 그 문은 "SELECT * FROM employees WHERE address >='바' and address < '빠';"가 된다. 그 질의문에서 LIKE 연산의 문자열 양식에 부합하는 문자열들은 '복현동', '비산동', '방이동', '본동', '봉천동' 등이다. 위 요구의 조건은 ORACLE 10g[8]의 정규 표현식을 이용한 "REGEXP_LIKE(name, '^[바-빛]')"와 SQL Server 2005[9]의 연산자 LIKE의 확장된 문법을 이용한 "name LIKE '[바-빛]%'"로 표현될 수 있다.

예제 2. "직원 테이블에서 이름이 세 개의 한글 음절들로 구성되며, 그 이름의 첫 번째 음절의 초성은 'ㅂ'이고, 두 번째 음절은 초성으로 'ㅇ'을 그리고 중성으로 'ㅕ'를 가지며, 세 번째 음절은 중성으로 'ㅓ'를 가지는 사원들을 인출하라".

위 요구를 SQL 표준의 기존 연산자들을 이용하여 표현하는 것은 현실적으로 매우 어렵거나 불가능할 수 있다. 예를 들어, KS X 1001 부호계의 한글 완성형에서 위 요구는 "SELECT * FROM employees WHERE name LIKE '___' and ((name >= '바여거' and name <= '바여겋') or (name >= '바여꺼' and name <= '바여껑') or ··· or (name >= '빛옇허' and name <= '빛옇헝')"로 표현될 수 있다. 위 질의에 부합하는 문자열들은 '박영철', '배연걸' 등이다. KS X 1001 부호계의 한글 완성형에서 초성자음 'ㅂ'을 초성으로 가지는 음절들은 129개이며, 음절 '여'의 초성자음 'ㅇ'과 모음 'ㅕ'를 초성과 중성으로 가지는 음절들은 16개이고, 모음 'ㅓ'를 중성으로 가지는 음절들의 연속된 범위들의 수효 즉, 초성자음들의 수효가 19이므로, SQL 표준의 기존 연산자들을 이용한 위 표현식은 39,216(129*16*19)개의 범위 속성들이 OR로 연결됨을 의미한다. 이는 표현식 생성의 불편함은 차치하고라도 그 표현식을 수행하는데 많은 비용이 소요됨을 의미한다. 위 요구의 조건은 정규 표현식을 이용한 "REGEXP_LIKE(name, '^[바-빛][여-옇][거-겋 | 꺼-껑 | 너-넣 | 더-덮 | ··· | 허-헝]$')"와 연산자 LIKE의 확장된 문법을 이용한 "name LIKE '[바-빛][여-옇][거-겋 | 꺼-껑 | 너-넣 | 더-덮 | ··· | 허-헝]'"로 표현될 수 있다.

예제 3. "직원 테이블에서 이름이 적어도 하나 이상의 임의의 문자로 시작하고, 그 임의의 문자들 바로 다음에 세 개의 한글 음절들이 오며, 그 한글 음절들 바로 다음에 적어도 하나 이상의 임의의 문자가 오는 그러한 이름을 가진 사원들을 인출하라. 단, 그 세 개의 한글 음절들에서 첫 번째 음절의 초성은 'ㅂ'이고, 두 번째 음절은 초성으로 'ㅇ'을 그리고 중성으로 'ㅕ'를 가지며, 세 번째 음절은 중성으로 'ㅓ'를 가진다."

SQL 표준의 기존 연산자들을 이용하여 위 요구를 만족하는 질의문을 작성하는 것은 불가능하다. 그러나, 위 요구의 조건은 정규 표현식을 이용하는 "REGEXP_LIKE(name, '(.)+[바-빛][여-옇][거-겋 | 꺼-껑 | 너-넣 | 더-덮 | ··· | 허-헝](.)+')"와 확장된 문자열 양식을 이용한 "name LIKE '%_[바-빛][여-옇][거-겋 | 꺼-껑 | 너-넣 | 더-덮 | ··· | 허-헝]_%'"로 표현될 수 있다.

정규 표현식 또는 확장된 문자열 양식을 이용하여 예제 1, 2, 3의 요구들을 표현할 수 있음에도 불구하고 그 방법은 다음과 같은 문제점들을 가진다.

첫째, 정규 표현식 또는 확장된 문자열 양식의 작성은 많은 불편을 수반하며 실수를 범할 가능성이 높다. 한글의 자음들과 모음들의 조합으로 형성될 수 있는 11,172자의 한글 음절들 중에서 KS X 1001 부호계의 한글 완성형에는 2,350자의 한글 음절들만이 정의되어 있다. 따라서, KS X 1001 부호계의 한글 완성형의 음절들을 초성자음 19자와 모음 21자에 의한 399개의 조합들로 나타내고 그 조합들의 음절들을 사전적 순서에 따라 배치할 경우, 46개의 조합들이 하나의 음절도 가지지 않으며, 5개의 조합들이 그 조합의 첫 번째 음절을 초성, 중성, 그리고 종성으로 구성된 음절로 가지며, 대부분의 조합들에서 그 조합의 마지막 음절의 종성자음은 일정한 규칙을 가지지 않는다. 따라서, 정규 표현식을 이용하여 문자열 양식을 작성하기 위해서는 그 조합들을 구축한 후 그를 참조하여 문자열 양식을 작성하여야 한다. 이는 매우 불편하며 그 문자열 양식의 작성시 오류를 범할 가능성이 높다.

둘째, SQL 응용 프로그램의 호환성 문제를 초래할 수 있다. 지원되는 한글 음절들은 문자 집합에 따라 다를 수 있다. 예를 들어, 초성자음 'ㅂ'과 모음 'ㅣ'를 초

성과 중성으로 가지는 한글 음절들의 범위는 유니코드의 경우에는 [바-빟]이지만 KS X 1001 부호계의 한글 완성형의 경우에는 [바-빛]이다. 따라서, 유니코드를 기반으로 작성된 SQL 응용 프로그램을 KS X 1001 부호계를 기반으로 하는 환경에서 실행하고자 하는 경우 또는 그 반대의 경우, 응용 프로그램이 수정되어야 한다. 이는 그 응용 프로그램이 호환성 문제를 가짐을 의미한다.[3)]

따라서, 특정 초성, 특정 초성과 중성, 또는 특정 중성을 가진 한글 음절들의 탐색을 SQL의 기존 연산자들, 정규 표현식, 또는 확장된 문자열 양식을 이용하여 표현하는 것은 실질적으로 많은 불편을 수반하며 DBMS의 문자 집합에 따라 SQL 응용 프로그램의 호환성 문제가 초래될 수 있다. 본 논문이 초성과 중성 기반 한글 음절 탐색 양식을 제안하는 목적은 한글 문자열 검색을 위한 직관적이고 일관되며 간단한 탐색 양식을 제공하고 그 탐색 양식을 통해 문자 집합에 따른 SQL 응용 프로그램의 호환성 문제를 극복하기 위함이다.

본 논문의 나머지 부분의 구성은 다음과 같다. 제2장에서 초성과 중성 기반 한글 음절 탐색 양식의 정의와 그 기능을 제시한다. 제3장에서 초성과 중성 기반 한글 음절 탐색 양식의 각 유형에 부합하는 문자들의 식별 방법을 제시한 후, 제4장에서 연산자 LIKE의 문자열 양식을 정규화하는 방법과 문자열 탐색을 위한 관련 자료구조를 제시하고 초성과 중성 기반 한글 음절 탐색 양식을 고려한 LIKE 연산의 수행 알고리즘을 제시한다. 색인에서 초성과 중성 기반 한글 음절 탐색 양식에 부합하는 문자열들을 식별하는 방법을 제5장에 제시하며 제6장에서 본 논문의 결론을 맺는다.

2. 초성과 중성 기반 한글 음절 탐색 양식의 정의와 그 기능

초성과 중성 기반 한글 음절 탐색 양식은 전치자(前置者, predecessor)와 탐색자(探索者, searcher)로 구성된다. 그 양식의 전치자는 연산자 LIKE의 탈출문자이거나 새롭게 정의된 예약된 문자일 수 있다. 예를 들어, 문자 '$'를 참고문헌 [10]과 같은 방법으로 새로운 예약된 문자로 정의하여 사용할 수 있다. 그 양식의 탐색자는 초성자음, 초성자음과 모음으로만 구성된 음절, 또는 모음일 수 있다. 그 양식의 탐색자가 초성자음인 경우에는 그 초성자음을 초성으로 가지는 한글 음절들이, 초성자음과 모음으로만 구성된 음절인 경우에는 그 초성자음과 모음을 초성과 중성으로 가지는 한글 음절들이, 그리고 모음인 경우에는 그 모음을 중성으로 가지는 한글 음절들이 그 양식에 부합한다. 본 논문의 나머지 부분에서 탈출문자가 그 양식의 전치자로 사용되며 별도의 선언을 하지 않는 한 '\'가 탈출문자로 선언된 것으로 가정한다.

초성과 중성 기반 한글 음절 탐색 양식이 필요한 실생활에서의 예로는 경찰서 신원 조회, 전화번호부 검색 등을 위해 어떤 사람을 찾고자 하는 경우를 들 수 있다. 찾고자 하는 사람이 '강정석'인지, '감정석'인지, '강전석'인지, '감전숙'인지 자세히 기억나지 않을 수 있다. 그 경우, 연산자 LIKE의 기존 문자열 양식을 이용한다면 "name LIKE '강정석' or name LIKE '감정석' or name like '강전석' or name LIKE '감전숙' or ……"과 같이 모든 경우를 고려하여 질의문을 작성하거나 "name > '감' and name < '강'"으로 질의문을 작성하고 그 결과 중 자신이 원하는 것을 일일이 찾아야 한다. 그러나, 초성과 중성 기반의 한글 음절 탐색 양식을 이용하면 "name like '\가\저\ㅅ' ESCAPE '\'"로 간단히 명시하여 원하는 결과를 얻을 수 있다. 초성과 중성 기반 한글 음절 탐색 양식을 이용함으로써 제 1장의 예제 1, 2, 3의 요구는 각각 "SELECT * FROM employees WHERE address LIKE '\ㅂ%' ESCAPE '\';", "SELECT * FROM employees WHERE name LIKE '\ㅂ\여\ㅓ' ESCAPE '\';", "SELECT * FROM employees WHERE name LIKE '%_\ㅂ\여\ㅓ_%' ESCAPE '\';"로 작성될 수 있다.

초성자음들로 사용되지 않는 종성자음들과 초성, 중성, 그리고 종성으로 구성된 한글 음절들은 초성과 중성 기반 한글 음절 탐색 양식의 탐색자로 사용되지 않는다. 이는 특정 종성을 가진 한글 음절들을 탐색하고자 하는 경우가 매우 드물 것으로 예측되는 것이 하나의 이유이며 다른 이유는 초성자음들과 종성자음들에 공통으로 사용되는 자음이 탈출문자 바로 다음에 명시된다면 KS X 5002 표준 자판을 사용하는 경우에는 그 자음이 초성자음인지 또는 종성자음인지의 구분이 불가능하기 때문이다. 이러한 이유로, 종성자음으로만 사용되는 자음이 탈출문자 바로 다음에 명시되면 그 종성자음을 초성과 중성 기반 한글 음절 탐색 양식의 탐색자가 아닌 그 자음 자체가 명시된 것과 동일하게 처리한다. 초성, 중성, 그리고 종성으로 구성된 한글 음절이 탈출문자 바로 다음에 명시된다면 이 또한 초성과 중성 기반 한글 음절 탐색 양식의 탐색자가 아닌 그 음절 자체가 명시된 것과 동일하게 처리한다. 그 이유는 그를 만족하는 한글 음절은 그 자신뿐이기 때문이다.

정의 1. 특정 초성자음이 초성자음들에서 차지하는 순서를 그 초성자음의 초성자음 색인이라 하며, 특정 모

3) 참고로, 정규 표현식의 문자가 사용하는 문자 집합의 문자 범위를 벗어난다면 ORACLE 10g는 그 정규 표현식을 오류로 처리한다.

음이 모음들에서 차지하는 순서를 그 모음의 모음 색인이라 한다. 이들 색인들은 0부터 시작한다.

예를 들어, 자음 'ㄱ'의 초성자음 색인은 0이며, 음절 '날'의 초성 'ㄴ'의 초성자음 색인은 1이고, 중성 'ㅏ'의 모음 색인은 0이며, 음절 '책'의 중성 'ㅐ'의 모음 색인은 1이다.

본 논문은 초성과 중성 기반 한글 음절 탐색 양식의 탐색 특징을 제시하기 위해 한글 음절 테이블(Korean Syllable Table, KST)을 고안하였다. 한글 음절 테이블은 초성자음들의 수인 19개의 행(行, row)들과 모음들의 수인 21개의 열(列, column)들로 구성된다. 한글 음절 테이블의 각 행은 특정 초성자음을 초성으로 가지는 한글 음절들을, 각 열은 특정 모음을 중성으로 가지는 한글 음절들을, 특정 행과 특정 열에 의해 형성되는 특정 셀(cell)은 그 행의 초성자음과 그 열의 모음을 각각 초성과 중성으로 가지는 한글 음절들을 가지며 각 셀 내부의 음절들은 그들의 사전적 순서에 따라 정렬된다. KSX 1001 부호계의 한글 완성형에 정의된 2,350자를 대상으로 구성한 한글 음절 테이블은 그림 1과 같다.

한글 음절 테이블에서 i($0 \le i \le 18$)번째 행을 KST_ROW_i로 나타내고 i를 그 행의 행 색인(row index)이라 하며 j($0 \le j \le 20$)번째 열을 KST_COLUMN_j로 나타내고 j를 그 열의 열 색인(column index)이라 한다. 한글 음절 테이블의 KST_ROW_i와 KST_COLUMN_j에 의해 형성되는 셀을 $KST_{i,j}$로 나타낸다. 한글 음절 테이블의 모든 행들과 열들, 그리고 각 셀의 음절들은 사전적 순서를 따라 정렬되어 있다. 따라서, 특정 초성자음의 초성자음 색인은 한글 음절 테이블에서 그 초성자음의 행 색인과 동일하며 특정 모음의 모음 색인은 한글 음절 테이블에서 그 모음의 열 색인과 동일하다. $KST_{i,j}$의 첫 번째 음절과 마지막 음절을 각각 $FS_{i,j}$와 $LS_{i,j}$라 하자. 그러면, KST_{ij}, KST_ROW_i, 그리고 KST_COLUMN_j의 음절들은 각각 [$FS_{i,j}$-$LS_{i,j}$], [$FS_{i,0}$-$LS_{i,20}$], 그리고 [$FS_{0,j}$-$LS_{0,j}$ | $FS_{1,j}$-$LS_{1,j}$ | … | $FS_{18,j}$-$LS_{18,j}$]의 범위에 존재한다. 예를 들어, $KST_{7,4}$, KST_ROW_7, 그리고 KST_COLUMN_4의 음절들은 각각 [$FS_{7,4}$-$LS_{7,4}$] (즉, [버-벚]), [$FS_{7,0}$-$LS_{7,20}$] (즉, [바-빛]), 그리고 [$FS_{0,4}$-$LS_{0,4}$ | $FS_{1,4}$-$LS_{1,4}$ | … | $FS_{18,4}$-$LS_{18,4}$] (즉, [거-겋 | 꺼-껑 | … | 허-헝])의 범위에 존재한다.

한글 음절 테이블을 기반으로 하면 특정 초성, 특정 초성과 중성, 또는 특정 중성을 가진 한글 음절들을 찾는 문제는 각각 한글 음절 테이블의 특정 행, 특정 셀, 또는 특정 열을 찾는 문제가 된다. 따라서, 본 논문은 특정 초성, 특정 초성과 중성, 또는 특정 중성을 가진 한글 음절들을 찾기 위한 초성과 중성 기반 한글 음절 탐색 양식을 각각 초성 유형(Type_ROW), 초성과 중성 유형(Type_CELL), 또는 중성 유형(Type_COLUMN)의 탐색 양식이라 한다.

3. 초성과 중성 기반 한글 음절 탐색 양식과 한글 음절의 비교

초성과 중성 기반 한글 음절 탐색 양식은 탐색자가 가지는 값에 따라 초성 유형, 중성 유형, 그리고 초성과 중성 유형의 세 가지로 구분된다. 본 장은 초성과 중성 기반 한글 음절 탐색 양식의 유형들을 식별하는 방법과 각 유형별 탐색 양식에 부합하는 한글 음절들을 판별하는 방법을 KS X 1001 부호계의 한글 완성형을 기반으로 제시한다.

3.1 KS X 1001 부호계의 한글 완성형에서 한글의 특징과 한글 첫음절 테이블

본 절은 KS X 1001 부호계의 한글 완성형에서 한글의 자음들, 모음들, 그리고 음절들의 특징을 간단히 기술한 후 그에따른 한글 음절들의 식별을 위하여 한글 첫음절 테이블(Korean first-syllable table, KFST)을 제시한다. KS X 1001 부호계의 한글 완성형에는 한글의 자음들 30자, 모음들 21자, 그리고 음절들 2,350자가

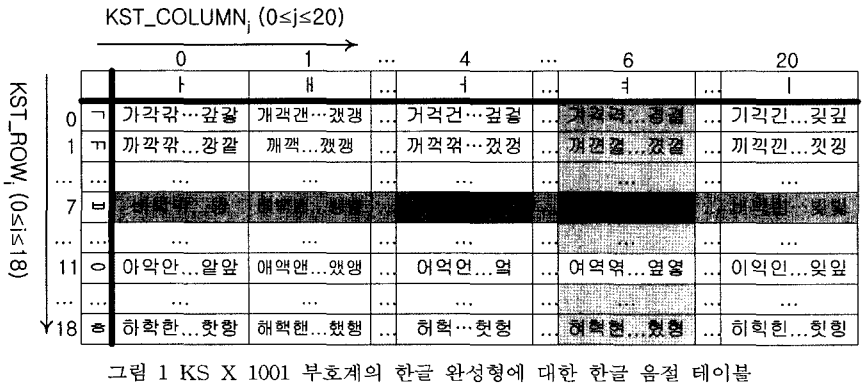

그림 1 KS X 1001 부호계의 한글 완성형에 대한 한글 음절 테이블

표 1 KS X 1001 부호계의 한글 완성형에 정의된 자음들, 모음들, 그리고 음절들

자음들 30자 (0xA4A1~0xA4BE)	ㄱ, ㄲ, ㄳ, ㄴ, ㄵ, ㄶ, ㄷ, ㄸ, ㄹ, ㄺ, ㄻ, ㄼ, ㄽ, ㄾ, ㄿ, ㅀ, ㅁ, ㅂ, ㅃ, ㅄ, ㅅ, ㅆ, ㅇ, ㅈ, ㅉ, ㅊ, ㅋ, ㅌ, ㅍ, ㅎ
모음들 21자 (0xA4BF~0xA4D3)	ㅏ, ㅐ, ㅑ, ㅒ, ㅓ, ㅔ, ㅕ, ㅖ, ㅗ, ㅘ, ㅙ, ㅚ, ㅛ, ㅜ, ㅝ, ㅞ, ㅟ, ㅠ, ㅡ, ㅢ, ㅣ
음절들 2,350자 (0xB0A1~0xC8FE)	가, 각, 간, …, 사, 삭, 삯, …, 차, 착, 찬, …, 타, 탁, 탄, …, 힙, 힛, 힝

사전적 순서를 따라 정렬되어 있다. 표 1은 KS X 1001 부호계의 한글 완성형에 정의된 한글의 자음들, 모음들, 음절들의 수와 순서, 그리고 부호값을 나타낸다.

한글의 자음들의 부호값은 이들 중 순서가 가장 앞인 'ㄱ'을 '0xA4A1'로 나타내며 순서에 따라 차례로 1씩 증가하는 연속된 값으로 나타내어 순서가 가장 뒤인 'ㅎ'을 '0xA4BE'로 나타낸다. 한글의 모음들의 부호값은 이들 중 순서가 가장 앞인 'ㅏ'를 '0xA4BF'로 나타내며 순서에 따라 차례로 1씩 증가하는 연속된 값으로 나타내어 순서가 가장 뒤인 'ㅣ'를 '0xA4D3'로 나타낸다. 한글의 음절들 중 순서가 가장 앞인 '가'는 '0xB0A1'의 값을 가지며 순서에 따라 차례로 1씩 증가하는 연속된 값으로 나타내어 순서가 가장 뒤인 '힝'은 '0xC8FE'의 값을 가진다. 한글 음절들에 할당된 0xB0A0에서 0xC8FF까지의 영역에서 끝이 A0와 FF인 값들은 어떤 음절에도 할당되지 않는다. 표 1에 사용된 상수 값들에 대한 가독성을 높이기 위하여 이들 상수 값들을 나타내는 상수들을 정리하여 표 2에 나타내었다.

본 논문이 제시하는 초성과 중성 기반 한글 음절 탐색 양식은 KS X 1001 부호계의 한글 완성형을 채택하고 있는 바다-II[11-13]에 구현되었으며 그 구현을 위해 바다-II의 구문 분석(parsing), 최적화(optimization), 실행(execution)의 세 부분이 수정되었다. 본 장과 제 4장에서 제시하는 알고리즘들은 실행 부분의 구현 내용이며 제 5장에서 제시하는 색인을 이용한 수행 알고리즘은 최적화 부분과 실행 부분의 구현 내용이다.

제1장에 제시한 바와 같이, KS X 1001 부호계의 한글 완성형의 음절들에 대한 한글 음절 테이블은 다음의

표 2 상수들의 정의

상수명	값	설명
CONSONANT_START	0xA4A1	'ㄱ'의 부호값
CONSONANT_END	0xA4BE	'ㅎ'의 부호값
VOWEL_START	0xA4BF	'ㅏ'의 부호값
VOWEL_END	0xA4D3	'ㅣ'의 부호값
HANGUL_START	0xB0A1	'가'의 부호값
HANGUL_END	0xC8FE	'힝'의 부호값
INITIAL_CON_SIZE	19	초성자음들의 수
VOWEL_SIZE	21	모음들의 수
CONSONANT_SIZE	30	자음들의 수

특징을 가진다. 첫째, 46개의 셀들이 하나의 음절도 가지지 않는다. 둘째, 5개의 셀들이 그 조합의 첫 번째 음절을 초성, 중성, 그리고 종성으로 구성된 음절로 가진다. 셋째, 대부분의 조합들이 그 조합의 마지막 음절의 종성자음으로 일정한 규칙을 가지지 않는다. 본 논문은 이러한 KS X 1001 부호계의 한글 완성형의 음절들에 대한 한글 음절 테이블의 특징으로 인하여 탐색자로 명시된 특정 음절이 초성과 중성으로만 구성된 음절인지 여부를 판단하고 초성과 중성 기반 한글 음절 탐색 양식들에 부합되는 음절들의 범위를 획득하기 위하여 한글 첫음절 테이블(Korean First Syllable Table, KFST)을 둔다.

한글 첫음절 테이블은 한글 음절 테이블과 같이 초성 자음들의 수만큼의 행들과 모음들의 수만큼의 열들로 구성되어 KFST_SIZE(INITIAL_CON_SIZE * VOWEL_SIZE 즉, 399) 개의 엔트리들을 가지며 각 엔트리는 하나의 한글 음절의 부호값을 가진다. 한글 첫음절 테이블에서 $i(0 \le i \le 18)$번째 행을 $KFST_ROW_i$로 나타내고 i를 그 행의 행 색인이라 하며 $j(0 \le j \le 20)$번째 열을 $KFST_COLUMN_j$로 나타내고 j를 그 열의 열 색인이라 한다. 한글 첫음절 테이블의 $KFST_ROW_i$와 $KFST_COLUMN_j$에 의해 형성되는 셀을 $KFST_{i,j}$로 나타내며 KFST[i][j]를 의미한다. 한글 첫음절 테이블의 $KFST_{i,j}$는 한글 음절 테이블에서 $KST_{i,j}$를 포함하여 그 셀 이후의 모든 셀들에서 최소의 음절의 부호값을 값으로 가진다. 즉, $KFST_{i,j}$는 $KST_{i,j}$에 음절이 존재하는 경우에는 그 셀의 첫 번째 음절의 부호값을 값으로 가지지만 그렇지 않은 경우에는 그 셀 이후의 모든 셀들에서 최소의 음절의 부호값을 값으로 가진다. 2차원 배열 KFST를 일차원으로 치환한 배열을 KFST_1D라 한다. 즉, $KFST_{i,j}$의 엔트리는 $KFST\_1D_{i*VOWEL\_SIZE+j}$ 즉, KFST_1D[i*VOWEL_SIZE+j]의 엔트리와 동일하다. 배열 KFST_1D의 구성은 그림 2와 같다.

예를 들어, $KFST_{7,8}$은 초성자음 색인이 7인 'ㅂ'과 모음 색인이 8인 'ㅗ'의 엔트리로서 $KST_{7,8}$의 첫 번째 음절인 '보'의 부호값을 값으로 가지며, $KFST_{13,12}$는 초성자음 색인이 13인 'ㅉ'과 모음 색인이 12인 'ㅛ'의 엔트리로서 $KST_{13,12}$의 첫 번째 음절인 '쭁'의 부호값을 값으로 가진다. 반면, $KFST_{2,3}$은 초성자음 색인이 2인 'ㄴ'과

```
static const unsigned short KFST_1D[] =
{
  '가','개','갸','걔','거','게','겨','계','고','과','괘','괴','교','구','궈','궤','귀','규','그','긔','기',
  '까','깨','꺄','꺠','꺼','께','껴','꼐','꼬','꽈','꽤','꾀','꾜','꾸','꿔','꿰','뀌','뀨','끄','끠','끼',
  '나','내','냐','너','너','네','녀','녜','노','놔','뇌','뇌','뇨','누','눠','눼','뉘','뉴','느','늬','니',
  '다','대','댜','더','더','데','뎌','뎨','도','돠','돼','되','됴','두','둬','뒈','뒤','듀','드','듸','디',
  '따','때','뗘','떠','떠','떼','뗘','뗴','또','똬','뙈','뙤','뚀','뚜','뚸','뛔','뛰','뜌','뜨','띄','띠',
  '라','래','랴','러','러','레','려','례','로','롸','뢨','뢰','료','루','뤄','뤠','뤼','류','르','릐','리',
  '마','매','먀','머','머','메','며','몌','모','뫄','뫼','뫼','묘','무','뭐','뭬','뮈','뮤','므','미','미',
  '바','배','뱌','버','벼','베','벼','볘','보','봐','봬','뵈','뵤','부','붜','붸','뷔','뷰','브','븨','비',
  '빠','빼','뺘','뻐','뼈','뻬','뼈','뽀','뽀','뾔','뾰','뾰','뾰','뿌','뿨','뿨','쀠','쀼','쁘','쁴','삐',
  '사','새','샤','섀','서','세','셔','셰','소','솨','쇄','쇠','쇼','수','숴','쉐','쉬','슈','스','싀','시',
  '싸','쌔','쌰','썌','써','쎄','쎤','쎈','쏘','쏴','쐐','쐬','쑈','쑤','쒀','쒜','쒸','쓔','쓰','씌','씨',
  '아','애','야','얘','어','에','여','예','오','와','왜','외','요','우','워','웨','위','유','으','의','이',
  '자','재','쟈','쟤','저','제','져','졔','조','좌','좨','죄','죠','주','줘','줴','쥐','쥬','즈','지','지',
  '짜','째','쨔','쨔','쩌','쩨','쪄','쪠','쪼','쫘','쫴','쬐','쬬','쭈','쭤','쭸','쮀','쮸','쯔','찌','찌',
  '차','채','챠','챼','처','체','쳐','쳬','초','촤','쵀','최','쵸','추','춰','췌','취','츄','츠','치','치',
  '카','캐','캬','컈','커','케','켜','켸','코','콰','쾌','쾨','쿄','쿠','쿼','퀘','퀴','큐','크','키','키',
  '타','태','탸','턔','터','테','텨','톄','토','톼','퇘','퇴','툐','투','퉈','퉤','튀','튜','트','틔','티',
  '파','패','퍄','퍠','퍼','페','펴','폐','포','퐈','퐤','푀','표','푸','풔','풰','퓌','퓨','프','픠','피',
  '하','해','햐','햬','허','헤','혀','혜','호','화','홰','회','효','후','훠','훼','휘','휴','흐','희','히',
}
```

그림 2 KSX 1001 부호계의 한글 완성형에 대한 한글 첫음절 테이블

모음 색인이 3인 'ㅐ'의 엔트리이지만 $KST_{2,3}$이 비어있고 $KST_{2,4}$가 음절을 가지므로 $KST_{2,4}$의 첫 번째 음절인 '너'의 부호값을 값으로 가지며, $KFST_{4,2}$는 초성자음 색인이 4인 'ㄸ'과 모음 색인이 2인 'ㅑ'의 엔트리이지만 $KST_{4,2}$와 $KST_{4,3}$이 비어있고 $KST_{4,4}$가 음절을 가지므로 $KST_{4,4}$의 첫 번째 음절인 '떠'의 부호값을 값으로 가진다.

고찰 1. KS X 1001 부호계의 한글 완성형에 정의된 초성과 중성으로만 구성된 모든 음절들은 한글 첫음절 테이블에 수록된다.

고찰 2. 한글 첫음절 테이블의 전체 399개의 엔트리들 중에서 여섯 개의 엔트리들만이 초성, 중성, 그리고 종성으로 구성된 음절들의 부호값을 가지며 나머지 모든 엔트리들은 초성과 중성으로만 구성된 음절들의 부호값을 가진다. 한글 첫음절 테이블의 엔트리들 중에서 초성, 중성, 그리고 종성으로 구성된 음절들은 '뢨', '쌍', '쎈', '쓩', '쭁'이며 '쎈'은 두 번 등장한다.

고찰 3. KS X 1001 부호계의 한글 완성형에 정의된 임의의 한글 음절 S는 그 음절의 부호값을 W라 할 때 다음의 특징을 가지는 엔트리 $KFST_1D_m$을 오직 하나 가진다.

$$KFST\_1D_m <= W < KFST\_1D_{m+1}$$
$$\text{if } m < KFST\_SIZE - 1,$$
$$KFST\_1D_m <= W < HANGUL\_END$$
$$\text{otherwise.}$$

본 논문은 KS X 1001 부호계의 한글 완성형에 정의된 임의의 한글 음절 S에 대하여 고찰 3을 만족하는 $KFST_1D_m$의 색인 m을 음절 S의 한글 첫음절 테이블 색인(KFST index)이라 정의한다. 즉, 음절 S의 한글 첫음절 테이블 색인은 배열 KFST_1D에서 S의 부호값보다 작거나 같은 값들을 가지는 엔트리들 중에서 최대 값을 가진 엔트리(들)의 최대 색인을 의미한다. 예를 들어, 한글 음절 '넙'의 부호값은 음절 '너'의 부호값인 $KFST_1D_{46}$과 음절 '네'의 부호값인 $KFST\_1D_{47}$ 사이에 존재하므로 음절 '넙'의 한글 첫음절 테이블 색인은 46이다.

음절 S의 부호값을 W라 하고 음절 S의 한글 첫음절 테이블 색인을 구하는 알고리즘을 KFST_index(W)라 할 때, 알고리즘 KFST_index(W)는 배열 KFST_1D에 대한 이진탐색을 기반으로 형성될 수 있으며 $O(\log_2 KFST_SIZE)$의 복잡성을 가진다. 음절 S의 초성의 초성자음 색인은 KFST_index(W)/VOWEL_SIZE이며 중성의 모음 색인은 KFST_index(W)%VOWEL_SIZE이다.

3.2 초성자음의 식별과 한글 음절과의 비교

KS X 1001 부호계의 한글 완성형에서 한글의 자음들은 사전적 순서에 따라 CONSONANT_START 부터 CONSONANT_END 사이의 연속된 부호값이 할당되어 있으며 초성자음들은 별도로 배치되어 있지 않다. 따라서, 본 논문은 한글의 자음들 중에서 각 초성자음의 초성자음 색인을 식별하기 위해 배열 IC_Index를 둔다. 배열 IC_Index는 CONSONANT_SIZE 개의 엔트리들을 가지며 그 배열의 i번째 엔트리는 i번째 자음이 초성자음인 경우에는 그 초성자음의 초성자음 색인을 값으로 가지며 초성자음이 아닌 경우에는 -1을 값으로 가진다. 그림 3은 배열 IC_Index를 나타낸다.

```
static const int IC_Index[] =
{
   0, 1, -1, 2, -1, -1, 3, 4, 5, -1, /* 'ㄱ','ㄲ','ㄳ','ㄴ','ㄵ','ㄶ','ㄷ','ㄸ','ㄹ','ㄺ' */
   -1, -1, -1, -1, -1, -1, 6, 7, 8, -1, /* 'ㄻ','ㄼ','ㄽ','ㄾ','ㄿ','ㅀ','ㅁ','ㅂ','ㅃ','ㅄ' */
   9, 10, 11, 12, 13, 14, 15, 16, 17, 18 /* 'ㅅ','ㅆ','ㅇ','ㅈ','ㅉ','ㅊ','ㅋ','ㅌ','ㅍ','ㅎ' */
}
```

그림 3 배열 IC_Index

예를 들어, IC_Index[0]은 0 번째 자음인 'ㄱ'의 초성 자음 색인이 0임을 나타내며, IC_Index[10]은 -1을 그 값으로 가져 10 번째 자음인 'ㄻ'이 초성자음이 아님을 나타내고, IC_Index[21]은 21 번째 자음인 'ㅆ'의 초성자음 색인이 10임을 나타낸다.

연산자 LIKE의 문자열 양식에서 2-바이트로 구성된 한글 한 문자 x가 한글의 자음들의 범위인 CONSONANT_START와 CONSONANT_END 사이의 값을 가지며 IC_Index[x - CONSONANT_START]의 값을 i라 할 때, i가 -1이 아닌 경우에 한하여 그 문자 x는 초성자음이며 i는 x의 초성자음 색인을 나타낸다. 초성자음 색인이 i인 초성자음을 초성으로 가지는 음절들의 부호값의 범위를 R이라 하고, LB ≤ R < UB라 할 때, LB와 UB는 한글 첫음절 테이블을 이용하여 각각 다음과 같이 정의된다.

$$LB = KSFT\_1D_{i*VOWEL\_SIZE},$$
$$UB = KSFT\_1D_{(i+1)*VOWEL\_SIZE}$$
$$\text{if } i < INITIAL\_CON\_SIZE - 1,$$
$$HANGUL\_END + 1$$
$$\text{otherwise.}$$

예를 들어, 초성자음 'ㅁ'을 초성으로 가지는 음절들의 부호값은, 초성자음 'ㅁ'의 초성자음 색인이 IC_Index['ㅁ'의 부호값 - CONSONANT_START] 즉, 7이므로, $KSFT_{7,0}$의 값인 '마'의 부호값 보다 크거나 같고, $KSFT_{8,0}$의 값인 '바'의 부호값 보다 작다. 따라서, 초성과 중성 기반 한글 음절 탐색 양식의 탐색자가 초성자음인 경우, 특정 한글 음절이 그 탐색 양식에 부합하는지를 비교하기 위해서는 그 초성자음에 대한 LB와 UB를 구하여 그 음절이 그 범위에 속하는 지를 비교하면 된다.

3.3 초성과 중성으로만 구성된 음절의 식별과 한글 음절과의 비교

연산자 LIKE의 문자열 양식에서 2-바이트로 구성된 한글 한 문자가 HANGUL_START와 HANGUL_END 사이의 값을 가질 경우, 그 문자는 한글 한 음절을 나타낸다. 제2장에서 언급한 바와 같이, 초성과 중성 기반 한글 음절 탐색 양식의 탐색자가 한글 음절인 경우에는 초성과 중성으로만 구성된 음절만을 탐색자로 수용한다. 따라서, 임의의 음절 S가 초성과 중성으로만 구성된 음절임을 식별하는 방법이 필요하다.

초성과 중성으로만 구성된 음절의 식별 방법. 한글 첫음절 테이블에 음절 S의 부호값이 존재하는 지를 검사한다. 그 값이 한글 첫음절 테이블에 존재하지 않으면 한글 첫음절 테이블에 대한 고찰 1에 의해 음절 S는 초성, 중성, 그리고 종성으로 구성된 음절이다. 그 값이 한글 첫음절 테이블에 존재하는 경우에는 한글 첫음절 테이블에 대한 고찰 2에 의해, 그 음절이 '뢨', '쌓', '쏀', '쑴', '쭁'의 다섯 개의 음절들 중의 하나이면 그 음절은 초성, 중성, 그리고 종성으로 구성된 음절이며 그렇지 않으면 그 음절은 초성과 중성으로만 구성된 음절이다.

한글 첫음절 테이블을 탐색하여 임의의 음절이 초성과 중성으로만 구성된 음절임을 확인한 경우, 그 음절의 한글 첫음절 테이블 색인을 m이라 할 때, 그 음절의 초성과 중성을 초성과 중성으로 가지는 한글 음절들의 부호값의 범위를 R이라 하고, LB ≤ R < UB라 하면, LB와 UB는 한글 첫음절 테이블을 이용하여 각각 다음과 같이 정의된다.

$$LB = KSFT\_1D_m$$
$$UB = KSFT\_1D_{m+1}$$
$$\text{if } m < KSFT\_SIZE - 1,$$
$$HANGUL\_END + 1$$
$$\text{otherwise.}$$

예를 들어, 초성과 중성 기반 한글 음절 탐색 양식의 탐색자가 '조'인 경우, 한글 첫음절 테이블을 탐색함으로써 '조'가 초성과 중성으로 구성된 한글 음절임을 확인하고 동시에 '조'의 한글 첫음절 테이블 색인이 260임을 알아냄으로써 탐색자 '조'의 LB는 $KSFT_1D_{260}$의 값인 '조'의 부호값이 되며 UB는 $KSFT_1D_{261}$의 값인 '좌'의 부호값이 된다. 따라서, 초성과 중성 기반 한글 음절 탐색 양식의 탐색자가 한글 음절인 경우, 그 탐색자의 음절들의 범위를 나타내는 LB와 UB를 구하여 비교할 음절이 그 범위에 속하는 지를 비교하면 된다.

3.4 중성모음의 식별과 한글 음절과의 비교

연산자 LIKE의 문자열 양식에서 2-바이트로 구성된 한글 한 문자 x가 한글의 모음들의 범위인 VOWEL_START와 VOWEL_END 사이의 값인 경우, 그 문자는 한글 모음이며 x - VOWEL_START는 그 문자의 모음 색인을 나타낸다. 초성과 중성 기반 한글 음절 탐

색 양식의 탐색자가 모음인 경우, 그 모음의 모음 색인을 j라 할 때, 임의의 음절 S가 그 탐색 양식에 부합하는 지를 식별하는 방법들은 음절 S의 부호값을 W라 할 때 다음과 같이 두 가지가 있을 수 있다.

첫 번째 방법은 음절 S의 중성의 모음 색인인 KSFT_index(W)%VOWEL_SIZE가 탐색자의 모음 색인 j와 같은 지를 검사하는 것이다. 예를 들어, 탐색자가 'ㅠ'이고 비교할 음절이 '쀼'이면 음절 '쀼'의 한글 첫음절 테이블 색인을 구한다. 즉, 한글 첫음절 테이블에서 '쀼' 보다 작거나 같은 값들 중에서 최대값을 가진 엔트리들인 182, 183, 184, 185번째 엔트리들 중에서 최대 색인의 값인 185를 구한다. 그리고, 그 최대 색인의 값인 185에 대해 VOWEL_SIZE로 나눈 나머지 값 즉, 185%VOWEL_SIZE의 값인 17이 '쀼'의 중성의 모음 색인이며 이는 그 탐색자 'ㅠ'의 모음 색인과 동일함을 알 수 있다.

두 번째 방법은 비교할 음절 S의 중성의 모음 색인을 탐색자의 모음 색인 j와 동일하다고 추측(guess)하고 그 추측이 맞는지 확인(verify)하는 방법이다. 다음과 같이 수행된다.

(1) 한글 첫음절 테이블의 j번째 칼럼의 엔트리들인 $KFST\_COLUMN_j$ 즉, $KSFT_{0,j}$, $KSFT_{1,j}$, …, $KSFT_{INITIAL_CON_SIZE-1,j}$에 대하여, 이들 중 S의 부호값인 W 보다 작거나 같은 값들 중에서 최대값을 가진 엔트리를 이진탐색으로 구한다. 그 엔트리의 색인을 m이라 하자.

(2) 엔트리 m이 한글 첫음절 테이블의 마지막 엔트리이거나, 음절 S의 부호값이 $KSFT\_1D_{m+1}$ 보다 작다면 음절 S의 모음 색인은 j이며 그렇지 않다면 음절 S의 모음 색인은 j가 아니다.

예를 들어, 비교할 음절이 '쀼'이고 탐색자가 'ㅠ'인 경우, 비교할 음절 '쀼'의 중성의 모음 색인을 그 탐색자의 모음 색인인 17로 추측하고 다음과 같이 확인한다. $KFST_COLUMN_{17}$에 대한 이진 탐색을 통해 '쀼'의 부호값 보다 작거나 같은 값들 중에서 최대값을 가진 엔트리는 $KSFT_{8,17}$임을 알 수 있으며, 그 엔트리의 색인은 185이고 '쀼'의 부호값 < $KSFT_1D_{186}$이므로 음절 '쀼'은 모음 'ㅠ'와 동일한 모음을 가짐을 알 수 있다.

위의 두 방법 모두 정확한 결과를 얻는다. 첫 번째 방법의 복잡성은 $O(\log_2 KFST_SIZE)$인 반면, 두 번째 방법의 복잡성은 $O(\log_2 INITIAL_CON_SIZE)$이다. 즉, 첫 번째 방법의 복잡도는 $O(\log_2 399)$이며 두 번째 방법의 복잡도는 $O(\log_2 19)$이다. 본 논문은 성능을 위하여 두 번째 방법을 채택하며 그 함수를 verify_vowel_index (vowel_index, S)로 설정한다. 함수 verify_vowel_index (vowel_index, S)는 특정 모음 색인 vowel_index와 특정 한글 문자 S에 대하여 (1) 그 문자 S가 한글 음절이며 (2) S의 모음 색인이 주어진 모음 색인 vowel_index와 동일한 지를 검사하는 함수로서 이들 두 조건을 모두 만족할 경우에 한하여 TRUE를 반환한다.

초성과 중성 기반 한글 음절 탐색 양식의 탐색자가 모음인 경우, 위에 제시한 방법에 추가하여 색인 탐색(index scan)에서 사용될 탐색 범위의 경계값을 구하기 위해 다음의 방법을 추가로 사용한다. 모음 색인이 j인 모음을 중성으로 가지는 음절들이 가질 수 있는 범위를 R이라 하고, $LB \le R < UB$라 할 경우, LB와 UB는 각각 다음과 같다. 아래의 표현에서 m은 ((INITIAL_CON_SIZE - 1) * VOWEL_SIZE) + j 이다. 이는 한글 첫음절 테이블에서 마지막 초성자음인 'ㅎ'이 모음 색인이 j인 모음과 이루는 엔트리의 색인을 나타낸다.

$$LB = KSFT\_1D_j$$

$$UB = KSFT\_1D_{m+1}$$

$$\text{if } j < VOWEL\_SIZE-1,$$

$$HANGUL\_END + 1$$

$$\text{otherwise.}$$

이 방법은 초성자음 색인이 0인 초성자음 'ㄱ'과 모음 색인이 j인 모음을 각각 초성과 중성으로 가지는 음절의 부호값보다 크거나 같으며 초성자음 색인이 INITIAL_CON_SIZE - 1인 'ㅎ'과 모음 색인이 j+1인 모음을 각각 초성과 중성으로 가지는 음절의 부호값 보다 작은 값들을 포함하게 되어 필요 이상으로 넓은 범위를 포함하게 된다. 그러나, 이 범위는 색인 탐색의 탐색 범위를 제한하는데 유용하게 사용될 수 있으며 오직 그 목적으로만 사용된다.

4. 초성과 중성 기반 한글 음절 탐색 양식을 고려한 부합 알고리즘

연산자 LIKE의 문자열 양식에 부합하는 문자열들을 식별하기 전에 그 문자열 양식은 정규화되어야 한다. 본 장은 초성과 중성 기반 한글 음절 탐색 양식을 포함하는 문자열 양식의 정규화 알고리즘을 제시한 후, 그 문자열 양식에 부합하는 문자열의 탐색 알고리즘을 제시한다. 본 논문은 연산자 LIKE의 문자열 양식으로 일반 문자들, 예약된 문자들인 '%'와 '_', 그리고 탈출 문자들만을 고려한다.[4)]

4.1 문자열 양식의 정규화

연산자 LIKE의 문자열 양식은 배열 StringPattern에,

4) MS SQL Server 2005와 같은 상용 DBMS가 지원하는 허용 문자열 양식인 '[patterns]'와 제한 문자열 양식인 '[^patterns]'는 고려하지 않는다. 예를 들어, 양식 '[a-f]'의 경우, 'a'부터 'f'까지의 문자들 중 어떠한 문자라도 그 양식에 부합하며, 양식 '[^a-f]'의 경우, 'a'부터 'f'까지의 문자들을 제외한 어떠한 문자라도 그 양식에 부합한다.

정규화한 문자열 양식은 배열 zPattern에 보관된다고 하자. 먼저, 초성과 중성 기반 한글 음절 탐색 양식을 고려하지 않은 문자열 양식에 대한 정규화 규칙은 다음과 같다.

(1) 연속된 예약된 문자 '%'는 하나의 예약된 문자 '%'로 유지한다.
(2) 예약된 문자 '%' 직후의 예약된 문자 '_'는 이들의 자리를 서로 바꾼다. 즉, "…%_…"를 "…_%…"로 유지한다.
(3) 예약된 문자 '%'와 '_'를 각각 일반 문자로 사용될 수 없는 부호값으로 표현한다. 본 논문은 예약된 문자 '%'를 0x16 즉, SYN으로, 예약된 문자 '_'를 0x1A 즉, SUB로 표현한다.
(4) 탈출문자 직후의 '%', '_', 또는 탈출문자는 탈출문자와 그들의 쌍을 그 문자와 동일 값을 가지는 하나의 일반 문자로 취급하며 정규화된 문자열 양식에는 하나의 일반 문자인 '%', '_', 또는 그 탈출문자로 각각 표현한다.

초성과 중성 기반 한글 음절 탐색 양식을 포함하는 문자열 양식을 정규화하기 위하여 정규화한 문자열 양식의 양식 유형을 나타내는 플래그인 배열 zPatternFlag, 각 초성과 중성 기반 한글 음절 탐색 양식의 LB와 UB를 보관하기 위한 배열 LBS와 배열 UBS, 그리고 아래의 정규화 규칙을 위에 제시한 정규화 규칙에 추가로 가진다.

(1) 배열 StringPattern의 각 초성과 중성 기반 한글 음절 탐색 양식의 전치자와 탐색자의 쌍에 대하여, zPattern에는 전치자를 저장하지 않고 탐색자만을 저장하며 그 탐색자의 LB와 UB를 제 3장에서 제시한 방법에 따라 구한 후, 그들을 배열 LBS와 배열 UBS에 각각 추가한다.
(2) $zPattern_k$가 배열 zPattern의 k 번째 탐색 양식을 나타낸다고 할 때, $zPatternFlag_k$는 $zPattern_k$의 양식 유형을 나타내며 $zPattern_k$가 차지하는 바이트 수만큼의 바이트들을 차지한다. 즉, $zPattern_k$의 탐색 양식이 한글 탐색 양식이 아닌 경우, $zPatternFlag_k$는 1-바이트로 표현되며 0의 값을 가지지만 그렇지 않은 경우에는 2-바이트로 표현되며 그 한글 탐색 양식이 초성과 중성 기반의 한글 음절 탐색 양식이 아니면 그 2-바이트는 모두 0의 값을 가지고 그 한글 탐색 양식이 초성과 중성 기반 한글 음절 탐색 양식이면 각 바이트는 다음과 같이 설정된다. 첫 번째 바이트의 선두 2-비트는 초성과 중성 기반 한글 음절 탐색 양식의 유형을 나타낸다. 그 값은 초성 유형, 초성과 중성 유형, 그리고 중성 유형에 따라 각각 01_2, 10_2, 11_2로 표기된다. 첫 번째 바이트의 나머지 6-비트는 탐색자의 유형이 중성일 경우에는 그 모음의 모음 색인을 값으로 가지지만 나머지 유형들의 경우에는 모두 0을 값으로 가진다. 두 번째 바이트는 그 양식의 LB와 UB가 배열 LBS와 배열 UBS에서 차지하는 위치인 range_index의 값을 가진다. 즉, k 번째 탐색 양식이 초성과 중성 기반 한글 음절 탐색 양식인 경우, $zPatternFlag_k$는 그 양식의 탐색자가 초성자음일 경우에는 <01000000_2 range_index>, 그 탐색자가 초성과 중성으로 구성된 음절일 경우에는 <10000000_2 range_index>, 그 탐색자가 모음일 경우에는 그 모음의 모음 색인을 구하여 <$11xxxxxx_2$ range_index>의 값으로 설정된다. 앞의 "$11xxxxxx_2$"에서 "xxxxxx"는 모음 색인에 대한 6-비트 표현을 나타낸다.

배열 LBS와 배열 UBS에 탐색자의 LB와 UB를 각각 저장하고 그 탐색자가 모음인 경우에는 그 모음의 모음 색인을 구하여 zPatternFlag에 보관하는 이유는 주어진 탐색 양식들을 다수개의 문자열들과 비교하는 데이타베이스 환경에서 그 탐색 양식들의 그 값들을 문자열과 비교할 때마다 반복적으로 구하는 것은 성능에 부담을 줄 수 있기 때문이다. 초성과 중성 기반 한글 음절 탐색 양식의 탐색자가 모음인 경우에도 LB와 UB를 구하는 이유는 그 값들이 색인의 탐색 범위를 제한하는 데 사용될 수 있기 때문이다.

예를 들어, "name LIKE '%_ab\ㅂ\여\ㅓ_%' ESCAPE '\'"의 경우, KS X 1001 부호계의 한글 완성형을 사용하여 문자를 표현하면, 문자열 양식을 나타내는 문자열 양식 배열(StringPattern)과 그를 정규화한 문자열 양식을 나타내는 정규화한 문자열 양식 배열(zPattern), 정규화한 문자열 양식에 대한 플래그 배열(zPatternFlag), 하한값 배열(LBS), 그리고 상한값 배열(UBS)은 그림 4와 같다. KS X 1001 부호계의 한글 완성형에서 각 아스키 문자는 1-바이트로, 각 한글 문자는 2-바이트로 표현된다. 그림 4에서, 예약된 문자들인 '%'와 '_'는 정규화한 문자열 양식 배열에 각각 SYN과 SUB로 표현되었으며 문자열 양식의 "%_"는 정규화한 문자열 양식 배열에 그 위치가 바뀌어 표현되었다. 아스키 문자인 'a'와 'b'의 경우, 정규화한 문자열 양식 배열에 각각 1-바이트 크기의 'a'와 'b'로 표현되고 이들에 대한 정규화한 문자열 양식에 대한 플래그 배열의 값들은 0이다. 초성과 중성 기반 한글 음절 탐색 양식들인 '\ㅂ', '\여', 그리고 '\ㅓ'는 정규화한 문자열 양식 배열에 각각 'ㅂ', '여', 그리고 'ㅓ'로 표현되었다. 초성과 중성 기반 한글 음절 탐색 양식 '\ㅂ'의 경우, 탐색자 'ㅂ'의 하한값인 '바'와 상한값인 '빠'가 하한값 배열과 상한값 배열의 위치 0에 각각 저장되고 정규화한 문자열 양식에 대한 플래그 배

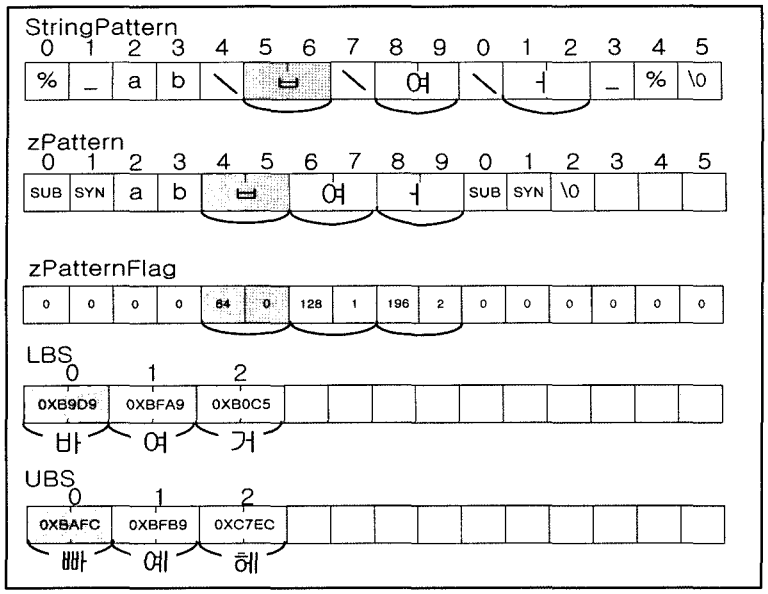

그림 4 문자열 양식 '%_ab\ㅂ\여\ㅓ_%'에 대한 문자열 탐색을 위한 관련 자료구조

열의 해당 엔트리에는 <01000000$_2$ 00000000$_2$>이 저장된다. 초성과 중성 기반 한글 음절 탐색 양식 '\여'의 경우, 탐색자 '여'의 하한값인 '여'와 상한값인 '예'가 하한값 배열과 상한값 배열의 위치 1에 각각 저장되고 정규화한 문자열 양식에 대한 플래그 배열의 해당 엔트리에는 <10000000$_2$ 00000001$_2$>이 저장된다. 초성과 중성 기반 한글 음절 탐색 양식 '\ㅓ'의 경우, 탐색자 'ㅓ'의 모음 색인 4를 구하고, 하한값인 '거'와 상한값인 '헤'가 하한값 배열과 상한값 배열의 위치 2에 각각 저장되고 정규화한 문자열 양식에 대한 플래그 배열의 해당 엔트리에는 <11000100$_2$ 00000010$_2$>이 저장된다.

정규화한 문자열 양식 배열의 엔트리들이 <P_0, P_1, ···, P_i, ···, P_{N-1}>이고, 정규화한 문자열 양식에 대한 플래그 배열의 엔트리들이 <F_0, F_1, ···, F_i, ···, F_{N-1}>이며, 문자들의 수효가 L인 비교 문자 배열의 엔트리들이 <V_0, V_1, ···, V_m, ···, V_{L-1}>라 하자.[5] 본 논문은 정규화한 문자열 양식 배열의 각 엔트리를 양식, 정규화한 문자열 양식에 대한 플래그 배열의 각 엔트리를 양식 플래그, 그리고 비교 문자 배열의 각 엔트리를 비교문자라 한다. 그림 4의 자료구조가 제시하는 바와 같이 특정 양식을 나타내는 P_i는 아스키 부호일 수도 있고 한글 문자일 수도 있다. 특정 양식 플래그를 나타내는 F_i는 P_i가 아스키 부호일 경우에는 1-바이트를 가지며 한글 문자일 경우에는 2-바이트로 표현된다. 특정 비교문자 V_m 또한 아스키 부호일 수도 있고 한글 문자일 수도 있다.

예를 들어, 그림 4의 자료구조에서 양식 P_0, P_2, P_4, P_5는 각각 SUB, 'a', 'ㅂ', '여'를 나타내며, 양식플래그 F_0, F_2, F_4, F_5는 각각 0, 0, <64, 0>, <128, 1>을 나타낸다. 따라서, <P_0, F_0>의 쌍은 0 번째 양식이 '_'임을 의미하며, <P_2, F_2>의 쌍은 2 번째 양식이 알파벳 'a'임을 의미하며, <P_4, F_4>의 쌍은 4 번째 양식이 초성과 중성 기반 한글 음절 탐색 양식의 초성 유형이고 그 초성자음은 'ㅂ'이며 탐색자 'ㅂ'의 하한값인 '바'와 상한값인 '빠'가 하한값 배열과 상한값 배열의 위치 0에 각각 저장되어 있음을 나타내며, <P_5, F_5>의 쌍은 5 번째 양식이 초성과 중성 기반 한글 음절 탐색 양식의 초성과 중성 유형이고 그 음절은 '여'이며 탐색자 '여'의 하한값인 '여'와 상한값인 '예'가 하한값 배열과 상한값 배열의 위치 1에 각각 저장되어 있음을 나타낸다. 제 4.2절에서 특정 <P_i, F_i>의 쌍에 특정 비교문자 V_m이 부합하는지를 검사하는 알고리즘을 그리고 제 4.3절에서 전체 문자열 양식에 특정 문자열이 부합하는 지를 검사하는 알고리즘을 차례로 제시한다.

4.2 특정 양식과 특정 문자의 비교

본 절은 특정 양식과 양식 플래그의 쌍에 대하여 특정 비교문자가 부합하는 지를 검사하는 알고리즘 Pattern_Match()를 제시한다. 그림 5의 알고리즘 Pattern_Match(Pi, Fi, V_m, L)에서 <P_i, F_i>의 쌍은 특정 양식과 양식 플래그의 쌍을 의미하며 V_m은 비교문자를, L은 비교문자 배열에서 비교 문자 V_m을 포함하여 나머지 문자들의 수를 나타낸다. 알고리즘 Pattern_Match()는

5) 알고리즘 제시의 편의를 위하여 L을 비교문자들의 수효로 설정하였다. 이는 정확한 표현이 아니다. 실제로는 비교문자들의 바이트 수효로 수행된다. 상세한 알고리즘은 참고문헌 [14]에 제시되어 있다.

```
int Pattern_Match(Pi, Fi, L, Vm)
{
  if (L == 0) return NO_MATCH_FOUND;
  if (Pi의 MSB == 0) { /* an ASCII character. MSB는 Most Significant Bit를 의미한다. */
      if (Pi != Vm) return NO_MATCH_FOUND;
  } else {/* 한글 탐색 양식 */
      if (Vm의 MSB == 0) return NO_MATCH_FOUND; /* an ASCII character */
      if (Fi == 0) { /* 기존 한글 탐색 양식 */
          if (Pi != Vm) return NO_MATCH_FOUND;
      } else { /* 초성과 중성 기반 한글 음절 탐색 양식 */
          type = Fi의 첫 번째 바이트의 선두 2-비트의 값;
          if (type == 1 || type == 2) { /* Type_ROW 또는 Type_CELL */
              range_index = Fi의 두 번째 바이트 값;
              if ((Vm<LBS[range_index])||(UBS[range_index]<=Vm))
                  return NO_MATCH_FOUND;
          } else { /* Type_COLUMN */
              vowel_index = Fi의 첫 번째 바이트의 마지막 6-비트;
              if (verify_vowel_index(vowel_index,Vm) == FALSE)
                  return NO_MATCH_FOUND;
          }
      }
  }
  return MATCH_FOUND;
}
```

그림 5 특정 양식과 특정 문자의 비교 알고리즘 Pattern_Match()

아스키 부호의 대문자와 소문자를 구분한다.

알고리즘 Pattern_Match()는 제 3장에서 제시한 바와 같이 초성과 중성 기반 한글 음절 탐색 양식에 대하여 Type_ROW와 Type_CELL의 경우에는 미리 설정해 둔 하한값과 상한값의 범위에 비교문자가 존재하는 지를 검사하며, Type_COLUMN의 경우에는 비교문자의 모음색인을 구하여 그 값이 미리 설정해둔 탐색자의 모음색인과 일치하는 지를 제 3.4절에 제시한 함수 verify_vowel_index()로 검사한다.

4.3 문자열 양식과 문자열의 비교

특정 비교 문자열이 연산자 LIKE의 문자열 양식에 부합하는 지의 식별은 그 비교 문자열에 대한 <비교 문자 배열, 비교 문자 배열의 길이>의 쌍이 그 문자열 양식에 대한 그림 4의 <정규화한 문자열 양식 배열, 정규화한 문자열 양식에 대한 플래그 배열, 하한값 배열, 상한값 배열>의 기준에 부합하는 지를 검사하는 것이다. 비교 문자열은 데이타베이스에 저장된 레코드의 특정 칼럼의 칼럼값 또는 그 칼럼을 기반으로 하는 색인에서 그 칼럼의 칼럼값이므로 비교 문자열이 NULL 문자로 끝남을 보장하지 않을 수 있다. 따라서, 비교 문자열의 길이를 반드시 제공하여야 한다. 정규화한 문자열 양식 배열의 앞부분인 <P_0, P_1, …, P_{i-1}>과 비교 문자 배열의 앞 부분인 <V_0, V_1, …, V_{m-1}>이 부합한다면 정규화한 문자열 양식 배열에서 잔여(residue) 양식들은 <P_i, P_{i+1}, …, P_{N-1}>이고, 정규화한 문자열 양식에 대한 플래그 배열에서 잔여 양식플래그들은 <F_i, F_{i+1}, …, F_{N-1}>이며, 잔여 비교문자들은 <V_m, V_{m+1}, …, V_{L-1}>이 된다. 이 경우, P_i를 현재양식, F_i를 현재양식플래그, V_m을 현재비교문자라 하고, P_{i+1}을 다음양식, F_{i+1}을 다음양식플래그, V_{m+1}을 다음비교문자라 한다. 그림 6의 알고리즘 LikeMatch(P, F, V, L)은 재귀 알고리즘(recursive algorithm)으로서 잔여 양식들 P, 잔여 양식플래그들 F, 잔여 비교문자들 V, 그리고 잔여 비교문자들의 수효 L을 입력으로 수행된다.

알고리즘 LikeMatch()에서 현재양식 P_i가 SYN 즉, 예약된 문자 '%'인 경우, 다음양식 P_{i+1}이 NULL 문자가 아닌 경우에 한하여, 다음양식 P_{i+1}과 다음양식플래그 F_{i+1}에 부합하는 임의의 비교문자 $V_m(x \le m < L)$을 구하고 그 비교문자 이후의 비교문자들이 다음양식 이후의 양식들과 다음양식플래그 이후의 양식플래그들에 부합하는 지를 확인한다. 그러한 잔여 비교문자 V_m을 찾았다면 MATCH_FOUND를, 그렇지 않다면 NO_MATCH_FOUND를 반환한다.

5. 초성과 중성 기반 한글 음절 탐색 양식을 고려한 색인 탐색 알고리즘

연산자 LIKE의 해당 칼럼에 색인이 존재하는 경우, 그 연산자의 문자열 양식에 부합하는 문자열들을 그 색인에서 탐색한다면 질의 수행의 성능은 향상될 수 있다. 본 장은 LIKE 연산을 색인을 이용하여 수행하는 경우, 초성과 중성 기반 한글 음절 탐색 양식을 고려한 색인의 탐색 방법을 관계 DBMS인 바다-II의 색인 탐색 기법을 기준으로 제시한다. 본 장에서 색인이라 함은 B^+-트리를 의미하며, 연산자 LIKE라 함은 LIKE와 NOT LIKE를 함께 의미한다.

바다-II의 색인 탐색은 탐색할 색인의 각 색인 칼럼에

```
int LikeMatch(P, F, V, L)
{
    i = 0;          /* 현재양식 Pi, 다음양식 Pi+1, 현재양식플래그 Fi, 다음양식플래그 Fi+1 */
    x = 0;          /* 현재비교문자 Vx, 다음비교문자 Vx+1 */

    while (Pi != NULL) {
        switch(Pi) {
            case SYN: /* 예약된 문자 '%' */
                if (Pi+1 == NULL) return MATCH_FOUND;
                else {
                    for (m = x; m < L; m= m+ 1) {
                        if ((Pattern_Match(&Pi+1, &Fi+1, L-m , &Vm) == MATCH_FOUND) &&
                          (LikeMatch(&Pi+2, &Fi+2, &Vm+1, L-m-1) == MATCH_FOUND))
                            return MATCH_FOUND;
                    }
                }
                return NO_MATCH_FOUND;
            case SUB: /* 예약된 문자 '_' */
                if (L== 0) return NO_MATCH_FOUND;
                break;
            default:
                if (Pattern_Match(&Pi, &Fi, L, &Vx) == NO_MATCH_FOUND)
                    return NO_MATCH_FOUND;
                break;
        } /* end of switch */
        i++; x++; L--;
    } /* end of while */
    if (L == 0) return MATCH_FOUND;
    else return NO_MATCH_FOUND;
}
```

그림 6 문자열 양식과 문자열의 비교 알고리즘 LikeMatch()

대하여 탐색할 칼럼값들의 하한값과 상한값의 범위를 가지는 칼럼값 범위(ColumnValueRange, CVRange)의 속성과 칼럼값 범위에 속하는 칼럼값들이 추가로 만족해야할 조건을 가지는 칼럼값 여과(ColumnValueFilter, CVFilter)의 속성들을 가질 수 있으며, 그 색인의 색인 칼럼들에 대한 칼럼값 범위 속성들의 리스트인 칼럼값 범위 속성 리스트(ColumnValueRangeList, CVRangeList)와 칼럼값 여과 속성들의 리스트인 칼럼값 여과 속성 리스트(ColumnValueFilterList, CVFilterList)를 기반으로 수행된다. 예를 들어, 칼럼 id에 대해 색인이 존재한다면 "id ≥ 5 and id < 10"의 WHERE 절을 수행하기 위한 그 색인 탐색의 칼럼값 범위 속성은 "5 ≤ id < 10"로 설정된다. 이 경우, 그 칼럼값 범위 속성은 그 WHERE 절과 동일한 의미를 가짐으로써 칼럼값 여과 속성을 필요로 하지 않는다. 그러나, "id ≥ 5 and id < 10 and id != 7"의 WHERE 절의 경우, 그 색인 탐색의 칼럼값 범위 속성은 "5 ≤ id < 10"가 되며 칼럼값 여과 속성은 "id != 7"이 된다. 이 경우, 그 색인 탐색에서 그 칼럼값 범위 속성을 만족하는 칼럼값들은 그 칼럼값 여과 속성에 명시된 조건을 추가로 만족해야 한다.

바다-II의 색인 탐색에서 하나의 칼럼값 범위 속성은 하나의 색인 칼럼에 대한 <칼럼명(ColumnName), 하한값(Lower Bound, LB), 하한값 포함 플래그(Lower Bound Inclusion Flag, LBIF), 상한값(Upper Bound, UB), 상한값 포함 플래그(Upper Bound Inclusion Flag, UBIF)>로 구성된다. 칼럼명은 색인 칼럼의 칼럼명을 나타내며, 하한값과 상한값은 각각 그 색인 칼럼에 대한 탐색 범위의 하한값과 상한값을 나타내고, 하한값 포함 플래그와 상한값 포함 플래그는 각각 하한값과 상한값에 설정된 값의 포함 여부를 나타내며 TRUE 또는 FALSE를 그 값으로 가진다. 바다-II의 색인 탐색에서 하나의 칼럼값 여과 속성은 하나의 색인 칼럼에 대한 <칼럼명(ColumnName), 연산자(Operator), 값(Value), 정규화한 문자열 양식 배열(zPattern), 정규화한 문자열에 대한 양식에 대한 플래그 배열(zPatternFlag), 하한값 배열(LBS), 상한값 배열(UBS)>로 구성된다. 칼럼명은 색인 칼럼의 컬럼명을 나타내며, 연산자는 NOT_EQUAL(연산자 !=), LIKE(연산자 LIKE), 또는 NOT_LIKE(연산자 NOT LIKE)를 그 값으로 가질 수 있다. 값은 그 칼럼값 여과 속성의 연산자가 NOT_EQUAL인 경우만을 위한 것이며 NOT_EQUAL 연산자의 오른쪽 피연산자의 값으로 설정된다. 즉, "C != *some_value*"의 경우, 필드 값은 *some_value*로 설정된다. 정규화한 문자열 양식 배열, 정규화한 문자열 양식에 대한 플래그 배열, 하한값 배열, 상한값 배열의 필드들은 칼럼값 여과 속성의 연산자가 LIKE와 NOT_LIKE인 경우만을 위한 것이다. 이들은 각각 연산자 LIKE의 문자열 양식에 대한 정규화한 문자열 양식 배열, 그 정규화한 문자

열 양식에 대한 플래그 배열, 그 정규화한 문자열 양식 배열에서 각 양식의 하한값과 상한값을 저장하기 위한 하한값 배열과 상한값 배열을 의미한다. 연산자 LIKE의 문자열 양식을 위한 칼럼값 여과 속성의 경우, 즉, 그 칼럼값 여과 속성의 연산자가 LIKE 또는 NOT_LIKE인 경우, 그 칼럼값 여과 속성의 정규화한 문자열 양식 배열, 정규화한 문자열 양식에 대한 플래그 배열, 하한값 배열, 그리고 상한값 배열의 필드들을 이용하여 색인의 칼럼값 범위 속성을 만족하는 칼럼값들이 그 칼럼값 여과 속성에 명시된 조건을 추가로 만족하는지를 제 4장에서 제시한 방법을 적용하여 검사한다.

연산자 LIKE의 문자열 양식에서 예약된 문자 '_'는 임의의 하나의 문자와 부합하며 예약된 문자 '%'는 임의의 길이의 임의의 문자열과 부합하므로 색인 탐색에서 이들이 부합할 수 있는 문자의 상한값을 무한대(無限大, infinity)의 의미를 가지는 문자값으로 설정해야 한다.

정의 3. 어떤 문자에도 할당되지 않았으며 어떤 문자에 할당된 값보다 더 큰 값을 MAX_CHARACTER라 하고 그 값을 0xFFFF로 설정한다.

KS X 1001 부호계에는 한글, 한자, 기타 문자 등을 포함하여 한글 완성형은 0xFDFE까지, 한글 조합형은 0xF9FE까지의 문자값들이 할당되어 있다. 따라서, KS X 1001 부호계에서 MAX_CHARACTER의 값을 0xFFFF로 설정하는 것은 정당성을 가진다.[6)]

본 장의 나머지 부분에서 연산자 LIKE의 문자열 양식에서 '%'와 '_'의 예약된 문자들과 초성과 중성 기반 한글 음절 탐색 양식의 탐색자들을 범위 부합 문자들(range match characters)이라 하고, 그 외의 다른 문자들을 단일 부합 문자들(single match characters)이라 한다.

정의 4. 칼럼값 범위 속성의 필드 하한값과 상한값이 동일 값을 가질 경우, 그 칼럼값 범위 속성을 정확한 부합 칼럼값 범위(exact match column value range) 속성이라 하며, 그렇지 않은 경우에는 그를 범위 부합 칼럼값 범위(range match column value range) 속성이라 한다.

SQL 문의 특정 술어들을 수행하기 위한 색인 탐색의 칼럼값 범위 속성이 정확한 부합 칼럼값 범위 속성일 경우에는 그 술어들에 대한 칼럼값 여과 속성을 설정할 필요가 없다. 따라서, 연산자 LIKE의 문자열 양식이 단일 부합 문자들로만 구성된 경우에는 정확한 부합 칼럼값 범위 속성으로 그 문자열 양식과 동일한 의미를 표현할 수 있다. 그러나, 연산자 LIKE의 문자열 양식이 범위 부합 문자들을 포함하는 경우에는 칼럼값 범위 속성만으로 그 문자열 양식과 동일한 의미를 가지도록 표현할 수 없고 칼럼값 범위 속성과 칼럼값 여과 속성의 쌍으로 표현해야 한다. 단, 연산자 LIKE의 문자열 양식이 예약된 문자 '%'로만 구성된다면 칼럼값 범위 속성만으로 그 문자열 양식과 동일한 의미를 표현할 수 있다.

하나의 "*column* LIKE *pattern*"의 연산을 수행하기 위한 색인 탐색의 칼럼값 범위 속성 CVRange와 칼럼값 여과 속성 CVFilter의 설정은 문자열 양식인 *pattern*에 대해 제 3장에서 제시한 기법으로 설정한 정규화한 문자열 양식 배열인 zPattern, 그 정규화한 문자열 양식 배열에 대한 양식 플래그 배열인 zPatternFlag, 그리고 그 정규화한 문자열 양식 배열에서 각 탐색 양식의 하한값과 상한값을 저장하는 하한값 배열인 LBS와 상한값 배열인 UBS를 바탕으로 그림 7의 알고리즘 Make_CVRange_CVFilter()로 구성된다.

예를 들어, 초성과 중성 기반 한글 음절 탐색 양식을 가진 "name LIKE '\ㅂ\여\ㅓ' ESCAPE '\'"의 경우, 칼럼 name에 대한 색인 탐색을 위한 칼럼값 범위 속성은 <'name', '바여거', TRUE, '빠', FALSE>로, 칼럼값 여과 속성은 <'name', LIKE, NULL, 'ㅂ여ㅓ', '01000000_2 00000000_2 10000000_2 00000001_2 11000100_2 00000010_2', '바여거', '빠예헤'>로 설정된다. 이는 문자열 양식 '\ㅂ\여\ㅓ'에 대하여 [정규화한 문자열 양식 배열, 정규화한 문자열 양식에 대한 플래그 배열, 하한값 배열, 상한값 배열]을 ['ㅂ여ㅓ', '01000000_2 00000000_2 10000000_2 00000001_2 11000100_2 00000010_2', '바여거', '빠예헤']로 제4장에서 제시한 기법으로 구한 후 이를 바탕으로 설정된 것이다.

하나의 칼럼에 대한 술어들이 연산자 NOT LIKE 또는 연산자 "!="만으로 구성된 경우, 질의 최적화기(query optimizer)는 그 술어들을 수행하기 위한 방법으로 색인을 이용한 탐색을 선택하지 않을 수 있다. 그러나, 하나의 칼럼에 대한 술어들이 연산자 NOT LIKE와 연산자 "!=" 만으로 구성되지 않은 경우, 질의 최적화기는 그 술어들을 수행하기 위한 방법으로 색인 탐색

6) KS X 1001 부호계에서뿐만 아니라 유니코드를 사용하는 경우에도 MAX_CHARACTER를 0xFFFF로 설정할 수 있다. 유니코드는 어떤 문자에도 할당되지 않도록 영구히 예약해 둔 66개의 부호점들 즉, 66개의 비문자(非文字, noncharacter)들을 가진다[15]. 부호점 0xFFFF는 그 비문자들의 부호점들 중의 하나이다. 즉, 유니코드에 정의된 어떠한 문자도 그 부호점으로 0xFFFF를 가지지 않는다. 유니코드의 문자들은 유니코드의 부호화 방법들(encoding schemes)[16]인 UTF-8, UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE, UTF-32LE 중의 어느 하나의 방법으로 부호화된다. 따라서, 유니코드의 부호화 방법과 유니코드의 문자에 할당된 부호점이 속한 범위에 따라 특정 부호화 방법으로 부호화된 유니코드의 한 문자는 최소 1-바이트에서 최대 4-바이트로 표현된다. 이들 중 어떤 부호화 방법을 사용하더라도 부호화된 문자의 1-바이트 값(부호화된 그 문자의 바이트 길이가 1인 경우) 또는 부호화된 그 문자의 선두 2-바이트 값(부호화된 그 문자의 바이트 길이가 2 이상인 경우)은 항상 0xFFFF보다 작다. 따라서, 유니코드에서 MAX_CHARACTER의 값을 0xFFFF로 설정하는 것은 정당성을 가진다.

Make_CVRange_CVFilter(CVRange,*Column*, zPattern, zPatternFlag, LBS, UBS)
{
단계_1. CVRange에 대하여 ColumnName을 *Column*으로, LB와 UB를 빈 문자열(empty string)로, 그리고 LBIF와 UBIF를 TRUE로 설정한다.
단계_2. 배열 zPattern의 첫 번째 양식에서 마지막 양식까지 차례로 각 양식 P_i에 대하여 다음의 규칙을 적용한다. 단, 양식 P_i에 해당하는 양식 플래그를 F_i라 한다.
(1) 양식 P_i가 SYN 또는 SUB인 경우. 다음의 세 개의 하위 단계들을 차례로 수행한다. 1-1) P_i가 SYN이면서 zPattern의 마지막 양식이 아니거나 P_i가 SUB인 경우에 한하여 CVRange의 LBIF를 FALSE로 설정한다. 참고로, 그 CVRange의 LB를 변경시키지 않는다. 1-2) CVRange의 UBIF가 TRUE인 경우에 한하여, CVRange의 UB에 MAX_CHARACTER를 추가하고 UBIF를 FALSE로 설정한다. 1-3) zPattern의 나머지 양식들을 고려하지 않고 단계 3으로 이동한다.
(2) 양식 P_i가 초성과 중성 기반 한글 음절 탐색 양식인 경우 즉, 양식 플래그 F_i가 0이 아닌 경우. 양식 플래그 F_i의 두 번째 바이트 값을 추출하여 변수 range_index에 설정하고, LBS[range_index]를 CVRange의 LB에 추가하며, CVRange의 UBIF가 TRUE인 경우에 한하여 UBS[range_index]를 CVRange의 UB에 추가하며 CVRange의 UBIF를 FALSE로 설정한다.
(3) 양식 P_i가 그 이외인 경우. 양식 P_i를 CVRange의 LB에 추가하고, CVRange의 UBIF가 TRUE인 경우에 한하여 양식 P_i를 CVRange의 UB에 추가한다.
단계_3. CVRange의 LB와 UB를 문자열로 만들기 위하여 이들에 '\0'를 추가한다.
단계_4. CVRange의 LBIF와 UBIF가 모두 TRUE이거나, LB가 빈 문자열이고 LBIF가 TRUE이며 UB가 MAX_CHARACTER 만으로 구성된 문자열일 경우에는 CVFilter를 생성하지 않고 NULL을 반환한다. 그 외의 경우에는 하나의 CVFilter를 생성하고 그 CVFilter의 ColumnName을 *Column*으로, Operator를 LIKE로 설정하며, zPattern, zPatternFlag, LBS, 그리고 UBS를 각각 입력인자인 zPattern, zPatternFlag, LBS, UBS로 설정한 후 그 CVFilter를 반환한다.
}

그림 7 칼럼값 범위 속성과 칼럼값 여과 속성의 설정

을 선택할 수 있다. 연산자 NOT LIKE 또는 연산자 "!="의 경우, 그 연산자의 대상 칼럼이 'name'이라면 그 칼럼값 범위 속성은 <'name', '', TRUE, 'MAX_CHARACTER', FALSE>로 설정된다. 즉, 이들의 칼럼값 범위 속성은 "name LIKE '%'"의 경우와 동일하다. 그러나, 칼럼값 여과 속성은 "name != *some_string*"의 경우에는 <'name', NOT_EQUAL, *some_string*, NULL, NULL, NULL, NULL>로 설정되며, "name NOT LIKE *some_pattern*"의 경우에는 *some_pattern*의 유형에 따라 다음의 두 가지 다른 방법들로 설정될 수 있다. 만약, *some_pattern*이 단일 부합 문자들로만 구성된다면 위의 술어는 "name != *some_pattern*"의 경우와 동일하게 처리된다. 그렇지 않다면, *some_pattern*에 대한 정규화한 문자열 양식 배열, 정규화한 문자열 양식에 대한 플래그 배열, 하한값 배열, 그리고 상한값 배열을 구한 후, 칼럼값 여과 속성을 <'name', NOT_LIKE, NULL, 정규화한 문자열 양식 배열, 정규화한 문자열 양식에 대한 플래그 배열, 하한값 배열, 상한값 배열>로 설정한다.

하나의 색인이 복합 칼럼들로 구성된 경우와 질의의 술어들이 복잡한 AND/OR로 연결된 경우, 색인 탐색을 위한 칼럼값 범위 속성 리스트와 칼럼값 여과 속성 리스트의 설정에 대한 논의는 본 논문의 범위를 벗어나므로 그에 대한 알고리즘의 제시는 본 논문에서 생략한다. 그 대신, 여러 술어들을 하나의 색인으로 수행하는 경우들 중에서 비교적 간단한 경우인 하나의 칼럼에 대한 여러 술어들이 AND 연산자로 연결된 경우에 대하여 색인 탐색을 위한 칼럼값 범위 속성 리스트와 칼럼값 여과 속성 리스트 설정의 기본 아이디어만을 제시한다. 그 기본 아이디어는 다음과 같다. 첫째, 동일 칼럼에 대한 각 술어에 대해 하나씩의 칼럼값 범위 속성과 칼럼값 여과 속성을 생성한다. 둘째, 그 칼럼값 범위 속성들의 공통 범위를 도출하고 그 범위를 그 색인의 칼럼값 범위 속성으로 설정하며 그 속성을 만족하는 칼럼값들에 적용할 칼럼값 여과 속성들을 그 술어들에 대한 칼럼값 여과 속성들로부터 찾아 그들을 그 색인의 칼럼값 여과 속성들로 설정한다. 셋째, 공통의 칼럼값 범위 속성을 도출할 수 없다면 그 술어들을 만족하는 결과 레코드가 없는 것으로 판단한다.

예를 들어, 질의 "SELECT name, address FROM employees WHERE name LIKE '\ㅂ\여\ㅓ' ESCAPE '\' AND name NOT LIKE '배연걸' AND name NOT LIKE '_열서';"를 칼럼 name에 대한 색인으로 탐색한다고 하자. 이 경우, 색인 탐색의 칼럼값 범위 속성 리스트와 칼럼값 여과 속성 리스트의 설정 과정은 다음과 같다. 첫째, 술어 "name LIKE '\ㅂ\여\ㅓ'"에 대하여 칼럼값 범위 속성을 <'name', '바여거', TRUE, '빠', FALSE>로 그리고 칼럼값 여과 속성을 <'name', LIKE, NULL, 'ㅂ여ㅓ', '01000000_2 00000000_2 10000000_2 00000001_2 11000100_2 00000010_2', '바여거', '빠예헤'>로

설정한다. 둘째, 술어 "name NOT LIKE '배연걸'"에 대하여 칼럼값 범위 속성을 <'name', '', TRUE, 'MAX_CHARACTER', FALSE>로 그리고 칼럼값 여과 속성을 <'name', NOT_EQUAL, '배연걸', NULL, NULL, NULL, NULL>로 설정한다. 셋째, 술어 "name NOT LIKE '_열서'"에 대하여 칼럼값 범위 속성을 <'name', '', TRUE, 'MAX_CHARACTER', FALSE>로 그리고 칼럼값 여과 속성을 <'name', NOT_LIKE, NULL, '*SUB*열서', '00000000_2 00000000_2 00000000_2 00000000_2 00000000_2', NULL, NULL>로 설정한다. 마지막으로, 칼럼값 범위 속성 리스트인 CVRangeList를 위의 세 개의 칼럼값 범위 속성들의 공통 범위인 첫 번째 칼럼값 범위 속성으로만 구성한다. 그러나, 칼럼값 여과 속성 리스트인 CVFilterList는 위의 세 개의 칼럼값 여과 속성들을 AND로 연결한 리스트로 형성한다.

6. 결 론

본 논문은 연산자 LIKE의 기존 한글 음절 탐색 양식에 추가하여 한글의 초성과 중성을 기반으로 하는 초성과 중성 기반 한글 음절 탐색 양식을 제안하였다. 초성과 중성 기반 한글 음절 탐색 양식은 전치자와 탐색자로 구성되며 전치자는 탈출문자 또는 새롭게 정의된 예약된 문자일 수 있으며 탐색자는 한글의 초성자음, 초성과 중성만으로 구성된 한글 음절, 그리고 한글의 모음일 수 있다.

본 논문은 초성과 중성 기반 한글 음절 탐색 양식에 부합하는 문자열들을 식별하는 알고리즘을 KS X 1001 부호계의 한글 완성형을 기반으로 제시하였으며 초성과 중성 기반 한글 음절 탐색 양식에 부합하는 문자열들을 색인을 이용하여 탐색하는 방법을 함께 제시하였다. KS X 1001 부호계의 한글 완성형을 기반으로 하는 본 논문의 초성과 중성 기반 한글 음절 탐색 양식은 관계 DBMS인 바다-II에 구현되었으며 유니코드의 UTF-8 부호화 방법을 기반으로 하는 초성과 중성 기반 한글 음절 탐색 양식은 임베디드(embedded) DBMS인 CellDB[17]에 구현되었다[18].

DB2[19], 오라클[20], MS SQL Server[21] 등의 DBMS들은 국제화, 지역화를 지원하기 위해 유니코드와 다국어 기능을 제공한다. 이에 추가하여 본 논문이 제시하는 초성과 중성 기반 한글 음절 탐색 양식과 같이 데이타베이스에 저장된 문자들의 언어적 특성을 고려한 연산자 LIKE의 문자열 양식이 DBMS에서 지원된다면 그 언어로 표현된 문자열 탐색이 매우 용이하게 될 것이며 문자 집합에 따른 SQL 응용 프로그램의 호환성 문제를 해결할 수 있을 것이다.

참 고 문 헌

[1] American National Standards Institute, The Database Language SQL, Standard No. X3.135-1992, New York, 1992.

[2] J. Melton, A. R. Simon, Understanding the new SQL: A complete guide, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1993.

[3] Korean Standards Information Center, Code for information interchange (Hangul and Hanja), Standard No. KS X 1001, 2002.

[4] The Unicode, http://www.unicode.org

[5] 김경석, 컴퓨터 속의 한글 이야기 -둘째 보따리-, 부산대학교 출판부, 1999.

[6] Korean Standards Information Center, Keyboard layout for information processing, Standard No. KS X 5002, 1982.

[7] 조관현, 한글 두음 문자열 검색 방법 및 장치 (METHOD AND APPARATUS FOR SEARCHING THE INITIAL SOUND STRING KOREAN CHARACTER), 대한민국 특허 등록 번호 10-02850119, 등록일자 2000년 12월 29일.

[8] ORACLE, *Oracle 10g Downloads*, http://www.oracle.com/technology/software/products/database/oracle10g/index.html, 2005.

[9] Microsoft, *LIKE (Transact-SQL)*, http://msdn2.microsoft.com/en-us/library/ms179859.aspx, 2006.

[10] Y. C. Park, J. H. Cho, G. J. Cha, and P. Scheuermann, "Efficient Schemes of Executing Star Operators in XPath Query Expressions," Proc. of the 11th International Conference on DASFAA, pp. 264-278, April 2006.

[11] 박준현, 박영철, 이진수, "B+-트리에서 키와 구분자의 저장과 탐색", 한국정보과학회 논문지(C), 제3권 제6호, pp. 568-580, 1997.

[12] S. H. Kim, M. S. Jung, J. H. Park, and Y. C. Park, "A Design and Implementation of Savepoints and Partial Rollbacks considering Transaction Isolation Levels of SQL2," Proc. of the 6th International Conference on DASFAA, pp. 303-312, April 1999.

[13] Y. C. Park, M. H. Cha, and J. H. Park, "An Efficient Scheme of Deleting All Records in a Table," Proc. of the 7th World Multiconference on Systemics, Cybernetics and Informatics, pp 203-208, 2003.

[14] 노은향, LIKE 연산에서 한글 탐색 양식[석사학위 논문], 경북대학교, 2006.

[15] The Unicode Consortium, Unicode 4.0 Special Areas and Format Characters, http://www.unicode.org/versions/Unicode4.0.0/ch15.pdf

[16] The Unicode Consortium, Unicode 4.0 Conformance, http://www.unicode.org/versions/Unicode4.0.0/ch03.pdf

[17] http://www.celldb.co.kr

[18] S. C. Park, E. H. Lo, J. C. Park, Y. C. Park, "A Korean Search Pattern in the LIKE Operation,"

Proc. of the 9th International Conference on Enterprise Information Systems, June 2007(to be appeared).
[19] S. Poon, M. Sud, R. Chong, Understanding DB2 Universal Database character conversion, http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0506chong/, IBM, 2005.
[20] S. Law, Globalization Support Oracle Unicode database support. An Oracle White Paper, Oracle Corporation, 2001.
[21] M. Kaplan, International Features in Microsoft SQL Server 2000, http://msdn.microsoft.com/library/default.asp? url=/library/en-us/dnsql2k/html/intlfeature-sinsqlserver2000.asp, 2001.

박 성 철
1996년 경북대학교 컴퓨터 과학과 학사 1998년 경북대학교 컴퓨터 과학과 석사 1998년~현재 경북대학교 컴퓨터 과학과 박사과정. 2000년~2005년 ㈜K4M 근무 관심분야는 XML 데이타베이스 시스템, 임베디드 데이타베이스 시스템, 웹 서비스 등

노 은 향
2004년 계명대학교 컴퓨터 공학과 학사 2006년 경북대학교 컴퓨터 과학과 석사 2007년~현재 한국신발피혁연구소 생산시스템 연구팀 연구원. 관심분야는 Database, Automation, Motion analysis, Biomechanics

박 영 철
1977년 서울대학교 전기공학과 학사. 1986년 Northwestern Univ. 전산학 석사 1989년 Northwestern Univ. 전산학 박사. 1993년~현재 경북대학교 전자전기컴퓨터학부 교수. 2007년~현재 한국정보과학회 이사. 관심분야는 임베디드 데이타베이스 시스템, 트랜잭션 처리, XML

박 종 철
2005년 계명대학교 학사. 2005년~현재 ㈜퓨전소프트. 관심분야는 임베디드 시스템, 모바일 게임 등