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

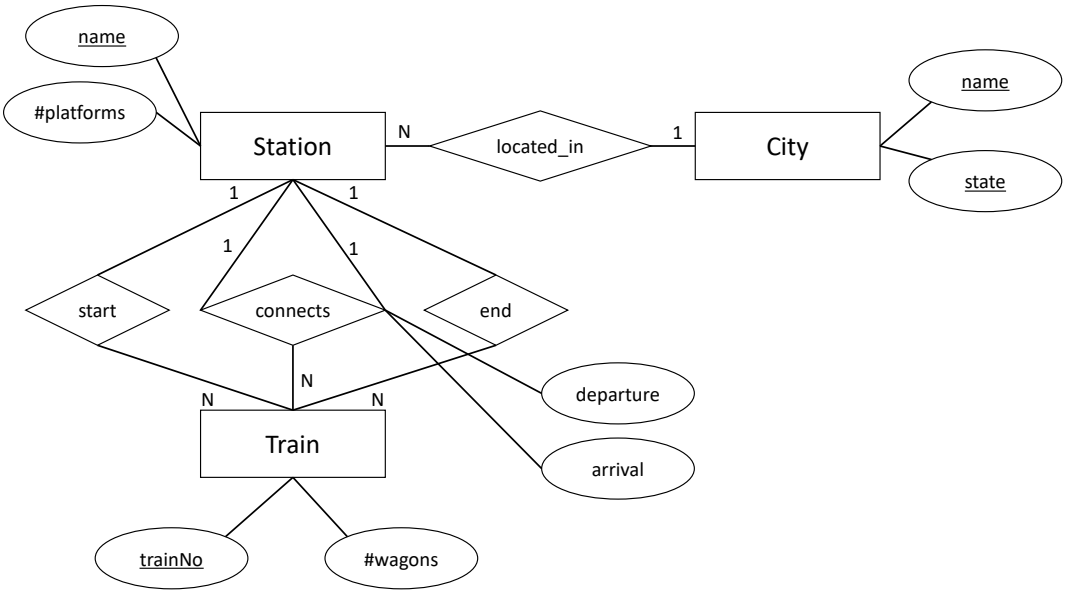
***Database System Concepts for Non-Computer Scientist* - WiSe 22/23**

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

Sheet 10

Exercise 1

Look at the following (familiar) ER-diagram and create SQL DDL statements to create the respective tables.



Lösung:

```
create table city (name varchar(50),
                   state varchar(50),
                   primary key(name, state)
);

create table station (name varchar primary key,
                      num_platforms int,
                      cityName varchar(50),
                      state varchar(50),
                      foreign key(cityName, state)
                             references city(name, state)
);

create table train (trainNo int primary key,
```

```
                    num_wagons int,
                    start varchar references station,
                    end varchar references station
);

create table connects (from varchar references station,
                       to varchar references station,
                       trainNo int references train,
                       departure date,
                       arrival date,
                       primary key(fromStation, trainNo)
);
```

Exercise 2

Write a SQL statement to create a view that gives an overview of the difficulty of each lecture. The difficulty of a lecture is defined as the sum of the weekly hours of that lecture and its direct predecessors. In our example instantiation of the university schema, the following query on your view should yield the result (only partially shown):

```
select * from LectureDifficulties;
```

lectureNr	title	difficulty
5216	Bioethik	6
4630	Die 3 Kritiken	4
...	...	...

Solution: Using a correlated subquery:

```
create view LectureDifficulties(lectureNr, title, difficulty) as (
select l.lectureNr, l.titel, l.weeklyhours
  + (select (case when sum(l2.weeklyhours) is null then 0
                  else sum(l2.weeklyhours) end)
     from Require r, Lectures l2
     where l.lectureNr = r.successor
       and r.predecessor = l2.lectureNr)
from Lectures l
);
```

Using a CTE/with-Statement:

```
create view LectureDifficulties(lectureNr, title, difficulty) as (
  with predecessor_sum as (
    select r.successor as lecturenr, sum(l.weeklyhours) as sum
    from require r, lectures l
    where l.lecturenr = r.predecessor
    group by r.successor
  )
  select l.lecturenr, l.title, (case when predecessor_sum is null
      then 0 else predecessor_sum end) + l.weeklyhours
  from lectures l left outer join predecessor_sum p on l.lecturenr =
       p.lecturenr
);
```

Exercise 3

Considering the following table definitions:

1)
```
create table A(a int primary key);
create table B(b int);
```

2)
```
create table A(a int primary key);
create table B(b int references A(a));
```

Assuming the cardinalities (number of tuples) of the relation A and B are $|A|$ and $|B|$, respectively. How many tuples are produced by the following queries. If no exact estimate is possible, give a range. Alternatively you can use mathematical set operations.

a) `select * from A, B;`

b) `select * from A join B on A.a = B.b;`

c) `select * from A left outer join B on A.a = B.b;`

d) `select * from A right outer join B on A.a = B.b;`

e) `select * from A full outer join B on A.a = B.b;`

Solution:

As ranges:

	1)	2)
a)	exactly: $\|A\| \cdot \|B\|$	exactly: $\|A\| \cdot \|B\|$
b)	between 0 and $\|B\|$	exactly: $\|B\|$
c)	between $\|A\|$ and $\|A\| + \|B\| - 1$	between $\max(\|A\|, \|B\|)$ and $\|A\| + \|B\| - 1$
d)	exactly: $\|B\|$	exactly: $\|B\|$
e)	between $\max(\|A\|, \|B\|)$ and $\|A\| + \|B\|$	between $\max(\|A\|, \|B\|)$ and $\|A\| + \|B\| - 1$

1) A.a is a primary key, B.b is not referencing A.a
   a) The query produces a cross product of A and B.
   b) The query joins A and B and produces pairs of matching tuples. Since B.b is not referencing A.a, B.b could contain only values that do not occur in A.a. In that case we get no results. If all tuples of B match elements of A, we get |B| elements.
   c) The query left outer joins A and B. If an element in A does not find a partner in B, it is still added to the result with B.b = null. If a tuple in A finds multiple partners in B, all combinations are added. So, we get at least |A| result tuples even if B does not contain any matches. If all elements in B.b match with the same A.a, we have the other extreme: |B| result tuples for the one A.a that matches all elements in B.b and |A|-1 result tuples for the other tuples in A.
   d) The query right outer joins A and B. If an element in B does not find a partner in B, it is still added to the result with A.a = null. Since A.a is a primary key, each B.b can only be equal to one A.a value and therefore no duplicates will be produced which gives us exactly |B| result tuples.
   e) The query full outer joins A and B. Elements of both sides are still added to the result with the respective opposite site set to null. We get at least as many elements as we have in |A| and in |B|: max(|A|, |B|). If we don't have any matches A.a = B.b, then all elements are inserted with the opposite site being set to null which gives us |A| + |B| elements in total.

2) A.a is a primary key, B.b references A.a
   a) The query produces a cross product of A and B.
   b) Since B.b is referencing A.a, there exists for each B.b an entry A.a with the same value, therefore we get exactly |B| result tuples.
   c) In the minimum case, we have either more elements in A and we find at most one match in |B| which gives us |A| elements, or we have more elemts in B and have for every element in A at least one match in B, then we get |B| result tuples. Since both is possible we have at least max(|A|,|B|) result tuples. If all elements in B match the same A.a this gives us |B| result entries and |A|-1 for the rest of the entries in A that have no match in B.
   d) Since each element in B has to find a partner in A, the right outer join will not add any additional result entries with A.a being null, so we get |B| elements like for the inner join.
   e) If A contains more elements and each element has at most one match in B, we get |A| elements. If we have more elements in B and all elements of A have at least one match, we get |B| elements. In total, we will get at least max(|A|,|B|) elements. Since each element in B has to find a match in A, we get the same max as for the left outer join: All elements of B are the same and matched to one element in A producing |B| result tuples, and |A|-1 result tuples for the rest of the elements in A.