Exercise 5: SQL II

Solution

The exercises marked with * will be discussed in the exercise session. You can solve the other exercises as practice, ask questions about them in the session, and hand them in for feedback. All exercises may be relevant for the exam.

Ask Zsolt (zsolt.istvan@inf.ethz.ch) for feedback on this week’s exercise sheet or give it to the TA of your session (preferably stapled and with your e-mail address).

This exercise sheet builds on the previous ones and it uses the Employees and ZVV schema and data.

1. Rewriting Queries (TPC-H)*

1. For the original query below, which other alternative queries are equivalent to it (ignoring attribute names)?

   − − Original
   
   SELECT MAX(supplypartcost) AS mostexpensive FROM supplypart

   √ − − Alt1
   
   SELECT DISTINCT supplypartcost
   FROM supplypart
   WHERE supplypartcost >= ALL (SELECT supplypartcost FROM supplypart)

   ○ − − Alt2
   
   SELECT supplypartcost spc
   FROM supplypart
   ORDER BY spc
   LIMIT 1
Explanation: The ORDER BY operation will order on most databases by default in ascending manner. To have an equivalent query we would need a descending order.

√  -- Alt3
WITH innerexp AS (  
    SELECT partid, supplierid  
    FROM supplypart  
    ORDER BY supplypartcost DESC  
    LIMIT 1)  
SELECT supplypartcost spc  
FROM supplypart sp JOIN innerexp ie  
on ie.partid=sp.partid AND ie.supplierid=sp.supplierid

Explanation: Note that we make two assumptions about the schema (which are fulfilled in the schema as we provided it to you): (1) partid and supplierid must be NOT NULL. Otherwise, the tuple of the inner query may not match with itself in the join. (2) partid, supplierid must be a key of supplypart. Otherwise, the tuple of the inner query could match several tuples in the join.

○  -- Alt4
WITH innerexp AS (  
    SELECT partid, supplierid  
    FROM supplypart  
    ORDER BY supplypartcost DESC  
    LIMIT 1)  
SELECT sp.supplypartcost spc  
FROM supplypart sp LEFT JOIN innerexp ie  
on ie.partid=sp.partid AND ie.supplierid=sp.supplierid

Explanation: The LEFT join will produce one output tuple for each tuple of the left relation, even if there is no matching tuple in the right relation. Therefore the result here will have all cost entries of the left table.

√  -- Alt5
(SELECT supplypartcost spc  
FROM supplypart  
ORDER BY spc ASC )  
INTERSECT  
(SELECT supplypartcost spc  
FROM supplypart  
ORDER BY spc DESC  
LIMIT 1)

2. Fill in the gaps in the skeleton query below to make it equivalent to the following query:

    -- Original  
    SELECT orderid  
    FROM orderline
WHERE olquantity > 10
GROUP BY orderid
HAVING min(oldiscount) > 0.09

2 Statistics on Salaries (Employees)

Hint: Whenever we refer to “current” or “present” in this exercise sheet, we refer to the date returned by the built-in NOW() function. Additionally, take a look at the date_part() function provided by PostgreSQL. It is useful if you want to extract the year, month, etc. of a date.

1. What is the current highest salary in each department if we only consider people who have been hired at the company before 1990?

Solution:

```
SELECT dp.dept_name AS department, MAX(salary) AS max_salary
FROM salaries s, dept_emp de, departments dp, employee emp
WHERE
de.emp_no = s.emp_no AND
de.to_date > NOW() AND
s.to_date > NOW() AND
dp.dept_no = de.dept_no AND
emp.emp_no = de.emp_no AND
DATE_PART('year', emp.hire_date) < 1990
GROUP BY dp.dept_name
```

Alternatively, if we do not want to use hire date and rely on salaries instead, we could look for salaries that have begun before 2001:

```
SELECT dp.dept_name AS department, MAX(salary) AS max_salary
FROM salaries s, dept_emp de, departments dp, salaries sother
WHERE
de.emp_no = s.emp_no AND
de.to_date > NOW() AND
s.to_date > NOW() AND
dp.dept_no = de.dept_no AND
sother.emp_no = s.emp_no AND
DATE_PART('year', sother.from_date) < 2001
GROUP BY dp.dept_name
```
2. We want to find what was the ratio between the lowest and highest starting salaries (salaries matching the hire_date of employees) in the Marketing department over the course of the years. Output the ratio for each year in which there were people starting in the Marketing department.

Hint: you can perform GROUP BYs on any column or expression specified in the SELECT clause.

Hint 2: you may need to use explicit casting to compute a floating point value.

Solution:

```sql
SELECT MIN(salary)/MAX(salary) AS salary_mult, DATE_PART('year', s.from_date) AS salyear
FROM salaries s, dept_emp de, departments dp, employees emp
WHERE emp.emp_no = de.emp_no
  AND de.emp_no = s.emp_no
  AND dp.dept_no = de.dept_no
  AND dp.dept_name = 'Marketing'
  AND emp.hire_date = s.from_date
  AND de.from_date = s.from_date
GROUP BY DATE_PART('year', s.from_date)
ORDER BY salyear
```

3. The following four queries try to determine what was the average salary of employees working in each department in 1997 (the average is not weighted by employment length). Mark the correct solution(s):

- SELECT dp.dept_name, AVG(salary) AS average_salary
  FROM salaries s, dept_emp de, departments dp
  WHERE de.emp_no = s.emp_no
    AND de.from_date < s.to_date
    AND de.to_date > s.from_date
    AND dp.dept_no = de.dept_no
    AND (s.from_date BETWEEN '1997-01-01' AND '1997-12-31' OR s.to_date BETWEEN '1997-01-01' AND '1998-01-01')
  GROUP BY dp.dept_name

- SELECT dp.dept_name, AVG(salary) AS average_salary
  FROM salaries s, dept_emp de, departments dp
  WHERE de.emp_no = s.emp_no
    AND de.from_date < s.to_date
    AND de.to_date > s.from_date
    AND s.from_date <= '1997-12-31'
    AND s.to_date > '1997-01-01'
    AND dp.dept_no = de.dept_no
  GROUP BY dp.dept_name

- SELECT dp.dept_name, AVG(salary) AS average_salary
  FROM salaries s, dept_emp de, departments dp
  WHERE de.emp_no = s.emp_no
    AND de.from_date < s.to_date
    AND de.to_date > s.from_date
  GROUP BY dp.dept_name

- SELECT dp.dept_name, AVG(salary) AS average_salary
  FROM salaries s, dept_emp de, departments dp
  WHERE de.emp_no = s.emp_no
    AND de.from_date < s.to_date
    AND de.to_date > s.from_date
    AND s.from_date <= '1997-12-31'
    AND s.to_date > '1997-01-01'
  GROUP BY dp.dept_name

Mark the correct solution(s):
with emp_sal AS (  
SELECT s.emp_no AS employee, s.salary AS salary,  
s.from_date AS started, s.to_date AS ended  
FROM salaries s  
WHERE  
s.from_date <= '1997-12-31' AND  
s.to_date > '1998-01-01'  
)  
SELECT dp.dept_name AS department,  
count(emp_sal.salary) AS appears, emp_sal.employee  
FROM emp_sal, departments dp, dept_emp de  
WHERE  
  dp.dept_no=de.dept_no AND  
  emp_sal.employee=de.emp_no AND  
  de.from_date<emp_sal.ended AND  
  de.to_date>emp_sal.started  
GROUP BY dp.dept_name, emp_sal.employee  
ORDER BY appears DESC

with emp_sal AS (  
SELECT emp_no AS employee, salary AS salary,  
from_date AS started  
FROM salaries  
WHERE  
  from_date <= '1997-12-31' AND  
  to_date > '1997-01-01'  
)  
SELECT dp.dept_name AS department, AVG(emp_sal.salary)  
FROM emp_sal, departments dp, dept_emp de  
WHERE  
  dp.dept_no=de.dept_no AND  
  emp_sal.employee=de.emp_no AND  
  de.from_date <= '1997-12-31' AND  
  de.to_date > '1997-01-01'  
GROUP BY dp.dept_name  
ORDER BY department

Solution: Explanation:

- The first option is incorrect. What if someone has been salaried at the department starting in 1996, ending in 1999?
- The third option is also incorrect. It returns how many times a person’s salary changed in 1997
- The last option is also incorrect. It might be that a person has changed departments and had two salaries in 1997. This could count both salaries for both departments, instead of just one each. In other words, there is no check of overlapping intervals.
3 Creating a view (Employees)*

1. Create a view (`current_employees`) that contains the employee numbers and salaries of employees who are currently working at the company. Based on this view compute the average salary in the current moment.

```
Solution:
--- Create view
CREATE VIEW current_employees AS
( SELECT emp_no, salary FROM salaries
WHERE to_date >= NOW() OR to_date is NULL );
--- Select average salary
SELECT AVG(salary) FROM current_employees;
```

2. Use the view to determine the employees (give their full names) earning between the 45th and 55th percentile of the current salary distribution in the company.

```
Solution: Option 1 – compute rank manually
WITH absolute_ranked_employees AS
( SELECT e1.emp_no AS emp_no, e1.salary AS salary,
COUNT(e2.emp_no) AS absolute_rank
FROM current_employees e1
JOIN current_employees e2
ON e1.salary <= e2.salary
GROUP BY e1.emp_no, e1.salary ),
ranked_employees AS
( SELECT *,
CAST(absolute_rank-1 AS FLOAT) / ((SELECT COUNT(*)-1)
FROM current_employees) AS rank
FROM absolute_ranked_employees
ORDER BY salary DESC )
SELECT * FROM ranked_employees
WHERE rank BETWEEN 0.45 AND 0.55;

Option 2 – with the rank function
WITH ranked_current_employees AS
( SELECT *, PERCENT_RANK() OVER (ORDER BY salary) AS rank
FROM current_employees )
SELECT e.first_name, e.last_name, salary
FROM ranked_current_employees ce
JOIN employees e ON e.emp_no = ce.emp_no
WHERE rank BETWEEN 0.45 AND 0.55;
```
Option 3 – using a window function

```sql
SELECT first_name, last_name, em.emp_no, salary
FROM current_employees ce, employees em
WHERE salary >= (SELECT percentile_disc(0.45) WITHIN GROUP (ORDER BY salary)
                FROM current_employees ce) AND
      salary < (SELECT percentile_disc(0.55) WITHIN GROUP (ORDER BY salary)
                FROM current_employees ce) AND
      em.emp_no = ce.emp_no
ORDER BY salary
```

Option 4a – use order and limit (wrong)

```sql
SELECT e.first_name, e.last_name, salary
FROM employees e
JOIN (SELECT * FROM current_employees ce
       ORDER BY ce.salary ASC
       LIMIT 0.55 * (SELECT COUNT(*) FROM current_employees)
       ) ce_lim ON ce_lim.emp_no = e.emp_no
ORDER BY ce_lim.salary DESC
LIMIT 0.1 * (SELECT COUNT(*) FROM current_employees)
```

While this option looks good, it does not compute the correct result in case several employees have the 45-percentile salary. In that case, the query returns some of them but does not return the others. That does not correspond to what it is supposed to return. However, we can use a similar idea as shown below.

Option 4b – use order and limit (fixed)

```sql
WITH interesting_salary_range AS (
    SELECT salary
    FROM (SELECT * FROM current_employees ce
           ORDER BY ce.salary ASC
           LIMIT 0.55 * (SELECT COUNT(*) FROM current_employees)
           ) AS lowest55
    ORDER BY salary DESC
    LIMIT 0.1 * (SELECT COUNT(*) FROM current_employees)
)
SELECT first_name, last_name, salary
FROM current_employees ce
JOIN employees em ON em.emp_no = ce.emp_no
WHERE salary >= (SELECT * FROM interesting_salary_range
                 ORDER BY salary ASC LIMIT 1) AND
      salary < (SELECT * FROM interesting_salary_range
                 ORDER BY salary DESC LIMIT 1)
ORDER BY salary
```
4 Top Dogs (Employees)*

1. Write a SQL query that finds the highest 3 salaries currently paid in each department. Include the rank (1st, 2nd and 3rd highest) as an extra column in the result.

   Hint 1: Try breaking the problem up into subproblems. Maybe you have already solved one of them.

   Hint 2: `SELECT DISTINCT` returns a set in which each identical record appears only once.

   Hint 3: It is possible to include a `SELECT` statement in the column list of another query. For example:

   ```sql
   SELECT name, (SELECT max(price) FROM purchases p WHERE p.paid_by = name) AS most_expensive_purchase
   FROM persons WHERE name = 'Jonas'
   ```

   **Solution:** Option 1 – Compute rank manually

   ```sql
   WITH current_dept_emp AS (SELECT * FROM dept_emp WHERE NOW() BETWEEN from_date AND to_date),
   current_employees_ranked_by_dept AS (SELECT e1.emp_no AS emp_no, e1.salary AS salary, de1.dept_no AS dept_no, COUNT(e2.emp_no) AS rank FROM current_employees e1 JOIN current_employees e2 ON e1.salary <= e2.salary JOIN current_dept_emp de1 ON e1.emp_no = de1.emp_no JOIN current_dept_emp de2 ON e2.emp_no = de2.emp_no AND de2.dept_no = de1.dept_no GROUP BY e1.emp_no, e1.salary, de1.dept_no ORDER BY dept_no, rank),
   SELECT * FROM current_employees_ranked_by_dept WHERE rank <= 3;
   ```

   Option 2 – Use rank function

   ```sql
   WITH current_employees_ranked_by_dept AS (SELECT *, RANK() OVER (PARTITION BY de.dept_no ORDER BY salary DESC) AS rank FROM current_employees ce -- use the view from the previous exercise!
   JOIN dept_emp de ON de.emp_no = ce.emp_no
   JOIN departments d ON d.dept_no = de.dept_no
   WHERE NOW() BETWEEN de.from_date AND de.to_date),
   SELECT * FROM current_employees_ranked_by_dept WHERE rank <= 3;
   ```

2. List the name of the people (from any department) who earn more at the moment than the 3rd highest current salary in the department "Marketing". Hint: create a view (`create view topsalaries AS ...`) with the solution to the previous exercise, and write a query against this view instead of the data.
Solution:

```sql
SELECT distinct first_name, last_name
FROM topsalaries, salaries, employees
WHERE topsalaries.department = 'Marketing' AND
      topsalaries.rank=3 AND
      salaries.emp_no = employees.emp_no AND
      salaries.salary > topsalaries.salary AND
      salaries.to_date > NOW()
```

3. How many people earn more than the 3rd highest current salary in 'Marketing' and also more than the 2nd highest salary in 'Finance'? Write the query using the view.

Solution:

```sql
SELECT count(salaries.salary)
FROM topsalaries t1, topsalaries t2, salaries, employees
WHERE t1.department = 'Marketing' AND
      t1.rank=3 AND
      t2.department = 'Finance' AND
      t2.rank=2 AND
      salaries.emp_no = employees.emp_no AND
      salaries.salary > t1.salary AND
      salaries.salary > t2.salary AND
      salaries.to_date > NOW()
```

5  Raffle (Employees)

1. Imagine that there is a raffle in the company and the person with the most 'followers' will get the prize. Each person (P) has an other person (F) AS follower if the first name of F and the last name of P start with the same three letters. Followers of F are also followers of P. If a person could be a follower to P on multiple "levels", it is counted multiple times. Give the employee number of person P with the largest following when counting up to 3 levels of followers (Hint: use the with clause).

What would happen to this query if we lifted the 3 level constraint and computed the whole following graph instead? Could it enter an infinite loop?

Example: P = John [Car]penter; F = [Car]ly Smith

Solution:

```sql
WITH RECURSIVE enames AS (  
  SELECT first_name, last_name, emp_no, 0 AS len  
  FROM employees  
  UNION ALL  
  SELECT p.first_name, p.last_name, e.emp_no, e.len+1 AS len
```
FROM employees p INNER JOIN enames e ON SUBSTRING(p.first_name, 1, 3)=
    SUBSTRING(e.last_name, 1, 3) AND p.emp_no<>e.emp_no AND len<=3

) SELECT emp_no, COUNT(*) AS cnt
FROM enames
GROUP BY emp_no
ORDER BY cnt DESC
LIMIT 1

Since we are in essence computing a transitive closure on this data, if there is a circle
in the person’s names the recursive query will never return unless we limit the depth to
which it can go.

6 Connections (ZVV)*

1. List all direct connections that depart from stations that contain 'ETH' in the station’s name
   (stop_name) and arrive at stations that contain 'Milchbuck' in the name. The departure
time (departure_time) and arrival time (arrival_time) must be between '13:00:00' and
   '14:00:00'. Show the stop_name of the departing station, stop_name of the arrival station,
   the tram_number of the tram line, the departure_time, arrival_time and trip direction
   (trip_headsign). Order the list in ascending order by departure_time.

   Solution:
   WITH
       connections AS (SELECT ss.stop_name AS start_name, es.stop_name AS end_name,
                           st.departure_time AS dtime, et.arrival_time AS atime,
                           tram_number, trip_headsign
                       FROM stops ss
                       JOIN stop_times st ON ss.stop_id = st.stop_id
                       JOIN stop_times et ON et.trip_id = st.trip_id
                       JOIN stops es ON es.stop_id = et.stop_id
                       JOIN trips t ON et.trip_id = t.trip_id
                       WHERE st.stop_sequence < et.stop_sequence)
   SELECT * FROM connections
   WHERE start_name LIKE '%ETH%' AND
         end_name LIKE '%Milchbuck%' AND
         dtime BETWEEN '13:00:00' AND '14:00:00' AND
         atime BETWEEN '13:00:00' AND '14:00:00'
   ORDER BY dtime

2. List all single transfer connections that depart FROM stations with a name that con-
tains 'ETH' and arrive at stations with a name that contains 'Seebach'. The departure
time (departure_time) and arrival time (arrival_time) must be between '13:00:00' and
'14:00:00'.

   With
For each connection show:

- For the first trip
  - departure stop_name,
  - tram_number,
  - departure_time,
  - trip direction (trip_headsign)
  - arrival_time at intermediate stop,
  - intermediate stop_name,

- For the second tram
  - tram_number,
  - departure_time,
  - trip direction (trip_headsign)
  - arrival_time at destination stop,
  - destination stop_name.

Clarification: a single transfer connection is associated with 2 trips (e.g. T1, T2) and 3 stops (e.g. S1, S2, S3), where T1 stops at S1 and S2, while T2 stops at S2 and S3. Pay attention to the order of the stops and that the arrival time for T1 at S2 is no later than the departure time of T2 at S2.

Solution: Option 1 – readable, but slow:

WITH connections AS (
  SELECT ss.stop_name AS start_name, es.stop_name AS end_name,
       st.departure_time AS dtime, et.arrival_time AS atime,
       tram_number, trip_headsign
  FROM stops ss
  JOIN stop_times st ON ss.stop_id = st.stop_id
  JOIN stop_times et ON et.trip_id = st.trip_id
  JOIN stops es ON es.stop_id = et.stop_id
  JOIN trips t ON et.trip_id = t.trip_id
  WHERE st.stop_sequence < et.stop_sequence

  -- Comment in the following two lines to make it faster
  -- AND st.departure_time BETWEEN '13:00:00' AND '14:00:00'
  -- AND et.arrival_time BETWEEN '13:00:00' AND '14:00:00'
)

SELECT c1.start_name, c1.tram_number, c1.trip_headsign, c1.dtime, c1.atime,
       c1.end_name, c2.tram_number, c2.trip_headsign, c2.dtime, c2.atime,
       c2.end_name
FROM connections c1
JOIN connections c2 ON c1.end_name = c2.start_name
WHERE c1.start_name LIKE '%ETH%' AND c2.end_name LIKE '%Seebach%' AND c1.atime < c2.dtime AND c1.dtime BETWEEN '13:00:00' AND '14:00:00' AND c2.atime BETWEEN '13:00:00' AND '14:00:00'

ORDER BY c1.dtime
Option 2 – much faster, but less readable:

```sql
SELECT depS.stop_name, ft.tram_number, ft.trip_headsign,
       depST.departure_time AS dep_time, st.arrival_time,
       sint.stop_name, str.trip_headsign,
       str.tram_number, st1.departure_time,
       st2.arrival_time AS arr_time, s.stop_name
FROM stop_times depST, stops depS, stop_times st, stop_times st1,
     stop_times st2, stops s, stops sint, trips ft, trips str
WHERE depS.stop_name LIKE '%ETH%'
AND depST.stop_id = depS.stop_id
AND depST.departure_time BETWEEN '13:00:00' AND '14:00:00'
AND st.trip_id = ft.trip_id
AND st.trip_id = depST.trip_id
AND st1.trip_id = str.trip_id
AND st.departure_time BETWEEN '13:00:00' AND '14:00:00'
AND st.arrival_time > depST.departure_time
AND st.stop_id = st1.stop_id
AND st.stop_id = sint.stop_id
AND st.arrival_time < st1.departure_time
AND st1.departure_time BETWEEN '13:00:00' AND '14:00:00'
AND st.trip_id != st1.trip_id
AND st1.trip_id = st2.trip_id
AND st1.arrival_time > st1.departure_time
AND st2.arrival_time BETWEEN '13:00:00' AND '14:00:00'
AND st2.stop_id = s.stop_id
AND s.stop_name LIKE '%Seebach%'
ORDER BY st2.arrival_time ASC
```

In theory, the database system should be able to transform one query to the other and execute both queries in the same time. Under this assumption, it is better to write the queries such that other programmers can easily understand them. In practice, unfortunately, there may be performance differences like here.

Note that the questions does not exclude journeys with a single trip where we get off a tram and then back on the same one. If we wanted to exclude that, we could do so with an additional `WHERE` clause that checks that the `trip_ids` of the to trips are different.