제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

리스트 컴프리헨션을 이용한 데이터베이스 접근

박경순, 우 균
동아대학교 컴퓨터공학과
e-mail : inim@nownuri.net, woogyun@daunet.donga.ac.kr

Access to Database Using List Comprehension

Kyung-soon Park, Gyun Woo
Dept. of Computer Engineering, Dong-a University

요 약

기존의 데이터베이스 응용 프로그램을 작성할 경우에 질의 언어와 개발 언어간의 이중 언어 정의 문제(impedance mismatch)가 발생한다. 이를 해결하기 위한 한가지 방법으로 본 논문에서는 Haskell의 리스트 컴프리헨션(list comprehension)을 이용한 데이터베이스 접근을 제안한다. 이 방식을 이용할 경우 문법 체계가 명확해 진다는 장점과 연속적인 집계연산자(aggregate operator)를 사용할 수 있다는 장점이 있다. 또한 앞에서 지적한 이중 언어 정의 문제를 해결할 수 있다.

1. 서론

데이터베이스를 이용하는 응용 프로그램을 작성할 경우, 그 데이터베이스에 접근하여 정보를 얻기 위해 데이터베이스 조작을 위한 질의어(SQL)을 사용하게 된다. 이 때, 질의어 SQL은 응용 프로그램을 위한 언어가 아니므로 이중적인 언어 구조를 가질 수 밖에 없다. 이러한 구조는 개발자에게 두 가지 언어 습득을 요구하며, 또한 질의어 SQL을 이용할 경우 복잡한 데이터 모델링의 경우 한계를 가진다[1].

본 논문은 순수 지연 함수형 언어의 표준인 Haskell을 질의어로 활용함으로써 기존 질의어인 SQL의 단점을 보완하고 개발언어와 질의어를 하나로 통합하는 방법을 제시하고자 한다. 다시 말해 SQL이 아닌 범용 프로그래밍 언어인 Haskell을 이용한 데이터베이스 접근을 시도하고자 한다.

Haskell은 이미 많은 응용 프로그램에서 개발언어로 채택되고 있으므로, 응용 프로그램 개발언어로서 충분히 활용 가능하다[2]. 또한 데이터베이스 질의문으로서의 Haskell에 대해서도 몇몇의 논문들[3, 4]에서 그 가능성을 언급하고 있다.

Peter Bunemann 등이 저술한 논문[3]에서는 컴프리헨션 문법으로 이루어진 언어와 SQL과의 대응 관계를 기술하고 있다. 특히 컴프리헨션 문장의 타입에 대해 집합, 다중집합(multiset) 그리고 리스트로 구분짓고, 데이터베이스의 레코드셋의 타입과의 타입호환에 관해 기술하고 있다. 하지만 자체적인 컴프리헨션 문법을 정의하여 쓰고 있으므로 이를 Haskell 문법에 적용시키기엔 무리가 있다.

Daan Leijen과 Erik Meijer의 논문[4]에서는 특화 언어(domain-specific embedded language)로서의 Haskell의 가능성을 타진하고 있다. 이 논문의 가장 큰 장점은 COM을 이용해 Haskell에서 데이터베이스 시스템에 직접 접근 가능하도록 한 것이다. 이 논문에서도 역시 Haskell의 컴프리헨션 기법과 유사한 방식을 채택하고 있으나 그 사용법이 다소 복잡하다.

따라서 본 논문에서는 Haskell의 리스트 컴프리헨션을 이용한 데이터베이스 접근 방안을 제안하고, 이 방법의 구현 가능성을 제시한다. 이를 통해 객체지향 데이터베이스의 질의 언어로 Haskell이 효과적으로 사용될 수 있음을 보인다.

본 논문의 구조는 다음과 같다. 먼저 2절에서 기존 SQL과 함수형 언어의 리스트 컴프리헨션을 비교하고, 3절에서 Haskell을 이용한 질의문 설계에 대하여 기술한다. 마지막으로 4절에서는 결론 및 향후 과제를 논한다.

2. SQL과 리스트 컴프리헨션의 비교

2.1 SQL을 통한 데이터베이스 접근

SQL은 관계형 데이터베이스의 표준 프로그래밍언어이다[5]. 일반적으로 SQL 질의는 그림 1과 같은 형태로 이루어져 있다.

```
SELECT    Columns
FROM      Tables
WHERE     Criteria
```

<그림 1> SQL의 일반적인 형태

SELECT 구문은 특정 컬럼을 선택하는 절이고, FROM절은 특정 테이블을 선택하며, WHERE 절은 특정 속성에 대한 조건을 기술한다. 따라서 다음의 각 관계 대수는 그림 2와 같은 SQL문으로 변경된다.

```
Supply WHERE City = "London"
SELECT *
FROM   Supply
WHERE  City = "London"

(Supply WHERE SNO = "S3") { SNAME }
SELECT SNAME
FROM   SUPPLY
WHERE  SNO = S3

(Supply JOIN Product) { SNAME, PNAME }
SELECT SNAME, PNAME
FROM   Supply as r, Product as s
WHERE  r.CITY = s.CITY
```

<그림 2> 관계 대수를 SQL로 변경한 예제

2.2 Haskell에 대한 소개

Haskell은 다형성 타입과 지연 계산(lazy evaluation)을 지원하는 순수 함수형 컴퓨터 프로그래밍 언어이다[2]. 여기서 지연 계산은 다른 말로 필요에 의한 호출(call-by-need)방법을 말한다. Haskell의 다른 특징으로는 사용자 정의 타입을 지원하며, 자체적인 클래스 개념을 가지고 있고, 무엇보다 강력한 타입 검사를 한다. 이로 인하여 복잡한 데이터를 다룰 때나 사용자 정의 데이터를 조작할 때 의미 보존을 명확하게 해주며, 타입 오류를 컴파일 시간에 식별할 수 있게 된다.

사용자 정의 타입은 data 선언문을 통해 작성한다. 그림 3과 같은 예제는 모두 사용자 정의 타입들이다.

```
data Bool     = False  |  True
data Color    = Red | Green | Blue
```

<그림 3> 사용자 정의 타입의 예

또한 타입 동의어(type synonyms)에 의해서 특정 타입에 대한 이름을 정의할 수도 있다.

```
type String   = [Char]
type Person   = (Name, Address)
type Name     = String
data Address  = None | Addr String
```

<그림 4> 타입동의어의 예

2.3 리스트 컴프리헨션을 이용한 데이터베이스 접근

Haskell에서 리스트는 어떤 함수의 데이터를 다른 함수로 전달하는데 쓰인다. 리스트의 표기는 각괄호([])와 쉼표를 이용한다. 리스트의 모든 구성 원소들은 같은 타입을 가져야 한다. 그 원소들의 구성은 집합과는 달리 같은 값을 하나 이상 가질 수 있다. 리스트의 또 다른 특징으로는 순서를 가진다. 따라서 두 리스트가 동일한 경우는 똑같은 값을 똑 같은 순서로 가질 경우 뿐이다.

리스트 컴프리헨션은 [e | c_1, ... , c_n]과 같은 형태를 취하고 있다. 각각의 한정자(quantifier) c_i는 생성자(generator)이거나 조건(condition)을 나타낸다. 생성자는 p <- s와 같은 형태이다. 조건식에는 불린(boolean)값을 가지는 임의의 식이 올 수 있으며, 일반적으로 특정 값만 걸러내는 역할을 담당하게 된다. 이와 같은 형태로 리스트 컴프리헨션을 쓰면 한정자들이 계산되고 그에 따른 원소들이 리스트로 반환된다.

이제 데이터베이스에 저장된 데이터들에 대해 생각해 보자. 데이터베이스의 데이터들은 튜플(tuple) 형태의 집합으로 볼 수 있다. 보다 정확하게 정의를 하자면, 중복을 허용하는 집합형태, 즉 다중집합 형태이다. 따라서 리스트 컴프리헨션을 데이터베이스에 적용하기 위해서는 리스트를 다중집합으로, 다중집합을 리스트 형태로 변경할 수 있도록 해주는 방법이 필요하다. 하지만, 다중집합과 리스트의 차이는 순서의 유무를 제외하곤 똑같다는 것을 알 수 있다. 따라서 본 논문에서는 다중집합의 구현 방식을 리스트 형태로 간주한다.

앞 선 관계형 대수들은 각각 그림 5와 같이 리스트 컴프리헨션으로 작성할 수 있다.

```
Supply WHERE City = "London"
[r |
       r <- openTable supply,
       r ! city = "London"]

(Supply WHERE SNO = "S3") { SNAME }
[r ! sname |
       r <- openTable supply,
       r ! sno = S3]

(Supply JOIN Product) { SNAME, PNAME }
[(r ! sname, s ! pname) |
       r <- openTable supply,
       s <- openTable product,
       r ! city = s ! city]
```

<그림 5> 리스트 컴프리헨션을 이용한 예제

2.4 SQL과 리스트 컴프리헨션의 비교

두 방식을 살펴보면 똑같은 질의문에 대하여 아주 유사한 형태로 접근함을 알 수 있다. 다시 말하자면, 어떤 테이블에서 어떤 조건을 가지는 데이터를 가지고 올 것인가에 중점을 둔 식임을 알 수 있다. 테이블 이름에 대해 명확하게 별칭을 둘 것인가의 차이점을 제외하고는 유사하다. 덧붙여 리스트 컴프리헨션을 이용할 경우, Haskell의 강력한 타입 검사 기능을 통해 타입 오류를 검출해 낼 수 있다. 이를 위해서는 데이터베이스에 저장된 데이터 타입과 Haskell의 타입 사이의 대응관계를 구성할 수 있어야 한다.

3. Haskell을 이용한 데이터베이스 접근

본 논문에서 제시하고 있는 데이터 접근은 크게 2부분으로 나눌 수 있다. 하나는 데이터베이스에서 정보를 추출하여 인터페이스를 맞추는 부분이고, 다른 하나는 리스트 컴프리헨션을 이용하여 데이터베이스에 접근하는 방법이다.

3.1 데이터베이스 인터페이스

데이터베이스 시스템의 접근 방식은 몇 가지가 있다. 그 중 본 논문에서는 ADO를 이용한 데이터베이스 접근을 시도할 것이다. 이를 이용한 접근 방식은 Daan Leijen과 Erik Meijer[4]에 의해서도 채택되었는데, 이 방식은 ODBC호환 가능한 데이터베이스들을 사용가능하며 COM 프레임워크에 기반하여 동작한다는 특징을 가지고 있다.

다음으로 데이터베이스 내의 특정 테이블을 검색하기 위해서는 그 데이터베이스의 구성정보, 즉 데이터베이스 스키마를 호스트 언어인 Haskell에 전달해 주어야 한다. 특히 SQL을 이용하지 않으므로 그 테이블의 구조가 Haskell의 문법에 맞게 정의되어야 한다. 따라서 특정 데이터베이스의 구조를 Haskell문법에 맞게 변환해주는 모듈(Trans 모듈)이 필요하다.

이 모듈은 3개의 인자를 가지는데, 그 내용은 각각 데이터베이스 이름, 출력파일이름, ADO 접속문자열이다. 즉 다음과 같은 형태로 데이터베이스를 읽어와서 Haskell 모듈로 변경한다.

```
main = adoRun
  $ do
    { args <- getArgs
    ; case (args) of
      [dbName, fileName, opts]
        -> do
    {createCatalog dbName fileName opts
    ; putStr "done!\n"
    }
  -> showHelp
}
```

<그림 6> Trans 모듈의 실행부분

Haskell모듈로 변경하는 과정에서 데이터베이스의 타입과 Haskell의 타입 호환을 위하여 추가적인 작업이 필요하다. 단순히 모든 값들을 문자열로 인식하는 방법도 생각해 볼 수 있으나, 자주 쓰이는 타입들의 경우 명시적으로 타입을 선언해 주는 방법이 더 효율적이다. 따라서 그림 7과 같은 형태로 타입 매핑을 정의한다.

```
typeMap :: [(Int,String)]
typeMap
 =      [ (48,"Int")    --tinyint
        , (50,"Bool")   --bit
        , (52,"Int")    --smallint
        , (55,"Integer") --decimal
        , (56,"Int")     --int
        , (58,"Double") --smalldatetime
        , (59,"Float")  --real
        , (60,"Integer") --money
        , (61,"String") --datetime
        , (62,"Double") --float
        , (63,"Integer")  --numeric
        , (122,"Integer") --smallmoney
        -- rest is "String"
        ]
```

<그림 7> 데이터베이스와 Haskell의 타입 호환

3.2 리스트 컴프리헨션을 이용한 데이터베이스 접근

앞선 그림 3의 2번째 예제를 다시 살펴보자

```
(Supply WHERE SNO = "S3") { SNAME }
[r ! sname | r <- openTable supply,
     r ! sno = S3]
```

이 예제에서 볼 수 있듯이 데이터베이스에 접근하기 위해 특정 테이블을 연다. 이렇게 함으로써 테이블을 사용할 것이라는 것을 명시하게 된다. 이 테이블은 Haskell의 특징인 지연 계산에 의해 실제 사용 시점에서 튜플로 이루어진 리스트로 변경된다.

리스트 컴프리헨션은 선택된 테이블에서 특정 데이터를 걸러내는 역할을 하게 된다. 생성자는 그 테이블에서 특정 속성만 걸러내는 역할을 하게 된다. 여기에서 '!' 연산자는 객체지향언어에서 객체의 내부필드접근에 쓰이는 '.' 와 비슷한 역할을 한다. 연산자 '.'을 그대로 사용할 수 없는 이유는 Haskell에서 '.' 연산자가 함수 합성을 하는 연산자로 미리 정의되어 있기 때문이다. 이 연산자는 데이터베이스의 테이블을 받아서 특정 컬럼만 선택하는 역할을 한다.

다음으로 하나 이상의 속성이 구조체 형식으로 되어있는 테이블을 생각해 보자. 이 예로 표 1과 같은 직원 테이블을 가정하자.

Name		Dnum	Status		Project
First	Last		Salary	Extension	

<표 1> Emp 테이블의 구조

이 테이블은 총 4개의 속성으로 이루어져 있다. 이

름과, 부서번호, 직원의 지위, 수행중인 프로젝트 이름으로 이루어져 있다. 이 중, 성명은 성과 이름으로 이루어져 있으며, 지위는 그 직원의 연봉과 내선번호로 이루어져 있다.

예를 들어 이 테이블에서 델타 프로젝트를 수행하고 있는 직원들 중에서 연봉이 1만 달러가 넘는 사원들의 이름을 검색하고 싶다면 그림 8과 같은 형태로 이루어진다.

```
[r ! name |    r <- e openTable emp,
       r ! status ! salary >= 10000,
       r ! project = "Delta" ]
```

<그림 8> 구조를 가지는 속성에 대한 접근 예

또 다음과 같은 상황을 가정해 보자. 어떤 데이터베이스의 테이블이 있는데 특정 속성의 값들의 총합에 대하여 평균을 낸다고 가정해보자. 예를 들어 아래와 같은 테이블에서 S#별 수량의 평균을 구한다고 생각해보자.

S#	P#	QTY
S1	P1	200
S1	P2	300
S1	P3	300
S2	P2	400
S3	P1	200
S3	P3	100

<표 2> SP 테이블

이 때 SQL의 경우는 연속적인 집계 연산자를 사용할 수 없으므로 아래와 같은 방식으로만 접근 가능하다.

```
SELECT   AVG (X)
FROM     (SELECT   SUM (SP.QTY) AS X
          FROM     SP
          GROUP BY SP.S#) AS POINTLESS
```

<그림 9> SQL을 이용한 집계연산

하지만, 리스트 컴프리헨션을 이용할 경우 집계 연산자(aggregate operator)를 중첩해서 쓸 수 있으므로, 간단히 표기할 수 있다. 따라서 그림 10과 같은 접근이 가능해진다.

```
[listAVG sum (r ! qty) |
              r <- openTable sp,
              group_by r.s#]
```

<그림 10> 리스트 컴프리헨션을 이용한 집계연산

4. 결론 및 향후 과제

2장과 3장을 통해 Haskell의 리스트 컴프리헨션을 이용한 데이터베이스 접근에 대하여 살펴보았다. 본 논문에서는 Haskell의 리스트 컴프리헨션을 이용하면 보다 간결한 문장만으로 데이터베이스에 접근할 수 있음을 보였다. 즉, 질의문의 간결성 및 명확성이 높아지는 효과를 얻을 수 있다. 그리고 완전한 데이터베이스 응용프로그램을 작성하기 위해서 데이터베이스 질의어와 프로그램 개발언어를 하나의 언어로 사용할 수 있음을 보였다.

본 논문에서는 리스트 컴프리헨션과 SQL의 호환가능성만을 보였다. 그러나 보다 완전한 호환을 위해서는 문법 변환 규칙을 완성해야 한다. 또 변환 규칙 완성한 후에는 기존 관계형 데이터베이스와의 호환을 다시 한번 더 검증할 필요가 있는데, 이는 향후 연구 과제로 남겨둔다.

참고문헌

[1] Adjeroh, D.A. and Nwose, KC, "Multimedia Database Management-Requirements and Issues," *IEEE Multimedia*, Vol.4, No.3, IEEE CS, 1997

[2] Simon Marlow, "Writing High-Performance Server Applications in Haskell, Case Study : A Haskell Web Server," *In Proceedings of Haskell Workshop*, Montreal, Canada, 2000,

[3] Peter Buneman, Leonid Libkin, Dan Suciu et al. "Comprehension Syntax," *SIGMOD Record*, 1994

[4] Daan Leijen and Erik Meijer, "Domain Specific Embedded Compilers", *In Proceedings 2nd USENIX Conference on Domain-Specific Languages*, 1999

[5] C.J. Date, *An Introduction to DATABASE SYSTEMS*, Addison-Wesley, 2000

[6] Simon Peyton Jones(Editor) et al. , *Report on the Programming Language Haskell98 A Non-strict, Purely Functional Language*, 1999

[7] Paul Hudak and John Peterson, "A Gentle Introduction to Haskell98", 1999

[8] www.haskell.org