That pattern — where a set of attributes like (employeeID, employeeName, sectorID, sectorName) repeats for each manager — is exactly what's meant by a repeating group in normalization theory.

To remove it (and move toward 1NF and beyond), you would:

Put managers in their own table (managerID, managerName, area).

Put employees in their own table (employeeID, employeeName, managerID as FK).

Put sectors in their own table (sectorID, sectorName).
why we didn't add empolyee id as foreign key in sectors table? what is the rule?

Excellent question — this gets to the heart of **how we decide where to place foreign keys** in normalization.

1. The guiding rule

- A **foreign key** is placed on the side of the relationship where **many** instances reference **one** instance.
- In other words:
  - The **child table** holds the foreign key.
  - The **parent table** keeps its own primary key only.

2. Applying it to Employee–Sector

- **Business meaning**: each employee belongs to one sector, but each sector can have many employees.
- That's a **1 : M relationship** (one sector → many employees).
- Therefore:
  - `sectorID` is the **primary key** in the Sector table.
  - `sectorID` becomes a **foreign key** in the Employee table.

- We do **not** put `employeeID` in the Sector table, because that would imply each sector can only have one employee (1 : 1), which is not true.

3. General rule of thumb

- **One-to-Many (1:M)** → foreign key goes on the "many" side.
- **One-to-One (1:1)** → foreign key can go on either side, depending on business rules.
- **Many-to-Many (M:N)** → create a separate junction table with a composite key.

4. Why not EmployeeID in Sector?

If you put `employeeID` inside Sector, you'd be saying:

- "This sector belongs to this one employee."
- That collapses the relationship into 1:1, which is wrong.
- You'd also need multiple rows for the same sector if it had multiple employees → redundancy and anomalies.