Chapter 4: Conceptual Data Warehouse Design

Advantages of using conceptual models in database design:

1. **Conceptual models facilitate communication between users and designers**
   – since they do not require knowledge about specific features of the underlying implementation platform.
2. **Schemas developed using conceptual models can be mapped to various logical models**
   – such as relational, object-relational, or object-oriented models, thus simplifying responses to changes in the technology used.
3. **Conceptual models facilitate the database maintenance and evolution**
   – since they focus on users' requirements; as a consequence, they provide better support for subsequent changes in the logical and physical schemas.

4.1 Conceptual Modeling of Data Warehouses

1. As studied in Chap. 2, the conventional database design process includes the creation of database schemas at three different levels: conceptual, logical, and physical.
2. A **conceptual schema** is a concise description of the users' data requirements without taking into account implementation details.
3. Conventional databases are generally designed at the conceptual level using some variation of the well-known entity-relationship (ER) model, although the Unified Modeling Language (UML) is being increasingly used.
4. Conceptual schemas can be easily translated to the relational model by applying a set of mapping rules.
5. Within the database community, it has been acknowledged for several decades that conceptual models allow better communication between designers and users for the purpose of understanding application requirements.
6. A conceptual schema is more stable than an implementation-oriented (logical) schema, which must be changed whenever the target platform changes.
7. Conceptual models also provide better support for visual user interfaces; for example, ER models have been very successful with users due to their intuitiveness.
8. However, there is no well-established and universally adopted conceptual model for multidimensional data.
9. Due to this lack of a generic, user-friendly, and comprehensible conceptual data model, data warehouse design is usually directly performed at the logical level, based on star and/or snowflake schemas (which we will study in Chap. 5), leading to schemas that are difficult to understand by a typical user.
10. Providing extensions to the ER and the UML models for data warehouses is not really a solution to the problem, since ultimately they represent a reflection and visualization of the underlying relational technology concepts and, in addition, reveal their own problems.
11. Therefore, conceptual data warehousing modeling requires a model that clearly stands on top of the logical level.

12. In this chapter, we use the **MultiDim** model, which is powerful enough to represent at the conceptual level all elements required in data warehouse and OLAP applications, that is, dimensions, hierarchies, and facts with associated measures.
13. The graphical notation of the **MultiDim** model is shown in Fig. 4.1.

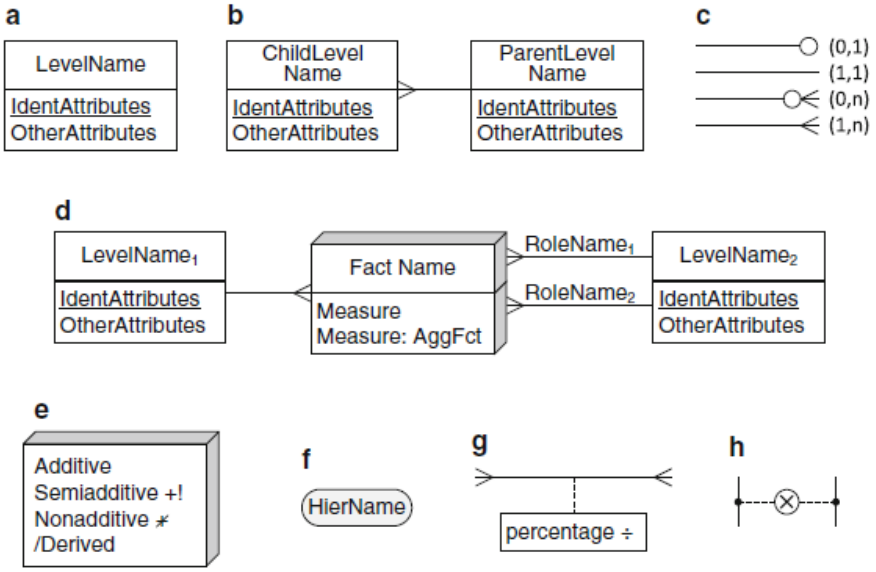


Fig. 4.1 Notation of the MultiDim model. (a) Level. (b) Hierarchy. (c) Cardinalities. (d) Fact with measures and associated levels. (e) Types of measures. (f) Hierarchy name. (g) Distributing attribute. (h) Exclusive relationships

14. As we can see, the notation resembles the one of the **ER model**, which we presented in Chap. 2.
15. A more detailed description of our notation is given in Appendix A.
16. In order to give a general overview of the model, we shall use the example in Fig 4.2 1, which illustrates the conceptual schema of the Northwind data warehouse.

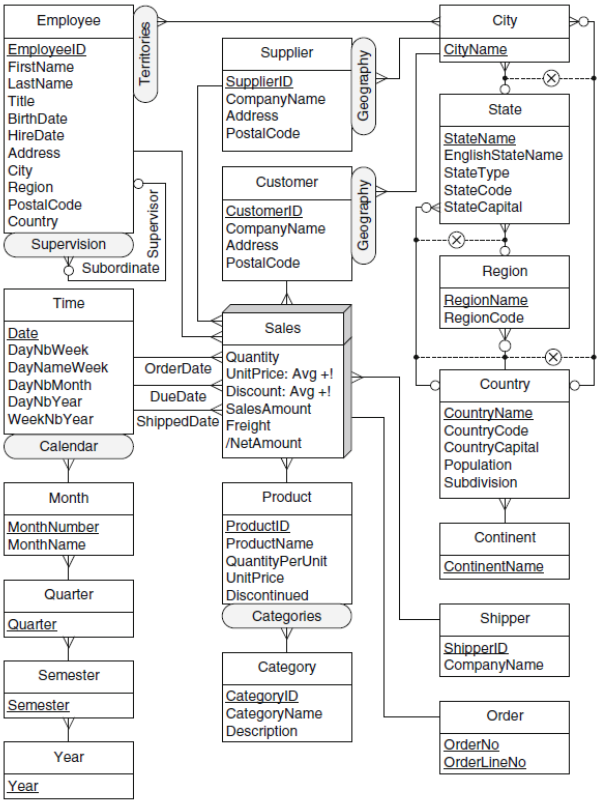
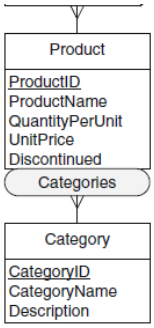


Fig. 4.2 Conceptual schema of the Northwind data warehouse

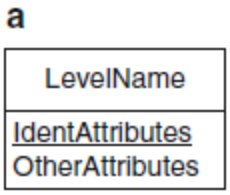
17. This figure includes several types of hierarchies, which will be presented in more detail in the subsequent sections.
18. We next introduce the main components of the model.

Components of the MultiDim Model

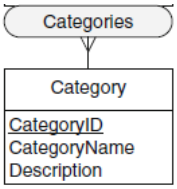
1. A **schema** is composed of a set of dimensions and a set of facts.
2. A **dimension** is composed of either one level or one or more hierarchies.
3. A **hierarchy** is in turn composed of a set of levels (we explain below the notation for hierarchies).
4. There is no graphical element to represent a dimension; it is depicted by means of its constituent elements.
5. A **level** is analogous to an entity type in the ER model.
6. It describes a set of real-world concepts that, from the application perspective, have similar characteristics.
7. For example, **Product** and **Category** are some of the levels in Fig. 4.2.



8. Instances of a level are called **members**.
9. As shown in Fig. 4.1a, a level has a set of attributes that describe the characteristics of their members.



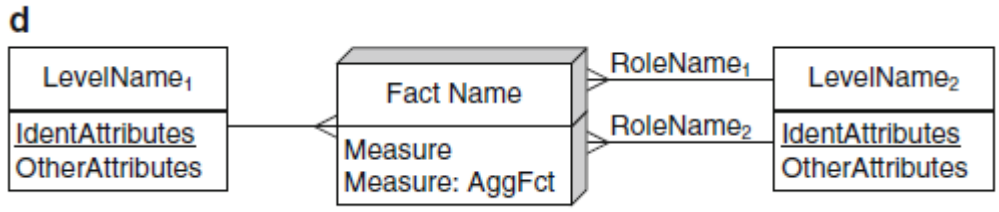
10. In addition, a level has one or several identifiers that uniquely identify the members of a level, each identifier being composed of one or several attributes.
11. For example, in Fig. 4.2, ***CategoryID*** is an identifier of the ***Category*** level.



12. Each attribute of a level has a **type, that is, a domain for its values**.
13. Typical value domains are **integer, real, and string**.
14. We do not include type information for attributes in the graphical representation of our conceptual schemas.

Facts and Measures

1. A **fact** (Fig. 4.1d) relates several levels.



2. For example, the Sales fact in Fig. 4.2 relates the Employee, Customer, Supplier, Shipper, Order, Product, and Time levels.

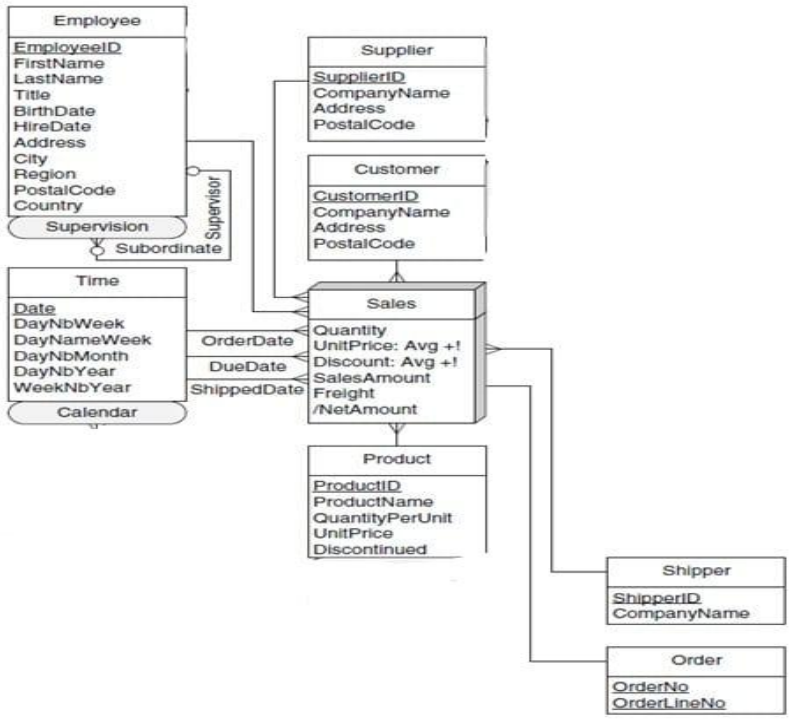
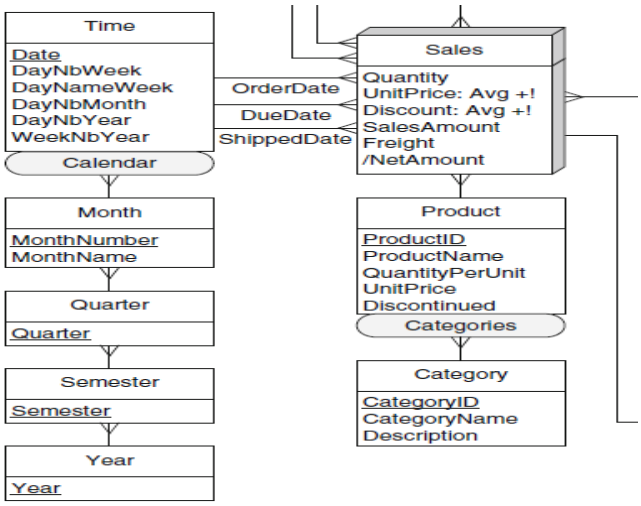
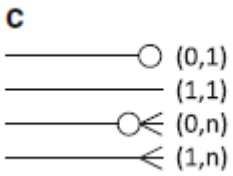


Fig. 4.2 Conceptual schema of the Northwind data warehouse

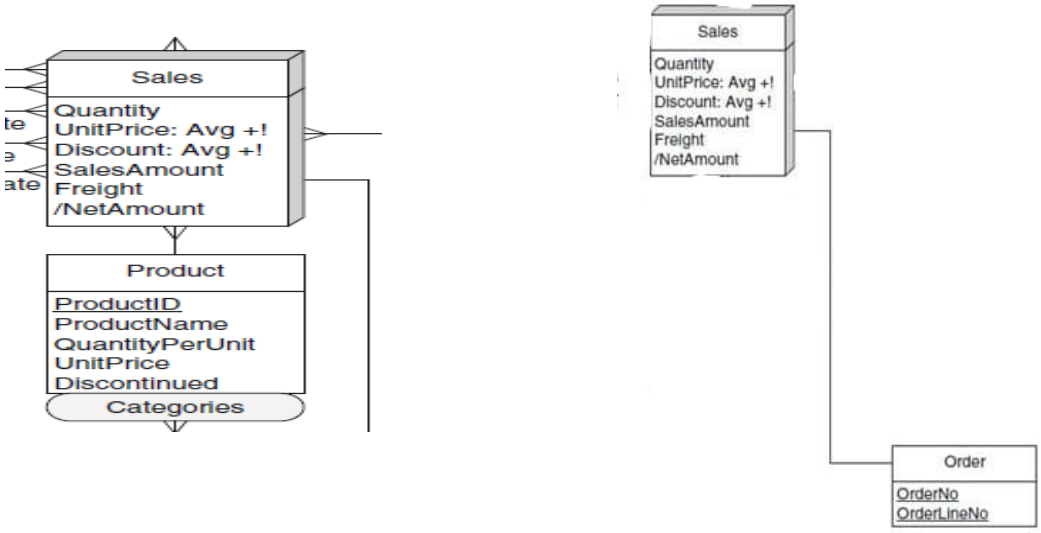
3. As shown in Fig. 4.1d, the same level can participate several times in a fact, playing different roles.
4. Each **role** is identified by a name and is represented by a separate link between the corresponding level and the fact.
5. For example, in Fig. 4.2, the Time level participates in the Sales fact with the roles ***OrderDate, DueDate,*** and **ShippedDate**



6. Instances of a fact are called **fact members**.
7. The **cardinality of the relationship** between facts and levels, as shown in **Fig. 4.1c,** indicates the minimum and the maximum number of fact members that can be related to level members.



8. For example, in **Fig. 4.2**, the Sales fact is related to the Product level with a one-to-many cardinality, which means that one sale is related to only one product and that each product can have many sales.

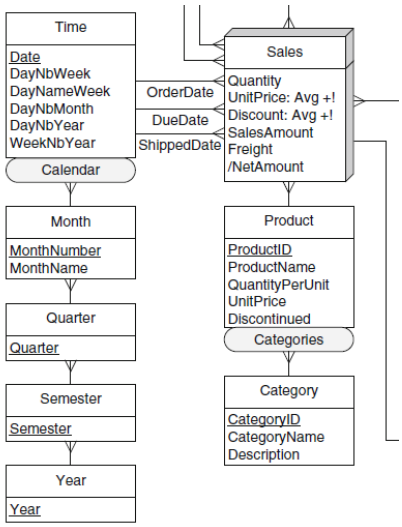


9. On the other hand, the Sales fact is related to the Order level with a one-to-one cardinality, which means that every sale is related to only one order line and that each order line has only one sale.
10. A fact may contain attributes commonly called **measures**.
11. These contain data (**usually numerical**) that are analyzed using the various perspectives.

Role-playing Dimensions and Hierarchies

1. The level in a hierarchy representing the most general data is called the **root level**.
2. It is usual (but not mandatory) to represent the **root of a hierarchy** using a distinguished level called ***All***, which contains a single member, denoted all.
3. The decision of including this level in multidimensional schemas is left to the designer.
4. In the remainder, we do not show the ***All*** level in the hierarchies (except when we consider it necessary for clarity of presentation), since we consider that it is meaningless in conceptual schemas.
5. The **identifier attributes** of a **parent level** define how child members are grouped.

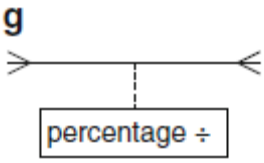
6. For example, in Fig. 4.2, ***CategoryID*** in the ***Category*** level is an **identifier attribute**; it is used for grouping different product members during the roll-up operation from the ***Product*** to the ***Category*** levels.



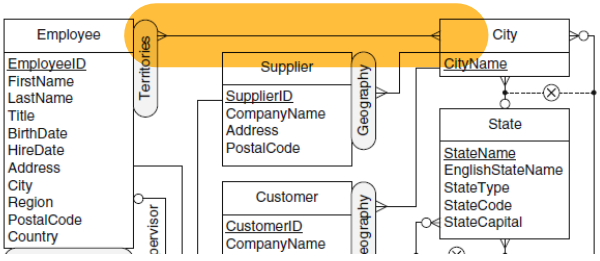
Distributing Attributes and Exclusive Relationships

- **Distributing Attributes:**

1. However, in the case of **many-to-many parent-child** relationships, it is also needed to determine how to distribute the measures from a child to its parent members.
2. For this, a **distributing attribute** (Fig. 4.1g) may be used, if needed.

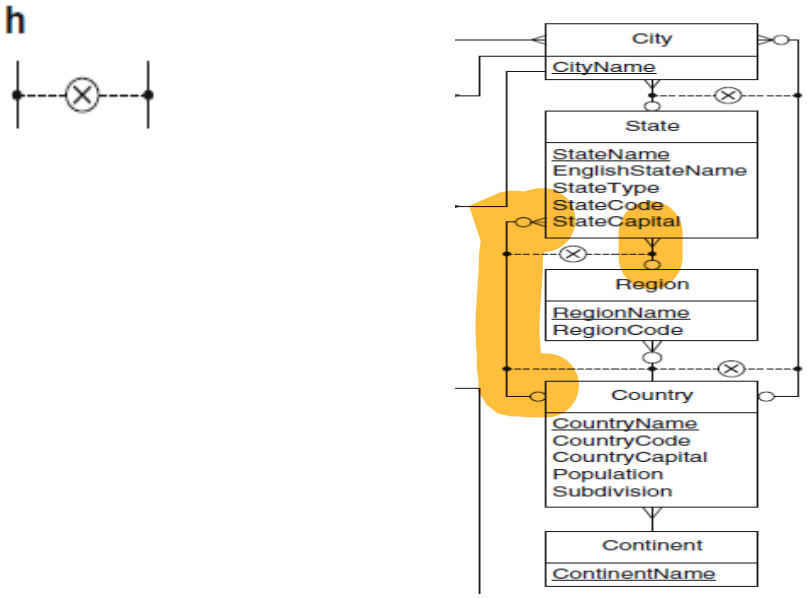


3. For example, in Fig. 4.2, the relationship between ***Employee*** and ***City*** is **many-to-many,** that is, **the same employee can be assigned to several cities**.



4. A **distributing attribute** can be used to store the percentage of time that an employee devotes to each city.

- **Exclusive Relationships:**
  5. Finally, it is sometimes the case that **two or more parent-child relationships** are **exclusive.**
  6. This is represented using the symbol '⊗', as shown in **Fig. 4.1h.**



  7. An example is given in **Fig. 4.2**, where states can be aggregated either into regions or into countries.
  8. Thus, according to their type, states participate in only one of the relationships departing from the State level.

✅ **What Does This Mean?**

Sometimes, in **dimensional hierarchies**, a **level** (like State) may roll up (aggregate) to **more than one parent level** (like Region or Country).
But here's the catch:

⚠️ **Each individual member** of the level (i.e., each **state**) can **only participate in one of the roll-ups**, **not both at the same time**.
This situation is called an **exclusive parent-child relationship**.

✅ **Symbol ⊗ Meaning:**

The **symbol ⊗** in your diagram is used to indicate **exclusive roll-up paths** — meaning:

> *A State either rolls up to a Region **OR** rolls up to a Country, **not both**.*

Conclusion

1. The reader may have noticed that many of the concepts of the **MultiDim** model are similar to those used in Chap. 3, when we presented the multidimensional model and the data cube.
2. This suggests that the **MultiDim model stays on top of the logical level**, hiding from the user the implementation details.
3. In other words, the model represents a conceptual data cube.
4. Therefore, we will call the model in **Fig 4.2 1**as the Northwind data cube.

4.2 Hierarchies

1. **Hierarchies are key elements in analytical applications**, since they provide the means to represent the data under analysis at different abstraction levels.

   📌 ***Example:*** A ***Product*** can be seen as part of a ***Category***, which is part of a ***Department***. These levels allow users to drill up/down depending on what abstraction they want.
2. In real-world situations, users must deal with complex hierarchies of various kinds.
3. Even though we can model complex hierarchies at a conceptual level, as we will study in this section, **logical models** of data warehouse and OLAP systems only provide a limited set of kinds of hierarchies.
4. Therefore, users are often unable to capture the essential semantics of multidimensional applications and must limit their analysis to considering only the predefined kinds of hierarchies provided by the tools in use.
5. Nevertheless, a data warehouse designer should be aware of the problems that the various kinds of hierarchies introduce and be able to deal with them.
6. In this section, we discuss **several kinds of hierarchies** that can be represented by means of the **MultiDim** model, although the classification of hierarchies that we will provide is **independent of the conceptual model** used to represent them.

   📌 ***Note:*** The classification is **universal** — valid whether you're using **ER, UML, or dimensional modeling.**
7. Since many of the hierarchies we study next are **not present in the Northwind data cube of Fig. 4.2**, we will introduce new ad hoc examples when needed.

4.2.1 Balanced Hierarchies

1. A **balanced hierarchy has only one path at the schema level**, where **all levels are mandatory.**
2. An example is given by hierarchy ***Product → Category*** in Fig. 4.2.
3. At the instance level, the members form a **tree where all the branches have the same length**, as shown in Fig. 4.3.

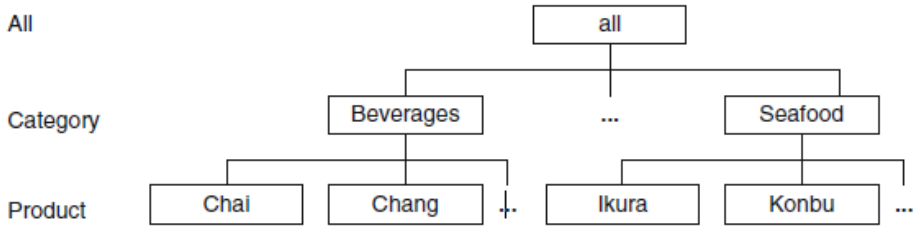


Fig. 4.3 Example of instances of the balanced hierarchy Product → Category in Fig. 4.2 (repeated from Fig. 3.3)

4. All parent members have at least one child member, and a child member belongs exactly to one parent member.
5. For example, in Fig. 4.3, each **category is assigned at least one product**, and **a product belongs to only one category**.

Unbalanced Hierarchies

1. An **unbalanced hierarchy has only one path at the schema level**, where **at least one level is not mandatory.**
2. Therefore, at the instance level, there can be **parent members without associated child members**.
3. **Figure 4.4a** shows a hierarchy schema in which a bank is composed of several branches, where a branch may have agencies; further, an agency may have ATMs.

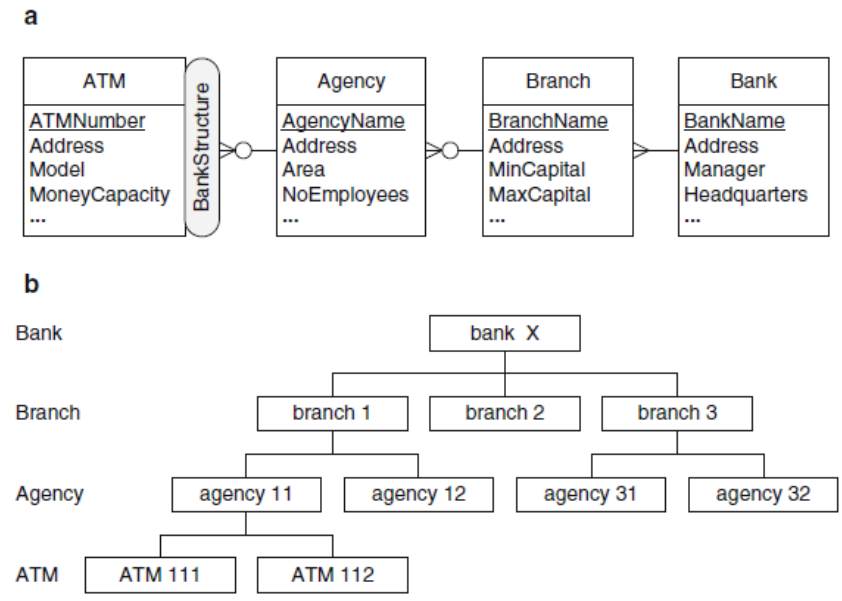


Fig. 4.4 An unbalanced hierarchy. (a) Schema. (b) Examples of instances

4. As a consequence, **at the instance level**, the members represent an **unbalanced tree**, that is, the **branches of the tree have different lengths**, since some parent members do not have associated child members.
5. For example, **Fig. 4.4b** shows a branch with no agency and several agencies with no ATM.
6. As in the case of balanced hierarchies, the cardinalities in the schema imply that **every child member should belong to at most one parent member**.

   📌 ***Explanation:*** Still **strict — one parent per child — but not all parents must have children.**
7. For example, in **Fig. 4.4,** **every agency belongs to one branch**.
8. **Unbalanced hierarchies include a special case** that we call **recursive hierarchies**, also called **parent-child hierarchies**.

   📌 ***Example:*** An employee reports to another employee — same entity, different roles.
9. In this kind of hierarchy, the **same level is linked by the two roles of a parent-child relationship** (note the difference between the notions of parent-child hierarchies and relationships).

   📌 ***Important:*** **The hierarchy is not between two levels, but within the same level**.
10. An example is given by **dimension Employee in Fig. 4.2**, which represents an organizational chart in terms of the employee–supervisor relationship.

11. The **Subordinate and Supervisor roles** of the parent-child relationship are linked to the Employee level.

    📌 ***Explanation:*** Employees can recursively roll-up under other employees

12. As seen in Fig. 4.5, this hierarchy is **unbalanced** since employees with no subordinate will not have descendants in the instance tree.

    📌 ***Example:*** The CEO has many subordinates; an intern has none

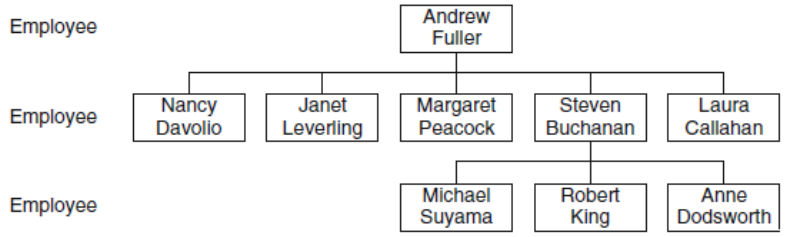


Fig. 4.5 Instances of the parent-child hierarchy in the Northwind data warehouse

---

4.2.2 Generalized Hierarchies

1. Sometimes, the **members of a level are of different types**.

   📌 ***Example:*** A typical example arises with **customers, which can be either companies or persons**.

2. Such a situation is usually captured in an ER model using the **generalization relationship** studied in Chap. 2.
3. Further, suppose that **measures pertaining to customers must be aggregated differently according to the customer type**, where **for companies** the aggregation path is ***Customer*** → ***Sector*** → ***Branch***, while **for persons** it is ***Customer*** → ***Profession*** → ***Branch***.
4. To represent such kinds of hierarchies, the **MultiDim model has the graphical notation shown in Fig. 4.6a**, where the common and specific hierarchy levels and also the parent-child relationships between them are clearly represented.

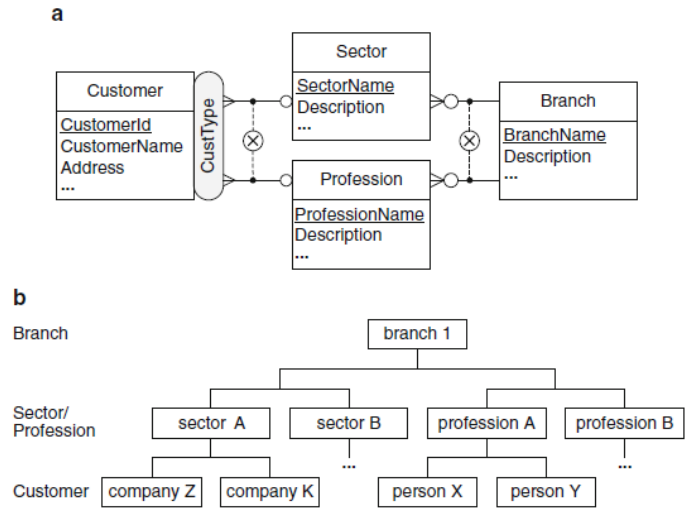


Fig. 4.6 A generalized hierarchy. (a) Schema. (b) Examples of instances

5. Such hierarchies are called **generalized hierarchies**.
6. At the schema level, a generalized hierarchy contains **multiple exclusive paths sharing at least the leaf level**; they may also share some other levels, as shown in Fig. 4.6a.
7. This figure shows the two aggregation paths described above, one for each type of customer, where both belong to the same hierarchy.
8. At the instance level, **each member of the hierarchy belongs to only one path**, as can be seen in Fig. 4.6b.

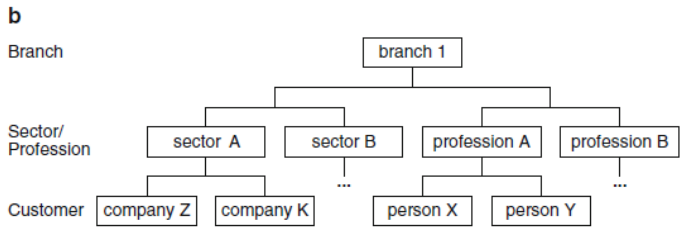


Fig. 4.6 A generalized hierarchy. (a) Schema. (b) Examples of instances

9. We use the symbol '⊗' to indicate that the paths are exclusive for every member.
10. Such a notation is **equivalent to the xor annotation used in UML**.
11. The levels at which the **alternative paths split and join** are called, respectively, the **splitting and joining levels**.

    📌 ***Example:*** Split at ***Customer***, join at ***Branch***.
12. The distinction between splitting and joining levels in generalized hierarchies is important to ensure correct **measure aggregation during roll-up operations**, a property called **summarizability**, which we discussed in Chap. 3.

    📌 **Summarizability** = no double counting or miscounting during aggregation.
13. **Generalized hierarchies are, in general, not summarizable**.

    📌 ***Why?*** Not every instance follows the full path or has equal path depth.
14. For example, not all customers are mapped to the Profession level.

    📌 *So*, aggregating by ***Profession*** misses company customers → invalid total.
15. Thus, the **aggregation mechanism should be modified when a splitting level is reached** in a roll-up operation.
16. In generalized hierarchies, it is not necessary that splitting levels are joined.
17. An example is the hierarchy in Fig. 4.7, which is used for analyzing **international publications**.

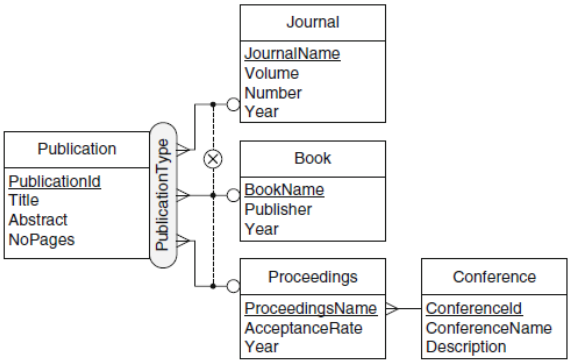


Fig. 4.7 A generalized hierarchy without a joining level

18. Three kinds of publications are considered: **journals, books, and conference proceedings**.
19. The latter can be aggregated to the **conference level**.
    (Proceeding → Conference → Field)
20. However, there is **not a common joining level for all paths**.
21. Generalized hierarchies include a special case commonly referred to as **ragged hierarchies**.

    Ragged = some levels are skipped in some paths
22. An example is the hierarchy **City → State → Region → Country → Continent** given in **Fig. 4.2**.
23. As can be seen in **Fig. 4.8**, some countries, such as **Belgium, are divided into regions**, whereas others, such as **Germany, are not**.

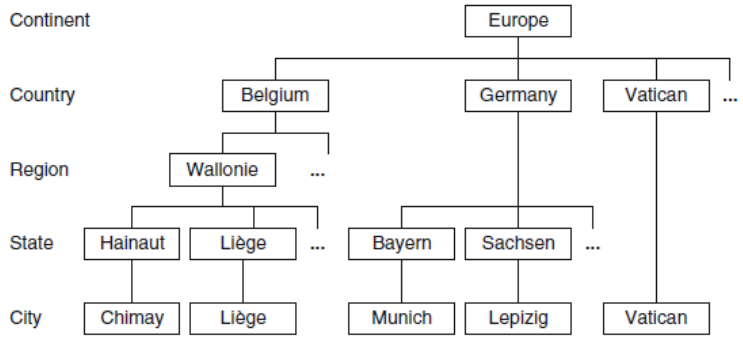


Fig. 4.8 Examples of instances of the ragged hierarchy in Fig. 4.2

24. Furthermore, **small countries like the Vatican have neither regions nor states**.
25. A **ragged hierarchy is a generalized hierarchy where alternative paths are obtained by skipping one or several intermediate levels**.
26. At the instance level, **every child member has only one parent member**, although the path length from the leaves to the same parent level can be different for different members.

4.2.3 Alternative Hierarchies

1. **Alternative hierarchies represent the situation** where at the schema level, there are **several nonexclusive hierarchies that share at least the leaf level**.
2. An example is given in **Fig. 4.9a**, where the **Time dimension includes two hierarchies** corresponding to different groupings of months into **calendar years and fiscal years**.

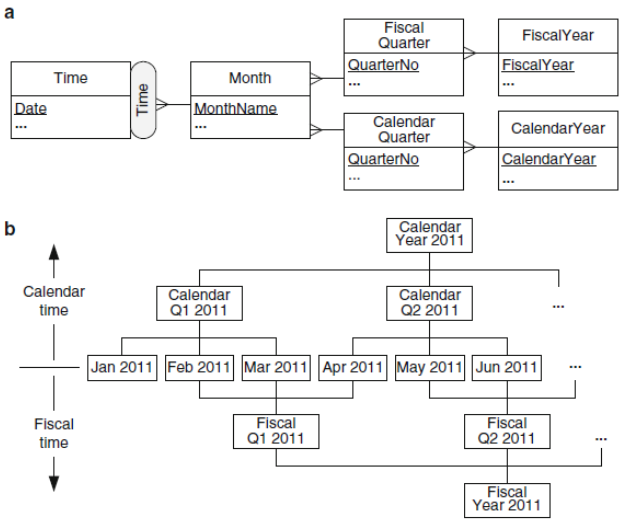


Fig. 4.9 An alternative hierarchy. (a) Schema. (b) Examples of instances

3. **Figure 4.9b** shows an instance of the dimension (we do not show members of the Time level), where it is supposed that **fiscal years begin in February**.
4. As it can be seen, the hierarchies form a **graph**, since a child member is associated with more than one parent member and these parent members belong to different levels.
5. **Alternative hierarchies are needed** when we want to analyze measures from a unique perspective (e.g., time) using **alternative aggregations**.

   📌 ***User picks the aggregation rule:*** fiscal vs. calendar view
6. Note the difference between **generalized and alternative hierarchies** (see **Figs. 4.6 and 4.9**).

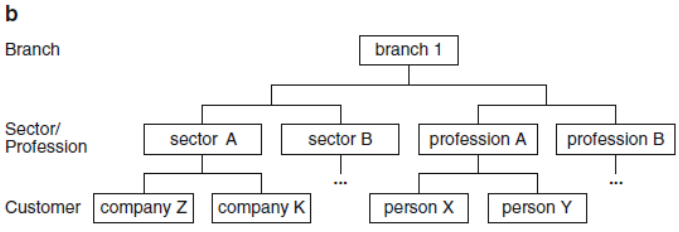


Fig. 4.6 A generalized hierarchy. (a) Schema. (b) Examples of instances

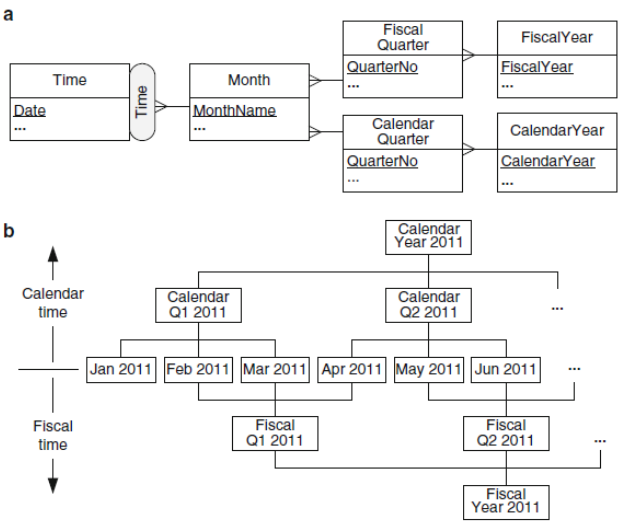


Fig. 4.9 An alternative hierarchy. (a) Schema. (b) Examples of instances

7. Although the two kinds of hierarchies share some levels, they represent **different situations**.
8. In a **generalized hierarchy**, a child member is related to only **one** of the paths.

   📌 ***Exclusive:*** ⊗ — either one path or the other.
9. Whereas in an **alternative hierarchy**, a child member is related to **all paths**, and the user must choose one of them for analysis.

   📌 ***Inclusive:*** All paths valid, choose during query.

4.2.4 Parallel Hierarchies

1. **Parallel hierarchies arise** when a dimension has several hierarchies associated with it, accounting for **different analysis criteria**.

   📌 ***Meaning:*** One dimension (e.g., Product) has multiple **independent ways** to organize data.

   🔸 **Example:** By **product group** vs. **distributor region**.
2. Further, the component hierarchies may be of **different kinds**.

   📌 *Note:* Some could be balanced, others generalized, etc.
3. **Parallel hierarchies can be dependent or independent** depending on whether the component hierarchies share levels.
4. **Figure 4.10 shows an example** of a dimension that has two **parallel independent hierarchies**.

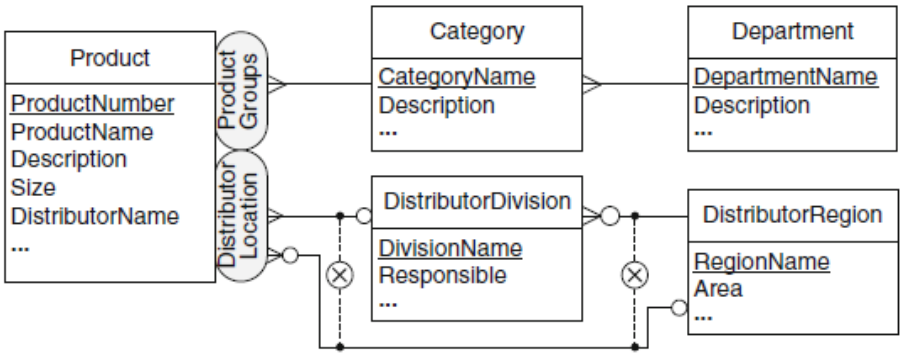


Fig. 4.10 An example of parallel independent hierarchies

5. The hierarchy **ProductGroups** is used for grouping products according to categories or departments.

   📌 ***Example:*** *Product → Subcategory → Category*
6. While the hierarchy **DistributorLocation** groups them according to distributors' divisions or regions.

   📌 ***Example:*** *Product → DistributorCity → DistributorRegion*

7. On the other hand, the **parallel dependent hierarchies given in Fig. 4.11 represent a company** that requires sales analysis for stores located in several countries.

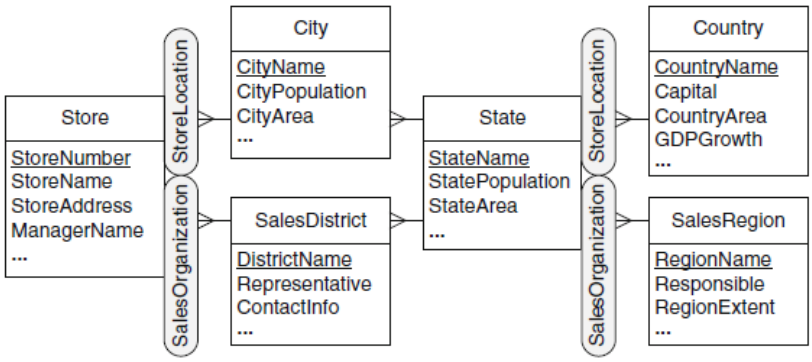


Fig. 4.11 An example of parallel dependent hierarchies

8. The hierarchy **StoreLocation** represents the geographic division of the store address.

   📌 ***Example:*** *Store → City → State → Country*

9. While the hierarchy **SalesOrganization** represents the organizational division of the company.

   📌 ***Example:*** *Store → District → Region → Country*

10. Since the two hierarchies **share the State level**, this level plays different roles according to the hierarchy chosen for the analysis.
11. **Sharing levels in a conceptual schema reduces the number of its elements** without losing its semantics, thus improving readability.

    ✅ Helps avoid duplication of levels unnecessarily.

12. In order to **unambiguously define the levels composing the various hierarchies**, the hierarchy name must be included in the sharing level for hierarchies that continue beyond that level.
13. This is the case of **StoreLocation and SalesOrganization** indicated on level State.
14. Even though both **alternative and parallel hierarchies share some levels** and may be composed of several hierarchies, **they represent different situations** and should be clearly distinguishable at the conceptual level.
15. This is done by including **only one (for alternative hierarchies)** or **several (for parallel hierarchies) hierarchy names**, which account for various analysis criteria.
16. In this way, the user is aware that in the case of **alternative hierarchies**, it is not meaningful to combine levels from different component hierarchies.
17. Whereas **this can be done for parallel hierarchies**.
18. For example, for the schema in Fig. 4.11, the user can safely issue a query
19. **"Sales figures for stores in city A that belong to the sales district B."**
20. Further, in **parallel dependent hierarchies**, a leaf member may be related to various different members in a shared level.
21. Which is **not the case for alternative hierarchies that share levels**.

22. For instance, consider the schema in Fig. 4.12, which refers to the **living place and the territory assignment of sales employees**.

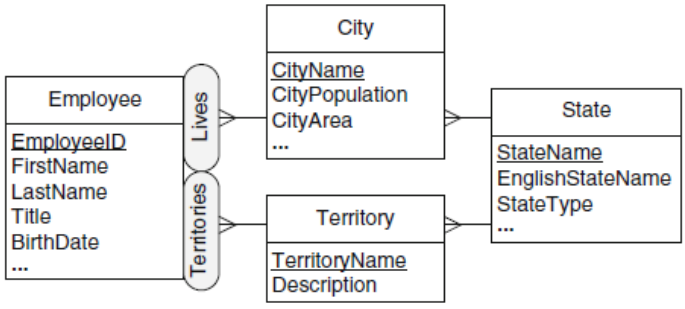


Fig. 4.12 Parallel dependent hierarchies leading to different parent members of the shared level

23. It should be obvious that **traversing the hierarchies Lives and Territory from the Employee to the State level will lead to different states** for employees who live in one state and are assigned to another.
24. As a consequence of this, **aggregated measure values can be reused for shared levels in alternative hierarchies**, whereas this is **not the case for parallel dependent hierarchies**.
25. For example, suppose that the amount of sales generated by employees **E1, E2, and E3 are $50, $100, and $150**, respectively.
26. If all employees live in **state A**, but only E1 and E2 work in this state, aggregating the sales of all employees to the **State level following the Lives hierarchy gives a total amount of $300**, whereas the corresponding value will be **equal to $150 when the Territories hierarchy is traversed**.
27. Note that both results are correct, since the **two hierarchies represent different analysis criteria**.

4.2.6 Nonstrict Hierarchies

➡️ **In the hierarchies studied so far**, we have assumed that each parent-child relationship has a **one-to-many cardinality**, that is, a child member is related to at most one parent member and a parent member may be related to several child members.

💡 **However**, many-to-many relationships between parent and child levels are very common in real-life applications.

📌 ***Examples:*** *A diagnosis may belong to several diagnosis groups, 1 week may span 2 months, a product may be classified into various categories, etc.*

📊 **A hierarchy** that has at least one many-to-many relationship is called **nonstrict**; otherwise, it is called **strict**.

📌 **Nonstrict hierarchy** = at least one **M:N parent-child relationship**.

🔶 The fact that a hierarchy is strict or not is **orthogonal to its kind**.

Meaning: Balanced, unbalanced, generalized — all can be strict or nonstrict.
Thus, the hierarchies previously presented can be either strict or nonstrict.

📚 Understanding Through Examples

👩‍💼 **Figure 4.2** shows a nonstrict hierarchy where an **employee may be assigned to several cities**. (***Example***: Janet Leverling → Atlanta, Orlando, Tampa)

- Some instances of this hierarchy are shown in **Fig. 4.13**.

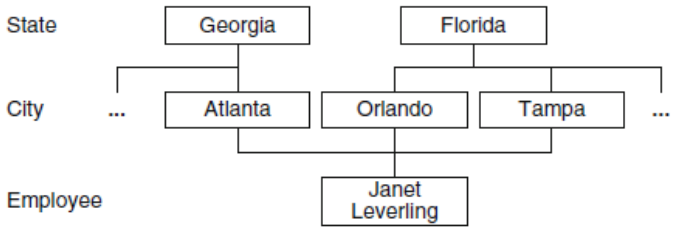


Fig. 4.13 Examples of instances of the nonstrict hierarchy in Fig. 4.2

Here, the employee **Janet Leverling is assigned to three cities** that belong to two states.

- Therefore, since at the instance level a child member may have more than one parent member, the **members of the hierarchy form an acyclic graph**.

Meaning: Instead of a **tree**, we get a **DAG (Directed Acyclic Graph).**

- 🧠 **Note the slight abuse of terminology**:

We use the term **"nonstrict hierarchy"** to denote an **acyclic classification graph**.

- **We use this term for several reasons:**
  - Firstly, the term **"hierarchy"** conveys the notion that users need to analyze measures at different levels of detail;
  - The term **"acyclic classification graph"** is less clear in this sense.
  - Further, the term "hierarchy" is already used by practitioners, and there are tools that support many-to-many parent-child relationships.
  - Finally, this notation is customary in data warehouse research.

🚨 **Problem: Double Counting**

🔺 ❗ **Nonstrict hierarchies induce the problem of double counting of measures** when a roll-up operation reaches a many-to-many relationship.

🔺 🔁 **Example in Fig. 4.14:**
Sales by employees aggregated along ***City*** and ***State*** levels, with **Janet Leverling's (100)** total sales = **100**.

🔺 **Fig. 4.14a (strict)** – employee assigned to **Atlanta**: ✅ Sum of sales is correct.

🔺 **Fig. 4.14b (nonstrict)** – employee assigned to **Atlanta, Orlando, Tampa**: ❌ Sales counted ***3* times**, leading to incorrect totals.

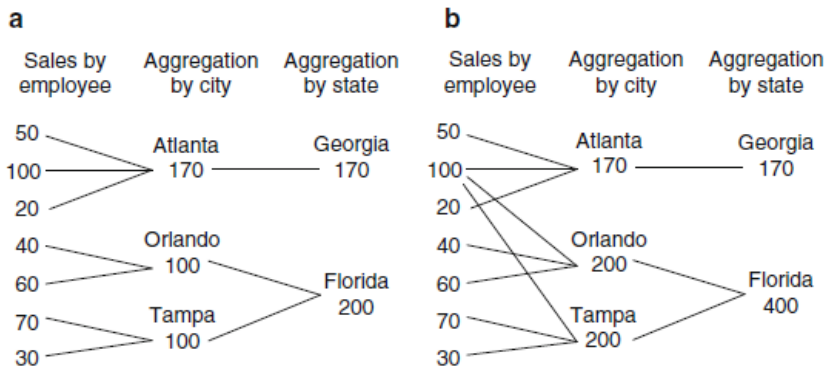


Fig. 4.14 Double-counting problem when aggregating a sales amount measure in Fig. 4.13. (a) Strict hierarchy. (b) Nonstrict hierarchy

---

🛠️ **Solutions to Double Counting**

1. **Transform to Strict Hierarchy**
   - Creating a **new member** for each set of parent members participating in a many-to-many relationship.
     ***E.g.,* make one new artificial *City* member: *Atlanta+Orlando+Tampa*.**
   - Also create a **new member in the state level** since those cities belong to **two states**.
     E.g.,* make one new *State* member: *Georgia+Florida
   - Reassign Janet to the new combined members
     E.g., Janet → Atlanta+Orlando+Tampa → Georgia+Florida
2. **Choose a Primary Parent**
   - Ignore others and choose one (e.g., **Atlanta**) as the **primary member**.
   - **Ignore** the rest (Orlando, Tampa)

   So: Janet → **Atlanta only**
   Atlanta → **Georgia**

   Now: City-level aggregation → $100 in **Atlanta only**
   State-level aggregation → $100 in **Georgia only**
   - ❌ **Orlando and Tampa get no data for Janet**
   - ❗ But both solutions are **flawed**:
     - The former introduces **artificial categories**
     - The latter **ignores valid analysis paths**

⚖️ **Alternative: Use Distributing Attributes**

📊 **Fig. 4.15 shows a nonstrict hierarchy where employees work in several sections.**

- A measure represents an employee's **overall salary**.
- Attribute stores **percentage of time** an employee works in each section.
- Depicted using symbol '÷', representing a **distributing attribute**.

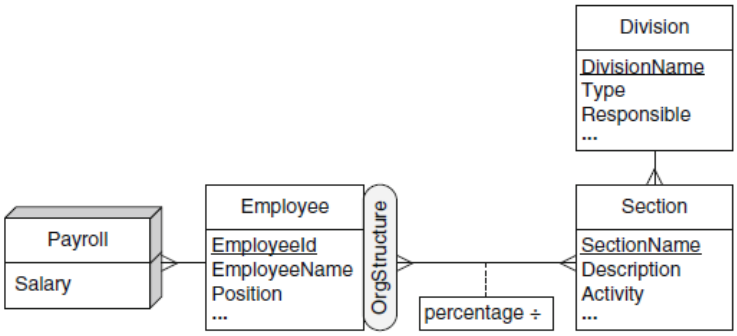


Fig. 4.15 A nonstrict hierarchy with a distributing attribute

📌 **Importance of Accurate Distribution**

- 💼 **Example**: If employee works **less time** in a section but holds a **higher position**, salary may be **higher** for that section.
- So **using time percentage** to divide salary may result in **inaccurate results**.

🔍 **Estimation When Unknown**

- If distributing attribute is **unknown**, we may **approximate**:
  E.g., divide total equally by number of associated parents
  ➡️ Janet Leverling in 3 cities → each city gets **1/3** of sales.

📊 **Example from Fig. 4.15:**

Let's say:

- **Employee:** Janet Leverling
- **Salary:** $90,000
- **Sections Worked:** Sales, HR, Finance
- **Time Distribution:**
  - Sales: 50%
  - HR: 30%
  - Finance: 20%

🎯 Using these percentages, we **distribute the salary** like this:

Section	Time %	Salary Assigned
Sales	50%	$45,000
HR	30%	$27,000
Finance	20%	$18,000

🧩 **Why this is important:**

📌 Sometimes, **time worked ≠ value contributed**.

➡️ For example:
If Janet works **only 20%** in Finance, but she's the **manager** there, her **contribution value** may be **greater** than 20%.
So blindly using **time** as a distribution metric could give **inaccurate** results.

⚠️ What if no distribution attribute is available?

If we don't know time % or workload, then:

➡️ We **approximate** by **equal distribution**:

Janet → Atlanta, Orlando, Tampa (3 cities)
Her total **sales = $90,000**
So each city gets:
$90,000 ÷ 3 = $30,000
But this method assumes **equal contribution**, which may be **unrealistic**.

🧮 Limitations of Approximation

- Even if the **Salary** measure is aggregated correctly (**Fig. 4.16**),
  ⚠️ **double counting still occurs** for **counting employees**.

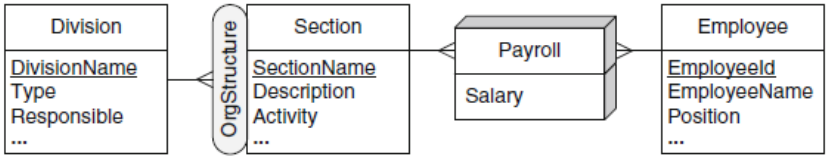


Fig. 4.16 Transforming a nonstrict hierarchy into a strict hierarchy with an additional dimension

👥 Employee Count Example (Fig. 4.17)

- **5 employees** assigned to sections.
- Counting by section gives correct totals ✅
- But counting by division (aggregating sections) gives **7** ❌ instead of **5**, due to **double counting** of E1 and E2.

Fig. 4.17 Double-counting problem for a nonstrict hierarchy

🔁 Alternative Solution: Independent Dimensions

- Fig. 4.16 transforms a non-strict hierarchy into **independent dimensions**.

  📌 **Example:**
  Instead of:
  Employee → Section → Division
  You use:
  - A **Section dimension**
  - A **Division dimension**
  - A separate **Employee dimension**
- Analysis focus shifts: from **employees' salaries** to **employees' salaries by section**.
- ✅ Can be used **only when** exact distribution is **known**.
- ❌ Cannot be used if **no distributing attribute exists** (e.g., Fig. 4.13).

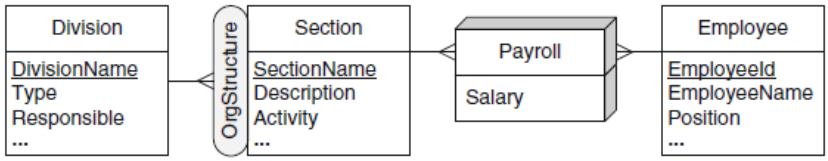


Fig. 4.16 Transforming a nonstrict hierarchy into a strict hierarchy with an additional dimension

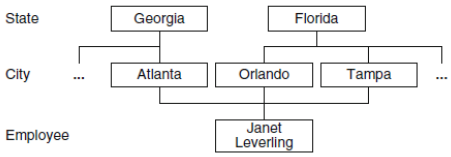


Fig. 4.13 Examples of instances of the nonstrict hierarchy in Fig. 4.2

📈 Benefit:

- You **analyze measures** (e.g., salaries) **by section or division**, without trying to force a parent-child relationship.
- This avoids **hierarchy-induced double counting**.

📌 In Summary, Non-Strict Hierarchies Can Be Handled by:

✨ Transforming to Strict Hierarchy:

- ➕ Creating a new parent member for each group of parent members linked to a single child.
- ➕ Choosing one parent as primary.
- ➕ Converting into independent dimensions.

✨ Using Distributing Attributes

✨ Approximating Distributing Attribute Values

🎯 Note:

Since **each solution has pros and cons** and requires **special aggregation procedures**, the **designer must select** the **appropriate solution** according to the **situation** and **users' requirements**.

Approach	Use When	Pros	Cons
Distributing Attributes	When % time/work per parent is known	Accurate salary aggregation	May still cause double counting of entities

Approach	Use When	Pros	Cons
Equal Distribution	When no distribution metric is known	Simple to apply	May be inaccurate, causes double counting
Independent Dimensions	When measure (e.g., salary) is focus, not entities	Avoids hierarchy issues, accurate analysis	Needs clear distribution structure

4.3 Advanced Modeling Aspects

In this section, we discuss particular modeling issues, namely, facts with multiple granularities and many-to-many dimensions, and show how they can be represented in the MultiDim model.

4.3.1 Facts with Multiple Granularities

📌 Sometimes, it is the case that **measures are captured at multiple granularities**.

💡 ***Example 1***: In **Fig. 4.18**, sales for the **USA might be reported per state**, while **European sales** might be reported **per city**.

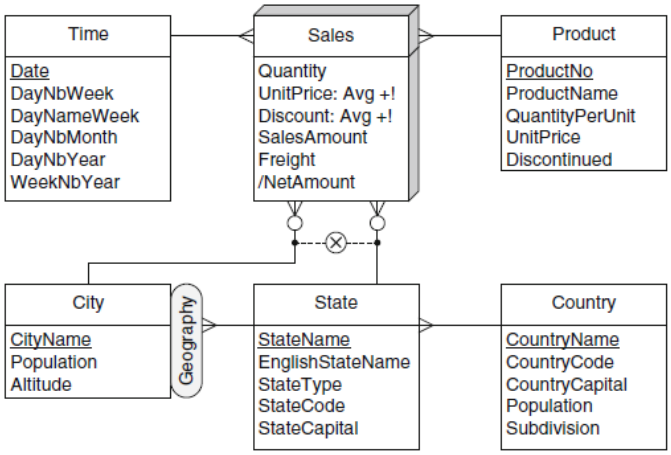


Fig. 4.18 Multiple granularities for the Sales fact

💡 ***Example 2***: A **medical data warehouse** analyzing patients, where there is a **diagnosis dimension** with levels:

- diagnosis
- diagnosis family
- diagnosis group

A patient may be related to a diagnosis at the **lowest granularity**, but may also have **more imprecise diagnoses** at higher levels.

🧩 As can be seen in **Fig. 4.18**, this situation can be modeled using **exclusive relationships between the various granularity levels**.

🔍 Obviously, the issue in this case is to **get correct analysis results** when fact data are registered at multiple granularities.

4.3.2 Many-to-Many Dimensions

📌 In a many-to-many dimension, **several members of the dimension participate in the same fact member**.

💡 ***Example***: In **Fig. 4.19**, since an **account can be jointly owned by several clients**, aggregation of the **balance** according to the clients will count this balance **multiple times**.

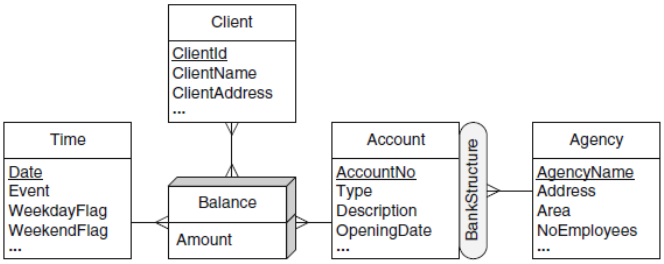


Fig. 4.19 Multidimensional schema for the analysis of bank accounts

📜 ***Fig. 4.20*** *Scenario*:
At time **T1**,

- Account **A1** (balance = 100) → clients: **C1, C2, C3**
- Account **A2** (balance = 500) → clients: **C1, C2**

👉 Total balance = **600**,
but aggregation gives **1300** due to **double counting**.

Time	Account	Client	Balance
T1	A1	C1	100
T1	A1	C2	100
T1	A1	C3	100
T1	A2	C1	500
T1	A2	C2	500

Fig. 4.20 An example of double-counting problem in a many-to-many dimension

---

📐 ✏️ **Ensuring Correct Aggregation: Multidimensional Normal Forms (MNFs)**

🧠 The **1st Multidimensional Normal Form (1MNF)** requires each measure to be **uniquely identified by the set of associated leaf levels**.

📌 The **schema in Fig. 4.19** does **not** satisfy the 1MNF because the measure is **not determined by all leaf levels**, and thus, the fact must be **decomposed**.

- **A balance (measure) is associated with:**
  - Time
  - Account
  - Clients (via many-to-many)

But ❌ the measure is ***not uniquely determined*** by this combination, because:

- The same balance belongs to multiple clients (e.g., joint accounts).
- So, Balance is repeated for each client (→ double counted).

Hence, it violates 1MNF.

🧩 Decomposition Solutions

✅ Solution 1 (Fig. 4.21a):

- Time and account **multidetermine the clients**
- Decompose into **two facts**: ***AccountHolders*** and ***Balance***
  - 🔹 **Fact 1: AccountHolders**
    - Connects **Account ↔ Clients**
    - Represents **ownership relationship**, e.g., which client owns which account.
  - 🔹 **Fact 2: Balance**
    - Connects **Time ↔ Account**
    - Stores the **true balance** of each account over time.
- If account holders **don't change over time**, the **Time–AccountHolders link** can be removed.

🧠 Why is this better?

- Now, **Balance** is **uniquely determined** by Time and Account (✅ satisfies 1MNF).
- No more confusion caused by multiple clients owning same account.
- Aggregation is now **correct**, as each balance appears **once**, and client relationships are kept separately.

✅ Solution 2 (Fig. 4.21b):

- Modeled with a **nonstrict hierarchy**

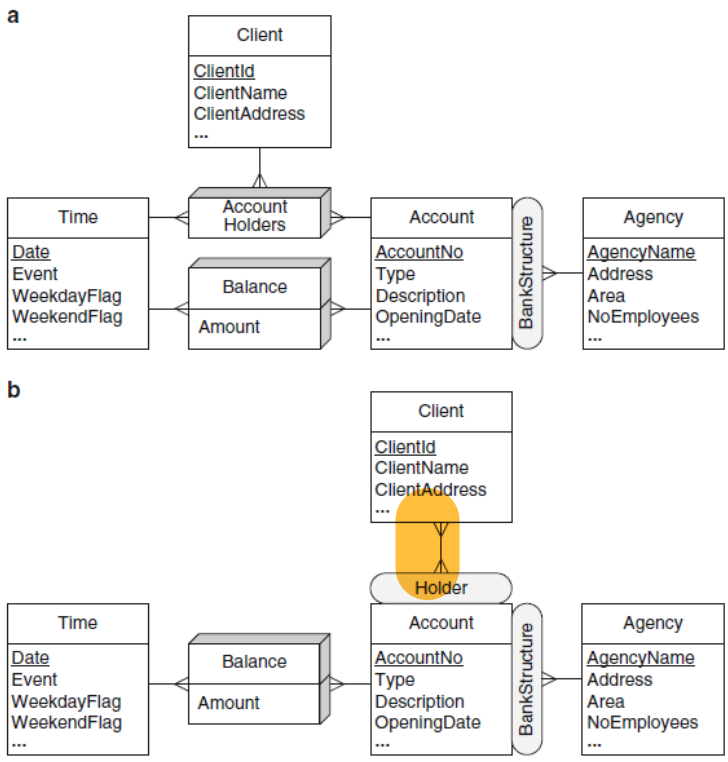


Fig. 4.21 Two possible decompositions of the fact in Fig. 4.19. (a) Creating two facts. (b) Including a nonstrict hierarchy

---

🛠️ Trade-offs in Solutions

📊 Although both solutions eliminate **double counting**, they require:

- Programming effort for **queries about individual clients**

- **Drill-across operations** between facts in **Fig. 4.21a**
- **Special aggregation procedures** for nonstrict hierarchies in **Fig. 4.21b**

⚙ In **Fig. 4.21a**, queries with **drill-across** are complex because they involve converting between **finer and coarser granularities**.

💰 Both solutions can represent **percentage of ownership**:

- As a **measure** in AccountHolders (**Fig. 4.21a**)
- As a **distributing attribute (symbol '÷')** in the relationship (**Fig. 4.21b**)

🧠 Alternative: Client Groups (Fig. 4.22)

📌 Another solution is to create an **additional level** representing **groups of clients** for joint accounts.

💡 *Example*:

- Group 1 → **C1, C2, C3**
- Group 2 → **C1, C2**

❗ But the schema in **Fig. 4.22** is **not in 1MNF**, since Balance is only determined by **Time and Account**, not all leaf levels.

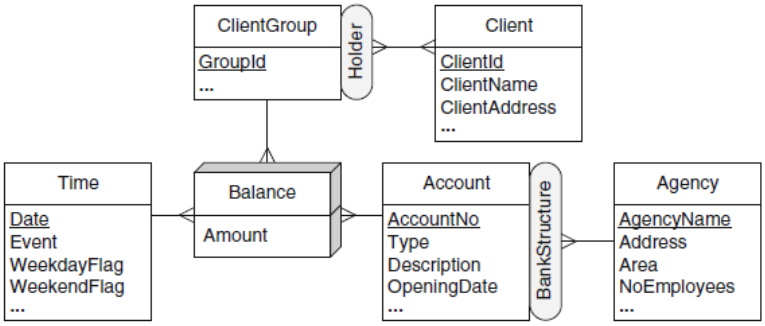


Fig. 4.22 Alternative decomposition of the fact in Fig. 4.19

👉 So, the schema must be decomposed again, leading to designs like **Fig. 4.21**, with ClientGroup and Client levels forming a **nonstrict hierarchy**.

❌ Simple But Risky Option

📌 To avoid many-to-many dimensions, one can:

- Choose **one client** as the **primary account owner**
- Ignore the other clients

✅ This lets us use the schema in **Fig. 4.19** without double counting issues.

❗ But this **does not represent real-world scenarios** and **excludes valid data** from analysis.

📚 Summary

📝 **Many-to-many dimensions** in multidimensional schemas can be avoided by:

- ✅ Transforming the schema into **two separate facts** (**Fig. 4.21a**)
- ✅ Using **nonstrict hierarchies** (**Fig. 4.21b**)

- ✅ Creating a **client group level** (**Fig. 4.22**)
- ❌ Ignoring additional clients by choosing a **primary account owner**

📌 **The choice between these alternatives depends on:**

- Functional and multivalued dependencies
- Types of hierarchies in the schema
- Implementation complexity

4.4 Querying the Northwind Cube Using the OLAP Operation

📊 **Chapter Conclusion: OLAP Operations on the Northwind Cube**

The idea of this section is to show how these operations can be used to express queries over a **conceptual model**, independently of the actual underlying implementation.

🔎 **Query 4.1**

Total sales amount per customer, year, and product category.

```
ROLLUP*(Sales, Customer → Customer, OrderDate → Year, Product →
Category, SUM(SalesAmount))
```

🧠 ROLLUP* specifies the levels at which dimensions are rolled up. Other dimensions are rolled up to All. SUM is used to aggregate SalesAmount.

🌍 **Query 4.2**

Yearly sales amount for each pair of customer country and supplier countries.

```
ROLLUP*(Sales, OrderDate → Year, Customer → Country, Supplier →
Country, SUM(SalesAmount))
```

Same logic as above, with a roll-up by countries of both customers and suppliers.

🗓️ **Query 4.3**

Monthly sales by customer state compared to those of the previous year.

```
Sales1 ← ROLLUP*(Sales, OrderDate → Month, Customer → State,
SUM(SalesAmount))
Sales2 ← RENAME(Sales1, SalesAmount ← PrevYearSalesAmount)
Result ← DRILLACROSS(Sales2, Sales1,
Sales2.OrderDate.Month = Sales1.OrderDate.Month AND
Sales2.OrderDate.Year+1 = Sales1.OrderDate.Year AND
Sales2.Customer.State = Sales1.Customer.State)
```

Uses DRILLACROSS to join current and previous year sales by same state and month.

📈 **Query 4.4**

Monthly sales growth per product.

```
Sales1 ← ROLLUP*(Sales, OrderDate → Month, Product → Product,
SUM(SalesAmount))
Sales2 ← RENAME(Sales1, SalesAmount ← PrevMonthSalesAmount)
```

```
Sales3 ← DRILLACROSS(Sales2, Sales1,
(Sales1.OrderDate.Month > 1 AND Sales2.OrderDate.Month+1 =
Sales1.OrderDate.Month AND
Sales2.OrderDate.Year = Sales1.OrderDate.Year) OR
(Sales1.OrderDate.Month = 1 AND Sales2.OrderDate.Month = 12 AND
Sales2.OrderDate.Year+1 = Sales1.OrderDate.Year))
Result ← ADDMEASURE(Sales3, SalesGrowth = SalesAmount -
PrevMonthSalesAmount)
```

Handles edge cases between January–December for correct SalesGrowth.

🏆 Query 4.5

Three best-selling employees.

```
Sales1 ← ROLLUP*(Sales, Employee → Employee, SUM(SalesAmount))
Result ← MAX(Sales1, SalesAmount, 3)
```

Selects top 3 employees with highest SalesAmount.

👨‍💼 Query 4.6

Best-selling employee per product and year.

```
Sales1 ← ROLLUP*(Sales, Employee → Employee, Product → Product,
OrderDate → Year, SUM(SalesAmount))
Result ← MAX(Sales1, SalesAmount) BY Product, OrderDate
```

Applies MAX grouped by product and year.

🌎 Query 4.7

Countries that account for top 50% of the sales amount.

```
Sales1 ← ROLLUP*(Sales, Customer → Country, SUM(SalesAmount))
Result ← TOPPERCENT(Sales1, Customer, 50) ORDER BY SalesAmount DESC
```

Ranks countries cumulatively until 50% of total sales is reached.

🗓️ Query 4.8

Total sales and average monthly sales by employee and year.

```
Sales1 ← ROLLUP*(Sales, Employee → Employee, OrderDate → Month,
SUM(SalesAmount))
Result ← ROLLUP*(Sales1, Employee → Employee, OrderDate → Year,
SUM(SalesAmount), AVG(SalesAmount))
```

Performs a double roll-up to derive monthly and yearly aggregates.

💰 Query 4.9

Total sales and discount amount per product and month.

```
Sales1 ← ADDMEASURE(Sales, TotalDisc = Discount * Quantity *
UnitPrice)
Result ← ROLLUP*(Sales1, Product → Product, OrderDate → Month,
SUM(SalesAmount), SUM(TotalDisc))
```

Adds a new measure before rolling up.

🗓️ Query 4.10

Monthly year-to-date (YTD) sales for each product category.

```
Sales1 ← ROLLUP*(Sales, Product → Category, OrderDate → Month,
SUM(SalesAmount))
Result ← ADDMEASURE(Sales1, YTD = SUM(SalesAmount) OVER OrderDate
BY Year ALL CELLS PRECEDING)
```

Uses a **windowed SUM** to compute cumulative sales within a year.

📊 Query 4.11

3-month moving average of sales amount by category.

```
Sales1 ← ROLLUP*(Sales, Product → Category, OrderDate → Month,
SUM(SalesAmount))
Result ← ADDMEASURE(Sales1, MovAvg = AVG(SalesAmount) OVER
OrderDate 2 CELLS PRECEDING)
```

Calculates moving average over 3 months (current + 2 previous).

📜 Query 4.12

Personal sales vs. total sales (including subordinates) in 1997.

```
Sales1 ← SLICE(Sales, OrderDate.Year = 1997)
Sales2 ← ROLLUP*(Sales1, Employee → Employee, SUM(SalesAmount))
Sales3 ← RENAME(Sales2, PersonalSales ← SalesAmount)
Sales4 ← RECROLLUP(Sales2, Employee → Employee, Supervision,
SUM(SalesAmount))
Result ← DRILLACROSS(Sales4, Sales3)
```

Uses RECROLLUP over a **supervision hierarchy** to compare sales.

📦 Query 4.13

Sales, number of products, and quantity per order.

```
ROLLUP*(Sales, Order → Order, SUM(SalesAmount), COUNT(Product) AS
ProductCount, SUM(Quantity))
```

Gathers total sales and product details per order.

📉 Query 4.14

Monthly stats: total orders, total sales, average sales per order.

```
Sales1 ← ROLLUP*(Sales, OrderDate → Month, Order → Order,
SUM(SalesAmount))
Result ← ROLLUP*(Sales1, OrderDate → Month, SUM(SalesAmount),
AVG(SalesAmount) AS AvgSales, COUNT(Order) AS OrderCount)
```

Two-level roll-up: from orders to months, with aggregations.

🌐 Query 4.15

Total sales, number of cities, and states per employee.

```
ROLLUP*(Sales, Employee → State, SUM(SalesAmount), COUNT(DISTINCT
City) AS NoCities, COUNT(DISTINCT State) AS NoStates)
```

Handles a **nonstrict hierarchy** (Territories) and avoids double counting.