From ER Models to Dimensional Models Part II: Advanced Design Issues

Daniel L. Moody

Department of Software Engineering
Charles University
Prague, Czech Republic
email: moody@ksint.ms.mff.cuni.cz

School of Business Systems,
Monash University
Melbourne, Australia 3800
email: dmoody@infotech.monash.edu.au

Mark A.R. Kortink

Kortink & Associates
1 Eildon Rd, St Kilda, Melbourne, Australia 3182
mark@kortink.com

1. INTRODUCTION

The first article in this series, which appeared in the Summer 2003 issue of the Journal of Business Intelligence (Moody and Kortink, 2003), described how to design a set of star schemas based on a data model represented in Entity Relationship (ER) form. However as in most design problems, there are many exceptions, special cases and alternatives that arise in practice. In this article, we discuss some advanced design issues that need to be considered. These include:

- Alternative dimensional structures: snowflake schemas and starflake schemas
- Slowly changing dimensions
- Minidimensions
- Heterogeneous star schemas (dimensional subtypes)
- Dealing with non–hierarchically structured data in the underlying ER model:
    - Many-to-many relationships
    - Recursive relationships
    - Subtypes and supertypes

The same example data model as used in the first article is used throughout to illustrate these issues.

2. ALTERNATIVE DIMENSIONAL STRUCTURES: STARS, SNOWFLAKES AND STARFLAKES

While the star schema is the most commonly used dimensional structure, there are (at least) two alternative structures which can be used:

- Snowflake schema
- Starflake schema

Snowflake Schema

A snowflake schema is a star schema with fully normalised dimensions – it gets its name because it forms a shape similar to a snowflake. Rather than having a regular structure like a star schema, the "arms" of the snowflake can grow to arbitrary lengths in each direction. A snowflake schema can be produced from a star schema by normalising each dimension table. Alternatively, it can be produced directly from an ER model by following the same steps in producing a star schema but skipping Step 4.2 (Collapse Hierarchies). Instead of collapsing classification entities into component entities to form dimension tables, they form the "arms" of the snowflake. The design of the fact table is the same as for the star schema. Figure 1 shows the snowflake schema which results for the Order Item transaction.

The performance impact of snowflaking depends on the DBMS and/or OLAP tool used

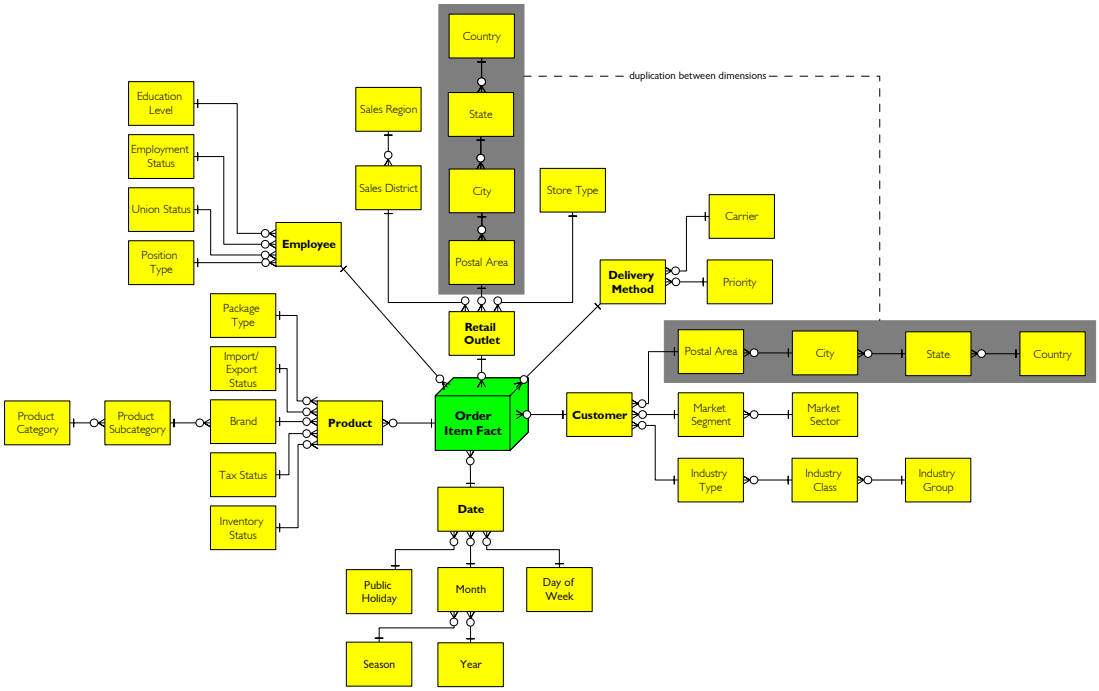

Figure 1. Snowflake Schema for Order Item Transaction

Kimball (1996) argues that "snowflaking" is undesirable because it adds unnecessary complexity, reduces query performance and doesn't substantially reduce storage space. However empirical tests show that the performance impact of snowflaking depends on the DBMS and/or OLAP tool used: some favour snowflakes while others favour star schemas (Spencer and Loukas, 1999). An advantage of the snowflake representation is that it explicitly shows the hierarchical structure of each dimension, which can help in understanding how the data can be sensibly analysed. Thus, whether a snowflake or a star schema is used at the physical level, views should be used to enable the user to see the structure of each dimension as required.

Starflake Schema

A star schema is a dimensional model with fully denormalised hierarchies, while a snowflake schema is a dimensional model with fully normalised hierarchies. However as in many design problems, the best solution is often a balance between two extremes. A *starflake* schema represents a compromise between a star schema and a snowflake schema. It is a star schema which is selectively normalised (or "snowflaked") to remove overlap between dimensions. Overlap between dimensions is undesirable as it increases complexity of load processes and can lead to inconsistent query results if hierarchies become inconsistent. Potential overlap between dimensions can be identified by *branch entities* in the underlying ER model. A branch entity is a classification entity with multiple one-to-many relationships – this indicates a point of intersection between hierarchies. In the example, Postal Area forms a branch entity (Figure 2). Postal Area is the "parent" of Customer and Retail Outlet, which are both components of the Order transaction. In the star schema representation, Postal Area, City, State and Country were included in both the Customer and Retail Outlet dimensions when hierarchies were collapsed.

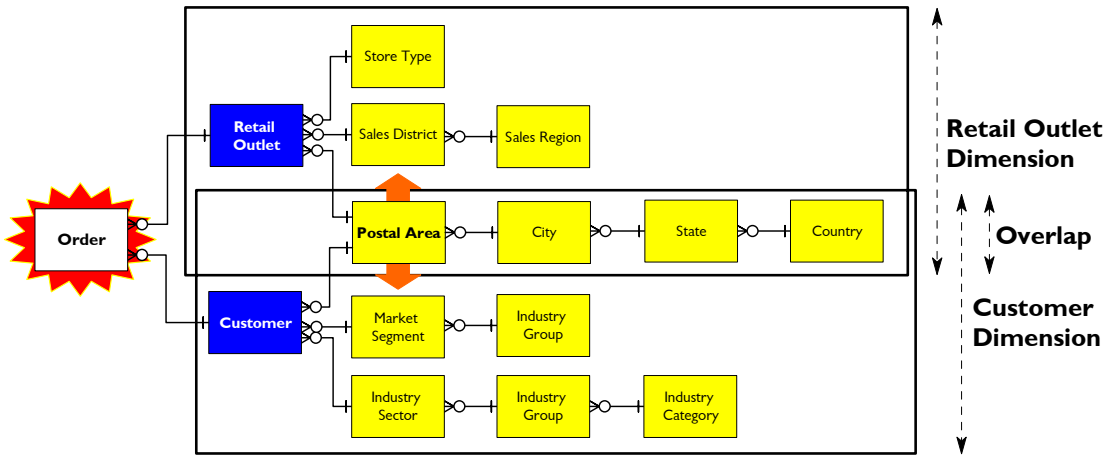

Figure 2. Overlapping Hierarchies in Example Data Model

A starflake schema is a star schema in which shared hierarchical segments are separated out into *subdimension* tables. These represent "highest common factors" between dimensions. A starflake schema is formed by collapsing classification entities from the top of each hierarchy until they reach either a branch entity or a component entity. If a branch entity is reached, a subdimension table is formed. Collapsing then begins again after the branch entity. When a component entity is reached, a dimension table is formed. Figure 3 shows the starflake schema which results for the Order Item transaction. The overlap between the Customer and Retail Outlet dimensions has been removed, with the common Postal Area-City-State-Country hierarchical segment factored out into a shared subdimension table. Again, the design of the fact table is the same as for the star schema.

A **starflake** schema is a star schema which is selectively normalised or "snowflaked" to remove overlap between dimensions

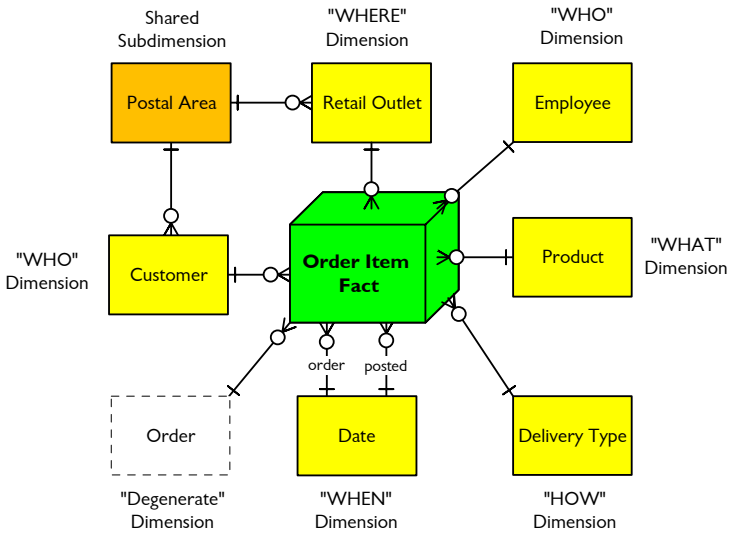

Figure 3. Starflake Schema for Order Item Transaction

Dimensional Design Trade-Offs

The alternative dimensional structures considered here represent different trade-offs between complexity and redundancy:

- The star schema is the simplest structure, as it contains the least number of tables – 8 tables in the example. However it also has the highest level of data redundancy, as the dimension tables all violate third normal form (3NF). This maximises understanding and simplifies queries to pairwise joins between tables.
- The snowflake schema has the most complex structure and consists of more than five times as many tables as the star schema representation (41 in the example). This will require multi-table joins to satisfy queries.
- The starflake schema has a slightly more complex structure than the star schema – 9 tables in the example. However while it has redundancy within each table (the dimension tables and subdimension tables all violate 3NF), redundancy between dimensions is eliminated, thus reducing the possibility of inconsistency between them.

All of these structures are semantically equivalent: they all contain the same data and support the same set of queries. As a result, *views* may be used to construct any of these structures from any other.

3. REFINEMENT OF THE DIMENSIONAL MODEL

Like most design tasks, dimensional modelling tends to be an iterative process. The first cut dimensional model may be refined in various ways to better support historical analysis, simplify user queries or improve query efficiency. Here we discuss some of the most important refinements which may be made to a dimensional model.

Slowly Changing Dimensions

This is one of the most important design issues in dimensional modelling (Kimball, 1996). A *slowly changing dimension* is a dimension table in which a new row is created each time the underlying component entity changes some important characteristic. The purpose of this is to record the state of the dimension entity *at the time each transaction took place*. This is very different to how changes are handled in operational systems: no matter how a customer changes, we want to ensure that we have only one customer row in the customer table. In fact, in CRM systems, a great deal of effort is expended to ensure that duplicate records are not created for the same customer. In a slowly changing dimension, we create a new instance of the customer each time they change address, marital status, employment status etc. Thus each row in a slowly changing dimension does not correspond to a different entity *per se* but a different "state" of that entity – a "snapshot" of the entity at a point in time.

A **slowly changing dimension** is one in which a new row is created each time the underlying entity changes its "state"

To create a slowly changing dimension table, the following design steps are required:

- Define which attributes of the dimension entity need to be tracked over time. This defines the conditions for creating each new dimensional instance.
- Generalise the key of the dimensional table to enable tracking of state changes. Usually this involves adding a version number to the original key of the dimension table.

Apart from the generalised key, the structure of the slowly changing dimension is the same as the original dimension. However insertion and update processes for the table will need to be modified significantly.

Splitting Dimensions: "Minidimensions"

A **minidimension** is a dimension table derived from a larger dimension table which contains a subset of attributes that can be efficiently browsed

In practice, dimension tables often consist of millions of rows, making them unmanageable for browsing purposes. To address this issue, the most heavily used attributes (e.g. demographic fields for customer dimensions) may be separated out into a *minidimension* table. This can improve performance significantly for the most common queries. The minidimension should contain a subset of attributes that can be efficiently browsed. As a rule of thumb, there should be less than 100,000 combinations of attribute values in a minidimension (i.e. less than 100,000 rows) to facilitate efficient browsing (Kimball, 1996). The number of attribute value combinations in a minidimension can be limited by:

- Including attributes in the minidimension that have discrete values (i.e. whose underlying domains consist of a fixed set of values). Demographic classifications like marital status, gender and education level fall into this category.
- Grouping continuously valued attributes into "bands". For example, age could be converted to a set of discrete ranges like child (0-17), young adult (18-29), adult (30-49), mature adult (50-64), senior citizen (65+).

Figure 4 shows how in the Order Item Star Schema, customer demographics could be factored out into a minidimension table. Some of the attributes in the original table have been transformed to reduce the number of rows in the minidimension table:

- Number of Employees and Annual Revenue have been converted to ranges.
- Date of First Order has been converted to Years of Service. This reduces the number of combinations to multiples of years rather than all possible dates.

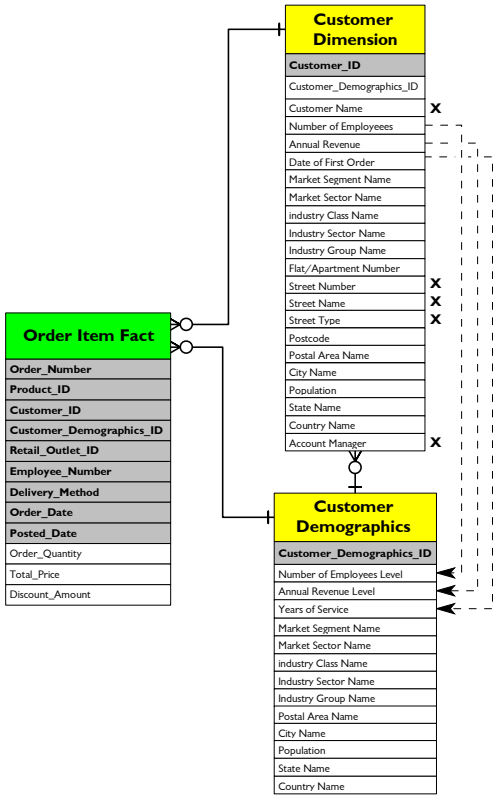

Figure 4. Minidimension Table (Detailed Level Design)

Combining Dimensions

The process of creating separate dimension tables for each component entity can sometimes result in an unmanageably large number of dimensions. This is often the case when there are a large number of classification entities attached directly to the transaction entity, representing very shallow (single level) hierarchies. Rather than create a lot of very small (possibly degenerate) dimension tables, these may be combined into a single dimension, with each row representing a valid combination of values. As a rule of thumb, there should be no more than seven dimensions in each star schema to ensure that it is cognitively manageable in size (following the "seven, plus or minus two" principle).

4. DEALING WITH NON-HIERARCHICAL DATA

A major source of complexity in dimensional modelling is dealing with non-hierarchically structured data. As discussed in the previous article, dimensional models assume an underlying hierarchical structure and therefore exclude data that is naturally non-hierarchical. The transformation procedure described in the previous article effectively "weeds out" non-hierarchically structured data from the underlying ER

model in the first step (classification of entities). So what do we do if important decision making data is stored in the form of many-to-many relationships? In this section, we describe how to handle some particular types of non-hierarchical structures that commonly occur in practice:

- Many-to-many relationships: these define network structures among entities, and cause major headaches in dimensional modelling because they occur so frequently in ER models.
- Recursive relationships: these represent "hidden" hierarchies, in which the levels of the hierarchy are represented in data instances rather than the data structure.
- Generalisation hierarchies: subtypes and supertypes require special handling in dimensional modelling, because of the issue of optional attributes. These represent hierarchies at the meta-data level only – data instances are not hierarchically related to each other so cannot be treated as hierarchies for dimensional modelling purposes.

Many-to-Many Relationships

Most of the complexities involved in "dimensionalising" ER models result from many-to-many relationships

Many-to-many relationships cause major headaches in dimensional modelling for two reasons. Firstly, they define *network structures* and therefore do not fit the hierarchical structure of a dimensional model. Secondly, they occur very commonly in practice. Here we consider three types of many-to-many relationships which commonly occur in practice:

- Time-dependent (history) relationships
- Generic (multiple role) relationships
- Multi-valued dependencies ("true" many-to-many relationships)

To include such relationships in a dimensional model generally requires converting them to one-to-many (hierarchical) relationships.

Type 1. Time-dependent (Historical) Relationships

A special type of many-to-many relationship which occurs very commonly in data warehousing applications is one which records the history of a single-valued relationship or attribute over time. That is, the attribute or relationship has only one value at a specific point in time, but has multiple values over time. As an illustration of this, suppose that history is maintained of employee positions in the example data model. As shown in Figure 5, the many-to-many relationship which results (Employee Position History) breaks the hierarchical chain and Position Type can no longer be collapsed into its associated component entity (Employee).

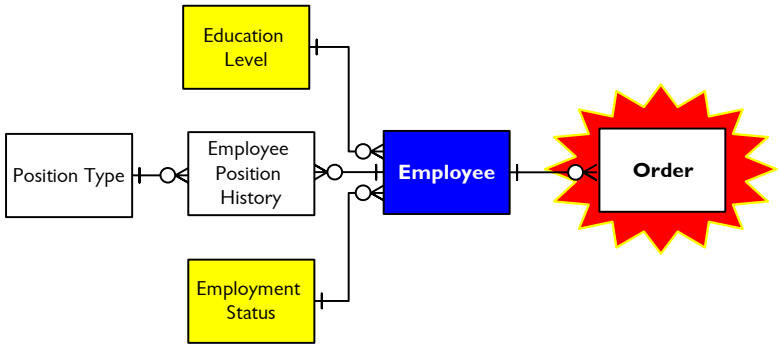

Figure 5. Time-dependent (Historical) Relationship

There are two ways to handle this situation:

- Ignore history: Convert the historical relationship to a "point in time" relationship, which records the *current* value of the relationship or attribute. In the example, this would mean converting Employee Position History to a one-to-many relationship between Employee and Position Type, which records the employee's current position. Position Type can then be collapsed into the Employee entity, and the Employee dimension will record the employee's current position (i.e. at the time of the query). The history of previous positions (including their position at the time of the order) will be lost. A disadvantage of this solution is that it may result in (apparently) inconsistent results to queries about past events.

- Slowly changing dimension: Define Employee as a slowly changing dimension and create a new instance in the Employee dimension table when an employee changes position. This means that Position Type becomes single valued with respect to Employee, since each instance in the Employee table now represents a snapshot at a point in time, and an employee can only have one position at a point in time. Position Type can again be collapsed into the Employee dimension. The difference between this and the previous solution is that the position recorded in the dimension table is the employee's position *at the time of the order*, not at the time of the query.

Type 2. Generic (Multiple Role) Relationships

Another situation which commonly occurs in practice is when a many-to-many relationship is used to represent a fixed number of different types of relationships between the same two entities. These correspond to different *roles* that an entity may play in relation to another entity. For example, in the example data model, an employee may have a number of possible roles in an order:

- They may *receive* the order
- They may *approve* the order
- They may *dispatch* the order

This may be represented in an ER model as a many-to-many relationship with a "role type" entity used to distinguish between the different types of relationships (

Figure 6). This is a recurring pattern in data models in practice (Hay, 1996, Barker, 1990, Allworth, 1999, Simsion and Witt, 2001).

The intersection entity between Employee and Order means that Employee cannot be considered as a component of the Order transaction, and therefore orders cannot be analysed by employee characteristics. To convert such a structure to dimensional form, the different types of relationships or *roles* represented by the generic relationship need to be factored out into separate one-to-many relationships (

Figure 6). Once this is done, Employee becomes a component of the Order transaction, and can form a dimension in the resulting star schema.

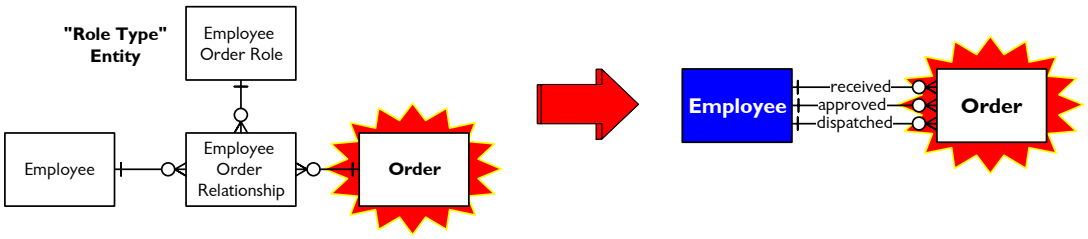

Figure 6. Generic (Multiple Role) Relationship Converted to Specific Relationships

Type 3. Multi-Valued Dependency (True Many-to-Many Relationship)

The final type of many-to-many relationship is when a true multi-valued dependency (MVD) exists between two entities: that is, when many entities of one type can be associated *in the same type of relationship* with many entities of another type *at a single point in time*. For example, in Figure 7, each customer may be involved in multiple industries. The intersection entity Customer Industry "breaks" the hierarchical chain and the industry hierarchy cannot be collapsed into the Customer component entity.

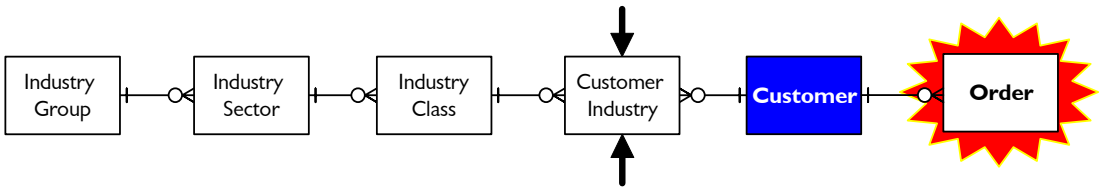

Figure 7. Multi-Valued Dependency (MVD)

One way of handling this is to convert the Customer Industry relationship to a one-to-many relationship, by identifying the main or "principal" industry for each customer. While each customer may be involved in many industries, there will generally be one industry in which they are primarily involved (e.g. earn most of their revenue). This converts the relationship into a one-to-many relationship, which means it can then be collapsed into the Customer table (Figure 8). Manual conversion effort may be required to identify which is the main industry if this is not recorded in the underlying production system.

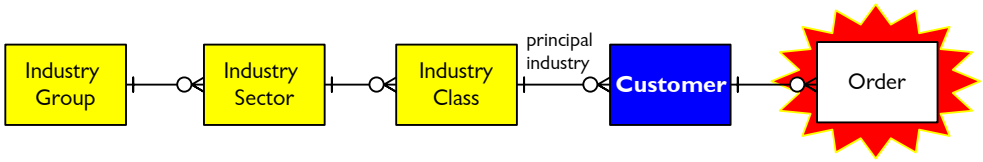

Figure 8. Conversion of Many-to-Many Relationship to One-to-Many Relationship

Recursive Relationships

Hierarchies are commonly represented in ER models using *recursive relationships* (Hay, 1996, Simsion and Witt, 2001). Using a recursive relationship, the levels of the hierarchy are represented as data instances rather than as entities. This gives a much more flexible structure, which can more easily handle changes in the levels of the hierarchy (Moody, 1996). However such structures are less useful in a data warehousing environment, as they reduce understandability to end users and increase complexity of queries (Spencer and Loukas, 1999). In converting an ER model to dimensional form, recursive relationships must be converted to explicit hierarchies, with each level shown as a separate entity. For example, the industry classification hierarchy in the example data model may be shown in ER form as a recursive relationship (Figure 9). To convert this to dimensional form, each row in the Industry Classification Type entity becomes a separate entity. Once this is done, the levels of the hierarchy (which become classification entities) can be easily collapsed to form dimensions.

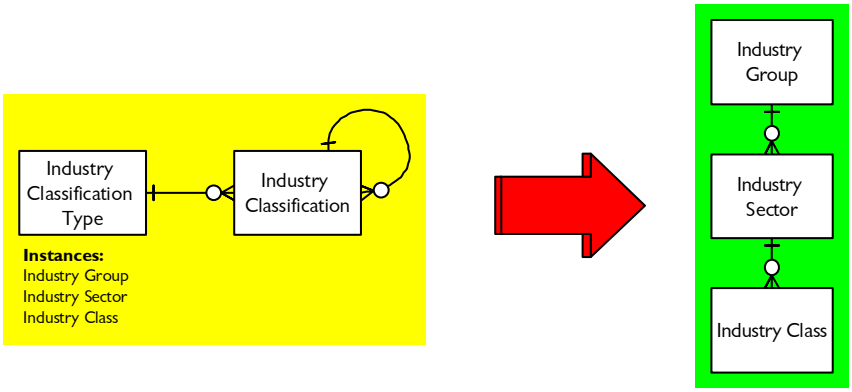

Figure 9. Conversion of Recursive Relationship to Explicit Hierarchy

Generalisation Hierarchies: Subtypes and Supertypes

In the simplest case, supertype/subtype relationships can be converted to dimensional form by merging the subtypes into the supertype and creating a "type" entity to distinguish between them. This can then be converted to a dimensional model in a straightforward manner as it forms a simple (two level) hierarchy. This will result in a dimension table with optional attributes for each subtype. This is the recommended approach when there is a relatively small number of subtype specific attributes and/or relationships.

In the more complex case – when there are many subtype-specific attributes and when different transaction entity attributes are applicable for different subtypes – separate dimensional models may need to be created for the supertype and each of the subtypes. These are called *heterogeneous star schemas*, and are the dimensional equivalent of subtypes and supertypes. In general, this will result in n+1 star schemas, where n is the number of subtypes (see Figure 10):

- One Core Star Schema ("dimensional supertype"): This will consist of a *core fact table*, a *core dimension table* plus other (non-subtyped) dimension tables. The core dimension table will contain all attributes of the supertype while the core fact table will contain transaction attributes (facts) that are common to all subtypes.

- Multiple Custom Star Schemas ("dimensional subtypes"): A separate Customer Star Schema should be optionally created for each subtype in the underlying ER model. Each custom star schema will consist of a *custom fact table*, a *custom dimension table* plus other (non-subtyped) dimension tables. Each custom dimension table will contain all common attributes plus attributes specific to that subtype. The custom fact table will contain all common facts plus facts applicable only to that subtype.

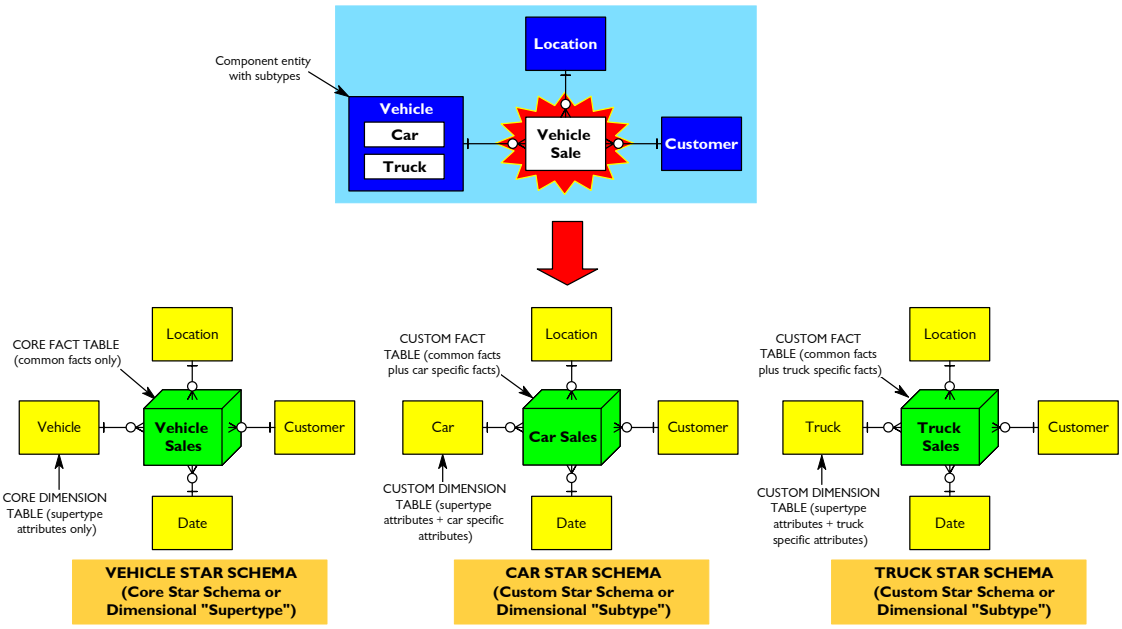

Figure 10. Heterogeneous Star Schemas ("Dimensional Subtyping")

5. CONCLUSION

Of course, there are many other ways a dimensional model can be refined and many other exceptions which can arise in practice. However it is impossible to prescribe how to handle all such situations in advance. In this article, we have focused on the most important issues you are likely to encounter in practice. Being aware of these should be enough to get you to a workable dimensional model in most situations. Beyond this, there is no substitute for experience…

REFERENCES

Allworth, S. (1999) "Classification Structures Encourage the Growth of Generic Industry Models", In **Proceedings of the Eighteenth International Conference on Conceptual Modelling (Industrial Track)**(Ed, Moody, D. L.), Springer, Paris, France.

Barker, R., **CASE*Method: Entity Relationship Modelling**, Addison-Wesley Professional, Wokingham, 1990.

Hay, D.C., **Data Model Patterns: Conventions of Thought**, Dorset House, New York, 1996.

Kimball, R., **The Data Warehouse Toolkit**, John Wiley and Sons, New York, 1996.

Moody, D.L., "The Seven Habits of Highly Effective Data Modellers", **Database Programming and Design**, 1996.

Moody, D.L. and Kortink, M.A.R., "From ER Models to Dimensional Models: Bridging the Gap between OLTP and OLAP Design", **Journal of Business Intelligence**, 8, 2003.

Simsion, G.C. and Witt, G.C., **Data Modeling Essentials: Analysis, Design, and Innovation (2nd Ed)**, The Coriolis Group, 2001.
Spencer, T. and Loukas, T., "From Star to Snowflake to ERD", **Enterprise Systems Journal**, 1999.