TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

***Database System Concepts for Non-Computer Scientist* - WiSe 22/23**

Alice Rey (rey@in.tum.de)
http://db.in.tum.de/teaching/ws2223/DBSandere/?lang=en

Sheet 09

Exercise 1

Answer the following questions on our university database using SQL:

a) Calculate how many lectures each student is attending. Students who do not attend any lecture should be included in the result as well ($attend\_count = 0$) (use outer joins).

b) Figure out how many students each professor knows: A professor knows students from one of their lectures or via a test they have supervised. Include professors not knowing any students and use outer joins. Hint: [1]

[1] Remember that SQL has set operations.

Solution:

a)
```
select s.studNr, s.name, count(a.studNr)
from Students s left outer join attend a on s.studnr = a.studnr
group by s.studNr, s.name
```

b)
```
select p.persNr, p.name, count(p.studNr)
from
((
  select p.persNr, p.name, t.studNr
  from Professors p
    left outer join test t on p.persNr = t.persNr
)
union
(
  select p.persNr, p.name, a.studNr
  from Professors p
    left outer join Lectures l on p.persNr = l.given_by
    left outer join attend a on l.lectureNr = a.lectureNr
)) p
group by p.persNr, p.name
```

Uncorrelated subqueries can be easily transformed into with-statements to make the query more readable:

```
with known_from_tests as (
   select p.persNr, p.name, t.studNr
   from Professors p
      left outer join test t on p.persNr = t.persnr
),
known_from_lectures as (
   select p.persNr, p.name, a.studNr
   from Professors p
      left outer join Lectures l on p.persNr = l.given_by
      left outer join attend a on l.lectureNr = a.lectureNr
),
known as (
   select * from known_from_tests
      union
   select * from known_from_lectures
)
select persNr, name, count(distinct studNr)
from known
group by persNr, name
```

Exercise 2

Find those students who have attended all lectures that they wrote a test in.

Solution:
The requirement that students in the query result should have attended all lectures that they were tested in, can be rephrased as follow: "For a given student, there should be no test/exam, that has no entry in *attend*". This can then be translated into sql easily.

```
select s.*
from Students s
where not exists(select * from test t
                 where s.studNr = t.studNr
                   and not exists
                          (select *
                           from attend a
                           where a.studNr = s.studNr
                             and a.lectureNr = t.lectureNr));
```

This query is an example of a "for all query" where the counting-based technique can not be applied. The reason is that we can not simply count the number of attended lectures, because we need to make sure that the attended lectures match the ones that were tested.
An alternative way that only requires one "not exists" would be to connect the students with their tests and if available add the corresponding attend entry. If there is no attend available, the "left outer join" will leave the "lecture" column empty (adds a "null" value). If we find in our "not exists" subquery an entry where the lecture is null, we can remove

```
with students_tests_optLectures as (
  select s.studnr student, t.lecturenr test, a.lecturenr lecture
  from students s inner join test t on s.studnr = t.studnr
    left outer join attend a on s.studnr = a.studnr and a.lecturenr =
        t.lecturenr
)

select *
from students
where not exists (select * from students_tests_optLectures where
    studnr = student and lecture is null)
```

A second alternative without "not exists" would be to directly search for those students with a null-entry in the with-statement with an additional where clause. The resulting "students_test_woLectures" contains a list of all students that took a test without attending the lecture. Since we are interested in the opposite, we use a set operation to select all students "except" those who took a test without attending the respective lecture.

```
with students_tooktest_didnotattendlecture as (
  select distinct s.studnr
  from students s inner join test t on s.studnr = t.studnr
    left outer join attend a on s.studnr = a.studnr and a.lecturenr =
        t.lecturenr
  where a.lecturenr is null
)

select studnr from students
  except
select * from students_tooktest_didnotattendlecture
```

Exercise 3

„Busy Students“: Find all students that have more weekly hours in total than the average student. Try to simplify the query using the with construct. (Also consider students that do not attend any lecture).

Solution:

The following query determines the „busy students“:

```
select s.*
from Students s
where s.studNr in
  (select a.studNr
   from attend a, Lectures l
   where a.lectureNr = l.lectureNr
   group by a.studNr
   having sum(weeklyHours) >
      (select sum(cast(weeklyHours as decimal(5,2)))
            / count(distinct(s2.studNr))
       from Students s2
          left outer join attend a2
                      on a2.studNr = s2.studNr
          left outer join Lectures l2
                      on l2.lectureNr = a2.lectureNr));
```

By using the **with** construct or **case**, we can write a query that is much easier to read. First with **with**:

```
with TotalWeeklyHours as (
  select sum(cast(weeklyHours as decimal(5,2))) as CountWeeklyHours
  from attend a, Lectures l
  where l.lectureNr = a.lectureNr
),
TotalStudents as (
  select count(studNr) as CountStudents
  from Students
)
select s.*
from Students s
where s.studNr in (
  select a.studNr
  from attend a, Lectures l
  where a.lectureNr = l.lectureNr
  group by a.studNr
  having sum(weeklyHours)
        > (select CountWeeklyHours / CountStudents
           from TotalWeeklyHours, TotalStudents));
```

And here with **case**:

```
with WeeklyHoursPerStudent as (
select s.studNr,
  cast((case when sum(l.weeklyHours) is null
            then 0 else sum(l.weeklyHours)
       end) as real) as CountWeeklyHours
 from Students s
```

```
  left outer join attend a on s.studNr = a.studNr
  left outer join Lectures l on a.lectureNr = l.lectureNr
 group by s.studNr
)

select s.*
from Students s
where s.studNr in (select weeklyHours.studNr
                   from WeeklyHoursPerStudent weeklyHours
                   where weeklyHours.CountWeeklyHours
                         > (select avg(CountWeeklyHours)
                            from WeeklyHoursPerStudent));
```

Exercise 4

Write SQL queries to answer the following questions (pizza.db.in.tum.de).

1. What is the name of the restaurant with the id 41884?

```
select name
from general
where res_id = 41884
```

2. Which dish costs 23.4 €?

```
select name
from menu
where price = 23.4
```

3. How many menus cost between 10 € und 20 €?

```
select count(*)
from menu
where price between 10 and 20
```

4. Which dish is the priciest?

```
select name, price
from menu
where price = (select max(price) from menu)
```

5. How much does a Paulaner Spezi with 0,5 l costs at least, on average, at most? (Make sure you allow different menu names like 'Paulaner Spezi, 0,5' or 'Paulaner Spezi 0,5L'))

```
select 'Paulaner Spezi' as name, min(price), avg(price), max(price)
from menu
where name like '%Paulaner%Spezi%0,5%'
```

6. How many different dishes are available per size (S, M, and L)?

```
select size, count(*)
from menu
where size is not null
group by size
```

Exercise 5

Write SQL queries to answer the following questions (pizza.db.in.tum.de).

1. Which restaurant offers the cheapest Pizza Margherita?

```
select g.name, m.name, m.price
from menu m, general g
where m.res_id = g.res_id and m.name like '%Margherita%'
and price = (select min(price) from menu where name like '%Margherita%')
```

2. Which restaurant offers the best Pizza Margherita?

   (Based on the average restaurant rating)

```
with margherita_restaurants as (
  select distinct m.res_id, r.average as rating, m.name, m.price
  from menu m, rating r
  where m.name like '%Margherita%' and m.res_id = r.res_id
)

select distinct g.name
from margherita_restaurants m, general g
where m.res_id = g.res_id
and rating = (select max(rating) from margherita_restaurants)
```

3. At which restaurants should you rather not order when an Italian is present (because you can order Pizza Hawaii but not Pizza Regina)?

```
with pizzas as (
    select *
    from menu
    where name like '%Pizza%' or section like '%Pizza%'
)

select distinct name
from general
where res_id in (select res_id from pizzas where name like '%Hawaii%')
and res_id not in (select res_id from pizzas where name like '%Regina%')
```

4. Which restaurants spelled 'Mozzarella' wrong?

```
select distinct g.name, m.name, m.description
from menu m, general g
where m.description like '%Moz%'
and not m.description like '%Mozzarella%'
and m.res_id = g.res_id
```

Restaurant Schema