EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Advanced SQL
Winter Term 2025/2026
Prof. Torsten Grust, Tim Fischer, Björn Bamberg
WSI — Database Systems Research Group

Assignment 11

Hand in this assignment until **Friday, January 23th 2026, 12:00 pm** at the latest.

🤔 Running out of ideas?
Are you hitting a roadblock? Are some of the exercises unclear? Do you just need that one hint to get the ball rolling? Refer to the `#forum` 🌐 channel on our Discord server—maybe you'll find just the help you need.

💡 Note
All the queries you write for this assignment require *recursive queries*.

Task 1: Travel by Car

Recursive CTEs | Directed Graphs

You and your friends are planning several road trips. Each person starts in a different city and ends in a different destination, and everyone is driving an electric car with a specific maximum battery capacity that requires regular recharging. You are given a map (in 📄 `travel.sql`) of the roads connecting the cities, including the distances between them.

`trips`

driver	start	end	max_batt
'Nils & Raphael'	'Koeln'	'Wuerzburg'	200
'Nicolas'	'Freiburg'	'Lindau'	20
'Timo'	'Landshut'	'Passau'	100
'Maren'	'Buggingen'	'Tuebingen'	250
'Nadja & Simion'	'Freiburg'	'Ulm'	100

A Now consider the table `trips`, which contains multiple trips with different starting cities and destination cities. **Write a SQL query** that computes, for each trip in `trips`, the length of the shortest possible path in terms of total distance from the starting city to the destination city. (For now, ingore `max_batt` and assume that the car has unlimited battery capacity.😉)

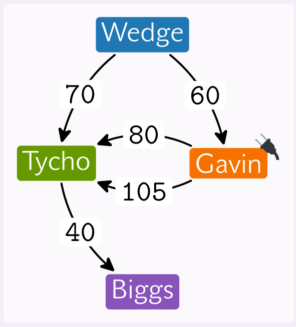
🤔 Unsure if you understood the task?
If not sure that you understood the task, take a look at this result, it shows how long the shortest paths are for the example instance of `trips` (including start and end points).

start	end	distance
'Koeln'	'Wuerzburg'	190
'Freiburg'	'Lindau'	130
'Landshut'	'Passau'	80
'Buggingen'	'Tuebingen'	160
'Freiburg'	'Ulm'	180

B To plan the trips, it is not sufficient to know only the shortest path, you must also ensure that the electric car can actually reach the destination. One unit of battery charge allows a car to travel one unit of distance, so if a vehicle runs out of charge while traveling between cities, the trip ends immediately. Fortunately, some cities are equipped with recharging stations. (To see if a city has recharging stations, check the `charger` column in the `cities` table in 📄 `travel.sql`.)

Navigation by Example

To give you an idea of how this navigation works, let's take a look at a smaller example — see the graph on the right. The graph depicts the roadways connection the four cities Wedge, Gavin, Tycho, and Biggs — Gavin being the only one with a charging station. Each roadway has a fixed length and goes only one way.



Now, starting our journey in Wedge, let's see which cities we can reach with 100 units of charge. We need 70 units of charge to reach Tycho, which leaves us with 30 units of charge. The only roadway leaving Tycho is the one towards Biggs, but that road takes 40 units of charge to traverse, so that ends our journey. If, on the other hand, we leave our starting city Wedge, following the roadway to Gavin, we can completely recharge at the charging station there. With a full charge we can then take 80 unit long roadway from Gavin to Tycho, leaving us with only 20 units of charge — which is also not enough to reach Biggs.

The result of your query should thus contain our starting city Wedge, and the cities we can reach with our electric car, Tycho and Gavin — with the city Biggs being the only one excluded, as its to far away to reach with the charging options we have.

Write a SQL query that determines, for each trip listed in `trips`, the shortest feasible path *from the starting city to the destination city*, taking into account the car's battery capacity and the availability of recharging stations along the route. The query should return a table with schema (`driver, path`), where `path` represents the ordered list of *all cities* visited from the starting city to the destination city. If no feasible path exists for a given driver, that driver should not be included in the result.

Unsure if you understood the task?

If not sure that you understood the task, take a look at this result, it shows the shortest path for each trip in `trips` (including start and end points).

driver	path
'Maren'	['Buggingen', 'Laufen', 'Freiburg', 'Tuebingen']
'Nils & Raphael'	['Koeln', 'K.-Lautern', 'Darmstadt', 'Wuerzburg']
'Simion'	['Freiburg', 'Konstanz', 'Lindau', 'Biberach', 'Ulm']
'Timo'	['Landshut', 'Passau']

Some bonus points on the finish line!

Solve the following task to gain up to 3 bonus points.

You have not yet decided on a destination, but want to determine which cities are possible options. You are driving an electric car with a maximum battery capacity of 150 units and you are starting your journey in *Tuebingen*. Taking into account battery capacity constraints and the availability of recharging stations, write a SQL query that returns a table containing *all the cities* reachable from *Tuebingen*.

Task 2: Stormtroopers On Patrol

RECURSIVE CTES

The Rebel Alliance needs your help to infiltrate an Imperial checkpoint. They require information on all safe positions within a rectangular room of arbitrary size. A position is considered unsafe if a stormtrooper could see it at any point during their patrol.

room

x	y	dir
1	1	'E'
3	1	'S'
4	1	'S'
1	2	'W'
2	3	'S'
3	3	'W'
4	3	'X'
2	4	'E'
3	4	'X'

On the right you can see an example for table room, each row corresponds to a position (x, y) in the room and specifies the direction a stormtrooper will take when reaching that position. The values `'E'`, `'W'`, `'N'`, and `'S'` indicate movement directions, while `'X'` marks the end of a patrol path.

Stormtroopers always begin their patrol at the north-west corner of the room at position $(1,1)$ and continue until they reach a position marked with `'X'`, which is never placed at $(1,1)$. The stormtroopers move in the direction they are looking. When they reach an indicated direction change, they split up—some continuing along in the direction they are currently moving and some in the indicated direction. Assume that there are always enough stormtroopers patroling to do so.

y \ x	1	2	3	4
1	→		↓	↓
2	←			
3		↓	←	×
4		→	×	

You can see a visualisation of the room table above. In this scenario, the stormtroopers start their patrol at position $(1,1)$ and immediately move east. When reaching positions $(3,1)$ and $(4,1)$, they may turn south, resulting in two separate patrol paths. Both paths are taken and terminate at positions $(3,4)$ and $(4,3)$, respectively.

Write a SQL query that returns a table with schema `(x, y)`, containing all positions in the room that are safe from stormtrooper surveillance.

🤔 Unsure if you understand the task?

If not sure that you understand the task, take a look at this result, which your query should produce for the example instance of room.

x	y
1	2
2	2
1	4