결합 비트(Combined Bit) 데이터 타입 제안 및 설계

이정준, 한윤희°
한국산업기술대 컴퓨터공학과
{yuni1016°, jjlee}@kpu.ac.kr

Proposing and Design of Combined Bit Data type

Jeong-Joon Lee, YunHee Han
Korea Polytechnic University

요 약

네비게이션 시스템, PDA, DMB단말기, 휴대폰 등과 같이 모바일 단말기가 보편화 되면서 이 시스템을 유지하기 위한 데이터베이스 관리시스템의 수요가 증가하고 있다. 이러한 모바일 단말기는 기존의 데이터베이스 관리 시스템과는 달리 공간적인 소형화와 에너지 소비량을 최소화 해야 한다. 본 논문은 두 가지에 중점을 두어 결합 비트(combined bit)데이터 타입을 제안하고 구현방법을 제시했다. 결합 비트를 사용함으로써 저장공간의 절약, 검색 속도 향상, 구성비트 복합 검색의 효율적 지원, 결합 비트의 도입으로 질의의 편리성의 이점이 있다.

1. 서 론

최근 들어 모바일 단말기(네비게이션, PDA, DMB단말기, 휴대폰 등)의 사용이 보편화 되고 있다. 이러한 모바일 단말기에서 필요로 하는 데이터의 양과 복잡성은 데이터베이스 관리 시스템을 사용하기에 충분하다. 일반적인 데이터베이스 관리 시스템과는 다르게 모바일 단말기에서는 데이터베이스 관리 시스템이 최소한의 사용공간을 가져야 하며 에너지 소비를 최소화 해야 한다[1,2,3].

본 논문에서는 모바일 단말기의 데이터베이스 관리 시스템에서 사용공간을 최소화 할 수 있는 결합비트(Combined Bit)데이터 타입을 제시하고 구현 방법을 제시한다.

본 논문의 구성은 다음과 같다. 2장은 기존 기술에 관해 설명하고 문제점을 제시한다. 3장에서는 결합 비트의 데이터 타입 및 구현 방법에 관해 설명하고 4장에서는 결론을 서술한다.

2. 기존 기술의 문제점

기존의 모바일 단말기에서의 구현된 기술에 대한 문제점을 제시한다.

(1) 숫자의 위치에 의미를 부여하는 코드 체계의 단점

기존의 코드 체계를 보면, 숫자의 위치에 의미를 부여하도록 하여 하나의 데이터 타입에서 여러 가지 데이터가 들어가도록 할 수 있다. 예를 들면, 그림 1 과 같이 주민번호 앞자리의 경우, 아래에서 앞의 두 자리(1,2번)는 생년, 3,4번 자리는 생월, 5,6번 자리는 생일을 의미하는 것을 알 수 있다. 그러나 이러한 방법은 다음과 같은 단점을 가지고 있다.

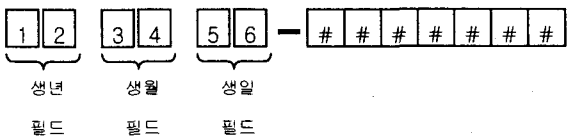

그림 1. 주민번호의 구성

첫째, 앞쪽에 포함된 생년 이외의 생월, 생일을 이용한 검색이 쉽지 않다. 정수 타입을 이용하여 저장하면 산술식을 이용하여(나머지, 몫의 값을 이용)검색해야 하지만 것은 인덱스를 사용할 수 없으므로 효율적이지 않다. 또한, 스트링을 이용하여 저장하더라도, 와일드 캐릭터(wild character : %, ?)를 prefix로 사용해야 하므로 인덱스를 사용할 수 없다. 따라서 첫 번째 의미 값을 제외한 다른 값들에 대해서는 검색을 효율적으로 할 수 없다.

둘째, 공간을 효율적으로 사용하지 못한다. 정수형 타입으로 저장하는 경우, 실제로 사용할 수 없는 영역까지 포함하고 있으므로 저장 공간이 낭비되고 있다. 예를 들어, 위의 주민번호 예제에서 3, 4번 자리는 생년월일이므로 01~12까지의 수만 사용하지만, 실제 공간은 00~99까지의 숫자를 저장할 수 있는 공간을 할애하고 있다. 즉, 01~12를 저장하기 위해서는 4bit가 필요하지만, 약 10.7(=32/3)bit를 사용한다. 왜냐하면, 6자리 이상의 정수를 저장하기 위해서는 3byte형의 데이터 타입이 없으므로, 4byte(32bit)형이 데이터 타입을 사용해야 하는데, 1/3 자리를 차지하므로, 32/3 비트만큼 사용하고 있다. 또한, 캐릭터 형을 사용하더라도, 한 문제에 1byte(8bit)이므로 생월 필드를 분리하여 비트 타입을 사용하는 경우

보다 두 배의 공간을 사용한다.

(2) 의미 별로 필드를 나누어 저장하는 방법의 단점
이 방법은 위의 주민번호 앞자리의 생년, 생월, 생일을 모두 분리하여 별도의 필드에 저장하도록 스키마를 구성하는 방법이다.

첫째, 기존의 비트 데이터 타입은 공간 낭비 문제점을 그대로 갖고 있다. 분리된 필드마다 별도의 비트 데이터 타입을 이용하여 저장하더라도, 기존의 사용데이터베이스는 8bit 미만의 비트 데이터 타입에 대해서도 최소 1byte(8bit)공간을 할애한다. 따라서 생월 필드의 경우 4bit 공간이면 월을 나타내는 01~12값을 충분히 저장하지만, 기존 시스템은 8bit를 사용한다. 또한, 생일 필드도 5bit면 날짜를 나타내는 01~31을 충분히 저장할 수 있지만, 8bit를 사용한다.

둘째, 분리된 필드를 통합하여 한 번에 비교하는 연산이 불편하다. 즉, 생년월일을 한번에 비교하는 경우도 많이 발생하는데, 이럴 경우 세 번의 비교 연산을 해야 한다. 즉, 생년, 생월, 생일 필드를 birth-year, birth-month, birthday라 하면 birth-year1=birth-year2, birth-month1=birth-month2, birthday1=birthday2와 같이 세 번의 비교를 해야 한다. 따라서 생년월일을 연속적으로 붙여서 한 번에 가리키는 통합된 필드 명이 있으면 더욱 편리할 것으로 사료된다. 즉, 생년월일을 비트단위로 붙여서 만든 필드를 birthdate 이라 부르면 birthdate1 = birthdate2 형태로 한 번에 비교할 수 있다.

따라서 이러한 단점들을 극복하여 사용자가 최소한의 공간 사용과 효율적인 검색, 그리고 분리된 필드단위의 연산과 통합된 필드단위의 연산이 모두 자유로운 방법이 필요하다.

3. 결합 비트의 데이터 타입 및 구현 방법

3.1. 결합비트(combined bit)의 정의

3.1.1. 비트 데이터 타입의 분류 및 정의

기존의 비트 데이터 타입을 비트의 수에 따라 다음과 같이 분류하고 이를 다음과 같이 부르기로 한다.
(1) 비트의 단수/복수에 의한 구분은 표 1과 같다.

표 1. 비트의 단수/복수에 의한 구분

구 분	정 의
단일비트(Single-Bit)	한 비트로 정의된 데이터 타입
다중비트(Multi-Bit)	두 개 이상의 비트로 정의된 데이터 타입

(2) 비트 수의 바이트(byte)(8bit) 초과 여부에 의한 구분은 표 2 와 같다

표 2. 비트 수의 초과 여부에 의한 구분

구 분	정 의
소형비트(Small-Bit)	8비트 이하의 비트들로 구성된 데이터 타입
대형비트(Large-Bit)	8비트를 초과하는 비트들로 구성된 데이터 타입

(3) 비트 데이터의 분류 예제
비트 데이터 타입은 특별한 문법의 변화 없이 단지 bit 다음 괄호 안의 숫자에 의해서 결정된다. 그 예는 표 3 과 같다.

표 3. 비트 데이터의 분류 예

비트 데이터의 예	분 류
bit(1)	bit와 동일, 단일비트, 소형비트
bit(3)	다중비트, 소형비트
bit(12)	다중비트, 대형비트

소형 비트는 실제로 종류가 많지 않은 값들을 코드화하여 사용할 경우에 흔히 발생한다. 예를 들면, 생월의 경우 01~12의 12개의 값으로 4 bit의 소형비트로 저장할 수 있고, 생일의 경우 01~31의 3가지 값으로 5 bit의 소형비트로 저장할 수 있으며, 성별은 0~1의 2개의 값으로 1 bit의 단일비트로 저장할 수 있다.

3.1.2. 결합 비트(combined bit) 데이터 타입의 정의

소형비트의 타입의 필드들은 기존의 데이터베이스에서는 비트 수와 관계없이 최소 한 바이트의 공간을 차지하였다. 이것은 저장 시스템에서 다루는 데이터 단위가 바이트이기 때문이다. 또한, 대형비트 타입의 필드도 8의 배수만큼의 비트를 할당해야 하므로, 나머지 공간을 낭비한다. 예를 들면, bit(11)은 16비트를 할당해야 하므로, 5 비트만큼 낭비한다. 소형비트 타입의 필드들이 한 테이블에 다수 개 있을 경우, 기존의 저장 시스템의 데이터 단위를 변경하지 않고, 단지 이들을 합쳐서 저장하는 방법을 통하여 저장 공간의 절약과 검색의 효율화를 달성하는 방법으로 결합 비트(combined bit)데이터 타입을 정의하고 저장하는 방법을 제시한다.

정의 : 결합 비트(combined bit) 데이터 타입
연속적으로 저장된 비트 데이터 타입들을 결합한 데이터 타입으로 정의한다.

3.1.3. 결합 비트 데이터 타입을 이용한 스키마 정의

결합 비트는 데이터 combine이라는 예약어를 사용하 결합된 데이터 타입을 정의한다. 즉, 그림 2 에서는 res_no는 3개의 bit 데이터 타입을 결합한 데이터 타입의 res_no 필드를 정의하고 있다. 본 발명에서는 결합 필드를 가리키는 res_no와 같은 필드를 결합 비트 라고 부른다. combine { } 에는 bit 데이터 타입만이 사용될 수 있다. 그리고 결합 비트를 구성하는 개별 필드를 구성

비트라고 부른다.

```
CREATE TABLE person {
  combine {
    birth_year bit(7),
    birth_month bit4),
    birth_day bit(5)
  } res_no,
  name char(10),
  phone_no char(11)
}
```

그림 2. 결합 비트의 타입 정의

3.2. 결합비트(combined bit)의 검색 방법

결합 비트로 정의된 데이터는 통합된 형태와 분리된 형태로 각각 사용될 수 있다. 첫째로, 통합된 형태는 결합 필드 명을 사용하여 접근되며, 비트 데이터도 모드 연속되어 붙어서 활용된다. 예를 들면, 그림 2의 person 테이블에서 아래와 같이 res_no 전체를 접근하는 질의가 가능하다. 그림 3 질의는 person 테이블에서 생년월일이 64년 4월 4일을 갖는 사람의 주민번호와 이름을 출력한다.

```
SELECT res_no, name FROM person
 WHERE  res_no = '10000000 0100 00100'
```

그림 3. 결합 필드의 통합 검색

결합 비트를 구성하는 개별적인 비트 데이터 타입은 그림 4 와 같이 접근 할 수 있다. 그림 4는 생월이 4월인 사람의 사람의 생년과 이름을 출력한다.

```
SELECT birth_year, name FROM person
WHERE birth_month = '0100'
```

그림 4. 결합 필드의 개별 검색

3.3. 결합 비트(combined bit)의 저장 방법

결합 비트 데이터 타입을 갖는 데이터는 결합된 상태로 통합되어 한 번에 저장된다. 즉, 기존의 바이트 단위로 운영되는 저장 시스템을 수정하지 않도록, 결합 비트 안의 비트 데이터를 순서대로 모두 결합할 경우에 발생하는 비트 수보다 큰 최소한의 8의 배수가 할당된다. 이것은 바이트 단위로 할당되도록 하기 위함이다. 즉, 그림 2의 person의 res_no의 경우 내부 비트의 합이 16bit이므로 8의 배수이다. 이런 경우에는 별도의 패드(pad) 비트 없이 그대로 res_no를 저장한다. 만일 8의 배수가 아니라면, 예를 들어 res_no의 비트의 합이 14비트라면 그림 5와 같이 별도의 패드 비트를 추가하여 8의 배수로 만들어 준다.

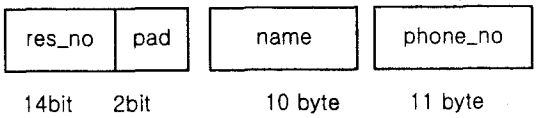

그림 5. 결합 비트의 저장

3.4. 결합 비트(combined bit)의 인덱스 설계 방법

결합 비트 데이터타입은 그림 3과 같이 결합 비트 전체를 이용하여 검색 될 수도 있고, 결합 비트를 구성하는 구성필드를 이용하는 검색도 가능하다. 따라서 각각의 경우에 모두 필요한 검색을 효율적으로 지원하기 위한 인덱스 설계 방법이 필요하다.

(1) 결합 비트 전체를 이용하는 검색을 위한 인덱스

결합 비트 전체를 이용하는 검색을 결합키 검색(combined key search)라고 부른다. 이 경우에는 결합 비트 전체를 하나의 키 값으로 사용하는 일반적인 경우와 다를 바가 없다. 따라서 이런 경우에는 일반적인 인덱스인 B-Tree 나 B+Tree를 이용할 수 있다.

(2) 구성 비트(component bit)를 위한 인덱스

결합 비트를 구성하는 구성 비트를 이용하는 검색을 우리는 구성키 검색(component key search)라고 부른다. 구성키 검색은 구성키 하나를 이용하는 단일 구성키 검색(single component key search)과 여러 개의 구성키를 이용하는 복합 구성키 검색(multi component key search)로 나누어진다.

① 단일 구성키 검색

하나의 구성키를 이용한다면, 키 값의 종류가 많지 않으므로 동일 키 값을 갖는 레코드가 많을 수 있다. 이 경우 그림 6과 같은 array index형태를 사용하여 구현하는 것도 가능하다.

② 복합 구성키 검색

결합 비트를 이용하는 구성 비트의 임의 조합을 이용하는 경우를 말한다. 예를 들면 주민번호의 예에서 (생년, 생일)을 이용한 검색, (생월, 생일)을 이용한 검색 등을 들 수 있다. 이러한 경우, 두 개의 필드를 연결한 값을 키 값으로 하는 인덱스를 생성해도 좋으나 이런 인덱스는 키 값을 구성하는 순서에 따라 성능이 크게 좌우된다. 따라서 이런 경우에는 MLGF(multi-level grid file)이나 R-Tree와 같은 다차원 인덱스를 이용하는 것이 효율적이다

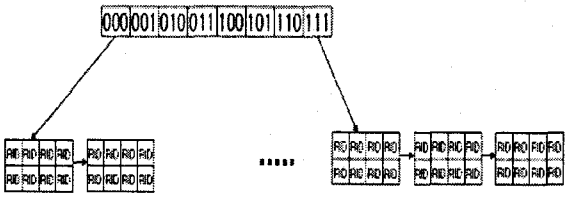

그림 6. 단일 구성키 검색을 위한 array index

4. 결 론

기존의 상용 데이터베이스 관리 시스템(DBMS)에 제안한 결합 비트를 적용하게 될 경우 다음과 같은 효과를 얻을 수 있을 것으로 사료된다.

첫째, 저장 공간 절약이 있다. 바이트 단위로 저장되는 기존 시스템을 이용하더도 저장 공간의 절약이 된다. 즉 비트 단위의 필드들을 묶어서 필드 하나마다 바이트 단위로 만들면서 낭비하던 공간을 비트 타입 필드 전체의 합을 바이트 단위로 만들기 위한 공간만 소모하면 된다. 예를 들면, 위에서 생년(7bit)에서 1bit가 남고, 생월(4bit)에서 4bit가 남고, 생일(5bit)에서 3bit가 남는 비트이다. 남는 bit는 8bit 이고 결합 비트를 사용하여 16bit로 표현이 가능하다. 따라서 1,000,000개의 레코드가 저장될 경우, 1MB 저장 공간이 절약된다.

둘째, 검색 속도 향상의 이점이 있다. 결합 비트 전체 검색의 경우, 결합 비트 전체에 대한 인덱스가 있으므로, 구성 비트별로 생성된 인덱스들을 이용하는 경우보다 효율적인 검색을 할 수 있다. 앞의 예의 경우처럼 생년월일처럼 결합 비트로 구성되는 필드들은 통합되어 검색되는 경우가 많은 필드이다. 구성비트의 복합 검색의 경우 R-Tree난 MLGF와 같은 다차원 인덱스를 도입하면, 임의의 구성키의 조합을 이용한 검색도 효율적으로 제공할 수 있다.

셋째, 결합 비트의 개념적인 도입으로 인한 질의의 편리성을 들 수 있다. 예를 들어 " 생년월일" 전체를 이용한 검색과 생년, 생월, 생일에서 임의의 조합을 이용한 검색을 편리하게 사용할 수 있으므로, 질의를 편리하게 할 수 있다. 기존의 구성 필드를 보면 생년월일을 통합하여 저장하였다면, 복합 구성키 검색을 위해 각각의 필드를 분리하여 검색해야 하므로, 검색이 효율적으로 이루어질 수 없다. 반대로 생년월일을 모두 분리하여 별도의 필드로 저장했다면, 각각의 필드마다 생성된 인덱스를 사용하더라도 효율적인 검색을 하기 쉽지 않다. 따라서 결합 비트 데이터 타입과 같이 연결된 전체 비트를 한 번에 비교할 수 있는 방법과 개별 필드를 각각 접근할 수 있는 방법이 모두 필요하다.

5. 참고문헌

[1] T. Imielinski, S. Viswanathan, and B.R. Badrinath, " Data on Air: Oragnization and Access," IEEE Trans. On Knowledge and Data Engineering, Vol.9, No. 3, May/June, 1997.

[2] Tacha Serif, George Ghinea, "versus PDA: a comparative study of the user out-of-box experience," Personal and Ubiquitous Computing Vol. 9. No. 4, pp. 238~249, 2005.

[3] Thorstein Lunde, Arve Larsen, "KISS the Tram: Exploring the PDA as Support for Everyday Activities," in Proc. of Ubicomp 2001, pp. 232~239, 2001.