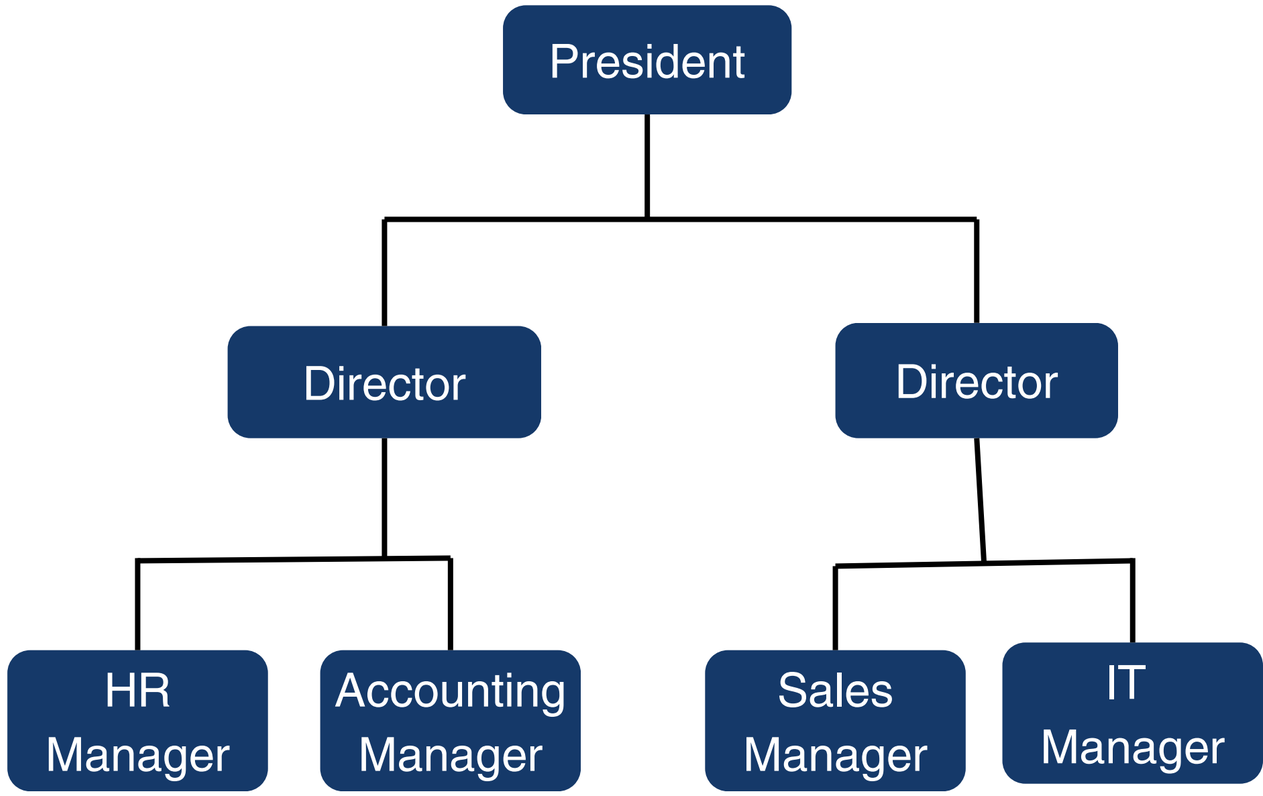
Hierarchical Data

Agenda

Hierarchical Data

- Hierarchical data refers to data that is organized in a **tree-like structure.**
- This type of data structure allows for easy representation of entities that have a natural **"parent-child" relationship.**
- Each parent record can have multiple child records, but each child record can have only one parent.



Problem Statement

Organizing Company Employees

- Imagine you're working for a large corporation with many employees, and the company has a **complex hierarchical organizational structure**.
- The company consists of different departments, each with its own **hierarchy**, such as managers, team leads, and regular employees.
- Your task is to design a database system that stores the **organizational structure** of the company

Ways to represent Hierarchical Data

Adjacency List Model

Each record contains a reference (often a foreign key) to its parent record

Path Enumeration Model

Each record stores the path from the root to itself, represented as a delimited string or an array.

Adjacency List Model

Structure:

- In the Adjacency List model, each record contains a reference (often a foreign key) to its parent record.
- Each table has a column that stores the **parent ID** or **parent reference**. This column is a foreign key that points to the same table.
- A recursive query to retrieve all descendants of a given node
- Efficient for representing hierarchical data in a relational database and easy to implement with basic SQL operations.

Example

Employee_ID	Designation	Parent_id
1	CEO	null
2	Director	1
3	Sales_Manager	2
4	Sales_Person	3

To retrieve all employees under Sales_Manager, you would use a query like:

```
SELECT * FROM Employees WHERE Parent_ID = (SELECT Employee_ID FROM
Employees WHERE Name = 'Sales_Manager');
```

Path Enumeration Model

Structure:

- In the Path Enumeration model, each record stores the path from the root to itself, represented as a delimited string or an array.
- This path contains the IDs of all parent records leading to the current record.
- The table has a column that stores the path (a string, typically with delimiters) from the root to the current record.
- It’s easy to query for all descendants or ancestors using LIKE or substring matching, which can be more efficient than recursive queries in the Adjacency List model.

Example

Employee_ID	Designation	Path
1	CEO	/1/
2	Director	/1/2/
3	Sales_Manager	/1/2/3/
4	Sales_Person	/1/2/3/4/

To retrieve all employees under Sales_Manager, you can query the Path column:

```
SELECT * FROM Employees WHERE Path LIKE '/1/2/3/%';
```

Comparison between the models

Feature	**Adjacency List Model**	**Path Enumeration Model**
Storage	Stores parent references (foreign keys)	Stores full path as a string
Querying Descendants	Requires recursive queries (CTEs)	Simple LIKE queries on the path
Querying Ancestors	Recursive or multiple joins	Simple LEFT match on path
Complexity of updates	Easy to update (just modify parent)	More complex (need to update paths of descendants
Easy of use	Simple and intuitive	More efficient for certain queries but harder to maintain

Thank You