COP 4710 – Database Systems – Fall 2013

Practice Problems for Exam #2 **KEY**

## Problem #1 - Queries

Use this sample database:

s (s#, name, rank, city, workers)
p (p#, name, color, weight, city)
j (j#, name, workers, city)
spj (s#, p#, j#, qty)

where: in s: rank is a numeric field, and workers is the number of employees of that supplier.
in p: city is the city in which the part is built.
in j: workers is the number of workers on that job.

1. List the names of all the suppliers who supply part P2 to any job.

### Relational Algebra

*Step-by-step technique:*

$T1 = \sigma_{(p\#=P2)}(spj)$ //get all shipment tuples that involve part P2
$T2 = s \times T1$ //combine all P2 shipments with all supplier tuples
$T3 = \sigma_{(s.s\# = spj.s\#)}(T2)$ //remove "junk" tuples from T2, want s# matches
$T4 = \pi_{(name)}(T3)$ //project only supplier names from T3

*Single expression technique:*

$$\pi_{(name)}(\sigma_{(s.s\# = spj.s\#)}(s \times (\sigma_{(p\#=P2)}(spj))))$$

*Another form that also works is:*

$$\pi_{(name)}(\sigma_{((s.s\# = spj.s\#) \text{ AND } (p\# = P2))}(s \times spj))$$

Think about why the first form might be better than the second form.

## Tuple Calculus

{t.name | t ∈ s and ∃u (u ∈ spj and u.p# = “P2” and u.s#=t.s# ) }

## SQL

```
SELECT name
FROM s NATURAL JOIN spj
WHERE spj.p# = “P2”;

-or –

SELECT sname
FROM s
WHERE s# IN (SELECT s#
                    FROM spj
                    WHERE p# = “P2”);
```

2. List the names of those cities in which there is a job located that employs more than 200 workers.

## Relational Algebra

*Step-by-step technique:*

$T1 = \sigma_{(workers > 200)}( j )$ //select all tuples in *j* with > 200 workers

$T2 = \pi_{(city)}(T1)$ //project the city attribute from tuples in T1

*Single expression technique:*

$$\pi_{(city)}(\sigma_{(workers > 200)}( j ))$$

## Tuple Calculus

{t.name | t ∈ j and t.workers > 200 }

## SQL

```
SELECT name
FROM j
WHERE workers > 200;
```

3. List the supplier names for those suppliers who supply at least one red part.

## Relational Algebra

*Step-by-step technique:*

$T1 = \sigma_{(color = red)}(p)$ // select all tuples involving red parts
$T2 = spj \times T1$ //combine all *spj* tuples with *T1* tuples
$T3 = \sigma_{(spj.p\# = T1.p\#)}(T2)$ //remove "junk" tuples from T2, match p#
$T4 = \pi_{(s\#)}(T3)$ //project out only *s#* attributes from T3
$T5 = s \times T4$ //combine all *s* tuples with *T4* tuples
$T6 = \sigma_{(s.s\# = T5.s\#)}(T5)$ //remove "junk" tuples from T5, match s#
$T7 = \pi_{(name)}(T6)$ //project the names

*Single expression technique:*

$$\pi_{(name)}(\sigma_{(s.s\# = spj.s\#)}(s \times \pi_{(spj.s\#)}(\sigma_{(spj.p\# = p.p\#)}(spj \times (\sigma_{(color = red)}(p))))))$$

## Tuple Calculus

{t.name | t ∈ s and ∃u (u ∈ spj and t.s# = u.s# and ∃v ( v ∈ p and v.color = "red"
and v.p# = u.p# ) ) }

## SQL

```
SELECT name
FROM s
WHERE EXISTS (SELECT *
                    FROM spj
                    WHERE spj.s# = s.s# AND EXISTS (SELECT *
                                                          FROM p
                                                          WHERE p.p# = spj.p#
                                                              AND
                                                                    color = "red"));
```

-or-

```
SELECT name
FROM s NATURAL JOIN spj
WHERE spj.p# IN (SELECT p#
                    FROM p
                    WHERE color = "red");
```

-or-

```
SELECT name
FROM s
WHERE s# IN (SELECT s#
                FROM spj
             WHERE p# IN (SELECT p#
                          FROM p
                          WHERE color = “red”));
```

4. Write a relational algebra expression that correctly answers the query: List the names of those suppliers who both job number J1 and J2 with any part.

**Relational Algbera**

$$\pi_{name}\left(\sigma_{spj.s\#=s.s\#}\left(s\times\left(\pi_{s\#}\left(s\right)-\left(\pi_{s\#}\left(\sigma_{p\#=P2}\left(spj\right)\right)\right)\right)\right)\right)$$

**Tuple Calculus**

{t.name | t ∈ s and NOT ∃u (u ∈ spj and u.s# = t.s# and u.p# = “P2”) }

**SQL**

```
SELECT name
FROM s
WHERE s# IN (SELECT s#
                FROM spj
                WHERE spj.j# = “J1”)
      AND
      s# IN (SELECT s#
             FROM spj
             WHERE spj.j# = “J2”);
```

5. Write a relational algebra expression that correctly answers the query: List the supplier names from those suppliers who do not supply part number P2.

**Relational Algebra**

$$\pi_{name}\left(\sigma_{s.s\#=spj.s\#}\left(s\times\left(\pi_{s\#}\left(\left(\sigma_{j\#="J1"}\left(spj\right)\right)\cap\left(\pi_{s\#}\left(\sigma_{j\#="J2"}\left(spj\right)\right)\right)\right)\right)\right)\right)$$

### Tuple Calculus

{t.name | t ∈ s and ∃u (u ∈ spj and u.s# = t.s# and u.j# = “J1” and ∃v ( v ∈ spj and v.s# = t.s# and v.j# = “J2” ) ) }

### SQL

```
SELECT name
FROM s
WHERE s# NOT IN (SELECT s#
                 FROM spj
                 WHERE p# = “P2”);
```

6. List all quadruples of the form (s#, p#, s#, p#) where the first supplier number ships the same part number as the second supplier, but the first supplier has at least one shipment of that part in a quantity greater than the second supplier. Eliminate from your result, all cases where the two supplier numbers happen to be the same. (Note that this query requires assigning an alias, i.e., renaming the shipments relation.)

   Let t alias spj.

### Relational Algebra

$$\pi_{(t.s\#,t.p\#,spj.s\#,spj.p\#)}\left(\sigma_{(t.p\#=spj.p\# \text{ AND } t.qty>spj.qty \text{ AND } t.s\#\neq spj.s\#)}\left(t\times spj\right)\right)$$

### Tuple Calculus

{t.s#, t.p#, u.s#, u.p# | t ∈ spj and u ∈ spj and t.s# <> u.s# and t.p# = u.p# and ∃v ( v ∈ spj and v.s# = t.s# and v.p# = u.p# and v.qty > u.qty ) }

### SQL

```
SELECT spj.s#, spj.p#, spj1.s#, spj1.p#
FROM spj CROSS JOIN spj as spj1
WHERE spj.s# <> spj1.s# and spj.p# = spj1.p# and
      spj.s# IN (SELECT s#
                 FROM spj as spj2
                 WHERE spj2.qty > spj1.qty and spj2.s# <> spj1.s#);
```

## Problem #2 – 3NF Decomposition

Given the relation scheme R, the set of functional dependencies F, and the set of keys K shown below, produce a 3NF decomposition scheme of R with respect to F. Clearly show the final decomposition scheme. Do NOT test for either the lossless join nor the preservation of dependencies – just do the decomposition.

R = ( number, name, status, city, quantity, division )
K = { number }
F = { status → quantity, city → division, name → city }

Step 1: Remove transitive dependency: number → status → quantity

R1 = ( number, name, status, city, division )
K1 = K = { number }

R2 = ( status, quantity ) in 3NF
K2 = { status }

Step 2: Remove transitive dependency: number → city → division

R11 = ( number, name, status, city )
K11 = K1 = K = { number }

R12 = ( city, division ) in 3NF
K12 = { city }

Step 3: Remove transitive dependency: number → name → city

R111 = ( number, name, status ) in 3NF
K111 = K11 = K1 = K = { number }

R112 = ( name, city ) in 3NF
K112 = { name }

Final decomposition scheme D = { R2, R12, R111, R112 }