Weekly exercises IN3020&4020 (2024)
Week 2: SQL Recap

Note: This exercise set is on paper, but feel free to experiment your queries with PostgreSQL. For your convenience, there is test data prepared for your experiments: `https://www-adm.uio.no/studier/emner/matnat/ifi/IN3020/v23/weekly_exercises/in3020_exercises_week2_test_data.sql`. Remember, however, that knowledge of the syntactic details of SQL will not be important during exam.

1 Basic SQL

Consider four relations with the following schemas (the domains and the meaning of the attributes are as expected):

Product [*manufacturer*, *model*, *type*]
PC [*model*, *speed*, *RAM*, *HDD*, *price*]
Laptop [*model*, *speed*, *RAM*, *HDD*, *screen*, *price*]
Printer [*model*, *colour*, *type*, *price*]

Write SQL queries that find the following:

1. the PC models that have speed at least 2.50;

   Solution:

   ```
   SELECT model
   FROM PC
   WHERE speed >= 2.50;
   ```

2. the manufacturers that produce printers (i.e., products of type 'printer'), but not PC's;

   Solution:

   ```
   (SELECT manufacturer
   FROM Product
   WHERE type = 'printer')
     EXCEPT
   ```

```
(SELECT manufacturer
FROM Product
WHERE type = 'pc');
```

3. the manufacturers that make laptops with a HDD of at least 500 GB;

   Solution:

```
SELECT DISTINCT p.manufacturer
FROM Product p INNER JOIN Laptop l
ON p.model = l.model
WHERE HDD >= 500;
```

4. the models and prices of all products made by manufacturer hp;

   Solution:

```
(SELECT q.model, q.price
FROM Product p INNER JOIN PC q
ON p.model = q.model
WHERE p.manufacturer = 'hp')
    UNION
(SELECT l.model, l.price
FROM Product p INNER JOIN Laptop l
ON p.model = l.model
WHERE p.manufacturer = 'hp')
    UNION
(SELECT r.model, r.price
FROM Product p INNER JOIN Printer r
ON p.model = r.model
WHERE p.manufacturer = 'hp');
```

5. the manufacturers of at least two different computers (PCs or laptops); with speed at least 2.0 (hint: use a `WITH` statement in this query);

   Solution:

```
WITH pc_and_laptop AS (
    (SELECT model
    FROM PC
    WHERE speed >= 2.0)
         UNION
    (SELECT model
    FROM Laptop
    WHERE speed >= 2.0)
)
SELECT DISTINCT p.manufacturer
FROM pc_and_laptop m INNER JOIN Product p
```

```
ON m.model = p.model
GROUP by p.manufacturer
HAVING COUNT(p.model) >= 2;
```

6. list the manufacturers with the number of black-and-white printer models and the number of colour printer models they produce; order by the number of colour printers (i.e., the result should be a table with 3 columns: *manufacturer*, *num_baw*, *num_colour*).

   Solution:

   ```
   SELECT p.manufacturer,
     SUM(CASE WHEN r.colour THEN 0 ELSE 1 END) AS num_baw,
     SUM(CASE WHEN r.colour THEN 1 ELSE 0 END) AS num_colour
   FROM Product p INNER JOIN Printer r ON p.model = r.model
   GROUP BY p.manufacturer
   ORDER BY num_colour DESC;
   ```

Give the examples of (states of) relations *Product* and *Laptop* so that

7. all their inner, left outer, right outer, and full outer joins on condition *Product.model* = *PC.model* are different;

   Solution: Any relations would work such that each of the two contains at least one tuple that does not join with any tuples in the other. A simple example is *Product* ⟨*'Apple', 'MacBook', 'Laptop'*⟩ and *Laptop* ⟨*'Dynabook', 2.5, 500, 20, 2000*⟩.

8. all the joins are the same.

   Solution: Any relations would work such that each of the two contains only tuples that join with tuples in the other. A simplest example is when both relations are empty.

2 Queries in Lectures

Have another look at the recursive queries and queries with subqueries in the lectures and make sure you understand how they work (see slides 17, 24-26, 29 at `https://www.uio.no/studier/emner/matnat/ifi/IN3020/v24/lecture_slides/in3020_lecture_4.pdf`).

3 Subqueries in SQL

Assuming the schemas in Section 1, write two significantly different (but equivalent) SQL queries for each of the following, so that each of your queries

should use at least one subquery with operators `EXISTS`, `IN`, `SOME`, `ALL`, `ANY`:

1. the manufacturers of laptops with speed at least 2.0;

   Solution:

   Option 1:

   ```
   SELECT DISTINCT p.manufacturer
   FROM Product p
   WHERE p.model IN
     (SELECT l.model
         FROM Laptop l
         WHERE l.speed >= 2.0);
   ```

   Option 2:

   ```
   SELECT DISTINCT p.manufacturer
   FROM Product p
   WHERE EXISTS
     (SELECT *
         FROM Laptop l
         WHERE l.model = p.model AND speed >= 2.0);
   ```

2. the printers with the highest price;

   Solution:

   Option 1:

   ```
   SELECT DISTINCT p.model
   FROM Printer p
   WHERE p.price >= ALL (SELECT q.price
                         FROM Printer q);
   ```

   Option 2:

   ```
   SELECT DISTINCT p.model
   FROM Printer p
   WHERE p.price = ANY (SELECT MAX(q.price)
                        FROM Printer q);
   ```

3. the laptops whose speed is slower than that of the fastest PC;

   Solution:

   Option 1:

```
SELECT l.model
FROM Laptop l
WHERE l.speed < (SELECT MAX(speed)
                 FROM PC p);
```

Option 2:

```
SELECT l.model
FROM Laptop l
WHERE l.speed < ANY (SELECT p.speed
                     FROM PC p);
```

4. the models (of PCs, laptops, and printers) with the lowest price;

   Solution:

   Option 1:

```
SELECT u.model
FROM ((SELECT model, price FROM PC)
       UNION ALL
      (SELECT model, price FROM Laptop)
       UNION ALL
      (SELECT model, price FROM Printer)) AS u
WHERE u.price <= ALL ((SELECT price FROM PC)
                      UNION ALL
                      (SELECT price FROM Laptop)
                      UNION ALL
                      (SELECT price FROM Printer));
```

   Option 2:

```
(SELECT model
FROM PC
WHERE price <= ALL((SELECT MIN(price) FROM PC)
    UNION ALL
    (SELECT MIN(price) FROM Laptop)
    UNION ALL
    (SELECT MIN(price) FROM Printer)))
             UNION ALL
(SELECT model
FROM Laptop
WHERE price <= ALL((SELECT MIN(price) FROM PC)
    UNION ALL
    (SELECT MIN(price) FROM Laptop)
```

```
    UNION ALL
    (SELECT MIN(price) FROM Printer)))
             UNION ALL
(SELECT model
FROM Printer
WHERE price <= ALL((SELECT MIN(price) FROM PC)
    UNION ALL
    (SELECT MIN(price) FROM Laptop)
    UNION ALL
    (SELECT MIN(price) FROM Printer)));
```

5. the manufacturers of colour printers with the highest price.

   Solution:

   Option 1:

```
SELECT DISTINCT p.manufacturer
FROM Product p
WHERE p.model IN (SELECT q.model
              FROM Printer q
              WHERE r.colour AND
                    q.price >= SOME (SELECT MAX(r.price)
                                     FROM Printer r
                                     WHERE r.colour));
```

   Option 2:

```
SELECT DISTINCT p.manufacturer
FROM Product p, Printer q
WHERE p.model = q.model AND q.colour AND
NOT EXISTS (SELECT *
            FROM Printer r
            WHERE r.colour AND r.price > q.price);
```

4 Recursive SQL

1. Consider the following schema concerning scheduled flights for tomorrow (the domains and the meaning of the attributes are as expected):

$$\mathit{Flights}\,[\mathit{origin}, \mathit{destination}, \mathit{arrival_time}, \mathit{departure_time}]$$

Write a recursive SQL query that finds all pairs of cities (x, y) such that it is

possible to get from x to y using a maximum of 5 connecting flights with at least one hour transfer time at each transfer.

Solution:

```
WITH RECURSIVE trans(origin, destination, arrival_time, num_transits) AS (
    (SELECT origin, destination, arrival_time, 0 FROM Flights)
          UNION
    (SELECT t.origin, f.destination, f.arrival_time, t.num_transits + 1
     FROM trans t, Flights f
     WHERE t.destination = f.origin
         AND f.departure_time >= t.arrival_time + TIME '01:00:00'
         AND t.num_transits < 5)
) SELECT DISTINCT origin, destination FROM trans ;
```

2. Consider now the following schema concerning flight routes:

$$Routes\,[origin, destination]$$

Write a recursive SQL query that finds all pairs of cities (x, y) such that it is possible to get from x to y using a *minimum*(!) of 5 connecting flights (assuming that all routes are realised via frequent enough flights).

Solution:

```
WITH RECURSIVE trans(origin, destination, num_trans_bnd) AS (
    (SELECT origin, destination, 0 FROM Routes)
          UNION
    (SELECT t.origin, r.destination,
        CASE WHEN t.num_trans_bnd < 5 THEN t.num_trans_bnd + 1 ELSE 5 END
     FROM trans t, Routes r
     WHERE t.destination = r.origin)
) SELECT DISTINCT origin, destination FROM trans WHERE num_trans_bnd = 5;
```

Note that we cannot just blindly replace '`< 5`' by '`> 5`' in the previous query (along with removing everything related to time). First, we need to accumulate all the paths, so the condition '`t.num_transits < 5`' inside the recursion is replaced by the condition in the outer query. A more subtle problem is that we cannot just collect all triples of the form $(origin, destination, num_transitions)$ in the recursive query, because there are possibly infinite number of them and hence the recursive part may not stop; to deal with this, we may 'count up to 5' by means of the `CASE` expression.

3(*). Do you think that there exists a recursion-free SQL query equivalent to the query in Question 1? In Question 2? Explain your answers.

Solution: For Question 1, it is possible, and the main reason is that the number of connections is bounded by a number that does not depend on the content of the table. We just need to write a union of 6 queries, each of which is a join of i, $i = 0, \ldots, 5$, copies of *Flights* (for $i = 0$, we just return pairs of the form (x, x)). For Question 2, it is not possible, because the number of joins is unbounded, and we have to use recursion.