DB HW 6

7조

2014-16757 김보창, 2014-16615 박은천, 2015-15448 송영우, 2017-16064 강태호

1.

a) )

b) )

c) )

d)

2.

a) )

b)

c)

d)

3. 모든 쿼리는 multi-set이 아닌 버전으로 작성하였음.

(중복을 허용하지 않음)

a)

select \* from r1

union

select \* from r2;

b)

select \* from r1

intersect

select \* from r2;

c)

select \* from r1

except

select \* from r2;

d)

select distinct A, B, C

from (select distinct r1.A, r1.B from r1) as r3 natural join (select distinct r2.B, r2.C from r2) as r4

4. company-name attribute가 varchar(30)으로 저장되어 있다고 가정.

a)

create function four\_func() returns table(company-name varchar(30))

return table

(select company-name

from works

group by company-name

having avg(salary) > (select avg(salary)

from works

where company-name = "First Bank Corporation"));

select \*

from table (four\_func());

b)

select company-name

from works

group by company-name

having avg(salary) > (select avg(salary)

from works

where company-name = "First Bank Corporation");

5.

a)

tuple-relational

{t | ∃s∈ employee(t[person\_name] = s[person\_name] ∧

∃u ∈ works( u[person\_name] = s[person\_name] ∧

u[company\_name] = “First Bank Corporation”) ) }

domain-relational

{<person\_name> | ∃ street, city (<person\_name, street, city> ∈ employee ∧

∃ company\_name, salary(<person\_name, company\_name, salary> ∈ works ∧

company\_name = “First Bank Corporation”) ) }

b)

tuple-relational

{t | ∃s∈ employee(t[person\_name] = s[person\_name] ∧ t[city] = s[city] ∧

∃u ∈ works( u[person\_name] = s[person\_name] ∧

u[company\_name] = “First Bank Corporation”) ) }

domain-relational

{<person\_name, city> | ∃ street (<person\_name, street, city> ∈ employee ∧

∃ company\_name, salary(<person\_name, company\_name, salary> ∈ works ∧

company\_name = “First Bank Corporation”) ) }

6.

a) {<A> | ∃ B,C (<A,B,C> ∈ r1) }

b) {<A,B,C> | <A,B,C> ∈ r1 ∧ B = 17}

c) {<A,B,C> | <A,B,C> ∈ r1 ∨ <A,B,C> ∈ r2}

d) {<A,B,C> | <A,B,C> ∈ r1 ∧ <A,B,C> ∈ r2}

e) {<A,B,C> | <A,B,C> ∈ r1 ∧ ￢(<A,B,C> ∈ r2)}

f) {<A,B,C> | ∃ C (<A,B,C> ∈ r1) ∧ ∃ A (<A,B,C> ∈ r2)}

7.

a)

JDBC와 ODBC의 common feature

1. communicateing with database system을 지원한다.

기본적으로, JDBC, ODBC 모두 기존에 존재하는 database system에 접속해서,

쿼리를 날려 해당하는 결과를 받아오거나 데이터베이스를 업데이트 할 수 있는 기능을 지원한다.

2. prepared statement를 지원한다.

prepared statement란, 데이터 베이스에서 한번 Complie된 쿼리문을

placeholder에 해당하는 값만 바꿔서 여러 번 사용하는 것으로,

필요한 부분만 유저 인풋으로 받아서, 내부 파싱을 거쳐 사용하여

쿼리 자체를 인풋으로 받을 때 일어날 수 있는 SQL인젝션을 막고,

여러번의 컴파일이 필요 없어 효율적으로 쿼리를 실행할 수 있다.

3. metadata features를 지원한다.

클라이언트 단에서, 데이터베이스에 있는 relation 정보나, 쿼리의 column에 대한 정보를 알기 위해서는 relation 혹은 query에 대한 metadata 정보가 필요한데, JDBC와 ODBC에서는 이 메타 데이터를 받아올 수 있는 함수가 존재한다.

4. transaction control을 지원한다.

기본적으로, SQL statement 각각은 서로 다른 transaction으로 취급되어, 자동으로 commit되곤 하는데, 이렇게 자동으로 commit되는것이 아니라, automatic commit을 끄고

유저가 원하는 시점에, statement들을 실행 후 commit하고, rollback할 수 있는 명령어를 둘다 모두 지원한다.

b)

외부 언어로 된 function/procedure를 사용하는 장점과 단점

장점 : mysql과 다르게 general purpose language를 사용하기 때문에,

sql 로 구현한 function/procedure보다 더 효율적이고,

sql로 구현하지 못하는 function/procedure도 구현할 수 있다.

단점 :

함수를 구현하기 위한 코드가 database system에 로드 되어야 하고,

외부에서 구현된 함수를 database system의 주소공간에서 실행해야 하므로,

외부에서 구현된 함수는 안전성을 보장할 수 없으므로,

외부에서 구현된 함수가 데이터베이스 구조를 의도치 않게 망가트릴 수 있고,

혹은 유저가 허용되지 않은 데이터에 접근할 수 있도록 할 수도 있어

security risk가 발생한다.

8.

EF codd는 1970년, 논문 "A Relational Model of Data for Large Shared Data Banks" 를 통해 relational model을 발표했다. IBM은 기존 IMS/DB를 통한 수입을 보호하기 위해 이 모델의 도입을 미뤘지만, Codd가 직접 IBM의 고객에게 이 모델의 유용성을 보여 IBM에게 도입을 압박하여 이 모델이 실제로 적용되는 데도 기여했다. Codd는 relational model과 관련된 relational algebra, relational calculus 도 창시하였고, Boyce-Codd normal form과 같은 개념의 도입을 이루어 내었다. 여러 데이터 베이스에서 그가 만들어 낸 relational model을 제대로 구현하지 못하고 이름만 가져가는 것이 보이자, 직접 관계형 데이터 베이스를 정의하는 12가지 규칙을 발표해서 관계형 데이터 베이스의 표준화에 기여했다.