EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

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

Assignment 12

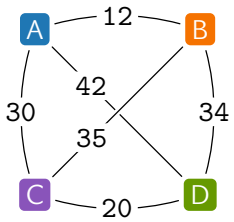
Hand in this assignment until **Friday, January 30th 2026, 12:00 pm** at the latest.

> 🤔 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.

Task 1: Traveling Salesman

RECURSIVE CTEs

In this task we will be taking a look at one of the classics in computer science: the *Traveling Salesman Problem* (TSP) 🌐. In laymen terms, the story reads as follows: A traveling salesman has a list of cities they need to visit, a map listing the connections between all of the cities, and wants to get back home early. So your task is to find the shortest route the salesman can take on which they would visit all cities and get back home.



The map is given in the form of a weighted, undirected, complete graph which we encode as the edge table `roads`—see the example to the right. To rehash, such an edge-based encoding lists all edges in a graph in terms of start points `here`, end points `there`, and edge weights `dist`. Since we are talking about an undirected graph here, the terms "start point" and "end point" can be misleading; all edges are two-way roads between the cities on the map, *i.e.*, they are bidirectional and have the same length in both directions.

`roads`

here	there	dist
A	B	12
A	C	30
A	D	42
B	C	35
B	D	34
C	D	20

A **Create a temporary table** called `all_roads`, that represents all directed edges implied by the directed edges in `roads`.

B **Create a temporary table** called `cities`, which extracts all unique city names in `roads` into a relation with a single column `city`.

C **Write a SQL query**, using the views `all_roads` and `cities` from the previous subtasks, which finds the shortest cyclic path that visits all cities in `cities`. This path must *start and end in the same city* and must *visit every other city exactly once*. Your query should output both the path as a list of the nodes along the path, in the order traveled, and also the path's overall length.

> 🧭 Is your heading right?
> You know you are on the right track if your query outputs the result to the right for the example instance of `roads` shown above.

path	dist
[A, C, D, B, A]	96

Task 2: Transport by Plane

RECURSIVE CTEs

You work in the air freight department of a shipping company, where you manage a fleet of `planes`, each with an individual maximum `capacity`. Flying a plane from one place to another can be quite expensive, especially if you do so with little cargo. So your boss asks you to write up a report with the packings of `items` that make the most of each planes load capacity.

`planes`

id	capacity
1	25000
2	19000
3	27000

`items`

id	weight
1	7120
2	8150
3	8255
4	9051
5	1220
6	12558
7	13555
8	5221
9	812
10	6562

Write a SQL query which computes the packings for your report. For each packing, your boss wants to see the plane's id, a list of included items, the total weight of the packing and also the plane's load capacity.

> 🏗️ Keep it simple!
> You can consider these packings to be independent from one another. Put differently, some items may appear in multiple packings. 😉

All packed up?

For the example instances of `planes` and `items` above, your query output the packings you see to the right.

plane	items	weight	capacity
1	[2,3,5,9,10]	24999	25000
2	[5,6,8]	18999	19000
3	[1,5,6,8,9]	26931	27000

Task 3: Heat Flow

RECURSIVE CTES · WINDOW FUNCTIONS

In the lecture we have dabbled with cellular automata in the form of *Conway's Game of Life* and even some physics simulations. In this task we pick up where we left of with those simulations and turn to simulating the spreading of heat energy through a 2D piece of material.

`heat`

x	y	h
1	1	10
1	2	0
1	3	0
2	1	0
2	2	0
2	3	0
3	1	0
3	2	0
3	3	0

Much like in Game of Life, our cellular automata consists of a grid of cells, each holding a value —this time the value in question is heat energy. We represent this grid as a three column table `heat`, with two of those columns identifiying the grid position of a cell, *i.e.*, (`x`, `y`), and the third the heat energy `h` contained therein. To keep things simple, we will assume that the heat energy contained in a cell can never be negative.

At each time step during our simulation we update the cell grid by applying the function $\Delta \mathtt{h}(\mathtt{x},\mathtt{y})$, defined below, to each cell. This function describes the change in heat energy in a cell based on the heat energy already in the cell $\mathtt{h}(\mathtt{x},\mathtt{y})$ and the heat energy currently contained in its immediate neighbors, *i.e.*, $\mathtt{h}(\mathtt{x}\pm 1,\mathtt{y})$ and $\mathtt{h}(\mathtt{x},\mathtt{y}\pm 1)$. The conductivity C of the material dictates how fast heat energy can flow from one cell to the next.

$$\Delta\,\mathtt{h}(\mathtt{x},\mathtt{y}) = C \cdot \big(\mathtt{h}(\mathtt{x}-1,\mathtt{y}) - 2\cdot\mathtt{h}(\mathtt{x},\mathtt{y}) + \mathtt{h}(\mathtt{x}+1,\mathtt{y}) + \mathtt{h}(\mathtt{x},\mathtt{y}-1) - 2\cdot\mathtt{h}(\mathtt{x},\mathtt{y}) + \mathtt{h}(\mathtt{x},\mathtt{y}+1)\big)$$

If we take the example instance of `heat` shown above and visualize the first 6 iteration steps using a conductivity of $C = {}^{1}/_{10}$, we get the following image. At each step the heat energy flows away from the top-left corner —quickly at first and then slowing down once the energy is spread between many cells.

Complete the SQL view `simulated` in `heat.sql` with a query that simulates the first `n` iterations heat energy flowing through the material, starting with the configuration defined in `heat`. Your query should output the final grid configuration configuration your simulation reached after all `n` iterations.

Hot or cold?

You can use the visualizations above to check if you are on the right track with your solution.