A Problem to Solve

We can list employees and salaries — but not who supervises whom

EmployeeID	Name	Salary
1	Alice Nguyen	$145,000
2	Brian Kim	$98,000
3	Carla Ruiz	$92,000
4	David Chen	$47,000
5	Frank Osei	$45,000
6	Henry Adams	$48,000

The Problem

Enhance the database to record the supervisor for each employee. Right now, Employee has no way to express "who does this person report to?"

Possible approaches:

Option A: Add a new table?

e.g. a Supervises(EmployeeID, SupervisorID) table listing who reports to whom, separate from Employee.

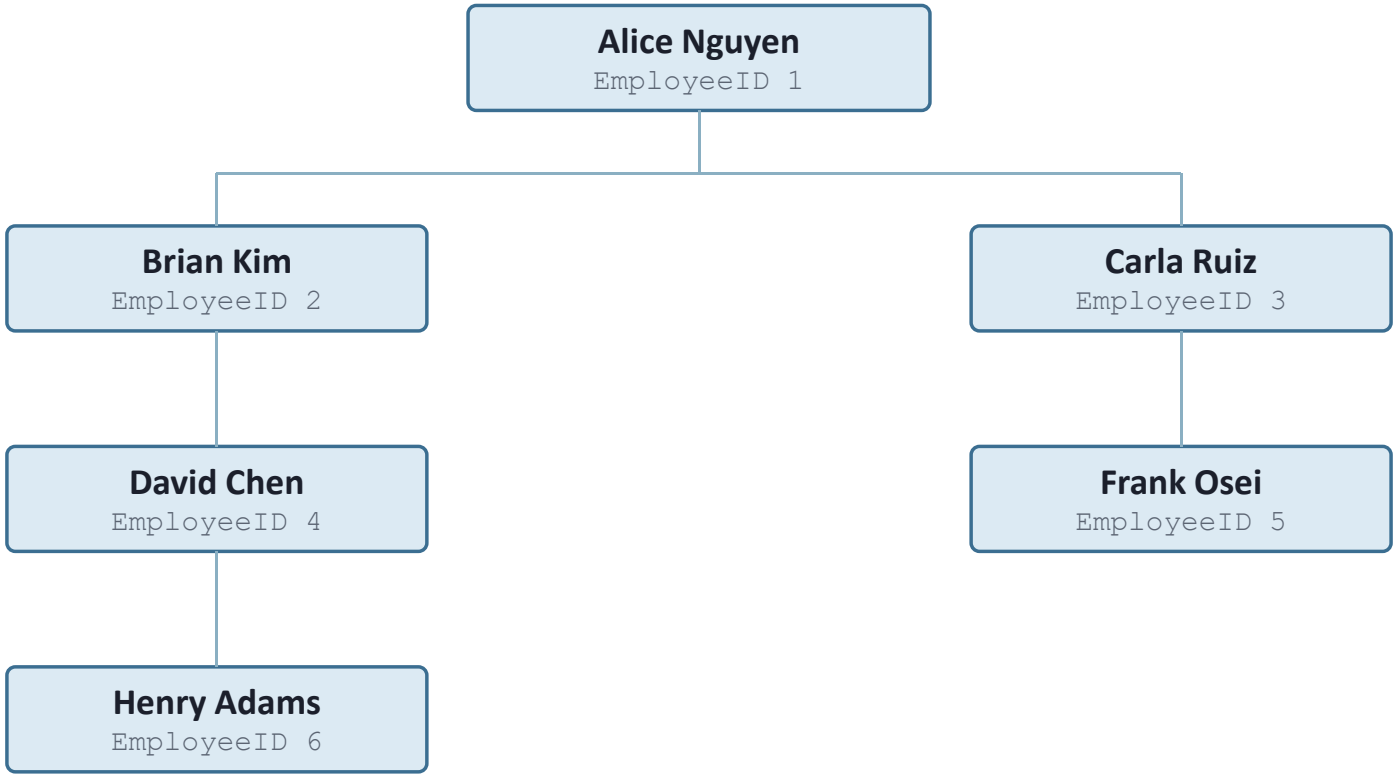
Option B: Add a new column to Employee?

e.g. a SupervisorID column directly on Employee, referencing another EmployeeID in the same table.

Either works — next, we'll try adding the column directly.

The Employee Hierarchy

SupervisorID turns the flat Employee table into a tree of who reports to whom



Alice sits at the top with no supervisor; each line below shows a SupervisorID relationship — the exact structure a self join reconstructs.

A New Table: Employee

Employees can supervise other employees — SupervisorID points back into this same table

EmployeeID	Name	Salary	SupervisorID
1	Alice Nguyen	$145,000	—
2	Brian Kim	$98,000	1
3	Carla Ruiz	$92,000	1
4	David Chen	**$47,000**	2
5	Frank Osei	**$45,000**	3
6	Henry Adams	**$48,000**	4

Self-Referencing Table

SupervisorID doesn't point to a different table — it points to another row's **EmployeeID** in this ***same*** table. That's what makes a self join possible.

Alice Nguyen
has no supervisor (SupervisorID is blank) — she's the top of the org chart.

Introducing Table Aliases

Give a table a short nickname so you can reference it more easily

Employees earning less than $50,000:

```
SELECT e.Name, e.Salary FROM Employee e WHERE e.Salary < 50000;
```

`Employee e` gives the table a short alias, **`e`**. From then on, **`e.Name`** and **`e.Salary`** mean “the Name/Salary column, from the table I called e.”

For a single table like this, an alias is just a convenience. **It becomes essential** once you need to reference the ***same*** table twice in one query — which is exactly what a self join does.

Self Join: Employee and Supervisor

List each employee and their supervisor's name

```
SELECT e.Name AS Employee, s.Name AS Supervisor
FROM Employee e JOIN Employee s ON e.SupervisorID = s.EmployeeID;
```

Employee (one physical table)

read as two different roles in the same query:

```
Employee AS e  — "the employee"
Henry Adams, SupervisorID = 4
```

```
Employee AS s  — "the supervisor"
David Chen, EmployeeID = 4
```

joined on: e.SupervisorID = s.EmployeeID

Result row: Employee = Henry Adams, Supervisor = David Chen

Full Result

Employee	Supervisor
Brian Kim	Alice Nguyen
Carla Ruiz	Alice Nguyen
David Chen	Brian Kim
Frank Osei	Carla Ruiz
Henry Adams	David Chen

Visualizing the Self Join

Same query, same table — two copies of Employee, joined to each other

```
SELECT e.Name AS Employee, s.Name AS Supervisor
FROM Employee e JOIN Employee s ON e.SupervisorID = s.EmployeeID;
```

Employee AS e

EmployeeID	Name	SupervisorID
1	Alice Nguyen	—
2	Brian Kim	1
3	Carla Ruiz	1
…	…	…

Employee AS s

EmployeeID	Name	SupervisorID
1	Alice Nguyen	—
2	Brian Kim	1
3	Carla Ruiz	1
…	…	…

Join Result

Employee	Supervisor
Brian Kim	Alice Nguyen
Carla Ruiz	Alice Nguyen
…	…