# Data Modeling with Snowflake: A Practical Guide to Accelerating Snowflake Development Using Universal Data Modeling Techniques

## Part 3: Solving Real-World Problems with Transformational Modeling

### Chapter 16: Modeling Hierarchies

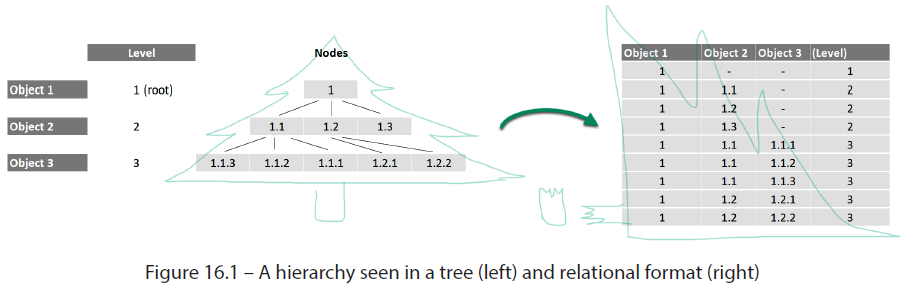
* In the previous chapter, we learned about methods + techniques for exploring semi-structured data in Snowflake
* While not everyone works w/ operational web or app data such as JSON, there is *another* type of **(semi-)structured data that exists across *all* organizations**: **hierarchies**
* Every company operates w/ hierarchical entities such as organization levels (tiers of management and the reporting relationships between managers + employees) or calendar dimensions (rollups of days, months, fiscal periods, and years)
* Whether formally maintained or naturally occurring, **hierarchies are used w/in organizations to organize entities into meaningful groups + subgroups to facilitate rollups or drill-downs in data analysis**
* Besides aiding in the analysis of facts, **hierarchies themselves can be examined to help organizations understand how they are structured, how they function, + how they can improve their performance by eliminating operational bottlenecks**
* This chapter will cover the various ways hierarchical data is modeled in RDBs + Snowflake-specific techniques that enable its rapid analysis + conversion
* Drawing upon the techniques covered in previous chapters, we will learn how to anticipate changes in hierarchical data + create models that allow for easy maintenance + impact analysis
* Main Topics:
* Understanding hierarchies + how they are used in DWs
* Distinguishing between the various hierarchy types
* Modeling techniques for maintaining each type of hierarchy
* Snowflake features for traversing a recursive tree structure
* Handling changes in hierarchy dimensions

#### Technical Requirements for Local Snowflake Work

* The scripts used to instantiate + load the examples in this chapter are available in the following GitHub repo: <https://github.com/PacktPublishing/Data-Modeling-with-Snowflake/tree/main/ch16>

#### Understanding and Distinguishing Between Hierarchies

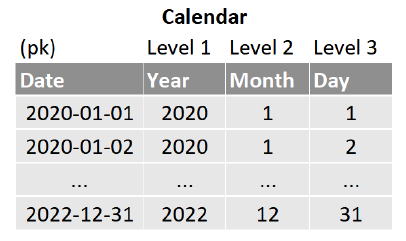
* A **hierarchy** is **a system in which people, objects, or concepts are organized into a tree-like structure, w/ each level representing a different category or grouping of data**
* In modeling, **hierarchies can be thought of as a series of descending one-to-many relationships**
* At the top of the hierarchy sits the **root node**, which contains **child nodes**, each **representing a subcategory** of the **data** **contained in the parent node** + may, in turn, have its own child nodes
* This arrangement of nodes + their relationships is often called a **tree structure** or **tree** **diagram**
* If to proverbially chop down such a tree + observe it laterally, we’d see the hierarchy in its *relational* form



* Hierarchies fall into **3 general categories depending on the variability in their levels**

##### 1) Fixed-Depth Hierarchy

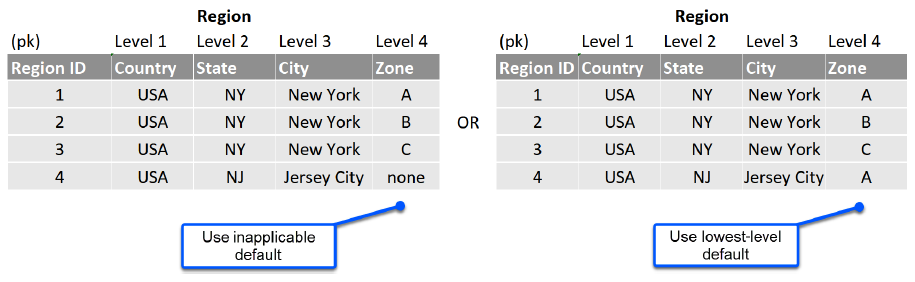
* A **fixed-depth hierarchy** has a **pre-defined + agreed-upon number of levels**
* Such a hierarchy is **simple to model + analyze** b/c it is just a **relational table w/ individual attributes for each level**
* Take a calendar dimension as an example: the day attribute rolls up into a month, which rolls up into a year
* It can be expressed as a simple table w/ 3 attributes, each corresponding to a separate level:



* In some cases, though, hierarchy levels are NOT completely fixed

##### Slightly-Ragged Hierarchy

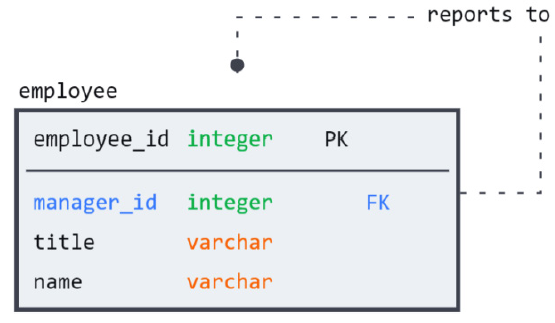
* A **slightly ragged** (or **slightly variable**) **hierarchy** also has a **fixed depth**, but **not *all* members occupy *all* the levels**
* Think of a region hierarchy that contains cities of various sizes
* A city such as *New York* is big enough to be broken down into smaller zones for analysis, while its neighbor, *Jersey City*, is tiny, + further segmentation is not useful
* B/c ***slightly* ragged hierarchies** have a fixed **depth**, they ***can* be modeled using the previously discussed structure**
* **Variable attribute levels can then be populated based on *business rules***
* The following 2 business rules are commonly used for fitting slightly ragged hierarchies into a fixed depth:
* **Mark inapplicable values:** When a node does not have a value for a given level of the hierarchy, use a **default value** (e.g., *none* or *N/A*) to label it accordingly
* This approach **ensures that aggregates + analytical filters treat “not applicable” values *differently* from those that *are* labeled**
* **Use the lowest level**: When a node does not have a value for a given level of the hierarchy, label it w/ the **lowest value of that level by default**
* This method allows all nodes in the hierarchy to be grouped under each level’s corresponding rank



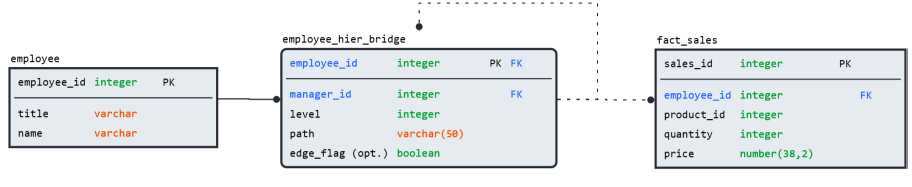
* **Fixed and slightly ragged hierarchies can be analyzed using standard relational methods**
* However, there are hierarchies whose nodes can vary wildly in the number of levels + children, and a standard relational approach is no longer viable

##### Ragged-Hierarchy

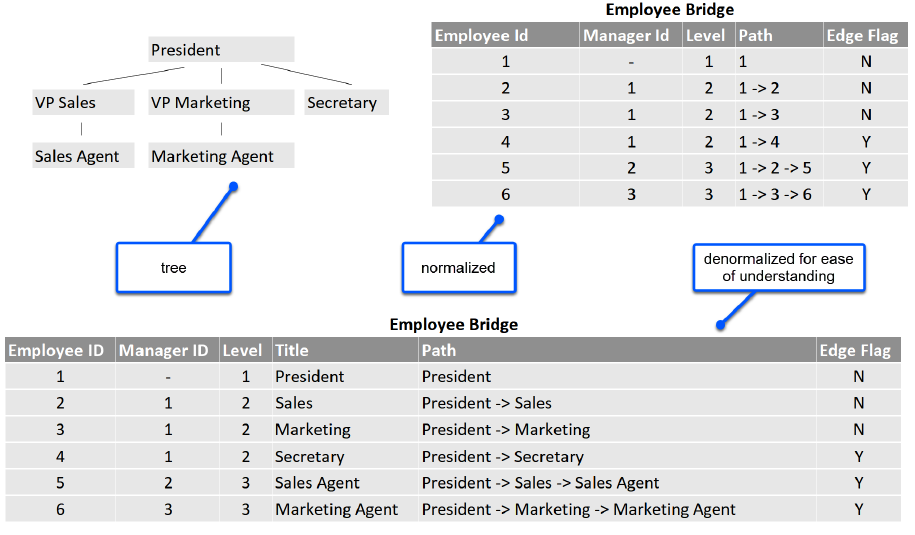
* A **ragged** (a.k.a. **variable-depth**, **unbalanced**, **uneven**) **hierarchy** contains **levels or categories that do not have a uniform number of members**
* One that most people will recognize is an organization hierarchy, where the highest level may have only one member (CEO), while the next level may have several VP’s w/ their own direct reports, + the CEO’s lone secretary
* Ragged hierarchies are **difficult to analyze because they are of indeterminate depth + do not fit neatly into a relational structure**
* A ragged hierarchy **does not have separate attributes corresponding to fixed levels but instead uses a self-referential PK/FK relationship to link parent + child nodes**
* While the parent/child relationship *can* be recorded in a single record, as seen in the following example of the recursive relationship between an employee + manager, analyzing such a hierarchy takes some effort



* **Snowflake provides several nifty extensions to ANSI SQL that allow users to traverse a ragged hierarchy’s recursive parent/child relationships at runtime**
* These same features can also aid in generating **a flattened representation of the hierarchy**, known as a **bridge table**, which **contains a row for every possible path in the ragged hierarchy and, as a standalone dimension, allows for alternate versions using familiar SCD patterns**
* By removing manager\_id from employee in the example above + moving it to a bridge table, both dimensions can change w/out impacting one another
* Most importantly, the **hierarchy can be analyzed or joined to corresponding entities using traditional relational methods**



* A **bridge table makes it easy to read the paths + levels of a ragged hierarchy**
* The model in this example also includes a column to identify **edges** (**nodes w/out any children**)
* An edge flag can simplify queries required to answer questions such as “*do all VPs have direct reports?“*
* The following is an example of a simple tree hierarchy + its representation as a bridge table (a denormalized version is provided for easier understanding):



* Now that we are familiar w/ the types of hierarchies + their structure, let’s learn how to handle changes + create a bridge table by traversing a recursive relationship

#### Maintaining Hierarchies in Snowflake

* As we have seen, **a hierarchy is a grouping of classifications + relationships**
* It is **an adjacent but separate dimension to the entity that it organizes**
* **As a standalone dimension, hierarchies are subject to the same changes in business rules as any master data**
* However, before we attend to change-tracking, we **must get comfortable traversing a ragged hierarchy’s recursive relationships**
* Once again, **Snowflake has native features that allow us to easily map out the paths + levels of a ragged hierarchy**

##### Recursively Navigating a Ragged Hierarchy

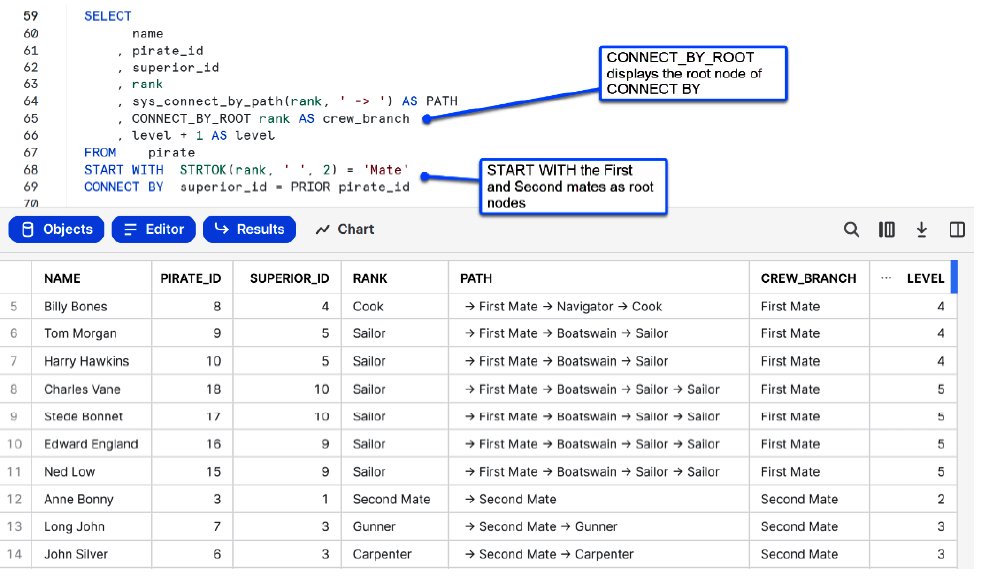
* You have been put in charge of the pirate frigate Queen Anne’s Revenge
* Its logbook shows all crew members, ranks, + direct reports



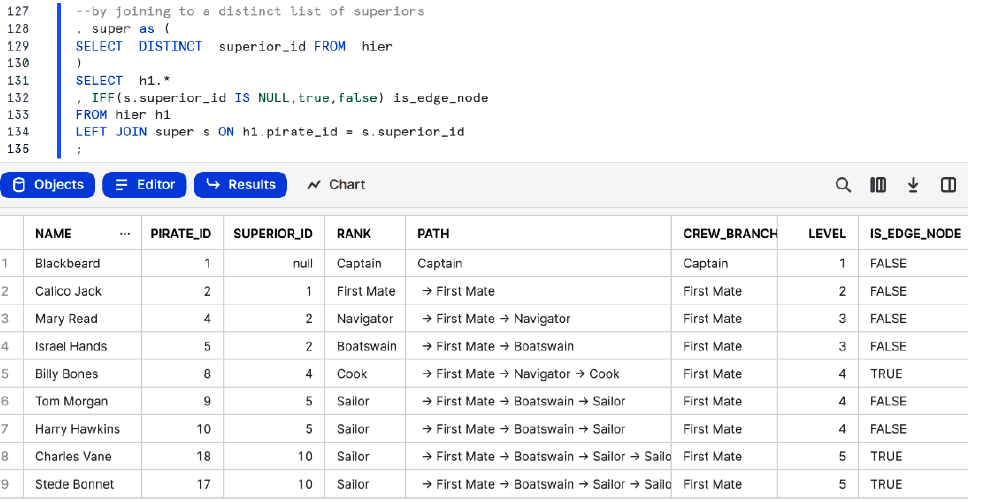
* As you review the records, you realize this list makes it difficult to make sense of the complex web of the **variable-depth relationships** of the crew
* To understand the ship’s chain of command, you **need a way to traverse the hierarchical tree structure**
* To accomplish this, **Snowflake offers the CONNECT BY** **clause**
* The **CONNECT BY clause *follows* a parent-child relationship *between rows* in a table**
* <https://docs.snowflake.com/en/sql-reference/constructs/connect-by>
* **Each row has a parent identified by a FK in the same table**
* **CONNECT BY uses a recursive query approach to traverse this hierarchical structure**, which **starts w/ a root row (which can be specified using the START WITH** **predicate) + then recursively traverses the structure by finding all children of the current row + repeating the process until it reaches the bottom of the tree**
* **W/in the CONNECT BY clause**, Snowflake uses a function called **SYS\_CONNECT\_BY\_PATH** to **generate the path**
* The **output is a string column**, which **shows the path from the root row to each row in the tree**
* This column is a **concatenated string of the values in the column that is used to establish the hierarchy**
* Use the code provided in the repository for this chapter to load the pirate table + observe the parameters in the CONNECT BY statement that generate the pirate hierarchy:



* **Using the CONNECT BY query result, we can easily determine how many levels + branches exist in the crew hierarchy**
* The pirate crew is as complex as any organization, containing 5 levels of depth + comprising 2 distinct branches (those of First Mate and Second Mate)
* If we wanted **to *label* these branches to make future analysis easier, we could use the CONNECT\_BY\_ROOT function to return the top-level root node of the branch we are traversing**
* **NOTE: This does NOT return our top-level of “Captain”**



* Now that we have multiple branches, add an **edge flag** to **identify members w/out direct reports**
* To do this, use LEFT JOIN for a distinct list of SUPERIOR\_ID values (e.g., pirates *with* direct reports) from the query result we obtained previously
* Any NULL value returned by the right table identifies an edge:



* Now that we **have all the details needed for the bridge table**, we can create it as a **standalone dimension for future analysis + subsequent changes**
* **Keeping the hierarchy separate from the source entity ensures that each can be maintained without impacting the other**
* Also, **we can use familiar modeling techniques to track changes in hierarchies of any kind**

##### Handling Changes

* **A hierarchy can be updated + modified as a standalone dimension *without* impacting the underlying entities**
* Imagine if hierarchy details were stored alongside an already large Type-2 employee dimension
* Every title or manager change would generate a massive cascade of new records
* Instead, we **can use SCD models to maintain changes to hierarchy dimensions efficiently and according to business needs**
* Suppose that region groupings (or pirate stations) changed regularly
* Our **organization may be interested in the historical (or proposed) changes, for planning and comparison purposes**
* **Some common historical tracking scenarios that can occur and how to treat them**:
* **History not required:** **Use Type 1 SCD**
* If tracking historical changes to the hierarchy is *not* required, then a Type 1 strategy of *overwrite* is the simplest way to handle changes
* **Compare last: Use Type 3 SCD**
* When a change *is* made (or proposed) to an existing hierarchy, an organization may wish to **compare the impact of the new grouping on the existing facts**
* For example, what if we grouped *New York City Zones A* and *B* but wanted to compare sales volumes to the previous classification?
* **Adding a Type 3 last value column to the hierarchy dimension allows the business to analyze the before-and-after impact using the same join they’d normally use to tie the fact to the hierarchy dimension**
* **Compare point-in-time: Use Type 2 SCD**
* When **historical versions of the hierarchy need to be tied back to the facts at various effective dates**, the Type 2 model of *add new row* is used
* This **approach allows the hierarchy dimension to be joined to a fact table using any effective date to obtain the classification or assignment at a given moment**
* This allows organizations to answer questions such as “*who did employee X report to last year versus currently?”*
* Now that we have familiarized ourselves w/ the types of hierarchies that exist in our organization and learned how to model + analyze them, let’s review what we’ve learned

#### Summary

* From how employees are organized to how customers or products are segmented, **hierarchies exist in every organization**
* By learning how to **model hierarchies as business dimensions**, organizations can **analyze + maintain them as they would any other entity**
* However, ***not all hierarchies behave the same way*, and learning to understand their differences is the key to modeling them accordingly**
* **The biggest difference between hierarchies is the degree of variance between their members**
* **Fixed + slightly-ragged hierarchies have a set number of levels corresponding to their attributes’ natural grouping**
* **Ragged hierarchies, such as organization levels, can vary in depth *and* the number of branches for each node**
* **B/c of their tree-like structure, ragged hierarchies present a challenge for RDBs that rely on a columnar structure**
* **Using native extensions to standard SQL**, Snowflake allows users to easily perform complex operations, such as **recursively traversing a tree structure + determining its branches + depth levels**
* The **result can be stored in a separate dimension known as a bridge table**, which **allows the hierarchy to change independently of the associated entity**
* Once hierarchy details are separated into dimension tables, they can be “historified” according to business needs using SCD techniques
* Now that we are familiar w/ the structures commonly found in a DW + the tools Snowflake provides to maintain them, we can discuss more advanced modeling patterns