From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design

Daniel L. Moody

Department of Information Systems,
University of Melbourne, Parkville, Australia 3052
email: d.moody@ dis.unimelb.edu.au

Mark A.R. Kortink

Simsion Bowles & Associates
1 Collins St, Melbourne, Australia 3000.

Abstract

This paper describes a method for developing dimensional models from traditional Entity Relationship models. This can be used to design data warehouses and data marts based on enterprise data models. The first step of the method involves classifying entities in the data model into a number of categories. The second step involves identifying hierarchies that exist in the model. The final step involves collapsing these hierarchies and aggregating transaction data to form dimensional models. A number of design alternatives are presented, including a *flat schema*, a *terraced schema,* a *star schema* and a *snowflake schema.* We also define a new type of schema called a *star cluster schema*. This is a restricted form of snowflake schema, which minimises the number of tables while avoiding overlap between different dimensional hierarchies. Individual schemas can be collected together to form *constellations* or *galaxies*. We illustrate the method using a simple example.

1. INTRODUCTION

The Problem Addressed

Data warehousing is currently one of the most important applications of database technology in practice. The total size of the data warehousing market, including software and hardware, was estimated to be US$8 billion in 1998 (Sen and Jacob, 1998). A significant proportion of IT budgets in most organisations is devoted to data warehousing applications. High levels of user satisfaction and return on investment have been reported in the literature for such applications (Graham et al, 1996).

One of the most important issues in data warehousing is how to design appropriate database structures to support end user queries (Pokorny, 1999). Existing approaches to data warehouse design advocate a "first principles" approach, where the structure of the data warehouse is derived directly from user query requirements. In this paper, we describe a method for designing data warehouses and data marts based on an enterprise data model represented in Entity Relationship form. This provides a more structured approach to data warehouse design, and ensures that structure of the data warehouse reflects the underlying semantic structure of the data. It also leads to a more flexible warehouse design, which is resilient to changes in analysis requirements over time.

Data Warehouse Architecture

In the early 90's, *data warehousing* was proposed as a general purpose solution to the problem of satisfying organisational management information needs. A data warehouse is a database which provides a single consistent source of management information for reporting and analysis across the organisation (Inmon, 1993; Love, 1994). Data warehousing requires a major shift in the relationship between IT departments and users, in that it advocates a "self-service" model rather than the traditional report-driven model. In a data warehousing environment, end users access data directly using user friendly query tools rather than relying on reports generated by IT specialists. This helps to reduce user reliance on IT staff to satisfy information needs.

Data warehousing is based on a supply chain metaphor. The data "product" is obtained from data "suppliers" (operational systems or external sources) and is temporarily stored in a central data "warehouse". The

Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2000)
Stockholm, Sweden, June 5-6, 2000
(M. Jeusfeld, H. Shu, M. Staudt, G. Vossen, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-28/

data is then delivered via data “marts” to data “consumers” (end users). Figure 1 shows a generic architecture for a data warehouse (rectangles indicate data stores, while circles indicate processes).

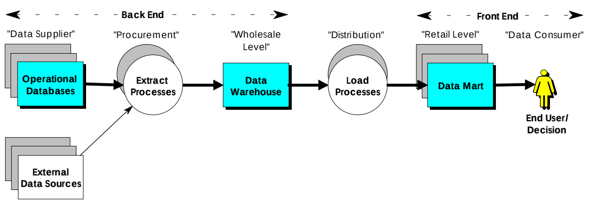

Figure 1. Data Warehouse Architecture

The architecture consists of the following components:

- Operational systems: these are systems which record details of business transactions. This is where most of the data required for decision support is produced.
- External sources: data warehouses often incorporate data from external sources (e.g. census data, economic data) to support analysis
- Extract processes: these processes “stock” the data warehouse with data on a regular basis (daily, weekly, monthly). Data is extracted from different sources, consolidated and reconciled together and stored in a consistent format. This corresponds to a procurement function.
- Central data warehouse: this acts as the central source of decision support data across the enterprise. This forms the “wholesale level” of the data warehouse environment and is used to supply data marts. The central data warehouse is usually implemented using a traditional relational DBMS.
- Load processes: these processes distribute data from the central data warehouse to the data marts. This corresponds to a distribution function.
- Data marts: These represent the “retail outlets” of the data warehouse which provide data in usable form for analysis by end users. Data marts are usually tailored to the needs of a specific group of users or decision making task. They may be “real” (stored as actual tables populated from the central data warehouse) or virtual (defined as views on the central data warehouse). Data marts may be implemented using traditional relational DBMS or OLAP tools.
- End users: write queries and analyses against data stored in data marts using “user friendly” query tools.

Dimensional Modelling

According to Kimball (1996, 1997), the data warehousing (OLAP[1]) environment is profoundly different from the operational (OLTP[2]) environment and techniques used to design operational databases are inappropriate for designing data warehouses. For this reason, Kimball proposed a new technique for data modelling specifically for designing data warehouses, which he called *dimensional modelling*. The method was developed based on observations of practice, and in particular, of data vendors who are in the business of providing data in “user-friendly” form to their customers. It is not based on any theory, and has never been empirically tested, but has clearly been very successful in practice. Dimensional modelling has been adopted as the predominant approach to designing data warehouses and data marts in practice, and represents an important contribution to the discipline of data modelling and database design.

Objectives of this Paper

Kimball argues that modelling in a data warehousing environment is radically different from modelling in an operational (transaction processing) environment, and that you must forget all you know about Entity Relationship modelling.

> “Entity relation models are a disaster for querying because they cannot be understood by users and cannot be navigated usefully by DBMS software. Entity relation models cannot be used as the basis for enterprise data warehouses”

In this paper, we argue that Entity Relationship modelling is equally applicable in data warehousing context as in an operational context and provides a useful basis for designing both data warehouses and data marts.

2. DIMENSIONAL MODELLING CONCEPTS

Objectives of Dimensional Modelling

There are two major differences between operational databases and data warehouses:

- End user access: In a data warehousing environment, users write queries directly against the database structure, whereas in an operational environment, users generally only access the database through an application system “front end”. In a traditional application system, the structure of the database is invisible to the user.
- Read-only: data warehouses are effectively read only databases—users can retrieve and analyse data,

[1] OLAP = On-Line Analytical Processing
[2] OLTP = On-Line Transaction Processing

but cannot update it. Data stored in the data warehouse is updated via batch extract processes.

The problem with using traditional database design techniques in a data warehousing environment is that it results in database structures which are too complex for end users to understand and use. A typical operational database consists of hundreds of tables linked by a complex web of relationships. Even quite simple queries will require multi-table joins, which are error-prone and beyond the capabilities of non-technical users. This is not a problem in transaction processing systems because the complexity of the database structure is hidden from the user by a layer of software.

A major reason for the complexity of operational databases is the use of normalisation. Normalisation tends to multiply the number of tables required, as it requires splitting out functionally dependent attributes into separate tables. The objective of normalisation is to minimise data redundancy (Codd, 1970). This maximises update efficiency because each change can be made in a single place, but tends to penalise retrieval (Kent, 1978). Redundancy is less of an issue in a data warehousing environment because data is not updated on-line.

The objective of dimensional modelling is to produce database structures that are easy for end users to understand and write queries against. A secondary objective is to maximise the efficiency of queries. It achieves these objectives primarily by minimising the number of tables and relationships between them. This reduces the complexity of the database and minimises the number of joins required in user queries.

Star Schemas

The basic building block used in dimensional modelling is the *star schema*. A star schema consists of one large central table called the *fact table*, and a number of smaller tables called *dimension tables* which radiate out from the central table (Figure 2).

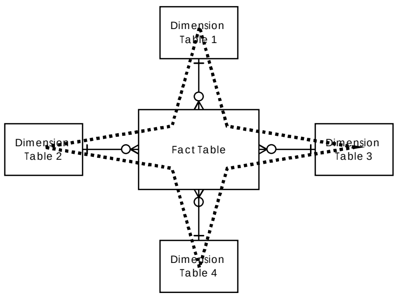

Figure 2. Star Schema (Generic Structure)

The fact table forms the "centre" of the star, while the dimension tables form the "points" of the star. A star schema may have any number of dimensions.

- The fact table contains *measurements* (e.g. price of products sold, quantity of products sold) which may be aggregated in various ways.
- The dimension tables provide the basis for aggregating the measurements in the fact table.
- The fact table is linked to all the dimension tables by one-to-many relationships
- The primary key of the fact table is the concatenation of the primary keys of all the dimension tables.

A more concrete example of a star schema is shown in Figure 3. In this example, sales data may be analysed by product, customer, retail outlet and date.

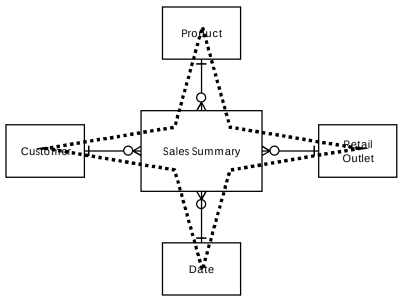

Figure 3. Star Schema Example

Dimension tables are often highly denormalised tables, and generally consist of embedded *hierarchies*. For example, Customer, which represents a single dimension in the star schema above, consists of three independent hierarchies, when they are normalised out (Figure 4).

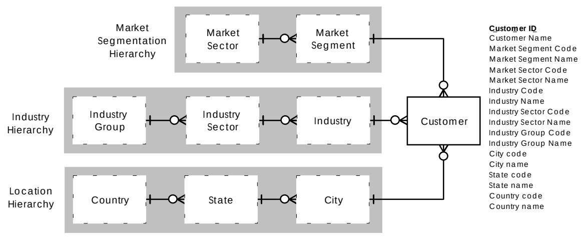

Figure 4. Embedded Hierarchies in Customer Dimension

The advantage of using star schemas to represent data is that it reduces the number of tables in the database and the number of relationships between them and therefore the number of joins required in user queries. Kimball (1996) claims that use of star schemas to design data warehouses results in 80% of queries being single table browses. Star schemas may either be implemented in specialist OLAP tools, or using traditional relational DBMS.

Star Schema Design Approach

Kimball's design method is a "first principles" approach, which is based on analysis of user query requirements. It begins by identifying the relevant "facts" that need to be aggregated, the dimensional attributes to aggregate by, and forming star schemas based on these. It results in a data warehouse design which is a set of discrete star schemas. However there are a number of practical problems with this approach:

- User analysis requirements are highly unpredictable and subject to change over time, which provides an unstable basis for design
- It can lead to incorrect designs if the designer does not understand the underlying relationships in the data
- It results in loss of information through premature aggregation, which limits the ways in which data can be analysed
- The approach is presented by examples rather than via an explicit design procedure

The approach described in this paper overcomes these problems by using an enterprise data model as the basis for data warehouse design. This makes use of the relationships in the data which have been already documented, and provides a much more structured approach to developing a data warehouse design.

3. DATA WAREHOUSE DESIGN APPROACH

We argue that different design principles should be used for designing the central data warehouse and data marts:

Central Data Warehouse Design

This represents the "wholesale" level of the data warehouse, which is used to supply data marts with data. The most important requirement of the central data warehouse is that it provides a consistent, integrated and flexible source of data. We argue that traditional data modelling techniques (Entity Relationship models and normalisation) are most appropriate at this level. A normalised database design ensures maximum consistency and integrity of the data. It also provides the most flexible data structure—new data can be easily added to the warehouse in a modular way, and the database structure will support any analysis requirements. Aggregation or denormalisation at this stage will lose information and restrict the kind of analyses which can be carried out. An enterprise data model, if one exists, should be used as the basis for structuring the central data warehouse.

Data Mart Design

Data marts represent the "retail" level of the data warehouse, where data is accessed directly by end users. Data is extracted from the central data warehouse into data marts to support particular analysis requirements. The most important requirement at this level is that data is structured in a way that is easy for users to understand and use. For this reason, dimensional modelling techniques are most appropriate at this level. This ensures that data structures are as simple as possible in order to simplify user queries. The next section describes an approach for developing dimensional models from an enterprise data model.

4. FROM ENTITY RELATIONSHIP MODELS TO DIMENSIONAL MODELS

This section describes a method for developing dimensional models from Entity Relationship models. This can be used as a basis for designing data marts based on an enterprise data model.

Example Data Model

A simple example is used to illustrate the design approach. Figure 5 shows an operational data model for a sales application. The highlighted attributes indicate the primary keys of each entity.

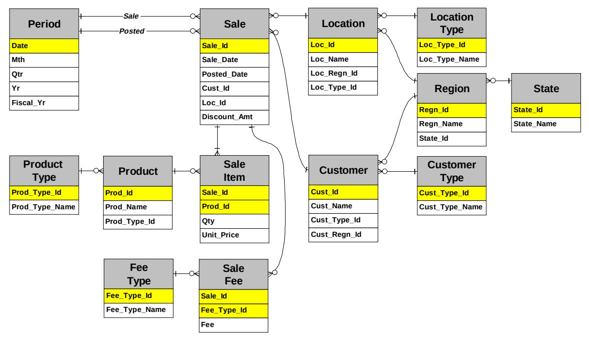

Figure 5. Example Data Model

Such a model is typical of data models that are used by operational (OLTP) systems. Such a model is well suited to a transaction processing environment. It contains no redundancy, thus maximising efficiency of updates, and explicitly shows all the data and the relationships between them. Unfortunately most decision makers would find this schema incomprehensible. Even quite simple queries require multi-table joins and complex subqueries. As a result, end users will be dependent on technical specialists to write queries for them.

Step 1. Classify Entities

The first step in producing a dimensional model from an Entity Relationship model is to classify the entities into three categories:

Transaction Entities

Transaction entities record details about particular events that occur in the business—for example, orders, insurance claims, salary payments and hotel bookings. Invariably, it is these events that decision makers want to understand and analyse. The key characteristics of a transaction entity are:

- It describes an event that happens at a point in time
- It contains measurements or quantities that may be summarised e.g. dollar amounts, weights, volumes.

For example, an insurance claim records a particular business event and (among other things) the amount claimed. Transaction entities are the most important entities in a data warehouse, and form the basis for constructing *fact tables* in star schemas. Not all transaction entities will be of interest for decision support, so user input will be required in identifying which transactions are important.

Component Entities

A component entity is one which is directly related to a transaction entity via a one-to-many relationship. Component entities define the details or "components" of each business transaction. Component entities answer the "who", "what", "when", "where", "how" and "why" of a business event. For example, a sales transaction may be defined by a number of components:

- Customer: *who* made the purchase
- Product: *what* was sold
- Location: *where* it was sold
- Period: *when* it was sold

An important component of any transaction is time—historical analysis is an important part of any data warehouse. Component entities form the basis for constructing *dimension tables* in star schemas.

Classification Entities

Classification entities are entities which are related to component entities by a chain of one-to-many relationships—that is, they are functionally dependent on a component entity (directly or transitively). Classification entities represent hierarchies embedded in the data model, which may be collapsed into component entities to form *dimension tables* in a star schema.

Figure 6 shows the classification of the entities in the example data model. In the diagram,

- Black entities represent Transaction entities
- Grey entities indicate Component entities
- White entities indicate Classification entities

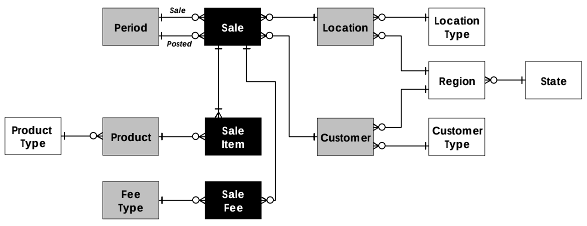

Figure 6. Entity Classifications

Resolving Ambiguities

In some cases, entities may fit into multiple categories. We therefore define a *precedence hierarchy* for resolving such ambiguities:

1. Transaction entity (highest precedence)
2. Classification entity
3. Component entity (lowest precedence)

For example, if an entity can be classified as either a classification entity or a component entity, it should be classified as a classification entity. In practice, some entities will not fit into any of these categories. Such entities do not fit the hierarchical structure of a dimensional model, and cannot be included in star schemas. This is where real world data sometimes does not fit the star schema "mould".

Step 2. Identify Hierarchies

Hierarchies are an extremely important concept in dimensional modelling, and form the primary basis for deriving dimensional models from Entity Relationship models. As discussed previously, most dimension tables in star schemas contain embedded hierarchies. A *hierarchy* in an Entity Relationship model is any sequence of entities joined together by one-to-many relationships, all aligned in the same direction. Figure 7 shows a hierarchy extracted from the example data model, with State at the top and Sale Item at the bottom.

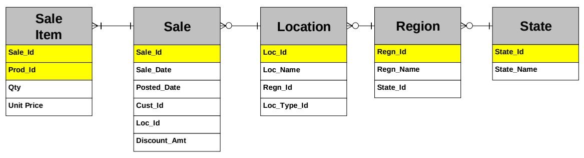

Figure 7 Example of Hierarchy

In hierarchical terminology:

- State is the "parent" of Region
- Region is the "child" of State
- Sale Item, Sale, Location and Region are all "descendants" of State
- Sale, Location, Region and State are all "ancestors" of Sale Item

Maximal Hierarchy

A hierarchy is called *maximal* if it cannot be extended upwards or downwards by including another entity. In all, there are 14 maximal hierarchies in the example data model:

- Customer Type-Customer-Sale-Sale Fee
- Customer Type-Customer-Sale-Sale Item
- Fee Type-Sale Fee
- Location Type-Location-Sale-Sale Fee
- Location Type-Location-Sale-Sale Item
- Period (posted)-Sale-Sale Fee
- Period (posted)-Sale-Sale Item
- Period (sale)-Sale-Sale Fee
- Period (sale)-Sale-Sale Item
- Product Type-Product-Sale Item
- State-Region-Customer-Sale-Sale Fee
- State-Region-Customer-Sale-Sale Item
- State-Region-Location-Sale-Sale Fee
- State-Region-Location-Sale-Sale Item

An entity is called *minimal* if it is at the bottom of a maximal hierarchy and *maximal* if it is at the top of one. Minimal entities can be easily identified as they are entities with no *one-to-many* relationships (or "leaf" entities in hierarchical terminology), while maximal entities are entities with no *many to one* relationships (or "root" entities). In the example data model there are

- Two minimal entities: Sale Item and Sale Fee
- Six maximal entities: Period, Customer_Type, State, Location Type, Product Type and Fee Type.

Step 3. Produce Dimensional Models

Operators For Producing Dimensional Models

We use two operators to produce dimensional models from Entity Relationship models.

Operator 1: Collapse Hierarchy

Higher level entities can be "collapsed" into lower level entities within hierarchies. Figure 8 shows the State entity being collapsed into the Region entity. The Region entity contains its original attributes plus the attributes of the collapsed table. This introduces redundancy in the form of a *transitive dependency*, which is a violation to third normal form (Codd, 1970). Collapsing a hierarchy is therefore a form of *denormalisation* (Tonkin, 1991).

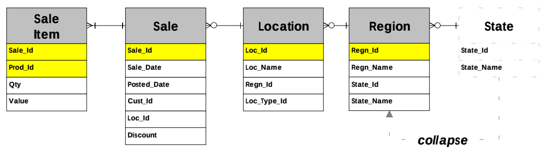

Figure 8. State Entity "collapsed" into Region

Figure 9 shows Region being collapsed into Location. We can continue doing this until we reach the bottom of the hierarchy, and end up with a single table (Sale Item).

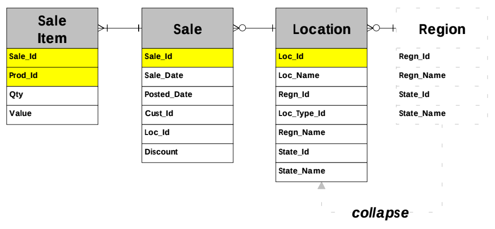

Figure 9. Region Entity "collapsed" into Location

Operator 2: Aggregation

The aggregation operator can be applied to a transaction entity to create a new entity containing summarised data. A subset of attributes is chosen from the source entity to aggregate (the *aggregation attributes*) and another subset of attributes chosen to aggregate by (the *grouping attributes*). Aggregation attributes must be numerical quantities.

For example, we could apply the aggregation operator to the Sale Item entity to create a new entity called Product Summary as in Figure 10. This aggregated entity shows for each product the total sales amount (quantity*price), the average quantity per order and average price per item on a daily basis. The aggregation attributes are quantity and price, while the grouping attributes are Product ID and Date. The key of this entity is the combination of the attributes used to aggregate by (grouping attributes). Note that aggregation *loses information*: we cannot reconstruct the details of individual sale items from the product summary table.

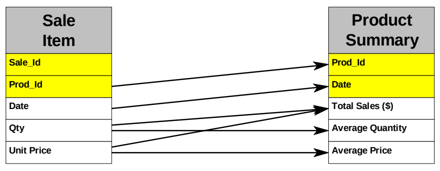

Figure 10. Aggregation Operator

Dimensional Design Options

There is a wide range of options for producing dimensional models from an Entity Relationship model. These include:

- Flat schema
- Terraced schema
- Star schema
- Snowflake schema
- Star cluster schema

Each of these options represent different trade-offs between complexity and redundancy. Here we discuss how the operators previously defined may be used to produce different dimensional models.

Option 1: Flat Schema

A *flat schema* is the simplest schema possible without losing information. This is formed by collapsing all entities in the data model down into the *minimal* entities. This minimises the number of tables in the database and therefore the possibility that joins will be needed in user queries. In a flat schema we end up with one table for each minimal entity in the original data model. Figure 11 shows the flat schema which results from the example data model.

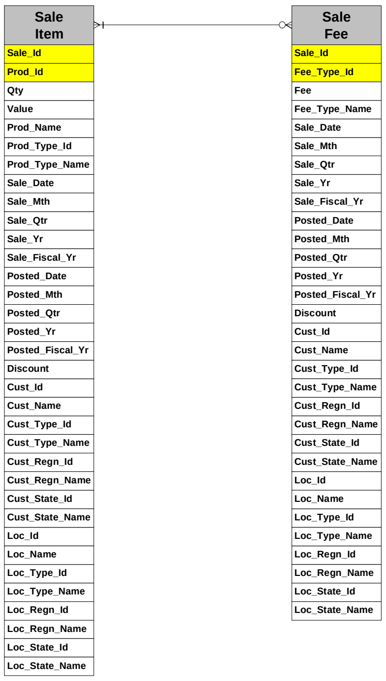

Figure 11. Flat Schema

Such a schema is similar to the "flat files" used by analysts using statistical packages such as SAS and SPSS. Note that this structure does not lose any information from the original data model. It contains redundancy, in the form of transitive and partial dependencies, but does not involve any aggregation. One problem with a flat schema is that it may lead to aggregation errors when there are hierarchical relationships between transaction entities. When we collapse numerical amounts from higher level transaction entities into another they will be repeated. In the example data model, if a Sale consists of three Sale Items, the discount amount will be stored in three different rows in the Sale Item table. Adding the discount amounts together then results in double-counting (or in this case, triple counting). Another problem with flat schemas is that they tend to result in tables with large numbers of attributes, which may be unwieldy. While the number of tables (*system complexity*) is minimised, the complexity of each table (*element complexity*) is greatly increased.

Option 2: Terraced Schema

A terraced schema is formed by collapsing entities down maximal hierarchies, but stopping when they reach a transaction entity. This results in a single table for each transaction entity in the data model. Figure 12 shows the terraced schema that results from the example data model. This schema is less likely to cause problems for an inexperienced user, because the separation between levels of transaction entities is explicitly shown.

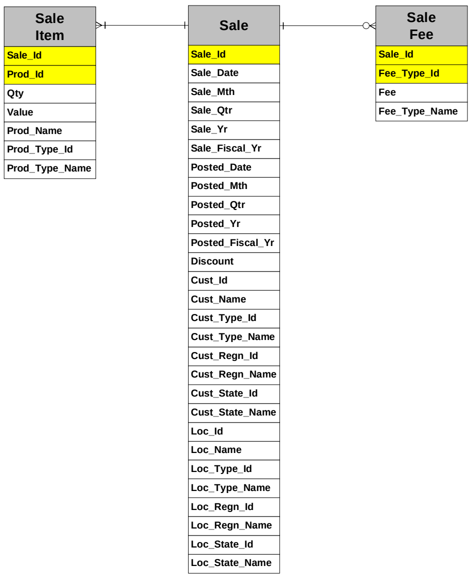

Figure 12. Terraced Schema

Option 3: Star Schema

A star schema can be easily derived from an Entity Relationship model. Each star schema is formed in the following way:

- A *fact table* is formed for each transaction entity. The key of the table is the combination of the keys of its associated component entities.
- A *dimension table* is formed for each component entity, by collapsing hierarchically related classification entities into it.
- Where hierarchical relationships exist between transaction entities, the child entity inherits all dimensions (and key attributes) from the parent entity. This provides the ability to "drill down" between transaction levels.

- Numerical attributes within transaction entities should be aggregated by key attributes (dimensions). The aggregation attributes and functions used depend on the application.

Figure 13 shows the star schema that results from the Sale transaction entity. This star schema has four dimensions, each of which contains embedded hierarchies. The aggregated fact is Discount amount.

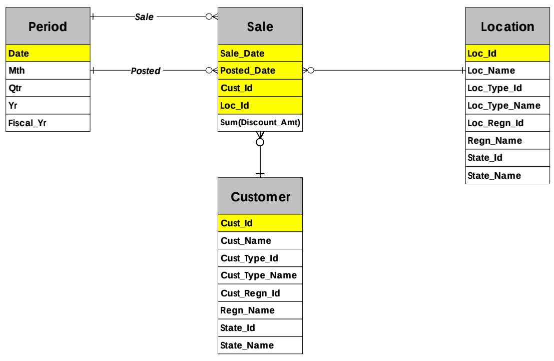

Figure 13. Sale Star Schema

Figure 14 shows the star schema which results from the Sale Item transaction entity. This star schema has five dimensions. This includes four dimensions from its "parent" transaction entity (Sale) and one of its own (Product). The aggregated facts are quantity and item cost (quantity * price).

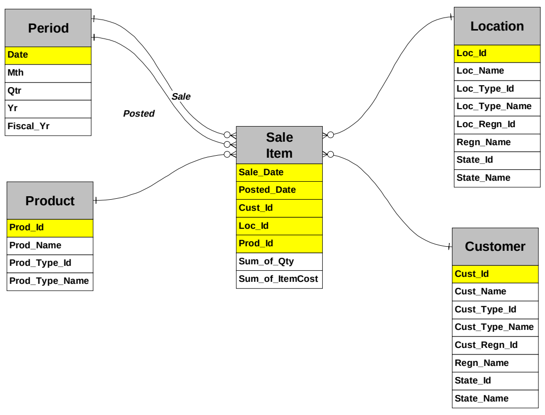

Figure 14. Sale Item Star Schema

A separate star schema is produced for each transaction table in the original data model.

Constellation Schema

Instead of a number of discrete star schemas, the example data model can be transformed into a *constellation schema*. A constellation schema consists of a set of star schemas with hierarchically linked fact tables. The links between the various fact tables provide the ability to "drill down" between levels of detail (e.g. from Sale to Sale Item). The constellation schema which results from the example data model is shown in Figure 15—links between fact tables are shown in bold.

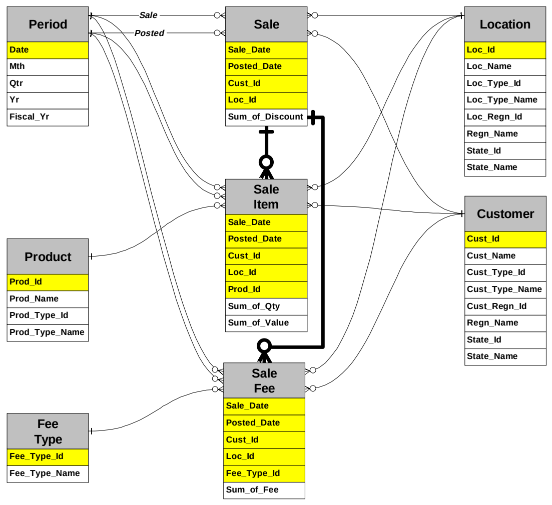

Figure 15. Sales Constellation Schema

Galaxy Schema

More generally, a set of star schemas or constellations can be combined together to form a *galaxy*. A galaxy is of a collection of star schemas with shared dimensions. Unlike a constellation schema, the fact tables in a galaxy do not need to be directly related.

Option 4: Snowflake Schema

In a star schema, hierarchies in the original data model are collapsed or denormalised to form dimension tables. Each dimension table may contain multiple independent hierarchies. A *snowflake schema* is a star schema with all hierarchies explicitly shown. A snowflake schema can be formed from a star schema by expanding out (normalising) the hierarchies in each dimension. Alternatively, a snowflake schema can be produced directly from an Entity Relationship model by the following procedure:

- A *fact table* is formed for each transaction entity. The key of the table is the combination of the keys of the associated component entities.
- Each component entity becomes a *dimension table*.
- Where hierarchical relationships exist between transaction entities, the child entity inherits all relationships to component entities (and key attributes) from the parent entity.
- Numerical attributes within transaction entities should be aggregated by the key attributes. The attributes and functions used depend on the application.

Figure 16 shows the snowflake schema which results from the Sale transaction entity.

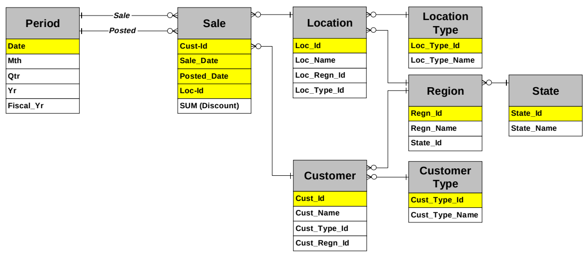

Figure 16. Sale Snowflake Schema

Option 5: Star Cluster Schema

Kimball (1996) argues that “snowflaking” is undesirable, because it adds complexity to the schema and requires extra joins. Clearly, expanding all hierarchies defeats the purpose of producing simple, user friendly database designs—in the example above, it more than doubles the number of tables in the schema. In this paper, we argue that neither a “pure” star schema (fully collapsed hierarchies) nor a “pure” snowflake schema (fully expanded hierarchies) results in the best solution. As in many design problems, the optimal solution is a balance between two extremes.

The problem with fully collapsing hierarchies occurs when hierarchies overlap, leading to redundancy *between* dimensions when they are collapsed. This can result in confusion for users, increased complexity in extract processes and inconsistent results from queries if hierarchies become inconsistent. For these reasons, we require that dimensions should be *orthogonal*.

Overlapping dimensions can be identified via “forks” in hierarchies. A fork occurs when an entity acts as a “parent” in two different dimensional hierarchies. This results in the entity and all of its ancestors being collapsed into two separate dimension tables. Fork entities can be identified as classification entities with multiple one-to-many relationships. The exception to this rule occurs when the hierarchy converges again lower down—Dampney (1996) calls this a *commuting loop*.

In the example data model, a fork occurs at the Region entity. Region is a parent of Location and Customer, which are both components of the Sale transaction. In the star schema representation, State and Region would be included in both the Location and Customer dimensions when the hierarchies are collapsed. This results in overlap between the dimensions.

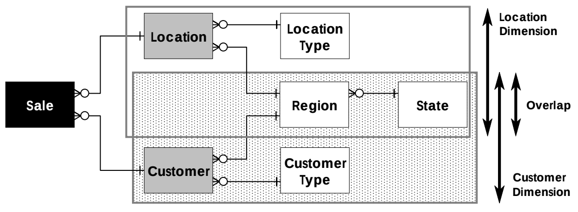

Figure 17. Intersecting Hierarchies in Example Data Model

We define a *star cluster schema* as one which has the minimal number of tables while avoiding overlap between dimensions. It is a star schema which is selectively “snowflaked” to separate out hierarchical segments or *subdimensions* which are shared between different dimensions. Subdimensions effectively represent the “highest common factor” between dimensions.

A star cluster schema may be produced from an Entity Relationship model using the following procedure. Each star cluster is formed by:

- A *fact table* is formed for each transaction entity. The key of the table is the combination of the keys of the associated component entities.
- Classification entities should be collapsed down their hierarchies until they reach either a fork entity or a component entity. If a fork is reached, a *subdimension table* should be formed. The subdimension table will consist of the fork entity plus all its ancestors. Collapsing should begin again after the fork entity. When a component entity is reached, a *dimension table* should be formed.
- Where hierarchical relationships exist between transaction entities, the child entity should inherit all dimensions (and key attributes) from the parent entity.
- Numerical attributes within transaction entities should be aggregated by the key attributes (dimensions). The attributes and functions used depend on the application.

Figure 18 shows the star cluster schema that results from the model fragment of Figure 17.

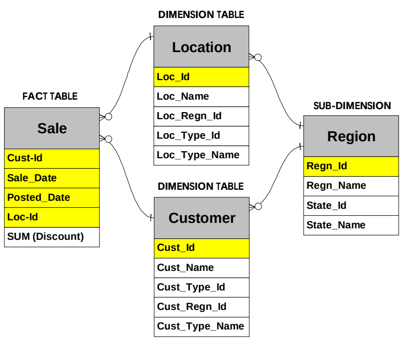

Figure 18. Star Cluster Schema

Figure 19 shows how entities in the original data model were clustered to form the star cluster schema. The overlap between hierarchies has now been removed.

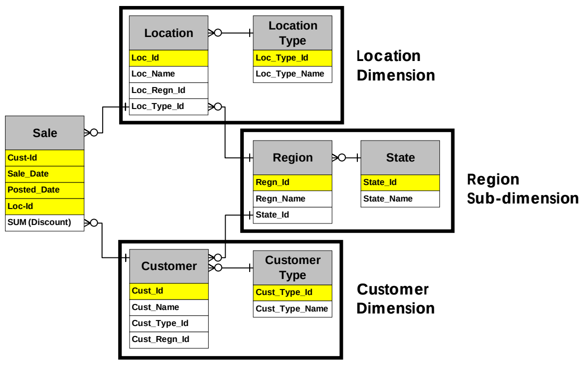

Figure 19. Revised Clustering

If required, *views* may be used to reconstruct a star schema from a star cluster schema. This gives the best of both worlds: the simplicity of a star schema while preserving consistency between dimensions. As with star schemas, star clusters may be combined together to form constellations or galaxies.

Step 4. Evaluation and Refinement

In practice, dimensional modelling is an iterative process. The clustering procedure described in Step 3 is useful for producing a first cut design, but this will need to be refined to produce the final data mart design. Most of these modifications have to do with further simplifying the model and dealing with non-hierarchical patterns in the data.

Combining Fact Tables

Fact tables with the same primary keys (i.e. the same dimensions) should be combined. This reduces the number of star schemas and facilitates comparison between related facts (e.g. budget and actual figures).

Combining Dimension Tables

Creating dimension tables for each component entity often results in a large number of dimension tables. To simplify the data mart structure, related dimensions should be consolidated together into a single dimension table.

Many to Many Relationships

Most of the complexities which arise in converting a traditional Entity Relationship model to a dimensional model result from many-to-many relationships or intersection entities. Many-to-many relationships cause problems in dimensional modelling because they represent a “break” in the hierarchical chain, and cannot be collapsed. There are a number of options for dealing with many-to-many relationships:

(a) Ignore the intersection entity (eliminate it from the data mart)

(b) Convert the many-to-many relationship to a one-to-many relationship, by defining a “primary” relationship

(c) Include it as a many-to-many relationship in the data mart—such entities may be useful to expert analysts but will not be amenable to analysis using an OLAP tool.

For example, in the model below, each client may be involved in a number of industries. The intersection entity Client Industry breaks the hierarchical chain and cannot be collapsed into Client.

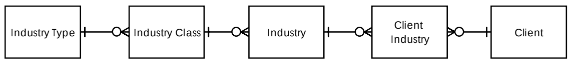

Figure 20. Multiple Classification

The options are (a) to exclude the industry hierarchy, (b) convert it to a one-to-many relationship or (c) include it as a many-to-many relationship.

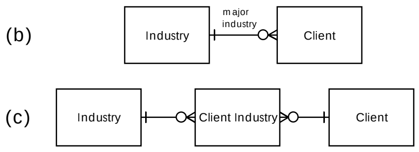

Figure 21. Design Options

Handling Subtypes

Supertype/subtype relationships can be converted to a hierarchical structure by removing the subtypes and creating a classification entity to distinguish between subtypes. This can then be converted to a dimensional model in a straightforward manner.

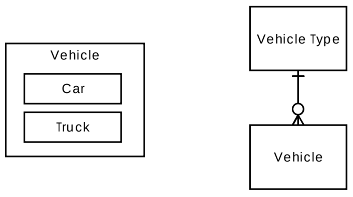

Figure 22. Conversion of Subtypes to Hierarchical Form

5. CONCLUSION

Summary of the Method

We have described a method for developing data warehouse and data mart designs from an enterprise data model. The method has now been applied in a wide range of industries, including manufacturing, health, insurance and banking. The method has evolved considerably as a result of experiences in practice. The steps of the method are:

1. Develop Enterprise Data Model (if one doesn't exist already)
2. Design Central Data Warehouse: this will be closely based on the enterprise data model, but will be a subset of the model which is relevant for decision support purposes. A staged approach is recommended for implementing the central data warehouse, starting with the most important subject areas.
3. Classify Entities: classify entities in the central data warehouse model as either transaction, component or classification entities.
4. Identify Hierarchies: identify the hierarchies which exist in the data model
5. Design Data Marts: develop star cluster schemas for each transaction entity in the central data warehouse model. Each star cluster will consist of a fact table and a number of dimension and subdimension tables. This minimises the number of tables while avoiding overlap between dimensions. The separate star clusters may be combined together to form constellations or galaxies.

Design Options

We have identified a range of options for developing data marts to support end user queries from an enterprise data model. These options represent different trade-offs between the number of tables (complexity) and redundancy of data (Figure 23).

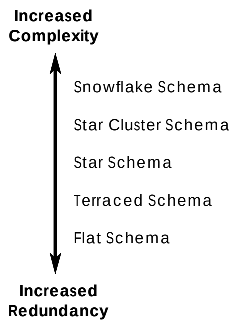

Figure 23. Design Trade-offs

Implications for Data Warehouse Design Practice

The advantages of this approach are:

- It provides a more structured approach to developing dimensional models than working from first principles
- It ensures that the data marts and the central data warehouse reflect the underlying relationships in the data
- Developing data warehouse and data mart designs based on a common enterprise data model simplifies extract and load processes
- An existing enterprise data model provides a useful basis for identifying information requirements in a bottom up manner—based on what data exists in the enterprise. This can be usefully combined with Kimball's (1996) top-down analysis approach.
- An enterprise data model provides a more stable basis for design than user query requirements, which are unpredictable and subject to frequent change
- It ensures that the central data warehouse is flexible enough to support the widest possible range of analysis requirements, by storing data at the level of individual transactions. Aggregation of data at this level reduces the granularity of data in the data warehouse, which limits the types of analyses which are possible.
- It maximises the integrity of data stored in the central data warehouse

While the approach is not entirely mechanical, it provides much more guidance to designers of data warehouses and data marts than previous approaches. Careful analysis is still required to identify the entities in the enterprise data model which are relevant for decision making and classifying them. However once this has been done, the development of a dimensional model can take place in a relatively straightforward manner. Using an Entity Relationship model of the data provides a much better starting point for developing dimensional models than starting from scratch, and can help avoid many of the pitfalls faced by inexperienced designers.

Federated Data Warehouse Architecture

Use of this approach also supports the development of independent but consistent data warehouses based on a common enterprise data model (*federated architecture*). The enterprise data model can be used to ensure the consistency of data definitions between the different warehouses, enabling data to be compared and combined between them. It also allows use of common extract processes. This may be a more feasible option than a single centralised data warehouse in large and diverse organisations.

REFERENCES

1. BANKER, R.D., AND KAUFFMAN, R.J. (1991): Reuse and Productivity in Integrated Computer Aided Software Engineering: An Empirical Study, MIS Quarterly,
2. BATINI, C., CERI, S. AND NAVATHE, S.B. (1994) *Conceptual Database Design: An Entity Relationship Approach*, Benjamin Cummings, Redwood City, California.
3. CHEN, P.P. (1976): The Entity Relationship Model: Towards an Integrated View of Data, *ACM Transactions on Database Systems*, 1(1), March: 9-36.
4. CHEN, P.P. (1983): A Preliminary Framework for Entity-Relationship Models, in Chen, P.P. (ed.) The Entity-Relationship Approach to Information Modelling and Analysis, North-Holland.
5. CODD, E.F. (1970) A Relational Model of Data for Large Shared Data Banks, *Communications of the ACM*, 13 (6), June: 377-387.
6. DAMPNEY, C.N.G. (1996): Corporate Genera or Corporate Trivia—Can We Really Base Applications on Corporate Data Models? *Proceedings of the First Australian DAMA Conference,* Melbourne, Australia, December 1-2.
7. DEVLIN, B. (1997): *Data Warehouse: From Architecture to Implementation*, Addison-Wesley, Reading, Mass.
8. GARDNER, S.R. (1998): Building the Data Warehouse, *Communications of the ACM*, September.
9. GILES, J. (1999): Data Warehousing: Is It Just First Aid?, in D.L. Moody (ed.) *From Data to Information to Knowledge: Managing the Unmanageable, Proceedings of the Fourth Australian DAMA Conference,* Melbourne, Australia, October 11-13.
10. GLASSEY, K. (1998): Seducing the End User, *Communications of the ACM,* September.
11. GOODHUE, D.L., KIRSCH, L.J., AND WYBO, M.D. (1992): The Impact of Data Integration on the Costs and Benefits of Information Systems, *MIS Quarterly*, 16(3), September: 293-311.
12. GRAHAM, S., COBURN, D. and OLESON, C. (1996): *The Foundations of Wisdom: A Study of the Financial Impact of Data Warehousing,* International Data Corporation (IDC) Ltd. Canada.
13. HALPIN, T. (1995): *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*, Prentice Hall, Sydney.
14. HITCHMAN, S. (1995) Practitioner Perceptions On The Use Of Some Semantic Concepts In The Entity Relationship Model, *European Journal of Information Systems,* 4, 31-40.
15. INMON, W. (1996): *Building the Data Warehouse (2nd Edition)*, New York: J. Wiley & Sons.
16. INTERNATIONAL STANDARDS ORGANISATION (ISO) (1987), *Information Processing Systems - Concepts and Terminology for the Conceptual Schema and the Information Base*, ISO Technical Report 9007.
17. KIMBALL, R. (1996): *The Data Warehouse Toolkit,* New York: J. Wiley & Sons.
18. KIMBALL, R. (1997): A Dimensional Manifesto, *DBMS Online*, August, 1997.
19. LEWIS, D. (1999): "Use and Abuse of a Meta-Data Registry", *Fourth Australian Data Management Conference,* Melbourne, Australia, October 11-13.
20. LOVE, B. (1994): *Enterprise Information Technologies*, Van Nostrum Reinhold, New York.
21. MARTIN, J. (1983): *Strategic Data Planning Methodologies*, Prentice-Hall.
22. MOODY, D.L. AND SIMSION, G.C. (1995) Justifying Investment in Information Resource Management, *Australian Journal of Information Systems,* 3(1), September: 25-37.
23. POKORNY, J. (1998): Data Warehouses: A Modelling Perspective, *Proceedings of the 7th International Conference on Information Systems Development,* Bled, Slovenia, Plenum Press.
24. SAGER, M.J. (1988): Data-centred Enterprise Modelling Methodologies - A Study Of Practice And Potential. *Australian Computer Journal,* August.
25. SEN, A. and JACOB, V.S. (1998): Industrial Strength Data Warehousing, *Communications of the ACM,* September.
26. SHANKS, G.G. (1996): "Enterprise Data Architectures: A Study of Practice", *First Australian Data Management Conference,* Melbourne, Australia, December 2-3.
27. SIMSION, G.C. (1994): *Data Modelling Essentials*, Van Nostrand Reinhold, New York.