Chapter 4
Conceptual Data Warehouse Design

The advantages of using conceptual models for designing databases are well known. Conceptual models facilitate communication between users and designers since they do not require knowledge about specific features of the underlying implementation platform. Further, 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. Moreover, 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.

In this chapter, we focus our study on conceptual modeling for data warehouses. In particular, we base our presentation in the MultiDim model, which can be used to represent the data requirements of data warehouse and OLAP applications. The definition of the model is given in Sect. 4.1. Since hierarchies are essential for exploiting data warehouse and OLAP systems to their full capabilities, in Sect. 4.2, we consider various kinds of hierarchies that exist in real-world situations. We classify these hierarchies, giving a graphical representation of them and emphasizing the differences between them. We also present advanced aspects of conceptual modeling in Sect. 4.3. Finally, in Sect. 4.4, we revisit the OLAP operations that we presented in Chap. 2 by addressing a set of queries to the Northwind data warehouse.

4.1 Conceptual Modeling of Data Warehouses

As studied in Chap. 2, the conventional database design process includes the creation of database schemas at three different levels: conceptual, logical, and physical. A **conceptual schema** is a concise description of the users' data requirements without taking into account implementation details.

A. Vaisman and E. Zimányi, *Data Warehouse Systems*, Data-Centric Systems and Applications, DOI 10.1007/978-3-642-54655-6_4,

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. Conceptual schemas can be easily translated to the relational model by applying a set of mapping rules.

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. A conceptual schema is more stable than an implementation-oriented (logical) schema, which must be changed whenever the target platform changes. Conceptual models also provide better support for visual user interfaces; for example, ER models have been very successful with users due to their intuitiveness.

However, there is no well-established and universally adopted conceptual model for multidimensional data. 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. 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. Therefore, conceptual data warehousing modeling requires a model that clearly stands on top of the logical level.

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. The graphical notation of the MultiDim model is shown in Fig. 4.1. As we can see, the notation resembles the one of the ER model, which we presented in Chap. 2. A more detailed description of our notation is given in Appendix A.

In order to give a general overview of the model, we shall use the example in Fig. 4.2, which illustrates the conceptual schema of the Northwind data warehouse. This figure includes several types of hierarchies, which will be presented in more detail in the subsequent sections. We next introduce the main components of the model.

A **schema** is composed of a set of dimensions and a set of facts.

A **dimension** is composed of either one level or one or more hierarchies. A hierarchy is in turn composed of a set of levels (we explain below the notation for hierarchies). There is no graphical element to represent a dimension; it is depicted by means of its constituent elements.

A **level** is analogous to an entity type in the ER model. It describes a set of real-world concepts that, from the application perspective, have similar characteristics. For example, Product and Category are some of the levels in Fig. 4.2. Instances of a level are called **members**. As shown in Fig. 4.1a,

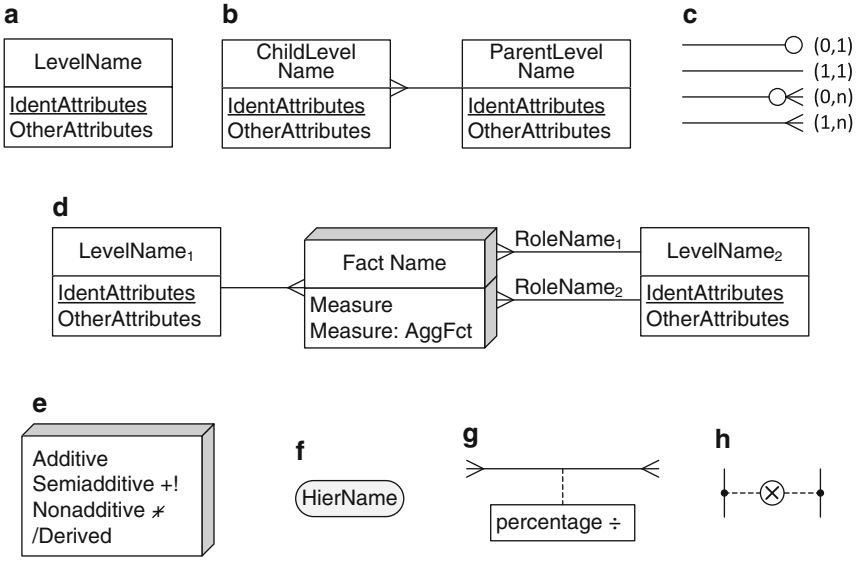

Fig. 4.1 Notation of the MultiDim model. (**a**) Level. (**b**) Hierarchy. (**c**) Cardinalities. (**d**) Fact with measures and associated levels. (**e**) Types of measures. (**f**) Hierarchy name. (**g**) Distributing attribute. (**h**) Exclusive relationships

a level has a set of **attributes** that describe the characteristics of their members. 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. For example, in Fig. 4.2, CategoryID is an identifier of the Category level. Each attribute of a level has a type, that is, a domain for its values. Typical value domains are integer, real, and string. We do not include type information for attributes in the graphical representation of our conceptual schemas.

A **fact** (Fig. 4.1d) relates several levels. For example, the Sales fact in Fig. 4.2 relates the Employee, Customer, Supplier, Shipper, Order, Product, and Time levels. As shown in Fig. 4.1d, the same level can participate several times in a fact, playing different **roles**. Each role is identified by a name and is represented by a separate link between the corresponding level and the fact. For example, in Fig. 4.2, the Time level participates in the Sales fact with the roles OrderDate, DueDate, and ShippedDate. Instances of a fact are called **fact members**. 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. 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. On the other hand, the Sales fact is related to the Order level with a one-to-one cardinality, which means that

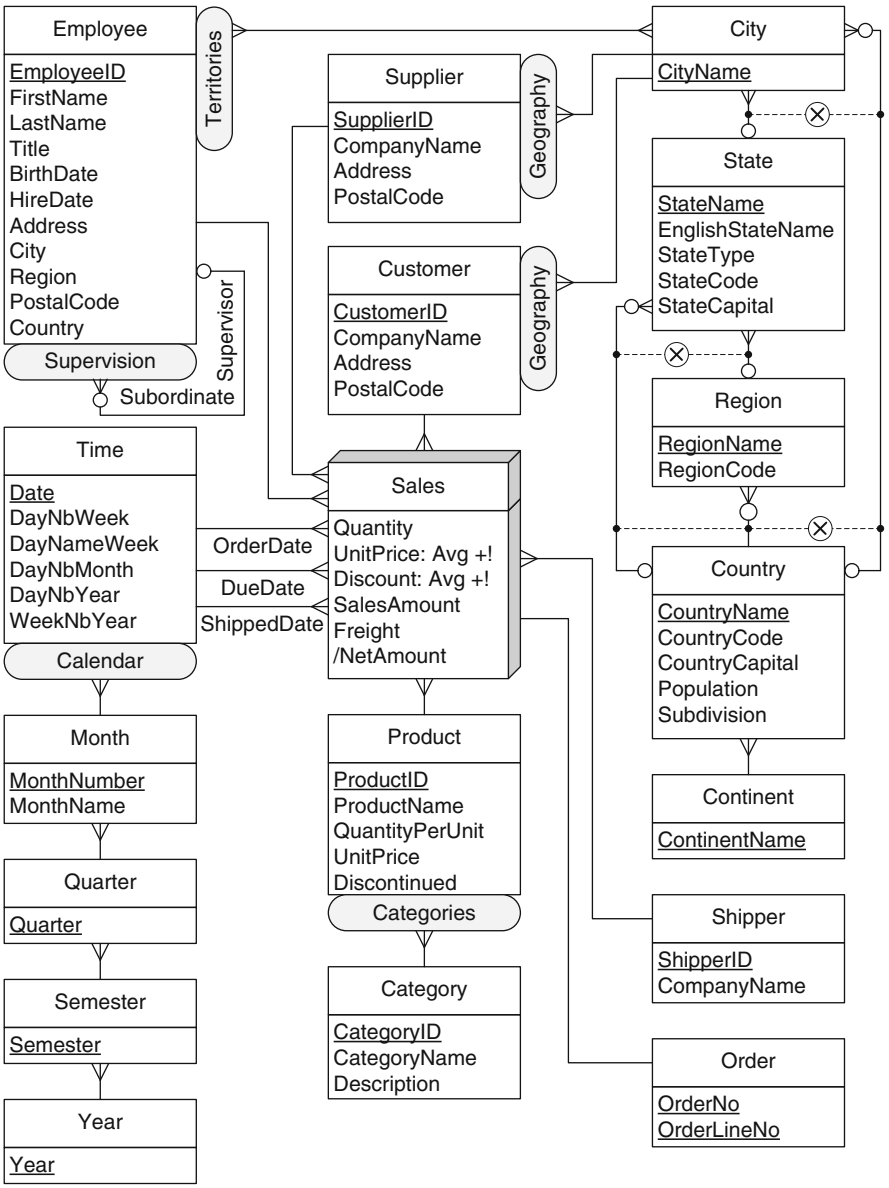

Fig. 4.2 Conceptual schema of the Northwind data warehouse

every sale is related to only one order line and that each order line has only one sale.

A fact may contain attributes commonly called **measures**. These contain data (usually numerical) that are analyzed using the various perspectives

represented by the dimensions. For example, the Sales fact in Fig. 4.2 includes the measures Quantity, UnitPrice, Discount, SalesAmount, Freight, and NetAmount. The identifier attributes of the levels involved in a fact indicate the granularity of the measures, that is, the level of detail at which measures are represented.

Measures are aggregated along dimensions when performing roll-up operations. As shown in Fig. 4.1d, the aggregation function associated with a measure can be specified next to the measure name, where the SUM aggregation function is assumed by default. In Chap. 3, we classified measures as **additive**, **semiadditive**, or **nonadditive**. As shown in Fig. 4.1e, we assume by default that measures are additive, that is, they can be summarized along all dimensions. For semiadditive and nonadditive measures, we include the symbols '+!' and '$\not+$', respectively. For example, in Fig. 4.2 the measures Quantity and UnitPrice are, respectively, additive and semiadditive measures. Further, measures and level attributes may be **derived**, where they are calculated on the basis of other measures or attributes in the schema. We use the symbol '/' for indicating derived measures and attributes. For example, in Fig. 4.2, the measure NetAmount is derived.

A **hierarchy** comprises several related levels, as in Fig. 4.1b. Given two related levels of a hierarchy, the lower level is called the **child** and the higher level is called the **parent**. Thus, the relationships composing hierarchies are called **parent-child relationships**. The **cardinalities** in parent-child relationships, as shown in Fig. 4.1c, indicate the minimum and the maximum number of members in one level that can be related to a member in another level. For example, in Fig. 4.2, the child level Product is related to the parent level Category with a one-to-many cardinality, which means that every product belongs to only one category and that each category can have many products.

A dimension may contain several hierarchies, each one expressing a particular criterion used for analysis purposes; thus, we include the **hierarchy name** (Fig. 4.1f) to differentiate them. If a single level contains attributes forming a hierarchy, such as the attributes City, Region, and Country in the Employee dimension in Fig. 4.2, this means that the user is not interested in employing this hierarchy for aggregation purposes.

Levels in a hierarchy are used to analyze data at various *granularities* or levels of detail. For example, the Product level contains specific information about products, while the Category level may be used to see these products from the more general perspective of the categories to which they belong. The level in a hierarchy that contains the most detailed data is called the **leaf level**. The name of the leaf level defines the dimension name, except for the case where the same level participates several times in a fact, in which case the role name defines the dimension name. These are called **role-playing dimensions**. The level in a hierarchy representing the most general data is called the **root level**. 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. The decision of including this level in multidimensional schemas is left to the designer. 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.

The identifier attributes of a parent level define how child members are grouped. 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. 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. For this, a **distributing attribute** (Fig. 4.1g) may be used, if needed. 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. A distributing attribute can be used to store the percentage of time that an employee devotes to each city.

Finally, it is sometimes the case that two or more parent-child relationships are **exclusive**. This is represented using the symbol '⊗', as shown in Fig. 4.1h. An example is given in Fig. 4.2, where states can be aggregated either into regions or into countries. Thus, according to their type, states participate in only one of the relationships departing from the State level.

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. This suggests that the MultiDim model stays on top of the logical level, hiding from the user the implementation details. In other words, the model represents a conceptual data cube. Therefore, we will call the model in Fig. 4.2 as the Northwind data cube.

4.2 Hierarchies

Hierarchies are key elements in analytical applications, since they provide the means to represent the data under analysis at different abstraction levels. In real-world situations, users must deal with complex hierarchies of various kinds. 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. 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. 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. 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. 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

A **balanced hierarchy** has only one path at the schema level, where all levels are mandatory. An example is given by hierarchy Product → Category in Fig. 4.2. At the instance level, the members form a tree where all the branches have the same length, as shown in Fig. 4.3. All parent members have at least one child member, and a child member belongs exactly to one parent member. For example, in Fig. 4.3, each category is assigned at least one product, and a product belongs to only one category.

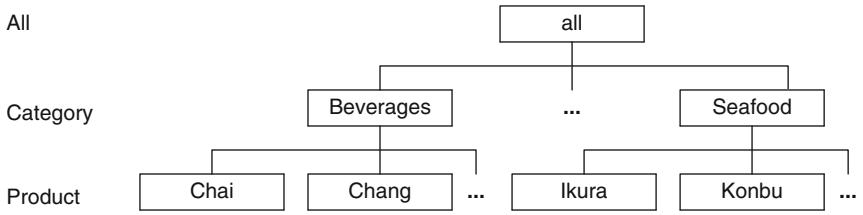

Fig. 4.3 Example of instances of the balanced hierarchy Product → Category in Fig. 4.2 (repeated from Fig. 3.3)

4.2.2 Unbalanced Hierarchies

An **unbalanced hierarchy** has only one path at the schema level, where at least one level is not mandatory. Therefore, at the instance level, there can be parent members without associated child members. 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. 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. For example, Fig. 4.4b shows a branch with no agency and several agencies with no ATM. 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. For example, in Fig. 4.4, every agency belongs to one branch.

Unbalanced hierarchies include a special case that we call **recursive hierarchies**, also called **parent-child hierarchies**. 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).

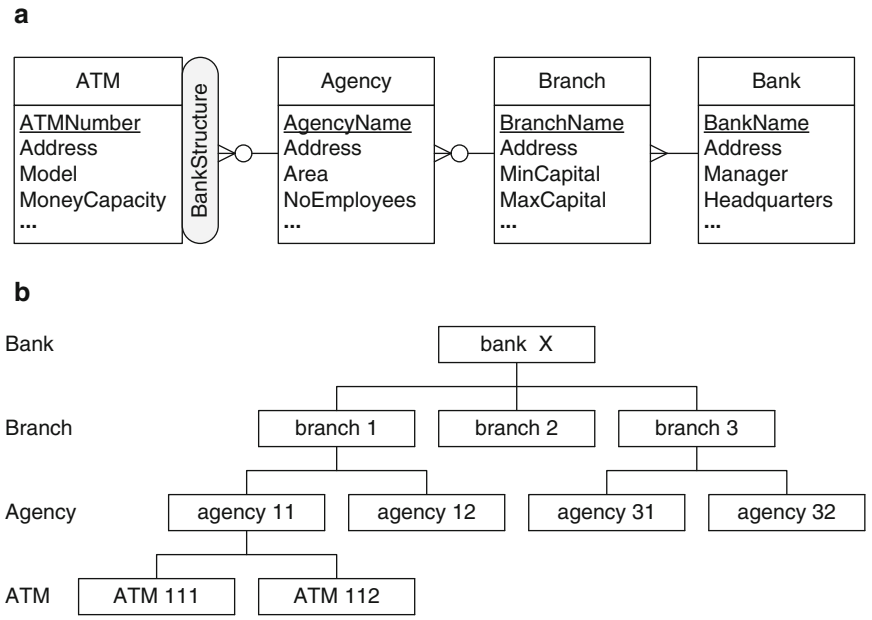

Fig. 4.4 An unbalanced hierarchy. (**a**) Schema. (**b**) Examples of instances

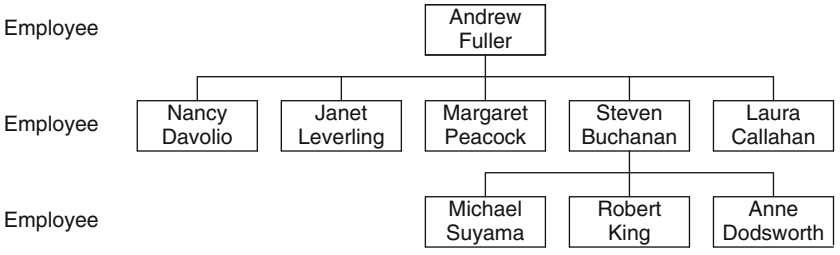

Fig. 4.5 Instances of the parent-child hierarchy in the Northwind data warehouse

An example is given by dimension Employee in Fig. 4.2, which represents an organizational chart in terms of the employee–supervisor relationship. The Subordinate and Supervisor roles of the parent-child relationship are linked to the Employee level. As seen in Fig. 4.5, this hierarchy is unbalanced since employees with no subordinate will not have descendants in the instance tree.

4.2.3 Generalized Hierarchies

Sometimes, the members of a level are of different types. A typical example arises with customers, which can be either companies or persons. Such

a situation is usually captured in an ER model using the generalization relationship studied in Chap. 2. 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. 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. Such hierarchies are called **generalized hierarchies**.

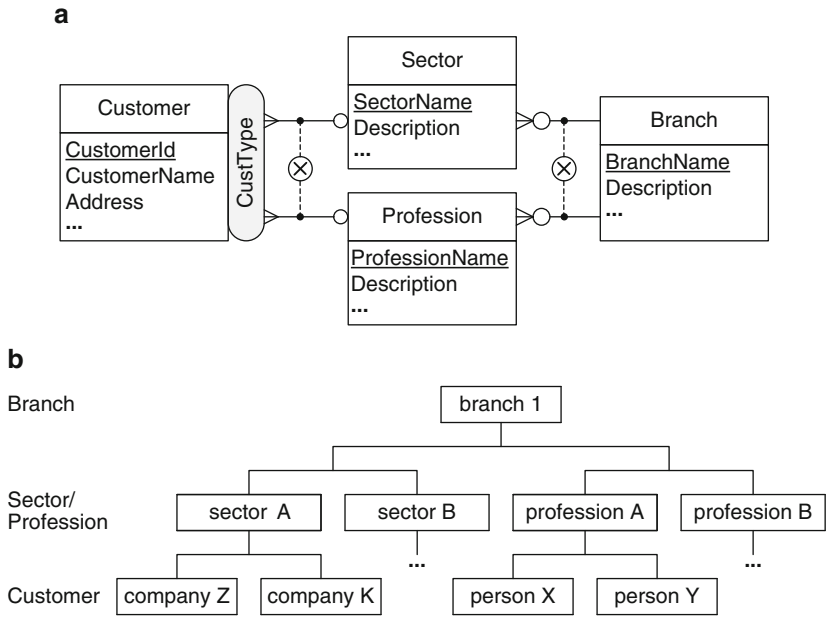

Fig. 4.6 A generalized hierarchy. (**a**) Schema. (**b**) Examples of instances

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. This figure shows the two aggregation paths described above, one for each type of customer, where both belong to the same hierarchy. At the instance level, each member of the hierarchy belongs to only one path, as can be seen in Fig. 4.6b. We use the symbol '⊗' to indicate that the paths are exclusive for every member. Such a notation is equivalent to the *xor* annotation used in UML. The levels at which the alternative paths split and join are called, respectively, the **splitting** and **joining levels**.

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.

Generalized hierarchies are, in general, not summarizable. For example, not all customers are mapped to the Profession level. Thus, the aggregation mechanism should be modified when a splitting level is reached in a roll-up operation.

In generalized hierarchies, it is not necessary that splitting levels are joined. An example is the hierarchy in Fig. 4.7, which is used for analyzing international publications. Three kinds of publications are considered: journals, books, and conference proceedings. The latter can be aggregated to the conference level. However, there is not a common joining level for all paths.

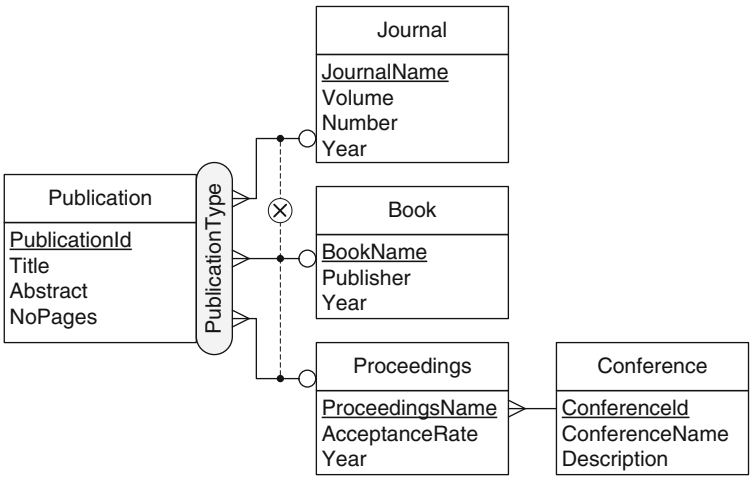

Fig. 4.7 A generalized hierarchy without a joining level

Generalized hierarchies include a special case commonly referred to as **ragged** hierarchies. An example is the hierarchy City → State → Region → Country → Continent given in Fig. 4.2. As can be seen in Fig. 4.8, some countries, such as Belgium, are divided into regions, whereas others, such as Germany, are not. Furthermore, small countries like the Vatican have neither regions nor states. A ragged hierarchy is a generalized hierarchy where alternative paths are obtained by skipping one or several intermediate levels. 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.

4.2.4 Alternative Hierarchies

Alternative hierarchies represent the situation where at the schema level, there are several nonexclusive hierarchies that share at least the leaf level.

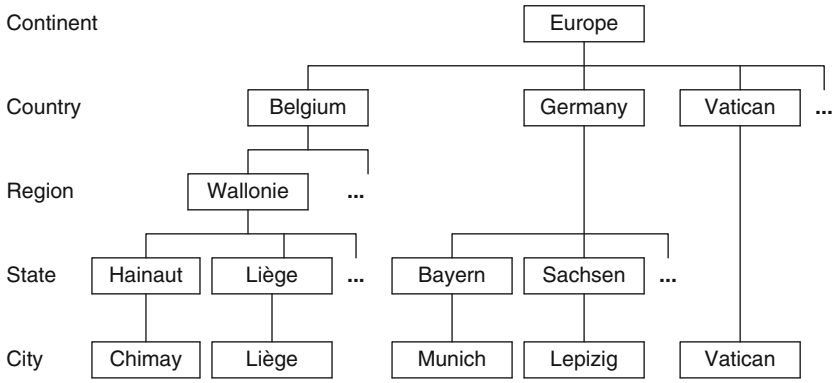

Fig. 4.8 Examples of instances of the ragged hierarchy in Fig. 4.2

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. 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. 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. Alternative hierarchies are needed when we want to analyze measures from a unique perspective (e.g., time) using alternative aggregations.

Note the difference between generalized and alternative hierarchies (see Figs. 4.6 and 4.9). Although the two kinds of hierarchies share some levels, they represent different situations. In a generalized hierarchy, a child member is related to *only one* of the paths, whereas in an alternative hierarchy, a child member is related to *all paths*, and the user must choose one of them for analysis.

4.2.5 Parallel Hierarchies

Parallel hierarchies arise when a dimension has several hierarchies associated with it, accounting for different analysis criteria. Further, the component hierarchies may be of different kinds.

Parallel hierarchies can be **dependent** or **independent** depending on whether the component hierarchies share levels. Figure 4.10 shows an example of a dimension that has two parallel independent hierarchies. The hierarchy ProductGroups is used for grouping products according to categories or departments, while the hierarchy DistributorLocation groups them according to distributors' divisions or regions. On the other hand, the

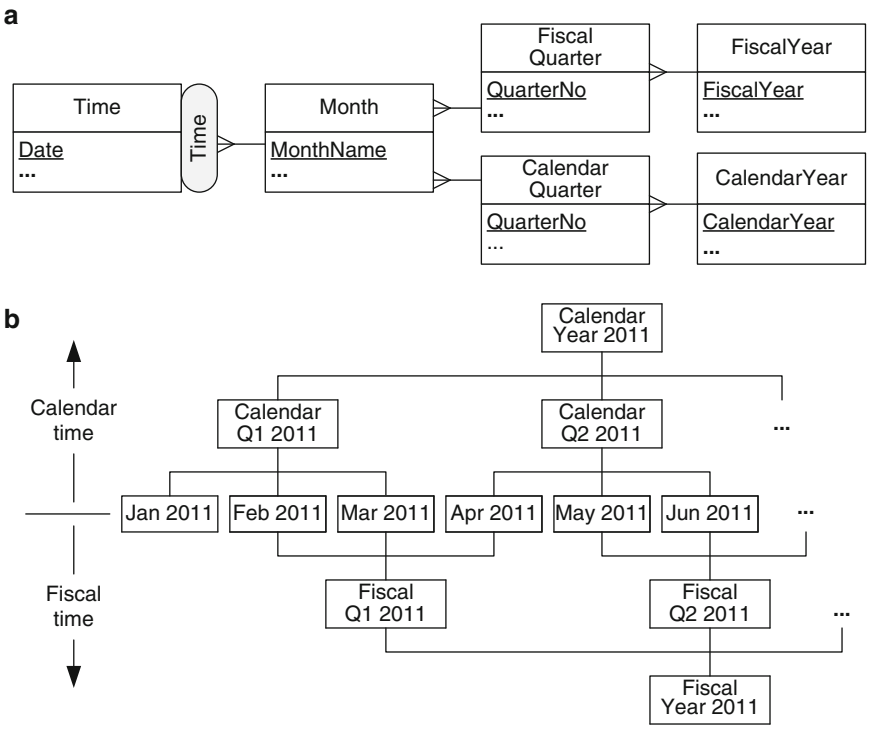

Fig. 4.9 An alternative hierarchy. (**a**) Schema. (**b**) Examples of instances

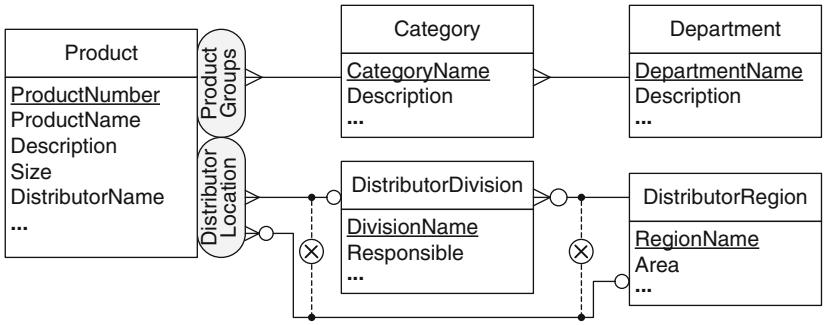

Fig. 4.10 An example of parallel independent hierarchies

parallel dependent hierarchies given in Fig. 4.11 represent a company that requires sales analysis for stores located in several countries. The hierarchy StoreLocation represents the geographic division of the store address, while the hierarchy SalesOrganization represents the organizational division of the company. Since the two hierarchies share the State level, this level plays different roles according to the hierarchy chosen for the analysis. Sharing

levels in a conceptual schema reduces the number of its elements without losing its semantics, thus improving readability. 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. This is the case of StoreLocation and SalesOrganization indicated on level State.

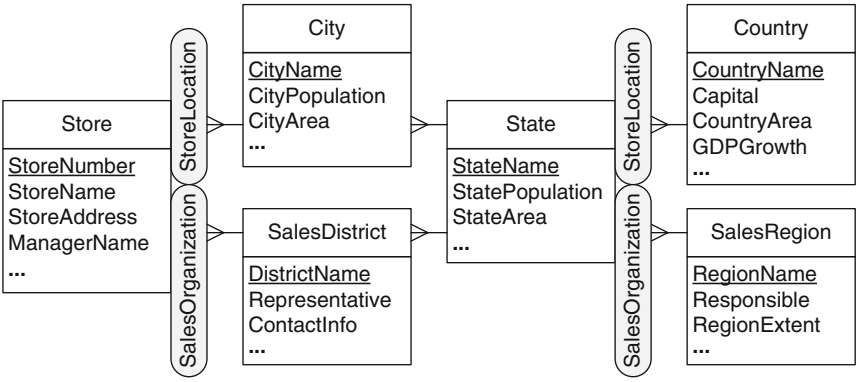

Fig. 4.11 An example of parallel dependent hierarchies

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. This is done by including only one (for alternative hierarchies) or several (for parallel hierarchies) hierarchy names, which account for various analysis criteria. 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, whereas this can be done for parallel hierarchies. For example, for the schema in Fig. 4.11, the user can safely issue a query "Sales figures for stores in city A that belong to the sales district B."

Further, in parallel dependent hierarchies, a leaf member may be related to various different members in a shared level, which is not the case for alternative hierarchies that share levels. For instance, consider the schema in Fig. 4.12, which refers to the living place and the territory assignment of sales employees. 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. 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. For example, suppose that the amount of sales generated by employees E1, E2, and E3 are \$50, \$100, and \$150, respectively. If all employees live in state A, but only E1 and E2 work in this state, aggregating

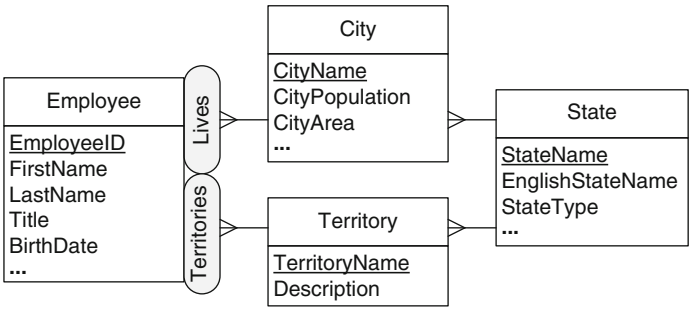

Fig. 4.12 Parallel dependent hierarchies leading to different parent members of the shared level

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. 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. For example, 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**. The fact that a hierarchy is strict or not is orthogonal to its kind. Thus, the hierarchies previously presented can be either strict or nonstrict. We next analyze some issues that arise when dealing with nonstrict hierarchies.

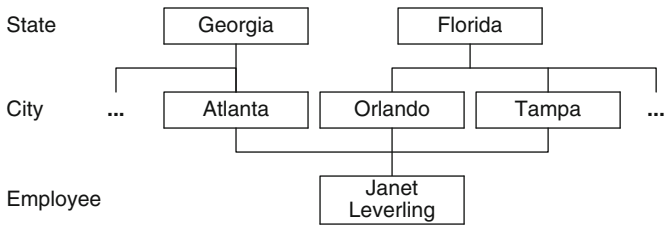

Fig. 4.13 Examples of instances of the nonstrict hierarchy in Fig. 4.2

Figure 4.2 shows a nonstrict hierarchy where an employee may be assigned to several cities. Some instances of this hierarchy are shown in Fig. 4.13. 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. 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.

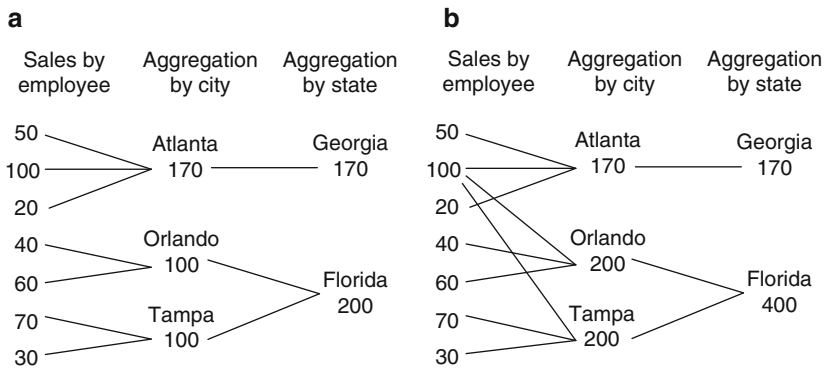

Fig. 4.14 Double-counting problem when aggregating a sales amount measure in Fig. 4.13. (**a**) Strict hierarchy. (**b**) Nonstrict hierarchy

Nonstrict hierarchies induce the problem of **double counting** of measures when a roll-up operation reaches a many-to-many relationship. Let us consider the example in Fig. 4.14, which illustrates sales by employees with aggregations along City and State levels (defined in Fig. 4.13), and employee Janet Leverling with total sales equal to 100. Figure 4.14a shows a situation where the employee has been assigned to Atlanta, in a strict hierarchy scenario. The sum of sales by territory and by state can be calculated straightforwardly, as the figure shows. Figure 4.14b shows a nonstrict hierarchy scenario, where the employee has been assigned the territories Atlanta, Orlando, and Tampa. This approach causes incorrect aggregated results, since the employee's sales are counted three times instead of only once.

One solution to the double-counting problem consists in transforming a nonstrict hierarchy into a strict one by creating a new member for each set of parent members participating in a many-to-many relationship. In our example, a new member that represents the three cities Atlanta, Orlando, and

Tampa will be created. However, in this case, a new member must also be created in the state level, since the three cities belong to two states. Another solution would be to ignore the existence of several parent members and to choose one of them as the primary member. For example, we may choose the city of Atlanta. However, neither of these solutions correspond to the users' analysis requirements, since in the former, artificial categories are introduced, and in the latter, some pertinent analysis scenarios are ignored.

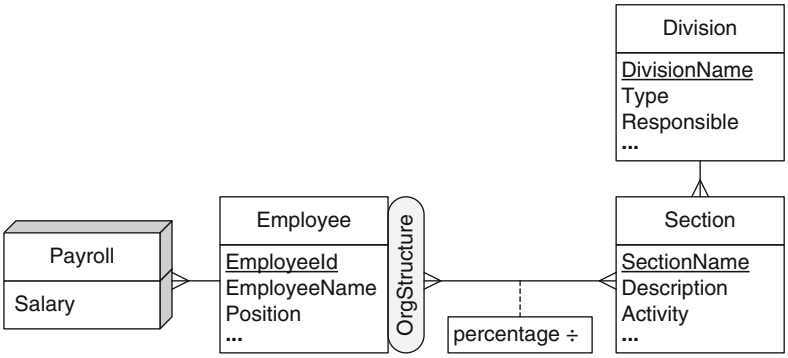

Fig. 4.15 A nonstrict hierarchy with a distributing attribute

An alternative approach to the double-counting problem would be to indicate how measures are distributed between several parent members for many-to-many relationships. For example, Fig. 4.15 shows a nonstrict hierarchy where employees may work in several sections. The schema includes a measure that represents an employee's overall salary, that is, the sum of the salaries paid in each section. Suppose that an attribute stores the percentage of time for which an employee works in each section. In this case, we depict this attribute in the relationship with an additional symbol '÷', indicating that it is a **distributing attribute** determining how measures are divided between several parent members in a many-to-many relationship.

Choosing an appropriate distributing attribute is important in order to avoid approximate results when aggregating measures. For example, suppose that in Fig. 4.15, the distributing attribute represents the percentage of time that an employee works in a specific section. If the employee has a higher position in one section and although she works less time in that section, she may earn a higher salary. Thus, applying the percentage of time as a distributing attribute for measures representing an employee's overall salary may not give an exact result. Note also that in cases where the distributing attribute is unknown, it can be approximated by considering the total number of parent members with which the child member is associated. In the example of Fig. 4.14, since we have three cities with which the employee

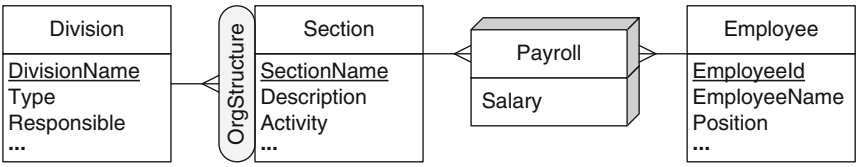

Fig. 4.16 Transforming a nonstrict hierarchy into a strict hierarchy with an additional dimension

Janet Leverling is associated, one-third of the value of the measure will be accounted for each city.

Figure 4.16 shows another solution to the problem of Fig. 4.15 where we transformed a nonstrict hierarchy into independent dimensions. However, this solution corresponds to a different conceptual schema, where the focus of analysis has been changed from employees' salaries to employees' salaries by section. Note that this solution can only be applied when the exact distribution of the measures is known, for instance, when the amounts of salary paid for working in the different sections are known. It cannot be applied to nonstrict hierarchies without a distributing attribute, as in Fig. 4.13.

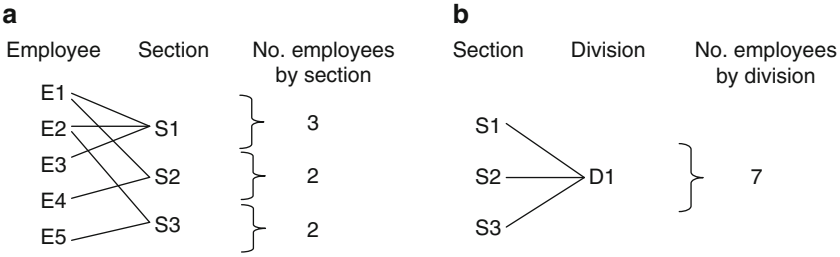

Fig. 4.17 Double-counting problem for a nonstrict hierarchy

Nevertheless, although the solution in Fig. 4.16 aggregates correctly the Salary measure when applying the roll-up operation from the Section to the Division level, the problem of double counting of the same employee will occur. Suppose that we want to use the schema in Fig. 4.16 to calculate the number of employees by section or by division; this value can be calculated by counting the instances of employees in the fact. The example in Fig. 4.17a considers five employees who are assigned to various sections. Counting the number of employees who work in each section gives correct results. However, the aggregated values for each section cannot be reused for calculating the number of employees in every division, since some employees (E1 and E2 in Fig. 4.17a) will be counted twice and the total result will give a value equal to 7 (Fig. 4.17b) instead of 5.

In summary, nonstrict hierarchies can be handled in several ways:

- Transforming a nonstrict hierarchy into a strict one:
  - Creating a new parent member for each group of parent members linked to a single child member in a many-to-many relationship.
  - Choosing one parent member as the primary member and ignoring the existence of other parent members.
  - Transforming the nonstrict hierarchy into two independent dimensions.
- Including a distributing attribute.
- Calculating approximate values of a distributing attribute.

Since each solution has its advantages and disadvantages and requires special aggregation procedures, the designer must select the appropriate solution according to the situation at hand and the users' requirements.

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**. An example is given in Fig. 4.18, where, for instance, sales for the USA might be reported per state, while European sales might be reported per city. As another example, consider a medical data warehouse for analyzing patients, where there is a diagnosis dimension with levels diagnosis, diagnosis family, and diagnosis group. A patient may be related to a diagnosis at the lowest granularity but may also have (more imprecise) diagnoses at the diagnosis family and diagnosis group 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. An example is shown in Fig. 4.19. Since an

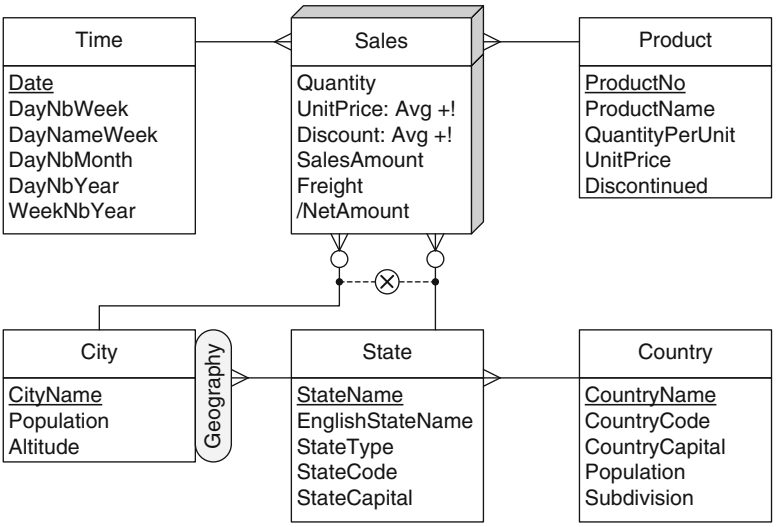

Fig. 4.18 Multiple granularities for the Sales fact

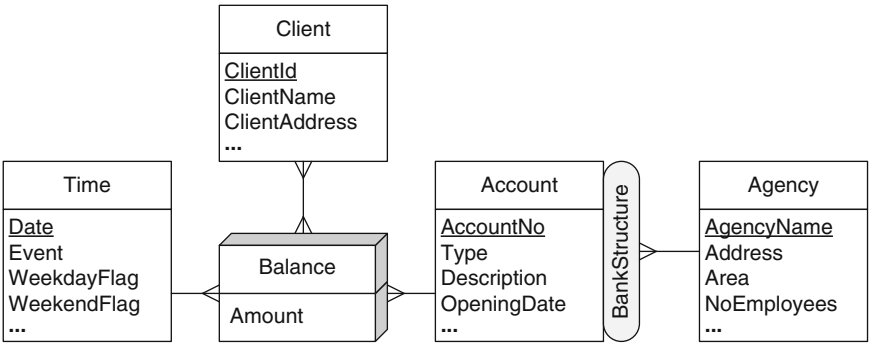

Fig. 4.19 Multidimensional schema for the analysis of bank accounts

account can be jointly owned by several clients, aggregation of the balance according to the clients will count this balance as many times as the number of account holders. For example, as shown in Fig. 4.20, suppose that at some point in time T1 there are two accounts A1 and A2 with balances of, respectively, 100 and 500. Suppose further that both accounts are shared between several clients: account A1 is shared by C1, C2, and C3 and account A2 by C1 and C2. The total balance of the two accounts is equal to 600; however, aggregation (e.g., according to the Time or the Client dimension) gives a value equal to 1,300.

The problem of **double counting** introduced above can be analyzed through the concept of **multidimensional normal forms** (MNFs). MNFs determine the conditions that ensure correct measure aggregation in the

presence of the complex hierarchies studied in this chapter. The first multidimensional normal form (1MNF) requires each measure to be uniquely identified by the set of associated leaf levels. The 1MNF is the basis for correct schema design. To analyze the schema in Fig. 4.19 in terms of the 1MNF, we need to find out the functional dependencies that exist between the leaf levels and the measures. Since the balance depends on the specific account and the time when it is considered, the account and the time determine the balance. Therefore, the schema in Fig. 4.19 does not satisfy the 1MNF, since the measure is not determined by all leaf levels, and thus the fact must be decomposed.

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

Let us recall the notion of multivalued dependency we have seen in Chap. 2. There are two possible ways in which the Balance fact in Fig. 4.19 can be decomposed. In the first one, the same joint account may have different clients assigned to it during different periods of time, and thus the time and the account multidetermine the clients. This situation leads to the solution shown in Fig. 4.21a, where the original fact is decomposed into two facts, that is, AccountHolders and Balance. If the joint account holders do not change over time, clients are multidetermined just by the accounts (but not the time). In this case, the link relating the Time level and the AccountHolders fact can be eliminated. Alternatively, this situation can be modeled with a nonstrict hierarchy as shown in Fig. 4.21b.

Even though the solutions proposed in Fig. 4.21 eliminate the double-counting problem, the two schemas in Fig. 4.21 require programming effort for queries that ask for information about individual clients. The difference lies in the fact that in Fig. 4.21a, a drill-across operation (see Sect. 3.2) between the two facts is needed, while in Fig. 4.21b, special procedures for aggregation in nonstrict hierarchies must be applied. In the case of Fig. 4.21a, since the two facts represent different granularities, queries with drill-across operations are complex, demanding a conversion either from a finer to a coarser granularity (e.g., grouping clients to know who holds a specific balance in an account) or vice versa (e.g., distributing a balance between different account holders). Note also that the two schemas in Fig. 4.21 could represent the information about the percentage of ownership of accounts by customers (if this is

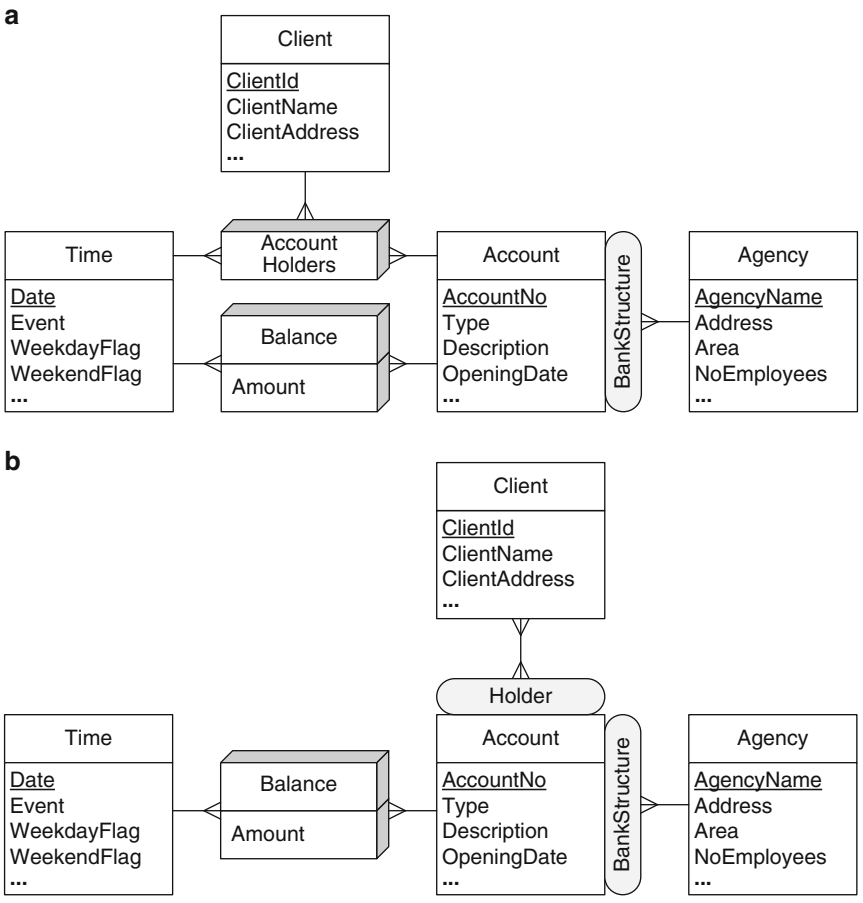

Fig. 4.21 Two possible decompositions of the fact in Fig. 4.19. (**a**) Creating two facts. (**b**) Including a nonstrict hierarchy

known). This could be represented by a measure in the AccountHolders fact in Fig. 4.21a and by a distributing attribute in the many-to-many relationship in Fig. 4.21b.

Another solution to this problem is shown in Fig. 4.22. In this solution, an additional level is created, which represents the groups of clients participating in joint accounts. In the case of the example in Fig. 4.20, two groups should be created: one that includes clients C1, C2, and C3 and another with clients C1 and C2. Note, however, that the schema in Fig. 4.22 is not in the 1MNF, since the measure Balance is not determined by all leaf levels, that is, it is only determined by Time and Account. Therefore, the schema must be decomposed leading to schemas similar to those in Fig. 4.21, with the difference that in

this case, the Client level in the two schemas in Fig. 4.21 is replaced by a nonstrict hierarchy composed of the ClientGroup and the Client levels.

Finally, to avoid many-to-many dimensions, we can choose one client as the primary account owner and ignore the other clients. In this way, only one client will be related to a specific balance, and the schema in Fig. 4.19 can be used without any problems related to double counting of measures. However, this solution may not represent the real-world situation and may exclude from the analysis the other clients of joint accounts.

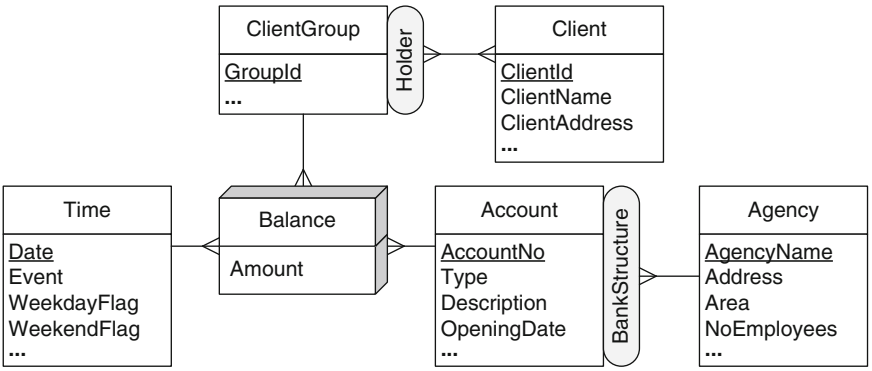

Fig. 4.22 Alternative decomposition of the fact in Fig. 4.19

In summary, many-to-many dimensions in multidimensional schemas can be avoided by using one of the solutions presented in Fig. 4.21. The choice between these alternatives depends on the functional and multivalued dependencies existing in the fact, the kinds of hierarchies in the schema, and the complexity of the implementation.

4.4 Querying the Northwind Cube Using the OLAP Operations

We conclude the chapter showing how the OLAP operations studied in Chap. 3 can be used to answer a series of queries addressed to the Northwind cube in Fig. 4.2. 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))

The ROLLUP* operation is used to specify the levels at which each of the dimensions Customer, OrderDate, and Product are rolled up. For the other dimensions in the cube, a roll-up to All is performed. The SUM operation is applied to aggregate the measure SalesAmount. All other measures of the cube are removed from the result.

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))
```

As in the previous query, a roll-up to the specified levels is performed while performing a SUM operation to aggregate the measure SalesAmount.

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)
```

Here, we first apply a ROLLUP operation to aggregate the measure SalesAmount. Then, a copy of the resulting cube, with the measure renamed as PrevYearSalesAmount, is kept in the cube Sales2. The two cubes are joined with the DRILLACROSS operation, where the join condition ensures that cells corresponding to the same month of two consecutive years and to the same client state are merged in a single cell in the result. Although we include the join condition for the Customer dimension, since it is an equijoin, this is not mandatory—it is assumed by default for all the dimensions not mentioned in the join condition. In the following, we do not include the equijoins in the conditions in the DRILLACROSS operations.

Query 4.4. Monthly sales growth per product, that is, total sales per product compared to those of the previous month.

```
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 )
```

As in the previous query, we first apply a ROLLUP operation, make a copy of the resulting cube, and join the two cubes with the DRILLACROSS operation. However, here the join condition is more involved than in the previous query, since two cases must be considered. In the first one, for the months starting from February (Month > 1), the cells to be merged must be consecutive and belong to the same year. In the second case, the cell corresponding to January must be merged with the one of December from the previous year. Finally, in the last step, we compute a new measure SalesGrowth as the difference between the sales amount of the two corresponding months.

Query 4.5. Three best-selling employees.

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

Here, we roll up all the dimensions of the cube, except Employee, to the All level, while aggregating the measure SalesAmount. Then, the MAX operation is applied while specifying that cells with the top three values of the measure are kept in the result.

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
```

In this query, we roll up the dimensions of the cube as specified. Then, the MAX operation is applied after grouping by Product and OrderDate.

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
```

Here, we roll up the Customer dimension to Country level and the other dimensions to the All level. Then, the TOPPERCENT operation selects the countries that cumulatively account for top 50% of the sales amount.

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))
```

Here, we first roll up the cube to the Employee and Month levels by summing the SalesAmount measure. Then, we perform a second roll-up to the Year level to obtain to overall sales and the average of monthly sales.

Query 4.9. Total sales amount and total discount amount per product and month.

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

Here, we first compute a new measure TotalDisc from three other measures. Then, we roll up the cube to the Product and Month levels.

Query 4.10. Monthly year-to-date 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)
```

Here, we start by performing a roll-up to the category and month levels. Then, a new measure is created by applying the SUM aggregation function to a window composed of all preceding cells of the same year. Notice that it is supposed that the members of the Time dimension are ordered according to the calendar time.

Query 4.11. Moving average over the last 3 months of the sales amount by product category.

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

In the first roll-up, we aggregate the SalesAmount measure by category and month. Then, we compute the moving average over a window containing the cells corresponding to the current month and the two preceding months.

Query 4.12. Personal sales amount made by an employee compared with the total sales amount made by herself and her subordinates during 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)
```

In the first step, we restrict the data in the cube to the year 1997. Then, in the second step, we perform the aggregation of the sales amount measure by employee, thus obtaining the sales figures independently of the supervision hierarchy. Then, in the third step, the obtained measure is renamed. In the fourth step, we apply the recursive roll-up, which performs an iteration over the supervision hierarchy by aggregating children to parent until the top level is reached. Finally, the last step obtains the cube with both measures.

Query 4.13. Total sales amount, number of products, and sum of the quantities sold for each order.

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

Here, we roll up all the dimensions, except Order, to the All level, while adding the SalesAmount and Quantity measures and counting the number of products.

Query 4.14. For each month, total number of orders, total sales amount, and average sales amount by 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)
```

In the query above, we first roll up to the Month and Order levels. Then, we perform another roll-up to remove the Order dimension and obtain the requested measures.

Query 4.15. For each employee, total sales amount, number of cities, and number of states to which she is assigned.

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

Recall that Territories is a nonstrict hierarchy in the Employee dimension. In this query, we roll up to the State level while adding the SalesAmount measure and counting the number of distinct cities and states. Notice that the ROLLUP* operation takes into account the fact that the hierarchy is nonstrict and avoids the double-counting problem to which we referred in Sect. 4.2.6.

4.5 Summary

This chapter focused on conceptual modeling for data warehouses. As is the case for databases, conceptual modeling allows user requirements to be represented while hiding actual implementation details, that is, regardless of the actual underlying data representation. To explain conceptual multidimensional modeling, we used the MultiDim model, which is based on the entity-relationship model and provides an intuitive graphical notation. It is well known that graphical representations facilitate the understanding of application requirements by users and designers.

We have presented a comprehensive classification of hierarchies, taking into account their differences at the schema and at the instance level. We started by describing balanced, unbalanced, and generalized hierarchies, all of which account for a single analysis criterion. Recursive (or parent-child) and ragged hierarchies are special cases of unbalanced and generalized hierarchies, respectively. Then, we introduced alternative hierarchies, which are composed of several hierarchies defining various aggregation paths for the same analysis criterion. We continued with parallel hierarchies, which

are composed of several hierarchies accounting for different analysis criteria. When parallel hierarchies share a level, they are called dependent; otherwise, they are called independent. All the above hierarchies can be either strict or nonstrict, depending on whether they contain many-to-many relationships between parent and child levels. Nonstrict hierarchies define graphs at the instance level. We then presented advanced modeling aspects, namely, facts with multiple granularities and many-to-many dimensions. These often arise in practice but are frequently overlooked in the data warehouse literature. In Chap. 5, we will study how all these concepts can be implemented at the logical level. We concluded showing how the OLAP operations introduced in Chap. 3 can be applied over the conceptual model, using as example a set of queries over the Northwind data cube.

4.6 Bibliographic Notes

Conceptual data warehouse design was first introduced by Golfarelli et al. [65]. A detailed description of conceptual multidimensional models can be found in [203]. Many multidimensional models have been proposed in the literature. Some of them provide graphical representations based on the ER model (e.g., [184, 205]), as is the case of the MultiDim model, while others are based on UML (e.g., [1, 120, 204]). Other models propose new notations (e.g., [67, 88, 207]), while others do not refer to a graphical representation (e.g., [86, 160, 166]). There is great variation in the kinds of hierarchies supported by current multidimensional models. A detailed comparison of how the various multidimensional models cope with hierarchies is given in [126, 158]. Multidimensional normal forms were defined in [113, 114].

The Object Management Group (OMG) has proposed the Common Warehouse Model (CWM)[1] as a standard for representing data warehouse and OLAP systems. This model provides a framework for representing metadata about data sources, data targets, transformations, and analysis, in addition to processes and operations for the creation and management of warehouse data. The CWM model is represented as a layered structure consisting of a number of submodels. One of these submodels, the resource layer, defines models that can be used for representing data in data warehouses and includes the relational model as one of them. Further, the analysis layer presents a metamodel for OLAP, which includes the concepts of a dimension and a hierarchy. In the CWM, it is possible to represent all of the kinds of hierarchies presented in this chapter.

[1] http://www.omg.org/docs/formal/03-03-02.pdf

4.7 Review Questions

4.1 Discuss the following concepts: dimension, level, attribute, identifier, fact, role, measure, hierarchy, parent-child relationship, cardinalities, root level, and leaf level.
4.2 Explain the difference, at the schema and at the instance level, between balanced and unbalanced hierarchies.
4.3 Give an example of a recursive hierarchy. Explain how to represent an unbalanced hierarchy with a recursive one.
4.4 Explain the usefulness of generalized hierarchies. To which concept of the entity-relationship model are these hierarchies related?
4.5 What is a splitting level? What is a joining level? Does a generalized hierarchy always have a joining level?
4.6 Explain why ragged hierarchies are a particular case of generalized hierarchies.
4.7 Explain in what situations alternative hierarchies are used.
4.8 Describe the difference between parallel dependent and parallel independent hierarchies.
4.9 Illustrate with examples the difference between generalized, alternative, and parallel hierarchies.
4.10 What is the difference between strict and nonstrict hierarchies?
4.11 Illustrate with an example the problem of double counting of measures for nonstrict hierarchies. Describe different solutions to this problem.
4.12 What is a distributing attribute? Explain the importance of choosing an appropriate distributing attribute.
4.13 What does it mean to have a fact with multiple granularities?
4.14 Relate the problem of double counting to the functional and multivalued dependencies that hold in a fact.
4.15 Why must a fact be decomposed in the presence of dependencies? Show an example of a fact that can be decomposed differently according to the dependencies that hold on it.

4.8 Exercises

4.1 Design a MultiDim schema for an application domain that you are familiar with. Make sure that the schema has a fact with associated levels and measures, at least two hierarchies, one of them with an exclusive relationship, and a parent-child relationship with a distributing attribute.
4.2 Design a MultiDim schema for the telephone provider application in Ex. 3.1.
4.3 Design a MultiDim schema for the train application in Ex. 3.2.

4.4 Design a MultiDim schema for the university application given in Ex. 3.3 taking into account the different granularities of the time dimension.

4.5 Design a MultiDim schema for the French horse race application given in Ex. 2.1. With respect to the races, the application must be able to display different statistics about the prizes won by owners, by trainers, by jockeys, by breeders, by horses, by sires (i.e., fathers), and by damsires (i.e., maternal grandfathers). With respect to the bettings, the application must be able to display different statistics about the payoffs by type, by race, by racetrack, and by horses.

4.6 In each of the dimensions of the multidimensional schema of Ex. 4.5, identify the hierarchies (if any) and determine its type.

4.7 Design a MultiDim schema for the Formula One application given in Ex. 2.2. With respect to the races, the application must be able to display different statistics about the prizes won by drivers, by teams, by circuit, by Grand Prix, and by season.

4.8 Consider a time dimension composed of two alternative hierarchies: (a) day, month, quarter, and year and (b) day, month, bimonth, and year. Design the conceptual schema of this dimension and show examples of instances.

4.9 Consider the well-known Foodmart cube whose schema is given in Fig. 4.23. Write using the OLAP operations the following queries[2]:

(a) All measures for stores.
(b) All measures for stores in the states of California and Washington summarized at the state level.
(c) All measures for stores in the states of California and Washington summarized at the city level.
(d) All measures, including the derived ones, for stores in the state of California summarized at the state and the city levels.
(e) Sales average in 1997 by store state and store type.
(f) Sales profit by store and semester in 1997.
(g) Sales profit percentage by quarter and semester in 1997.
(h) Sales profit by store for the first quarter of each year.
(i) Unit sales by city and percentage of the unit sales of the city with respect to its state.
(j) Unit sales by city and percentage of the unit sales of the city with respect to its country.
(k) For promotions other than "No Promotion," unit sales and percentage of the unit sales of the promotion with respect to all promotions.
(l) Unit sales by promotion, year, and quarter.

[2]The queries of this exercise are based on a document written by Carl Nolan entitled "Introduction to Multidimensional Expressions (MDX)."

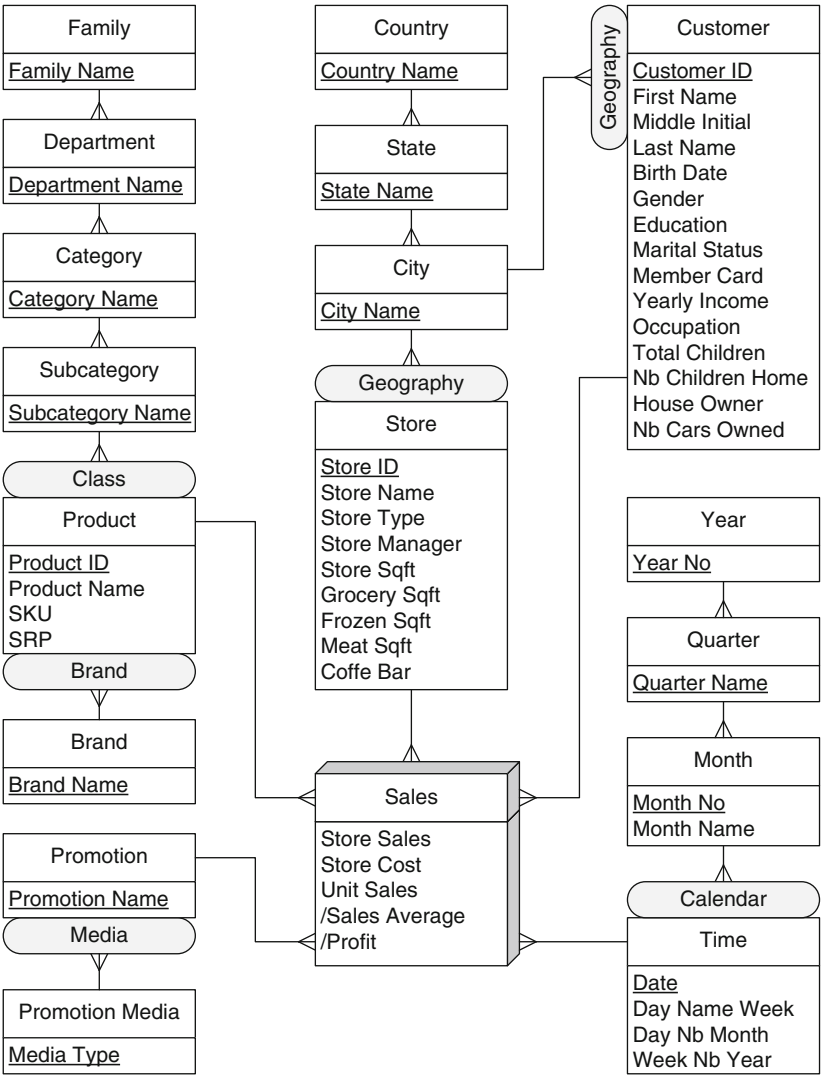

Fig. 4.23 Conceptual schema of the Foodmart cube

(m) Unit sales by promotion and store, for stores in the states of California and Washington.
(n) Sales profit by month and sales profit growth with respect to the previous month.
(o) Sales profit by month and sales profit growth with respect to the same month of the previous year.

(p) Sales profit by month and percentage profit growth with respect to the previous month.
(q) For every month in 1997, unit sales and unit sales difference with respect to the opening month of the quarter.
(r) Monthly year-to-date sales by product subcategory in 1997.
(s) Unit sales by product subcategory, customer state, and quarter.
(t) Sales profit in 1997 by store type and store city, for cities whose unit sales in 1997 exceeded 25,000.
(u) Sales profit in 1997 by store type and store city, for cities whose profit percentage in 1997 is less than the one of their state.
(v) All measures for store cities between Beverly Hills and Spokane (in the USA) sorted by name regardless of the hierarchy.
(w) All measures for store cities sorted by descending order of sales count regardless of the hierarchy.
(x) All measures for the top-five store cities based on sales count.
(y) All measures for the top-five store cities based on sales count and all measures for all the other cities combined.
(z) Store cities whose sales count accounts for 50% of the overall sales count.
(aa) For store cities whose sales count accounts for 50% of the overall sales count, unit sales by store type.
(bb) Unit sales and number of customers by product subcategory.
(cc) Number of customers and number of female customers by store.
(dd) For each product subcategory, maximum monthly unit sales in 1997 and the month when that occurred.
(ee) For 1997 and by brand, total unit sales, monthly average of unit sales, and number of months involved in the computation of the average.

http://www.springer.com/978-3-642-54654-9

Data Warehouse Systems
Design and Implementation
Vaisman, A.; Zimányi, E.
2014, XVI, 625 p. 133 illus., Hardcover
ISBN: 978-3-642-54654-9