An Analysis of Many-to-Many Relationships Between Fact and Dimension Tables in Dimensional Modeling

Il -Yeol Song
College of Information Science and Technology
Drexel University
Philadelphia, PA 19104
songiy@drexel.edu

William Rowen
College of Information Science and Technology
Drexel University
Philadelphia, PA 19104
msis@drexel.edu

Carl Medsker
Arynth, Inc.
Cinnaminson, NJ 08077
cmedsker@arynth.com

Edward Ewen, M.D
Christiana Care Health System
Wilmington, DE 19899
eewen@christianacare.org

Abstract

Star schema, which maintains one-to-many relationships between dimensions and a fact table, is widely accepted as the most viable data representation for dimensional analysis. Real-world DW schema, however, frequently includes many-to-many relationships between a dimension and a fact table. Having those relationships in a dimensional model causes several difficult issues, such as losing the simplicity of the star schema structure, increasing complexity in forming queries, and degrading query performance by adding more joins. Therefore, it is desirable to represent the many-to-many relationships with correct semantics while still keeping the structure of the star schema.

In this paper, we analyze many-to-many relationships between a dimension table and a fact table in dimensional modeling. We illustrate six different approaches and show the advantages and disadvantages of each. We propose two ad-hoc methods that maintain a star schema structure by denormalizing the dimensions to avoid many-to-many relationships. This method allows quick query processing by using a concatenated attribute with minimal overhead. Other issues addressed are data redundancy, weighting factors, storage requirements, and performance concerns.

Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2001)
Interlaken, Switzerland, June 4, 2001
(D. Theodoratos, J. Hammer, M. Jeusfeld, M. Staudt, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-39/

1. Introduction

The data warehouse (DW) is an integrated repository of data, generated and used by an entire organization. The data warehouse employs a suite of tools that transforms raw data into meaningful business information. This information depicts a view of a distinct business process to identify trends and patterns and serves as a foundation for decision-making.

The dimensional model is a logical representation of a business process whose significant features are user understandability, query performance, and resilience to change. Dimensional modeling is widely accepted as the viable technique for delivering data to end users in a data warehouse [KRRT98, AM97, AV98, AS97, DSHB98, MC98]. The main components of a dimensional model are fact tables and dimension tables. A fact table contains measurements of the business or records events. A dimension table contains attributes used to constrain, group, or browse the fact data. There are two primary advantages of using a dimensional model in data warehouse environments. First, a dimensional model provides a multidimensional analysis space in relational database environments; we are analyzing factual data using dimensions. Second, a typical denormalized dimensional model has a simple schema structure, which simplifies end-user query processing and improves performance.

The dimension tables contain a large number of attributes, reflecting the details of the business processes. Browsing is a user activity that explores the relationships between attributes in a dimension table. The attributes will serve as row headers and constraints for these views. It is common to have more than one hundred attributes in a real world application. Dimension tables are considered wide for this reason. Denormalization of dimension tables is an acceptable practice in data warehousing. A dimensional model with highly normalized dimension structure is called a snowflake schema [KRRT98]. Any attempts to normalize a dimension table into a series of tables could reduce the browsing capabilities of the user, resulting in more complex queries and increased retrieval time. Our experiences with real-world data warehouse development shows that browsing and group-by queries are the two salient issues that drive the design of data warehouses.

The fact table is where the numerical measurements of the business processes are stored. These measurements or events are related to each dimension table by foreign keys. The fact table contains thousands, or even millions of rows of records. A typical query will compress or extract a large number of records into a handful of rows using aggregation. Therefore, the most useful facts are numeric, continuously valued, and additive; Kimball calls this premise the holy grail of dimensional database design [Kimb96].

The grain of the fact table is a very important characteristic. The grain is the level of detail at which measurements or events are stored. It determines the dimensionality of the data warehouse and dramatically impacts the size and conversely the performance.

The goal in designing the data warehouse model is to keep it simple to understand, simple to load with operational data, and as fast as possible to query [Kimb96, Kimb97, KRRT98, AM97, AV98]. We would like to have neophyte and experienced business analysts creating reports, so the logical model needs to be easy to comprehend. Most business analysts frequently have a difficult time finding data in both highly normalized designs and abstract object designs. The flatter the dimensional model, the better for end-users. The more complex the model, the more complex will be the extract/transform/load (ETL) routines to create and run.

Finally, queries against the database will run faster if a minimal number of one-to-many relationships and joins are present. To provide users with the views they need for analysis, the one-to-many relationships between facts and dimensions should be flattened into a series of views or derived tables. For instance, the statistician may want to create a regression model against diagnoses with the grain of the analysis being a single visit to the hospital. Therefore, each row must completely define a visit with columns for specific diagnoses or columns that represent groups of diagnoses. To meet the fundamental goal of empowering end users to perform their own queries and analyses, the design must balance elegance in conceptual design with understandability, usability, and performance.

Design principles dictate that one should identify any dimensional attribute that has a single value for an individual fact table record. The designer can build outward from the grain of the fact table and relate as many dimensions as the business process demands. Therefore, dimension tables are typically joined to the fact table with a one-to-many relationship. When all the dimensions are related by one-to-many relationships with the fact table, the schema is called a star schema. However, real-world DW schema frequently includes many-to-many relationships between a dimension and a fact table [KRRT98, AS97]. Having those relationships in a dimensional model causes several difficult issues, such as losing the star schema structure, increasing complexity in forming queries, and degrading query performance by adding more joins. Therefore, it is desirable that we handle the many-to-many relationships while still keeping the structure of the star schema.

In this paper, we analyze many-to-many relationships between a dimension table and a fact table in dimensional modeling. Even though there are some previous studies on how to represent a data warehouse conceptual schema [GR98, SBHD98, TBC99] or how to derive/design a data warehouse schema [AM97, KS97, TS98, LAW98, PJ99, MK00, HLB00], the specific method of handling many-to-many relationships is rarely addressed. Two sources we found are books by Kimball et al. [KRRT98] and Giovinazzo [Giov00]. Not being satisfied by those approaches for our real-world project, we have performed a thorough study on how to handle many-to-many relationships. In this paper, we illustrate six different approaches and show the advantages and disadvantages of each. We propose two ad-hoc methods that maintain a star schema structure by denormalizing the dimension to avoid many-to-many relationships. These methods allow us to quickly process queries. Other issues that will be addressed include data redundancy, weighting factors, storage requirements, and performance concerns.

The remainder of this paper is organized as follows: Section 2 presents a motivation example. Section 3 presents six approaches and discusses the advantages and disadvantages. Sections 4 presents a summary table and Section 5 concludes our paper.

2. Motivational Example

In the healthcare billing process, there are usually multiple diagnoses for each patient visit. A design problem arises in modeling a diagnosis dimension that has a many-to-many relationship with a fact table as shown in Figure 1. We will explore specific data warehousing structures to analyze this predicament. We will use, as an illustrated example, the patient-billing situation throughout this paper to compare and contrast the different solutions.

In Figure 1, the relationship between the diagnosis dimension and the billable patient encounter fact table is illustrated as a many-to-many. This considers the situation where a patient has more than one diagnosis for each billable encounter.

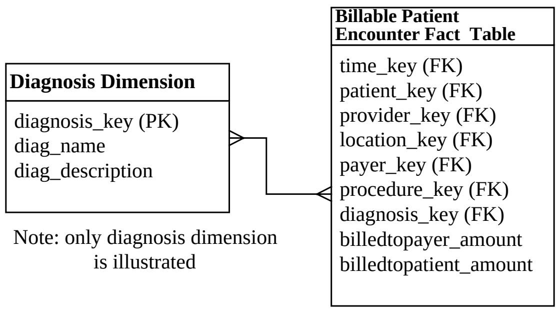


Figure 1: Healthcare Billable Encounter Schema [KRRT98]

There are inherent problems with many-to-many relationships between a fact table and a dimension. Querying for records to find a particular combination of diagnoses requires multiple correlated subqueries. Consider the query for retrieving 'billed to payer amount' and 'patient key' for patients who have a combination of diagnoses named 'heart' and 'cancer' (for the remainder of the discussion we will refer to the diagnosis dimension as DD and the billable patient encounter fact table as BPE).

```
SELECT      patient_key, billedtopayer_amount
FROM        BPE
WHERE       patient_key IN
     (SELECT        patient_key
     FROM           DD, BPE
     WHERE          diag_name = 'cancer'
            AND     DD.diagnosis_key =
                    BPE.diagnosis_key
     INTERSECT
             SELECT         patient_key
             FROM           DD, BPE
            WHERE           diag_name = 'heart'
                    AND     DD.diagnosis_key =
                            BPE.diagnosis_key);
```

The subquery will select all patient numbers that have both heart and cancer diagnosis names. Queries for finding patients with N different diagnoses will need N-level subqueries. Therefore, report generation is very complex and slow; you must search a large number of records with multiple correlated subqueries, increasing both the processing time and the number of joins.

When one requests additive measurements through the relationship, the user may receive incorrect results. It is necessary to implement a weighting factor to give each separate diagnosis its appropriate contribution to the total bill [AV98].

An additional problem with this design is frequently a user may not want all of the diagnoses. When an end-user retrieves fewer than all the diagnoses then the weighting factor will not directly add up (see Section 3.1.1 for the issues of weighting factors in a many-to-many relationship). You must guarantee by some other means that the correct weight is applied for any subset of diagnoses. Users must be protected from retrieving a subset of data that aggregates incorrectly, which will occur if no precautions are taken.

3. Methods for Handling Many-to-Many Relationships

3.0. Assumptions

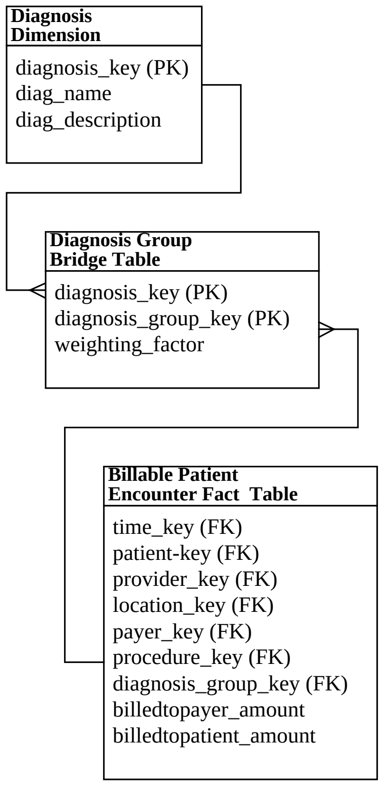
Based on our experience of building a real-world data warehouse in a patient-billing domain, we have assumed the following data for our analysis. The fact table contains patient billing information and each bill is assigned one primary diagnosis and one to many secondary diagnoses. Although it is theoretically possible to have hundreds of diagnoses, the maximum in practice is twenty or less. Frequency distributions on an existing operational database show that most bills have fewer than five secondary diagnoses, with very few bills having more than 10. These are entered into the operational system in no particular order. There is no qualitative difference between secondary 1 and secondary 20. In addition, government and insurance claim forms typically provide space for a maximum four or eight secondary diagnoses, so the practical limit is fixed. All have equal potential importance, depending on the context of use or the type of information that is compiled.

Making a few assumptions can approximate the estimated size of the dimension and fact tables.

- Fact table contains 1,000,000 records
- There are maximum 20 billable diagnoses for each encounter.
- There are maximum 500 billable diagnoses.
- There are on the average five separate diagnoses for each encounter
- All numerical field widths are an average four bytes, names are eight bytes, and descriptions are 15 bytes.

3.1. Method A: The Bridge Table

Figure 2 depicts Kimball's use of the bridge table to connect multiple diagnoses to a fact table [KRRT98]. The bridge table is an intersection table between a diagnosis dimension table and the fact table. This table is similar to an intersection table that is created for a many-to-many relationship between two entities. However, what distinguishes this bridge table in data warehouse modeling from an intersection table in data modeling is the use of weighting factors and a diagnosis group key. A diagnosis group key is assigned to clusters of diagnosis codes and the combinations are inserted into the bridge table.



Note: only diagnosis dimension is illustrated

Figure 2: Solving Multiple Diagnoses with a Bridge Table [KRRT98]

3.1.1. Weighting Factor

Observe the weighting factor attribute in Figure 2. The weighting factor is a percentage that identifies the contribution of the diagnosis to the specific encounter. Within a diagnosis group, the sum of all the weighting factors must equal one. The weighting factor is multiplied by fact values, through the joining of the two tables with the diagnosis group key. In this manner, the involvement of each diagnosis in the diagnosis group is correctly calculated. Conversely, the user can request an impact analysis, ignoring the weighting factors [KRRT98]. Such impact reports will erroneously aggregate the amounts. It will produce a summation based on the impact each diagnosis has in relation to total amounts associated with that diagnosis. Consider the following example, which shows only necessary attributes:

Table 1:Diagnosis Dimension

DD (Diagnosis Dimension)	
diagnosis_key	**diag_name**
DK1	Cancer
DK2	Heart
DK3	Lung

Query: Given Tables 1, 2, and 3, find the 'billed to payer amount' contributed by each diagnosis.

Situation 1 Impact Report:

```
SELECT      diag_name, SUM (billedtopayer_amount)
FROM        DD,DGB,BPE
WHERE       DD.diagnosis_key = DGB.diagnosis_key
AND         DGB.diagnosis_group_key =
            BPE.diagnosis_group_key
GROUP BY    diag_name;
```

Results:

diag_name	billed_to_payer_amount
Cancer	$ 3,000
Heart	$ 3,000
Lung	$ 2,000

The results clearly indicate the inherent problem in a many-to-many situation where the aggregation is counted for the total amount for each occurrence of a diagnosis in the records (total amount billed is $8,000). Cancer occurred in diagnosis group one and two, thus it was counted twice ($1,000 from diagnosis group one and $ 2,000 from diagnosis group two returning an impact total of $ 3,000).

Situation 2 Weighting Factor Report:

```
SELECT          diag_name,
   SUM (billedtopayer_amount *weighting_factor)
FROM            DD, DGB, BPE
WHERE           DD.diagnosis_key = DGB.diagnosis_key
 AND            DGB.diagnosis_group_key =
                BPE.diagnosis_group_key
GROUP BY        diag_name;
```

Results:

diag_name	billed_to_payer_amount
Cancer	$ 2,000
Heart	$ 800
Lung	$ 200

Table 2: Diagnosis Group Bridge Table

DGB (Diagnosis Group Bridge)		
Diagnosis key	**Diagnosis group_key**	**Weighting factor**
DK1	DG1	0.8
DK2	DG1	0.2
DK1	DG2	0.6
DK2	DG2	0.3
DK3	DG2	0.1

Table 3: Billable Patient Encounter Schema

BPE (Billable Patient Encounter)		
Patient key	**Diagnosis group_key**	**billedtopayer amount**
P1	DG1	$ 1,000
P2	DG2	$ 2,000

The weighting factors produce a correct totaled report ($3,000). During the summation, the weighting factor for each diagnosis key will be related to each bill through the foreign key (diagnosis group key) found in the billable patient encounter table.

The weighting factor is necessary when using a bridge implementation to produce correct reports [KRRT98, AV98]. However, it is not always possible to rationalize the weighting factors for each diagnosis. In that case, it would be possible to count the total diagnoses and produce an average cost through additional design measures. One method would be to add an additional attribute to the bridge, call it *number_of_diagnosis*; thus, you could divide your impact total by this value to produce an average cost per diagnosis. This brute force method takes away from the usefulness of your decision support based reports. Thus, it is recommended to use this method only when the correct calculation of the weighting factors is not necessary.

A major benefit of this design is there is no fixed upper limit, other than total possible diagnoses. Although in this study, we have set an upper limit of twenty diagnoses, to meet the user requirements. The bridge method, as you can observe, implements a compound primary key for the bridge table comprised of diagnosis group key and diagnosis key. It is possible to find a group of related diagnoses because the diagnosis group value is repeated for every member row in a set of diagnoses.

There may be other such many-to-many dimensions related to the same fact table, and the load times and query times can be expected to be lengthy. For instance, there are many procedure codes and Diagnosis Related Group (DRG) codes assigned to a single visit or patient bill. A DRG is a classification of a hospital stay in terms of what was wrong with and what was done for a patient. There are approximately 500 DRG codes, which are determined by a program based on diagnoses and procedures coded in a standard International Classification of Disease (ICD-9) format and on patient attributes such as age, sex, and duration of treatment. The DRG frequently determines the amount of reimbursements, regardless of the actually costs incurred. A hospital visit is often coded by multiple systems, such as Systematized Nomenclature of Medicine (SNOMED), Current Procedural Terminology (CPT4), and others, all of which share a many-to-many relationship with the billable patient encounter fact table. Considering the complexity of the healthcare billing system, the design and performance using bridge tables will get quite complex.

The size of the bridge table would increase considerably if one encounter has many related diagnoses. We used an average of five diagnoses per encounter for this example; this parameter produced a bridge table comparable to the size of the fact table, as we will now demonstrate.

3.1.2. Database Sizing for the Bridge Method.

Base fact: 1,000,000 records
Key fields = 7; Fact fields = 2; Total Fields = 9
Fact table size = 1,000,000 records * 9 fields * 4 bytes =
<u>36 MB</u>
Diagnosis dimension: 500 records
Key fields = 1; Name field =1; Description field =1
Record size = 4+8+15 =27 bytes
Dimension table size = 500 records * 27 bytes =
<u>13,500 Bytes</u>

Bridge table: 1,000,000 facts joined to 5 distinct diagnoses in a diagnosis group = 5,000,000 records
Key fields = 2; Weighting factor = 1; Total fields = 3
Bridge table size=5,000,000 records * 3 fields * 4 bytes=
<u>60 MB</u>
Notice, if the average number of diagnosis is increased to ten, our bridge table size will grow to **<u>120 MB</u>**, nearly four times the size of our fact table
<u>Total disk space = 96.1MB</u>

In summary, the bridge method can be considered a logical solution for a many-to-many relationship with less redundancy. There are, however, various disadvantages to this method. Assigning weighting factors could prove to be difficult or cumbersome in a real-world environment; additionally, adding a new diagnosis requires recalculating of the weighting factors. The logical structure would lose the simplicity and understandability of the star schema. More joins increase the overhead and query time. As pointed out the size of the bridge table could increase considerably based on the number of diagnosis assigned to each diagnosis group.

3.2. Method B: Denormalizing the Dimension Table by Positional-Flag Attributes

Figure 3 illustrates denormalizing the diagnosis dimension using the positional-attribute approach. By *positional* we mean that the location of each attribute is fixed. For example, the first attribute is cancer; the second attribute is heart, etc. Thus, the same disease is always indicated in the same column. In this method, each diagnosis becomes a Boolean attribute being set to either 'TRUE' or 'FALSE'. For brevity and clarity, only five attributes have been included in Figure 3.

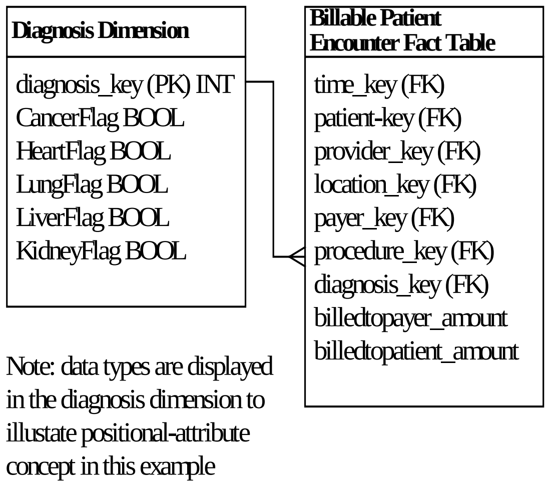


Figure 3: Denormalizing the Dimension Table by Positional Flag attributes

This technique requires a very large diagnosis dimension table. N diagnoses require 2^N records; for this trivial example of five diagnoses, the table size is 32 records. Consider Table 4 that lists all the unique diagnosis patterns.

If we were to extend our model to include 10 diagnoses, the table would be 1024 records in length; 20 diagnoses would require 1,048,576 records; 40 diagnoses would require about one trillion records.

Additional disadvantages of this method include:

- adding a new diagnosis value would require to rebuild the dimension table and the fact table. We need to use Data Definition Language (DDL) to add a column and reload the diagnosis dimension by adding (2^{N+1} - 2^N) rows and updating the diagnosis_key in the fact table;
- there are no approaches to handle a weighting factor.

However, a bitmap index scheme [OG95] can be implemented on each positional attribute, which would improve the query performance in this approach. It is clear that this method would only be applicable when the number of positional-attributes is limited and fixed.

3.2.1. Database Sizing for the Positional-Flag Attribute Method

Number of base fact records: 1,000,000 records
Key fields = 7; Fact fields = 2; Total Fields = 9
Fact table size = 1,000,000 records * 9 fields *4 bytes = **36MB**
Consider the total size of the dimension for 40 diagnoses:
Diagnosis dimension: 2^{40} records ≈ 1126.4 * 10^9 records
Number of key fields = 1; Number of attribute fields = 40;
Assuming 1 bit for each flag
Record size = 4+(1*40)/8= 9Bytes
Dimension table size ≈ 1126.4 * 10^9 records * 9 bytes ≈ **10.1TB**

Total disk space = 10.1TB
(for 40 diagnoses)

3.3. Method C: Denormalizing the Dimension Table by Non-Positional-attributes & a Concatenated Field

In this approach, each attribute in the dimension will store a different diagnosis value. By *non-positional* we mean that each attribute can have a different value in different records. Other than the primary diagnosis, there is no difference between secondary 1 and secondary 20.

Table 4: Diagnosis Dimension with Five Positional-attributes

diagnosis_key	CancerFlag	HeartFlag	LungFlag	LiverFlag	KidneyFlag
001	FALSE	FALSE	FALSE	FALSE	FALSE
002	FALSE	FALSE	FALSE	FALSE	TRUE
003	FALSE	FALSE	FALSE	TRUE	FALSE
004	FALSE	FALSE	FALSE	TRUE	TRUE
005	FALSE	FALSE	TRUE	FALSE	FALSE
…	…	…	…	…	…
031	TRUE	TRUE	TRUE	TRUE	FALSE
032	TRUE	TRUE	TRUE	TRUE	TRUE

An example of this method is illustrated in Table 5. A primary diagnosis and multiple secondary diagnoses can be assigned. Here, we introduce the notion of a *concatenated field* to support query processing. A concatenated field is used to store the primary and all the secondary values of the diagnoses using the variable character data type. For example, VARCHAR2 type in Oracle would be able to store 4,000 characters. The LIKE clause of SQL could be employed to search for constrained information. The concatenated value attributes will store diagnosis values in a sorted order. One drawback is most bills have approximately five diagnoses; therefore, there will be many null values in secondary diagnoses. While queries across diagnosis fields can be accomplished with multiple OR clauses, the LIKE clause to the concatenated field will simplify the search query.

However, we note that most commercial database systems do not employ B-tree type index for searching when LIKE clause begins with a wild character. Thus, an efficient string indexing or string search mechanism will enhance the query performance.

In order to resolve the problem of LIKE clause, we can enhance the non-positional model by incorporating the benefits of positional flag attributes. Additional Boolean attributes can be created for common or frequent diagnoses. See Table 6 for an example. Bitmap indexes [OG95, CI98] can be created for these Boolean attributes to facilitate searching based on these common diagnoses. Unusual or intriguing diagnoses could also be included for specific business intelligence purposes. The hybrid method allows both pattern matching with the LIKE command and an index search through a limited number of Boolean fields. The main advantage here would be to constrain the size of the dimension while allowing fast and efficient queries by maintaining the star schema. Consider Table 6 to observe the usefulness of this approach. Most users are interested in a disease category or combination of categories, not a single disease billing code. Multiple codes can be assigned that all indicate the presence of a disease. There may be as many as 20 codes that all indicate the patient has some form of diabetes. The analyst, for reporting or regression purposes, simply needs a field for diabetes that contains "TRUE" or "FALSE". In On-line Analytical Processing (OLAP) designs users can combine (Boolean "AND") diseases by simply selecting "Yes" across a series of OLAP categories. Pre-calculating and storing these clusters makes it simpler for users to query the database and for developers to create OLAP cubes.

Table 5 Denormalized Non-Positional Diagnosis Dimensional Table

Diagnosis Dimension
diagnosis_key (PK)
primary_diagnosis
secondary_diagnosis1
secondary_diagnosis2
secondary_diagnosis3
secondary_diagnosis20
concatenated_ diagnoses (CD)

Note: secondary diagnosis 4 - 19 omitted for brevity

Observe the field concatenated_diagnoses (CD), which is a concatenation of the primary and all the secondary diagnoses related to a patient fact record. The primary diagnosis is included in the concatenated diagnosis to enable the user to search for all assigned diagnoses for a specific medical condition.

Although we normally avoid fields with patterns or lists, an exception in this case is useful. There is no order or weighting to the secondary diagnoses, except the order in which they come to mind of the evaluating physician or the order in which lab tests results become available, so when this diagnosis dimension table is loaded, all the diagnoses in a group are first sorted ascending and inserted across as many diagnosis fields as required. This permits the use of a “wild card” query rather than multiple OR statements, to test whether a specific diagnosis was assigned to a patient bill, regardless of its ordinal position. When searching for a certain disease state, it usually does not matter if the diagnosis is primary or secondary; the physician just wants to ascertain if “any” diagnosis is for example “heart”.

Referring to Table 6:

```
SELECT      Patient_key, BPE. billedtopayer_amount
FROM        DD,BPE
WHERE       DD.diagnosis_key = BPE.diagnosis_key
AND         DD.concatenated_diagnoses
LIKE        ‘%heart%’;
```

Table 6: Denormalized (non-positional) Diagnosis Dimension Table with Positional-attributes

Diagnosis Dimension								
DK	**PD**	**SD1**	**SD2**	**…**	**SD20**	**CD**	**AsthmaFlag (BOOL)**	**DiabetesFlag (BOOL)**
1	heart	lung	asthma	**…**	null	"asthma,heart,lung"	TRUE	FALSE
2	cancer	diabetics	null	**…**	null	"cancer,diabetics"	FALSE	TRUE
3	liver	heart	null	**…**	null	"heart,liver"	FALSE	FALSE
4	heart	lung	asthma	**…**	null	"asthma,heart,lung"	TRUE	FALSE
…								

Note: secondary diagnosis 3 - 19 omitted for brevity

The query is less difficult to write than a query with multiple OR clauses. Adding flag columns for disease groups can further enhance the design. For instance, columns for diabetes and asthma can be used to tag all rows having specific diagnosis codes. Columns for certain DRG codes can be included, since the dimension can be as wide as the designer desires to increase the usefulness to the end user. Note that there is a one-to-many relationship between a flag field and diagnosis codes. That is, the presence of any one or more of a set of diagnosis codes may indicate an overall condition of diabetes. Flag fields are an elegant way for users to create simple queries that ask broad disease questions, but the determination and loading of these fields during the ETL process is complex and usually requires a mapping table created by medical experts. They can be simple TRUE/FALSE flags that allow rapid queries such as:

```
SELECT      *
FROM        DD
WHERE       Asthma = 'TRUE'
AND         Diabetes  = 'TRUE';
```

Note tuples one and four in Table 6. Both tuples have the same primary and secondary diagnoses but have different diagnosis keys. The designer must make a decision, is this type of redundancy acceptable or should measures be taken to search for existing diagnosis patterns before issuing a new diagnosis key? It will be a trade-off between more required memory space, or develop ad-hoc stored procedures to handle this situation.

This method of using non-positional attributes can be implemented in two different ways: by one-to-one or one-to-many relationship between the diagnosis dimension and the fact table, depending on the allowance of redundant tuples.

This scheme described in this section allows the users to only be concerned with a single join between the fact table and dimension table.

3.3.1 Method C-1: One-to-One Relationship between Dimension and Fact Tables

When each record in the diagnosis dimension can be related to one fact record, there exists a one-to-one relationship between the tables (Figure 4). That is, we are creating one dimension record for each new billing encounter. The drawbacks in this design are three. First, most bills have fewer than 5 secondary diagnoses, so there will be many null values. Second, queries across secondary diagnosis fields will require multiple OR clauses, which are complex to write and slow to run. However, this disadvantage can be solved using the concatenated attribute and LIKE clause as we explained in the previous section. Third, it will take more storage. However, the most significant advantage of this approach is to maintain the simple star schema structure. Here, design is simpler in most ways and easier for analysts not trained in data modeling to understand at the expense of significant storage.

A weighting factor could be added to the diagnosis dimension, but will create a complexity in the actual usage and is not recommended in this approach.

3.3.1.1. Database Sizing for Denormalized Dimension Method C-1

Number of base fact: 1,000,000 records
Key fields =7; Number of fact fields =2; Total Fields = 9
Fact table size = 1,000,000 records * 9 fields * 4 bytes =
<u>36 MB</u>
Diagnosis dimension: 1,000,000 records
Key fields =1; Primary diagnosis size = 8 +15 = 23 bytes
Average secondary diagnoses size = 5 * (8+15)= 115 bytes
Average Concatenated field size = 5*8 = 40 bytes
Number of Boolean Flag fields = 3 bits (1 bit each)
Record size = 4 + 23 + 115 + 40 + 1 = 183 bytes
Dimension table size = 1,000,000 records * 183 bytes =
<u>183 MB</u>
<u>Total disk space = 219 MB</u>

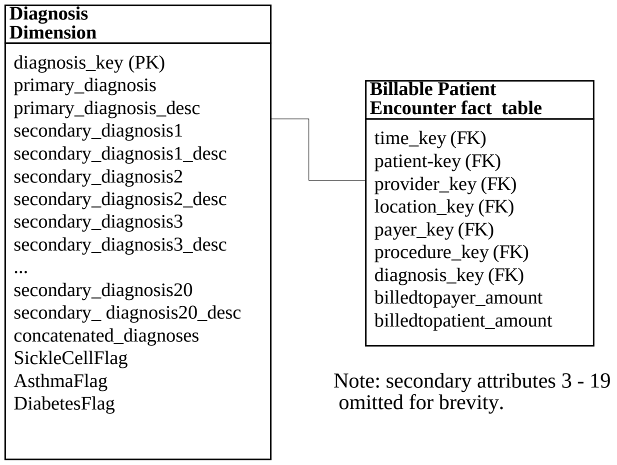


Figure 4 Denormalized Non-Positional Diagnosis Dimension Table with Flag Attributes: One-to-One Relationship between Dimension and Fact Tables

3.3.2. Method C-2: One-to-Many Relationship between Dimension and Fact Tables

The diagnosis dimension can be related to the fact table in a one-to-many relationship (Figure 5). Thus, the same diagnosis pattern is associated with multiple encounters.

This method introduces many null values similar to method C-1, but the redundancy is largely reduced. A sorting/concatenation procedure similar to the one used to create the dimension table explained in section 3.3 can be employed to search for the same diagnosis pattern from existing records, however the insertion process will tend to be complex and slow. Due to the one-to-many relationship, weighting factors cannot be exploited.

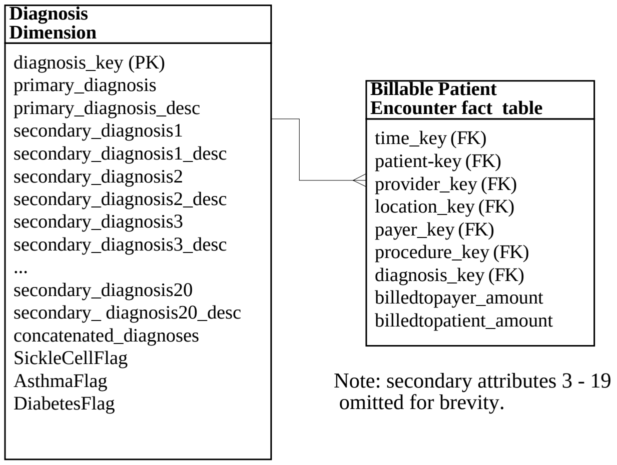


Figure 5: Denormalized Non-Positional Diagnosis Dimension Table with Flag Attributes: One-to-Many Relationship between Dimension and Fact Tables

In our project, we adopted this C-2 method. Many medical centers will purchase or download all necessary diagnosis codes and descriptions as flat files, then load this data into a database table. We were able to create the initial diagnosis dimension using historic legacy data. For each billable encounter, a lookup is performed for the diagnosis description in a lookup table, the results are sorted, and a new record is inserted. For future claims records, a maintenance function will query the diagnosis dimension to see if the pattern already exists. If the pattern does not exist, the lookup table is accessed for a description and will update the dimension accordingly.

3.3.2.1. Database Sizing for Denormalized Dimension Method C-2.

Number of base fact: 1,000,000 records
Key fields = 7; Fact fields = 2; Total Fields = 9
Fact table size =1,000,000 records * 9 fields * 4 bytes =
<u>36 MB</u>
Diagnosis dimension records: 200,000 records
(Assumed on the average one pattern is associated with five encounters.)
Key fields =1;
Primary diagnosis size = 8 +15 = 23 bytes
Average secondary diagnoses size = 5 * (8+15)= 115 bytes
Average Concatenated field size = 5*8 = 40 bytes
Number of Boolean Flag fields = 3 bits (1 bit each)
Record size = 4 + 23 + 115 + 40 + 1 = 183 bytes
Dimension table size = 200,000 records * 183 bytes =
<u>36.6 MB</u>

<u>Total disk space = 72.6 MB</u>

3.4. Method D: Lowering the Grain of the Fact Table

Method D will lower the grain of the fact table to the dimension grain level (Figure 6). This method is briefly discussed By Giovinazzo [Giov00]. For each event there will be multiple fact records relating to that specific event. An option is to add a diagnosis_group_key in the fact table to group the multiple fact records. There is no need to compute a weighting factor; each diagnosis can be directly billed (see table 7). The star schema is retained in this approach, giving the user a concise, clear logical view of the business process, at the expense of increasing the size of a fact table.

Since you can have many diagnoses for a singular event, there is a need to calculate the aggregate of that instance. The approach will be straightforward; sum the amounts of each diagnosis for a specific date, grouping by diagnosis_group_key. The size of the fact table will increase depending upon how many diagnoses will be stored in the fact table. There will also be redundant data stored for other billing fields.

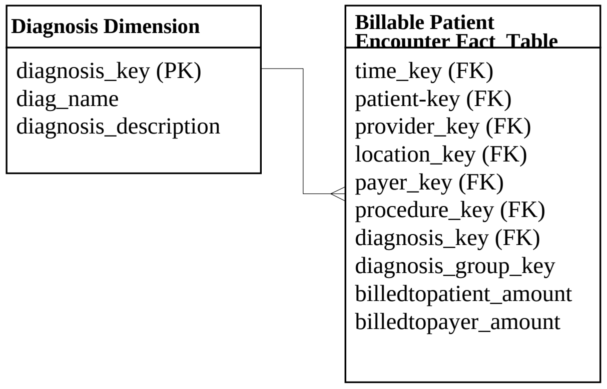


Figure 6: Method D: Lowering the Grain of the Fact Table

Table 7: An Example of the Fact Table at Individual Grain

BPE				
Patient key	**Diagnosis group_key**	**Diagnosis key**	**Other FK's**	**Billed to payer amount**
P1	DG1	cancer	.	$ 1,000
P1	DG1	heart	.	$ 2,000
P2	DG2	cancer	.	$ 1,000
P2	DG2	heart	.	$ 2,000
P2	DG2	lung	.	$3,000

3.4.1. Database Sizing for Method D: Modifying the Fact Table

Number of base fact records: 5,000,000 records (assume five diagnoses on average).

Key fields =7; Number of fact fields =3; Total Fields=10
Fact table size =5,000,000 records * 10 fields * 4 bytes =
<u>200 MB</u>
Diagnosis dimension: 500 records
Key fields = 1; Name field =1; Description field =1
Record size = 4+8+15 =27 bytes
Dimension table size = 500 records * 27 bytes =
<u>13,500 Bytes</u>

<u>Total disk space = 200.1 MB</u>

3.5. Method E: Lower the Grain of the Fact Table: Separating Diagnosis from Billing Data

Method E is similar to method D, but separates the diagnosis data from the billing data by employing two fact tables (Figure 7). Diagnosis data is recorded at individual diagnosis grain in the Patient Medical Record fact table, while a billing is recorded at each billing transaction grain. Using this structure, allows us to use two fact tables in different ways. When we perform diagnosis-related analysis, the Patient Medical Record fact table can be used. When we perform billing–related analysis, the Billing fact table can be used. When we analyze billing related to diagnosis, both fact tables will be used. A billing event is joined to diagnosis through the billing key. A Weighting factor can be used if necessary.

In this approach, the patient medical record would contain redundant data about the patient due to the granularity of the table, which is at the diagnosis level. This method, however, causes less redundancy than Method D because billing is recorded only once per event in its own fact table. The size of the fact table will increase depending upon how many diagnoses are stored in the fact table. There will also be redundant data caused by the foreign keys of an additional fact table.

We note that if Method E is simplified by removing all foreign keys except diagnosis_key and billing_key in the Patient Medical Record fact table, the structure is similar to the method used with the Bridge table.

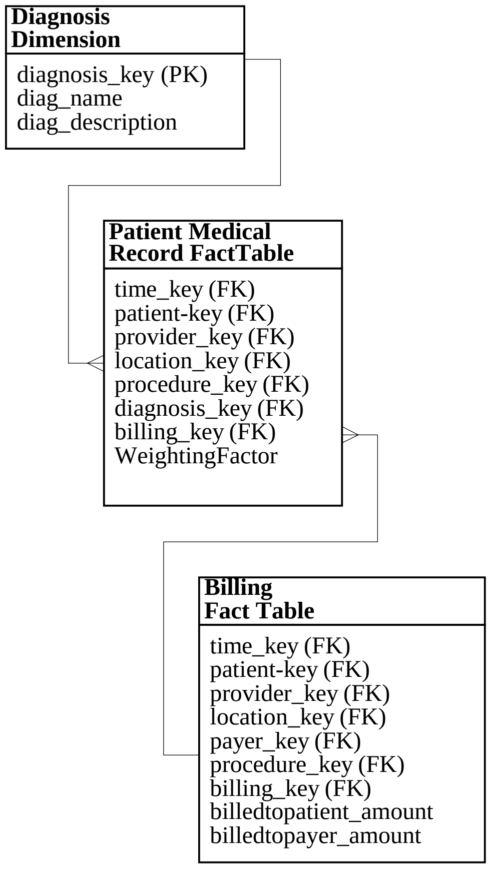


Figure 7: Method E: Lower the Grain of the Fact Table and Separate Diagnosis/Billing Data

Table 8: Summary of Six Methods

Name	Weighting Factor	Queries	Redundancy	Recommendation
			Storage	
Method A Kimball's Bridge	Necessary	Additional join increases query time	Slight	Ideal for dimension with no upper limit in Many side and WF can be easily calculated or not needed.
	WF assignment can be cumbersome			
	Adding new diagnoses require recalculation.		**96.1 MB**	A clean solution
Method B Denormalizing Dimensional Table with Positional-Flag attributes	Cannot handle a weighting factor	Requires many OR commands, slows processing response	Dimension table will list all possible combination, can be extremely large	Because of enormous storage requirement, only applicable when the number of positional-attributes is very small (less than 10) and fixed
	Can use a primary diagnosis flag	Bitmap index scheme can speed up query processing	**10.1TB (for 40 diagnoses)**	
Method C Denormalizing Dimensional Table with Non-positional-attributes & Concatenated field	Depends on relationship between dimension and fact table	Use of concatenated attribute and LIKE clause.	Can be very large	Can maintain star schema structure for ease of understanding and query forming
				Use when the number of dimension values is small and query performance is important
			See C-1 and C-2 below.	Creates many null values
Method C-1 One-to-One	Can create a concatenated weighting factor, but cumbersome	LIKE command disables indexing.	Can be very large	Use when a dimension pattern appears in fact table only 1-2 times.
		Can use flags and bitmap indexes for the flags	**219 MB**	
Method C-2 One-to-Many	Cannot use a weighting factor because of one-to-many nature	Like command disables indexing.	Less redundancy than Method C-1	Use when a dimension pattern appears in fact table many times.
		Can use flags and bitmap indexes for the flags	**72.6 MB**	Requires complex logic to find existing records before inserting a new code.
Method D Lower Grain of Fact Table to Line Item level	No need for a WF	View is on single table	Can cause redundancy for other FK's in fact table	The fact table can become very large
	Can use a primary diagnosis flag.	Need to calculate aggregation.	**201.1 MB**	Use only for a limited number of attributes and line items per transaction
		Fast queries		
Method E Lower Grain & Separate Facts into two tables	Can use a WF	View is on single table	Can cause redundancy for other FK's in fact table	Use when two fact tables can be used separately.
		Must deal with two fact tables.		
		Fast queries		
	Can use a primary diagnosis flag.	Need to calculate aggregation	**196.1 MB**	

3.5.1 Database Sizing for the Positional-attribute Method E

Number of Patient Medical facts: 5,000,000 records (Assume five diagnoses on average).
Key fields = 7; Fact fields = 1
Record size = 7*4 + 4 = 32 bytes
Fact table size = 5,000,000 records * 32 bytes =
<u>160 MB</u>
Diagnosis dimension: 500 records
Key fields = 1; Name field =1; Description field =1
Record size = 4+8+15 =27 bytes
Dimension table size = 500 records * 27 bytes =
<u>13,500 Bytes</u>
Number of billing fact: 1,000,000 records
Key fields =7; Number of fact fields =2; Total Fields =9
Fact table size = 1,000,000 records * 9 fields * 4 bytes =
<u>36 MB</u>

<u>Total disk space = 196.1 MB</u>

4. Summary and Discussion

In this section, we present the summary of the six methods discussed in the paper. The results are summarized in Table 8.

Kimball's bridge method [KRRT98] produces an elegant design for many situations. Redundancy is kept at a minimum, with the added advantage of correct weighted summaries. The cost is the size of the bridge table and added joins can deteriorate the querying process. We note that a weighting factor is needed when using a bridge. It is not always needed when fact and dimension tables are flattened. We recommend this solution as a clean and maintainable solution when the cardinality of many-to-many is not limited and weighting factors can be easily calculated or are not needed.

Method B, positional flag attributes, seems a likely solution when the number of positional-attributes is very limited (say less than 10 or 12) and fixed. As the number of attributes increases, the storage requirement becomes explosive. We do not recommend this approach for most situations.

Methods C-1 and C-2 use non-positional attributes with a concatenated attribute and flags. . By maintaining the star schema structure, these methods enhance understandability of the model by non-professional analysts and support easy query formation. However, the designer must take into consideration the null values and redundancy. In addition, Method C-2 requires an efficient procedure to find an existing diagnosis key for each entry of a fact table. We recommend Method C-1 when a dimension pattern appears in the fact table only 1-2 times and Method C-2 when a dimension pattern appears in the fact table many times. We recommend these methods only when the number of the dimension value is small and fixed.

Methods D and E offer still additional approaches that will work well if the number of diagnoses is limited. The designer must consider the size of the fact table and redundancy in relation to the query processing times.

There are several distinct types of users of the warehouse; the executive browser, the data analyst, and the professional OLAP analysts. The executive browser will see the data through interfaces designed by the warehouse developers and the development will be simpler using Method C-1 or C-2 because there are fewer joins in the queries and no flattening views to create. Data analysts and professional OLAP analysts such as statisticians will need to extract data from the warehouse tables into reports employing statistical and data mining programs. Therefore, the understandability of the model becomes very important. Data from a flatter, horizontal dimension is easier to query into off-the-shelf application packages than data in a vertically oriented table. There is also no concern that analysts will forget to apply the weight factors to the facts.

Maintenance is not appreciably different from a modeling and physical implementation perspective, which are handled using a data-modeling tool. The extract, transform, and load (ETL) routines may be more complex and slow to run for the bridge method due to the multiple tables that have to be tested and the need to use cursors and procedural code to populate the bridge tables. Conversely, the ETL plans for method C may be easier to write and should run faster. It is also important to note, that some fact tables will have multiple many-to-many relationships with dimensions other than diagnosis. While theoretically, this poses no problem; in practice, the load times and query times may be excessive.

We finally note that the storage calculation in this paper was based on our assumption. When the domain and assumption changes, the storage requirement needs to be recalculated.

5. Conclusion

While we were building a real-world patient-billing data warehouse, we met a many-to-many relationship problem between a fact table and a dimension table. A survey of literature showed us two methods (Method A and Method D). After a thorough study on the subject, we have identified four additional methods. In this paper, we have analyzed those six different approaches for handling many-to-many relationships. We have illustrated and shown the advantages and disadvantages of each solution.

Our preferred methods include two ad-hoc approaches that maintain a star schema structure by denormalizing the dimension to avoid many-to-many relationships. For our project, we implemented Method C-2. For a new encounter, we query the Diagnosis dimension to see if the pattern already exists. If the pattern does not exist, the lookup table is accessed for a description and will update the dimension accordingly. We also note that Method C-2 uses the minimum storage. Our experience shows that the Method C-2 proposed in this paper were easy to use and efficient given our situation. However, we recommend each data warehouse designer carefully evaluate each case to adopt the best method considering various options. Just as database designers trade off normalization for query response, a data warehouse designer must also resolve many issues to solve this many-to-many dilemma.

Acknowledgements

The authors thank the many students at Drexel University who participated in the discussion on the many-to-many dilemma in data warehousing modeling. Their insight has been invaluable in developing these solutions.

6. References

[AM97] Anahory, S. and Murray, D., *Data Warehousing in the Real World*, Addison Wesley, 1997

[AV98] Adamson, C. and Venerable, M., *Data Warehouse Design Solutions*, John Wiley, 1998.

[AS97] Axel, M. and Song, I. -Y., "Data Warehouse Design for Pharmaceutical Drug Discovery Research," *Proc. of 8th International Conference and Workshop on Database and Expert Systems Applications (DEXA97)*, September 1-5, 1997, Toulouse, France, pp. 644-650.

[CI98] Chan, C.-Y. and Ioannidis, Y. , Bitmap Index Design and Evaluation, *Proc. of 1998 SIGMOD Conference*, pp. 355-366.

[DSHB98] Dinter, B., Sapia, C., Hofling, G., and Blaschka, M., The OLAP Market: State of the Art and Research Issues, Proc. *of Int'l Workshop on Data Warehousing and OLAP*, Washington, D.C., 1998, pp. 22-27.

[Giov00] Giovinazzo, W.A., Object-Oriented Data Warehouse Design: Building a Star Schema, Prentice Hall, 2000.

[GR98] Golfarelli, M. and Rizzi, S., A Methodological Framework for Data Warehouse Design, *Proc. of Int'l Workshop on Data Warehousing and OLAP*, Washington, D.C., 1998, pp. 3-9.

[HLB00] Husemann, B., Lechtenborger, J., and Vossen, G., Conceptual Data Warehouse Design, *Proc. Of Int'l Workshop on Design and Management of Data Warehouses*, Stockholm, Sweden., 2000.

[Kimb96] Kimball, R. (1996). *The Data Warehouse Toolkit*. New York: John Wiley & Sons, Inc.

[Kimb97] Kimball, R., A Dimensional Manifesto, *DBMS*, August 1997, pp. 58-70.

[KRRT98] Kimball, R., Reeves, L., Ross, M., & Thornthwaite, W. (1998). *The Data Warehouse Lifecycle Toolkit.* New York: John Wiley & Sons, Inc.

[KS97] Krippendorf, M. and Song, I.-Y, "Translation of Star Schema into Entity-Relationship Diagrams," *Proc. of 8th International Conference and Workshop on Database and Expert Systems Applications (DEXA97)*, September 1-5, 1997, Toulouse, France, pp. 390-395.

[LAW98] Lehner, W., Albrecht, J., and Wedekind, H., Normal Forms for Multidimensional Databases, *Proc. SSDBM, 1998, pp. 63-72.*

[MC98] Maier, D. and Cannon,C., *Building a Better Data Warehouse*, Prentice Hall, 1998.

[MK00] Moody, D. and Kortink, M.A.R., From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart design, *Proc. of Int'l Workshop on Design and Management of Data Warehouses*, Stockholm, Sweden., 2000.

[OG95] O'Neil, P. and Graefe, G., Multi-table Joins Through Bitmapped Join Indices, *SIGMOD Record*, Vol. 24, No.3, Sept. 1995, pp. 8-11.

[PJ99] Pedersen, T.B. and Jensen, C.S., Multidimensional Data Modeling for Complex Data, *Proc. of 15th ICDE*, Sidney, Australia, 1999, pp. 336-345.

[SBHD98] Sapia, C., Blaschka, M., Hofling, G., and Dinter, B., "Extending the E/R Model for the Multidimensional Paradigm," *Advances in Database Technologies (ER '98 Workshop Proceedings)*, Springer-Verlag, pp. 105-116.

[TBC99] Tryfona, N., Busborg, F., and Christiansen, J.G.B., "starER: A Conceptual Model for Data Warehouse Design," *Proc. of Int'l Workshop on Data Warehousing and OLAP (DOLAP 99)*, Kansas City, MO., pp. 3-8.

[TS98] Theodoratos, D., and Sellis, T, Data Warehouse Schema and Instance Design, *Proc. of 17th International Conf. On Conceptual Modeling (ER98)*, Singapore, Nov. 1998, pp.363-376.