투플 관계해석

관계 해석

- "어떻게 검색할 것인가 (relational algebra)" 보다 "무엇을 검색할 것인가" 만을 기술하는 선언적 표현법 (Declarative)을 사용하는 비절차적 질의어
- SQL을 포함한 많은 상업용 관계 언어들이 관계 해석에 기반을 두고 있음.
- 투플 관계 해석 (tuple relational calculus)와 ~~도메인 관계 해석~~ (domain relational calculus)으로 구분됨.

관계 대수와의 차이점

- 관계 해석은 하나의 선언적 (declarative) 해석식으로 검색 질의를 명시하며 비절차적 언어
- 관계 대수 (8가지 연산)에서는 연산들을 순차적으로 사용하므로 절차적인 성질을 가짐
- 두 언어의 표현력 (expressive power)은 동등.

SQL 문장을 받으면 Query Tree (관계 대수식)를 만든다. 관계 해석은 관계 대수 연산자로 바꿀 수 있다.

관계적 완전성 (relationally completeness)

- 어떤 관계 질의어 L이 관계 해석 또는 관계 대수로 표현 가능한 어떤 질의도 표현할 수 있으면 L은 "관계적으로 완전 (relationally complete) 하다."라고 한다.
- 대부분의 관계 질의어들은 관계적으로 완전하며, 집단 함수 (aggregate functions), 그룹화 (grouping), 순서화(ordering) 등의 연산들을 제공하므로 관계 해석보다 표현력이 강해진다.

투플 관계해석의 표현과 식

투플 관계해석의 일반식 형태

{t1.A1, t2.A2, ... , tn.An | COND(t1, t2, ... , tn, tn+1, tn+2, ... , tn+m})

- n개는 자유변수, m개는 속박변수

존재 정량자와 전체 정량자

- 정량자 (quantifiers)가 식에 사용될 수 있음
- 전체 정량자 (universal quantifier) (∀) (for all)
- 존재 정량자 (existential quantifier) (∃) (there exists)
- 자유 (free) 투플 변수와 속박 (bound) 투플 변수
- 어떤 식 F가 원자인 경우, 투플 변수의 어커런스(occurrence)는 F에서 자유롭다 (자유투플 변수)
- 식 (F1 and F2), (F1 or F2), not(F1), not(F2)에 나타난 투플 변수 t가 자유로운가 여부는 F1 이나 F2에서 자유로운가에 달려있다.
- F내의 투플 변수 t의 모든 자유 어커런스들은 F' = (∃t)(F)나 F'=(∀t)(F) 형태의 식에서 정량자에 속박.

∃x (x>1) x∈{1,2,3,4} : True (조건에 만족하는 x는 2,3,4가 있음. x는 속박변수)

∀x (x>1) x∈{1,2,3,4} : False (조건에 만족하지 않은 x가 1이 있기 때문.)

정량자가 있으면 속박변수, 그렇지 않으면 자유변수

자유변수가 있으면 명제가 아닌 질의(query)이다.

Example)

F1 : d.Dname = 'Research'

F2 : (∃t)(d.Dnumber = t.DNO)

d는 F1과 F2에서 자유롭다.

t는 F2에서 ∃에 속박.

자유변수는 질의

{y|∀x x∈{1,2,3,4}} => y는 자유(free)변수.

{y|∀x (x<y) x, y∈{1,2,3,4}} => y는 공집합

{y|∀x (x≤y) x, y∈{1,2,3,4}} => y는 최댓값

{y|∀x (x≥y) x, y∈{1,2,3,4}} => y는 최솟값

∃x∃y (x, y∈{1,2,3,4}, x≥y) => True

∃x∀y (x, y∈{1,2,3,4}, x≥y) => True. 최댓값이 반드시 존재하는지

∃x∀y (x, y∈{1,2,3,4}, x≤y) => True. 최솟값이 반드시 존재하는지

프로젝션의 표현

Example)

Employee에 남자인 이름을 찾으시오

- {t.fname, t.lname | Employee(t) and t.sex='M'}

Reasearch 부서에 속하는 직원 lname

- {e.lname | Employee(e) and ∃d(Department(d) and d.dname='Research' and e.dno=d.dnumber) }
- Select e.lname from Employee e where EXISTS (select * from Department d where d.name='Research' and e.dno=d.dnumber)

모두가 남자로만 구성되어 있는 부서

- {d.dname | Department(d) and ∀e(not ∃(e) or not(d.dnumber=e.dno) or e.sex='M')}

최대 급여를 받는 직원의 lname

- {e.lname | Employee(e) and ∀t(not ∃(t) or e.salary ≥ t.salary)}

부서별 최대 급여를 받는 직원의 lname

- {e.lname | Employee(e) and ∀t(not ∃(t) or not(e.dno = t.dno) or e.salary ≥ t.salary)}

전체 정량자와 존재 정량자 사이의 변환

- (∀ x) (P(x)) ≡ not(∃ x) (not(P(x)))
- (∃ x) (P(x)) ≡ not(∀ x) (not(P(x)))
- (∀ x) (P(x) and Q(x)) ≡ not(∃ x) (not(P(x)) or not(Q(x)))

- (∀ x) (P(x) or Q(x)) ≡ not(∃ x) (not(P(x)) and not(Q(x)))
- (∃ x) (P(x) or Q(x)) ≡ not(∀ x) (not(P(x)) and not(Q(x)))
- (∃ x) (P(x) and Q(x)) ≡ not(∀ x) (not(P(x)) or not(Q(x)))

최대 급여를 받는 직원의 lname

- {e.lname | Employee(e) and not ∃(t) (∃(t) and e.dno=t.dno and e.salary < t.salary) }

(∀x) (P(x)) ⇒ (∃x) (P(x)) True 연역(deduction)

(∃x) (P(x)) ⇒ (∀x) (P(x)) False

not(∃x) (P(x)) ⇒ not(∀x) (P(x)) True

not(∀x) (P(x)) ⇒ not(∃x) (P(x)) False

5번 부서에 의해 관리되는 모든 프로젝트들에 참여하는 사원들의 이름을 찾아라

- {e.lname, e.fname | Employee(e) and ((∀x) (not Project(x)) or (not x.dnum=5) or (∃x works_on(w) and w.ESSN=e.SSN and x.PNUMBER=w.PNO)}
- 전체 정량자로부터 존재 정량자로의 변환
- {e.lname, e.fname | Employee(e) and (not ∃x (Project(x)) and (x.dnum=5) or not(∃x works_on(w) and w.ESSN=e.SSN and x.PNUMBER=w.PNO)}

부양가족이 없는 사원들의 이름을 찾아라

- {e.fname, e.lname | Employee(e) and not ∃d (Dependent(d) and e.SSN = d.ESSN) }
- {e.fname, e.lname | Employee(e) and not ∀d (not Dependent(d) or not(e.SSN = d.ESSN)) }

부양가족이 적어도 한 명 있는 관리자들의 이름을 나열하라

- {e.fname, e.lname | Employee(e) and ∃d ∃p (Department(d) and Dependent(p) and e.SSN = d.mgrssn and p.essn = e.ssn)}

안전식 (safe expression)

- 결과로서 유한개의 투플들을 생성하는 것이 보장된 식
- 불안전식은 무한개의 투플들을 생성할 수 있고, 투플들의 타입이 서로 다를 수 있음
- 불완전한 식의 예제 {t | not(EMPLOYEE(t)}
- 가능한 모든 투플들 중에서 EMPLOYEE가 아닌 모든 투플들을 생성함
- 이러한 투플들은 무한개의 투플들로 구성됨