Integration of Multidimensional Data in Heterogeneous Data Marts

A thesis submitted for the degree of

Doctor of Philosophy

Dariush Riazati B.Comp., M.Sc, M.Tech.

School of Computer Science and Information Technology

College of Science, Engineering and Health,

RMIT University,

Melbourne, Victoria, Australia.

August, 2012

Declaration

I certify that except where due acknowledgment has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

Dariush Riazati

School of Computer Science and Information Technology

RMIT University

August, 2012

Acknowledgments

In the Name of God, the Most Exalted, the Most Holy

"Heaven in every being is its utmost perfection within its limits."

Bayan (Unity II, Chapter XVI)

This is the most significant milestone in my academic career in which I invested most of my after-work time for seven years. This would not have been possible without the sacrifices that my wife, my best friend Minoo has made, and the support she has given to me during these years. I am always grateful to her.

I also thank my parents who supported me with more than everything that they had in their possession, and were capable of. I would also like to thank my dearest sister Azar_Mah to whom I had to say farewell recently, in the saddest moment of my life. She is and will for ever be in my heart and my mind; I miss very much your warm and heartfelt support.

I would like to thank my primary supervisor Associate Professor James Thom for countless long review sessions and for his extraordinary support and many useful ideas and suggestions for improvement. I have been extremely fortunate to have Professor Thom as my supervisor. I would also like to thank Dr. Xiuzhen Zhang my secondary supervisor for her many feedbacks. Dr Zhang's insight into data warehousing was critical in improving the quality of this thesis.

Credits

Parts of the material in this thesis have previously appeared in the following publications:

- D. Riazati and J. A. Thom. Matching star schemas. *In Proceedings of the 22nd international conference on Database and expert systems applications - Volume Part II* [Riazati and Thom, 2011].
- D. Riazati, J. A. Thom, and X. Zhang. Inferring aggregation hierarchies for integration of data marts. *In Proceedings of the 21st international conference on Database and expert systems applications: Part II* [Riazati et al., 2010].
- D. Riazati, J. A. Thom, and X. Zhang. Enforcing strictness in integration of dimensions: Beyond instance matching. *In Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP* [Riazati et al., 2011].
- D. Riazati, J. A. Thom, and X. Zhang. Drill across and visualization of cubes with non-conformed dimensions. *In Proceedings of the nineteenth conference on Australasian database* [Riazati et al., 2008].

The thesis was written in the `WinShell` editor on Windows 7, and typeset using the LaTeX 2_ε document preparation system.

Note

Unless otherwise stated, all fractional results have been rounded to the displayed number of decimal figures.

Contents

List of Figures

List of Tables

Abstract

Data analysts often require access to integrated multidimensional data from local and external data warehouses. The integration process is often undertaken by expert database practitioners who will need to analyze the structure of the data, and match schemas and data before creating an integrated view of the data for visualization and analysis. Such a manual process may be acceptable for databases used in transaction processing applications but does not help decision makers who need access to the information quickly and cost effective in a constantly changing environment.

This thesis addresses several challenges towards automating the integration of data warehouses based on a dimensional model known as Star schema. We recognize that the structure of multidimensional data, namely dimension hierarchies, is critical to the accuracy of the integration but is not always available or accessible. To address this problem, we infer dimension hierarchies from their instances, and demonstrate that they are sufficient to ensure the accuracy of the integration even though they may vary from the intended hierarchies.

To improve the accuracy of matching Star schemas, we propose a more precise represen-

tation of Star schemas and demonstrate its effectiveness by comparing it against the existing approaches that treat Star schemas as relational models.

To match instances of dimensions, we demonstrate that a graph matching algorithm is effective and performs with a high level of accuracy. We propose algorithms which enforce the tree structure of integrated data which is necessary for correct aggregation, and reduce false positive cases occurring during the instance matching. The effectiveness of our algorithms is shown through experiments with real life data.

Despite perfectly matching schemas and hierarchies, there are often dimensions with mismatching data which restrict the scope of the integration. We propose to relax the requirement for dimension compatibility, and introduce measures that quantify the loss of data resulting from the less strict requirement. These measures enable data analysts to identify lossless fragments of data, and thereby, extend the scope of the integrated data. To provide a more comprehensive view of data for analysis, we link the integrated data with the data exclusive to each source by extending the navigation operation for multidimensional data.

These contributions help towards shifting the integration problem away from expert database practitioners to empowered data analysts in combining multidimensional data from multiple sources in real time, and in a cost effective manner.

Chapter 1

Introduction

> ”Powerful is he, who is knowledgeable,
>
> Young is the heart of the old from knowledge.”

Ferdowsi (940 - 1020)

If Ferdowsi, the most famous Persian poet, could travel in time to the year 2012, the advice he would give to the decision makers of today would be the same as that he gave to the kings one thousand years ago. Knowledge is the most effective tool in problem solving and decision making. The more accessible and comprehensive, the more effective it is. By implication Ferdowsi tells us that it is by ”knowing” that we can gain confidence, feel secure and empowered. Problems that we face today are much more complex than in his time but the tool being knowledge remains the same. Knowledge is the concise and appropriate collection of information in a way that makes it useful [Satama, 2012]. Dispersed information is difficult to use effectively and therefore is not high value knowledge. This thesis concerns integration of analytical data empowering decision makers.

After the global financial crisis in 2008, most believed it could be prevented if the information gathered on the vehicle market, housing market, lenders performance and employment figures were shared and analyzed together [Shefrin, 2009]. Similarly, subsequent investigations into the tragic September 11 terrorist attacks revealed that the main cause of failure was not the absence of the information but that they were not shared for more effective cross analysis [Best, 2007].

Companies invest millions of dollars in producing hundreds of reports, yet, in many cases, they need to rely on experts to combine the information from multiple but existing sources. These accounts point to a challenge beyond information gathering: how to combine the information gathered on related topics to provide a more complete picture quickly and cost effectively.

Classic business intelligence applications have been developed using an approach similar to transaction processing applications. An iterative process that starts with identifying a set of questions users regularly need to have answers for and ends with a set of reports with answers to those questions. However, data analysts are no longer satisfied with prepackaged reports and find it restrictive to specify precisely what their questions are.

They usually know the first question but their subsequent questions depend entirely on the answers they get from the previous questions, a process that could be shown using a cause and effect diagram. Adding to the complexity of the integration process, external unfamiliar data such as past weather reports and statistical data collected by the Bureau of Statistics and others often need to be integrated with the local data.

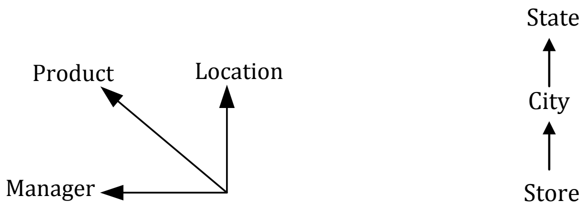
Many autonomous data marts in large enterprises are developed over years. The integration of these data marts into an enterprise warehouse for enterprise-wide large scale analysis is a strategic business objective [Business Objects and Teradata, 2007]. Therefore, a different approach is needed to meet these challenges. Business intelligence applications are now entering into a new generation whereby greater emphasis is made on empowering data analysts to navigate through, integrate and analyze data from multiple sources.

The objective of this thesis is to address the main issues in integration of data marts specifically. In the remainder of this section, the background is first introduced, then the research problems and our proposed solutions are described.

1.1 Multidimensional Databases and OLAP

Multidimensional databases are structured along *dimensions* and *facts* [Torlone, 2008] and used for data analysis. Facts are measures such as `Sales Amount` that can be summarized, and dimensions determine the groupings by which facts are summarized. Using the example in Figure 1.1(a) we could consider `Location`, `Product` and `Manager` as dimensions of a fact measure called `Sales Amount`.

The model for multidimensional data should support fast aggregation of data according to the hierarchical structure of dimensions. Figure 1.1(b) shows the dimension hierarchy for `Location`. For example, the summarized sales amount for each state is obtained by summing the measure for their cities, and the summarized data for cities is obtained by summing the same measure for their stores.



(a) Dimensions of `Sales Amount`.

(b) Dimension hierarchy for `Location`.

Figure 1.1: Dimensions and facts in multidimensional data

1.1.1 Data Cubes

A *data cube* is a visualization of multidimensional data where its axes are represented by dimensions and its cells are represented by facts. It is often referred to as a hypercube since it cannot be visualized once the number of dimensions exceeds three.

OLAP Operations: Online Analytical Processing (OLAP) and its related technologies enable data analysts gain insight into multidimensional data through a set of OLAP operations against (OLAP) data cubes. OLAP operations transform raw data so that it reflects the real dimensionality of the enterprise as understood by users [OLAP Council, 1997]. *Slicing* is selecting data using one of the dimensions [Han and Kamber, 2006]. Figure 1.2 shows visualization of a data cube with three dimensions: `Location`, `Product` and `Manager`. It also shows all slices of cubes for each of the three dimensions.

Dicing is selection of data using two or more dimensions and applying some constraint on the remaining dimensions [Han and Kamber, 2006]. Figure 1.2 shows dicing of the data cube using different combinations of dimensions.

Obtaining the aggregated data at a higher level such as `State`, by summing the values at a lower level such as `City`, is done using an operation that is referred to as *roll-up*. Con-

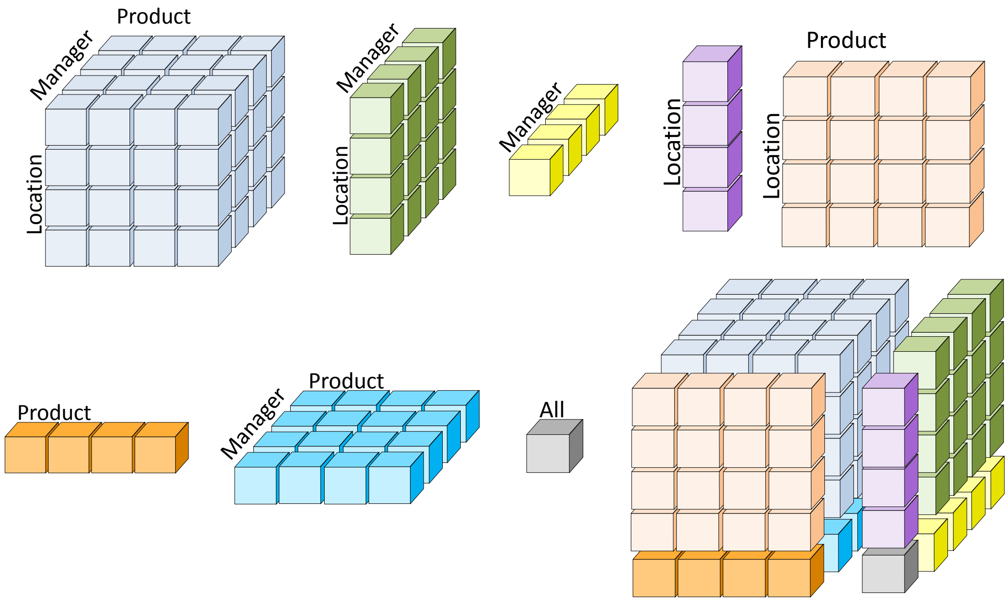


Figure 1.2: Visualization of slicing and dicing of an OLAP data cube.

versely, obtaining the data at a lower level is done using an operation that is referred to as *drill-in*.

In the remainder of this section, OLAP objects and operations to which we refer in this thesis are described. ROLAP (Relational OLAP) and MOLAP (Multidimensional OLAP) are the two main implementations of OLAP. MOLAP servers directly store multidimensional data in an array structure and implement OLAP operations over these data structure, whereas, ROLAP servers store data in relational databases and use SQL and its extensions to implement the OLAP operations [Chaudhuri and Dayal, 1997].

Cube Lattice: A data cube is often represented using smaller cubes called *cuboid*, where each cuboid represents a different level of aggregation [Han and Kamber, 2006]. Figure 1.3 shows a lattice of cuboids and their relationships using none, 1, 2, and 3 dimensions. The data cube in Figure 1.2 is only one of the cuboids which sits at the base of the lattice.

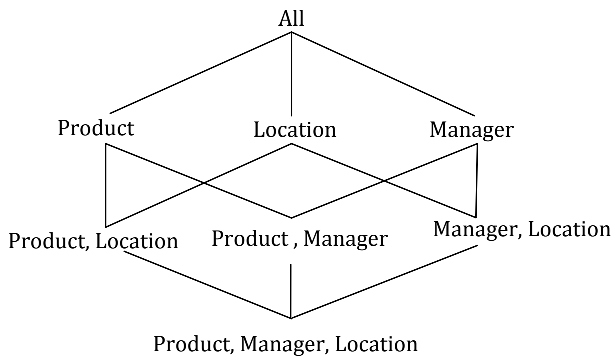


Figure 1.3: Lattice formation of cuboids in a data cube.

1.2 Data Warehouses

A *data warehouse* is a large collection of historical data used for analytical reporting [Hurtado et al., 1999] where related data is organized into subject areas such as Sales, Inventory, and Orders which form the basis for reporting.

There are two approaches to build data warehouses: (i) Kimball and Ross [2002] consider a data warehouse as a union of smaller data warehouses (also called *data marts*) for each subject area, modeled based on Star schema and sourced directly from data sources; (ii) Inmon [2005] defines a data warehouse as a subject oriented, integrated and time variant collection of data.

Data warehouses are sourced from Online Transaction Processing (OLTP), or the operational databases, using the ETL process described in Section 1.2.2. There are several reasons why OLTP databases are not directly used for the purpose of reporting. Some of the more important reasons are as follows [Chaudhuri and Dayal, 1997; Kimball and Ross, 2002]:

- Accessing replicated records from the warehouse shields the operational database from deadlocks resulting from concurrent access, degradation of query performance, unauthorized access and database corruption.

- The data warehouse provides an integrated view of data from multiple sources.
- The model for data warehouse is designed specifically for enhanced query performance.
- The data warehouse contains historical records, whereas, the OLTP database may only include the most recent changes.

1.2.1 Star Schemas

A data model that describes multidimensional data using relational tables is called a dimensional model. *Star schema* is a relational model for describing dimensional models. It consists of two types of tables, *dimension* and *fact* tables.

Dimension is a concept which when implemented in the context of Relational OLAP, is called dimension table. It represents the scheme of a dimension, a relation over a set of attributes one of which represents a unique key. The relation $D(A_1, ..., A_n)$ represents a dimension table D of n attributes. An instance of a dimension table is the data with which it is populated at some point in time.

A fact table is the relational implementation of cube cells each of which is at the intersection of multiple dimensions. It is a relation over a set of fact elements, and a set of key attributes each uniquely identifying a tuple in an instance of a dimension tables it refers to. Therefore, each tuple in the fact table is uniquely identified by joining the fact table to its dimension tables. Fact elements include additive measures (i.e. measures that can be aggregated), whereas, dimension tables include categories by which measures are aggregated.

The topology of a dimensional model based on Star schema resembles a star at the center of which is a single fact table and all around it are the dimension tables. The relationship from the dimension table to fact tables is of 1:M (i.e. one-to-many). That is for each tuple in a dimension table there may be multiple tuples in the fact table, and each tuple in a fact table may refer to no more than one tuple in any dimension table. Relationships are enforced using foreign key constraints. Figure 1.4 shows a Star schema for reporting on car sales using dimensions `MAKE`, `MODEL`, `FISCAL_CAL` and `DEALER`.

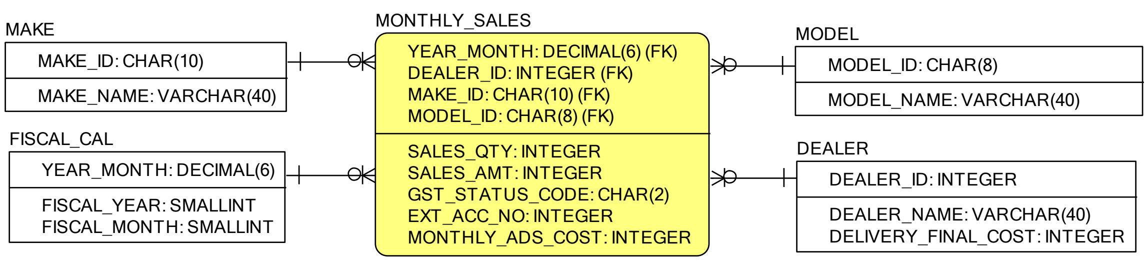


Figure 1.4: An example of a Star schema.

1.2.2 Building Data Warehouses

Using Inmon's approach, data marts are created from the integrated (or enterprise-wide) data warehouse (EDW) as opposed to being directly from the original sources of data. Choosing between the two approaches depends on the number of original data sources and complexity of their transformations into a unified data warehouse. Inmon's approach is more flexible and reduces the risk of non-performing queries but is more costly to build.

Figure 1.5 shows the life cycle of data from sources to cubes. Representing Kimball's

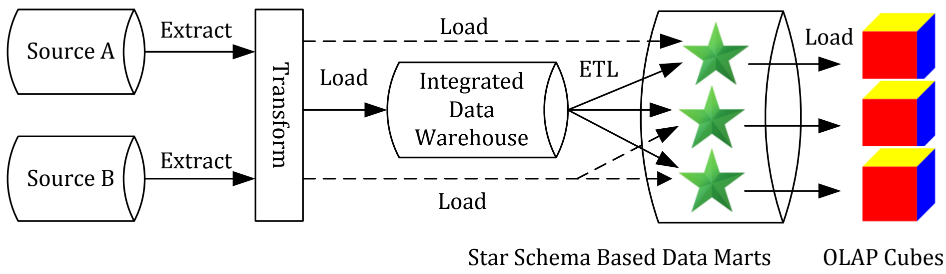


Figure 1.5: Data warehouse life cycle.

approach, dashed lines represent the option to bypass the EDW and update the data marts directly. In this case the collection of data marts is the data warehouse.

Changes to the source data known as delta are incrementally extracted through a change data capture technology. Following Inmon's approach, given the target data model in EDW and consistent mapping definitions from multiple sources into EDW, delta changes go through a set of transformations and are finally loaded into the EDW. This process is known as *Extract*, *Transform*, *Load* (or ETL). A second round of ETL processes transforms the data from the EDW into the data marts. Following Kimball's approach, delta changes are transformed directly into the data marts. At this time the data in the data marts is in a multidimensional structure and can be loaded into data cubes.

1.3 Statement of Problems

Integrating existing data marts into a single data mart is also done using the ETL process. It requires mapping between Star schemas for each data mart, identifying dimension hierarchies, and the matching and integration of their data. These functions are usually done by expert database practitioners and are, therefore, expensive and time consuming. This thesis is concerned with automating these tasks. It aims to shift the integration problem away from database practitioners to the business analysts. To achieve this objective, the following challenges are met:

1. Schema matching: Can we improve the quality of matching between Star schemas if we describe them more precisely than standard description of relational schemas?

   There has been significant research invested in matching relational schemas using schema, semantic and data properties. However, very little study has been made on taking advantage of the simplicity and the predictable topology of multidimensional data based on the Star Schema model. The main drawbacks of these studies are that they are concerned with a limited set of Star Schema properties, and do not provide a necessary precursor process which discovers these properties.

2. Inferring aggregation hierarchies: Dimension hierarchies are not always known or accessible for multidimensional data on heterogeneous data sources. In these scenarios, how do we infer hierarchies in such a way that they ensure the accuracy of the integration of multidimensional data?

   Existing work on inferring aggregation hierarchies using ontologies and dictionaries do not guarantee a fitting hierarchy for the data. The existing literature proposes a method for inferring relationships between dimension attributes based on the data, but it does not determine hierarchy levels and distinct hierarchical paths.

3. Enforcing strictness: A dimension whose instance conforms to the tree structure of its hierarchy is said to be *strict*. What are the issues that cause non-strictness when

integrating strict dimensions and how do we overcome them?

Much of the existing work on enforcing strictness is based on the premise that the data matching has produced perfect match results. They are mostly concerned with managing the non-strict cases. In fact many of the non-strict cases resulting from integration of originally strict data are caused by false positive matching cases, and the presence of homonym data values. This condition has been overlooked in the current literature.

4. Extending the scope of integration: Integrating dimensions must be compatible, that is they must share similarity between their levels, hierarchies and instances. How do we get the most out of the data integration in the presence of non-compatible dimensions? How do we extend the analysis space for the integrated data to include the related but non-compatible dimensions?

   There is considerable literature on using various data operations to maximize the extent of the data integration. What they lack is that they do not help identify segments of data that benefit from such operations. Moreover, the integration and visualization of multidimensional data has been limited to common data, making cumbersome the navigation from the integrated data to the data that is not common in the original sources.

1.4 Contributions

In this thesis, the following contributions are made:

1. We propose a more precise description of Star schemas (StarMod) and represent that using UML and OWL languages. This representation describes multidimensional data in terms of properties of Star schema model. StarMod is a more expressive and rich representation of Star schemas than their relational representation. To measure the effectiveness of StarMod in improving matching Star schemas, we perform an evaluation of the approach using two well cited matching algorithms against 18 pairs of Star and non-Star schemas described using relational and StarMod properties. Our experiments show that StarMod improves matching results for Star schemas and can be also effectively used for arbitrary relational schemas.

   Our proposed approach to matching Star schemas has several advantages over the existing related work: (i) it provides an extensible ontology for Star schemas; (ii) it takes advantage of a more comprehensive set of Star Schema properties; (iii) it can be extended to include semantic properties.

2. We propose algorithms to infer dimension hierarchies from instances of dimension tables. We establish that where dimension hierarchies are not available as part of the schema definition, inferred hierarchies are sufficient to ensure the accuracy of the integration.

   Our proposed method does not stop at calculating the cardinality between dimension attributes. Its advantage over the existing work is that it derives hierarchy levels as well as distinct hierarchy paths.

3. False positive cases resulting from instance matching are the likely causes of inconsis-

tencies in the integrated data leading to non-strict cases. We propose algorithms that reduce false positive cases and enforce the strictness at the same time. The effectiveness of our approach and algorithms are demonstrated using experiments with real life data.

To the best of our knowledge, none of the existing work has taken advantage of enforcing strictness to enhance the quality of matching results between instances of dimensions.

4. We relax the requirement for compatibility by excluding the requirement to have fully matching instances of dimensions. We propose measures that quantify the loss resulting from integration of dimensions with partially matching instances, and use these measures to identify lossless fragments of data. The operation to navigate between data marts is extended to include data related to non-compatible dimensions. We also propose an extension to Pivot tables, used to visualize multidimensional data, to support the extended navigation operation.

   Our contribution in this area complements existing work aimed at extending the scope of data integration by performing operations that provide better mapping between otherwise mismatching data. We have offered several methods to measure the loss of data resulting from the integration of data using two or more dimensions. Our proposed extension to the drill across operation enables the user to remain in the same analysis space despite the presence of non-compatible dimensions. A working model of this extension is future work.

1.5 Organization of the Thesis

In Chapter 2, the evolution and basic concepts in data warehousing are described. The review of literature concerns the four areas of schema matching, dimension hierarchies, instance matching, and extending the scope of integration.

Chapter 3 concerns the schema matching for Star schemas which is a key aspect of integrating multidimensional data.

To set the necessary ground for instance matching, Chapter 4 describes the algorithms used to infer aggregation hierarchies.

Chapter 5, addresses instance matching for dimension tables and resolves non-strict cases in the integrated data.

Chapter 6, explains methods to quantify the loss during the integration of non-compatible dimensions and describes an approach to extending the navigation operation and Pivot tables to include non-compatible dimensions.

Finally, Chapter 7 concludes the thesis and describes directions for future work.

Chapter 2

Integration of Multidimensional Databases - Literature Review

> ”If the word integration means anything, this is what it means: that we, with love, shall force our brothers to see themselves as they are, to cease fleeing from reality and begin to change it.”
>
> James Arthur Baldwin (1924 - 1987)

The earliest work on data warehousing originates from a joint research project conducted by General Mills and Dartmouth University in the 1960s from which the concepts of dimensions and facts originate. Later, during the 1970s, multidimensional data marts were first introduced by AC Nielsen for the purpose of reporting [Kimball and Ross, 2002]. In 1992, Inmon described in detail the concept of the data warehousing architecture in a classic text book called *Building the Data Warehouse* [Inmon, 2005]. In 1996, Kimball described his view of how a data warehouse should be built in his book *Data Warehouse Toolkit* [Kimball and Ross, 2002]. Whilst, the merits of the approaches introduced by Inmon and Kimball have been widely debated amongst data warehouse practitioners, these concepts continue to be employed today in most data warehousing projects.

Data vault is a hybrid approach that encompasses Third Normal Form (3NF) (introduced by Inmon) and Star schema (introduced by Kimball) [Linstedt et al., 2011], but is not optimized for query performance [O'Neil, 2004]. It was introduced by Linestedt in 1990 [Linestedt, 2011]. The first book on data vaults [Linstedt et al., 2011] was released in 2011. The concept of Data Vaults is a variation of approaches proposed by Kimball and Inmon, but has not been widely received by the industry or research community.

In this chapter, we present from the existing literature, those most relevant to our contributions. In Section 2.1, we describe the state of the art methods in representation and matching of Star schemas. In Sections 2.2 to 2.4, we present an overview of works related to inferring dimension hierarchies. In Section 2.5, we review the existing literature concerned with matching instances of hierarchical data, and finally, in Section 2.6 we describe approaches to extending the scope of integration and visualization of multidimensional data.

2.1 Schema Matching

Schema matching for relational databases is a difficult process to automate, and has been the subject of research in last decades. Batini et al. [1986], Pavel and Euzenat [2004] and Gal [2006] have completed major surveys of research in this area.

Schema matching is a process that uses one or more matchers to find mappings between elements of the two schemas with the motivation to integrate schemas [Rahm and Bernstein, 2001]. The approach used in schema matching depends on how the schemas are represented. Examples are data definition language (DDL), entity-relationship (ER) model, Unified Modeling Language (UML), XML, document type definition (DTD), or ontology web language (OWL) [Rahm and Bernstein, 2001].

Rahm and Bernstein [2001] classify approaches to automate schema matching as being at either the level of element or structure. The advantage of the structure level approach is that in addition to element-level properties such as name, data type and constraints, it also considers the similarity between related elements within the same structures.

2.1.1 Representing Schemas as Graphs

The structure and properties of relational databases can be described using graphs. Each relational object, such as instances of tables, columns, keys and datas type is represented as a node of a graph. The edges in the graph simply relate each pair of nodes using role names.

Figure 2.1 is a graph representation of the table `DIMENSION_FISCAL_CAL` which appeared in Figure 1.4. It follows a similar representation used by Melnik et al. [2002b].

2.1.2 Similarity Flooding

Similarity Flooding [Melnik et al., 2002b] is a versatile graph matching algorithm. It calculates similarity between nodes of two graphs A and B. For propagation purpose, it forms a pairwise connectivity graph PCG from $A \times B$ connecting nodes of the graphs (x, p, x')

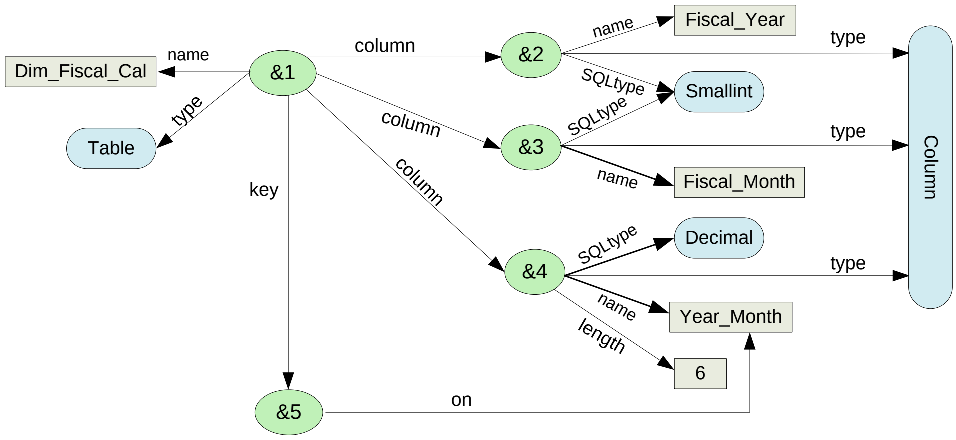
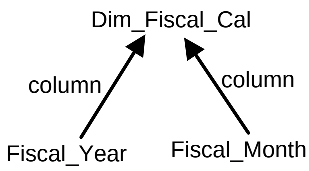
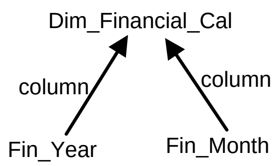


Figure 2.1: Graph representation of dimension table `DIMENSION_FISCAL_CAL`.

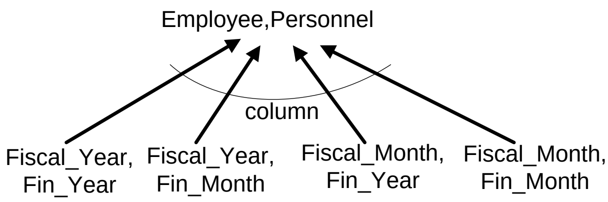
and (y, p, y') and is defined by Melnik et al. [2002b] as: $((x, y), p, (x', y')) \subseteq PCG(A, B) \iff (x, p, x') \subseteq A$ and $(y, py') \subseteq B$. Figure 2.2 shows the connectivity graph between two graphs each representing a column of a table.



(a) Column 1.



(b) Column 2.



(c) Connectivity graph of columns 1 and 2.

Figure 2.2: An example of a connectivity graph.

At each level of the connectivity graph, the similarity score for each node in PCG is calculated as follows:

- Each node in PCG receives an initial similarity value which indicates the string similarity between the labels of the nodes from A and B.
- Each edge in PCG is represented using a forward link (from parent to the child node) and a backward link (from the child to the parent node).
- The weight of the forward link is calculated as 1 divided over the number of child nodes for the same parent.
- The weight of the backward link is calculated as 1 divided over the number of parent nodes for the same child; in a tree structure, this is always 1.
- The similarity score for the parent node is incremented by the sum of the score for each child node multiplied by the weight of its backward link.
- The similarity score for each child node is incremented by the similarity score of its parent node multiplied by the weight of its forward link.
- The scores are normalized in each iteration by being divided over the largest score in the iteration.
- In each iteration, the algorithm calculates the similarity scores for all nodes. The algorithm stops when the change in the scores becomes insignificant. Finally, the best match candidates are selected through some filters.

2.1.3 COMA++

COMA [Do and Rahm, 2002] is another well known schema matching approach. As demonstrated by Duchateau et al. 2008, the quality of match results from COMA++ is generally higher than that of Similarity Flooding. The approach used by COMA++ also turns the schema into an internal graph representation where each element is represented using a *path* or a sequence of nodes following the containment links from the root to each node.

COMA++ uses a library of matchers which can be plugged into COMA and configured. COMA++ has three types of matchers: *Simple*, *Hybrid* and *Reuse-oriented*. Simple matchers benefit from syntactical similarities between the element names. Hybrid matchers exploit the structure of graphs.

COMA++ also learns from feedbacks obtained from user. At the end of the matching process, COMA++ combines match results returned from its matchers. This process consists of an *aggregation* of scores and a *selection* from match candidates.

In the remainder of this section we review the literature related to specifically matching of Star schemas. Precursors to the matching of Star schemas are discovery and representation of their properties. Next, we survey the literature related to these two problems.

2.1.4 Inferring Star Schema Properties

Various sources of information have been used to identify properties of Star schemas. The earliest work goes back to when Pokorny [2001] used elements and sub-elements of Document Type Definition (DTD) to describe dimensional data. Similarly, Jensen et al. [2001; 2003] use a DTD as a source of information. They identify the multidimensional structure by following elements with ID/IDREFS attributes and suggest UML diagrams for snowflake dimensions, and facts. DTDs are superseded by XML Schema which has a much richer syntax. A more fundamental shortcoming of these approaches is that they do not go beyond dimensions and their hierarchies.

Song et al. [2007] produce Star schema from Entity Relationship Diagram (ERD) by analyzing their structure and by measuring the number of direct and indirect M:1 (many-to-one) relationships for each entity. ERD is a rich source of information on the structure and semantic information such as the relationships, but it is seldom available. Moreover, their algorithm stops at identifying the dimensions and fact tables.

Jensen et al. [2004] use the metadata model and data to discover dimensions and their hierarchies. Their proposed approach appears to include transitive relationships and also uses SQL queries which makes the approach less practical.

Golfarelli et al. [2001] use DTDs as well as key/keyref elements in XML schemas relating to web pages to construct a graph from which attribute trees are constructed, but the user is expected to use the graph to identify facts.

Romero et al. [2009] discover functional dependencies to identify the multidimensional structures only as far as dimension hierarchies. They use domain ontologies as the source of information which is seldom available. Cabibbo et al. [1998] identify dimensions and facts by following the foreign key relationships.

Carmè et al. [2010] have a focus on identifying fact tables and their measures only. Their approach is based on heuristics that take advantage of relationships, the volume of data, and clusters of numeric data. Similar to Song et al. [2007], Carmè et al. use the number of foreign keys to identify fact tables. Using the volume of tables as a measure to detect fact tables is not reliable since fact tables in each data mart may refer to only a subset of data in each dimension.

2.1.5 Representation of Star Schemas

Representation of multidimensional data using ER diagrams has received considerable attention in the past decade [Chen and Hsu, 2007; Franconi and Sattler, 1999; Kamble, 2008]. These works suffer from similar problems as those concerned with discovery of Star schema properties, that is, they leave out properties such as surrogate keys, degenerate facts, and degenerate dimensions.

More recent works have benefited from UML to describe multidimensional data. Akoka et al. [2001] present a meta model for dimension hierarchies using UML operations such as generalization and aggregation but only focus on describing aggregation hierarchies.

Abelló et al. [2005] present a meta model called YAM2 for multidimensional data by extending UML stereotypes. Their model captures facts, dimensions, levels, measures and summarizations. The Star schema described using this model is, however, designed from the user requirements and is, hence, more suitable for designing new models. They also omit representation of important properties such as snowflaked dimensions, degenerate dimensions and facts.

Luján-Mora et al. [2006] propose a conceptual model using UML with a wider coverage of properties of multidimensional data than YAM2. It covers a comprehensive set of properties including hierarchies, degenerate dimensions and degenerate facts. It is, however, aligned with the logical or conceptual model and as such it excludes physical properties such as keys, foreign key relationships and snowflaked dimensions, and therefore, it is not considered suitable for schema matching.

2.1.6 Matching Star Schemas

Matching relational schemas is a well researched area, a summary of different approaches is provided by Pavel and Euzenat [2004]. Many of the concepts used in matching relational schemas are also applicable to Star schemas, we are however, concerned with the research that exploits Star schema properties.

Li and Yang [2004] discuss matching Star schemas specifically. Their approach converts schemas to a binary schema tree with the fact table as the root of the tree and dimensions forming child paths to the root node. The proposed matching uses linguistic properties and fixed similarity values for different combinations of data types. It recognizes properties of multidimensional data but only as far as dimensions, dimension hierarchies and facts.

The matching algorithm used by Banek et al. [2008] recognizes the value in matching dimensions, facts, levels, measures and attributes. They do not describe the model they use and how the Star properties used in the matching process were obtained. Moreover, the matching process excludes properties such as keys, degenerate dimensions and degenerate facts. The distinction with their approach is said to be the treatment of aggregation hierarchies. To the best of our knowledge, none of the works above was available to be used in our evaluation.

2.1.7 Measuring the Quality of Match Results

Measuring the quality of matching in Information Retrieval is based on *Precision* and *Recall*. Precision (P) is the ratio of the number of correct matchings (C) over the number of suggested matchings (N). Recall (R) is the ratio of the number of correct matchings (C) over the number of expected matchings (M). A measure that combines these two is known as the *F-Measure* and calculated as follows:

$$\text{F-Measure} = \frac{2PR}{P+R} \tag{2.1}$$

Melnik et al. [2002b] calculate the quality of match results differently to the calculation of accuracy using F-Measure. They calculate accuracy in terms of the cost of modifying the proposed matching results to the expected match results. The cost is a ratio of the sum of the number of false positives ($N - C$) and false negatives ($M - C$) over the number of expected matchings. Subtracting this cost from 1 returns what they refer to as a measure of accuracy of match results. We refer to this as the *A-Measure*.

The value of A-Measure is always less than, or equal to F-Measure since it penalizes the measure of match quality for false positives and false negatives. It returns a negative value when the precision is less than 0.5, that is, more than half of the matches are wrong. This is where the cost of correcting the false positives and false negatives outweigh the cost of manual matching.

$$\text{A-Measure} = 1 - \frac{(N-C)+(M-C)}{M} = \frac{C}{M}\left(2 - \frac{N}{C}\right) = R \times \left(2 - \frac{1}{P}\right) \tag{2.2}$$

2.1.8 Schema Matching: Case Studies

In this section we look at two real life cases in the insurance industry, where there is a need to use an automated schema matching tool to perform the initial matching before the integration of data marts.

The first case concerns an insurance company which has over the years acquired other insurance companies offering similar products for cars, boats, home, third party, and life. Each of these companies has developed their own data marts.

Whilst, the integration of the source systems remains a long term objective, the short and medium term objective is however to be able to make a combined analysis of data across these data marts. What helps is that all of the acquired insurers have a similar business model, and the data marts cover similar subject areas such as Policy Renewals, Policy Cancellations, and New Policies.

A business analyst is able to identify data marts from similar subject areas. An automated schema matching can provide the first draft of the match between dimensions and measures. This process will save time and leaves the analyst with accepting or reject the matches.

The second case concerns a large insurance company with a large number of data marts extracted from a third normal form data warehouse. The company has decided that the existing strategy to transform data using ETL into a data warehouse and then into data marts is time consuming and expensive. Instead, they use high performance appliances based on massive parallel processing to create data marts directly from the source systems using database views.

Whilst, the existing data marts are yet to be replaced, there is a large overlap between the new and existing data marts. Again, to save time and cost, an automated schema matching process can be used to perform the initial matching followed by a review of the match results by the analyst.

2.2 Structure of Multidimensional Data

2.2.1 Dimension Hierarchies

There are many but consistent definitions of dimension hierarchy. Hurtado and Mendelzon [2001] consider a hierarchy as being a directed acyclic graph (DAG) of levels (each corresponding to a node of the graph) where each roll-up relationship is an edge connecting a pair of nodes.

A commonly understood definition is that, it contains several related levels, and that the relationships are used for *roll-up* and *drill-down* operations [Malinowski and Zimányi, 2006]. Dimension hierarchies are also critical to storage optimization, summarizability and accurate integration of multidimensional data. They can be also used as the basis for data clustering which can significantly improve performance of queries against data marts [Samet, 1990; IBM DB2, 2009]. Assuming two levels l_1 and l_2, we need to know the roll-up relationship between these two levels to establish if the data is summarizable as far as these two levels are concerned.

Dimension hierarchies are designed after comprehensive analysis of real life data in its fullness and in respect to its domain rules. They are then ideally defined as part of the schema and enforced by the database management system.

A dimension hierarchy can be mapped to ER (Entity Relationship) where each relationship can be shown using (0,M) on the (child) level and (1,1) at the other (parent) level [Malinowski and Zimányi, 2006]. Figure 2.3 shows ER representation of the levels of a hierarchy of a `Store` dimension.

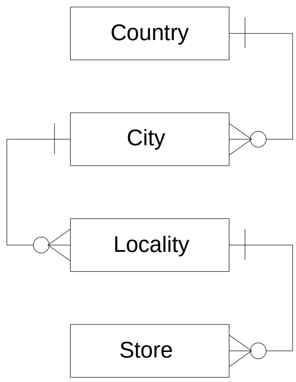


Figure 2.3: ER representation of a hierarchy.

Hacid and Sattler [1997] define a dimension hierarchy as a finite, partially ordered set $(H, \succ)$, where $\succ$ is a strict ordering and each level is a subset $l \in H$. Similarly, Torlone [2008] defines a dimension hierarchy as a partial order $\preceq$ on a finite set of levels L where each partial order relationship $l_i \preceq l_j$ corresponds to a roll-up.

Another way to describe hierarchies is to consider them as constraints applied against their instances. Hurtado and Mendelzon [2002] consider hierarchies as constraints that augment schemas to model dimension instances. They consider a dimension constraint as a Boolean combination of two sorts of atoms: *path atoms* and *equality atoms*. Path atoms start from a unique category called the root of the constraint. For example, if `Store` rolls up to `City` then the root of the `location` dimension is `Store` and the constraint consists of a single path atom `Store_City`.

There may be multiple hierarchies in a dimension in which case each hierarchy is a Direct Acyclic Graph (DAG) or a tree. Each hierarchy consists of a number of levels, and each level is associated with a category or grouping on which measures are aggregated. For example, levels of a location hierarchy are `Country`, `State` and `City`. The hierarchy is defined using partial order relationships (that is M:1) between levels, e.g. `Town` $\preceq$ `City` $\preceq$ `Country`. Each partial order relationship corresponds to a *roll-up*. The partial order relationships between levels determine the expected order in which measures are aggregated.

2.2.2 Summarizability

Earlier in this section, we said that the roll-up relationship between levels of the hierarchy is a M:1 relationship. According to Rafanelli et al. [1990], data is *summarizable* when every measure m at any level l_i can be computed by summing m at the level l_{i-1} where $l_{i-1} \preceq l_i$. It is a property of summarizable data Strictness preventing double counting. It requires that when aggregating at the higher levels, the tree structures implied by roll-up relationships are maintained in respect to the data. Where this holds true, the dimension is said to be *strict* [Rafanelli, 2003].

For example, given the hierarchy in Figure 2.3, the instance of the hierarchy shown in Figure 2.4 is not strict and, hence, the data is not summarizable. The sales amount in store `Store_B` is double counted as it is attributed to two cities. Rafanelli and Shoshani [1990] refer to this case as a multi-valued mapping. Non-strictness in dimensions may occur during updates and can be prevented using data integrity constraints [Hurtado et al., 1999].

It is possible that members of a parent level with the same label, refer to different

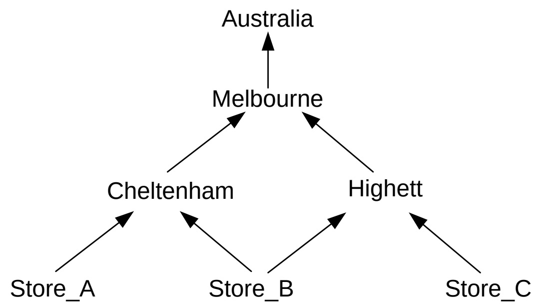


Figure 2.4: An example of a non-strict instance.

real world entities i.e. *homonym* members [1]. For example, the two cities `Melbourne` and `Melbourne` relate to two different states `Victoria` and `Florida`. Although this case does not result in double counting, it can lead to some other aggregation problems. For example, if we aggregate the sales amount based on `City` only, we will have an over-estimated measure of sales amount for an ambiguous city. Rafanelli and Shoshani [1990] refer to this as a special case of single-valued mapping. As a solution, they recommend that the member of the child level also includes the member at the parent level.

2.2.3 Levels

Torlone conceptually defines a dimension D as a finite set $L = \{l_1, ..., l_n\}$ of levels where each level is a distinct grouping of data [Torlone, 2008]. Levels are synonymous to *aggregation levels*. The partial order or roll-up relationships between levels determine the dimension hierarchies. The lowest level of the hierarchy, called the *base level*, determines the data at its finest granular level and is denoted by $\perp$. The top level denoted by $\top$ represents the highest level of grouping of data.

[1] The correct term is in fact *homograph*, we use *homonym* because it is more widely known as such.

2.2.4 Members

Members are distinct values in each level. In the context of the relational implementation, attribute values correspond to members of levels. Torlone [2008] classifies members into those that refer to real world entities belonging to the base levels, and the rest being groups of members. For example, in Figure 2.4, the member `Store_A` of the level `Store` refers to a distinct real world entity and the member `Melbourne` of the level `City` is a group of stores.

2.2.5 Dimension Table

Dimension table is the relational implementation of a dimension (as an abstract concept). It is a relation over a set of columns referred to as *dimension attributes*. A level in a dimension table is associated with a unique set of dimension attributes and determines a distinct grouping of data. Distinct values of dimension attributes correspond to members of the level to which the attributes belong to. Dimension hierarchies are defined over levels of a dimension table.

2.2.6 Data Mart

An instance of a Star schema is a data mart. Torlone [2008] formally defines data mart f over a set D of dimensions composed of a scheme (Star schema) and an instance of it. The scheme $f[A_i : l_1, ..., A_n : l_n] \rightarrow \langle M_1 : \tau_1, ..., M_m : \tau :_m \rangle$, where each A_i is a distinct attribute, l_i is a level of some dimension in D, each M_j is a distinct measure, and each τ_j is some base type.

An instance is defined as a partial function mapping coordinates for f to facts for F, where a coordinate is a tuple over the attributes of f mapping each attribute name A_i to a member of l_i, and a *fact* is a tuple over the measures of f mapping each measure m_i to a value in the domain of type τ_j.

2.3 Inferring Dimension Hierarchies

2.3.1 Schema Based Hierarchies

Schema Based Hierarchies are explicitly defined as part of the schema and enforced by the database management system, or defined and enforced as part of the application. The following is an example of how dimension hierarchies such as those of the `StoreDimension` are defined on an Oracle database [Oracle, 2005].

```
CREATE DIMENSION StoreDimension
   LEVEL Region   IS (Region_Name)
   LEVEL Country  IS (Country_Name)
   LEVEL City     IS (City_Name)
   LEVEL Locality IS (Locality_Name)
   LEVEL Store    IS (Store_Name)
   LEVEL Division IS (Division_Name)

   HIERARCHY StoreRollup_1 (
      Store         CHILD OF
      Locality      CHILD OF
      City          CHILD OF
      Country       CHILD OF
      Region)

   HIERARCHY StoreRollup_2 (
      Store         CHILD OF
      Division      CHILD OF
      Country       CHILD OF
      Region)
```

This is, however, a proprietary description of hierarchies provided by a specific vendor. Other database management systems either do not have a provision for describing hierarchies, or describe them differently.

2.3.2 Inferred Hierarchies

Schema based aggregation hierarchies may not be available in the following circumstances: (i) when accessing unfamiliar data sources from external sources, (ii) where the database management system does not support inclusion of the dimension hierarchy as part of the schema, (iii) no documentation is available to describe the hierarchy.

A dimension hierarchy that is inferred by any mean other than from the schema or the application, is an *inferred hierarchy*. In Chapter 3, we propose to infer aggregation hierarchies from their instances. In what follows next in this chapter, we describe the related work on this problem.

2.3.3 Inferring Functional Dependencies

In Section 2.2.1, we explained that aggregation hierarchies could be defined in terms of partial order relationship between levels. It happens that partial order corresponds to functional dependency. For example, in Figure 2.5, the roll-up $\texttt{Store} \preceq \texttt{Locality}$ corresponds to the functional dependency $\texttt{Store} \rightarrow \texttt{Locality}$. That is, `Store` determines the `Locality` and `Locality` is determined by `Store`.

Inferring functional dependencies has been studied by several authors [Carpineto et al., 2009; Kantola et al., 1992; Mannila and Räihä, 1994; Romero et al., 2009] against relational databases for a wide range of applications including data clustering, database design and query optimization.

Functional dependencies over a number of attributes is a set of unique combinations of those attributes on the left and right hand side of the dependency. As such, the complexity of the algorithms to infer these dependencies is significant.

Mannila and Räihä [1994] suggest several algorithms for inferring functional dependencies from example data in relational databases. Their improved sort-based algorithm has a complexity of $O(mnp\ log\ p\ + n2^{2n})$ where n is the number of attributes, p is the number of tuples, and m is the number of sorts.

2.3.4 Inferred Dimension Hierarchies

Jensen et al. [2004] discover multidimensional structures in relational databases using physical metadata from the schema. They also propose a method for discovery of dimension hierarchies. They consider attribute values as being single members of each level.

Akoka et al. [2001] derive dimension hierarchies from UML schemas. They do so by mapping M:1 (i.e. many-to-1) aggregation into a dimension hierarchy. This approach can work well only if the (UML) schema information on dimension hierarchies is available.

Romero et al. [2009] have suggested using conceptual representations of the domain ontology in discovering functional dependencies. This approach relies on analysis of assertions concerning functional dependencies in ontologies. It uses an ontology reasoner to discover the closure of asserted functional dependencies. Similar to the problem with unavailability of schema defined hierarchies, it is even less likely for domain ontologies to be available and, hence, the approach has limited application. Nevertheless, subject to the availability of the domain ontologies, accuracy and the completeness of the rules, the hierarchies inferred using this method can be closer to the intended hierarchies than using data which may not reflect the real world population of the dimension.

Mazón et al. [2006] explore semantic relations by using lexical repositories such as WordNet [Fellbaum et al., 2011] to identify levels of aggregations. WordNet is an electronic lexicon which captures hypernym (i.e. superordinate) and meronym (i.e. subordinate) relations for nouns. This work does not rely on schema information, however, the accuracy of derivation depends largely on whether the members can be successfully looked up in the lexicon. Unfortunately, in more cases than otherwise, members are codes and abbreviated names.

2.4 Matching Requirements for Integration of Dimensions

2.4.1 Conformed Dimensions

Several people have investigated the pre-integration requirements for dimensions. *Conformity* between dimension tables is a well known set of such requirements and is also widely used in the industry. Kimball and Ross [2002] define two dimension tables as being conformed if they have identical keys, and identical attributes, and that matching attributes must have identical values, or one must be a subset of the other. They also add that conformed dimension tables must mean the same thing. Similarly, Mundy et al. [2006] define two dimension tables as being conformed if they have the same name and contents. Giovinazzo [2000] requires the same instance of a conformed dimension table to be joined to multiple fact tables. These very similar descriptions of conformity concern relational implementation of dimensions and are fairly strict since their scope concerns all attributes of a dimension table.

2.4.2 Compatible Dimensions

Torlone [2008] acknowledges that Kimball's definition of conformed dimension tables is not suitable to autonomous data marts and defines dimension compatibility as an alternative. He offers a comprehensive analysis of the requirements for *compatible dimensions* in terms of three properties of the matching between dimensions, but only against their matching levels. These are *coherence*, *consistency* and *soundness* defined as constraints against matching levels of a pair of dimensions. In what follows next, we explain Torlone's requirements for compatible dimensions.

The mapping μ between the integrating dimensions (d_1 and d_2) is said to be *coherent* if every partial order between each pair of levels l and l' of d_1 also exists between those levels of d_2 that match with l and l'. This property ensures that the hierarchies involving matching levels in d_1 and d_2 are identical:

Definition 2.1. ***Coherence***: *the matching μ between dimensions d_1 and d_2 is coherent if, for each pair of levels l, l' of d_1 on which μ is defined, $l \preceq l'$ if and only if $\mu(l) \preceq \mu(l')$ [Torlone, 2008].*

Figure 2.5 shows the mapping between the matching levels of `Store` and `Shop` dimensions. For example, `Country` $\preceq$ `Region` appears in `Store` dimension only and `City` $\preceq$ `Division` appears in `Shop` dimension only. Also, the roll-up `Locality` $\preceq$ `Division` exists in `Shop` dimension but not in `Store` dimension. Therefore, the mapping between the two dimensions is not coherent.

The second property is *consistency* and involves hierarchies and their instances. This property requires that integrated instances conform to the original hierarchies.

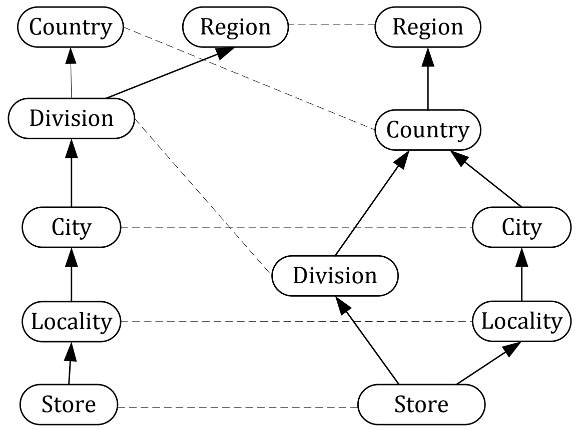
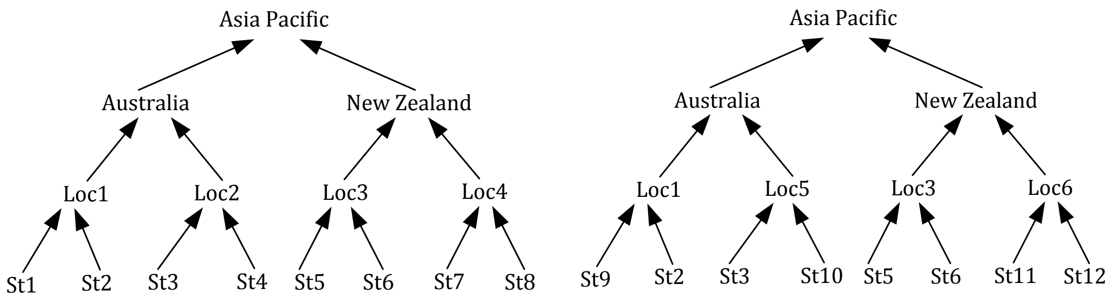


Figure 2.5: Incoherent mapping between `Shop` *and* `Store` *dimensions.*

Definition 2.2. ***Consistency***: *The matching μ is consistent if, for each pair of levels $l \preceq l'$ of d_1 on which μ is defined, $\rho_1^{l \to l'} = \rho_2^{\mu(l) \to \mu(l')}$ [Torlone, 2008].*

Figure 2.6 shows instances of the two `Store` and `Shop` dimensions using levels which make the matching between their hierarchies (`Store` $\equiv$ `Shop`) $\preceq$ (`Locality` $\equiv$ `Locality`) $\preceq$ (`Country` $\equiv$ `Country`) $\preceq$ (`Region` $\equiv$ `Region`) coherent. However, the matching between the `Store` and `Shop` dimensions is not consistent because the partial order relationship (`Store` $\equiv$ `Shop`) $\preceq$ (`Locality` $\equiv$ `Locality`) will not hold after the integration of their dimensions. The reason is that the member `St3` belonging to the level (`Store` $\equiv$ `Shop`) rolls up to two different localities `Loc2` and `Loc5`.

If the test for coherence fails, then there is no need to continue with the test for consistency. To pass the test for coherence, it is possible to exclude from the hierarchies (and consequently from the integration) those levels that are the cause of the incoherence. By removing `Division` and (`Country` or `Region`) the test for consistency would pass using the remaining levels and, hence, we could achieve partial integration.



(a) An instance of `Store` dimension (b) An instance of `Shop` dimension

Figure 2.6: Inconsistency between `Store` *and* `Shop` *dimensions.*

Failing to satisfy the requirements for consistency results in the violation of the M:1 relationships in the integrated result, as it no longer conforms to the hierarchy in the original dimensions. Therefore, consistency ensures the strictness (a property of summarizability) in the integrated data.

The third property is *soundness*. The matching between two dimensions is sound if for every member of every level in one dimension there is a matching member in the corresponding matching level of the matching dimension:

Definition 2.3. ***Soundness****: The matching μ is sound if, for each level $l \subseteq L_1$ on which μ is defined, every member of level l must have a matching member in the level with which l matches, that is $m_1(l) = m_2(\mu(l))$ [Torlone, 2008].*

In Figure 2.6, the matching between the two dimensions is not sound since there are mismatching members `Loc2, Loc3, Loc4, Loc5, Loc6` in `Locality`, and `St1, St4, St7, St8, St9, St10, St11, St12` in `Store/Shop` levels.

Definition 2.4. *μ is called a perfect matching if all of the matchings are coherent, sound and consistent [Torlone, 2008].*

Acknowledging that soundness is difficult to achieve in particular with heterogeneous dimensions, Torlone defines a less strict variation of perfect matching:

Definition 2.5. *Two dimensions d_1 and d_2 are μ-compatible if there exists two lossless expressions E_1 and E_2 such that μ is perfect matching between $E_1(d_1)$ and $E_2(d_2)$ [Torlone, 2008].*

For brevity, this thesis will use the term 'compatible' to refer to 'μ-compatible'.

Although Torlone's definitions of compatibility is at a conceptual level, and conformity concerns relational implementation of dimensions, we can say that compatibility is less strict than conformity, because (i) it is not applied against all levels of dimensions, and (ii) if there are some lossless expressions applied against dimensions, then the soundness is only applied against a subset of members. Therefore, every pair of conformed dimension tables is also compatible but the reverse may not be true.

A similar set of requirements to compatibility by Grossmann and Moshner [2007] verify for three types of conflicts due to: (i) mismatch in one-to-many relationships, (ii) mismatches in hierarchies caused by coarse requirements, and (iii) conflicts according to non-corresponding value domains. These are, in effect, very similar to the requirements for compatible dimensions.

2.5 Instance Matching and Enforcing Strictness

2.5.1 Duplicate Detection

Generally, the literature classifies the instance or data matching problem as *duplicate detection*. That is, finding two or more rows in the same table that are likely to refer to the same entity. We divide the approaches for solving this problem into hierarchical and non-hierarchical. Approaches taken for non-hierarchical data exploit string similarity, supervised and unsupervised learning, and data clustering [Elmagarmid et al., 2007]. Hierarchy based approaches also look at the relationship between attributes, and between tuples (i.e. the hierarchies).

Dimensional data conform to their hierarchy and are, hence, classified as hierarchical data. Hierarchies can be effectively used in instance matching. For example, matching child members are expected to have matching parents, and if child members mostly mismatch, then their parents must be also different. As such, we are interested in methods that exploit the hierarchical structure of data. In the remainder of this section, we review the literature related to duplicate detection in hierarchical data and enforcing strictness.

The first work that exploited hierarchies in detecting duplicates in hierarchical data was DELPHI [Ananthakrishna et al., 2002]. DELPHI uses a measure called *Foreign Key Containment Metric* (fkcm) which measures the similarity between the children of two parent objects. The problem addressed by the authors is detecting duplicate rows in dimension tables with foreign key constraint between them (i.e. snowflaked dimensions). The main issue with DELPHI as noted by Weis and Naumann [2005] is that the similarity measures are not symmetric, that is member m may be a match to m' but the reverse may not be true.

Inspired by DELPHI, Weis and Naumann [2002] use a similar intuition. They measure the similarity between XML elements using the similarity between parent members, the similarity of data, and the hierarchy of their children. The similarity measure between two sets of tokens is the Inverse Document Frequency (IDF) value of their common tokens to the

IDF value of all non-common tokens. Similar to Ananthakrishna et al. [2002], their work also follows a top down approach.

Weis and Naumann [2004] later extended their work by development of Dogmatix [Weis and Naumann, 2005], which provides a framework for detecting duplicate entities in XML documents. There are two main heuristics used in the pair wise comparison of objects. These are *r-distant ancestors* and *r-distant descendants*. They are based on the intuition that the closer the information are to the element, the more related they are. For example, the city `'LA'` is more related to the state `'CAL'` than the suburb `'Beverly Hills'`. All elements whose depth in the document is within a given radius are used in creating the description of the object. In this respect Dogmatix benefits from similarities at the child as well as at the parent levels.

Calado et al. [2010] review recent approaches in XML duplicate detection. The review makes two important findings: (i) Dogmatix is a more effective method; (ii) missing data affects similarity measures more than typographical errors and duplicate erroneous data. The algorithms used in the above works are not designed to identify matchings between two instances.

Milano et al. [2006] and Kailing et al. [2004] use the structure of XML documents to calculate *tree edit distance*. The algorithm suggested by Milano et al. [2006] attempts to find similarity between tree structures by finding the most optimal overlays.

Other hierarchy based approaches include additional instance matcher added to COMA++ [Engmann and Massmann, 2007] for matching ontologies. It does not, however, provide for defining relationship between matching levels and, hence, it matches members from mismatching levels, and therefore, increases the false positives.

Similarity flooding has been used to identify similar structures and concepts. Marshall et al. [2006] use this algorithm for matching concept maps. They have introduced a "node anchoring" mechanism by which key terms and common abbreviations are identified as anchor points. Whenever these terms are found, they are locked as best matches. Pan et al. [2010] use key terms and abbreviations in place of string matching to improve the recall.

2.5.2 Consistent Query Answering

Where integrity constraints are violated (due to non-strict data), the database instance is said to be *inconsistent* as query results will not be consistent with the constraints. This problem was first investigated by Arenas et al. [1999].

Consistent query answering aims to return consistent answers where the data itself is not. This is usually done by obtaining a *minimal repair*, that is a variation of the instance that satisfies the integrity constraints and its variation is minimal relative to other repairs. This problem differs to the problem we are concerned with in the sense that it deals with inconsistencies within a single source.

Various approaches have been suggested to obtain consistent query results. They include deletion of inconsistent tuples, computing all possible repairs [Wijsen, 2006], returning a range of values for each set of consistent data [Arenas et al., 2001; Betrossi et al., 2009; Sismanis et al., 2009], and nullifying inconsistent data at the attribute level [Chomicki, 2006; Liu et al., 2008].

2.5.3 Enforcing Strictness

A different approach to deal with inconsistent data is to view it as a summarizability issue resulting from non-strictness [Horner and Song, 2005; Horner et al., 2004]. These approaches are concerned with modifying the instance such that it becomes consistent with its hierarchy. These approaches are, however, concerned with resolving non-strictness inherent within a single source. They cannot, however, guarantee consistency after the integration. In the following we briefly review the major works in this area.

Considerable research has been invested into solving this issue in the past decade. Pedersen et al. [1999] suggested creating a new parent by fusing the old parent values and linking it to what the old parents linked to, but cut the old members from their original parents.

Bertossi et al. [2009] create a canonical dimension that isolates inconsistent members. In the canonical dimension, inconsistent parent members are fused together and a range of values for fused members are returned.

A different approach by Luján-Mora et al. [2001] suggests reducing inconsistency by resolving string differences resulting from the use of different case letters, omission and inclusion of accents which can be useful before the matching.

Mazón et al. [2009] provide a comprehensive summary of the approaches to deal with non-strictness and other causes of unsummarizable data. These approaches are only effective if we have prior knowledge of intended match results.

To prevent inconsistency, specific operators such as *addInstance* are introduced to ensure that the added element reaches the same element in the levels above it [Hurtado et al., 1999;

Letz et al., 2002].

Another approach to resolve inconsistency in a single source is to somehow merge the tuples involved. However, this approach assumes that the match is correct and inconsistent tuples are in fact the same which may not be the case. For example, resolution strategy by Anokhin and Morto [2001] suggests reducing inconsistent tuples into one by choosing each attribute value from one of the merging tuples based on some predetermined rules. A similar method used by Wisjen et al. [2006] chooses the more up to date tuple. Greco and Molinaro [2008] suggest updating the inconsistent members with a domain (for an unknown set of values) which contains the inconsistent members.

2.5.4 Resolving Inconsistency Across Multiple Sources

The problem of resolving inconsistencies across multiple sources has received little attention. Agrawal et al. [1995] investigate integration of multiple databases that may be mutually inconsistent. Their approach is not concerned with discovery of the actual inconsistencies but rather extends each relation by attributes that would allow operations such as merge, equivalence, selection, and union to resolve the inconsistencies. Extended relations enable the definition of integrated Views over relations that may be inconsistent. They provide a model and a set of operations that enable resolve inconsistencies.

Another notable work is by Torlone [2008]. The author exploits hierarchies to recover a null parent member in one source by looking at the parent of the same member in another source. For any pair of matching child members where one of the parents member is null, Torlone proposes to replace the null parent member with the parent of the corresponding child member. The main concern with this approach is that it assumes that the match between the child members is a true positive.

2.6 Extending the Scope of Integration

Earlier in this chapter, we described conformity and compatibility between dimensions for accurate integration and maintaining summarizability. We also said that compatibility was less strict than conformity and therefore, it allows for greater segments of data to be integrated. The main operation used for integrating data marts is drill-across. In this section, we review the literature in respect to extending the drill-across operation to maximize the scope of the integration. We also review the related work on visualization of integrated multidimensional data from multiple data cubes or marts.

Drill-across is an OLAP operation used to navigate from an origin data cube to a target data cube using some common coordinates (in other words compatible dimensions). Cabibbo and Torlone [2004] define drill-across as an extension to the natural join where the intersection of the two dimensions is aggregated at the finest grain of the dimensions.

Abello et al. [2002; 2003] define the operation *drill-across* as changing subject (facts) in the same analysis space. They also find the strict requirement of dimension conformity to be restrictive for the purpose of drill-across. This operation, according to them, requires that selected instances in dimensions of the origin source to determine instances in the dimensions of the target source, and that the domains are related in some way. They have identified semantic relationships: *Derivation*, *Generalization*, *Association* and *Flow* to extend possibilities to perform drill-across by using these operations to map what would be otherwise mismatching members and, thereby, improve soundness. These relationships can be compared with lossless expressions used by Torlone [2008] for defining compatible dimensions. In Chapter 6, we further relax the requirements for compatible dimensions to allow for greater scope of integration.

In the following, we briefly review major works on visualization of multiple data cubes. Various methods have been proposed to visualize multiple but related data cubes and provide flexible representation of dimension structure using graphical representations of data.

Vinnik and Mansmann [2006] show a tree like structure of dimensions as an effective method in visualizing dimension structures. Polaris [Stolte et al., 2008] achieves visualization of multiple data cubes by partitioning data into groups and allocating them into independent panes.

ADVIZOR introduced by Eick [2000] uses a set of linked views displayed on the same screen where each view is used to show a number of measures. There is, however, no evidence that ADVIZOR can visualize multiple related data cubes.

Visual Pivot, introduced by Conklin et al. [2002], aims at visualization of data structures composed of multiple intersecting hierarchies called *Polyarchies* sharing at least one node. The aim of Visual Pivot is to track similar information in multiple hierarchies.

CoDecide introduced by Gebhardt et al. [1998] is an OLAP data visualization tool that enables multiple users to have different views of one or multiple data cubes, and allows users participate in a cooperative analysis of the subject.

2.7 Drill Across and Data Visualization: Case Study

Case studies described in Section 2.1.8 require integration of data marts which may share only some dimensions and measures. For example, one may be concerned with car insurance policies and another with home. They both include common dimensions such as policy number, inception date, due date, payment frequency, and common measures such as number of risks, and premium amount. Whilst the user can query each one of the data marts separately through a visualization device that supports ROLAP data cubes, the user needs to know how many risks come up for renewal on any day regardless of whether the type of the risk being insured is a car or a home.

From those risks that are up for a renewal for a given day, users require to re-group the home policies by construction age and the car policies by driver age. Naturally, these dimensions are only exclusive to the original data marts. The conventional approach is to navigate back to each one of the original data marts and re-apply the same constraints to the dimensions to obtain the remaining exclusive information. This approach is cumbersome as the user will have to repeat the same process each time a new renewal date is chosen.

Figure 2.7 shows two data cubes sharing two dimensions `Renewal Date and Payment Frequency`, and exclusive dimensions `Construction Age and Driver Age`. The drill-across forms a cartesian join of the two cubes which needs to be related back to the dimensions that are not common in the original data cubes.

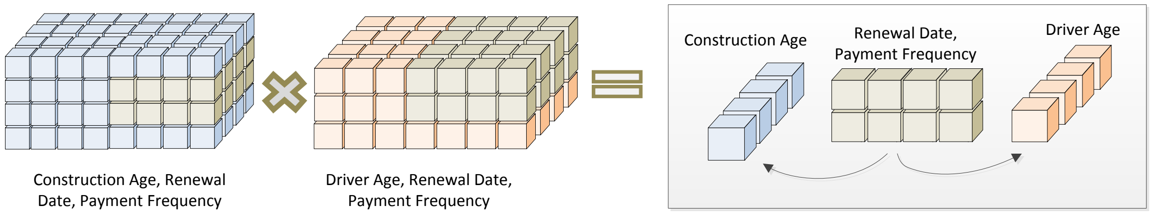


Figure 2.7: Extending the drill across to exclusive dimensions.

2.8 Discussion

In this chapter, we described data warehousing concepts to which we refer to in this thesis, and discussed the literature relevant to four related areas namely schema matching, inferring aggregation hierarchies, instance matching, and extending the scope of integration.

Schema matching: We described several existing approaches for discovery and representation multidimensional structures from various sources such as XML, DDL, DTD, UML, and ER diagrams. The main shortcomings of these approaches which we aim to address in Chapter 3 are that they omit some important properties of Star schemas and that the information they use to infer Star schemas from may not be available.

We discussed several approaches for matching data warehouse schemas. Whilst, these approaches benefit from some properties of multidimensional structures, they fall short of taking advantage of some of their important physical properties. They also do not follow any specific model and do not demonstrate how their approach compares with others.

Inferring aggregation hierarchies: We highlighted the absence of aggregation hierarchies as the main motivation to infer them, and described several approaches in the literature on inferring hierarchies. These approaches use various sources of information including metadata, UML schemas, domain ontologies and lexical repositories. One approach inferred hierarchies from data but it also used SQL queries and did not eliminate transitive relationships.

The main problem with using metadata, domain ontologies and lexical repositories is that these they cannot prevent a false positive case. That is, where the relation between two levels is M:1, these approaches may return what is in fact a M:M (i.e. many-to-many), because the information they use may be insufficient and imprecise. This problem cannot occur when using data. Moreover, information such as UML schemas and ontologies may not be available. Since instances of dimensions need to conform to their hierarchies, they can provide a solid clue to their hierarchies. For this reason, in Chapter 4, we use data to infer aggregation hierarchies.

Instance matching and enforcing strictness: We explained the significance of exploiting the hierarchical structure of multidimensional data and focused on methods for instance matching. Except for DELPHI which is aimed at relational data, the majority of such approaches are in fact aimed at matching XML data which is highly structured and hierarchical. To the best of our knowledge there is no algorithm specifically for matching multidimensional data. However, with some difficulty this problem can be mapped to the problem of duplicate detection in XML data. Unfortunately, implementations of these approaches were not available to us to compare their effectiveness.

Similarity flooding is an approach that we use to perform schema matching (discussed in Chapter 3) but, as we will demonstrate in Chapter 5, is even more effective and suitable for matching multidimensional data.

One side effect of using algorithms which exploit hierarchies is that the number of false positives can increase when there is a high volume of missing data in either or both instances. This problem highlighted by Calado et al. [2010] was also experienced during our experiments with similarity flooding described in Chapter 5.

Extending The Scope of Integration: We discussed the requirements for integration of dimensions and that compatibility was less strict than conformity. Although, the definition for compatible dimensions does take into consideration the use of lossless expressions to extend the scope of the integration, assuming a single pair of such expressions is also restrictive. In Chapter 6, we further relax the requirements for soundness by considering multiple expressions applied to different fragments of data. What is is useful to discovery of expressions is to identify which fragments of data, and to what extent is the problem of mismatching members. We address this problem when we extend the scope of integration in Chapter 6 by providing several metrics to measure the extent of mismatching members.

Another notable work on extending the integration is by Abello et al. [2002; 2003] who propose certain operations that would improve the match between members. These operations play a similar role to lossless expression. A shortcoming of the drill-across is that it makes no provision to link the common data to the non-conformed or non-compatible dimensions.

We reviewed some of the main advances made in visualization of multiple data cubes. What is however lacking is that, where coordinates of data change in one analysis space, the same to also occur in related spaces of data sharing some common dimensions. In Chapter 6, we discuss a conceptual representation of data from multiple data marts which also includes the related non-compatible dimensions.

Chapter 3

Matching Star Schemas

> ”We are all in the gutter, but some of us are looking at the stars.”
>
> Oscar Wilde (1854 - 1900)

Star schema is a relational model, and therefore, existing approaches for matching relational schemas also apply to Star schemas. As we discussed in Chapter 2, Star schemas have distinct and predictable properties and relationships. Consequently, a customized and more precise representation of the relational model that is designed to include specific properties and relationships of Star schemas could be considered.

In this chapter, we propose StarMod as a more precise representation of Star schemas. The immediate benefit of StarMod is that Star schemas described using StarMod would be more expressive and would be defined consistently. But, more importantly we demonstrate that when compared to using the relational model, using StarMod improves the quality of match results between Star schemas. We also demonstrate that StarMod can be also effective for matching arbitrary relational schemas.

Our representation of Star schemas covers similar concepts to those suggested in the literature but, it includes additional properties to improve their match results and expressiveness. Star schemas described as instances of StarMod (i.e. those that use StarMod properties for their representation) will, therefore, be standardized, and can be used in an automated process. We also present a set of heuristics to infer instances of StarMod (used in the matching process) from XML schemas corresponding to their relational schemas.

In our evaluation, we use Similarity Flooding (SF) [Melnik et al., 2002b] to perform matching between a number of Star and non-star schemas described using the relational model and StarMod, and compare their results. All results are then compared against those obtained from COMA++ [Do and Rahm, 2002] for the same schemas.

3.1 Motivations for Automated Matching of Star Schemas

Schema matching is a prerequisite to the integration of data marts. Automation of this process is critical to reducing reliance on database experts and making the integration process fast and cost effective. This problem is more acute for unseen heterogeneous data marts.

Despite attempts to provide industrial strength solutions to automate the matching of relational schemas [Bernstein et al., 2004; Haas et al., 2005], most organizations prefer to employ experts to perform the matching and integration of their relational databases. This is tolerated as organizational changes that necessitate the integration of their data sources do not occur frequently. Moreover, the integrated schema must be designed in such a way that it is flexible enough to accommodate future business requirements and supports transaction processing operations.

However, there are different considerations in a business intelligence environment where the emphasis is on the agility of the integration process as there is always a need to integrate data from multiple sources at short notice. In many cases, the resulting schema from the integration has a short life span.

A common application of a semi-automated Star schema matching is where a data analyst wishes to have a combined view of data marts for related subject areas. Other motivating scenarios are:

- where a business enterprise with disparate data warehouses for different lines of business such as banking, insurance and wealth needs to combine some of its data marts around customer and address information for marketing campaigns; and
- the need to integrate local data marts with externally sourced data marts such as those sourced from Bureau of Statistics and market surveys.

Furthermore, organizations with large data warehouses develop many data marts overtime, many of which overlap [Business Objects and Teradata, 2007]. Matching and integration of these data marts using a semi-automated process guided by a domain analyst will improve the agility and reduces costs for providing new reports from combining existing data marts.

Star schemas conforming to principles specified by Kimball and Ross [2002] have a certain topology and, hence, are more restricted than arbitrary relational schemas which makes them to be more predictable. This allows us to anticipate their model and identify their distinct properties.

Schema matching relies on the classification of database properties such as tables, columns, data types, integrity constraints, keys and even data. Intuitively, the specialization of these properties can improve the quality of the matching process. Incidentally, relational properties in Star schemas can be defined more granularly. Generic relational properties such as *table* and *column* fail to describe properties of Star schemas precisely. This motivates us to consider whether a more precise representation of Star schemas can improve the matching for Star schemas.

3.2 Why StarMod?

The motivation for StarMod is multi-fold and related:

- to provide a more expressive, precise, rich and consistent description of multidimensional data,
- to use the schemas in an automated matching process, and
- to improve schema matching results.

It will also make it easier for data modelers to standardize schemas by following what is, essentially, a template for describing these schemas. By exposing the properties of Star schemas through StarMod, designers can potentially improve the model by reducing or eliminating degenerate dimensions and degenerate facts which can also improve the quality of match results because they are no longer incorrectly matched with facts and dimensions.

Given that Star schema model is a specialized form of relational model, we define property classes in StarMod as subclasses of three main classes Table, Column and Key in the relational model. This allows simplification of the StarMod since properties such as name, data type, length, etc. can be inherited from the relational model. Figure 3.1 shows the UML representation of the relational model corresponding to `Relational.owl`, an OWL representation of relational model by de Laborda and Conrad [2005]. The highlighted classes are added by the author of this thesis.

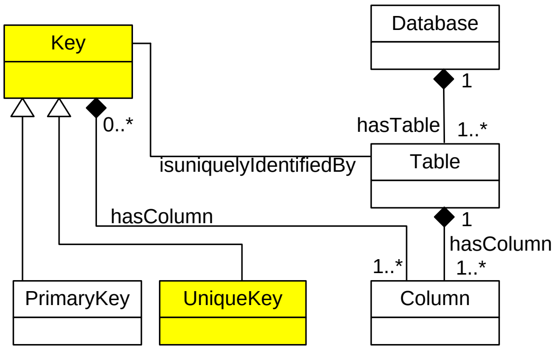


Figure 3.1: UML diagram for relational model.

3.3 StarMod Properties and their Application in Schema Matching

StarMod includes a predefined set of concepts, relationships and constraints for describing multidimensional data modelled on Star schema. It can be used as a template to define Star schemas for any domain. In this section, we describe StarMod properties corresponding to the classes shown in UML presentation of StarMod in Figure 3.2, and their applications in matching Star schemas.

A **Star** represents a single Star schema and is defined as an aggregation of a number of dimension and fact tables. Each *Star* may have none, or more dimension tables but must have precisely one fact table. Conversely, a fact table may belong to one Star only. Dimension tables may be in one or more Stars. These rules, together with the fact that *Elements* of *FactTables* may refer to *Attributes* of *DimensionTables* only, ensures that *Stars* are only constellated through the shared or conformed dimension tables [Giovinazzo, 2000]. Each dimension table is a subtype of a relational table and is an aggregate of one or more *Attributes* being a subtype of the relational column. Dimension and fact tables are related through keys. Making distinction between dimension and fact tables improves the similarity between matching dimension and fact tables as well as their respective columns.

There are three types of *Attributes*:

i. A **SurrogatekeyAttribute** is a unique sequential number that is generated for every new row. Use of surrogate primary keys is very common in dimension tables [Imhoff et al., 2003]. A surrogate key value is generated at the time the row is inserted. Its definition is vendor specific, for example in an IBM DB2 database, the following statement is used to define a *Surrogate key* to have its value starting from 1 and incremented by one each time a new row is inserted:

```
ALTER COLUMN "DEALER_KEY" GENERATED ALWAYS AS IDENTITY START WITH 1, INCREMENT BY 1
```

It is important to distinguish surrogate keys from other types of attributes, as they do not participate in any aggregation function, and their classification, as such, reduces their similarity with natural keys and other attributes. For example, `customer_key` may be a customer number in one dimension and a surrogate key in another.

ii. A **DegenerateFact** is an additive attribute to which data functions such as *Sum* or *Average* can be applied. For example, `Number_of_Claims` as an attribute of dimension table `Claimant` is classified as a *DegenerateFact*, whereas, `Claim_Number` as an attribute of dimension table `Claim` is not, even though they have identical data types and similar labels. Degenerate facts are matched with *measures* and degenerate facts. They are, however, rare and are not considered part of the model introduced for Star schemas by Kimball and Ross [2002], and as such are not recommended. They are best placed in fact tables and classified as measures.

iii. **DataAttributes** are generally used as categories by which measures are summarized. Their classification as such allows them to be matched with data attributes and degenerate dimensions (described later in this section).

A **FactTable** is also a subtype of *Table* and is an aggregate of one or more *Elements* being subtypes of *Column*. There are four types of *Elements*:

i. **SurrogateKeyReference** is a foreign key that refers to a *SurrogateKeyAttribute* of a dimension table. By classifying surrogate key references as such, they are better distinguished from surrogate keys in dimension tables.

ii. The role of **SurrogateKeyElement** in a fact table is similar to that of *SurrogateKeyAttribute* in a dimension table.

iii. A **Measure** is an additive (or quantitative) element such as `Number_of_Claims` in fact tables [Giovinazzo, 2000]. Additive elements are those to which aggregate functions such as `Sum` or `Avg` can be applied. Classifying elements as measures allows them to have stronger matching with measures than with other properties.

iv. A **DegenerateDimension** is a dimension attribute for which no dimension exists [Imhoff et al., 2003]. Use of degenerate dimensions is common in Star schemas. By classifying degenerate dimensions as such, we are able to separate them from measures and increase their similarity with other data attributes. For example, `Account_Number` in a fact table is classified as a degenerate dimension and is distinguished from a measure such as `Number_of_Accounts`.

Hierarchy and Level: Each dimension *hierarchy* consists of a set of *levels* between which there exists a partial order named *rollsUpTo*. For example, given levels l_1 and l_2 where $l_1 \preceq l_2$, we say that l_1 *rollsUpTo* l_2. Each *level* has one or more *DataAttributes* that uniquely identify it. Aggregation of measures occurs alongside levels of hierarchies.

We represent hierarchies as part of StarMod for completeness of the representation, but do not use them in the schema matching process for several reasons:

- Matching attributes may belong to mismatching hierarchies. Rejecting these matchings as early as the schema matching can be restrictive.
- In many cases, as we will see in Chapters 5 and 6, it is possible to enforce coherence between hierarchies by applying some constraints against the data.
- Even if hierarchies are found to be matching, the data after integration may not conform to the original hierarchies [Torlone, 2008], that is, the matching is not consistent. This is discussed in Chapter 5.

- As discussed in Section 2.3.2 dimension hierarchies are not always readily available and, where inferred, may not be the same as their intended hierarchies. Inference of hierarchies is discussed in Chapter 4.

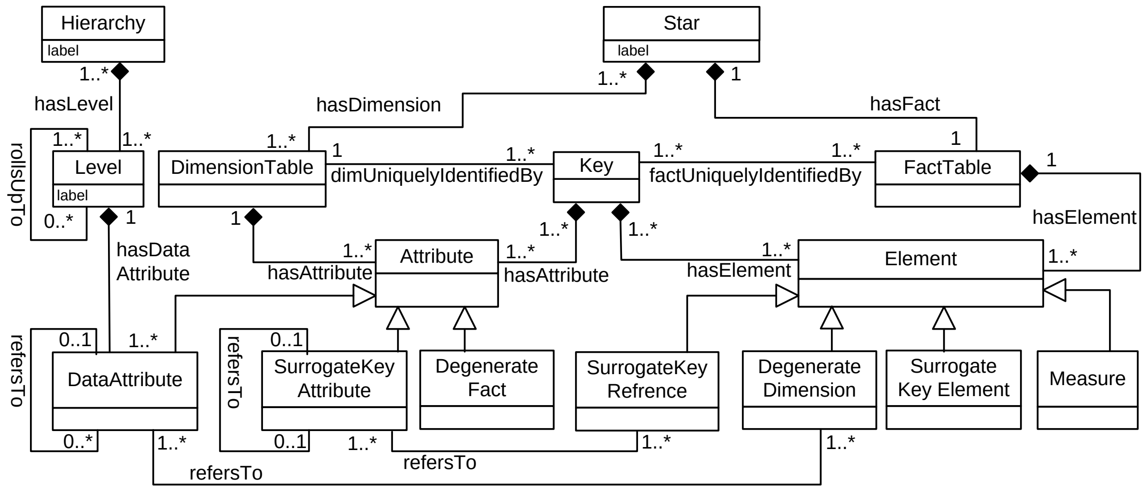


Figure 3.2: UML representation of StarMod.

3.3.1 OWL Description of StarMod

We use UML (Unified Modeling Language) to provide a visual representation of StarMod and implement it using RDFS (Resource Description Framework Specification), and OWL (Web Ontology Language) with their constructs corresponding to the UML representation.

An instance of StarMod for a given domain could be also described using UML. However, it is necessary to implement it in some language for the purpose of automated schema matching. We have defined a corresponding version of StarMod in OWL which allows us to define its instances also in OWL. The properties in the OWL description are extensions of `Relational.owl` introduced by de Laborda and Conrad 2005.

Figures 3.3 and 3.4 show the specialization of relational properties into StarMod properties. Highlighted in gray are the relational properties from `Relational.owl`.

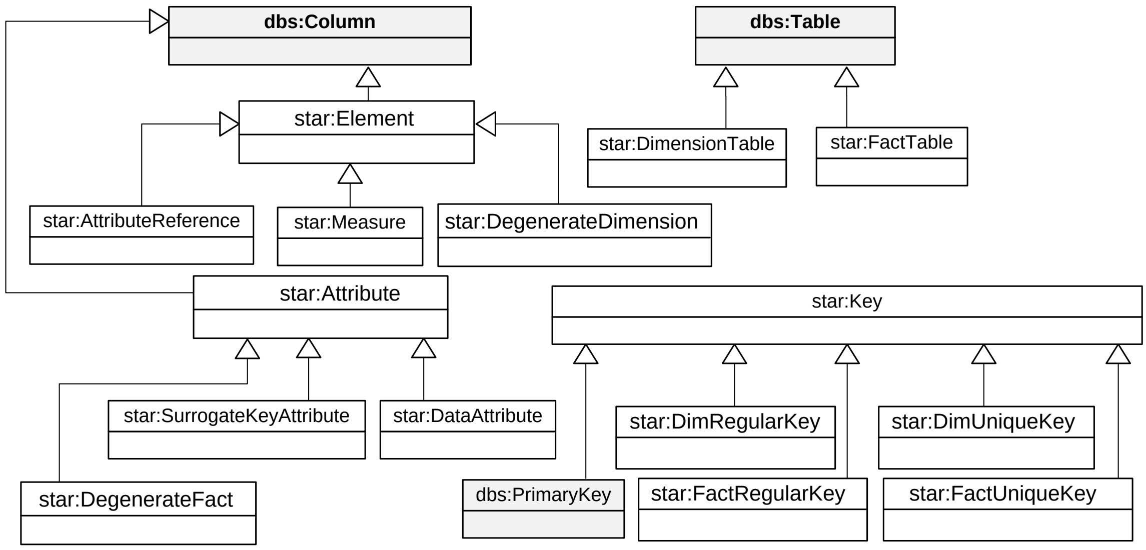


Figure 3.3: Specialization of relational objects in StarMod.

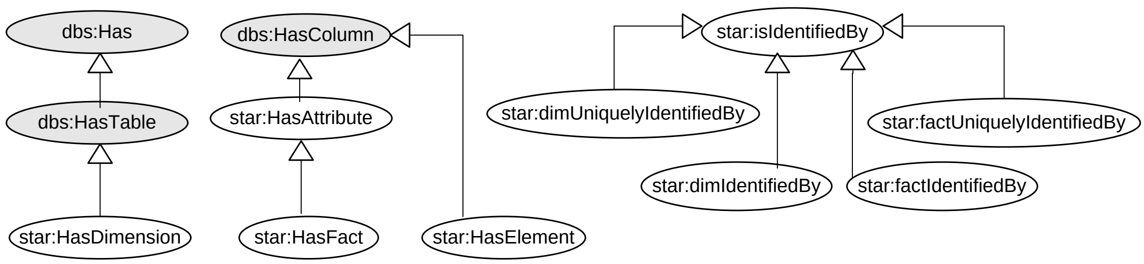


Figure 3.4: Specialization of relationships in relational model in StarMod.

Although, the potentials of an OWL representation are realized when used for its reasoning capability, there are several reasons why the use of OWL is advantageous to writing code in a language such as Java as proposed by Luján-Mora et al. [2006]:

- UML has a formal equivalence with OWL [W3C, 2009].
- UML is also suggested for visual representation of OWL ontologies [Brockmans et al.,

2004].

- There are readily available tools that can convert UML to OWL [Eclipse, 2010].
- RDF statements can be easily extracted from OWL for machine processing.
- The OWL version provides the flexibility for the model to include additional semantic information such as domain ontologies.

The following shows a fragment of StarMod. A complete listing of StarMod is presented in Appendix D.

```
<owl:Class rdf:ID="Star">
  <rdfs:subClassOf rdf:resource="&rdf;Bag"/>
  <rdfs:label xml:lang="en">Star Schema</rdfs:label>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasDimension">
  <rdfs:label xml:lang="en">Has Dimension</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="#has"/>
  <rdfs:domain rdf:resource="#Star"/>
  <rdfs:range rdf:resource="#DimensionTable"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="DimensionTable">
  <rdfs:subClassOf rdf:resource="&rdf;Seq"/>
  <rdfs:label xml:lang="en">Dimension Table</rdfs:label>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasAttribute">
  <rdfs:label xml:lang="en">Has Attribute</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="#has"/>
```

```
  <rdfs:domain rdf:resource="#DimensionTable"/>
  <rdfs:range rdf:resource="#Attribute"/>
</owl:ObjectProperty>
```

3.3.2 Inferring StarMod Properties from Relational Schemas

In this section, we describe a set of heuristics that we use to infer Star properties described earlier in this section. These heuristics use a combination of data types, keys (which also include indexes), and foreign keys. The result is an OWL description of the schema that conforms to StarMod.

FactTable and DimensionTable: Similar to Golfarelli et al. [2001] where xsd:key and xsd: keyref elements are used to establish dimension hierarchies, we say that if the primary key of a table does not appear as foreign key in any table, then we classify the table as a *FactTable*, otherwise it is classified as a *DimensionTable*.

SurrogateKeyAttribute: A column whose owning table is a *DimensionTable* is classified as a *SurrogateKeyAttribute*, if it is defined as a primary key and has a constraint for having its value generated.

DegenerateFact: Identifying degenerate facts is a difficult task as it requires domain information on the meaning and purpose of the column. In absence of such information, we safely classify a column as a *DegenerateFact* only if its owning table is classified as a *DimensionTable* and its data type is *xsd:decimal* with a restriction defined as *xsd:fractionDigits*.

DataAttribute: A column whose owning table is a *DimensionTable* and is not a *SurrogateKeyAttribute* or *DegenerateFact* is classified as *DataAttribute*.

SurrogateKeyElement: A column whose owning table is a *FactTable* is classified as a

SurrogateKeyElement, if it is defined as part of a key, and has a constraint for having its value generated (similar to the *SurrogateKeyAttribute*).

SurrogateKeyReference: A column whose owning table is a *FactTable* is classified as a *SurrogateKeyReference*, if it refers to a *SurrogateKeyAttribute*.

DegenerateDimension: Similar to *DegenerateFacts*, accurate classification of degenerate dimensions requires additional semantic information. We safely classify a column as a *DegenerateDimension*, if its owning table is a *FactTable*, and one of the following is true: i) its data type is *xsd:string*, ii) its data type is *xsd:integer* or *xsd:short* or *xsd:decimal*, and is defined as part of a key or a foreign key. A miss-classification is possible where, for example, a column such as `POSTCODE` defined as a *xsd:short* is not part of a key or foreign key, which is still a degenerate dimension, but by default is classified as a *measure* (described next). This can result in a mismatch if one column is included in the key or the foreign key but its matching column is not. Although degenerate dimensions are used frequently, it is considered to be a good practice to minimize their use and define them as string.

Measure: A column whose owning table is a *FactTable* is classified a *Measure*, if one of the following is true: i) it has a data type *xsd:decimal* with the restriction *xsd:fractionDigits*, ii) it has one of the data types *xsd:short*, *xsd:integer* or *xsd:decimal* and is not defined as part of a key or a foreign key. Miss-classification of measures is possible but to a lesser extent, e.g. `no_of_cylinders` which does not appear in a key or foreign key and has a data type *xsd:short* is not a measure. Presence of such elements highlight a design shortcoming where non-metric data items are more accurately defined as string.

Key: Similar to de Laborda [2005], every *attribute* or *element* that appears in a key is also made an *attribute* or *element* of a *Key* for the respective dimension or fact table.

Hierarchies: As described in Section 2.3.4, dimension hierarchies are not always available and may need to be inferred. For reasons described earlier in Section 3.3, they are not used in the matching process.

Snowflaked dimensions exist where a column is classified as *DataAttribute* or *SurrogateKeyAttribute* and refers to a *DataAttribute* or *SurrogateKeyAttribute*. The relationship *refersTo* between surrogate key attributes, and between data attributes indicate that their dimension tables are snowflaked.

3.3.3 Implementation of StarMod and Matching of Star Schemas

The input to the process of inferring StarMod from relational schemas is XML schema obtained from an existing relational databases through a tool such as IBM WebSphere [IBM, 2012]. Programs written in XSLT language use heuristics described in Section 3.3.2, to transform the XML schemas (corresponding to the relational schema) to instances of StarMod and Relational.owl described in OWL. Additional XSLT programs are used to transform the OWL predicates into RDF statements which are then directly used by the similarity flooding algorithm. Additional programs were written in Java to fully automate the matching process. Figure 3.5 shows the sequence of processes starting with acquiring schemas to the matching process.

Figure 3.6 shows a Star schema which we will use in as one of the two schemas for our discussion on schema matching.

The following shows an OWL fragment of a StarMod instance inferred using the rules

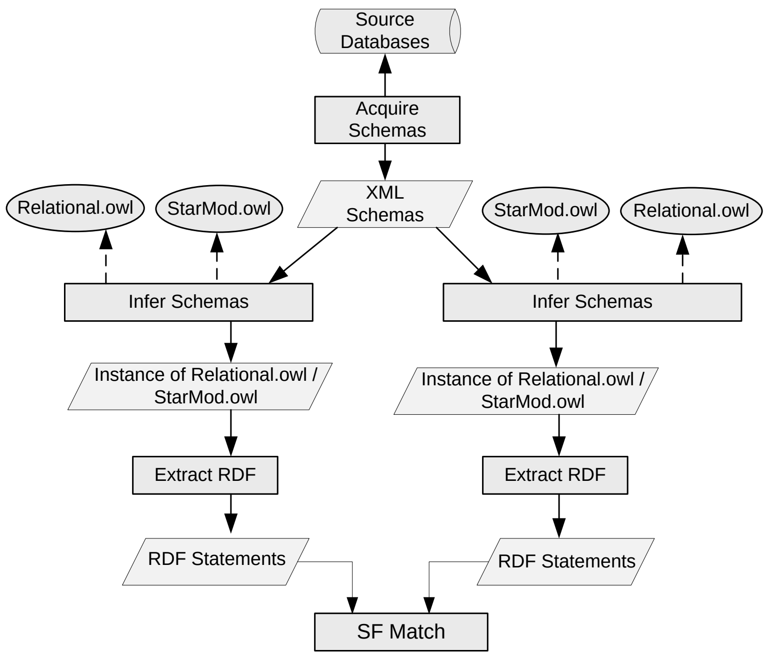


Figure 3.5: Automation of the schema matching process.

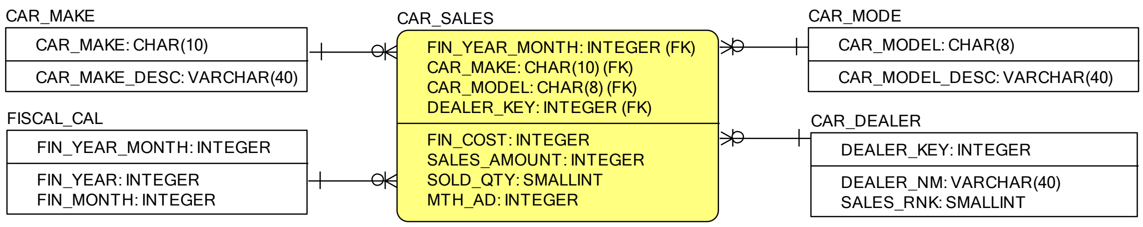


Figure 3.6: The Star schema (identified as PP2*) matched against the schema in Figure 3.7.*

above for the Star schema shown in Figure 3.6.

```
<rdf:type rdf:resource="&amp;star;Star"/>

    <star:hasDimension rdf:resource="#Dimension__FINANCIAL_CAL"/>

    <star:hasDimension rdf:resource="#Dimension__CAR_DEALER"/>

    <star:hasDimension rdf:resource="#Dimension__CAR_MODEL"/>

    <star:hasDimension rdf:resource="#Dimension__CAR_MAKE"/>

    <star:hasFact rdf:resource="#Fact__CAR_SALES"/>

</owl:Class>
```

```
<owl:Class rdf:ID="Dimension__FINANCIAL_CAL">
    <rdf:type rdf:resource="&amp;star;Dimension"/>
    <star:name rdf:resource="FINANCIAL_CAL"/>
    <star:hasAttribute rdf:resource="#Att__FINANCIAL_CAL__FIN_YEAR_MONTH"/>
    <star:hasAttribute rdf:resource="#Att__FINANCIAL_CAL__FIN_YEAR"/>
    <star:hasAttribute rdf:resource="#Att__FINANCIAL_CAL__FIN_MONTH"/>
    <star:dimUniquelyIdentifiedBy>
        <star:PrimaryKey>
            <star:hasAttribute rdf:resource="#Att__FINANCIAL_CAL__FIN_YEAR_MONTH"/>
            </star:PrimaryKey>
        </star:dimUniquelyIdentifiedBy>
</owl:Class>
<owl:DataTypeProperty rdf:ID="Att__FINANCIAL_CAL__FIN_YEAR_MONTH">
    <rdfs:domain rdf:resource="#Dimension__FINANCIAL_CAL"/>
    <star:dataFormatType rdf:resource="#DataType__integer"/>
    <star:key rdf:resource="YES"/>
    <star:name rdf:resource="FIN_YEAR_MONTH"/>
    <rdf:type rdf:resource="&amp;star;DataAttribute"/>
</owl:DataTypeProperty>
```

The following shows a subset of RDF statements produced for the above instance of StarMod. Each statement consists of a source, predicate and target.

```
SF, rdf_type, Star
SF, star_hasDimension, Dimension__FINANCIAL_CAL
SF, star_hasDimension, Dimension__CAR_DEALER
SF, star_hasDimension, Dimension__CAR_MODEL
SF, star_hasDimension, Dimension__CAR_MAKE
```

```
SF, star_hasFact, Fact__CAR_SALES

Dimension__FINANCIAL_CAL, rdf_type, Dimension

Dimension__FINANCIAL_CAL, star_name, FINANCIAL_CAL

Dimension__FINANCIAL_CAL, star_hasAttribute, Att__FINANCIAL_CAL__FIN_YEAR_MONTH

Dimension__FINANCIAL_CAL, star_hasAttribute, Att__FINANCIAL_CAL__FIN_YEAR

Dimension__FINANCIAL_CAL, star_hasAttribute, Att__FINANCIAL_CAL__FIN_MONTH

Dimension__FINANCIAL_CAL, star_dimUniquelyIdentifiedBy, PK__Dimension__FINANCIAL_CAL

PK__Dimension__FINANCIAL_CAL, star_type, PrimaryKey

PK__Dimension__FINANCIAL_CAL, star_keyedOn, PrimaryKey_on__Att__FINANCIAL_CAL__FIN_YEAR_MONTH

PrimaryKey_on__Att__FINANCIAL_CAL__FIN_YEAR_MONTH, star_includesAttribute,

               Att__FINANCIAL_CAL__FIN_YEAR_MONTH

Att__FINANCIAL_CAL__FIN_YEAR, rdfs_domain, Dimension__FINANCIAL_CAL

Att__FINANCIAL_CAL__FIN_YEAR, star_dataFormatType, DataType__integer

Att__FINANCIAL_CAL__FIN_YEAR, star_name, FIN_YEAR

Att__FINANCIAL_CAL__FIN_YEAR, rdf_type, DataAttribute
```

3.4 Evaluation of StarMod in Schema Matching

Our objective is to establish that when compared to using a relational model, using a more precise representation of Star schemas can help improve their match results. We use two of the well known schema matching approaches for which there are readily available implementations. We used two types of schemas in our evaluation: schemas that are based on the Star schema model and those that are based on non-Star relational model.

We used Similarity Flooding (described in Section 2.1.2) for the evaluation because it was flexible and adaptable since it uses RDF statements which can be easily obtained from OWL instances of StarMod.

We also compared the results against those obtained from using COMA++ version (2008c) for the same schemas. This helps us establish how SF using both relational model and StarMod compares with COMA (i.e. COMA++), and whether there is a potential for improving COMA results by using more granular relational properties such as those defined in StarMod.

3.4.1 Discussion of Match Results for Example Schemas

Using the two schemas in Figures 3.7 which also appeared in Figure 1.4, and 3.6 as our running example, we compare and discuss their match results suggested by SF, SF* and COMA. For brevity we refer to SF* where we use the StarMod, and SF where we use the relational model to describe the schemas. To measure the quality of match results returned from SF, SF* and COMA, we use the same measure used by Melnik et al. [2002b] to which we referred to as *A-measure* as described in Section 2.1.7. Table 3.1 shows the match results suggested by SF, SF* and COMA.

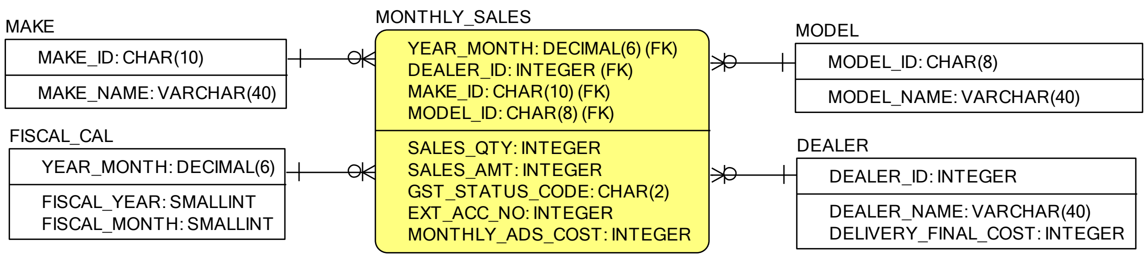


Figure 3.7: Star schema for our running example (identified as `PP1`*).*

The first two columns show the elements from the source and target schemas. Columns `SF`, `SF*` and `COMA` indicate if they considered the pair to be a match when using the StarMod and the relational model respectively. The column `Experts` shows the match results agreed

Match Results - Formula 'C'					
Schema: PP1.xsd	Schema:PP2.xsd	SF*	SF	COMA	Experts
DEALER.DEALER_ID	CAR_DEALER.DEALER_KEY	✓		✓	✓
DEALER.DEALER_NAME	CAR_DEALER.DEALER_NM	✓		✓	✓
FISCAL_CAL.FISCAL_MONTH	FINANCIAL_CAL.FIN_MONTH	✓	✓	✓	✓
FISCAL_CAL.FISCAL_YEAR	FINANCIAL_CAL.FIN_YEAR	✓	✓	✓	✓
FISCAL_CAL.YEAR_MONTH	FINANCIAL_CAL.FIN_YEAR_MONTH	✓	✓	✓	✓
MAKE.MAKE_NAME	CAR_MAKE.CAR_MAKE_DESC	✓			✓
MAKE.MAKE_NAME	CAR_MAKE.CAR_MAKE			✓	
MAKE.MAKE_ID	CAR_MAKE.CAR_MAKE	✓		✓	✓
MODEL.MODEL_ID	CAR_MODEL.CAR_MODEL	✓		✓	✓
MODEL.MODEL_NAME	CAR_MODEL.CAR_MODEL_DESC	✓			✓
MONTHLY_SALES.DEALER_ID	CAR_SALES.DEALER_KEY	✓		✓	✓
MONTHLY_SALES.MAKE_ID	CAR_SALES.CAR_MAKE	✓		✓	✓
MONTHLY_SALES.MODEL_ID	CAR_SALES.CAR_MODEL	✓		✓	✓
MONTHLY_SALES.MONTHLY_ADS_COST	CAR_SALES.FIN_COST	✓			
MONTHLY_SALES.SALES_AMT	CAR_SALES.SALES_AMOUNT	✓	✓	✓	✓
MONTHLY_SALES.SALES_QTY	CAR_SALES.SOLD_QTY	✓		✓	✓
MONTHLY_SALES.YEAR_MONTH	CAR_SALES.FIN_YEAR_MONTH	✓		✓	✓
DEALER	CAR_DEALER	✓	✓	✓	✓
FISCAL_CAL	FINANCIAL_CAL	✓	✓	✓	✓
MAKE	CAR_MAKE	✓	✓	✓	✓
MODEL	CAR_MODEL	✓	✓	✓	✓
MONTHLY_SALES	CAR_SALES	✓	✓	✓	✓
DEALER.DELIVERY_FINAL_COST	CAR_SALES.FIN_COST		✓	✓	
MONTHLY_SALES.SALES_QTY	CAR_DEALER.SALES_RNK		✓		
MONTHLY_SALES.MONTHLY_ADS_CST	CAR_SALES.MONTH_AD				✓
Number of correct matches		20	9	18	
Total number of matches		21	11	20	21
A-Measure		0.90	0.33	0.76	

Table 3.1: Comparison of match results for the example schemas.

by at least 2 of the 3 experts who manually matched the two schemas. The results from this example show higher values of *A-measure* for SF* over SF and COMA. Next, we compare the results of the three approaches.

SF versus SF*: SF benefits from relatively limited information such as data type, column name and table name and, hence, it is not able to make sufficient distinction between columns with strong similarity of their names. This has led to significantly lower precision for SF and is particularly visible where columns with identical names appear in dimension and fact tables. On the other hand, stronger classification of tables, columns and relationships by SF* has resulted in higher values of *A-measure*.

COMA versus SF and SF*: COMA, benefiting from a combination of matchers outperforms SF by a significant margin. When however, comparing its results with SF*, we find that the improved SF* result competes well with COMA's result. The false positive match between `MONTHLY_SALES.MONTHLY_ADS_COST` and `CAR_SALES.FIN_COST` returned by SF* is explained by two factors: (i) Their similarity is increased by the fact that they are both classified as *measures* this prevented a false positive match between `CAR_SALES.FIN_COST` a measure, and `DEALER.DELIVERY_FINAL_COST` a *DataAttribute*. (ii) Both, `CAR_SALES.FIN_COST` and `CAR_SALES.MTH_AD` competing to match against `MONTHLY_SALES.MONTHLY_ADS_COST` are measures but the former has a much stronger similarity of name with that of the target column.

We also observe that SF* has better *recall* and *precision* than COMA, however, there are instances where COMA's result could be improved by using specialized properties, e.g. the false negative match between `MONTHLY_SALES.MONTHLY_ADS_COST` and `CAR_SALES.FIN_COST` could be prevented if these were classified as *Measures*, and `DEALER.DELIVERY_FINAL_COST` which has a false positive match with `MONTHLY_SALES.MONTHLY_ADS_COST` was classified as a *DataAttribute*. In the next section, we validate our findings by using a larger collection of Star and non-Star schemas.

3.4.2 Evaluation of Using StarMod in Matching Schemas on a Larger Scale

We used 18 pairs of schemas, 14 of which were based on those collected from text books, industry and internet resources of which 8 pairs were Star schemas, and 6 pairs were non-Star schemas. We also included our running example and the 3 less complex relational schemas used by Melnik et al. [2002b] in their experiment.

The 18 pairs of schemas were divided into 3 subsets, each of which included 3 pairs of Star schemas and 3 pairs of non-Star schemas. We used 9 experts from the industry and academics to participate in the manual matching of schemas. Appendix A includes instructions given to the participants for performing this task.

These experts were also divided into 3 equal groups. Each group was allocated a subset of schema pairs. All members of each group were asked to match all schema pairs allocated to their groups. Schema diagrams and DDLs were provided for each schema, however, participants were not given any indication as to whether the schemas were Star or not. Appendix B includes diagrams of the models and their DDLs as provided to the participants. We ran SF and SF* against the 18 pairs of schemas.

To have confidence in our implementation, we also used the OIM model used by Melnik et al. [2002b] and made sure that results returned by SF were identical to those returned when using the OIM model and matched with the results of the three pairs of schemas used by Melnik.

Table 3.2 includes two sub-tables showing the results of our experiments against non-Star and Star schemas [1]. The equal or higher scores are highlighted in **bold**. In each sub-table, the first column shows the schema pair. The next three columns show the *A-measure* returned from SF, SF* and COMA. The explanation for negative *A-measure* scores is given in Section 2.1.7. Please see Appendix C for detailed results on the suggested matchings and expected results.

We now compare the three approaches for Star and non-Star based schemas.

SF versus SF*: In respect to the schema pairs involving Star schemas, the match results

[1]Minor variations compared to the published results at DEXA2011 are due to rounding and review of the calculations.

Non-Star schemas			
Schema Pair	SF	SF*	COMA
M7L, M7R	**1.0000**	**1.0000**	0.8000
M8L, M9R	0.3000	0.3000	**0.5000**
R05, R05A	0.3750	0.1250	**0.6250**
M8L, M8R	0.5294	**0.7059**	0.4706
R01, R02	0.2381	**0.5238**	0.4286
R03, R06	0.2308	**0.4615**	0.2308
R06, R07	**0.2857**	0.2143	0.0714
R07, R03	0.6667	0.7222	**0.8889**
R08A, R08B	0.3529	0.2353	**0.5294**
Mean	0.4421	0.4764	0.505

Star schemas			
Schema Pair	SF	SF*	COMA
PP1, PP2	0.3333	**0.9048**	0.7143
T01A, T01B	0.6250	**0.8333**	**0.8333**
T07A, T09A	0.0000	**0.3333**	-0.1667
A01, A02	0.3548	**0.6774**	0.6129
T02A, T02B	-0.3077	**0.0769**	0.000
T10, T11	0.3500	**0.4500**	0.4500
T05A, T05B	0.0476	**0.0952**	0.0000
T06B, T04	-0.5556	**-0.2222**	-0.3333
T07B, T11B	0.2727	**0.4091**	0.2273
Mean	0.1139	0.3953	0.2597

Table 3.2: Accuracy measures for schemas used in the evaluation.

show consistent improvement for SF* over the SF. The results are also consistent with our observations during the running example. In respect to the schema pairs involving non-Star schemas, the results are mixed with an overall similar performance for using SF or SF*. A closer examination of the schemas shows that SF performs better than SF* where the topology of tables are very different between the two schemas. This difference (as expected) results in a mismatch between how properties are classified. Conversely, SF* performs better than SF where there are reference tables (resembling Star models), and where the structure of tables across the two schemas are similar.

SF* versus COMA: In respect to the schema pairs involving Star schemas, we find that *A-measure* for SF* is higher (by a smaller margin) or the same as COMA. The results are consistent with our findings from the running example showing that COMA's relatively higher number of false positive cases can be reduced by using a more precise classification of relational properties for Star schemas. In respect to the schema pairs involving non-Star schemas, as with SF versus SF*, the results are again mixed with an overall similar performance for both.

SF versus COMA: For Star and non-Star schemas, COMA has an overall better results than SF because of higher number of true positive cases for COMA which is even stronger for Star schemas. This indicates that COMA's matchers appear to make good use of the structural similarity.

The results in Table 3.2 support our hypothesis that using a more precise description of Star schemas improves their match results and can be also effective for arbitrary relational schemas. They also indicate that COMA could benefit from StarMod properties.

Table 3.3 shows the probability values for the corresponding null hypotheses based on the paired 2-tailed *t-test* with 8 degrees of freedom. It shows that results in Table 3.2 are statistically significant in supporting our hypotheses in respect to the performance of SF* against Star schemas with the probability values being less than 0.05. As for non-Star schemas, the results are not statistically significant enough for the performance of any of the methods over the others.

Non-Star schemas	
Hypothesis	P-Value
SF* performing better over SF	0.5679
COMA performing better over SF	0.3370
COMA performing better over SF*	0.7579

Star schemas	
Hypothesis	P-Value
SF* performing better over SF	0.0005
COMA performing better over SF	0.0383
SF* performing better over COMA	0.0283

Table 3.3: Probability values for the null hypotheses.

Complexity:

The complexity for using StarMod is similar to the complexity of the Similarity Flooding algorithm as described by Melnik et al. [2002a]. In Section 3.3, we described properties of StarMod. These properties are different classifications of the same properties in the relational model. Similarly, the relationships introduced for StarMod and shown in Figure 3.4

are specializations of relationships that exist in the relational model. Therefore the computational complexity for using StarMod is similar to that of the relational model.

3.5 Discussion

Considerable research has been done on matching relational schemas. However, the predictable topology and properties of Star schemas motivates us to consider a more precise representation of them for matching purpose.

We proposed StarMod as such a representation and described it using UML and OWL languages. We used the UML version to visualize StarMod and the OWL version to automate the schema matching process.

Our proposed UML version is similar to the one suggested by Luján-Mora et al. [2006] which is aimed at physical properties of Star schemas, and includes additional properties such as degenerates and keys. Our OWL representation of StarMod corresponds to the UML representation and is an extension of `Relational.owl` by de Laborda and Conrad [2005]. It provides the flexibility to include additional domain ontologies for even a more precise description and accurate matching.

We described the inference of Star properties and their applications in matching Star schemas. The main distinction between our approach and previous work is that our approach is model driven and exploits a more comprehensive set of properties. We also described our approach in using Similarity Flooding to match Star schemas represented using StarMod. Existing works do not compare the effectiveness of their approach with others.

We used two well known schema matching algorithms in the evaluation of our approach. Using Similarity Flooding, we demonstrated that a more precise model such as StarMod can improve the quality of match results for Star schemas, and that it could be also effective against arbitrary non-Star relational schemas. Comparing the results with those obtained from COMA++, demonstrates that even though COMA++ performs more accurately than Similarity Flooding when using basic relational properties, it fails to outperform Similarity Flooding when using StarMod.

As we saw in Section 3.3.2, inference of Star properties from relational schemas may not be guaranteed to be accurate for some properties. This problem was more visible in relational schemas than Star schemas, and concerned measures, degenerate dimensions and degenerate facts. Our experiments show that the gained true positives substantially outweigh the side effect resulting in more false positive and false negative cases. The fact that Star schemas described using StarMod properties are implemented in OWL provides the opportunity to augment the schemas with additional domain ontologies to help with the inference of those properties using an OWL reasoner.

In the next chapter, we propose algorithms to infer aggregation hierarchies which we will require for instance matching discussed in Chapter 5.

Chapter 4

Inferring Aggregation Hierarchies

> ”The top entrusts the understanding of detail to the lower levels, whilst the lower levels credit the top with understanding of the general, and so all are mutually deceived.”

Karl Marx (1818 - 1883)

Compatibility between dimensions is the key to accurate integration of multidimensional data. It ensures that the integrated data is lossless, summarizable and conforms to the common hierarchy using the matching levels. As discussed in Chapter 2, Torlone describes *coherence*, *consistency* and *soundness* as conditions necessary for compatibility between dimensions. We explained in Section 2.3.2 that schema based aggregation hierarchies are not often available for heterogeneous and external data marts, or even local data marts. In their absence, we propose to infer the aggregation hierarchies for dimension tables from their instances. We formulate the problem of inferring aggregation hierarchies as computing from data, a minimal directed graph for the roll-up relationships between levels, and propose algorithms to this end.

In Section 4.1, we define multidimensional properties in terms of their relational implementation and describe how they map to their definitions at a conceptual level. In Section 4.2, we introduce algorithms to infer aggregation hierarchies from instances of dimension tables. In Section 4.3, we establish the relationship between the intended hierarchies and the inferred hierarchies. We prove that inferred hierarchies are sufficient for establishing compatibility and summarizability of integrated data. Finally, in Section 4.5, we discuss our experiments and findings.

4.1 Relational Representation of Multidimensional Databases

The relational implementation of OLAP databases are based on the Star schema model described in Section 1.2.1. In Section 2.4.2, we explained properties of multidimensional databases at a conceptual level. In this section, we define concepts such as dimension, level, member and aggregation hierarchy in the context of their physical implementation in relational multidimensional databases.

As seen in Sections 2.2.1 and 2.4.2, Torlone [2008] defines dimensions in terms of levels and members. A dimension table is the relational implementation of a dimension, and is defined in terms of attributes as opposed to levels.

For simplicity, we assume that each level is associated with a single attribute of a dimension table and, therefore, in this thesis we refer to levels as being dimension attributes. In this case, values of dimension attributes correspond to members of levels. We will infer aggregation hierarchies as being the partial order relationship between dimension attributes.

We name each level the same as its attribute. In some cases, multiple attributes could have the same partial order relationship with every other attribute of the dimension. We refer to these as *co-level attributes*. In such cases, we use a combination of the names of co-level attributes (separated using '/') as the name of the level.

Having mapped the definition of dimension to that of dimension table, we are able to apply Torlone's definition of compatibility and its requirements being coherence, consistency and soundness to ensure the accuracy of integration.

Definition 4.1 below draws from definitions of dimension and hierarchy in [Rafanelli and Shoshani, 1990], [Shoshani, 1997], and [Cabibbo and Torlone, 2005].

Definition 4.1. *A dimension table D has an aggregation hierarchy H of levels. Given two levels l and l' of a dimension table D, we say level l rolls-up to level l' (which we denote as $l \preceq l'$) if we can compute aggregated facts at level l' from facts at level l. The roll-up relationship $\preceq$ forms a partial order over the levels. The aggregation hierarchy H is a directed acyclic graph (DAG) with no (explicit) transitive edge, where the nodes of the graph are the levels and the edges are those roll-up relationships in the covering relation of the partial order between the levels.*

A dimension table may have multiple hierarchies, although, our proposed algorithm infers all hierarchies, for simplicity in our definitions, we assume a single hierarchy for a dimension table.

Definition 4.2. *Given the dimension table D, its instance $I(D)$ is a set of tuple $\{t_1, ..., t_s\}$ where each tuple $t_a = \langle v_{a_1}, ..., v_{a_n} \rangle$ contains n values, and each value v_{a_i} is an element from the corresponding domain of attribute A_i.*

We can define a partial order over the attributes in a dimension table.

Definition 4.3. *Given an instance $I(D)$ of a dimension table $D(A_1, ..., A_n)$, we say $A_i \leq A_j$ if for every pair of tuples $t_a = \langle v_{a_1}, ..., v_{a_n} \rangle$ and $t_b = \langle v_{b_1}, ..., v_{b_n} \rangle$ in $I(D)$, $v_{a_i} = v_{b_i}$ implies $v_{a_j} = v_{b_j}$. The $\leq$ relationship forms a partial order P over the set of dimension attributes.*

If there is a functional dependency $A_i \rightarrow A_j$ between two attributes of the relation $D(A_1, ..., A_n)$ representing the dimension table, then it follows that $A_i \leq A_j$ must hold in every instance $I(D)$. Furthermore, if two attributes A_i and A_j correspond to different levels l and l' such that $l \preceq l'$, then we require $A_i \leq A_j$. Likewise, if two attributes A_i and A_j correspond to the same level, then we require $A_i \leq A_j$ and $A_j \leq A_i$.

Where $I(D)$ conforms to the hierarchy of its dimension table, it also forms a DAG or tree in which each member is a node and the edges are the same as those that connect the members' respective levels.

Definition 4.4. *A Schema-defined (or schema based) Aggregation Hierarchy (SAH) is an aggregation hierarchy that is defined as part of the schema of D and constitutes a constraint on the tuples in any instance $I(D)$.*

SAH describes roll-up relationships between levels as intended during the design phase to fit all possible instances. Ideally, an aggregation hierarchy must be defined as part of the schema definition and then implemented as constraints enforced by the DBMS (Database Management System), or by the application that populates the dimension to ensure that these constraints are not violated. For example, Oracle's syntax for creating a dimension table allows specifying the aggregation hierarchy through explicit description of each level, attributes for each level and the relationship between the levels [Oracle, 2005].

This syntax is not, however, part of the standard SQL syntax and is not supported by all database vendors. Other reasons for dimension hierarchies not being available include absence of design artifacts, and access to heterogeneous dimension tables or those external to the organization. In the following section, we describe algorithms to infer the levels and hierarchies from instances of dimension tables.

4.2 Inferring the Partial Order Attributes

There are two major motivating factors for inferring hierarchies from instances of dimension tables: (i) An instance of a dimension table is constrained by its SAH, and (ii) The instance does not need to be interpreted in any way. There is, however, a drawback, that is, the population of a dimension table may be partial. Therefore, the resulting inferred aggregation hierarchies (IAH) may vary from the intended SAH. How similar is the IAH for a dimension table to the SAH for the same dimension table depends on how closely the instance represents the SAH. Whilst, partial population of dimension tables can occur, it is, however, rare in the real world. We will discuss this issue when we establish the viability of IAHs in testing for dimension compatibility in Section 4.4.

Definition 4.5. *Given an instance $I(D)$ of dimension table D, the inferred partial order of attributes for D is the set of partial order relationships (P) inferred from the partial order relationships between attributes of D and inferred from $I(D)$.*

In line with the definitions in Section 4.1, and in this section, we obtain the IAH in three steps. In the first step, we obtain the partial order between dimension attributes. In the second step, we remove the transitive partial order relationships and, finally, we obtain the levels and the inferred aggregation hierarchy or hierarchies.

4.2.1 Inferring the Partial Order of Attributes

We propose Algorithm 4.1 for inferring the partial order of attributes. We explain this algorithm using the following example: Let us suppose we wish to determine if `Country` $\leq$ `City`. The algorithm first sorts the tuples in $I(D)$ on `Country`. It then scans the values of `Country` and `City`. As long as the value in `Country` remains the same from one tuple to the next, the value in `City` must also remain the same on the same tuples. If this holds true for the entire $I(D)$ then the roll-up relationship will hold. Given the sample data for `Store` in Table 4.1, this roll-up relationship does not hold. By scanning $I(D)$ for `Country` against the remaining attributes we can see that only `Country` $\leq$ `Region` holds true. This process is applied for each attributes. The scan of $I(D)$ for each pair of pair of attributes can, however, stop as soon as it is established that the partial order relationship does not hold.

Figure 4.1 shows the partial order of attributes inferred from the sample data for `Store` in Table 4.1, where dashed lines represent transitive relationships.

Region	Country	Division	City	Locality	Store
Asia Pacific	Australia	Div1	Sydney	Ryde	st1
Asia Pacific	Australia	Div1	Sydney	Ryde	st2
Asia Pacific	Australia	Div1	Melbourne	Epping	st3
Asia Pacific	Australia	Div1	Melbourne	Morang	st4
Asia Pacific	Australia	Div1	Melbourne	Brighton	st5
Asia Pacific	Australia	Div2	Geelong	Hill	st6

Table 4.1: Sample data for a `Store` *dimension table.*

The complexity of inferring partial order of attributes: The algorithm performs a sort for each dimension attribute with the complexity in the order of $n\,p \log p$ where n is the

Algorithm 4.1 Inferring the partial order relationships.

Input Tuples $I(D) = \{t_1, t_2, ..., t_p\}$ in the instance of a dimension table $D(A_1, A_2, ..., A_n)$, and
p is the number of tuples.
Output Partial order P of attributes.

```
P := {}
for each attribute A_i do
  Sort I(D) on A_i
  for each attribute A_j do
    for each tuple do
      if A_i on current tuple equals A_i on the previous tuple then
        if A_j on current tuple does not equal A_j on the previous tuple then
          exit this loop
        end if
      end if
    end for
    if end of tuples was reached then
      P := P ∪ {(A_i, A_j)}
    end if
  end for
end for
```

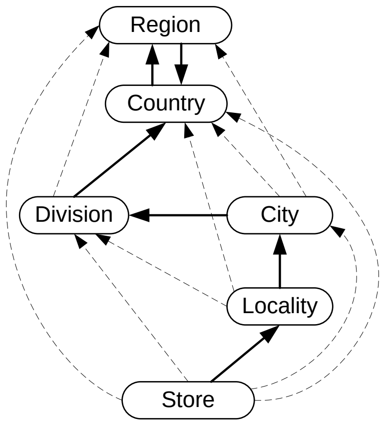


Figure 4.1: The partial order of attributes.

number of attributes and p is the number of tuples. We also scan $I(D)$ for every pair of attributes ($n^2 - 1$) with the complexity in the order of $n^2\,p$, though in some cases only a subset of $I(D)$ is scanned. Therefore, the complexity of the Algorithm 4.1 is $O(n^2\,p + n\,p \log p)$.

Observe that Algorithm 4.1 obviously computes all partial order relationships between each pair of levels.

4.2.2 Cover for Partial Order of Attributes

Definition of aggregation hierarchy does not include transitive roll-up relationships as values at each level can be computed from the next immediate child level. In order to remove transitive partial order relationships, we can use existing algorithms [Aho et al., 1972] that remove transitive edges from a directed graph. Figure 4.2 shows the partial order of attributes after removing transitive partial order relationships.

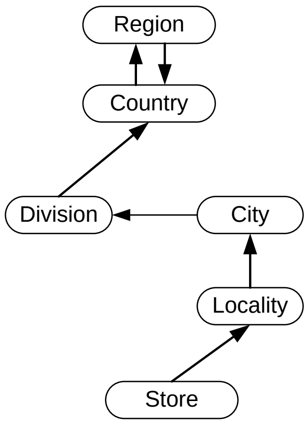


Figure 4.2: The partial order of attributes with no transitive relationship.

4.2.3 The Levels and the Inferred Aggregation Hierarchy

Based on Definition 4.3, the two attributes `Country` and `Region` in Figure 4.2 correspond to the same level. We propose Algorithm 4.2 to obtain the levels being disjoint subsets of attributes (L), and then associate each attribute with its corresponding level. The result is the inferred aggregation hierarchy over a set of levels with roll-up relationship between them. The resulting IAH is also an acyclic directed graph.

We assume that $q = |P|$ and $r = |L|$. Statements 2 to 8 of Algorithm 4.2 with complexity of q^2, add to L, each attribute of any partial order (p_m) as a level, unless there is another partial order (p_n) which makes their attributes to correspond to the same level in which case the added level will include both attributes. Statements 9 to 13 with the complexity of r^2 combine those subsets of L that have at least one common attribute. At this point, L contains disjoint subsets of attributes that correspond to the same level. Statements 14 to 16 with the complexity of $q\,r$ revisit the partial orders (copied into H_L as roll-ups) and assign to each level, a name that is derived from names of attributes that the level represents. Finally, duplicate roll-ups are removed. The complexity of statement 17 is q^2. The overall complexity for Algorithm 4.2 is $O(2\,q^2 + r^2 + q\,r)$.

Figure 4.3 shows the final inferred aggregation hierarchy after grouping co-level attributes and assigning distinct levels.

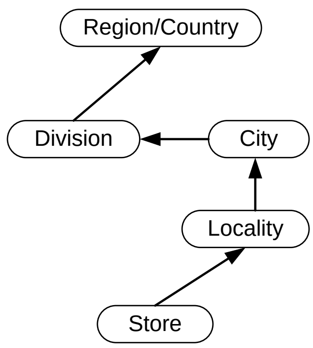


Figure 4.3: The inferred aggregation hierarchy.

Algorithm 4.2 Identifying levels and roll-ups.

Input P is the partial order of attributes with no transitive relationship.
Output H_L is the inferred aggregation hierarchy.
L is a set of levels corresponding to disjoint subsets of attributes.

$H_L := P, L = \{\}$
for each pair of partial order relationships ρ_m and ρ_n in P **do**
  if $\rho_m = A_i \leq A_j$ and $\rho_n = A_j \leq A_i$ **then**
    $L := L \cup \{A_i, A_j\}, H_L := H_L - \rho_m, \rho_n$
  else
    $L := L \cup \{A_i\}, \{A_j\}$
  end if
end for
for all $x \in L$ and $y \in L$ where $x \neq y$ **do**
  if $x \cap y \neq \emptyset$ **then**
    $L := L - \{x\}, L := L - \{y\}, L := L \cup \{(x \cup y)\}$
  end if
end for
for each partial-order $p_m = (A_i, A_j)$ in H_L and each subset of levels l_s in L **do**
  Replace any A_i and A_j that appear in l_s with a level name that is a combination of attribute names in l_s (separated by a '/').
end for
Remove any duplicate roll-up relationship from H_L.

4.3 Inferred Hierarchies Subsume Schema-Defined Hierarchies

Based on the inferred partial order relationships obtained from Algorithm 4.1 and shown in Figure 4.1, we have `Country` $\leq$ `Region` which is a valid partial order relationship, and `Region` $\leq$ `Country` which may, indeed, be spurious due to the incomplete instance of `Store` in Table 4.1. If a tuple ⟨`Asia Pacific,New Zealand,Div3,Wellington,Brooklyn,st7`⟩ is added to the instance of the `Store` in Table 4.1, the second partial order will not hold true. Other spurious partial orders are `Locality` $\leq$ `Division` and `City` $\leq$ `Division`. This example indicates that as the population of the dimension table grows, the partial order relationships in IAH converge towards those in the SAH. This is formulated in the following theorem.

Theorem 4.1. *Given an instance $I(D)$ of a dimension table D and the inferred partial order of its attributes P derived using Algorithm 4.1, the partial order of attributes P', and its covering relation, for the schema-defined aggregation hierarchy for D must be subgraph of P.*

Proof. It is possible to obtain a covering relation of the partial order over the attributes from the schema-defined hierarchy. We can also obtain the transitive closure of the partial order over the attributes from its covering relation.

If all domain members of each base level of each schema-defined hierarchy are included in a given instance of a dimension table, since all partial order relationships of attributes are captured in Algorithm 4.1, then P' is equivalent to P.

Otherwise, suppose that some members of the base levels are missing. While some spurious roll-ups are added, none of the partial orders from P is removed. The latter point can be proved easily by the fact that the data from which the inferred partial order is derived is constrained by the schema-defined aggregation hierarchy. □

From the above proof we have the following corollary.

Corollary 4.1. *If all domain members of each base level of each schema-defined hierarchy are included in a given instance of a dimension table, the inferred aggregation hierarchy H' also exists as a schema-defined aggregation hierarchy H.*

Proof. If there is a partial order relationship p_i in H but not in H', it means that the instance did not conform to the constraint implied by p_i. If there is a partial order relationship p_i in H' but not in H, it means that the instance lacked the data that supports p_i. □

If not all members of base levels are present in a given dimension table, all roll-up relationships in the schema-defined hierarchy are present or implied (by transitive roll-ups) in the inferred aggregation hierarchy.

Definition 4.6. *Given two levels l_1 and l_2, and summary values for l_1, the roll-up $\rho = l_1 \preceq l_2$ is summarizable if using ρ yields correct summary values for l_2.*

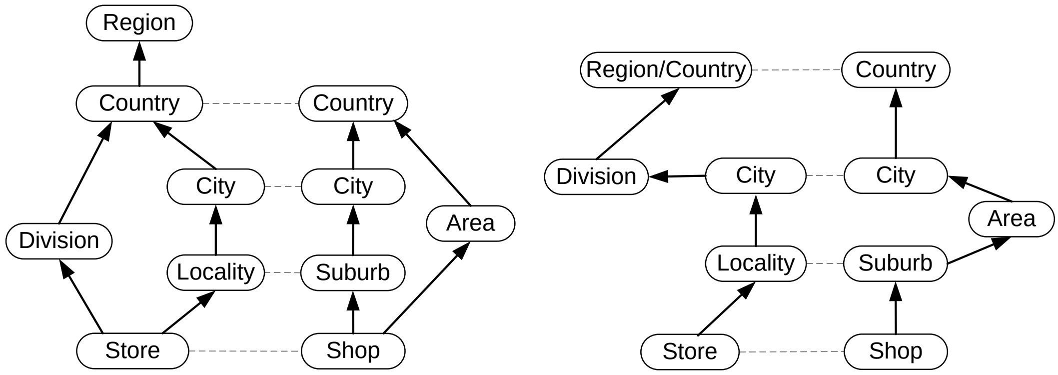
Using either type of the aggregation hierarchies, summary values for each level can be computed by summing the values at the lower level. This is due to the fact that the roll-up relationships either constrain the data, or they are inferred from data. Consequently, using either of them guarantees the summarizability.

4.4 Integration of Matching Dimension Tables using Inferred Aggregation Hierarchies

In this section, we consider the properties of the matching between dimension tables using their SAHs and compare them with the same properties established using their IAHs. Each property may be true or false for either type of hierarchy, resulting in 4 different cases for each property. We examine each case and show that use of inferred aggregation hierarchies is sufficient for establishing compatibility and ensuring that the integrated data is summarizable.

4.4.1 Properties of Compatible Dimension Tables

As discussed in Section 2.4.2, for accurate integration of data marts, the matching between dimensions over their matching levels must be compatible. That is, the matching between



(a) Schema-defined matching. (b) Inferred matching.

Figure 4.4: Matchings levels in `Store` *and* `Shop` *dimension tables.*

each pair of matching levels must be coherent, consistent and sound. Similarly, as discussed in Section 4.1, we can apply the requirements of compatibility to dimension tables.

Example 4.1. *Figure 4.4(a) shows a schema-defined matching between levels of dimension tables* `Store(Region,Country,Division,City,Locality,Store)` *, and* `Shop(Country,Area,City, Suburb,Shop)`*. Suppose that the instance for the Store, and Shop are as shown in Tables 4.1 and 4.2. The matching is sound, coherent and consistent.*

Country	City	Area	Suburb	Shop
Australia	Sydney	NE	Ryde	st1
Australia	Sydney	NE	Ryde	st2
Australia	Melbourne	NT	Epping	st3
Australia	Melbourne	NT	Morang	st4
Australia	Melbourne	SW	Brighton	st5
Australia	Geelong	NW	Hill	st6

Table 4.2: Sample data for a `Shop` *dimension table.*

It is important to ensure that the data remains summarizable after the integration. We make the following observation.

Theorem 4.2. *Integration based on sound, coherent and consistent matchings ensures summarizability.*

Proof. We assume that before the integration, instances of dimension tables conform to their hierarchies and are summarizable. The coherence and consistency ensure that the integrating hierarchies are identical and that the data after the integration satisfies roll-up relationships in the original hierarchies. Therefore, for the result of the integration not to be summarizable, it can only be that the instance of least one of the dimension tables is not summarizable. □

Integration based on the matching levels of the `Store` and `Shop` will ensure the correctness of summarization for drill-across queries. When using inferred aggregation hierarchies for integration, the summarizability of facts after integration relies on that, testing for soundness, coherence and consistency between matching levels of inferred aggregation hierarchies succeed.

Let a matching defined using inferred hierarchies be called an *inferred matching*. Similar to a matching defined using schema-defined hierarchies, an inferred matching between dimension tables comprises a set of one-to-one mappings between their matching levels.

Soundness of inferred matchings: Sound matching between (schema-defined or inferred) dimension tables requires that for all matching levels, their members match. For example, all members of `Locality` in `Store` and those of `Suburb` in `Shop` must match.

Since soundness does not depend on the roll-up relationship between levels, obviously testing for soundness using inferred aggregation hierarchies is the same as that using schema-defined aggregation hierarchies.

In the next two sections, we show that testing for coherence and consistency is feasible using inferred aggregation hierarchies.

4.4.2 The Coherence of Inferred Matchings

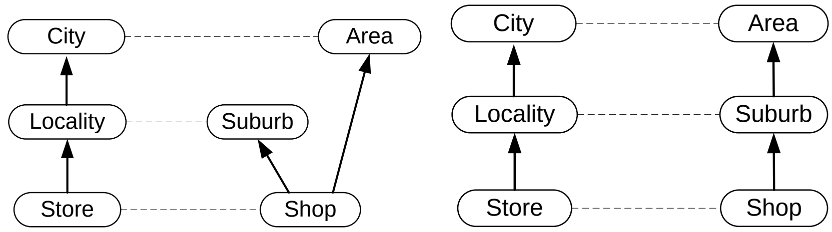
When inferred aggregation hierarchies are the same as the schema defined hierarchies, matchings defined for inferred hierarchies are the same as those defined for schema defined hierarchies. However, due to the partial population of the dimension tables, the inferred aggregation hierarchy may contain spurious roll-up relationships that are not present in the schema-defined hierarchy. As a result, a matching defined for inferred aggregation hierarchies may be different to the matching defined for schema-defined hierarchies. The following scenarios may arise:

True coherence: The coherence of an inferred matching defined on a set of levels is true coherence if the matching on these levels for the schema-defined matching is also coherent. This occurs when the non-spurious as well as the spurious roll-ups (if any) over matching levels are the same between inferred hierarchies.

Example 4.2. *Based on the sample data for* `Store` *and* `Shop`*, their inferred aggregation hierarchies and their matching is shown in Figure 4.4(b). The matching is sound and coherent. In comparison with the matching for schema-defined hierarchies shown in Figure 4.4(a), this matching using inferred hierarchies is truly sound and coherent.*

True incoherence: The incoherence of an inferred matching defined on a set of levels is true incoherence if the matching on these levels for the schema-defined matching is also incoherent. This occurs when the inferred hierarchies are the same as schema-defined hierarchies and/or the spurious roll-ups are different between the inferred hierarchies.

False coherence: The coherence of an inferred matching on a set of levels is false co-



(a) Incoherent schema defined matching (b) A coherent inferred matching

Figure 4.5: A false coherent matching that uses inferred hierarchies.

herence if the matching defined on these levels for the schema-defined hierarchies is not coherent. False coherence occurs when there are some spurious roll-up relationships in one of the inferred hierarchies that are also present in the other inferred hierarchy but as non-spurious roll-ups. In this case the matching is, indeed, coherent for instances from which the hierarchies are inferred.

Example 4.3. *The matching on the schema-defined hierarchies shown in Figure 4.5(a) is incoherent, but the inferred matching shown in Figure 4.5(b) is coherent. The inferred matching is a false coherent matching. This is made possible because of the spurious roll-up* `Suburb` $\preceq$ `Area`.

False incoherence: The incoherence of an inferred matching on a set of levels is false incoherence if the matching defined on these levels for the schema-defined hierarchies is indeed coherent.

False incoherence occurs when the spurious roll-up relationships relate the matching levels differently or some spurious roll-up relationships are missing in only one of the inferred hierarchies.

Example 4.4. *Suppose that* `Division` *and* `Area` *were matching levels in Figure 4.4(a). In this case, the schema-defined matching between* `Store` *and* `Shop` *remains coherent. However, the inferred*

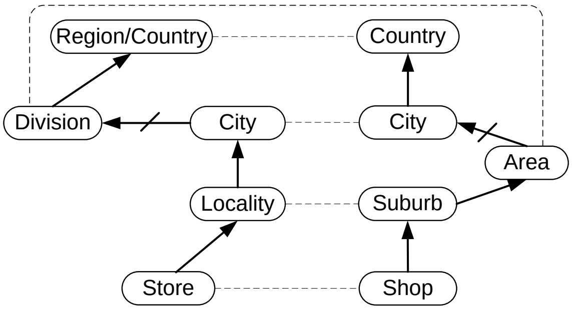


Figure 4.6: False incoherent inferred matching.

matching is incoherent. Figure 4.6 shows spurious roll-ups (denoted by $\nrightarrow$) relating matching levels differently.

4.4.3 The Consistency of Inferred Matchings

True consistency: The consistency of an inferred matching on a set of levels is true consistency if the matching defined on these levels for the schema-defined hierarchies is also consistent.

Example 4.5. *Based on Tables 4.1 and 4.2, the result of the integration of the two sample data for* `Store` *and* `Shop` *satisfy the constraints in the schema-defined and inferred hierarchies for these two dimension tables.*

True inconsistency: The inconsistency of an inferred matching on a set of levels is true inconsistency if the matching defined on these levels for the schema-defined hierarchies is also inconsistent.

Example 4.6. *Given the schema-defined matchings for* `Store` *and* `Shop` *dimension tables (as shown in Figure 4.4(a)), if we considered updating the* `Store` *with the tuple* `<Asia Pacific,Australia, Div3,Melbourne,Chelsea,st8>`*, and include the tuple* `<UK,London,WC,Chelsea, st9>` *in the* *dimension* `Shop`*, then based on their schema-defined hierarchies, the two dimension tables would be coherent but not consistent. The reason is that the data after integration does not reflect the roll-ups* `Locality` $\preceq$ `City` *and* `Suburb` $\preceq$ `City` *present in the original hierarchies of* `Store` *and* `Shop` *respectively. The matching using inferred hierarchies would be also inconsistent, if the two tuples were also present in the instance of* `Store` *(Table 4.3) and* `Shop` *(Table 4.4) dimension tables.*

Region	Country	Division	City	Locality	Store
Asia Pacific	Australia	Div1	Sydney	Ryde	st1
Asia Pacific	Australia	Div1	Sydney	Ryde	st2
Asia Pacific	Australia	Div1	Melbourne	Epping	st3
Asia Pacific	Australia	Div1	Melbourne	Morang	st4
Asia Pacific	Australia	Div1	Melbourne	Brighton	st5
Asia Pacific	Australia	Div2	Geelong	Hill	st6
Asia Pacific	Australia	Div3	Melbourne	Chelsea	st8

Table 4.3: True inconsistency: sample instance for `Store` *dimension table.*

Country	City	Area	Suburb	Shop
Australia	Sydney	NE	Ryde	st1
Australia	Sydney	NE	Ryde	st2
Australia	Melbourne	NT	Epping	st3
Australia	Melbourne	NT	Morang	st4
Australia	Melbourne	SW	Brighton	st5
Australia	Geelong	NW	Hill	st6
UK	London	WC	Chelsea	st9

Table 4.4: True inconsistency: sample instance for `Shop` *dimension table.*

False consistency: The consistency of an inferred matching on a set of levels is false consistency if the matching defined on these levels for the schema-defined hierarchies is not consistent. This case implies that instances of a pair of dimension tables are not consistent using the schema-defined hierarchies for some roll-up relationships, but they are consistent using the inferred hierarchies. Similar to false coherence, this occurs when for example, the inferred hierarchy for one dimension is the same as the schema-defined hierarchy, but the inferred hierarchy for the other dimension includes spurious roll-ups which makes them to have a consistent matching. Again, the matching is, indeed, consistent for the purpose of integrating the instances from which the hierarchies are inferred.

False inconsistency: The inconsistency of an inferred matching on a set of levels is false inconsistency if the matching defined on these levels for the schema-defined hierarchies is consistent.

Example 4.7. *Suppose that the roll-up relationships* `Locality` $\preceq$ `City` *and* `Suburb` $\preceq$ `City` *were not defined as part of the schema-defined hierarchies for* `Store` *and* `Shop` *respectively. If the sample data for dimension table* `Shop` *in Table 4.2 included the tuple* `<Australia,Sydney,NT,Epping,st3>` *in place of* `<Australia,Melbourne,NT,Epping,st3>` *(resulting in Table 4.6), then the integrated data would conform to the schema-defined hierarchies but not to the inferred hierarchies which do include these roll-ups and are now violated since* `Epping` *rolls-up to different members in* `City`.

Region	Country	Division	City	Locality	Store
Asia Pacific	Australia	Div1	Sydney	Ryde	st1
Asia Pacific	Australia	Div1	Sydney	Ryde	st2
Asia Pacific	Australia	Div1	Melbourne	Epping	st3
Asia Pacific	Australia	Div1	Melbourne	Morang	st4
Asia Pacific	Australia	Div1	Melbourne	Brighton	st5
Asia Pacific	Australia	Div2	Geelong	Hill	st6

Table 4.5: False inconsistency: sample data for `Store` *dimension table.*

Country	City	Area	Suburb	Shop
Australia	Sydney	NE	Ryde	st1
Australia	Sydney	NE	Ryde	st2
Australia	Sydney	NT	Epping	st3
Australia	Melbourne	NT	Morang	st4
Australia	Melbourne	SW	Brighton	st5
Australia	Geelong	NW	Hill	st6

Table 4.6: False inconsistency: sample data for `Shop` *dimension table.*

Although, false incoherence and false inconsistency may prevent the integration, what is critical is that where we do proceed with the integration, the result of the integration will be correct and summarizable. Based on the above, use of IAHs to test for compatibility guarantees the summarizability of data after integration, and therefore, and IAHs are viable for testing compatibility for instances from which the aggregation hierarchies are inferred. Compared to the situation where we are unable to ensure the accuracy of the integration due to the absence of SAHs, this is a significant outcome and a viable solution to the problem. This is summarized in the theorem below.

Theorem 4.3. *A sound, coherent and consistent inferred matching is sufficient but not necessary for the summarizability of integration.*

Proof. A summarizable integration on the matching levels must have sound, coherent and consistent inferred matching. But, from the above discussions, it can be seen that an inferred matching may present as incoherent or inconsistent based on the current dimension instances, but they are indeed, coherent and consistent and, thus, can be integrated and the result is summarizable. □

4.5 Experiments

The purpose of our experiments is to measure the effectiveness of our algorithms for inferring aggregation hierarchies from dimension tables with real life data. For each one of our two experiments, the expected hierarchies are obtained from the business operations manuals. All algorithms are implemented in Java and run on PC with dual core and 2.5 GHz CPU and 3 GB memory.

In each experiment, all data from each dimension table was retrieved into a comma separated text file. The output from running the first algorithm is a set of pairs of attributes between which there is a partial order relationship. This partial order is then used in the second algorithm which removes the transitive partial order relationships and also infers the paths in the hierarchy. Finally, co-level attributes are grouped into a single level.

The first experiment involves a dimension table with 9 attributes for insurance products from 25 sub-companies with an instance that includes 8,614 rows. The runtime to infer all partial order relationships was 356 milliseconds.

By experimenting with the data related to each company separately, we were able to get the expected hierarchy. In some cases, co-level attributes were not correctly classified as belonging to the same level. For example, `PRODUCT_CLASS_CODE` and `PRODUCT_CLASS_DESC` must actually belong to the same level. However, varying synonymous values appearing in `PRODUCT_CLASS_DESC` prevented this expected grouping. Our first observation was that information on false negative roll-ups could be used for data cleansing.

The second experiment concerns a dimension table called `OCCUPATION` which is based on the ANZSIC standard for occupation codes. This dimension table has also 9 attributes and its instance includes 16,208 rows; the run time for inferring partial orders was 469 milliseconds. The hierarchies in this dimension table were also enforced by the application but separately across different companies. The expected hierarchies were returned after running the algorithm separately against the data for each company.

The following shows the information returned by our algorithms in inferring the hierarchies for only one of the companies. The first segment shows co-level attributes being re-grouped and assigned a new label such as X0. The second segment shows the inferred hierarchies from the instance of the dimension table.

```
Co-Level Attributes:
 -------------------
X0 :  REPORTING_CODE, REPORTING_CODE_DESC

The unique paths in the hierarchy are:
 ------------------------------------
ANZSIC_OCCUPATION_CODE, ANZISC_CODE_DESC, COMPANY_NO
ANZSIC_OCCUPATION_CODE, ANZSIC_OCCUPATION_FLAG, COMPANY_NO
ANZSIC_OCCUPATION_CODE, X0, STRUCT_LEVEL_3, STRUCT_LEVEL_2, STRUCT_LEVEL_1, COMPANY_NO
```

The inferred hierarchies are shown in Figure 4.7.

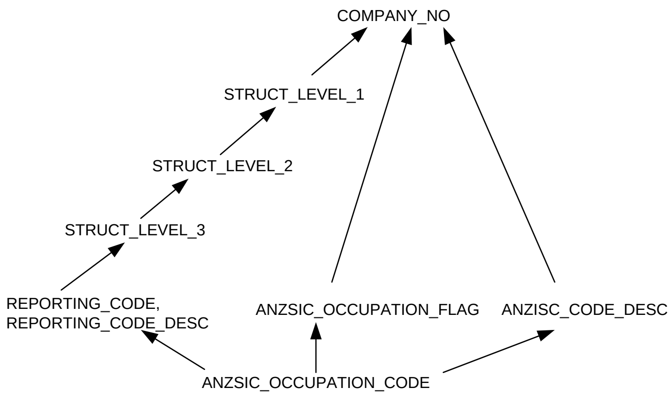


Figure 4.7: Visual representation of the inferred hierarchy from the second experiment.

As we can see from this example, there is a similar problem with ANZSIC_CODE_DESC since it should have same partial order relationships as ANZSIC_OCCUPATION_CODE, and therefore, these two attributes should have been grouped into a single level. It can be seen that the main hierarchy used for aggregation of data in this experiment is the one with the longest path with the remaining paths being in fact spurious.

When inferring hierarchies using data from other companies, the result includes some variations of spurious paths but they share the main path. This leads to our second observation that inferred hierarchies can be compared by a domain expert to identify spurious hierarchies to be excluded them from the integration as they can be restrictive in establishing compatibility. The following shows the output of our algorithms for a different company in dimension table `OCCUPATION`.

```
Co-Level Attributes:
 -------------------
X0 :  COMPANY_NO, ANZSIC_OCCUPATION_FLAG
X1 :  REPORTING_CODE, REPORTING_CODE_DESC

The unique paths in the hierarchy are:
 -------------------------------------
ANZSIC_OCCUPATION_CODE, X0
ANZSIC_OCCUPATION_CODE, ANZISC_CODE_DESC, X0
ANZSIC_OCCUPATION_CODE, X1, STRUCT_LEVEL_3, STRUCT_LEVEL_2, STRUCT_LEVEL_1, X0
```

We also learn from this experiment that when integrating these dimension tables with their matching dimension tables, their instances may need to be restricted to have a fitting hierarchy for different fragments of data. A less attractive alternative option is to exclude some levels from the integration.

These experiments establish that our algorithms are not only effective in inferring hierarchies, they provide additional information that can be used to enforce compatibility. Another application of inferring hierarchies is to establish if correct indexes and data clusters are defined for the dimension table to improve query performance.

4.6 Dealing with Imprecision and Uncertainty of Data

Imprecision in the context of dimensions occurs when null value is used as members of levels of dimension hierarchies to represent the unknown or not applicable. For example, we may not know the ISBN of a book, or the ISBN may be not applicable, because the book may be sold electronically. When inferring aggregation hierarchies, null values can be ignored. Consequently the roll-up relationships are only determined using the non-null values in attribute values.

Uncertainty is however, where probability values are assigned to possible values of an attribute (Burdick et al. 2007). Uncertainty may occur in the fact tables, where for example, members of levels in a StoreLocation dimension are clearly defined but the store name is not recorded at the time of capturing the sales transaction. In such case, members of a higher level such as City are used in place of Store (Burdick et al. 2007). In this case, the store could be any of the stores in that city.

It is not however an effective method to construct a dimension whose members have some degree of uncertainty associated with them. This is best managed by capturing the uncertainty in the fact table. For example, if the diagnosis of a particular illness is uncertain and could be one of many, then the hierarchy of the members between the two levels Diagnosis and Illness is designed such that it describes all possible cases, but when capturing the diagnosis for a given patient, then the value of the diagnosis in the fact table is set to the name of the illness instead (Burdick et al. 2007). Extending OLAP functions to support imprecision and uncertainty is not in the scope of this thesis.

A special form of uncertainty in the context of dimensions could be the presence of multi-valued mappings which may cause double counting. For example, the Country (such as Turkey) may not always have a clear relationship with the Continent. In this case, our proposed Algorithm discards the partial order relationship between Country and Continent. There has been some work on preventing double counting resulting from multi-valued mapping (Burdick et al. 2007).

4.7 Discussion

In this chapter, we provided formal definitions of a dimension, level and aggregation hierarchy in the context of relational multidimensional databases. We proposed for the aggregation hierarchies to be inferred from instances of dimension tables in three steps. In the first step, the partial order of attributes is computed using the cardinality of every pair of attributes. In the second step, transitive partial order relationships are identified and removed. Finally, attributes with bidirectional partial order relationship are grouped into a single level with each remaining attribute being also assigned to a single level. The result may include multiple hierarchies.

Inferring dimension hierarchies can be compared with discovering functional dependencies used to discover key attributes. Our proposed algorithm is less complex as it discovers relationships between single dimension attributes. This is because levels with composite attributes are rare, and using data only to identify such levels potentially increases the number of false positive partial order relationships. Future work is required to use both domain ontologies and data to infer levels with composite attributes.

Existing works use physical metadata, domain ontologies, and UML schemas as sources of information to infer dimension hierarchies. The problem with these approaches is that these may not be available either. Moreover, they are likely to produce false negative partial order relationships. That is, we may find that there is no partial order relationship between some levels when in fact there is such relationship. Using lexical repositories is even more problematic because members are often labeled in abbreviated forms and may include numbers.

Jensen et al. [2004] discover dimension hierarchies from data. This approach requires some SQL test and does not eliminate transitive relationships.

The more representative is the population of a dimension table of its domain, the closer is the inferred hierarchy to the intended schema based hierarchy. Therefore, the inferred hierarchy may not be the same as the schema defined hierarchy. We established the relationship between the two hierarchies as the former subsuming the latter. This implies that there are the same or more roll-up relationships that would need to match between the inferred hierarchies to satisfy the requirements for coherence and consistency.

We explained that using our approach to infer hierarchies will not lead to any false negative partial order relationship, but some discovered partial order relationships may well be false positive. This may result in the test for *coherence* and *consistency* to be false negative which would restrict the integration, that is, only levels involved in the mismatching roll-up relationships would be excluded from the integration. We emphasized, however, that the accurate integration of dimension tables if we decide to do so, is more critical.

As a result, there are two important characteristics of the inferred hierarchies: (i) They can guarantee the summarizability of the integrated data, and (ii) they ensure that the matching between the integrating dimension tables is *coherent* and *consistent*.

We used two real life dimension tables for inferring their hierarchies. These experiments showed effectiveness of our algorithms and that they can provide useful information for data cleansing and enforcing compatibility.

Having inferred aggregation hierarchies, in the next chapter, we use the hierarchies to match instances of dimension tables and enforce strictness.

Chapter 5

Enforcing Strictness: Beyond Instance Matching For Dimensions

> ”Man’s mind, once stretched by a new idea, never regains its original dimensions.”
>
> Oliver Wendell Holmes (1841 - 1935)

In the previous chapters, we addressed inferring dimension hierarchies and schema matching as precursors to automate the integration of data marts. The next step is the matching of instances of dimension tables and relies on the results of the previous steps.

Many approaches have been proposed in the literature for matching instances of relational tables, but they do not address the problem of matching instances of dimension tables specifically. Furthermore, the relationship between inaccurate results from instance matching and the non-strictness problem in the integrated data has been overlooked.

We assume that our original instances of dimension tables are strict, in other words, any case of multi-valued mapping or any special case of single-valued mapping is resolved in each instance before the integration. Having made this assumption, we may still have the problem where the integrated data is not strict. We show in this chapter that in most cases, this problem is related to incorrect cases during the instance matching, and that by enforcing strictness we also reduce false positive cases.

In Section 5.1, we identify cases where non-strictness occurs during the integration of dimension tables resulting in inconsistency in data. In Section 5.2, we motivate the need to resolve inconsistencies after the integration. In Section 5.3, we discuss suitability of matching algorithms that exploit hierarchies for the purpose of matching instances of dimension tables. In Sections 5.4 and 5.6, we propose algorithms to enforce strictness against the integrated result. Experiments described in Section 5.7 on real life data demonstrates the effectiveness of our proposed approach.

5.1 Case Analysis

In this section, we discuss scenarios that result in non-strictness when integrating dimension tables with strict instances. We consider the following cases:

- This case concerns the presence of a multi-valued mapping (which we refer to as *m-mapping*) across integrating dimension tables. This is a true non-strict case. For example, there is only one river called 'Rhine' as a member of a level which contains rivers in Germany, but when integrated with a similar dimension table with a matching level whose members include rivers in Netherlands, the result is no longer strict.

Given that the two matching levels are from the same domain, such cases are in fact rare and mostly occur when the population of at least one of the dimension tables is not fully representative of its domain, in other words they are partially populated. One suggested solution is to discard such tuples [Wijsen, 2006]. In Section 2.5.3, we discussed other approaches to address this problem. The algorithm we propose to enforce strictness results in excluding such tuples.

- There is a variation of *m-mapping* which involves synonym members (i.e. same members labeled differently) being related to different parent members. This is a special case of *m-mapping*, where the data is not strict, even though the roll-up constraints appear to be satisfied. Where such synonym cases are identified correctly, same solution as for multi-valued mapping is applicable. Discovery of false positive match between synonym members must be resolved during instance matching and is outside the scope of this chapter.

- There are *homonym* members relating to different parent members. As we explained in Section 2.2.2, this case does not result in double counting but results in over-estimation of measures for ambiguous members. This is of course, if there is no (false) positive match between them. Where there is a false positive match between them, we treat this case in a similar way to the multi-valued mapping case. An example is where the product category `Keyboard` in an `Electronic` department is falsely matched with `Keyboard` in a `Musical` department. This case in fact accounts for most of the non-strict cases. Discarding such matchings enforces the strictness and reduces false positive cases.

- This is where the perceived non-strict case is in fact caused by false negative match pairs at the parent level. This is a false non-strict case. In this case, the instance matching algorithm fails to identify the matching pair and the integrated result appears to be strict. For example, let us suppose that a product `Prod_x` relates to product categories `Keyboard` and `KBD`, and that the two product categories are in fact the same but the matching algorithm has identified them as being different. Improving the match results to recover missing matching pairs is not in the scope of our work. Discovery of false negative matchings must be resolved during instance matching and is outside the scope of this chapter. Our proposed algorithm to enforce strictness also discards the true positive match between these members (i.e. products labeled `prod_a`). This side effect has a negative impact on the recall. In Section 5.7, we discuss the extent of this side effect.

5.2 Motivating Example

In this section we describe our motivating example used throughout this chapter. It concerns a hypothetical departmental store that sells run out model products it buys from other stores. The store needs to regularly integrate its dimension table `Product` with purchased items in the dimension table `Item`. Manual matching of large dimension tables such as `Product` and `Item` which potentially contain thousands of members with inconsistent labels is not feasible. Figures 5.1 and 5.2 show sample instances of those dimension tables. Each instance conforms to its hierarchy, and therefore, is strict.

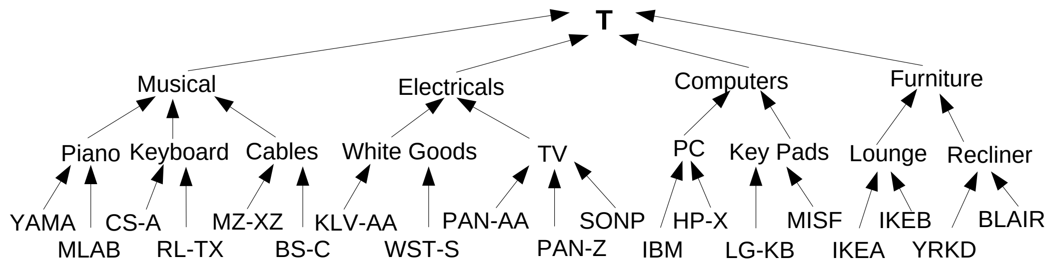


Figure 5.1: An instance of dimension table `Product`.

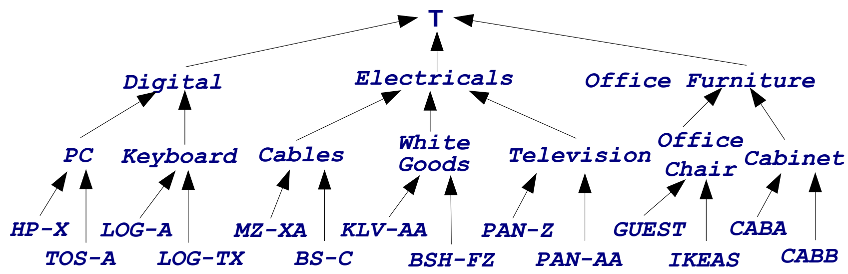


Figure 5.2: An instance of dimension table `Item`.

Figure 5.3 represents the ideal matchings between the two instances as determined by a domain expert. For simplicity, we omit non-matching leaf nodes and sub-trees containing mismatching nodes. Members of levels of dimension table `Item` appear in italicized font. The followings illustrate examples of non-strict cases which we address:

- A product identified as `BS-C` is an electrical cable as well as a musical (audio) cable. This is a case of *m-mapping* and illustrates that a perfect match does not necessarily guarantee strictness. As part of enforcing strictness we will discard such matching. Where the integration is by intersection, this would amount to deletion of inconsistent tuples from the integrated result.

- There are two different product categories both labeled `Keyboard`, one of them is a musical keyboard and the other is a computer keyboard. We will be also discarding the match between these two categories during the enforcing of strictness.

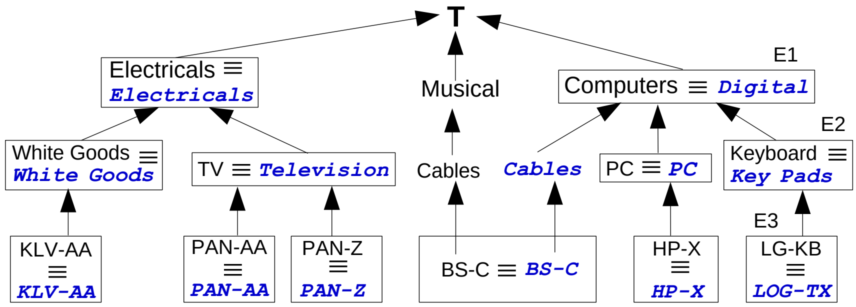


Figure 5.3: Ideal matchings between `Product` *and* `Item`.

5.3 Exploiting Hierarchies for Instance Matching

Apart from the similarity between labels, roll-up relationships between levels provide the most significant clue in finding matching members. For example, if the cities in `United States` and `US` are more similar to one another than to the cities of any other country, then `United States` and `US` are likely to be the same country even though they have very different labels. If majority of cities in the `US` and `AUS` are different, then the two countries are likely to be different and in return, their two `Melbourne` cities are also likely to be different cities, even though, they have the same label and their two countries have similar labels. Therefore, it is intuitive that the similarity between members of the child levels influences the similarity of the members at the parent levels and the similarity between members of the parent levels to influence the similarity of the members at the child levels.

As discussed in Chapter 2, similarity flooding (SF) calculates similarity scores between nodes of two graphs G and G'. The result is the connectivity graph $\hat{G}$ whose nodes represent pairs of matching members. We represent each node in $\hat{G}$ as $(m \in l, m' \in l')$ with an associated similarity score σ. m and m' are suggested matching members and σ is a relative measure of similarity between m and m'. l and l' are matching levels in G and G'.

Similarity flooding is an iterative fixed point computation algorithm that propagates similarity scores between nodes of G and G' through the common edges. It is based on the intuition that nodes of two graphs are similar when their adjacent (i.e. parent and child) nodes are similar. The similarity scores are relative to all other match candidates in the range of 0 to 1. The algorithm stops when the change in similarity scores becomes insignificant. Finally, the best match candidates are selected.

We name the edges (i.e. roll-ups) in G and G' the same to enable the creation of the connectivity graph and to also limit the calculation of the initial string matching to members of matching levels. The suggested matching pairs for our motivating example, as shown in Figure 5.4, are obtained using this algorithm. Members of levels of dimension table `Item` appear in italicized font. Solid rectangles and dashed ovals indicate true positive (TP) and false positive (FP) matches respectively. Rounded scores outside of each node are σ values returned from SF. Note, that with the matching results from similarity flooding, the result is not strict, e.g. `Keyboard` $\equiv$ `Keyboard` rolls-up to `Digital` and `Keyboard`.

5.4 Enforcing Strictness

Instance matching algorithms, in general, can produce false positive matching cases and do not enforce strictness. In this section, we describe the causes of non-strictness with respect

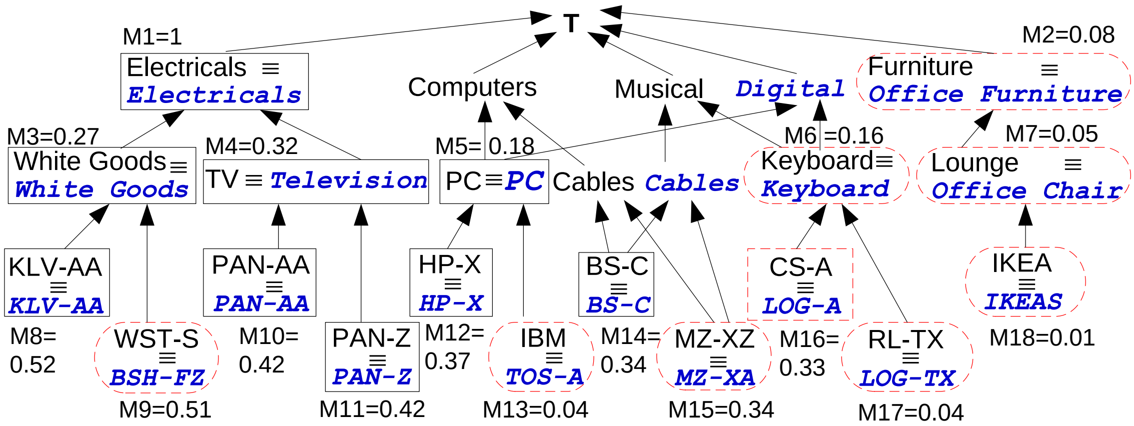


Figure 5.4: Suggested matchings between `Product` *and* `Item`.

to inaccurate matchings and inconsistent data. To illustrate the causes of non-strictness, let us compare the ideal match results in Figure 5.3 with the suggested match results in Figure 5.4:

- The suggested match `Keyboard` ≡ `Keyboard` (i.e. the node labeled as `M6`), a *homonym* case, is false and leads to non-strictness. Similarly, if one of the members in this suggested match was instead labeled `Key Board`, then it would be a false synonym case and it would also lead to non-strictness.

- The suggested match `BS-C` ≡ `BS-C` (`M14`) is true positive and leads to non-strictness.

- The suggested match `WST-S` ≡ `BSH-FZ` (`M9`) is a false positive but does not lead to non-strictness.

- The suggested match `PC` ≡ `PC` (`M5`) is a true positive match, but the false negative match between their respective parent members makes it to be perceived as *non-strict*.

- The suggested match `TV` ≡ `Television` (`M4`) is a true positive match involving *synonym* members and does not lead to non-strictness.

- The matching pair `Cables` $\equiv$ `Cables` is a true negative match involving *homonym* members. It is a special case of single-valued mapping which is discussed in Section 2.2.2. The approach we take is to re-label the members to resolve the ambiguity and address the apparent non-strictness.

In summary, the causes of non-strictness are due to *m-mapping*, or false positive matchings at the parent or child levels.

Next, we explain our main idea of enforcing strictness. We said in the previous section that similarity scores propagate through the levels. The matching pairs at the higher levels are less likely to be false positive than pairs at the lower levels because the number of members in levels decreases as we move up the hierarchy, and the similarity scores at the lower levels propagate to the higher levels. Therefore, the matching pairs at the root level are likely to be more accurate than at any other level.

For these reasons, we start at the root level and assume that the match results at this level are correct. At every level lower and for each matching (child) pair, we expect that their parent members also match, otherwise, we discard the matching child pair. This will discard those matchings that result in *s-mapping* and *m-mapping* cases. This amounts to deletion of inconsistent tuples when integration is by intersection and nullifying of the corresponding attributes when integrating by union.

Algorithm 5.1 follows a top-down iterative process. The input to the algorithm is the connectivity graph $\hat{G}$ (e.g. Figure 5.4) returned from some instance matching algorithm that exploits the (common) hierarchy $H = \{l_1 \equiv l'_1 \preceq l_2 \equiv l'_2, l_2 \equiv l'_2, ...\}$ between matching levels. For simplicity we will use the levels in one of the hierarchies to refer to both matching levels. The output $\hat{T}$ a subgraph of $\hat{G}$ is strict. Lines 8-9 include in $\hat{T}$ a pair of matching members only if their parent members are determined to be matching in the previous iteration. Figure 5.5 shows the matching pairs after applying this algorithm. The scores outside of each node will be described in Section 5.6.

Algorithm 5.1 Enforcing strictness on match results.

Input: Association graph $\hat{G}$, hierarchy H
Output: Strict connectivity graph $\hat{T}$ a subgraph of $\hat{G}$.

1: Insert into $\hat{T}$ nodes in $\hat{G}$ relating to the root level of H
2: **for** each roll-up relationship $l_\alpha \preceq l_\beta$ in H **do**
3:    C = matching child pairs in $\hat{G}$ relating to the level l_α
4:    P = matching parent pairs in $\hat{T}$ relating to the level l_β
5:    **for** each matching pair $n_i = (m_1 \in l_\alpha, m'_1 \in l'_\alpha)$ in C **do**
6:       m_2 = the parent member for m_1
7:       m'_2 = the parent member for m'_1
8:       **if** the pair $(m_2 \in l_\beta, m'_2 \in l'_\beta)$ is in P **then**
9:          Insert n_i into $\hat{T}$ connecting it to its parent.
10:       **end if**
11:    **end for**
12: **end for**
13: Remove childless nodes from $\hat{T}$.

If we have perfect matching between two dimension tables and the result of the integration is not strict, then Algorithm 5.1 would eliminate the *m-mapping* cases as the only possible cases of non-strictness. However, *m-mapping* is not common in the context of integrating strict dimension tables, as it would indicate that the data was not originally strict with the inconsistencies hidden in separate instances.

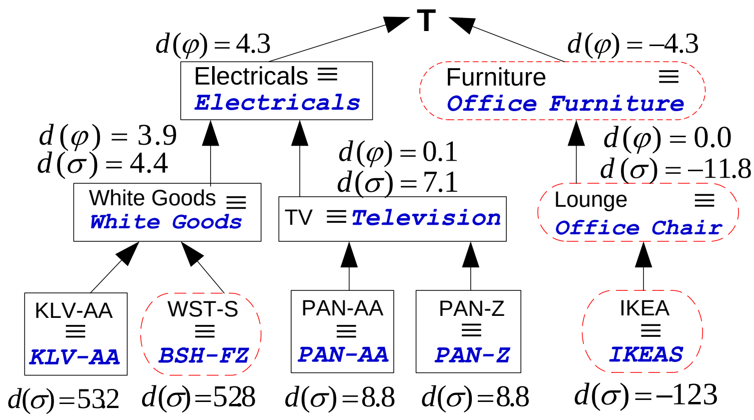


Figure 5.5: Matching pairs from Figure 5.4 after Algorithm 5.1.

5.5 The Effect of Enforcing Strictness on Match Quality

Table 5.1 summarizes the effects of Algorithm 5.1 on the ideal match results. The effects will show in the precision and recall which we will use to measure the performance of our algorithms. Columns 2 and 3 indicate the truth or falsehood of the (mis) matching parent and child pairs. Column 4 describes the effect of the algorithm on the ideal match results. Column 5 gives an example of the case by referring to the nodes in Figures 5.3 and 5.4, where applicable. There is no example applicable to cases involving FN and TN at the parent or child level (except for where both are FN), as they do not appear in the suggested or ideal match results.

Cases 5, 8 and 9 are highlighted with * to indicate a *m-mapping* case. In cases 5, 8 and 12 strictness is not enforced due to a false positive or true negative match at the parent level. These represent false strict cases. In Section 5.6 we propose an algorithm to reduce these cases. In case 6, we are unable to remove the mismatching child pair because strictness is not enforced, again due to the incorrect match at the parent level. In case 9, a *m-mapping*

	Parent Pair	Child Pair	Effect	Example
1	TP	TP	Nil	M3 $\preceq$ M1
2		FP	Nil	M13 $\preceq$ M5
3		TN	Nil	Not applicable
4		FN	Nil	Not applicable
5*	FP	TP	Nil	*See section 5.6*
6		FP	Mismatching child pair not removed	M18 $\preceq$ M2
7		TN	Nil	Not applicable
8*		FN	Nil	*See section 5.6*
9*	TN	TP	True matching child pair removed	Not applicable
10		FP	False matching child pair removed	Not applicable
11		TN	Nil	Not applicable
12		FN	Nil	*See section 5.6*
13	FN	TP	True matching child pair removed	Not applicable
14		FP	False matching child pair removed	Not applicable
15		TN	Nil	Not applicable
16		FN	Nil	E3 $\preceq$ E2

Table 5.1: Impact of Algorithm 5.1 on ideal match results.

case, strictness is enforced at the expense of removing a true matching child pair which reduces the recall. In case 10, strictness is enforced and as a positive side effect, a false positive match is also removed which improves the precision. In case 13, although, the algorithm removes a true positive match, the result becomes strict. This also reduces the recall but also alerts a human expert to a possible false negative match at the parent level. Case 14 is interesting because although the application of the algorithm is not warranted, the net effect is the removal of a false positive match.

To summarize the effects of Algorithm 5.1: (i) strictness is guaranteed, (ii) the precision is improved by reducing the false positives, and (iii) the recall may be reduced. Given the intuition behind the algorithm supported by our findings from experiments in Section 5.7, the improved precision clearly outweighs possible reduction in the recall.

5.6 Reducing False Strictness

In Section 5.5, we saw the three cases (5, 8 and 12) where child members form seemingly strict M:1 roll-up relationships. Matching algorithms tend to be greedy in finding matchings. An example of this case is the pair `Lounge` ≡ `Office Chair` in Figure 5.5. The similarity score for this pair (as shown in Figure 5.4) is very low in comparison with other pairs at the same level and yet the pair is selected as a match. The following explains this match: (i) The matching score for this pair is not incremented by the string similarity between the labels of its members but rather by the similarity score of its (only) child pair `IKEA` ≡ `IKEAS`; (ii) The string similarity between labels of the pair `IKEA` ≡ `IKEAS`, and the missing correct matching member to `IKEA` in `Item`, or the missing correct matching member to `IKEAS` in `Product` helps the pair to be a winning match; (iii) The missing correct match to `Lounge` in `Item`, or the correct match to `Office Chair` in `Product` helps the pair to be a winning match.

Moreover, the pair `Furniture` ≡ `Office Furniture` is the root of a leaner branch with fewer descendants when compared to other pairs at the `Department` level. This is indicative of the lower degree of match between the descendants of the members of this pair.

Given the above observations, our aim is to identify pairs that are selected as matching but more due to the missing correct matching members in the other instance. This problem is more apparent where only a small percentage of members from each instance match. It occurs in SF and is also reported to occur in state of the art duplicate detection algorithms in XML documents in the context of missing data [Pavel and Euzenat, 2004].

These false positive cases can potentially result in false strict cases (5 and 8 in Table 5.1). In Figure 5.5, even if the pair `IKEA` ≡ `IKEAS` were truly matching, we would still want to remove the pair as it would be a *m-mapping* case. However, the false positive match between the parent members would not allow this to occur and, hence, the result would be falsely strict.

In this section, we propose an algorithm to discard matching pairs such as `Lounge` $\equiv$ `Office Chair` described above. Reducing these false positive matchings can potentially reduce occurrences of false strict cases. The algorithm is based on the observations described above, and hence, our heuristic, that if a matching pair has a very low similarity score σ and also its parent pair has a very low *match factor* (described next), then it is likely to be a false positive match.

Match factor measures the strength of a matching pair (m, m') in respect to both, the match degrees for the child and leaf members of m and m'. It is based on the intuition that members of false positive matching pairs are less likely to succeed in having as many number of their descendant members matching.

For a given pair of matching members m and m', the *child match degree* (φ_1) measures the degree of the child members of m having a match with child members of m'. $\hat{T}$ is the connectivity graph after applying Algorithm 5.1.

$$\varphi_1(m, m') = \frac{|\text{ Child nodes of } {\scriptstyle (m,m')} \text{ in } {\scriptstyle \hat{T}} \mid {\scriptstyle \times 2}}{|\text{ Child nodes of } {\scriptstyle m_1 \ in \ G} \mid + \mid \text{Child nodes of } {\scriptstyle m'_1} \text{ in } {\scriptstyle G'} \mid}$$

Similarly, for a given pair of matching members m and m', the *leaf match degree* (φ_2) measures the degree of the leaf members of m having a match with leaf members of m'.

$$\varphi_2(m, m') = \frac{|\text{Leaf nodes of } {\scriptstyle (m,m')} \text{ in } {\scriptstyle \hat{T}} \mid {\scriptstyle \times 2}}{|\text{ Leaf nodes of } {\scriptstyle m_1} \text{ in } {\scriptstyle G} \mid + \mid \text{Leaf nodes of } {\scriptstyle m'_1} \text{ in } {\scriptstyle G'} \mid}$$

The *match factor* is $\varphi = \varphi_1 \times \varphi_2$. The intuition is to reward/penalize the match degree for the child nodes by how successful they are in having their leaf nodes have a match. For example, $\varphi($`Electricals` $\equiv$ `Electricals` $) = 4/5 \times 8/11 = 0.58$ and $\varphi($ `Furniture` $\equiv$ `Office Furniture` $) = 0.5 \times 0.25 = 0.125$.

Following from the observations above, matching pairs involving missing matching members have a much lower σ and φ than other matching pairs at the same level. To determine the threshold for these values, we use the outlier detection method and borrow the method introduced by Knorr and Ng [1998]. They consider a value to be an outlier if its distances (d) from normal distribution is equal to, or more than 3 standard deviations from the mean. For our purpose, where we want to identify much weaker matching pairs, we consider a value to be an outlier, if its distances (d) from normal distribution is 3 standard deviations below the mean. In Figure 5.5, $d(\sigma)$ and $d(\varphi)$ are shown for each pair of matching members.

Algorithm 5.2 follows a bottom-up approach. Again, the input to this algorithms is the connectivity graph $\hat{T}$ returned from Algorithm 5.1. At each level, we remove those matching pairs whose similarity score is an outlier and whose parent pair has a *match factor* that also happens to be an outlier. Similar to Algorithm 5.1, the order of complexity for Algorithm 5.2 is $L \times N$ where L is the number of levels and N is the number of matching pairs at each level. Again, for simplicity we use the levels in one of the hierarchies to refer to both matching levels.

Applying Algorithm 5.2 to our example in Figure 5.5, the matching pair `Lounge` $\equiv$ `Office Chair` is discarded as a false positive match since its matching score, as well as the *match factor* of its parent pair, are both considered outliers. This will first result in re-

Algorithm 5.2 Reducing weak matching pairs.

Input: Association graph $\hat{T}$, hierarchy H
Output: $\hat{S}$ a subgraph of $\hat{T}$

$\hat{S} = \hat{T}$
for each roll-up relationship $l_i \preceq l_{i+1}$ in H **do**
  $C =$ matching pairs from $\hat{T}$ relating to level $= l_i$
  Calculate $d(\sigma)$ for m and m' for each pair of C.
  $P =$ matching pairs from $\hat{T}$ relating to level $= l_{i+1}$
  Calculate $d(\varphi)$ for m and m' for each node of P.
  for each node n_j of C **do**
    if $d(\sigma)$ for n_j is $\leq$ $Threshold$ **then**
      n'' = parent node of n_j in P
      if $d(\varphi)$ for n'' is $\leq$ $Threshold$ **then**
        Remove descendants of n_j from $\hat{S}$.
        Remove n_j from $\hat{S}$.
      end if
    end if
  end for
end for
Remove childless nodes from $\hat{S}$.

moval of its only child pair `Lounge` $\equiv$ `Office Chair` followed by the removal of the pair itself (i.e. `Lounge` $\equiv$ `Office Chair`). Finally, its parent pair `Furniture` $\equiv$ `Office Furniture` is removed since it has no other child. Figure 5.6 shows the result of applying Algorithm 5.2 to the result of Algorithm 5.1 shown in Figure 5.5.

5.7 Experiments

The purpose of this set of experiments is to:

i. measure the performance of SF in matching instances of dimensions with different degrees of noise;

ii. measure the performance of SF in matching instances of dimensions with different de-

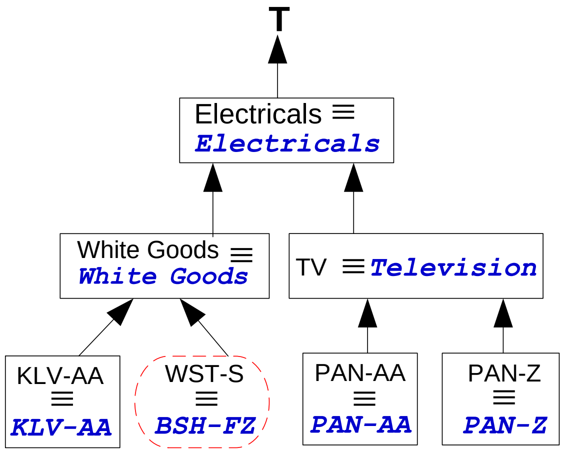


Figure 5.6: Matching pairs from Figure 5.5 after Algorithm 5.2.

grees of missing matching members;

iii. demonstrate that integration of strict (instances of) dimension tables can result in non-strictness;

iv. demonstrate that Algorithm 5.1 can effectively enforce strictness and reduce false positive matches;

v. demonstrate that using instances of dimensions with high volume of mismatching members can increase the number of false positive cases;

vi. demonstrate that where instances of dimensions have a high volume of mismatching members, Algorithm 5.2 can help identify false positive matching cases that could not be identified using Algorithm 5.1;

vii. determine the strength of SF in dealing with multiple and ragged hierarchies.

Source of data: We used the Mondial geographical web data base [May, 1999] with about 4600 distinct members. There are three hierarchies in this data set: (i) $City \preceq Province \preceq Country \preceq Continent$, (ii) $City \preceq Province \preceq Country \preceq GovernmentStyle$, (iii) $City \preceq Sea$. Some cities do not belong to any province but do belong to some countries making the first two hierarchies to be ragged. This condition is present where a member of a level does not have a parent member in its immediate parent level but in an ancestor level.

Target: We used the source as the basis to create an instance of the target dimension table. DirtyXML [Puhlman, 2004] is a tool that is primarily used to produce duplicate elements in XML files for the purpose of duplicate detection. Unfortunately, it could not be used for our purpose because it provided no mechanism to track changes to labels. This is necessary to facilitate the calculation of precision and recall. We used this tool for the purpose of reducing members but developed a program to introduce noise into labels. Similar to DirtyXML, our program randomly applies insertion, deletion, duplication and swapping of characters. More importantly, it tracks changes by recording the labels before and after the changes.

In Section 5.7.1, we measure the performance of SF, and our proposed Algorithms 5.1 and 5.2 when using a single hierarchy. In Section 5.7.2, we measure the performance of SF when using multiple hierarchies.

5.7.1 Using A Single Hierarchy

To avoid the bias that the use of multiple hierarchies may introduce, we use the same single hierarchy ($City \preceq Province \preceq Country \preceq GovernmentStyle$) for all experiments described in this section. Tables 5.2 and 5.3 show precision and recall values obtained from the initial matching (by SF), after applying Algorithm 5.1, and after applying both Algorithms 5.1 and 5.2. Table 5.4 shows F-measure values calculated using the precision and recall values. The highest value for each case is shown in bold.

Each case uses a different degree of reduction resulting in missing matching members, and noise. The larger the reduction, the higher is the degree of missing matching members between the two instances. The experiments are designed to measure the performance of these 3 algorithms in respect to different amounts of noise and missing matching members.

Performance measures for SF: We can see that in cases 1 to 12, where there is a low to medium degree of missing matching members, the precision is impressively high above 0.94. These cases indicate that SF performs extremely well in matching instances even with a large level of noise.

In cases 13 to 16, where the degree of missing matching members is increased to 75%, the precision drops to below 0.84. In cases 13 and 14, the precision remains relatively lower even though there is no or little noise introduced. Nevertheless, precision, and in particular the recall is generally higher where there is less noise. These cases also support our initial finding that instance matching algorithms tend to produce more false positives where there is a high degree of missing matching members between the two instances.

Performance measures for SF + Algorithm 5.1: In all cases, Algorithm 5.1 has consistently reduced the number of false positive cases resulting in higher precision (when compared to before applying the algorithm) except for cases 1 and 2 where the precision has remained almost unchanged. The increase in the precision is a positive side effect of enforcing strictness which is the primary objective of Algorithm 5.1.

			Precision		
Case No	Reduction	Noise	SF	SF+ALG1	SF+ALG1+ALG2
1	00%	00%	0.9879	0.9878	**0.9884**
2	00%	25%	0.9825	0.9813	**0.9836**
3	00%	50%	0.9751	**0.9831**	**0.9831**
4	00%	75%	0.9693	**0.9696**	0.9688
5	25%	00%	0.9820	0.9820	**0.9827**
6	25%	25%	0.9740	0.9798	**0.9805**
7	25%	50%	0.9622	0.9770	**0.9779**
8	25%	75%	0.9519	0.9864	**0.9951**
9	50%	00%	0.9530	0.9585	**0.9589**
10	50%	25%	0.9492	0.9564	**0.9597**
11	50%	50%	0.9718	0.9913	**0.9932**
12	50%	75%	0.9415	0.9488	**0.9520**
13	75%	00%	0.8248	0.8577	**0.9151**
14	75%	25%	0.8333	0.8726	**0.9485**
15	75%	50%	0.8102	0.8774	**0.9022**
16	75%	75%	0.7976	0.8522	**0.8950**

Table 5.2: Precision values for different degrees of noise and missing members.

The recall is slightly reduced in most cases. The main reason for this is that some true positive matchings at the child level are discarded due to false negative matchings at their parent levels. For each pair of tuples across the two instances, a pair of tuples will not match unless all of their attribute values match. Therefore, discarding the match at the child level, when there is no match at the parent level, will not reduce the extent of the data integration any further. Moreover, the report on missing matching parents can provide valuable information to review what could be false negative cases.

The overall F-measure, a reflection of both precision and recall is not greatly impacted. These cases demonstrate that whilst, Algorithm 5.1 enforces the strictness, it can improve the precision by reducing the number of false positive cases. These two important benefits come with the cost of a slight reduction in the recall.

			Recall		
Case No	Reduction	Noise	SF	SF+ALG1	SF+ALG1+ALG2
1	00%	00%	**0.8510**	0.8475	0.8405
2	00%	25%	**0.7606**	**0.7606**	0.7424
3	00%	50%	**0.8651**	0.8476	0.8453
4	00%	75%	**0.6221**	0.5923	0.5514
5	25%	00%	**0.7496**	0.7455	0.7289
6	25%	25%	**0.9197**	0.9057	0.8926
7	25%	50%	**0.8538**	0.8339	0.8261
8	25%	75%	**0.7603**	0.7221	0.5650
9	50%	00%	**0.8908**	0.8881	0.8806
10	50%	25%	**0.8376**	0.8349	0.8091
11	50%	50%	**0.8000**	0.7782	0.6975
12	50%	75%	**0.7156**	0.6719	0.6650
13	75%	00%	**0.9648**	0.9606	0.9586
14	75%	25%	**0.9420**	0.9359	0.9228
15	75%	50%	**0.8822**	0.8719	0.8574
16	75%	75%	0.7976	**0.8057**	0.7591

Table 5.3: Recall values for different degrees of noise and missing members.

Performance measures for SF + Algorithm 5.1 + Algorithm 5.2: The precision has consistently improved for the vast majority of cases. As for recall, there are slight reductions, largely where there is a greater level of noise. This is also visible from the F measures in Table 5.4.

Cases 13 to 16 involve a high degree of missing match candidates between the two instances. A good sign for the presence of such a condition is that we have a relatively lower precision to start with. In these cases, where the precision resulting from the previous two methods were around 0.80, the precision has significantly improved to around 0.90.

Therefore, consistent with our hypothesis, Algorithm 5.2 improves the precision by reducing even further, the number of false positive cases which Algorithm 5.1 fails to identify. Again, there is a slight reduction in the recall value.

			F-Measure		
Case No	Reduction	Noise	SF	SF+ALG1	SF+ALG1+ALG2
1	00%	00%	**0.9144**	0.9123	0.9085
2	00%	25%	**0.8574**	0.8570	0.8461
3	00%	50%	**0.9168**	0.9103	0.9090
4	00%	75%	**0.7578**	0.7354	0.7028
5	25%	00%	**0.8502**	0.8476	0.8370
6	25%	25%	**0.9461**	0.9413	0.9345
7	25%	50%	0.9048	**0.9779**	0.8956
8	25%	75%	**0.8454**	0.8338	0.7208
9	50%	00%	0.9209	**0.9220**	0.9181
10	50%	25%	0.8899	**0.8915**	0.8780
11	50%	50%	**0.8776**	0.8719	0.8195
12	50%	75%	**0.8132**	0.7867	0.7830
13	75%	00%	0.8893	0.9062	**0.9363**
14	75%	25%	0.8843	0.9031	**0.9355**
15	75%	50%	0.8447	0.8595	**0.8792**
16	75%	75%	0.7976	**0.8282**	0.8215

Table 5.4: F-Measure results for different degrees of noise and missing members.

5.7.2 Using Multiple Hierarchies

In this part of the experiment, we use all three hierarchies listed earlier in Section 5.7, but use the SF algorithm only. This is because Algorithms 5.1 and 5.2 use a single hierarchy; however, both algorithms can be applied in multiple iterations, each time, using a different hierarchy.

With respect to Algorithm 5.1, to ensure that the instance of the dimension table is strict for all hierarchies, in each iteration, the input $\hat{G}$ is set to the output $\hat{T}$ from the previous iteration. With respect to Algorithm 5.2, it is possible that the same matching pair is removed when using one hierarchy but, not when using another. A simple approach to unify the result is to finalize the removal of a match between a pair of members, only if it is removed by using every hierarchy in which the owning level appears.

Performance measures for SF: The sample we used to measure SF's strength in dealing with multiple hierarchies included 25% reduction and noise in both instances. The precision, recall and F-measures were 0.9790, 0.7031 and 0.8409 respectively. The precision returned from SF is very close to being perfect, but not the recall. The reason for the poor recall in this case can be explained by the fact that the similarity score for a matching child pair is much more likely to be distributed into more than one parent pair and, hence, the number of true positives is lower.

We then tried the same sample, but using a single hierarchy ($City \preceq Province \preceq Country \preceq GovernmentStyle$). The precision, recall and F-measures using a single hierarchy are 0.9740, 0.9197 and 0.9461 respectively. Whilst, the precision remains the same, the recall has improved significantly. The comparison indicates that SF does not perform as well with multiple and ragged hierarchies as with single hierarchies.

5.8 Discussion

In this chapter we addressed the problem of enforcing strictness in integration of originally strict dimension tables, and also thereby, reduce the number of false positive cases during the instance matching. To the best of our knowledge, there is no previous algorithm specifically for matching instances of dimension tables that exploits dimension hierarchies. Moreover, current research has overlooked enforcing strictness against the integrated result and the relation between non-strictness and the accuracy of instance matching results. Through our experiment, we have shown the effectiveness of similarity flooding in matching instances of dimension tables and its weakness in dealing with multiple hierarchies.

We proposed an algorithm which enforces strictness against the integrated data from strict dimension tables and also reduces the number of false positive cases. It discards any matching child pair with more than one parent matching pair in the connectivity graph that is created from the graph representation of the two instances. This algorithm exploits the fact that the inconsistencies resulting from integration of strict dimension tables are more likely to be the result of false positive cases during the initial instance matching.

We proposed a second algorithm that can help discard false positive matchings that could not be discarded using the first algorithm. This algorithm is designed to address the problem associated with instances that have high degree of missing matching members.

In our experiments, we used a real life geographical web database to show the effectiveness of our approach and algorithms. We showed that in presence of noise in data, our first algorithm is effective in improving the precision with very limited impact on recall.

We also showed that our second algorithm is also effective in improving the precision where there is a large volume of missing matching members from either of instances. A side effect of this algorithm is, however, that the recall value is mildly reduced where there is a low level volume of missing matching members with some noise introduced. Future work is required to improve the match factor used in this algorithm to reduce the impact on the recall value. Moreover, the similarity flooding algorithm could be also enhanced to improve its match quality where the instances include multiple hierarchies.

Chapter 6

Extending the Scope of Integration

> ”What counts in making a happy marriage is not so much how compatible you are, but how you deal with incompatibility.”
>
> Leo Nikolaevich Tolstoy (1828 - 1910)

One of the problems we face with integrating heterogeneous data marts is that they have some common but not identical information. In Section 2.4, we described conformity as well as compatibility as the requirements for the integration of dimensions. These require dimensions to be similar in terms of their schemas, hierarchies and instances. In Chapters 3, 4, and 5, we described our approach for matching data mart schemas, inferring aggregation hierarchies and matching instances of dimensions.

The schema matching approach discussed in Chapter 3 returns matching dimension attributes irrespective of their hierarchies or their data. The starting maximal subset of compatible dimensions (X) to be integrated is limited to these matching dimension attributes.

In the next step, aggregation hierarchies are inferred using the approach described in this chapter from matching dimensions identified during the schema matching process. Matching hierarchies are those with matching levels consisting of matching dimension attributes (X). Any dimension attribute not included in the matching levels are excluded from X.

During the next and final step, using the hierarchies inferred, we match instances of the corresponding dimensions using the approach described in this chapter. We then apply Algorithm 5.1 to enforce the strictness, and Algorithm 5.2, if there is a prior knowledge that there is a large volume of mismatching members between the two instances.

By this time, we have a final maximal subset of compatible dimension attributes to form the basis of the integration in terms of schema and data. During the course of these steps, we had to however, exclude some dimension attributes. The purpose of this chapter is to maximize the scope of the integration by salvaging as many dimension attributes that were excluded because they did not fit into the common hierarchy. We see two challenges in meeting this objective: (i) to enforce compatibility between some of dimension attributes that do not fit into the common hierarchy and thereby extend X; (ii) relate X to the exclusive non compatible dimension attributes in each of the original data marts.

Next, we re-visit the requirements for dimension compatibility explained in Section 4.1:

- The matching between some levels may be coherent and consistent but not sound. It is more difficult to have soundness in particular when dealing with heterogeneous dimension tables. This is where the requirement for soundness can be restrictive.
- The existing drill-across operation returns the common data only. It ignores the data related to levels that have no match, or their matchings are not coherent and con-

sistent. This is not effective when users need to link the common data back to the exclusive data in each data mart.

In this chapter, the problems above are investigated, and the following contributions are made:

- In order to maximize the scope of the integration, we relax the requirements for compatible dimension tables by excluding the requirement for soundness.
- We introduce measures to quantify the loss of data in respect to levels which do not have a sound matching. These are used in identifying lossless fragments of the combined data by applying OLAP operations such as slicing, dicing and roll-up guided by those measures.
- We extend the navigation operation drill-across to return the data related to exclusive levels in original data marts.
- We propose an extension to pivot tables to support the extended result of drill-across.

In Section 6.1, the problems addressed in this chapter are further elaborated. In Section 6.2, a less restrictive requirement for integrating dimension tables is proposed. In Sections 6.3, 6.4, and 6.5, we introduce methods for measuring the loss of data. In Section 6.7, we describe the extension to drill-across. In Section 6.8, we discuss extending pivot tables (at a conceptual level) to support the extended drill-across.

6.1 Motivation

Organizations often end up with a large number of data marts over similar subject areas which need to be consolidated [Business Objects and Teradata, 2007]. Their consolidation enables users to benefit from combined related information and also reduces the need for building new data marts. Moreover, there are data marts from external sources, for instance, the Bureau of Statistics and agencies collecting marketing information, which are of interest and when combined, can add value to local data marts.

Consider the Star schemas in Figures 6.1 and 6.2. The matching between the two `Time` dimension tables as well as the matching between `Product` and `Item` dimension tables are coherent and consistent over all of their levels, whereas, `Invoice` and `Accessory` are exclusive to their data marts. The two fact tables have their own exclusive measures.

The first problem highlighted in the preamble to this chapter is that even though the matching between the two `Product` and `Item` dimension tables is coherent and consistent, it is not considered a perfect matching since the matchings between their levels are not sound. This can be seen from their instances in Tables 6.1 and 6.2, where for example, the member `Musical` in the level `Department` in dimension table `Product`, does not appear in its matching level `Area` in dimension table `Item`.

Table 6.1 shows the same instance of the `Product` dimension table as in Figure 5.1 after re-labeling of some members to resolve the ambiguities for synonym and homonym members. Similarly, Table 6.2 shows the same instance of the `Item` dimension table as in Figure 5.2 after making similar changes.

We assume that there is a coherent and consistent matching between all matching lev-

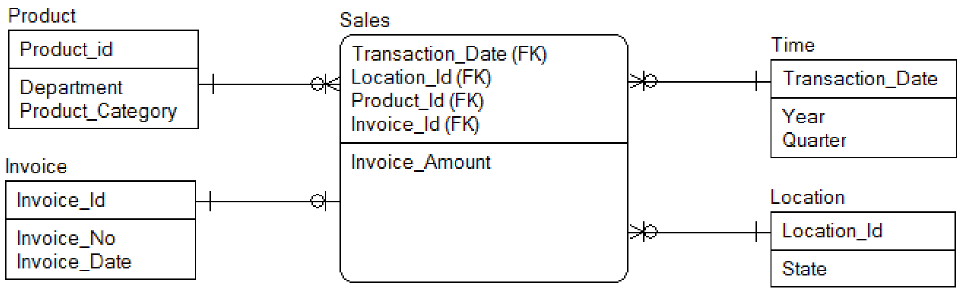


Figure 6.1: Star schema for `Sales` *data mart.*

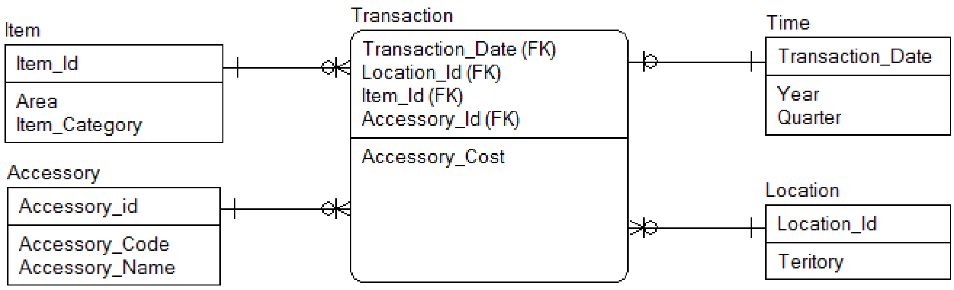


Figure 6.2: Star schema for `Transaction` *data mart.*

els of `Product` and `Item`, between `Time` and `Time`, and between `Location` and `Location` dimension tables.

Absence of soundness results in loss of data during the integration. The loss is not, however, uniform with respect to each data mart. In order to maximize the scope of the integrated data, we propose to relax the requirement for compatibility by excluding soundness. At the same time, several measures are provided to quantify the loss resulting from the absence of soundness. By quantifying the loss for all levels and their combinations, a user is able to benefit from partial integration by locating lossless fragments of data through OLAP operations. For example, if a user is interested in combined data from Figures 6.1 and 6.2 for all `Product_Category/Item_Category` but only only for the member `Computers/Digital` of

Department	Product_Category	Product_Id
Musical	Piano	YAMA
Musical	Piano	MLAB
Musical	Musical Keyboard	CS-A
Musical	Musical Keyboard	RL-TX
Musical	Musical Cables	MZ-XZ
Musical	Musical Cables	Musical BS-C
Electricals	White Goods	KLV-AA
Electricals	White Goods	WST-S
Electricals	Television/TV	PAN-AA
Electricals	Television/TV	PAN-Z
Electricals	Television/TV	SONP
Computers/Digital	PC	IBM
Computers/Digital	PC	HP-X
Computers/Digital	Digital Keyboard	LOG-TX/LG-KB
Computers/Digital	Digital Keyboard	MISF
Furniture	Lounge	IKEA
Furniture	Lounge	IKEB
Furniture	Recliner	YRKD
Furniture	Recliner	BLAIR

Table 6.1: An instance of dimension table Product *in* Sales *data mart.*

Department	Item_Category	Item_Id
Computers/Digital	PC	HP-X
Computers/Digital	PC	TOS-A
Computers/Digital	Digital Keyboard	LOG-A
Computers/Digital	Digital Keyboard	LOG-TX/LG-KB
Electricals	Electrical Cables	MZ-XA
Electricals	Electrical Cables	BS-C
Electricals	White Goods	KLV-AA
Electricals	White Goods	BSH-FZ
Electricals	Television/TV	PAN-Z
Electricals	Television/TV	PAN-AA
Office Furniture	Office Chair	GUEST
Office Furniture	Office Chair	IKEAS
Office Furniture	Cabinet	CABA
Office Furniture	Cabinet	CABB

Table 6.2: An instance of dimension table Item *in* Transaction *data mart.*

the level Department/Area, then there is no loss.

As for the second problem, it is often necessary to relate the common data back to the data related to non-compatible dimensions in the original data marts. For example, once we know the total Invoice_Amt and Accessory_Cost for all matching states and product/items, then we would have to refer to the original sources when we need to know their respective invoice and accessory details. This is obviously a tedious manual task. To overcome this problem, the existing operation of drill-across is extended to also return the data related to levels that have no matching, or their matchings are not coherent or consistent. We also propose to extend pivot tables to support the extended operation of drill-across.

6.2 Non-Compatible but Combinable Dimension Tables

Definitions for perfect matching dimensions, and μ-compatible dimensions were explained in Section 2.4.2. Once again, Torlone [2008] defines two dimensions d_1 and d_2 as being *μ-compatible*, if there are lossless expressions E_1 and E_2 over dimensions d_1 and d_2 such that μ is a perfect matching. This up-front use of a single pair of expressions to enforce soundness as Torlone suggests, is not however, sufficient to effectively exploit the common data.

It is important to note that the loss resulting from the absence of soundness may occur with respect to only one of the data marts or both. A user may wish to continue with the integration despite the loss in another data mart, and therefore a perfect matching may not be necessary. Also, it is possible that the data marts use *factless facts* (i.e. fact tables that have no measure) [Kimball and Ross, 2002], in which case, there would be no need for aggregated measures. But more importantly, there may be many different expressions that make the matching between levels of two dimensions to be sound.

We propose to remove soundness from the requirements of compatibility, and at the same time introduce measures to quantify the loss and, thereby, allow users decide where the integration is meaningful. By providing the loss measures to users, they are able to discover lossless expressions which may vary for different fragments of data.

To be able to use compatibility in the context of relational implementation of multidimensional data, we make the same assumptions as in Section 4.1. Again, for simplicity but without loss of generality, it is assumed that each level of any dimension table is associated with a single attribute, and therefor, we can use levels to refer to attributes.

Definition 6.1. *Two levels l_1 and l_2 are combinable if, and only if, they match and their matching is coherent and consistent.*

Definition 6.2. *Two dimension tables D_1 and D_2 are combinable if the matching μ over their matching levels is combinable.*

This requirement does not require the matching levels to have identical members. Therefore, every pair of compatible dimension tables is also combinable. However, not every pair of combinable dimension tables is compatible.

Relaxing the requirement for compatibility means that we need to identify where the loss of data occurs. In the next section, several measures are introduced to calculate the loss resulting from integration of combinable (but not compatible) levels.

In the next section, several measures are introduced to calculate the loss resulting from integration of combinable (but not compatible) levels. The methods defined in the next section rely on the following information obtained during the instance matching in Chapter 5: (i) the number of distinct instances of each dimension attribute from each dimension, and (ii) the number of matching distinct instances between each pair of dimension attributes.

6.3 Absolute Loss Ratio

The *absolute loss ratio* is calculated for a pair of combinable levels. It measures the degree of mismatch between members of two such levels. It is applicable to where there is a need to minimize the loss regardless of whether members have any corresponding fact. For example, inclusion of all employees from an `Employee` dimension table which represent the organization hierarchy is required for browsing of the complete employee hierarchy regardless of whether some employees have made any sales or not.

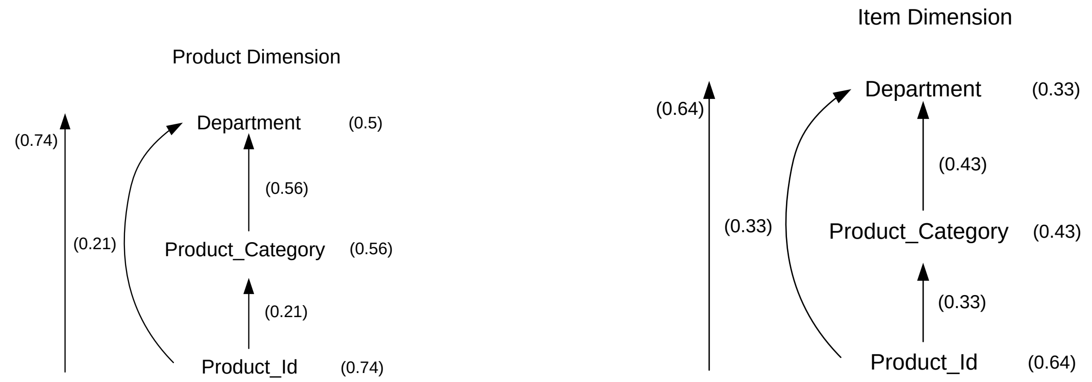
It is calculated by dividing the number of distinct matching members of the two levels divided over the number of distinct members of the level belonging to the data mart for which the loss is calculated. The loss measures are not symmetrical, they are relative to each data mart. The intuition is that the user may be concerned with the loss in only one of the data marts.

The following calculates the absolute loss ratio Δ_{DM} for a pair of combinable levels l_i in D and l'_j in D', with respect to data mart DM.

$$\Delta_{DM}(D(A_i : l_i), D'(A'_j : l'_j)) = 1 - \frac{\left|\pi_{A_i}(D) \cap \pi_{A'_j}(D')\right|}{|\pi_{A_i}(D)|} \tag{6.1}$$

Based on the instances in Tables 6.1 and 6.2, Δ_{Sales}(Department,Area) is 0.5 (i.e. $1 - 2/4$), and 0.33 (i.e. $1 - 2/3$) for $\Delta_{\text{Transaction}}$(Department,Area). In other words, 33% of the members of `Department` in the `Sales` data mart have no match in the level `Area` of `Transaction`.

Pre-calculation of loss ratios for all levels and their combinations is useful for consideration during the integration. Figure 6.3 shows the absolute ratios calculated for all combinations of levels with respect to the `Sales` data mart shown in Figure 6.3(a), and with respect to `Transaction` data mart shown in Figure 6.3(b), and based on their instances in Tables 6.1 and 6.2.



(a) Loss ratios for `Product` dimension in `Sales` data mart.

(b) Loss ratios for `Item` dimension in `Transaction` data mart.

Figure 6.3: Absolute loss ratios for levels of `Product` *and* `Item` *dimension tables.*

6.4 Relative Loss Ratio

Not all members of levels have a corresponding fact in their fact table. The *relative loss ratio* excludes those tuples of a dimension table that do not refer to any tuple in the fact table. Therefore, this ratio is either the same or less than the absolute loss ratio for the same pair of levels.

Unlike absolute loss ratio where users are interested in the complete set of members, relative loss ratio is applicable to where we are more interested in the measures than matching members of levels. The following relational algebra shows calculation of relative loss ratio (δ_{DM}) with respect to a data mart (DM). F and F' are fact tables to which D and D' refer to respectively.

$$\delta_{DM}(D(A_i : l_i), D'(A' : l'_j)) = 1 - \frac{\left|\pi_{A_i}(D \bowtie F) \cap \pi_{A'_j}(D' \bowtie F')\right|}{|\pi_{A_i}(D \bowtie F)|}$$

Consider Tables 6.3 and 6.4 as some instances of the two schemas in Figures 6.1 and 6.2. For simplicity, `Sales` data mart (represented as data cube $C1$) is summarized over `State`, `Product_Category` and `Invoice_No`, and `Transaction` data mart (represented as data cube $C2$) is summarized over `State`, `Item_Category` and `Accessory_Code`. It is assumed that there were no sales made for products in the state of `Vic` and, hence, they do not appear in `Sales` data mart. The relative loss ratios calculated for the two combinable levels (`State, Area`) and (`Product_Category,Item_Category`), and their combinations are shown below in Figure 6.4. No partial order relationship is assumed between these two levels.

(a) Relative Loss ratios for some combinable levels in `Sales` data mart. (b) Relative Loss ratios for some combinable levels in `Transaction` data mart.

Figure 6.4: Relative loss ratios for some combinable levels from `Sales` *and* `Transaction` *data marts.*

Compared to absolute loss ratio (shown in Figure 6.3), the relative loss for (`Product_Category,Item_Category`) is less for the same levels, and there is no loss for (`State,Area`) with respect to either of the two data marts.

State	Product_Category	Invoice_No	Invoice_Amt
NSW	White Goods	INV0077	800
NSW	White Goods	INV0088	900
NSW	Television/TV	INV0099	1200
NSW	Television/TV	INV0012	1400
NSW	Television/TV	INV0014	2100
SA	PC	INV0016	1100
SA	PC	INV0018	1200
SA	Digital Keyboard	INV0032	1300
SA	Digital Keyboard	INV0034	3100
NT	Lounge	INV0036	4200
NT	Lounge	INV0038	1900
NT	Recliner	INV0042	1800
NT	Recliner	INV0044	600

Table 6.3: `Sales` *data mart, an instance of the schema in Figure 6.1.*

State	Item_Category	Accessory_Code	Accessory_Cost
SA	PC	A#0010	150
SA	PC	A#0022	50
SA	PC	A#0023	55
SA	Digital Keyboard	A#0044	70
SA	Digital Keyboard	A#0056	60
NSW	Electrical Cables	A#0060	10
NSW	Electrical Cables	A#0025	100
NSW	White Goods	A#0035	50
NSW	White Goods	A#0015	20
NSW	Television/TV	A#0040	60
NSW	Television/TV	A#0090	70
WA	Office Chair	A#0011	10
WA	Office Chair	A#0012	10
NT	Cabinet	A#0019	20
NT	Cabinet	A#0077	30

Table 6.4: `Transaction` *data mart, an instance of the schema in Figure 6.1.*

6.5 Constrained Loss Ratio

If there is still unacceptable loss using the relative loss ratios, then we can consider customized lossless expressions for different levels. The loss ratio calculated after applying such expressions is a *constrained loss ratio* (denoted by χ_{DM}), and can be based on the absolute or relative loss ratio. For example, members of `Quarter` in `Sales` may be `Q1,Q2,Q3,Q4`, but `1 2,3,4` as members of the same level in `Transaction` data mart. In this case, a constraint such as $\sigma_{(\text{substring(Sales.Time.Quarter,2,1)})}(\text{Sales.Time})$ can eliminate the loss.

A similar application of the constrained loss ratio is to use expressions that align the domains of levels and, thereby, reduce their loss ratio. For example, given the level `PostCode`, and the level `Locality` which includes a combinations of city and post code, we can have an expression that derives `PostCode` from `Locality` making the two levels to have a sound matching.

6.6 Exploiting Dimension Hierarchies for Calculation of Loss Ratios

In calculating the loss ratio for all levels and dimensions, we can exploit dimension hierarchies to save in calculations.

Lemma 6.1. *Given the coherent and consistent matching μ between the two dimension tables D_1 and D_2 over their matching levels $L = l_1, l_2, ..., l_n$ and $L' = l'_1, l'_2, ..., l'_n$, the loss ratios for the two dimension tables using all of their combinable levels from the root of their hierarchies down to the matching levels l_i and l_j is equal to the loss ratios for l_i and l_j.*

Proof. There are two scenarios to consider:

- There are members of l_i and l_j that do not match, but their parent members at some levels of L and L' do match. This case will not affect the result since the mismatch between l_i and l_j is taken into consideration anyhow.
- There are members of l_i and l_j that do match but their parent members at some levels of L and L' do not match. This case is not possible, otherwise, the matching between the two levels will not be consistent. □

The two levels l_i and l_j may be base levels in which case the loss for these levels is the same as the loss for their respective dimension tables. If we roll-up measures to a higher level, then it is considered the new base level. This will give us a loss ratio that is bound to be less than or equal to the loss ratio at a lower level. This can be easily seen by the proof of Lemma 6.1. By doing so, we raise the granularity of data, but we are more likely to have a lossless integration.

It is also possible to calculate loss ratio for a combination of levels between which there may or may not be a partial order relationship. For example, levels from `Product/Item` and `Location` dimension tables do not necessarily have a partial order relationship between them. In this case, where in the formula, we calculate the number of common values of attributes, we require that all of the values of matching attributes match. For example, using `Department`, `Product_Category` and `Product_Id`, the absolute loss ratio is 0.74 (that is $1 - 5/19$) for the `Sales` data mart and 0.64 (that is $1 - 5/14$) for the `Transaction` data mart.

The above is also applicable to situations where the levels in the source or target data marts belong to different dimension tables. Calculation of loss ratios at each level of a hierarchy as well as, for different combination of levels, provides users with the information they need to perform the slice, dice and roll-up operations to have a potentially lossless integration.

Whilst, absolute and relative loss ratios can be used before and during the integration and visualization, the expressions used to calculate constrained loss ratios are more easily discovered by using OLAP operations during the visualization. In Section 6.8.1, it is shown how loss ratios can be included in the visualization of the integrated data to guide these operations.

6.7 Extending Drill-Across to Non-Combinable Levels

The main motivation for extending drill-across is to extend the data analysis space by including the data corresponding to the non-combinable levels and linking them to the common data. The non-compatible levels are those whose attributes were found to be mismatching during the schema matching, or were found to be causing incoherence for the inferred hierarchies. Consider $C1$, tuples of a fact table F that refers to p dimension tables and is summarized over s levels, with its first k attributes corresponding to combinable levels. F has i measures (m):

$C1 = \gamma_{(A_1,...,A_k,A_{k+1},...,A_s),Sum(m_1,...,m_i)}(D_1 \bowtie F, ...D_p \bowtie F)$.

The symbol γ denotes summarization. Similarly, $C2$ represents tuples of a fact table F' with q dimension tables summarized over t levels, with its first k attributes corresponding to combinable levels. F' has j measures (n):

$C2 = \gamma_{(A'_1,...,A'_k,A'_{k+1},...,A'_t),Sum(n_1,...,n_j)}(D'_1 \bowtie F, ...D'_q \bowtie F')$.

Resulting from drill-across between the two data marts owning F and F', is the integrated fact table X whose tuples are obtained through a natural join using the combinable levels: $X = \pi_{A_1,...,A_k,Sum(m_1,...,m_i,n_1,...,n_j)}\sigma_{A_1=A'_1,...,A_k=A'_k}(C1 \bowtie C2)$.

We propose to extend drill-across to also return $C1'$ and $C2'$ which are subsets of $C1$ and $C2$, being restricted to matching members of combinable levels, and summarized over their non-combinable levels: $C1' = \gamma_{(A_{k+1},...,A_s),Sum(m_1,...,m_i)}$ and $C2' = \gamma_{(A'_{k+1},...,A'_t),Sum(n_1,...,n_j)}$.

Figure 6.5 is a visualization of drill-across between the two data marts represented as data cubes $C1$ and $C2$. The result X includes the combined measures summarized over the combinable levels (`State /Area`) and (`Product_Category /Item_Category`).

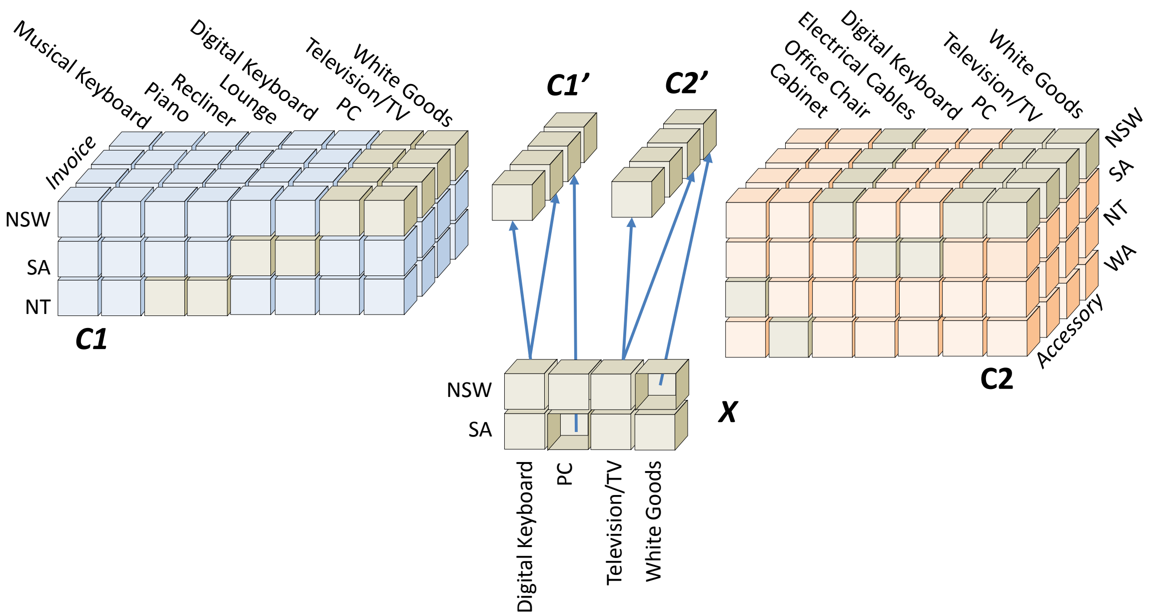


Figure 6.5: Extending drill-across.

For simplicity, let us assume that the matching members of the k combinable levels are in the first g tuples of $X = \{x_1, ..., x_g\}$ being $x_1 =< v_{1,1}, ..., v_{1,k} >$, ..., $x_g =< v_{n,1}, ..., v_{g,k} >$, where v denotes value of the dimension attribute, with its first subscript referring to the attribute, and its second subscript referring to the value of the attribute. Each tuple in X relates to some tuples in $C1'$, and some tuples in $C2'$, that is:

$x_1 : \pi_{A_{k+1},...,A_s}, Sum(m_1, ..., m_i)\ \ \sigma_{A_1=v_{1,1},...,A_k=v_{1,k}}$, ...,

$x_g : \pi_{A_{k+1},...,A_s}, Sum(m_1, ..., m_i)\ \ \sigma_{A_1=v_{g,1},...,A_k=v_{g,k}}$ and

$x_1 : \pi_{A'_{k+1},...,A'_t}, Sum(n_1, ..., n_j)\ \ \sigma_{A'_1=v_{1,1},...,A'_k=v_{1,k}}$, ...,

$x_g : \pi_{A'_{k+1},...,A'_t}, Sum(n_1, ..., n_j)\ \ \sigma_{A'_1=v_{g,1},...,A'_k=v_{g,k}}$

The symbol ":" is means "relates to".

Example 6.1. *Given the sample data in Tables 6.1,6.2, the common data X is:*

State/Area	Product_Category/Item_Category	Invoice_Amt	Accessory_Cost
NSW	Television/TV	4700	130
NSW	White Goods	1700	70
SA	PC	2300	255
SA	Digital Keyboard	4400	130

Table 6.5: Data related to combinable levels.

A given tuple $< SA, PC >$ in the combined data area (X) relates to the following tuples from $C1'$ representing the data related to a non-combinable level `Invoice_No` *from the* `Sales` *data mart, and to $C2'$ the data related to a non-combinable level* `Accessory_Code` *in the* `Transaction` *data mart:*

Invoice_No	Invoice_Amt
INV0016	1100
INV0018	1200

Accessory_Code	Accessory_Cost
A#0010	150
A#0022	50
A#0023	55

Table 6.6: Data related to non-combinable levels.

6.8 Visualizing the Extended Drill-Across

The benefit of extending drill-across is only realized if the device for visualizing multidimensional data can support it. In this section, we describe at a conceptual level how these three related data, namely the common data X and the data related to the non-combinable levels (i.e. $C1'$ and $C2'$), can be shown using pivot tables.

A pivot table is a tabular representation of multidimensional data. Although, more flexible data structures such as trees would be more suitable to visualize multidimensional data [Vinnik and Mansmann, 2006], for simplicity, we use pivot tables to show X, $C1'$ and $C2'$.

We propose that the visualization of multidimensional data using pivot tables includes a Parent Pivot Table (PPT) corresponding to X, and multiple Child Pivot Tables corresponding to sub-cubes of $C1'$ and $C2'$. For each tuple in the PPT, there are one, or more tuples in each of the CPTs. In this respect, there is a nested relation between the tuples represented by the parent and child pivot tables.

Figure 6.6 shows a conceptual layout of nested pivot tables related to the three areas of data. Highlighted areas show one tuple from the common data (X) linked to some tuples from $C1'$ and $C2'$.

PPT (X)

State /Area	Product Category /Item Category	Invoice Amt	Accessory Cost
NSW	White Goods	1700	70
	Television - TV	4700	130
	Total	6400	200
SA	PC	2300	255
	Digital Keyboard	4400	130
	Total	6700	385
Total		13100	585

CPTs ($C1'_{1,...}$)

Invoice No	Invoice Amt
INV0077	800
INV0088	900
Total	1700

Invoice No	Invoice Amt
INV0099	1200
INV0012	1400
INV0014	2100
Total	4700
	6400

Invoice No	Invoice Amt
INV0016	1100
INV0018	1200
Total	2300

Invoice No	Invoice Amt
INV0032	1300
INV0034	3100
Total	4400
	6700
	13100

CPTs ($C2'_{1,...}$)

Accessory Code	Accessory Cost
A#0035	50
A#0015	20
Total	70

Accessory Code	Accessory Cost
A#0040	60
A#0090	70
Total	130
	200

Accessory Code	Accessory Cost
A#0010	150
A#0022	50
A#0023	55
Total	255

Accessory Code	Accessory Cost
A#0044	70
A#0056	60
Total	130
	385
	585

Figure 6.6: Nesting pivot tables.

OLAP operations such as slicing, dicing, roll-up, and roll-in against the PPT will require re-grouping of data in CPTs. Any OLAP operation against the CPT does not affect the PPT, however, the same operation has to be applied against the remaining sibling CPTs.

6.8.1 Inclusion of Loss Ratio in Visualization of Multidimensional Data

Inclusion of loss ratio during the visualization of integrated results enables users to interactively select lossless fragments of data through OLAP operations roll-up, slice and dice.

For example, Figure 6.7 shows relative loss ratios for `State` and Product_Category individually, and also for both of them. It also shows the constrained loss ratios where `State/Area` is restricted to a certain member.

$\delta_{Sales}((State, Area), (Product\_Category, Item\_Category)) = 0.33$, $\delta_{Transaction}((State, Area), (Product\_Category, Item\_Category)) = 0.43$			
State/Area	Product Category/Item Category		
$\delta_{Sales}(State, Area) = 0$, $\delta_{Transaction}(State, Area) = 0.25$	$\delta_{Sales}(Product\_Category, Item\_Category) = 0.33$, $\delta_{Transaction}(Product\_Category, Item\_Category) = 0.43$	Invoice Amt	Accessory Cost
NSW	White Goods	1700	70
	Television - TV	4700	130
	$\chi_{Sales}(Product\_Category, Item\_Category) = 0$, $\chi_{Transaction}(Product\_Category, Item\_Category) = 0.33$		
	Total	6400	200
SA	PC	2300	255
	Digital Keyboard	4400	130
	$\chi_{Sales}(Product\_Category, Item\_Category) = 0$, $\chi_{Transaction}(Product\_Category, Item\_Category) = 0$		
	Total	6700	385
Total		13100	585

Figure 6.7: Exploiting loss ratios during the data visualization.

The example shows that there is no loss related to the `Sales` data mart when using both levels. Also, if we we restrict the `State/Area` to `SA`, then there is no loss related to either data marts for the level `Product_Category/Item_Category`.

6.9 Discussion

In this chapter, the concept of combinable dimension tables was introduced. It has similar requirements to dimension compatibility, but does not require soundness, and therefore, extends the scope of integrating data marts. The main motivation is to empower users to manage the soundness and maximize the scope of the integrated data.

As discussed in Section 2.4.2, Torlone's definition of *μ-compatible* dimensions provides limited flexibility for extending the scope of the integration, but remains restrictive and difficult to use because: (i) it still makes soundness (in some limited way) to be a pre-condition for compatibility as a requirement for accurate integration; (ii) it is difficult to find and formulate lossless expressions before the integration; (iii) there may be many lossless expressions for different levels with coherent and consistent matchings; (iv) the concern for loss of data may be limited to only one of the data marts; (v) the loss may have no impact on the accuracy of aggregated measures.

Therefore, upfront application of a single pair of expressions cannot be effective. Instead, combinable levels and methods for calculating the loss ratio were introduced. These measures are used to calculate three types of measures of loss resulting from integration of non-combinable levels. They guide the user determine where lossless integration can be achieved using roll-up, slice, dice or a combination of these operations.

Whilst, expressions that correspond to these operations can be applied before the integration, they are more effectively exploited when applied interactively during the visualization of the integrated data. This is particularly true with constrained loss ratio which requires discovery of lossless expressions.

The operation drill-across was extended to return the data related to non-combinable levels to relate them to the common data. The motivation for this extension is to be able to analyze the common and related exclusive data together. The extended operation for relational tables was described using relational algebra, future work is required to define the extended drill-across for MOLAP databases.

A conceptual view of how pivot tables could be extended to support the extended drill-across was discussed. Future work is however, required to apply the extension to more suitable visualization techniques.

Beyond the use of loss ratio to maximize the scope of the integration, they could be also effective if used in matching Star schemas. In Section 2.1.2, we explained that in similarity flooding algorithm, the initial similarity values for each pair of nodes (that is tables, columns, indexes, etc.) is obtained using the similarity between their labels. A more attractive alternative is to include the use of similarity ratio (that is $1 - \Delta$) for the initial similarity value between dimension attributes. Future work is required to investigate the effectiveness of this approach.

Chapter 7

Conclusions and Future Work

> ”Be yourself and think for yourself; and while your conclusions may not be infallible, they will be nearer right than the conclusions forced upon you.”
>
> Elbert Hubbard (1856 - 1915)

In this thesis, we have taken several key steps towards automating the integration of multidimensional databases. The proposed solutions empower data analysts to reduce their reliance on experts such as database practitioners and developers, and therefore, reduce costs, and shorten the time required to deliver integrated data to data analysts. In this section, we summarize our key findings and contributions, and discuss directions for future work.

7.1 Representation and Matching of Star Schemas

We started with schema matching for Star schemas in Chapter 3. We emphasized that although Star schemas are in fact relational schemas, their distinct and predictable properties allow us describe them more precisely. The immediate benefits of a more precise description of Star schemas are that it makes them more understandable by humans and provides a more clear mapping to the structure of multidimensional data.

A more significant benefit is that we are able to get improved results with matching of Star schemas. We extended the relational properties with properties of Star schemas, and provided a description of Star schemas using UML diagram as well as OWL language. We referred to this extended representation as StarMod. We used the UML version for visual description of StarMod and the OWL version for automating the schema matching process.

We described a set of rules which we use to infer instances of the OWL version of StarMod from relational schemas. For this purpose, we have implemented the inference rules in XML transformation language (XSLT). OWL schemas are then further transformed to RDF statements used by similarity flooding algorithm for matching.

To demonstrate that using instances of StarMod improves the match results of Star schemas, we ran experiments using 18 pairs of Star and non-Star schemas. We repeated our experiments against both specifications of the same schemas (i.e. using the relational properties as well as StarMod). We used similarity flooding, a well known graph matching algorithm because it provided flexible representation of the schemas using RDF, and a basic implementation of it was available to use.

Our experiments show consistently higher accuracy when using StarMod compared with using basic relational properties. They also show that StarMod can be effective in matching arbitrary relational schemas.

We then compared the match results obtained from using relational as well as StarMod against those of COMA++ which is also a well known schema matching algorithm for relational schemas. We found that although COMA outperforms similarity flooding when using relational properties, it fails to outperform similarity flooding using StarMod. This raises the opportunity for future work to examine improving COMA++ by using StarMod properties.

Increase in the number of distinct properties resulting from the specialization of the properties adds to the computation time, future work is required to measure the impact. Discovery of StarMod properties such as degenerate dimensions and facts using physical properties is imprecise and requires additional semantic information. This would reduce the number of false positive and false negative cases. At the same time, instances of StarMod implemented using OWL language have the potential to be augmented with additional domain ontologies to help infer those properties more accurately and precisely.

7.2 Inferring Aggregation Hierarchies

A precursor to instance matching is the availability of aggregation hierarchies. The absence of a standard representation of these hierarchies as part of schema definition is the main reason why they may not be available. It is even less likely for these hierarchies to be available for heterogeneous dimensions.

In Chapter 4, we introduced algorithms to infer aggregation hierarchies from data. The problem with other sources of information such as domain ontologies, UML schemas and ER diagrams is that they may not be available either, or may return false negative partial order relationships. The fact that hierarchies are enforced either by the application or the data base management system (DBMS) makes the data to be indicative of its structure.

The first part of our proposed process to infer hierarchies is the identification of the partial order relationships between every pair of dimension attributes. The result will include redundant transitive relationships which we eliminate in the second phase of the process. Some dimension attributes have identical relationship with all other attributes. As part of the final phase, we group such attributes into a single level and assign distinct levels to the remaining dimension attributes.

If a dimension is only partially populated, we may find partial order relationships that are, in fact false and, hence, the partial order relationships in intended hierarchies are always subsets of those in inferred hierarchies. Consequently, inferred aggregation hierarchies subsume the intended hierarchies.

This leads us to the conclusion that if the matching is compatible using the inferred hierarchies, then it must be also compatible using the intended hierarchies. Therefore, using the inferred hierarchies would be sufficient to validate the matching between dimensions in terms of coherence and consistency.

Presence of false spurious partial order relationships may falsely result in the matchings to be found incoherent and/or inconsistent. What is however critical is that where we do proceed with the integration based on the inferred hierarchies, the integrated result is accurate.

A limitation of our approach is that it does not cover rare cases where a level is a composition of multiple dimension attributes. For example, there may be no partial order relationship between dimension attributes `State`, `Locality` and `Postcode`, but there may be one between `State` and {`Locality, Postcode`}. Using data alone to discover such levels is not sufficient because we may group unrelated attributes into a level. Future work could consider using both, domain information as well as data to identify such levels.

7.3 Instance Matching

The schema match results and dimension hierarchies are keys to identifying matching members in matching levels. In Chapter 5, we addressed the problem of instance matching for dimension tables. We explained that the hierarchical structure of multidimensional data provides significant clues to identifying matching members and, hence, we are interested in algorithms that exploit hierarchies.

We demonstrated through experiments with real life data that similarity flooding algorithm was very effective in matching instances of dimensions because the algorithm effectively exploits the hierarchical relationship between levels. In fact, compared with our experience with using similarity flooding for schema matching, we found that this algorithm is far more effective in matching instances of dimensions.

We argued that where original dimensions are strict, any non-strict case resulting from the integration is more likely to be due to incorrect instance matching results. This meant that we could enforce strictness and reduce the number of false positive cases at the same time. Our first algorithm in this section was designed to achieve this objective.

A side effect of using algorithms that exploit hierarchies is that the number of false positive cases increases with the increase in missing matching members in either of the instances. Our second algorithm addresses this problem by discarding those suggested matchings whose similarity scores as well as match factors of their parent matching pairs (in the connectivity graph) are outlier values. To calculate the match factor, we take into consideration, the number of matching pairs at the child and at the leaf levels. This is based on the intuition that accidental matching pairs contribute much less to the similarity scores of their descendants and ascendants.

Our experiments show that where the volume of missing data is not significant, the first algorithm is able to improve the precision with little impact on the recall despite SF performing well with very high precision and recall values. They also show that where the problem of missing data is significant, similarity flooding does not perform as well. This is where our second algorithm is able to improve the result significantly.

Our experiments also show that similarity flooding does not perform as well where there are multiple hierarchies in a dimension. Future work is required to investigate instance matching algorithms that recognize and exploit multiple hierarchies.

It was shown that our second algorithm can have a negative impact on the recall value. This occurs where the problem associated with missing data is not significant. Future work is required to improve the calculation of the match factor and identify possible relationship(s) between the threshold we use in the second algorithm, and the values of precision and recall.

7.4 Extending the Scope of Integration

In the previous chapters, we proposed methods for matching Star schemas and their instances, and for inferring aggregation hierarchies. These methods provide the necessary information to establish what is compatible between dimensions and what is not. The compatible dimensions form the basis for the integration of data marts using the drill-across operation. In this chapter, we have taken further steps towards the extending the scope of the integration beyond the compatible dimensions.

First, we relaxed the requirements for compatibility, and introduced combinable levels and dimensions. For the matching between levels to be combinable we require that the matching between them is coherent and consistent only. The definition of μ-compatible dimensions offers some relief for soundness, but is not sufficient as it considers a single pair of lossless expressions for the integration to be accurate.

Second, to exploit lossless fragments of integrated data, we proposed several methods for measuring the loss resulting from the absence of soundness. These measures are used to guide the user in performing OLAP operations such as roll-up, slice and dice to identify lossless fragments of data.

As for levels that are not combinable or have no match, they are still valuable data that need to be linked to the common data. Existing navigation (or integration) operator drill-across only returns data related to combinable levels. Third, we extend the operation drill-across to also return the data related to non-combinable levels.

The benefit of the extended drill-across is only realized if the visualization of multidimensional data supports it. It was shown in Chapter 6 (at a conceptual level), that this could be achieved by nesting pivot tables which include a parent pivot table for the common data and multiple child pivot tables for the data related to non-combinable levels.

We described the extended operation of drill-across in relational algebra for ROLAP databases. Future work is required to provide a formal framework for MOLAP databases, and to consider more sophisticated graphical techniques to show the data related to combinable and non-combinable levels in separate panes (or windows), and at the same time enable OLAP operations against one pane to be cascaded to other dependent panes.

Future direction would include a framework that brings these solutions together, and use a web service enabled discovery and retrieval of heterogeneous data marts.

In summary, our contributions help shift the problem of integrating heterogeneous data marts away from experts to data analysts. This will reduce costs and significantly saves time to deliver the integrated data to data analysts which will in turn allow the business to react to events promptly.

Appendix A

Instructions to Participants for Matching Schemas

This appendix contains instructions to the participants in the manual matching of schemas.

Dear Participant,

Thank you for participating in this experiment. Please refer to the document entitled "*Plain Language Statement*", if you wish to read again what is expected from you and what your rights are.

This document contains the following pages:

- Page 1 (This page): instructions and Introduction
- Page 2: A sample schema matching task to help you fill in the match results table

Instructions:

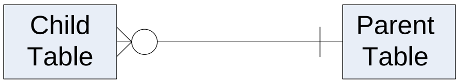
By now you have signed and returned the consent form. Following is a summary of steps you need to take:

1. There are six tasks in your package. Each task has two parts:
    a. The first part requires you to study the two schemas appearing in page 1 of every task. You are then required to fill in a table that appears on page 2 of every task. You will use this table to record the best match candidates.
    b. The second part also appears on page 2. It requires you to respond to 3 multiple choice questions about the task itself.
2. Return the answer sheet to the principal investigator using the envelope provided to you.

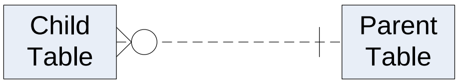
Important: Please note the following concerning the schema matching tasks:

- Your match results must cover columns as well as tables.
- A column/table in the schema on the left hand side may not match with any column/table from the schema on the right hand side, in which case no entry is required in the match results table.
- A column/table in the schema on the left hand side may match with none, one or more column or table in the schema on the right hand side.
- If a column/table in the schema on the left hand side matches with more than one column/table from the schema on the right hand side, then they must be listed separately in the match result table. Please refer to the sample schema matching task on next page.

Explanation of the notations used for the relationships in the models:



For every row in the parent table there may be zero, one or more rows in the child table. The foreign key column in the child table is part of the primary key (i.e. identifying relationship).



For every row in the parent table there may be zero, one or more rows in the child table. The foreign key column in the child table is **not** part of the primary key (i.e. non-identifying relationship).

Appendix B

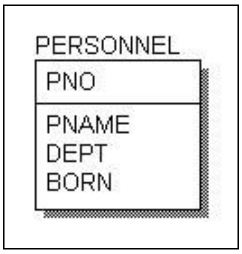
Schemas for Evaluation of StarMod

This appendix contains details of the schemas provided to the participants to match manually. Each of the 18 pairs of the schemas are identified using a reference name also used in chapter 3 for reference purpose. The visual representation and the DDL of each schema is provided.

Pair 1

Left Hand Schema Id: M7L

Model:

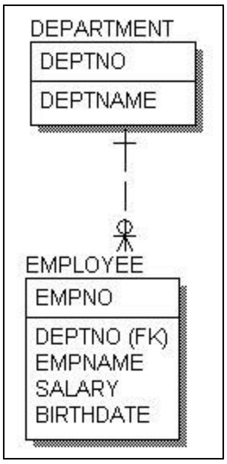


DDL:

```
1  CREATE TABLE M07L.PERSONNEL (
2   PNO INTEGER NOT NULL PRIMARY
KEY,
3   PNAME CHAR(40),
4   DEPT CHAR(40),
5   BORN DATE,
6   PRIMARY KEY (PNO)
  );
```

Right Hand Schema Id: M7R

Model:



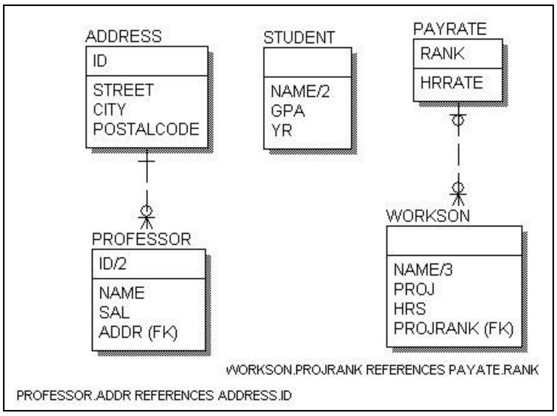
DDL:

```
a  CREATE TABLE M7R.DEPARTMENT  (
b    DEPTNO INTEGER NOT NULL ,
c    DEPTNAME VARCHAR(70),
d    PRIMARY KEY (DEPTNO) );

e  CREATE TABLE M7R.EMPLOYEE  (
f   EMPNO INTEGER NOT NULL ,
g   EMPNAME VARCHAR(50) ,
h   DEPTNO INTEGER ,
i   SALARY DECIMAL(15,2) ,
j   BIRTHDATE DATE,
    PRIMARY KEY     (EMPNO),
    FOREIGN KEY     (DEPTNO)
    REFERENCES M7R.DEPARTMENT
   );
```

Pair 2

Left Hand Schema Id: M8L

Model:

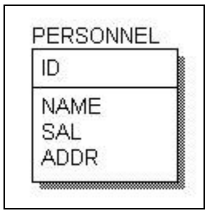


DDL:

```
CREATE TABLE M8L.ADDRESS (
  ID INTEGER NOT NULL PRIMARY KEY,
  STREET CHAR(40),
  CITY CHAR(40),
  POSTALCODE INTEGER  );

CREATE TABLE M8L.PROFESSOR (
  ID INTEGER NOT NULL PRIMARY KEY,
  NAME CHAR(40),
  SAL DOUBLE,
  ADDR INT ,
  FOREIGN KEY (ADDR) REFERENCES
          M8L.ADDRESS(ID) );

CREATE TABLE M8L.STUDENT (
  NAME CHAR(40),
  GPA DOUBLE,
  YR INTEGER  );

CREATE TABLE M8L.PAYRATE (
  RANK INTEGER NOT NULL PRIMARY KEY,
  HRRATE DOUBLE  );

CREATE TABLE M8L.WORKSON (
  NAME CHAR(40),
  PROJ CHAR(40),
  HRS INTEGER,
  PROJRANK INT,
FOREIGN KEY (PROJRANK) REFERENCES M8L.PAYRATE(RANK)  );
```

Right Hand Schema Id: M9R

Model:



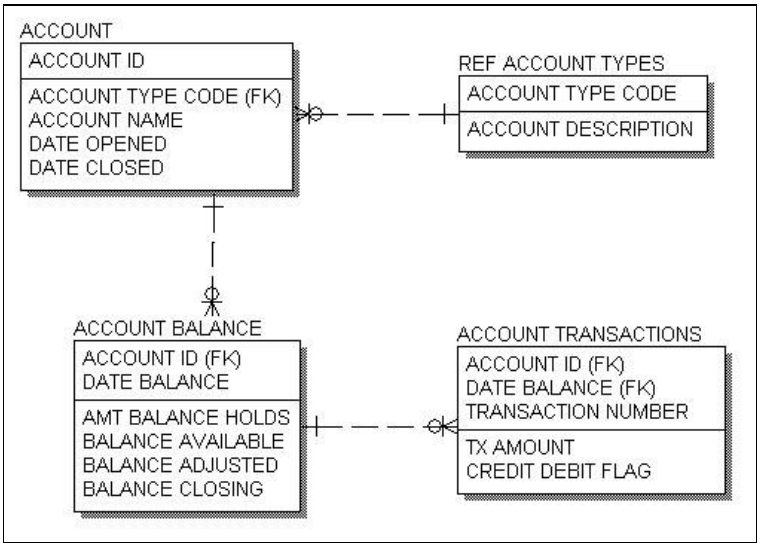
DDL:

```
CREATE TABLE M9R.PERSONNEL (
  ID INTEGER NOT NULL PRIMARY KEY,
  NAME CHAR(40),
  SAL DOUBLE,
  ADDR CHAR(40)
);
```

Pair 3

Left Hand Schema Id: R05

Model:



DDL:

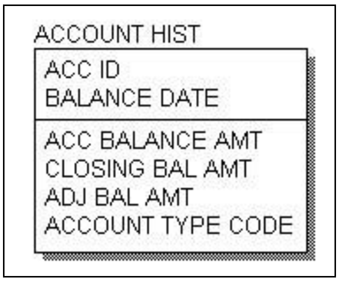
```
1   CREATE TABLE R05.REF_ACCOUNT_TYPES (
2     ACCOUNT_TYPE_CODE     CHAR(4) NOT NULL,
3     ACCOUNT_DESCRIPTION   VARCHAR(40),
      PRIMARY KEY (ACCOUNT_TYPE_CODE)    );

4   CREATE TABLE R05.ACCOUNT (
5     ACCOUNT_ID            INTEGER NOT NULL,
6     ACCOUNT_TYPE_CODE     CHAR(4) NOT NULL,
7     ACCOUNT_NAME          VARCHAR(50),
8     DATE_OPENED           DATE,
9     DATE_CLOSED           DATE,
      PRIMARY KEY (ACCOUNT_ID),
      FOREIGN KEY (ACCOUNT_TYPE_CODE)   REFERENCES
                   R05.REF_ACCOUNT_TYPES    );

10  CREATE TABLE R05.ACCOUNT_BALANCE (
11    ACCOUNT_ID            INTEGER NOT NULL,
12    DATE_BALANCE          DATE NOT NULL,
13    AMT_BALANCE_HOLDS     DOUBLE PRECISION,
14    BALANCE_AVAILABLE     DOUBLE PRECISION,
15    BALANCE_ADJUSTED      DOUBLE PRECISION,
16    BALANCE_CLOSING       DOUBLE PRECISION,
      PRIMARY KEY (ACCOUNT_ID, DATE_BALANCE),
      FOREIGN KEY (ACCOUNT_ID)  REFERENCES R05.ACCOUNT    );

17  CREATE TABLE R05.ACCOUNT_TRANSACTIONS (
18    ACCOUNT_ID            INTEGER NOT NULL,
19    DATE_BALANCE          DATE NOT NULL,
20    TRANSACTION_NUMBER    INTEGER NOT NULL,
21    CREDIT_DEBIT_FLAG     CHAR(1),
```

```
22    TX_AMOUNT            DOUBLE PRECISION,
      PRIMARY KEY (ACCOUNT_ID, DATE_BALANCE, TRANSACTION_NUMBER),
      FOREIGN KEY (ACCOUNT_ID, DATE_BALANCE)  REFERENCES
                   R05.ACCOUNT_BALANCE    );
```

Pair 3

Right Hand Schema Id: R05A

Model:



DDL:

```
a   CREATE TABLE R05A.ACCOUNT_HIST (
b     ACC_ID               INTEGER NOT NULL,
c     BALANCE_DATE         DATE NOT NULL,
d     ACC_BALANCE_AMT      DOUBLE PRECISION,
e     CLOSING_BAL_AMT      DOUBLE PRECISION,
f     ADJ_BAL_AMT     DOUBLE PRECISION,
g     ACCOUNT_TYPE_CODE    CHAR(4),
      PRIMARY KEY (ACC_ID, BALANCE_DATE)    );
```

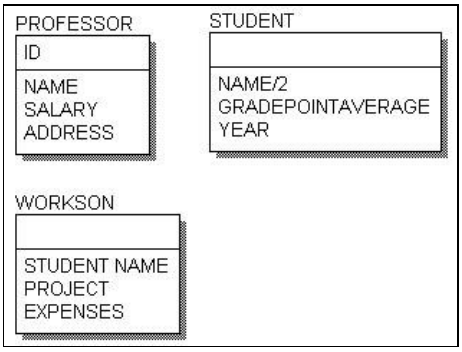
Pair 4

Left Hand Schema Id: M8L

As per left hand schema of pair 2.

Right Hand Schema Id: M8R

Model:



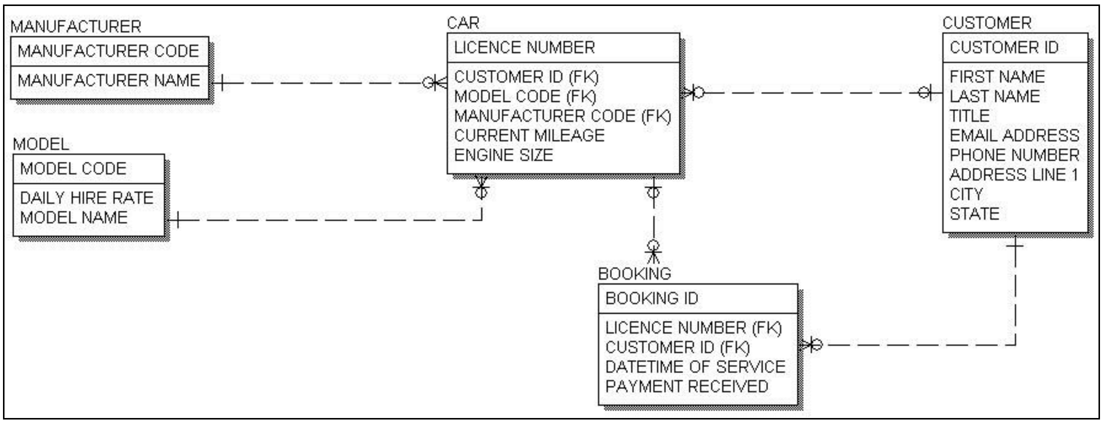
DDL:

```
a   CREATE TABLE M8R.PROFESSOR (
b     ID INTEGER NOT NULL PRIMARY KEY,
c     NAME CHAR(40),
d     SALARY DOUBLE,
e     ADDRESS CHAR(40)
    );

f   CREATE TABLE M8R.STUDENT (
g     NAME CHAR(40),
h     GRADEPOINTAVERAGE DOUBLE,
i     YEAR INTEGER
    );

j   CREATE TABLE M8R.WORKSON (
k     STUDENTNAME CHAR(40),
l     PROJECT CHAR(40),
m     EXPENSES DOUBLE
    );
```

Pair 5

Left Hand Schema Id: R01

Model:



DDL:

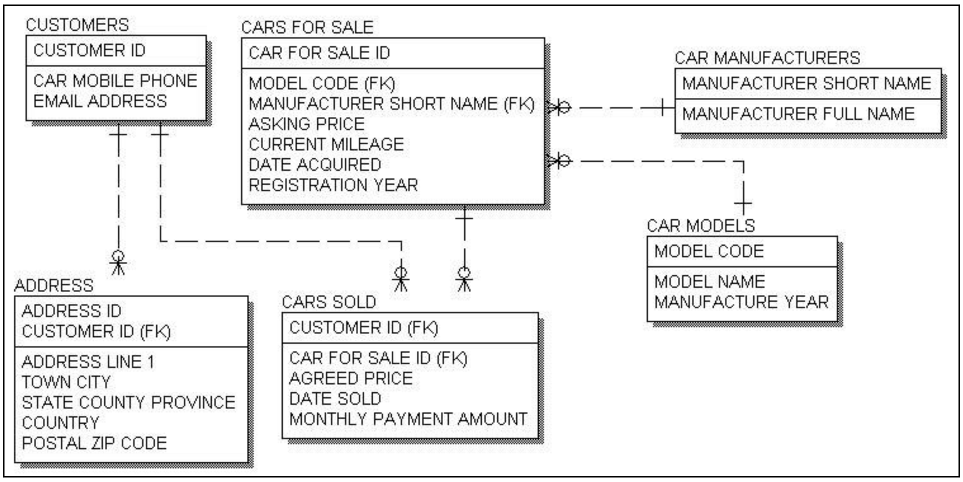
```
CREATE TABLE R01.CUSTOMER (
  CUSTOMER_ID          INTEGER NOT NULL,
  FIRST_NAME           VARCHAR(30),
  LAST_NAME            VARCHAR(40),
  TITLE                CHAR(4),
  EMAIL_ADDRESS        VARCHAR(40),
  PHONE_NUMBER         VARCHAR(20),
  ADDRESS_LINE_1       VARCHAR(60),
  CITY                 VARCHAR(30),
  STATE                VARCHAR(30),
  PRIMARY KEY (CUSTOMER_ID)    );


CREATE TABLE R01.MODEL (
  MODEL_CODE           CHAR(5) NOT NULL,
  DAILY_HIRE_RATE      DECIMAL(5,2),
  MODEL_NAME           VARCHAR(40),
  PRIMARY KEY (MODEL_CODE)    );


CREATE TABLE R01.MANUFACTURER (
  MANUFACTURER_CODE    CHAR(5) NOT NULL,
  MANUFACTURER_NAME    VARCHAR(40),
  PRIMARY KEY (MANUFACTURER_CODE)    );


CREATE TABLE R01.CAR (
  LICENCE_NUMBER       VARCHAR(15) NOT NULL,
```

```
21    CUSTOMER_ID          INTEGER,
22    MODEL_CODE           CHAR(5),
23    CURRENT_MILEAGE      INTEGER,
24    ENGINE_SIZE          DECIMAL(3,2),
25    MANUFACTURER_CODE    CHAR(5),
      PRIMARY KEY (LICENCE_NUMBER),
      FOREIGN KEY (CUSTOMER_ID)  REFERENCES R01.CUSTOMER,
      FOREIGN KEY (MODEL_CODE) REFERENCES R01.MODEL,
      FOREIGN KEY (MANUFACTURER_CODE) REFERENCES R01.MANUFACTURER );


26  CREATE TABLE R01.BOOKING (
27    BOOKING_ID           INTEGER NOT NULL,
28    CUSTOMER_ID          INTEGER,
29    LICENCE_NUMBER       VARCHAR(15),
30    DATE_OF_SERVICE      DATE,
31    PAYMENT_RECEIVED     DOUBLE,
      PRIMARY KEY (BOOKING_ID),
      FOREIGN KEY (LICENCE_NUMBER)  REFERENCES R01.CAR,
      FOREIGN KEY (CUSTOMER_ID)  REFERENCES R01.CUSTOMER   );
```

Pair 5

Right Hand Schema Id: R02

Model:



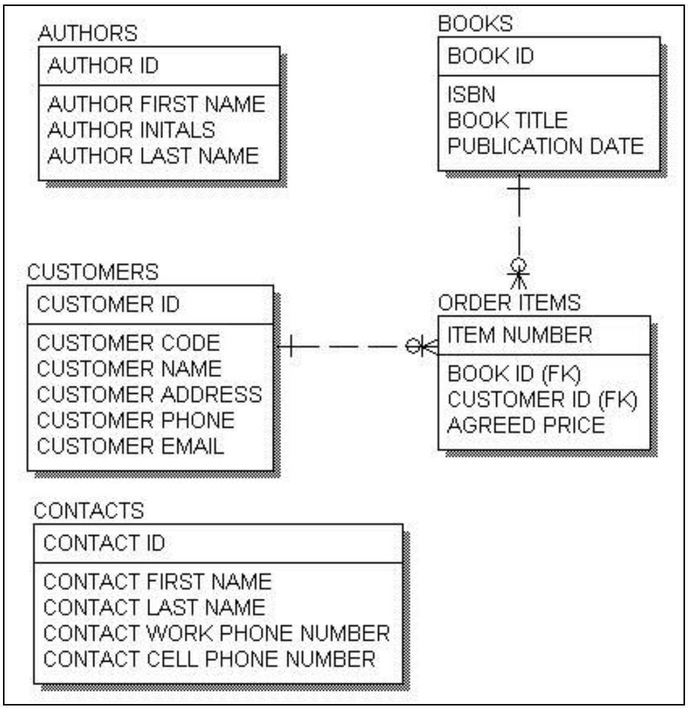
DDL:

```
a   CREATE TABLE R02.CAR_MODELS (
b     MODEL_CODE           CHAR(4) NOT NULL,
c     MODEL_NAME           VARCHAR(30),
d     MANUFACTURE_YEAR     SMALLINT,
      PRIMARY KEY (MODEL_CODE)    );


e   CREATE TABLE R02.CAR_MANUFACTURERS (
f     MANUFACTURER_SHORT_NAME VARCHAR(10) NOT NULL,
g     MANUFACTURER_FULL_NAME VARCHAR(40),
      PRIMARY KEY (MANUFACTURER_SHORT_NAME)    );


h   CREATE TABLE R02.CARS_FOR_SALE (
i     CAR_FOR_SALE_ID      INTEGER NOT NULL,
j     ASKING_PRICE         DOUBLE PRECISION,
k     MODEL_CODE           CHAR(4),
l     CURRENT_MILEAGE      INTEGER,
m     MANUFACTURER_SHORT_NAME VARCHAR(10),
n     DATE_ACQUIRED        DATE,
o     REGISTRATION_YEAR    SMALLINT,
      PRIMARY KEY (CAR_FOR_SALE_ID),
      FOREIGN KEY (MODEL_CODE)  REFERENCES R02.CAR_MODELS,
      FOREIGN KEY (MANUFACTURER_SHORT_NAME)  REFERENCES
                   R02.CAR_MANUFACTURERS    );


p   CREATE TABLE R02.CUSTOMERS (
q     CUSTOMER_ID          INTEGER NOT NULL,
r     CAR_MOBILE_PHONE     VARCHAR(20),
s     EMAIL_ADDRESS        VARCHAR(40),
      PRIMARY KEY (CUSTOMER_ID)    );


t   CREATE TABLE R02.CARS_SOLD (
u     CAR_FOR_SALE_ID      INTEGER,
v     CUSTOMER_ID          INTEGER NOT NULL,
w     AGREED_PRICE         DOUBLE PRECISION,
x     DATE_SOLD            DATE,
y     MONTHLY_PAYMENT_AMOUNT DOUBLE PRECISION,
      PRIMARY KEY (CUSTOMER_ID),
      FOREIGN KEY (CAR_FOR_SALE_ID)  REFERENCES R02.CARS_FOR_SALE,
      FOREIGN KEY (CUSTOMER_ID)  REFERENCES R02.CUSTOMERS    );


z   CREATE TABLE R02.ADDRESS (
aa    ADDRESS_ID           INTEGER NOT NULL,
bb    CUSTOMER_ID          INTEGER NOT NULL,
cc    ADDRESS_LINE_1       VARCHAR(60),
dd    TOWN_CITY            VARCHAR(30),
ee    STATE_COUNTY_PROVINCE VARCHAR(40),
ff    COUNTRY              VARCHAR(40),
gg    POSTAL_ZIP_CODE      VARCHAR(15),
      PRIMARY KEY (CUSTOMER_ID, ADDRESS_ID),
      FOREIGN KEY (CUSTOMER_ID)  REFERENCES R02.CUSTOMERS    );
```

Pair 6

Left Hand Schema Id: R03

Physical Model:



DDL:

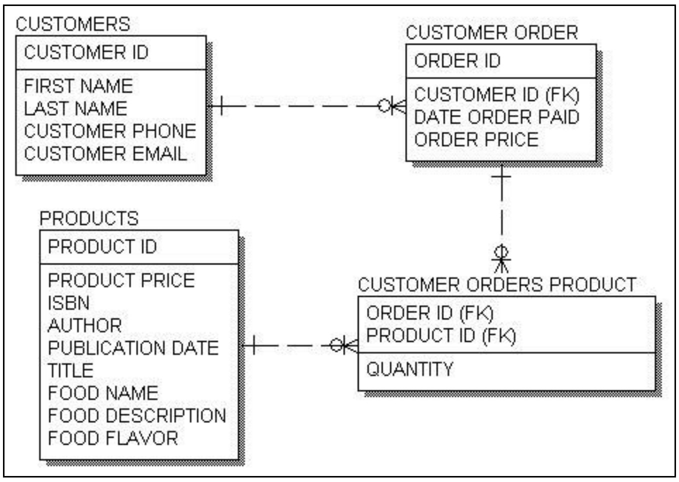
```
CREATE TABLE R03.BOOKS (
  BOOK_ID          INTEGER NOT NULL,
  ISBN             VARCHAR(30),
  TITLE            VARCHAR(40),
  BOOK_PUBLICATION_DATE DATE,
  PRIMARY KEY (BOOK_ID)    );

CREATE TABLE R03.CUSTOMERS (
  CUSTOMER_ID          INTEGER NOT NULL,
  CUSTOMER_CODE        CHAR(3),
  CUSTOMER_NAME        VARCHAR(30),
  CUSTOMER_ADDRESS     VARCHAR(80),
  CUSTOMER_PHONE       VARCHAR(20),
  CUSTOMER_EMAIL       VARCHAR(20),
  PRIMARY KEY (CUSTOMER_ID)    );

CREATE TABLE R03.ORDER_ITEMS (
  ITEM_NUMBER          SMALLINT NOT NULL,
  BOOK_ID              INTEGER NOT NULL,
  CUSTOMER_ID          INTEGER NOT NULL,
  AGREED_PRICE         DOUBLE PRECISION,
```

```
      PRIMARY KEY (ITEM_NUMBER),
      FOREIGN KEY (BOOK_ID)  REFERENCES
              R03.BOOKS,
      FOREIGN KEY (CUSTOMER_ID)  REFERENCES
              R03.CUSTOMERS    );

18  CREATE TABLE R03.CONTACTS (
19    CONTACT_ID             INTEGER NOT NULL,
20    CONTACT_FIRST_NAME     VARCHAR(30),
21    CONTACT_LAST_NAME      VARCHAR(30),
22    CONTACT_WORK_PHONE_NUMBER VARCHAR(20),
23    CONTACT_CELL_PHONE_NUMBER VARCHAR(20),
      PRIMARY KEY (CONTACT_ID)    );

24  CREATE TABLE R03.AUTHORS (
25    AUTHOR_ID         INTEGER NOT NULL,
26    AUTHOR_FIRST_NAME      VARCHAR(40),
27    AUTHOR_INITALS         CHAR(2),
28    AUTHOR_LAST_NAME       VARCHAR(40),
      PRIMARY KEY (AUTHOR_ID)    );
```

Pair 6

Right Hand Schema Id: R06

Physical Model:



DDL:

```
a   CREATE TABLE R06.PRODUCTS (
b     PRODUCT_ID             INTEGER NOT NULL,
c     PRODUCT_PRICE          DOUBLE PRECISION,
d     BOOK_ISBN              VARCHAR(30),
```

```
e     BOOK_AUTHOR           VARCHAR(40),
f     PUBLICATION_DATE      DATE,
g     BOOK_TITLE            VARCHAR(40),
h     FOOD_NAME             VARCHAR(30),
i     FOOD_DESCRIPTION      VARCHAR(60),
j     FOOD_FLAVOR           VARCHAR(20),
      PRIMARY KEY (PRODUCT_ID)     );

k   CREATE TABLE R06.CUSTOMERS (
l     CUSTOMER_ID           INTEGER NOT NULL,
m     FIRST_NAME            VARCHAR(30),
n     LAST_NAME             VARCHAR(30),
o     CUSTOMER_PHONE        VARCHAR(20),
p     CUSTOMER_EMAIL        VARCHAR(30),
      PRIMARY KEY (CUSTOMER_ID)     );

q   CREATE TABLE R06.CUSTOMER_ORDER (
r     CUSTOMER_ID           INTEGER NOT NULL,
s     ORDER_ID              CHAR(10) NOT NULL,
t     DATE_ORDER_PAID       DATE,
u     ORDER_PRICE           DOUBLE PRECISION,
      PRIMARY KEY (ORDER_ID),
      FOREIGN KEY (CUSTOMER_ID)  REFERENCES
              R06.CUSTOMERS    );

v   CREATE TABLE R06.CUSTOMER_ORDERS_PRODUCT (
w     ORDER_ID              CHAR(10) NOT NULL,
x     PRODUCT_ID            INTEGER NOT NULL,
y     QUANTITY              SMALLINT,
      PRIMARY KEY (ORDER_ID, PRODUCT_ID),
      FOREIGN KEY (PRODUCT_ID)   REFERENCES
              R06.PRODUCTS,
      FOREIGN KEY (ORDER_ID) REFERENCES
              R06.CUSTOMER_ORDER    );
```

Pair 7

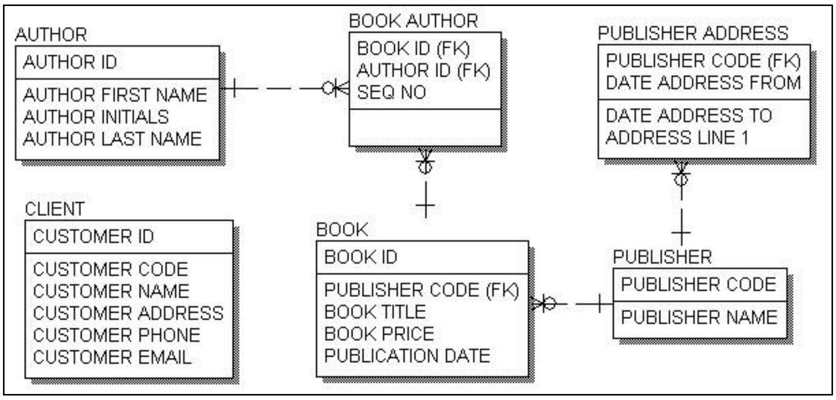
Left Hand Schema Id: R06

As pair right hand schema of pair 6.

Pair 7

Right Hand Schema Id: R07

Physical Model:



DDL:

```
a   CREATE TABLE R07.CLIENT (
b     CUSTOMER_ID           INTEGER NOT NULL,
c     CUSTOMER_CODE         CHAR(4),
d     CUSTOMER_NAME         CHARACTER(30),
e     CUSTOMER_ADDRESS      VARCHAR(120),
f     CUSTOMER_PHONE        CHAR(10),
g     CUSTOMER_EMAIL        VARCHAR(15),
      PRIMARY KEY (CUSTOMER_ID)    );

h   CREATE TABLE R07.PUBLISHER (
i     PUBLISHER_CODE        CHAR(4) NOT NULL,
j     PUBLISHER_NAME        VARCHAR(40),
      PRIMARY KEY (PUBLISHER_CODE));

k   CREATE TABLE R07.PUBLISHER_ADDRESS (
l     PUBLISHER_CODE        CHAR(4) NOT NULL,
m     DATE_ADDRESS_FROM     DATE NOT NULL,
n     DATE_ADDRESS_TO       DATE,
o     ADDRESS_LINE_1        VARCHAR(80)
```

```
      PRIMARY KEY (PUBLISHER_CODE, DATE_ADDRESS_FROM),
      FOREIGN KEY (PUBLISHER_CODE) REFERENCES R07.PUBLISHER    );

p   CREATE TABLE R07.BOOK (
q     BOOK_ID              INTEGER NOT NULL,
r     PUBLISHER_CODE       CHAR(4),
s     BOOK_TITLE           VARCHAR(40),
t     BOOK_PRICE           DECIMAL(6,2),
u     PUBLICATION_DATE     DATE,
      PRIMARY KEY (BOOK_ID),
      FOREIGN KEY (PUBLISHER_CODE)  REFERENCES R07.PUBLISHER    );

v   CREATE TABLE R07.AUTHOR (
w     AUTHOR_ID            INTEGER NOT NULL,
x     AUTHOR_FIRST_NAME    VARCHAR(30),
y     AUTHOR_INITIALS      CHAR(3),
z     AUTHOR_LAST_NAME     CHARACTER(30),
      PRIMARY KEY (AUTHOR_ID)    );

aa   CREATE TABLE R07.BOOK_AUTHOR (
bb    BOOK_ID              INTEGER NOT NULL,
cc    AUTHOR_ID            INTEGER NOT NULL,
dd    SEQ_NO    SMALLINT,
      PRIMARY KEY (BOOK_ID, AUTHOR_ID, SEQ_NO),
      FOREIGN KEY (BOOK_ID)   REFERENCES R07.BOOK,
      FOREIGN KEY (AUTHOR_ID)    REFERENCES R07.AUTHOR    );
```

Pair 8

Left Hand Schema Id: R07

AS per right hand schema of pair 7.

Pair 8

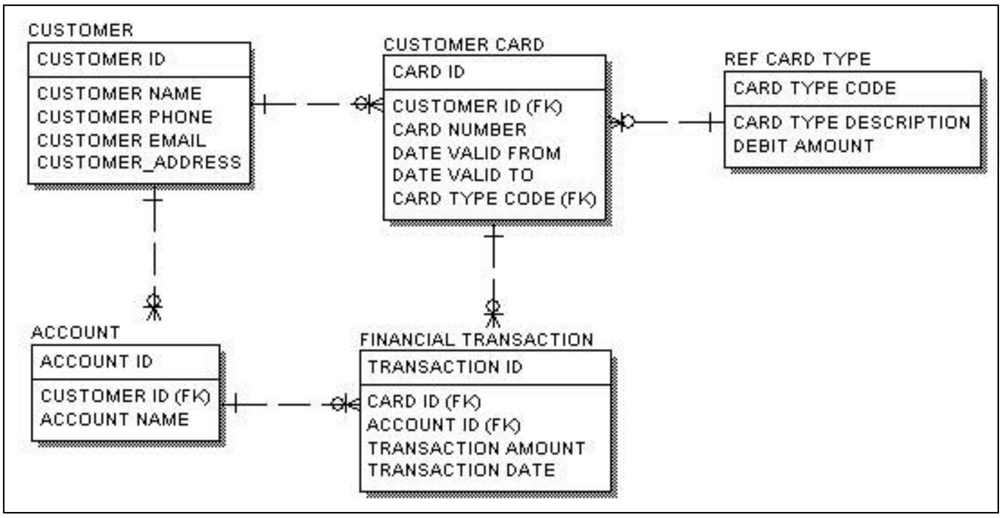
Right Hand Schema Id: R03

As per left hand schema of pair 6.

Pair 9

Left Hand Schema Id: R08A

Physical Model:



DDL:

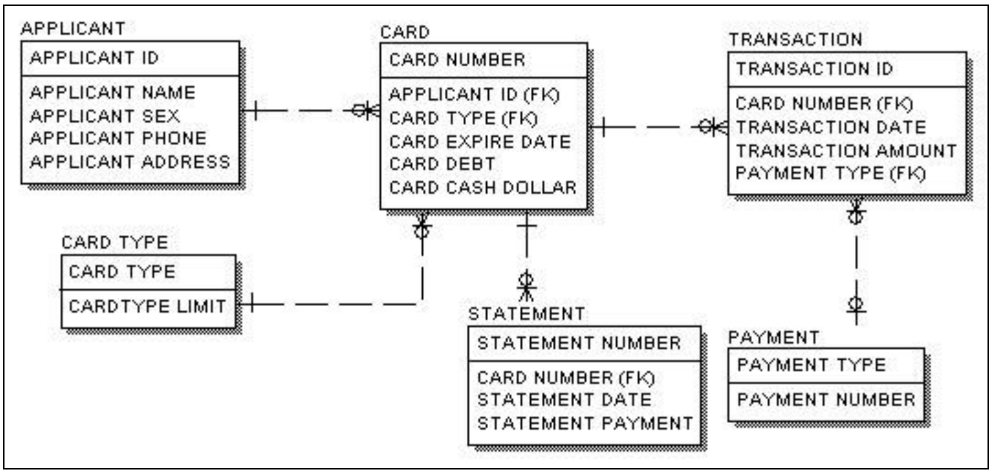
```
1    CREATE TABLE R08A.REF_CARD_TYPE (
2       CARD_TYPE_CODE        CHAR(1) NOT NULL,
3       CARD_TYPE_DESCRIPTION VARCHAR(40),
4       DEBIT_AMOUNT          DOUBLE PRECISION,
        PRIMARY KEY (CARD_TYPE_CODE)    );

5    CREATE TABLE R08A.CUSTOMER (
6      CUSTOMER_ID           INTEGER NOT NULL,
7      CUSTOMER_NAME         CHARACTER(40),
8      CUSTOMER_PHONE        CHAR(10),
9      CUSTOMER_EMAIL        VARCHAR(30),
10     CUSTOMER_ADDRESS      VARCHAR(80),
       PRIMARY KEY (CUSTOMER_ID)    );

11   CREATE TABLE R08A.CUSTOMER_CARD (
12     CARD_ID               INTEGER NOT NULL,
13     CUSTOMER_ID           INTEGER NOT NULL,
14     CARD_NUMBER           CHARACTER(28),
15     DATE_VALID_FROM       DATE,
16     DATE_VALID_TO         DATE,
17     CARD_TYPE_CODE        CHAR(1) NOT NULL,
       PRIMARY KEY (CARD_ID),
       FOREIGN KEY (CARD_TYPE_CODE)  REFERENCES R08A.REF_CARD_TYPE,
       FOREIGN KEY (CUSTOMER_ID) REFERENCES R08A.CUSTOMER    );
```

```
18  CREATE TABLE R08A.ACCOUNT (
19     ACCOUNT_ID           INTEGER NOT NULL,
20     ACCOUNT_NAME         VARCHAR(40),
21     CUSTOMER_ID          INTEGER NOT NULL,
       PRIMARY KEY (ACCOUNT_ID),
       FOREIGN KEY (CUSTOMER_ID) REFERENCES R08A.CUSTOMER    );

22  CREATE TABLE R08A.FINANCIAL_TRANSACTION (
23     TRANSACTION_ID       VARCHAR(20) NOT NULL,
24     CARD_ID              INTEGER NOT NULL,
25     TRANSACTION_AMOUNT   DOUBLE PRECISION,
26     ACCOUNT_ID           INTEGER NOT NULL,
27     TRANSACTION_DATE     DATE,
       PRIMARY KEY (TRANSACTION_ID),
       FOREIGN KEY (CARD_ID)  REFERENCES R08A.CUSTOMER_CARD,
       FOREIGN KEY (ACCOUNT_ID)  REFERENCES R08A.ACCOUNT  );
```

Pair 9

Right Hand Schema Id: R08B

Physical Model:

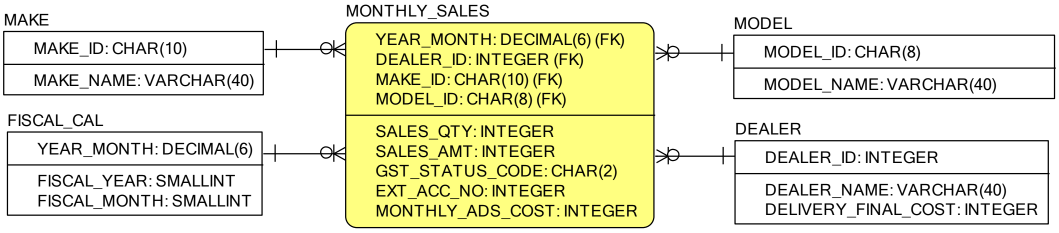


DDL:

```
a   CREATE TABLE R08B.APPLICANT (
b     APPLICANT_ID         INTEGER NOT NULL,
c     APPLICANT_NAME       VARCHAR(40),
d     APPLICANT_SEX        CHAR(1),
e     APPLICANT_PHONE      VARCHAR(15),
f     APPLICANT_ADDRESS    VARCHAR(80),
      PRIMARY KEY (APPLICANT_ID)    );

g   CREATE TABLE R08B.CARD_TYPE (
h     CARD_TYPE            CHAR(1) NOT NULL,
i     CARDTYPE_LIMIT       DOUBLE PRECISION,
      PRIMARY KEY (CARD_TYPE)    );

j   CREATE TABLE R08B.CARD (
k     CARD_NUMBER          VARCHAR(28) NOT NULL,
l     CARD_EXPIRE_DATE     DATE,
m     APPLICANT_ID         INTEGER NOT NULL,
n     CARD_DEBT            DOUBLE PRECISION,
o     CARD_TYPE            CHAR(1) NOT NULL,
p     CARD_CASH_DOLLAR     DOUBLE PRECISION,
      PRIMARY KEY (CARD_NUMBER),
      FOREIGN KEY (APPLICANT_ID) REFERENCES R08B.APPLICANT,
      FOREIGN KEY (CARD_TYPE)  REFERENCES R08B.CARD_TYPE    );

q   CREATE TABLE R08B.PAYMENT (
r     PAYMENT_TYPE         CHAR(1) NOT NULL,
s     PAYMENT_DESCRIPTION  VARCHAR(40),
      PRIMARY KEY (PAYMENT_TYPE)    );

t   CREATE TABLE R08B.TRANSACTION (
u     TRANSACTION_ID       VARCHAR(20) NOT NULL,
v     CARD_NUMBER          VARCHAR(28) NOT NULL,
w     TRANSACTION_DATE     DATE,
x     TRANSACTION_AMOUNT   DOUBLE PRECISION,
      PAYMENT_TYPE         CHAR(1) NOT NULL,
      PRIMARY KEY (TRANSACTION_ID),
      FOREIGN KEY (CARD_NUMBER)  REFERENCES R08B.CARD,
      FOREIGN KEY (PAYMENT_TYPE)  REFERENCES R08B.PAYMENT    );

y   CREATE TABLE R08B.STATEMENT (
z     STATEMENT_NUMBER     INTEGER NOT NULL,
aa    CARD_NUMBER          VARCHAR(28) NOT NULL,
bb    STATEMENT_DATE       DATE,
cc    TRANSACTION_ID       VARCHAR(20) NOT NULL,
dd    STATEMENT_PAYMENT    DOUBLE PRECISION,
      PRIMARY KEY (STATEMENT_NUMBER),
      FOREIGN KEY (CARD_NUMBER)  REFERENCES R08B.CARD    );
```

Pair 10

Left Hand Schema Id: PP1

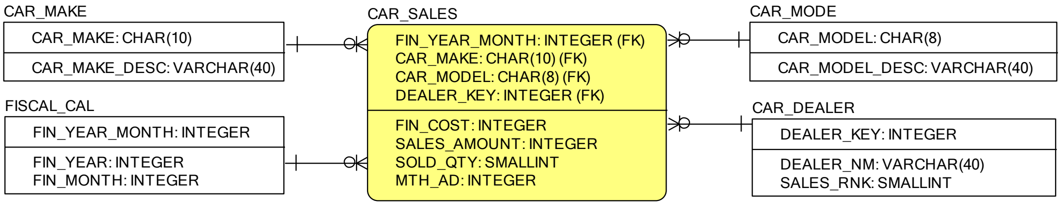


DDL

```
1  CREATE TABLE PP1.FISCAL_CAL  (
2    YEAR_MONTH DECIMAL(6,0) NOT NULL ,
3    FISCAL_YEAR SMALLINT ,
4    FISCAL_MONTH SMALLINT ),
     PRIMARY KEY (YEAR_MONTH);

5  CREATE TABLE PP1.MAKE  (
6      MAKE_ID CHAR(10) NOT NULL ,
7      MAKE_NAME VARCHAR(40),
       PRIMARY KEY (MAKE_ID)   );

8  CREATE TABLE PP1.MODEL  (
9      MODEL_ID CHAR(8) NOT NULL ,
10     MODEL_NAME VARCHAR(40),
       PRIMARY KEY (MODEL_ID)    );

11 CREATE TABLE PP1.DEALER  (
12     DEALER_ID SMALLINT NOT NULL ,
13     DEALER_NAME VARCHAR(40) ,
14     DELIVERY_FINAL_COST INTEGER,
       PRIMARY KEY (DEALER_ID)    );

-- The key values for the DEALER_ID are generated by the Database Manager
-- sequentially starting from 1 and incremented by 1 for each row.
   ALTER COLUMN DEALER_ID
    GENERATED ALWAYS AS IDENTITY START WITH 1, INCREMENT BY 1

15 CREATE TABLE PP1.MONTHLY_SALES  (
16     YEAR_MONTH DECIMAL(6,0) NOT NULL ,
17     MAKE_ID CHAR(10) NOT NULL ,
18     MODEL_ID CHAR(8) NOT NULL ,
19     DEALER_ID SMALLINT NOT NULL ,
20     SALES_QTY SMALLINT ,
21     SALES_AMT INTEGER ,
22     MONTHLY_ADS_CST INTEGER ,
23     GST_STATUS_CODE CHAR(2) ,
24     EXT_ACC_NO INTEGER,
```

```
       PRIMARY KEY (YEAR_MONTH, DEALER_ID, MODEL_ID, MAKE_ID),
       FOREIGN KEY (YEAR_MONTH) REFERENCES PP1.FISCAL_CAL (YEAR_MONTH),
       FOREIGN KEY (MAKE_ID) REFERENCES PP1.MAKE (MAKE_ID),
       FOREIGN KEY (MODEL_ID) REFERENCES PP1.MODEL (MODEL_ID),
       FOREIGN KEY (DEALER_ID) REFERENCES PP1.DEALER (DEALER_ID)    );
```

Pair 10
Right Hand Schema Id: PP2



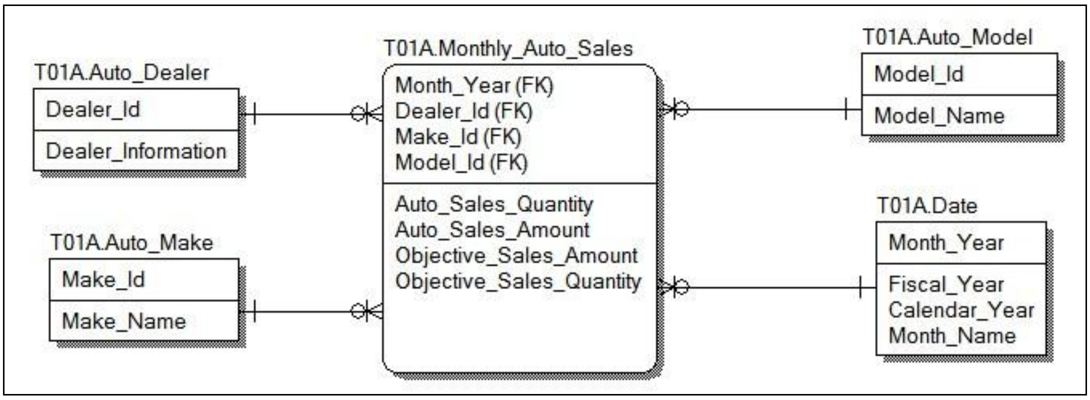
DDL

```
a  CREATE TABLE PP2.FINANCIAL_CAL (
b   FIN_YEAR_MONTH INTEGER NOT NULL ,
c   FIN_YEAR INTEGER ,
d   FIN_MONTH SMALLINT,
    PRIMARY KEY (FIN_YEAR_MONTH)
   );

e  CREATE TABLE PP2.CAR_DEALER (
f   DEALER_KEY SMALLINT NOT NULL ,
g   DEALER_NM VARCHAR(40) ,
h   SALES_RNK SMALLINT,
    PRIMARY KEY (DEALER_KEY)
  );

-- The key values for the DEALER_KEY are generated by the Database
-- Manager and sequentially starting from 1 and incremented by 1 for
-- each row.
   ALTER COLUMN DEALER_KEY
    GENERATED ALWAYS AS IDENTITY START WITH 1, INCREMENT BY 1

i  CREATE TABLE PP2.CAR_MAKE (
j   CAR_MAKE CHAR(10) NOT NULL ,
k   CAR_MAKE_DESC VARCHAR(40),
    PRIMARY KEY (CAR_MAKE)
   );

l  CREATE TABLE PP2.CAR_MODEL (
m   CAR_MODEL CHAR(8) NOT NULL ,
n   CAR_MODEL_DESC VARCHAR(40),
    PRIMARY KEY(CAR_MODEL)
   );

o  CREATE TABLE PP2.CAR_SALES (
```

```
p   FIN_YEAR_MONTH INTEGER NOT NULL,
q   DEALER_KEY SMALLINT NOT NULL,
r   CAR_MODEL CHAR(8) NOT NULL,
s   CAR_MAKE CHAR(10) NOT NULL,
t   FIN_COST INTEGER NOT NULL,
u   SALES_AMOUNT INTEGER NOT NULL,
v   MONTH_AD INTEGER NOT NULL,
w   SOLD_QTY SMALLINT NOT NULL,
    FOREIGN KEY (DEALER_KEY) REFERENCES PP2.CAR_DEALER (DEALER_KEY),
    FOREIGN KEY (CAR_MODEL)  REFERENCES PP2.CAR_MODEL(CAR_MODEL),
    FOREIGN KEY (CAR_MAKE)  REFERENCES PP2.CAR_MAKE (CAR_MAKE),
    FOREIGN KEY (FIN_YEAR_MONTH)  REFERENCES
                PP2.FINANCIAL_CAL(FIN_YEAR_MONTH)
  );
```

Pair 11

Left Hand Schema Id: T01A:



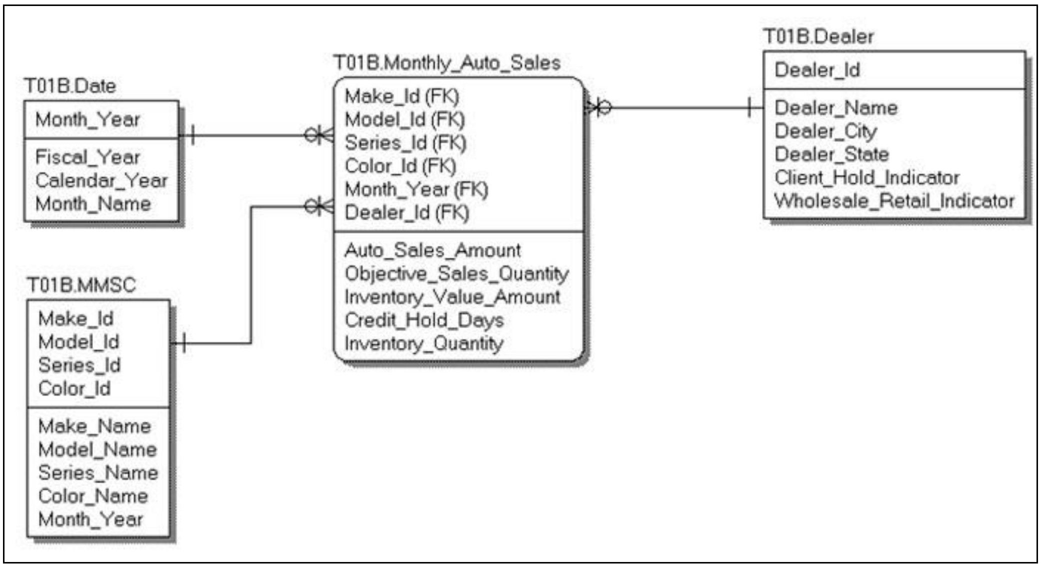
<u>DDL</u>

```
CREATE TABLE T01A.AUTO_DEALER (
       DEALER_ID            INTEGER NOT NULL,
       DEALER_INFORMATION   VARCHAR(100) NOT NULL,
       PRIMARY KEY (DEALER_ID)    );

CREATE TABLE T01A.AUTO_MAKE (
       MAKE_ID              CHAR(8) NOT NULL,
       MAKE_NAME            VARCHAR(20),
       PRIMARY KEY (MAKE_ID   );

CREATE TABLE T01A.AUTO_MODEL (
       MODEL_ID             CHAR(8) NOT NULL,
       MODEL_NAME           VARCHAR(30),
       PRIMARY KEY (MODEL_ID)    );

CREATE TABLE T01A.DATE (
       MONTH_YEAR           DECIMAL(6) NOT NULL,
       FISCAL_YEAR          SMALLINT NOT NULL,
       CALENDAR_YEAR        SMALLINT NOT NULL,
       MONTH_NAME           CHARACTER(10) NOT NULL,
       PRIMARY KEY (MONTH_YEAR)    );


CREATE TABLE T01A.MONTHLY_AUTO_SALES (
       MONTH_YEAR           DECIMAL(6) NOT NULL,
       DEALER_ID            INTEGER NOT NULL,
       MAKE_ID              CHAR(8) NOT NULL,
       MODEL_ID             CHAR(8) NOT NULL,
       AUTO_SALES_QUANTITY   INTEGER NOT NULL,
       AUTO_SALES_AMOUNT    INTEGER NOT NULL,
       OBJECTIVE_SALES_AMOUNT   INTEGER,
       OBJECTIVE_SALES_QUANTITY   INTEGER,
       PRIMARY KEY (MAKE_ID, MONTH_YEAR, DEALER_ID, MODEL_ID),
```

```
          FOREIGN KEY (MODEL_ID)  REFERENCES T01A.AUTO_MODEL,
          FOREIGN KEY (DEALER_ID)   REFERENCES T01A.AUTO_DEALER,
          FOREIGN KEY (MONTH_YEAR) REFERENCES T01A.DATE,
          FOREIGN KEY (MAKE_ID)  REFERENCES T01A.AUTO_MAKE    );
```

Pair 11

Right Hand Schema Id: T01B:



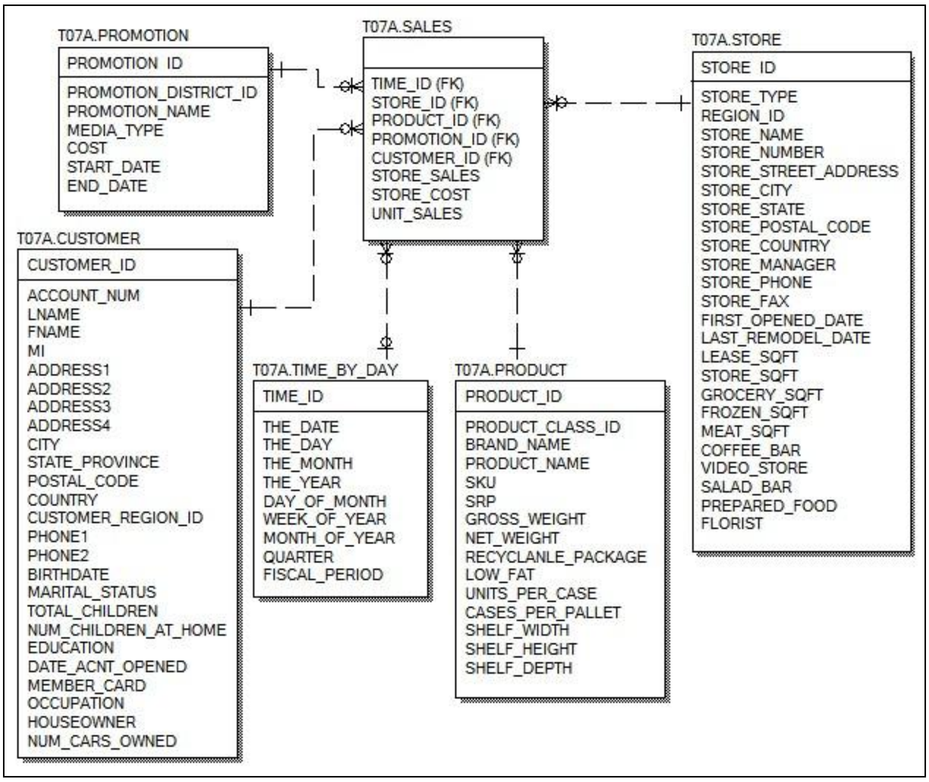
<u>DDL</u>

```
a  CREATE TABLE T01B.DATE (
b         MONTH_YEAR          INTEGER NOT NULL,
c         FISCAL_YEAR         SMALLINT,
d         CALENDAR_YEAR       SMALLINT,
e         MONTH_NAME          CHAR(10),
          PRIMARY KEY (MONTH_YEAR)    );

f   CREATE TABLE T01B.DEALER (
g         DEALER_ID           SMALLINT NOT NULL,
h         DEALER_NAME         VARCHAR(40),
i         DEALER_CITY         VARCHAR(40),
j         DEALER_STATE        VARCHAR(40),
k         CLIENT_HOLD_INDICATOR   CHAR(1),
l         WHOLESALE_RETAIL_INDICATOR   CHAR(1),
          PRIMARY KEY (DEALER_ID)    );

m   CREATE TABLE T01B.MMSC (
n         MAKE_ID             CHAR(6) NOT NULL,
o         MODEL_ID            CHAR(6) NOT NULL,
```

```
p          SERIES_ID              CHAR(10) NOT NULL,
q          COLOR_ID               CHAR(10) NOT NULL,
r          MAKE_NAME              VARCHAR(40),
s          MODEL_NAME             VARCHAR(40),
t          SERIES_NAME            VARCHAR(40),
u          COLOR_NAME             VARCHAR(40),
v          MONTH_YEAR             DECIMAL(6),
           PRIMARY KEY (MAKE_ID, MODEL_ID, SERIES_ID, COLOR_ID)    );

w   CREATE TABLE T01B.MONTHLY_AUTO_SALES (
x          MAKE_ID                CHAR(6) NOT NULL,
y          MODEL_ID               CHAR(6) NOT NULL,
z          SERIES_ID              CHAR(10) NOT NULL,
aa         COLOR_ID               CHAR(10) NOT NULL,
bb         MONTH_YEAR             INTEGER NOT NULL,
cc         DEALER_ID              SMALLINT NOT NULL,
dd         AUTO_SALES_AMOUNT      DECIMAL(11,2),
ee         OBJECTIVE_SALES_QUANTITY   SMALLINT,
ff         INVENTORY_VALUE_AMOUNT   DECIMAL(11,2),
gg         OBJECTIVE_SALES_AMOUNT   DECIMAL(11,2),
hh         CREDIT_HOLD_DAYS       INTEGER,
ii         INVENTORY_QUANTITY     SMALLINT,
           PRIMARY KEY (MONTH_YEAR, DEALER_ID, MAKE_ID, MODEL_ID,
                        SERIES_ID, COLOR_ID),
           FOREIGN KEY (MAKE_ID, MODEL_ID, SERIES_ID, COLOR_ID)
                        REFERENCES MMSC,
           FOREIGN KEY (DEALER_ID) REFERENCES T01B.DEALER,
           FOREIGN KEY (MONTH_YEAR) REFERENCES T01B.DATE    );
```

Pair 12

Left Hand Schema Id: T07A



DDL

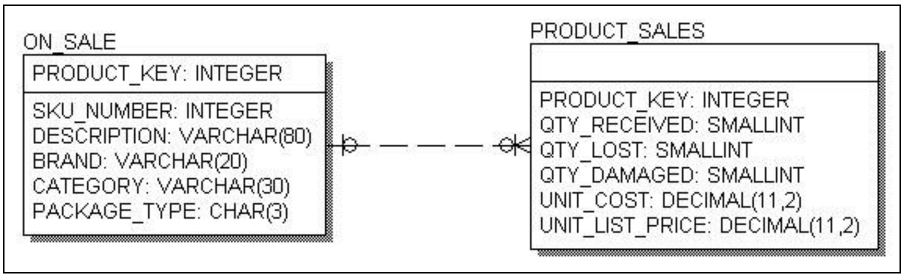
```
CREATE TABLE  T07A.CUSTOMER    (
 CUSTOMER_ID  INTEGER  NOT NULL,
 ACCOUNT_NUM  DOUBLE  NOT NULL,
 LNAME  VARCHAR(100) ,
 FNAME  VARCHAR(50) ,
 MI  VARCHAR(20) ,
 ADDRESS1  VARCHAR(100)  NOT NULL,
 ADDRESS2  VARCHAR(100) ,
 ADDRESS3  VARCHAR(100) ,
 ADDRESS4  VARCHAR(100)  NOT NULL,
 CITY  VARCHAR(50)  NOT NULL,
 STATE_PROVINCE  VARCHAR(50)  NOT NULL,
 POSTAL_CODE  CHAR(6)  NOT NULL,
 COUNTRY  VARCHAR(50)  NOT NULL,
 CUSTOMER_REGION_ID  INTEGER  NOT NULL,
 PHONE1  VARCHAR(16)  NOT NULL,
 PHONE2  VARCHAR(16)  NOT NULL,
 BIRTHDATE  DATE  NOT NULL,
 MARITAL_STATUS  CHAR(1)  NOT NULL,
 TOTAL_CHILDREN  SMALLINT  NOT NULL,
 NUM_CHILDREN_AT_HOME  SMALLINT  NOT NULL,
```

```
22   EDUCATION  VARCHAR(30)  NOT NULL,
23   DATE_ACNT_OPENED  DATE  NOT NULL,
24   MEMBER_CARD  VARCHAR(50)  NOT NULL,
25   OCCUPATION  VARCHAR(50)  NOT NULL,
26   HOUSEOWNER  CHAR(1)  NOT NULL,
27   NUM_CARS_OWNED  SMALLINT  NOT NULL,
     PRIMARY KEY (CUSTOMER_ID)     )  ;

28  CREATE TABLE  T07A.PRODUCT      (
29   PRODUCT_ID  INTEGER  NOT NULL,
30   PRODUCT_CLASS_ID  INTEGER  NOT NULL,
31   BRAND_NAME  VARCHAR(40)  NOT NULL,
32   PRODUCT_NAME  VARCHAR(40)  NOT NULL,
33   SKU  DOUBLE  NOT NULL,
34   SRP  DOUBLE  NOT NULL,
35   GROSS_WEIGHT  DOUBLE  NOT NULL,
36   NET_WEIGHT  DOUBLE ,
37   RECYCLANLE_PACKAGE  CHAR(1) ,
38   LOW_FAT  CHAR(1) ,
39   UNITS_PER_CASE  SMALLINT  NOT NULL,
40   CASES_PER_PALLET  SMALLINT  NOT NULL,
41   SHELF_WIDTH  FLOAT ,
42   SHELF_HEIGHT  SMALLINT  NOT NULL,
43   SHELF_DEPTH  SMALLINT  NOT NULL,
     PRIMARY KEY (PRODUCT_ID)    ) ;


44  CREATE TABLE  T07A.PROMOTION      (
45   PROMOTION_ID  INTEGER  NOT NULL,
46   PROMOTION_DISTRICT_ID  INTEGER  NOT NULL,
47   PROMOTION_NAME  VARCHAR(40)  NOT NULL,
48   MEDIA_TYPE  VARCHAR(2)  NOT NULL,
49   COST  DOUBLE  NOT NULL,
50   START_DATE  DATE  NOT NULL,
51   END_DATE  DATE  NOT NULL ,
     PRIMARY KEY (PROMOTION_ID)  ) ;


52  CREATE TABLE  T07A.STORE      (
53   STORE_ID  INTEGER  NOT NULL,
54   STORE_TYPE  CHAR(4) ,
55   REGION_ID  INTEGER  NOT NULL,
56   STORE_NAME  VARCHAR(40) ,
57   STORE_NUMBER  NAME  VARCHAR(10) NOT NULL,
58   STORE_STREET_ADDRESS  VARCHAR(120)  NOT NULL,
59   STORE_CITY  VARCHAR(50) ,
60   STORE_STATE  VARCHAR(50) ,
61   STORE_POSTAL_CODE  CHAR(6) ,
62   STORE_COUNTRY  VARCHAR(50) ,
63   STORE_MANAGER  VARCHAR(16) ,
64   STORE_PHONE  VARCHAR(16) ,
65   STORE_FAX  VARCHAR(16) ,
66   FIRST_OPENED_DATE  DATE ,
67   LAST_REMODEL_DATE  DATE ,
68   LEASE_SQFT  DOUBLE ,
69   STORE_SQFT  DOUBLE  NOT NULL,
70   GROCERY_SQFT  DOUBLE,
71   FROZEN_SQFT  DOUBLD ,
```

```
72   MEAT_SQFT  DOUBLE ,
73   COFFEE_BAR  CHAR(1) ,
74   VIDEO_STORE  CHAR(1) ,
75   SALAD_BAR  CHAR(1) ,
76   PREPARED_FOOD  CHAR(1) ,
77   FLORIST  CHAR(1) ,
     PRIMARY KEY (STORE_ID)  )  ;

78  CREATE TABLE  T07A.TIME_BY_DAY      (
79   TIME_ID  INTEGER  NOT NULL,
80   THE_DATE  DATE  NOT NULL,
81   THE_DAY  VARCHAR(15) ,
82   THE_MONTH  VARCHAR(15) ,
83   THE_YEAR  SMALLINT  NOT NULL,
84   DAY_OF_MONTH  SMALLINT ,
85   WEEK_OF_YEAR  SMALLINT  NOT NULL,
86   MONTH_OF_YEAR  SMALLINT  NOT NULL,
87   QUARTER  CHAR(2)  NOT NULL,
88   FISCAL_PERIOD  DATE  NOT NULL ,
     PRIMARY KEY (TIME_ID)   )   ;

89  CREATE TABLE  T07A.SALES      (
90   STORE_SALES  DOUBLE  NOT NULL,
91   STORE_COST  DOUBLE  NOT NULL,
92   UNIT_SALES  DOUBLE  NOT NULL,
93   STORE_ID  INTEGER ,
94   PRODUCT_ID  INTEGER ,
95   PROMOTION_ID  INTEGER ,
96   CUSTOMER_ID  INTEGER ,
97   TIME_ID  INTEGER ,
     FOREIGN KEY (STORE_ID) REFERENCES T07A.STORE(STORE_ID) ,
     FOREIGN KEY (PRODUCT_ID) REFERENCES T07A.PRODUCT(PRODUCT_ID) ,
     FOREIGN KEY (PROMOTION_ID) REFERENCES T07A.PROMOTION(PROMOTION_ID),
     FOREIGN KEY (CUSTOMER_ID) REFERENCES T07A.CUSTOMER(CUSTOMER_ID) ,
     FOREIGN KEY (TIME_ID) REFERENCES T07A.TIME_BY_DAY(TIME_ID)      );
```

Pair 12

Right Hand Schema Id: T09A

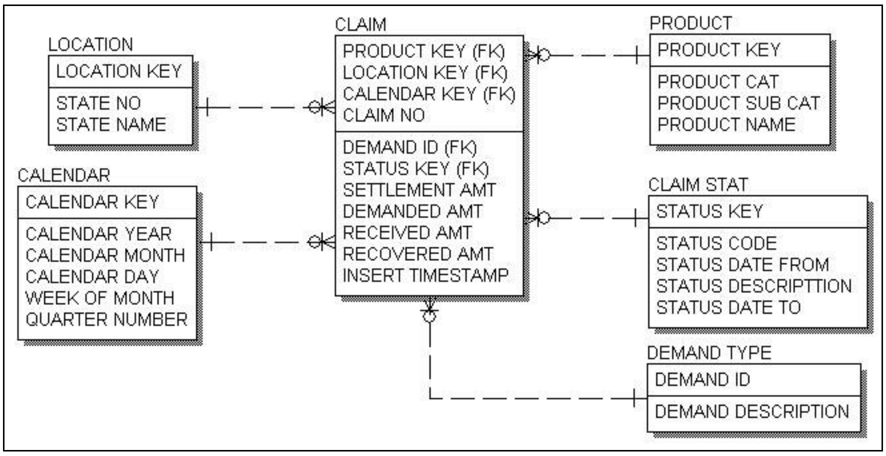


<u>DDL</u>

```
a   CREATE TABLE T09A.ON_SALE (
b      PRODUCT_KEY           INTEGER NOT NULL,
c      SKU_NUMBER            INTEGER,
d      DESCRIPTION           VARCHAR(80),
e      BRAND                 VARCHAR(20),
f      CATEGORY              VARCHAR(30),
g      PACKAGE_TYPE          CHAR(3),
       PRIMARY KEY (PRODUCT_KEY)    );


h   CREATE TABLE T09A.PRODUCT_SALES (
i      QTY_RECEIVED          SMALLINT,
j      PRODUCT_KEY           INTEGER,
k      QTY_LOST              SMALLINT,
l      QTY_DAMAGED           SMALLINT,
m      UNIT_COST             DECIMAL(11,2),
n      UNIT_LIST_PRICE       DECIMAL(11,2),
       FOREIGN KEY (PRODUCT_KEY) REFERENCES T09A.ON_SALE    );
```

Pair 13

Left Hand Schema Id: A01



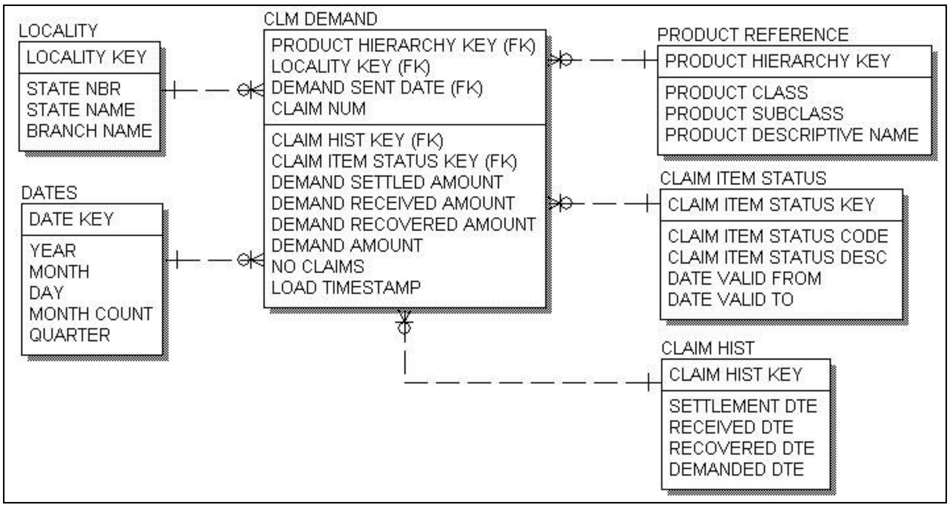
DDL

```
1   CREATE TABLE A01.DEMAND_TYPE (
2     DEMAND_ID            CHAR(4) NOT NULL,
3     DEMAND_DESCRIPTION   VARCHAR(40),
      PRIMARY KEY (DEMAND_ID)    );

4   CREATE TABLE A01.CLAIM_STAT (
5      STATUS_KEY           SMALLINT NOT NULL,
6      STATUS_CODE          CHAR(4),
7      STATUS_DESCRIPTTION  VARCHAR(40),
8      STATUS_DATE_FROM     DATE,
9      STATUS_DATE_TO       DATE,
       PRIMARY KEY (STATUS_KEY)    );

10  CREATE TABLE A01.PRODUCT (
11     PRODUCT_KEY          SMALLINT NOT NULL,
12     PRODUCT_CAT          CHAR(5),
13     PRODUCT_SUB_CAT      CHAR(5),
14     PRODUCT_NAME         VARCHAR(40),
       PRIMARY KEY (PRODUCT_KEY)    );

15  CREATE TABLE A01.LOCATION (
16     LOCATION_KEY         SMALLINT NOT NULL,
17     STATE_NO             SMALLINT,
18     STATE_NAME           VARCHAR(20),
       PRIMARY KEY (LOCATION_KEY)    );

19  CREATE TABLE A01.CALENDAR (
20    CALENDAR_KEY         DATE NOT NULL,
21    CALENDAR_YEAR        SMALLINT,
22    CALENDAR_MONTH       SMALLINT,
23    CALENDAR_DAY         SMALLINT,
```

```
24    WEEK_OF_MONTH          SMALLINT,
25    QUARTER_NUMBER         SMALLINT,
      PRIMARY KEY (CALENDAR_KEY)    );

26  CREATE TABLE A01.CLAIM (
27    PRODUCT_KEY            SMALLINT NOT NULL,
28    LOCATION_KEY           SMALLINT NOT NULL,
29    CALENDAR_KEY           DATE NOT NULL,
30    CLAIM_NO               integer NOT NULL,
31    DEMAND_ID              CHAR(4) NOT NULL,
32    STATUS_KEY             SMALLINT NOT NULL,
33    SETTLEMENT_AMT         DECIMAL(11,2),
34    DEMANDED_AMT           DECIMAL(11,2),
35    RECEIVED_AMT           DECIMAL(11,2),
36    RECOVERED_AMT          DECIMAL(11,2),
37    INSERT_TIMESTAMP       TIMESTAMP,
      PRIMARY KEY (PRODUCT_KEY, LOCATION_KEY, CALENDAR_KEY,
             CLAIM_NO),
      FOREIGN KEY (DEMAND_ID) REFERENCES A01.DEMAND_TYPE,
      FOREIGN KEY (STATUS_KEY) REFERENCES A01.CLAIM_STAT,
      FOREIGN KEY (PRODUCT_KEY) REFERENCES A01.PRODUCT,
      FOREIGN KEY (LOCATION_KEY)  REFERENCES A01.LOCATION,
      FOREIGN KEY (CALENDAR_KEY) REFERENCES A01.CALENDAR    );
```

Pair 13

Right Hand Schema Id: A02



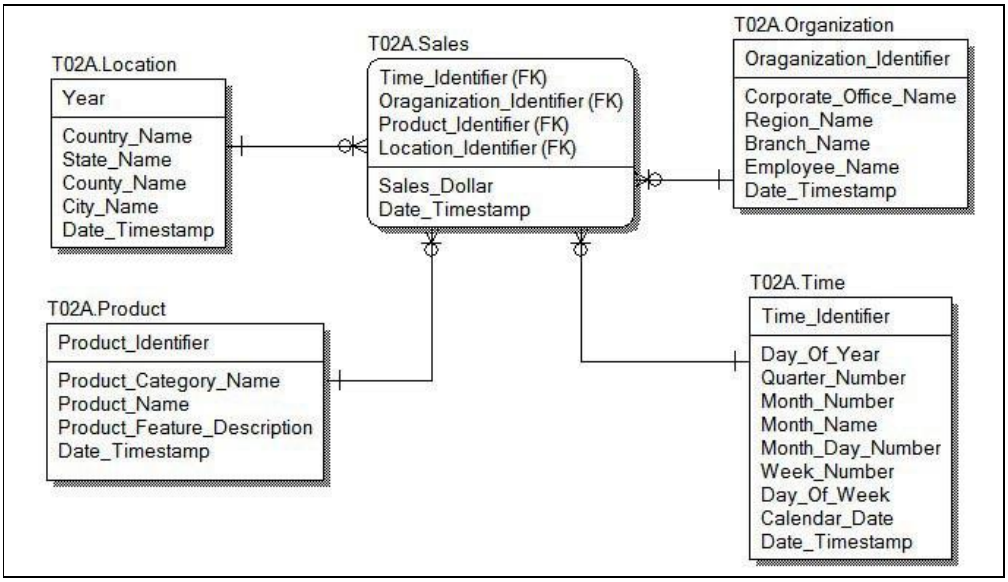
DDL

```
a   CREATE TABLE A02.CLAIM_HIST (
b     CLAIM_HIST_KEY    INTEGER NOT NULL,
c     SETTLEMENT_DTE    DATE,
d     RECEIVED_DTE      DATE,
e     RECOVERED_DTE     DATE,
f     DEMANDED_DTE      DATE,
      PRIMARY KEY (CLAIM_HIST_KEY)    );


g   CREATE TABLE A02.PRODUCT_REFERENCE (
h     PRODUCT_HIERARCHY_KEY SMALLINT NOT NULL,
i     PRODUCT_CLASS         CHAR(4),
j     PRODUCT_SUBCLASS      CHAR(5),
k     PRODUCT_DESCRIPTIVE_NAME VARCHAR(40),
      RIMARY KEY (PRODUCT_HIERARCHY_KEY)    );

l   CREATE TABLE A02.CLAIM_ITEM_STATUS (
m     CLAIM_ITEM_STATUS_KEY SMALLINT NOT NULL,
n     CLAIM_ITEM_STATUS_CODE CHAR(4),
o     CLAIM_ITEM_STATUS_DESC VARCHAR(40),
p     DATE_VALID_FROM       DATE,
q     DATE_VALID_TO         DATE,
      PRIMARY KEY (CLAIM_ITEM_STATUS_KEY)    );

r   CREATE TABLE A02.LOCALITY (
s     LOCALITY_KEY          SMALLINT NOT NULL,
t     STATE_NBR             SMALLINT,
u     STATE_NAME            VARCHAR(20),
v     BRANCH_NAME           VARCHAR(30),
      PRIMARY KEY (LOCALITY_KEY)    );


w   CREATE TABLE A02.DATES (
x     DATE_KEY              DATE NOT NULL,
y     YEAR                  SMALLINT,
z     MONTH                 SMALLINT,
aa    DAY                   SMALLINT,
bb    MONTH_COUNT           SMALLINT,
cc    QUARTER               SMALLINT,
      PRIMARY KEY (DATE_KEY)    );

dd  CREATE TABLE A02.CLM_DEMAND (
ee    PRODUCT_HIERARCHY_KEY SMALLINT NOT NULL,
ff    LOCALITY_KEY          SMALLINT NOT NULL,
gg    DEMAND_SENT_DATE      DATE NOT NULL,
hh    CLAIM_NUM             INTEGER NOT NULL,
ii    CLAIM_HIST_KEY        INTEGER,
jj    CLAIM_ITEM_STATUS_KEY SMALLINT NOT NULL,
kk    DEMAND_SETTLED_AMOUNT   DECIMAL(11,2),
ll    DEMAND_RECEIVED_AMOUNT  DECIMAL(11,2),
mm    DEMAND_RECOVERED_AMOUNT DECIMAL(11,2),
nn    NO_CLAIMS             SMALLINT,
oo    LOAD_TIMESTAMP        TIMESTAMP,
      PRIMARY KEY (PRODUCT_HIERARCHY_KEY, LOCALITY_KEY,
             DEMAND_SENT_DATE, CLAIM_NUM),
```

```
      FOREIGN KEY (CLAIM_HIST_KEY) REFERENCES A02.LATEST_DEMAND,
      FOREIGN KEY (PRODUCT_HIERARCHY_KEY)  REFERENCES A02.PRODUCT_REFERENCE,
      FOREIGN KEY (CLAIM_ITEM_STATUS_KEY) REFERENCES A02.CLAIM_ITEM_STATUS,
      FOREIGN KEY (LOCALITY_KEY)  REFERENCES A02.LOCALITY,
      FOREIGN KEY (DEMAND_SENT_DATE) REFERENCES A02.DATES    );
```

Pair 14

Left Hand Schema Id: T02A



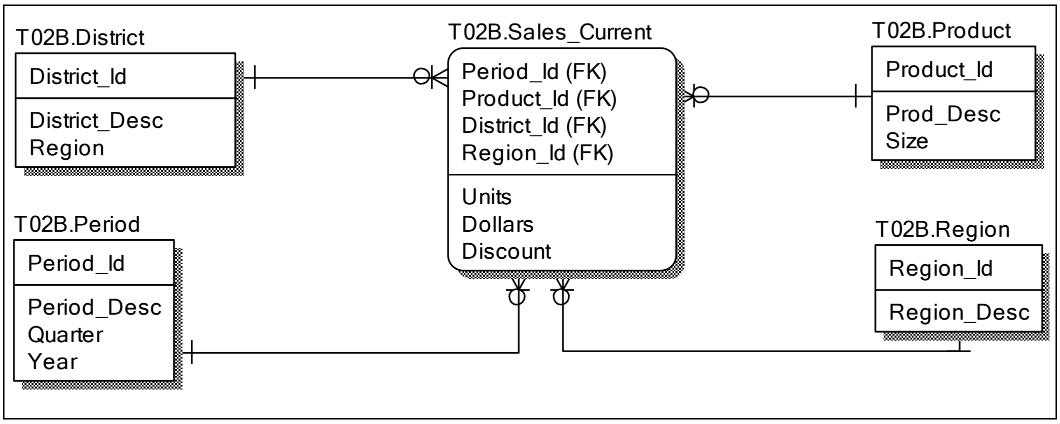
DDL

```
1   CREATE TABLE A01.DEMAND_TYPE (
2     DEMAND_ID             CHAR(4) NOT NULL,
3     DEMAND_DESCRIPTION    VARCHAR(40),
      PRIMARY KEY (DEMAND_ID)    );

4   CREATE TABLE A01.CLAIM_STAT (
5      STATUS_KEY            SMALLINT NOT NULL,
6      STATUS_CODE           CHAR(4),
7      STATUS_DESCRIPTTION   VARCHAR(40),
8      STATUS_DATE_FROM      DATE,
9      STATUS_DATE_TO        DATE,
       PRIMARY KEY (STATUS_KEY)    );

10  CREATE TABLE A01.PRODUCT (
11     PRODUCT_KEY           SMALLINT NOT NULL,
12     PRODUCT_CAT           CHAR(5),
13     PRODUCT_SUB_CAT       CHAR(5),
```

```
   PRODUCT_NAME         VARCHAR(40),
   PRIMARY KEY (PRODUCT_KEY)    );

CREATE TABLE A01.LOCATION (
   LOCATION_KEY         SMALLINT NOT NULL,
   STATE_NO             SMALLINT,
   STATE_NAME           VARCHAR(20),
   PRIMARY KEY (LOCATION_KEY)    );

CREATE TABLE A01.CALENDAR (
  CALENDAR_KEY         DATE NOT NULL,
  CALENDAR_YEAR        SMALLINT,
  CALENDAR_MONTH       SMALLINT,
  CALENDAR_DAY         SMALLINT,
  WEEK_OF_MONTH        SMALLINT,
  QUARTER_NUMBER       SMALLINT,
  PRIMARY KEY (CALENDAR_KEY)    );

CREATE TABLE A01.CLAIM (
  PRODUCT_KEY          SMALLINT NOT NULL,
  LOCATION_KEY         SMALLINT NOT NULL,
  CALENDAR_KEY         DATE NOT NULL,
  CLAIM_NO             integer NOT NULL,
  DEMAND_ID            CHAR(4) NOT NULL,
  STATUS_KEY           SMALLINT NOT NULL,
  SETTLEMENT_AMT       DECIMAL(11,2),
  DEMANDED_AMT         DECIMAL(11,2),
  RECEIVED_AMT         DECIMAL(11,2),
  RECOVERED_AMT        DECIMAL(11,2),
  INSERT_TIMESTAMP     TIMESTAMP,
  PRIMARY KEY (PRODUCT_KEY, LOCATION_KEY, CALENDAR_KEY,
         CLAIM_NO),
  FOREIGN KEY (DEMAND_ID) REFERENCES A01.DEMAND_TYPE,
  FOREIGN KEY (STATUS_KEY) REFERENCES A01.CLAIM_STAT,
  FOREIGN KEY (PRODUCT_KEY) REFERENCES A01.PRODUCT,
  FOREIGN KEY (LOCATION_KEY)  REFERENCES A01.LOCATION,
  FOREIGN KEY (CALENDAR_KEY) REFERENCES A01.CALENDAR    );
```

Pair 14

Right Hand Schema Id: T02B



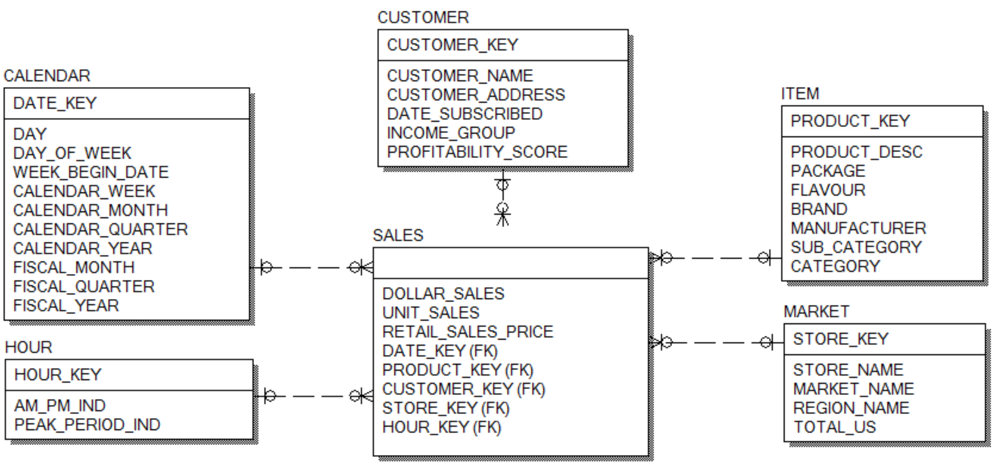
<u>DDL</u>

```
a   CREATE TABLE A02.CLAIM_HIST (
b     CLAIM_HIST_KEY     INTEGER NOT NULL,
c     SETTLEMENT_DTE     DATE,
d     RECEIVED_DTE       DATE,
e     RECOVERED_DTE      DATE,
f     DEMANDED_DTE       DATE,
      PRIMARY KEY (CLAIM_HIST_KEY)     );


g   CREATE TABLE A02.PRODUCT_REFERENCE (
h     PRODUCT_HIERARCHY_KEY SMALLINT NOT NULL,
i     PRODUCT_CLASS         CHAR(4),
j     PRODUCT_SUBCLASS      CHAR(5),
k     PRODUCT_DESCRIPTIVE_NAME VARCHAR(40),
      RIMARY KEY (PRODUCT_HIERARCHY_KEY)     );

l   CREATE TABLE A02.CLAIM_ITEM_STATUS (
m     CLAIM_ITEM_STATUS_KEY SMALLINT NOT NULL,
n     CLAIM_ITEM_STATUS_CODE CHAR(4),
o     CLAIM_ITEM_STATUS_DESC VARCHAR(40),
p     DATE_VALID_FROM       DATE,
q     DATE_VALID_TO         DATE,
      PRIMARY KEY (CLAIM_ITEM_STATUS_KEY)     );

r   CREATE TABLE A02.LOCALITY (
s     LOCALITY_KEY          SMALLINT NOT NULL,
t     STATE_NBR             SMALLINT,
u     STATE_NAME            VARCHAR(20),
v     BRANCH_NAME           VARCHAR(30),
      PRIMARY KEY (LOCALITY_KEY)     );
```

```
w   CREATE TABLE A02.DATES (
x     DATE_KEY                DATE NOT NULL,
y     YEAR                    SMALLINT,
z     MONTH                   SMALLINT,
aa    DAY                     SMALLINT,
bb    MONTH_COUNT             SMALLINT,
cc    QUARTER                 SMALLINT,
      PRIMARY KEY (DATE_KEY)      );

dd  CREATE TABLE A02.CLM_DEMAND (
ee    PRODUCT_HIERARCHY_KEY SMALLINT NOT NULL,
ff    LOCALITY_KEY            SMALLINT NOT NULL,
gg    DEMAND_SENT_DATE        DATE NOT NULL,
hh    CLAIM_NUM               INTEGER NOT NULL,
ii    CLAIM_HIST_KEY          INTEGER,
jj    CLAIM_ITEM_STATUS_KEY SMALLINT NOT NULL,
kk    DEMAND_SETTLED_AMOUNT    DECIMAL(11,2),
ll    DEMAND_RECEIVED_AMOUNT   DECIMAL(11,2),
mm    DEMAND_RECOVERED_AMOUNT DECIMAL(11,2),
nn    NO_CLAIMS               SMALLINT,
oo    LOAD_TIMESTAMP          TIMESTAMP,
      PRIMARY KEY (PRODUCT_HIERARCHY_KEY, LOCALITY_KEY,
             DEMAND_SENT_DATE, CLAIM_NUM),
      FOREIGN KEY (CLAIM_HIST_KEY) REFERENCES A02.LATEST_DEMAND,
      FOREIGN KEY (PRODUCT_HIERARCHY_KEY)  REFERENCES A02.PRODUCT_REFERENCE,
      FOREIGN KEY (CLAIM_ITEM_STATUS_KEY) REFERENCES A02.CLAIM_ITEM_STATUS,
      FOREIGN KEY (LOCALITY_KEY)  REFERENCES A02.LOCALITY,
      FOREIGN KEY (DEMAND_SENT_DATE) REFERENCES A02.DATES     );
```

Pair 15

Left Hand Schema Id: T10



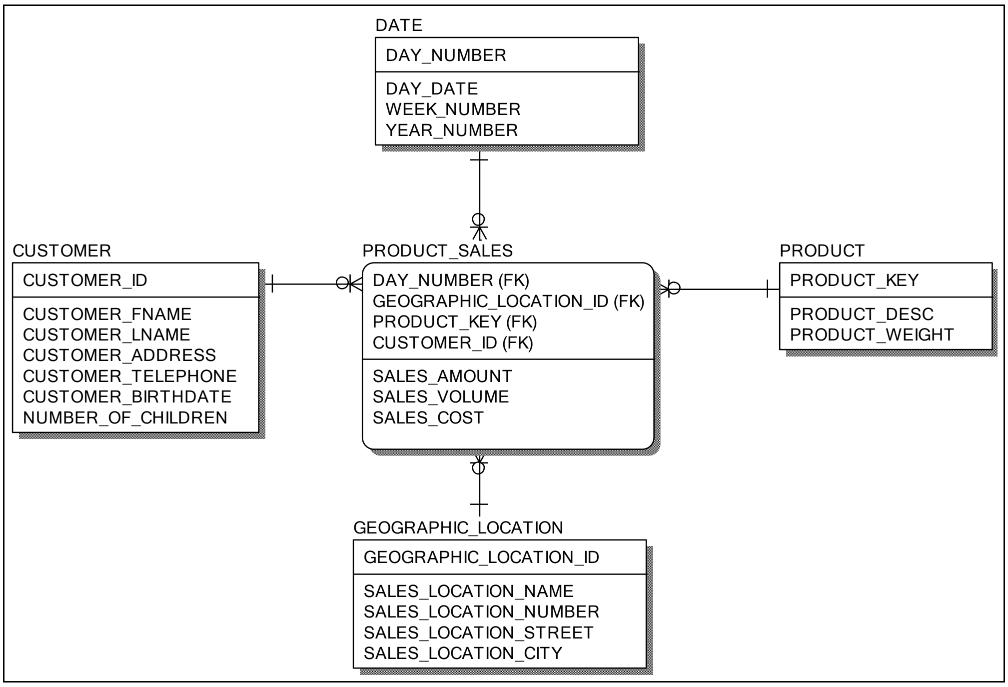
DDL

```
CREATE TABLE T10.CUSTOMER (
  CUSTOMER_KEY          INTEGER NOT NULL,
  CUSTOMER_NAME         VARCHAR(40),
  CUSTOMER_ADDRESS      VARCHAR(120),
  DATE_SUBSCRIBED       DATE,
  INCOME_GROUP          CHAR(1),
  PROFITABILITY_SCORE   SMALLINT,
  PRIMARY KEY (CUSTOMER_KEY)  );

CREATE TABLE T10.MARKET (
  STORE_KEY             INTEGER NOT NULL,
  STORE_NAME            VARCHAR(40),
  MARKET_NAME           VARCHAR(40),
  REGION_NAME           VARCHAR(40),
  TOTAL_US              DOUBLE PRECISION,
  PRIMARY KEY (STORE_KEY)   );

CREATE TABLE T10.ITEM (
  PRODUCT_KEY           INTEGER NOT NULL,
  PRODUCT_DESC          VARCHAR(60),
  PACKAGE               VARCHAR(20),
  FLAVOUR               CHAR(3),
  BRAND                 VARCHAR(40),
  MANUFACTURER          VARCHAR(40),
  SUB_CATEGORY          VARCHAR(40),
  CATEGORY              CHARACTER(40),
  PRIMARY KEY (PRODUCT_KEY)  );


CREATE TABLE T10.HOUR (
  HOUR_KEY              TIME NOT NULL,
  AM_PM_IND             CHAR(1),
  PEAK_PERIOD_IND       CHAR(1),
  PRIMARY KEY (HOUR_KEY)  );

CREATE TABLE T10.CALENDAR (
  DATE_KEY              DATE NOT NULL,
  DAY                   SMALLINT,
  DAY_OF_WEEK           SMALLINT,
  WEEK_BEGIN_DATE       DATE,
  CALENDAR_WEEK         SMALLINT,
  CALENDAR_MONTH        SMALLINT,
  CALENDAR_QUARTER      SMALLINT,
  CALENDAR_YEAR         SMALLINT,
  FISCAL_MONTH          SMALLINT,
  FISCAL_QUARTER        SMALLINT,
  FISCAL_YEAR           SMALLINT,
  PRIMARY KEY (DATE_KEY)  );


CREATE TABLE T10.SALES (
  CUSTOMER_KEY          INTEGER NOT NULL,
  STORE_KEY             INTEGER NOT NULL,
  PRODUCT_KEY           INTEGER NOT NULL,
```

```
44    HOUR_KEY             TIME NOT NULL,
45    DATE_KEY             DATE NOT NULL,
46    DOLLAR_SALES         DECIMAL(11,2),
47    UNIT_SALES           DECIMAL(11,2),
48    RETAIL_SALES_PRICE DECIMAL(11,2),
      FOREIGN KEY (CUSTOMER_KEY) REFERENCES T10.CUSTOMER,
      FOREIGN KEY (STORE_KEY) REFERENCES T10.MARKET,
      FOREIGN KEY (PRODUCT_KEY) REFERENCES T10.SALES_ITEM,
      FOREIGN KEY (HOUR_KEY) REFERENCES T10.HOUR,
      FOREIGN KEY (DATE_KEY) REFERENCES T10.CALENDAR );
```

Pair 15

Right Hand Schema Id: T11



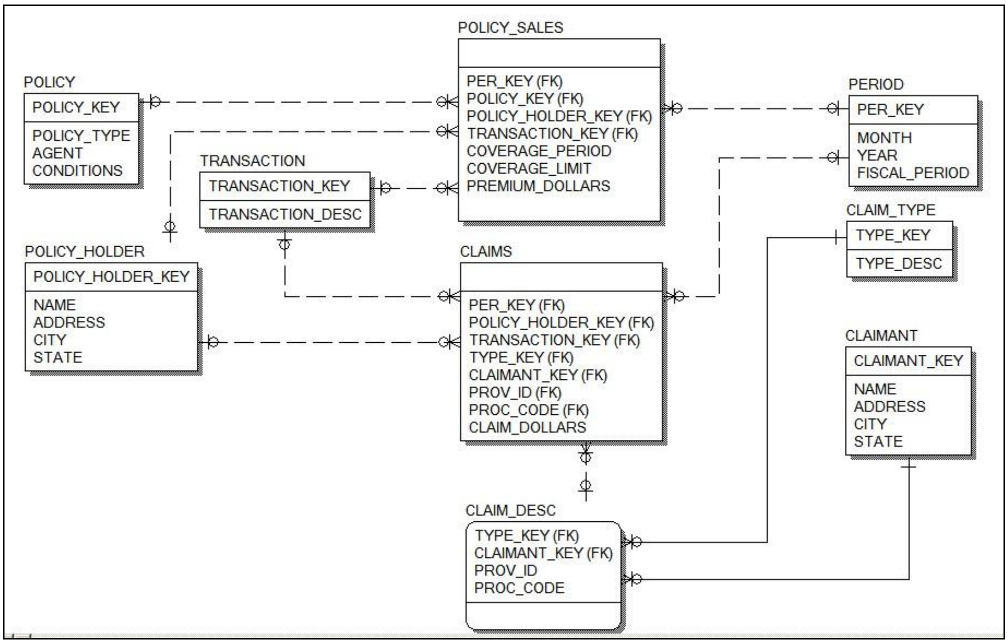
DDL

```
a   CREATE TABLE T11.GEOGRAPHIC_LOCATION (
b     SALES_LOCATION_ID       INTEGER NOT NULL,
c     SALES_LOCATION_NAME     VARCHAR(40),
d     SALES_LOCATION_NUMBER   SMALLINT,
e     SALES_LOCATION_STREET   VARCHAR(40),
f     SALES_LOCATION_CITY     VARCHAR(30),
      PRIMARY KEY (SALES_LOCATION_ID)
    );
```

```
g   CREATE TABLE T11.PRODUCT (
h     PRODUCT_KEY          INTEGER NOT NULL,
i     PRODUCT_DESC         VARCHAR(40),
j     PRODUCT_WEIGHT       DOUBLE,
      PRIMARY KEY (PRODUCT_KEY)
    );

k   CREATE TABLE T11.CUSTOMER (
l     CUSTOMER_ID          INTEGER NOT NULL,
m     CUSTOMER_FNAME       VARCHAR(40),
n     CUSTOMER_LNAME       VARCHAR(40),
o     CUSTOMER_ADDRESS     CHAR(120),
p     CUSTOMER_TELEPHONE   CHAR(10),
q     CUSTOMER_BIRTHDATE   DATE,
r     NUMBER_OF_CHILDREN   SMALLINT,
      PRIMARY KEY (CUSTOMER_ID)
    );


s   CREATE TABLE T11.DATE (
t     DAY_NUMBER           INTEGER NOT NULL,
u     DAY_DATE             DATE,
v     WEEK_NUMBER          SMALLINT,
w     MONTH_NUMBER         SMALLINT,
x     YEAR_NUMBER          SMALLINT,
      PRIMARY KEY (DAY_NUMBER)
    );

y   CREATE TABLE T11.PRODUCT_SALES (
z     SALES_LOCATION_ID    INTEGER NOT NULL,
aa    PRODUCT_KEY         INTEGER NOT NULL,
bb    CUSTOMER_ID          INTEGER NOT NULL,
cc    DAY_NUMBER           INTEGER NOT NULL,
dd    SALES_AMOUNT         DOUBLE PRECISION,
ee    SALES_VOLUME         SMALLINT,
ff    SALES_COST           DOUBLE PRECISION,
      FOREIGN KEY (SALES_LOCATION_ID) REFERENCES T11.GEOGRAPHIC_LOCATION,
      FOREIGN KEY (PRODUCT_CODE) REFERENCES T11.PRODUCT,
      FOREIGN KEY (CUSTOMER_ID)  REFERENCES T11.CUSTOMER,
      FOREIGN KEY (DAY_NUMBER)   REFERENCES T11.DATE
    );
```

Pair 16

Left Hand Schema Id: T05A



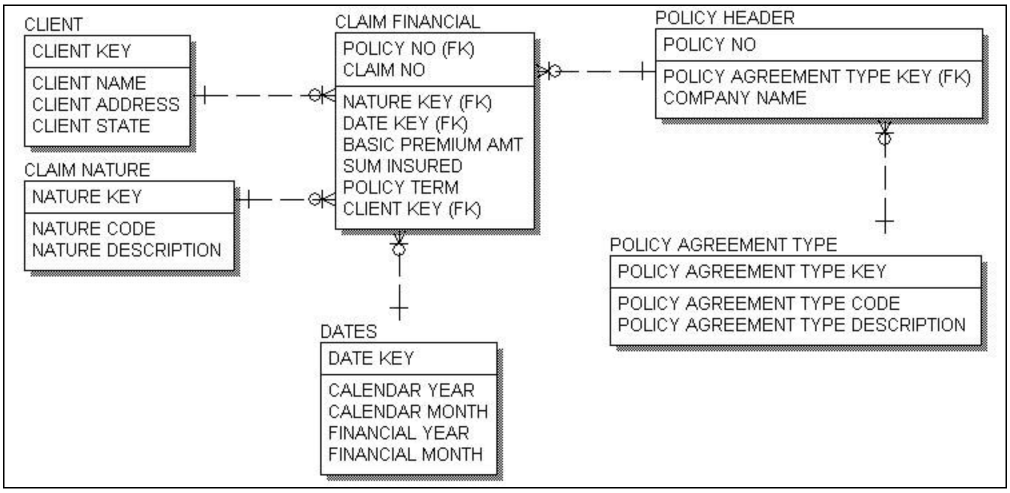
DDL

```
1   CREATE TABLE T05A.CLAIM_TYPE (
2    TYPE_KEY  SMALLINT  NOT NULL
3    TYPE_DESC  VARCHAR(40),
     PRIMARY KEY (TYPE_KEY)  )    ;

4   CREATE TABLE T05A.CLAIMANT (
5    CLAIMANT_KEY  INTEGER  NOT NULL
6    NAME  CHAR(40)  ,
7    ADDRESS  VARCHAR(60)  NOT NULL,
8    CITY  VARCHAR(20)  NOT NULL,
9    STATE  CHAR(4)  NOT NULL,
     PRIMARY KEY (CLAIMANT_KEY)   );

10  CREATE TABLE T05A.PERIOD (
11   PER_KEY  CHAR(06)  NOT NULL
12   MONTH  SMALLINT  NOT NULL,
13   YEAR  SMALLINT  NOT NULL,
14   FISCAL_PERIOD  DATE  NOT NULL,
     PRIMARY KEY (PER_KEY)   );

15  CREATE TABLE T05A.POLICY (
16   POLICY_KEY  INTEGER  NOT NULL,
17   POLICY_TYPE  CHAR(4)  NOT NULL,
18   AGENT  VARCHAR(40)  ,
```

```
19   CONDITIONS  VARCHAR(40)  NOT NULL,
     PRIMARY KEY (POLICY_KEY)   );

20  CREATE TABLE T05A.POLICY_HOLDER (
21   POLICY_HOLDER_KEY  INTEGER  NOT NULL,
22   NAME  VARCHAR(40)  NOT NULL,
23   ADDRESS  VARCHAR(60)  NOT NULL,
24   CITY  VARCHAR(40)  NOT NULL,
25   STATE  CHAR(4)  ,
     PRIMARY KEY (POLICY_HOLDER_KEY)   );

26  CREATE TABLE T05A.TRANSACTION (
27   TRANSACTION_KEY  INTEGER  NOT NULL,
28   TRANSACTION_DESC  VARCHAR(80)  NOT NULL,
     PRIMARY KEY (TRANSACTION_KEY)   );

29  CREATE TABLE T05A.CLAIM_DESC (
30   TYPE_KEY  SMALLINT  NOT NULL,
31   CLAIMANT_KEY  INTEGER  NOT NULL,
32   PROV_ID  SMALLINT  NOT NULL,
33   PROC_CODE  CHAR(4)  NOT NULL,
     PRIMARY KEY (TYPE_KEY,CLAIMANT_KEY,PROV_ID,PROC_CODE),
     FOREIGN KEY (TYPE_KEY) REFERENCES T05A.CLAIM_TYPE(TYPE_KEY),
     FOREIGN KEY (CLAIMANT_KEY) REFERENCES T05A.CLAIMANT(CLAIMANT_KEY)  );

37  CREATE TABLE T05A.CLAIMS (
38   CLAIM_DOLLARS  DECIMAL(11,2)  NOT NULL,
39   PER_KEY  SMALLINT  ,
40   POLICY_HOLDER_KEY  INTEGER  ,
41   TRANSACTION_KEY  INTEGER  ,
42   TYPE_KEY  SMALLINT  ,
43   CLAIMANT_KEY  INTEGER  ,
44   PROV_ID  SMALLINT  ,
45   PROC_CODE  CHAR(4)  ,
     FOREIGN KEY (PER_KEY) REFERENCES T05A.PERIOD(PER_KEY),
     FOREIGN KEY (POLICY_HOLDER_KEY) REFERENCES
                 T05A.POLICY_HOLDER(POLICY_HOLDER_KEY),
     FOREIGN KEY (TRANSACTION_KEY) REFERENCES
                 T05A.TRANSACTION(TRANSACTION_KEY),
     FOREIGN KEY (TYPE_KEY, CLAIMANT_KEY, PROV_ID, PROC_CODE)
                REFERENCES
            T05A.CLAIM_DESC(TYPE_KEY, CLAIMANT_KEY, PROV_ID, PROC_CODE)   );

46  CREATE TABLE T05A.POLICY_SALES (
47   PER_KEY  CHAR(6)  ,
48   POLICY_KEY  INTEGER  ,
49   POLICY_HOLDER_KEY  INTEGER  ,
50   TRANSACTION_KEY  INTEGER  ,
51   COVERAGE_PERIOD  SMALLINT  NOT NULL,
52   COVERAGE_LIMIT  INTEGER  NOT NULL,
53   PREMIUM_DOLLARS  DECIMAL(11),
     FOREIGN KEY (PER_KEY) REFERENCES T05A.PERIOD(PER_KEY),
     FOREIGN KEY (POLICY_KEY) REFERENCES T05A.POLICY(POLICY_KEY),
     FOREIGN KEY (POLICY_HOLDER_KEY) REFERENCES
                 T05A.POLICY_HOLDER(POLICY_HOLDER_KEY),
     FOREIGN KEY (TRANSACTION_KEY) REFERENCES
                 T05A.TRANSACTION(TRANSACTION_KEY) );
```

Pair 16

Right Hand Schema Id: T05B



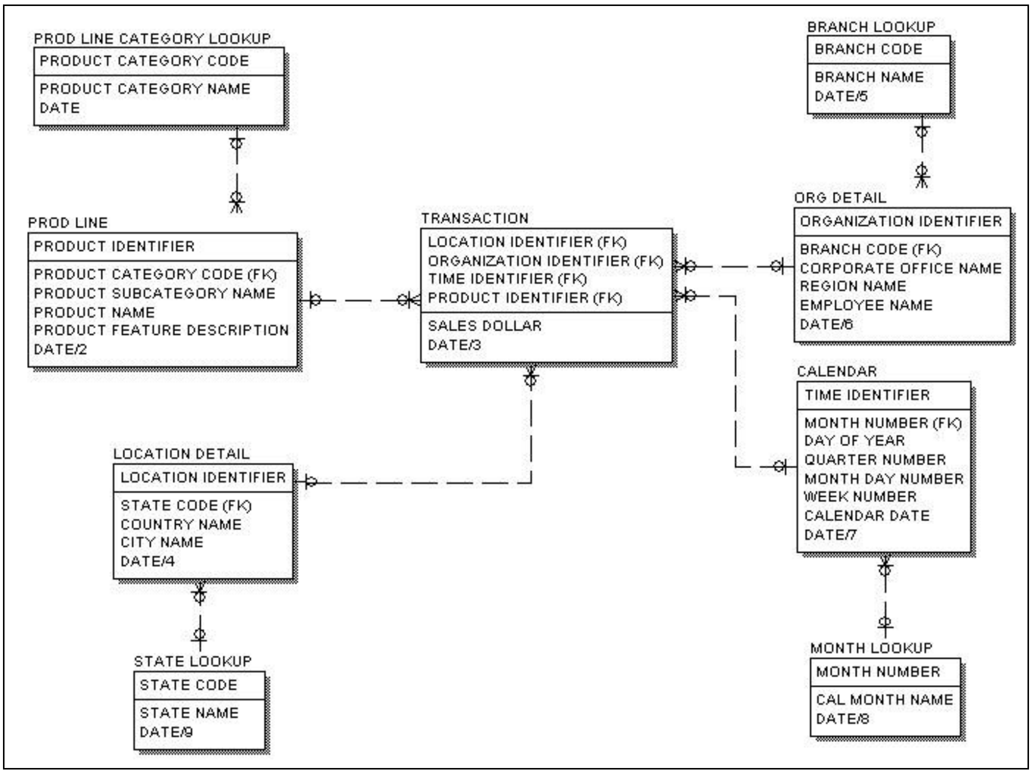
<u>DDL</u>

```
a   CREATE TABLE T05B.CLAIM_NATURE (
b     NATURE_KEY           SMALLINT NOT NULL,
c     NATURE_DESCRIPTION   CHARACTER(40),
d     NATURE_CODE          CHAR(4),
      PRIMARY KEY (NATURE_KEY)    );

e   CREATE TABLE T05B.POLICY_AGREEMENT_TYPE (
f     POLICY_AGREEMENT_TYPE_KEY INTEGER NOT NULL,
g     POLICY_AGREEMENT_TYPE_CODE CHAR(4),
h     POLICY_AGREEMENT_TYPE_DESC VARCHAR(40),
      PRIMARY KEY (POLICY_AGREEMENT_TYPE_KEY)    );


i   CREATE TABLE T05B.DATES (
j     DATE_KEY             DATE NOT NULL,
k     CALENDAR_YEAR        SMALLINT,
l     CALENDAR_MONTH       SMALLINT,
m     FINANCIAL_YEAR       SMALLINT,
n     FINANCIAL_MONTH      SMALLINT,
      PRIMARY KEY (DATE_KEY)    );
```

```
o   CREATE TABLE T05B.POLICY_HADER (
p     POLICY_NO            INTEGER NOT NULL,
q     COMPANY_NAME         VARCHAR(40),
r     POLICY_AGREEMENT_TYPE_KEY INTEGER,
      PRIMARY KEY (POLICY_NO),
      FOREIGN KEY (POLICY_AGREEMENT_TYPE_KEY)
                REFERENCES T05B.POLICY_AGREEMENT_TYPE    );

s   CREATE TABLE T05B.CLIENT (
t     CLIENT_KEY           INTEGER NOT NULL,
u     CLIENT_NAME          CHAR(18),
v     CLIENT_ADDRESS       VARCHAR(120),
w     CLIENT_STATE         CHAR(3),
      PRIMARY KEY (CLIENT_KEY)    );

x   CREATE TABLE T05B.CLAIM_FINANCIAL (
y     CLAIM_NO             INTEGER NOT NULL,
z     FINALISED_DATE       DATE NOT NULL,
aa    POLICY_NO            INTEGER NOT NULL,
bb    BASIC_PREMIUM_AMT    DECIMAL(11,2),
cc    NATURE_KEY           SMALLINT,
dd    SUM_INSURED          DECIMAL(11,2),
ee    POLICY_TERM          SMALLINT,
ff    CLIENT_KEY           INTEGER NOT NULL,
      PRIMARY KEY (POLICY_NO, CLAIM_NO),
      FOREIGN KEY (NATURE_KEY)  REFERENCES T05B.CLAIM_NATURE,
      FOREIGN KEY (FINALISED_DATE)  REFERENCES T05B.DATES,
      FOREIGN KEY (POLICY_NO)  REFERENCES T05B.POLICY_HADER,
      FOREIGN KEY (CLIENT_KEY)  REFERENCES T05B.CLIENT    );
```

Pair 17

Left Hand Schema Id: T06B



DDL

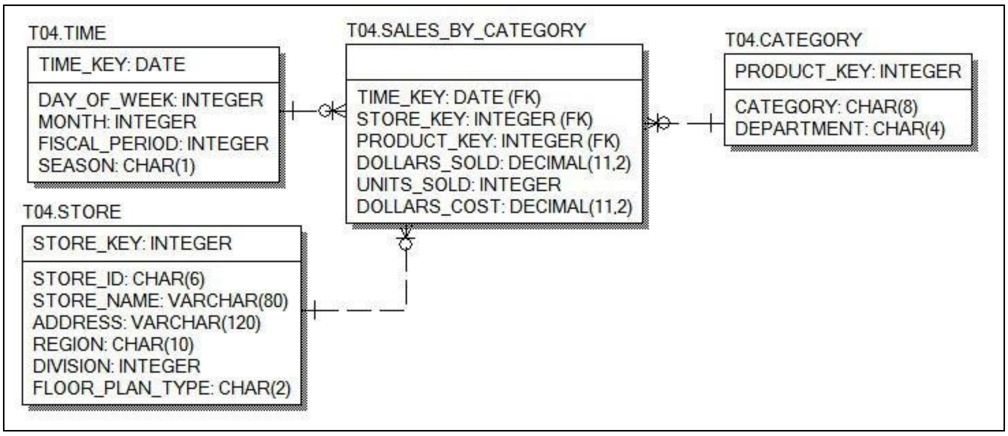
```
1   CREATE TABLE T06B.STATE_LOOKUP (
2      STATE_CODE             CHAR(3) NOT NULL,
3      STATE_NAME             VARCHAR(30),
4      DATE                   TIMESTAMP,
       PRIMARY KEY (STATE_CODE)    );

5   CREATE TABLE T06B.LOCATION_DETAIL (
6     LOCATION_IDENTIFIER  SMALLINT NOT NULL,
7     STATE_CODE           CHAR(3),
8     COUNTRY_NAME         VARCHAR(30),
9     CITY_NAME            VARCHAR(30) NOT NULL,
10    DATE                 TIMESTAMP NOT NULL,
      PRIMARY KEY (LOCATION_IDENTIFIER),
      FOREIGN KEY (STATE_CODE)  REFERENCES T06B.STATE_LOOKUP    );
```

```
11  CREATE TABLE T06B.BRANCH_LOOKUP (
12    BRANCH_CODE          CHAR(3) NOT NULL,
13    BRANCH_NAME          CHAR(40) NOT NULL,
14    DATE                 TIMESTAMP NOT NULL,
      PRIMARY KEY (BRANCH_CODE)    );

15  CREATE TABLE T06B.ORG_DETAIL (
16    ORGANIZATION_IDENTIFIER SMALLINT NOT NULL,
17    BRANCH_CODE          CHAR(3),
18    CORPORATE_OFFICE_NAME VARCHAR(40) NOT NULL,
19    REGION_NAME          VARCHAR(20) NOT NULL,
20    EMPLOYEE_NAME        VARCHAR(40) NOT NULL,
21    DATE                 TIMESTAMP NOT NULL,
      PRIMARY KEY (ORGANIZATION_IDENTIFIER),
      FOREIGN KEY (BRANCH_CODE)  REFERENCES T06B.BRANCH_LOOKUP    );

22  CREATE TABLE T06B.MONTH_LOOKUP (
23    MONTH_NUMBER         SMALLINT NOT NULL,
24    CAL_MONTH_NAME       CHAR(9) NOT NULL,
25    DATE                 TIMESTAMP NOT NULL,
      PRIMARY KEY (MONTH_NUMBER)    );

26  CREATE TABLE T06B.CALENDAR (
27    TIME_IDENTIFIER      DATE NOT NULL,
28    MONTH_NUMBER         SMALLINT,
29    DAY_OF_YEAR          SMALLINT NOT NULL,
30    QUARTER_NUMBER       SMALLINT NOT NULL,
31    MONTH_DAY_NUMBER     SMALLINT NOT NULL,
32    WEEK_NUMBER          SMALLINT NOT NULL,
33    CALENDAR_DATE        DATE NOT NULL,
34    DATE                 TIMESTAMP NOT NULL,
      PRIMARY KEY (TIME_IDENTIFIER),
      FOREIGN KEY (MONTH_NUMBER)   REFERENCES T06B.MONTH_LOOKUP    );

35  CREATE TABLE T06B.PROD_LINE_CATEGORY_LOOKUP (
36    PRODUCT_CATEGORY_CODE CHAR(4) NOT NULL,
37    PRODUCT_CATEGORY_NAME VARCHAR(40) NOT NULL,
38    DATE                 TIMESTAMP NOT NULL,
      PRIMARY KEY (PRODUCT_CATEGORY_CODE)    );

39  CREATE TABLE T06B.PROD_LINE (
40    PRODUCT_IDENTIFIER   INTEGER NOT NULL,
41    PRODUCT_CATEGORY_CODE CHAR(4),
42    PRODUCT_SUBCATEGORY_NAME VARCHAR(40) NOT NULL,
43    PRODUCT_NAME          VARCHAR(40) NOT NULL,
44    PRODUCT_FEATURE_DESCRIPTION VARCHAR(40) NOT NULL,
45    DATE                 TIMESTAMP NOT NULL,
      PRIMARY KEY (PRODUCT_IDENTIFIER),
            FOREIGN KEY (PRODUCT_CATEGORY_CODE)  REFERENCES
       T06B.PROD_LINE_CATEGORY_LOOKUP    );

46  CREATE TABLE T06B.TRANSACTION (
47    LOCATION_IDENTIFIER  SMALLINT NOT NULL,
48    ORGANIZATION_IDENTIFIER SMALLINT NOT NULL,
49    TIME_IDENTIFIER      DATE NOT NULL,
50    PRODUCT_IDENTIFIER   INTEGER NOT NULL,
51    SALES_DOLLAR         DECIMAL(11,2),
52    DATE                 TIMESTAMP NOT NULL,
```

```
      PRIMARY KEY (LOCATION_IDENTIFIER, ORGANIZATION_IDENTIFIER,
             TIME_IDENTIFIER, PRODUCT_IDENTIFIER),
      FOREIGN KEY (LOCATION_IDENTIFIER)   REFERENCES T06B.LOCATION_DETAIL,
      FOREIGN KEY (ORGANIZATION_IDENTIFIER)   REFERENCES T06B.ORG_DETAIL,
      FOREIGN KEY (TIME_IDENTIFIER)  REFERENCES T06B.CALENDAR,
      FOREIGN KEY (PRODUCT_IDENTIFIER) REFERENCES T06B.PROD_LINE    );
```

Pair 17

Right Hand Schema Id: T04



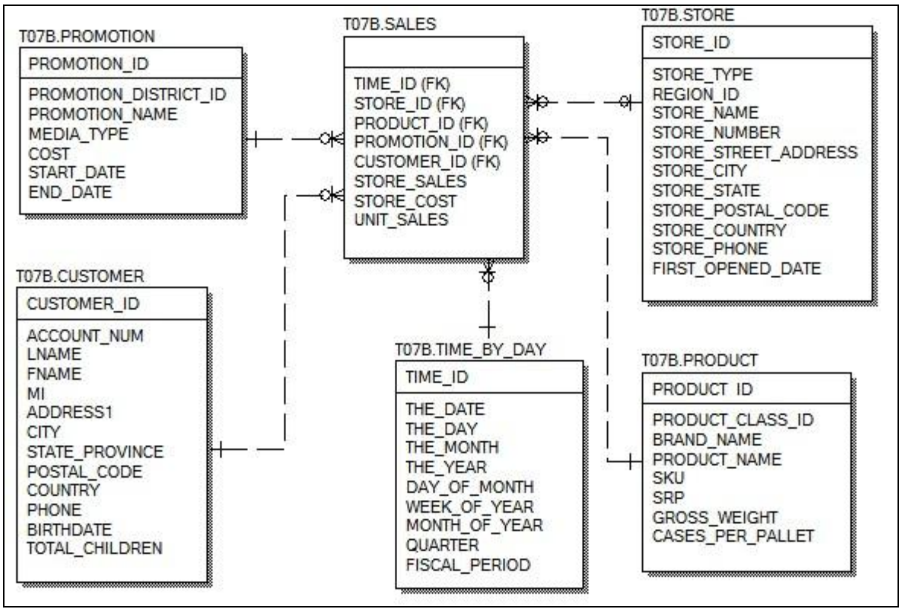
DDL

```
a    CREATE TABLE T04.CATEGORY    (
b       PRODUCT_KEY  INTEGER NOT NULL,
c       CATEGORY  CHAR(8) NOT NULL,
d       DEPARTMENT  CHAR(4) NOT NULL,
        PRIMARY KEY (PRODUCT_KEY)    );
e    CREATE TABLE T04.SALES_BY_CATEGORY    (
f       TIME_KEY  DATE NOT NULL,
g       STORE_KEY  INTEGER NOT NULL,
h       PRODUCT_KEY  INTEGER NOT NULL,
i       DOLLARS_SOLD  DECIMAL(11,2) NOT NULL,
j       UNITS_SOLD  INTEGER NOT NULL,
k       DOLLARS_COST  DECIMAL(11,2) NOT NULL,
        FOREIGN KEY (TIME_KEY) REFERENCES T04.TIME(TIME_KEY),
        FOREIGN KEY (STORE_KEY) REFERENCES T04.STORE(STORE_KEY),
        FOREIGN KEY (PRODUCT_KEY) REFERENCES T04.CATEGORY(PRODUCT_KEY) ) ;

l    CREATE TABLE T04.STORE    (
m       STORE_KEY  INTEGER NOT NULL,
n       STORE_ID  CHAR(6) NOT NULL,
o       STORE_NAME  VARCHAR(80) NOT NULL,
```

```
      ADDRESS  VARCHAR(120) NOT NULL,
      REGION  CHAR(10) NOT NULL,
      DIVISION  INTEGER NOT NULL,
      FLOOR_PLAN_TYPE  CHAR(2) NOT NULL,
      PRIMARY KEY (STORE_KEY)   );

   CREATE TABLE T04.TIME    (
      TIME_KEY  DATE NOT NULL,
      DAY_OF_WEEK  INTEGER NOT NULL,
      MONTH  INTEGER NOT NULL,
      FISCAL_PERIOD  INTEGER NOT NULL,
      SEASON  CHAR(1) NOT NULL,
      PRIMARY KEY (TIME_KEY)   );
```

Pair 18

Left Hand Schema Id: T07B



DDL

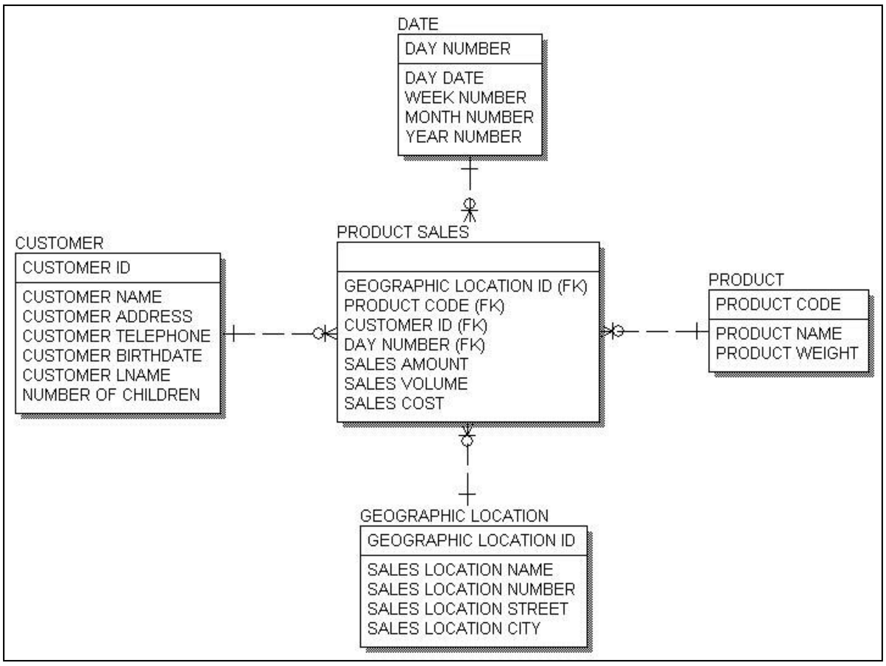
```
   CREATE TABLE  T07B.CUSTOMER    (
    CUSTOMER_ID  INTEGER  NOT NULL,
    ACCOUNT_NUM  DOUBLE  NOT NULL,
    LNAME  VARCHAR(100) ,
```

```
5     FNAME  VARCHAR(50) ,
6     MI  VARCHAR(20) ,
7     ADDRESS1  VARCHAR(100)  NOT NULL,
8     CITY  VARCHAR(50)  NOT NULL,
9     STATE_PROVINCE  VARCHAR(50)  NOT NULL,
10    POSTAL_CODE  CHAR(6)  NOT NULL,
11    COUNTRY  VARCHAR(50)  NOT NULL,
12    PHONE  VARCHAR(16)  NOT NULL,
13    BIRTHDATE  DATE  NOT NULL,
14    TOTAL_CHILDREN  SMALLINT  NOT NULL,
      PRIMARY KEY (CUSTOMER_ID)  )   ;

15   CREATE TABLE  T07B.PRODUCT      (
16    PRODUCT_ID  INTEGER  NOT NULL,
17    PRODUCT_CLASS_ID  INTEGER  NOT NULL,
18    BRAND_NAME  VARCHAR(40)  NOT NULL,
19    PRODUCT_NAME  VARCHAR(40)  NOT NULL,
20    SKU  DOUBLE  NOT NULL,
21    SRP  DOUBLE  NOT NULL,
22    GROSS_WEIGHT  DOUBLE  NOT NULL,
23    CASES_PER_PALLET  SMALLINT  NOT NULL,
      PRIMARY KEY (PRODUCT_ID)   )  ;


24   CREATE TABLE  T07B.STORE      (
25    STORE_ID  INTEGER  NOT NULL,
26    STORE_TYPE  CHAR(4) ,
27    REGION_ID  INTEGER  NOT NULL,
28    STORE_NAME  VARCHAR(40) ,
29    STORE_NUMBER  DOUBLE  NOT NULL,
30    STORE_STREET_ADDRESS  VARCHAR(120)  NOT NULL,
31    STORE_CITY  VARCHAR(50) ,
32    STORE_STATE  VARCHAR(50) ,
33    STORE_POSTAL_CODE  CHAR(6) ,
34    STORE_COUNTRY  VARCHAR(50) ,
35    STORE_PHONE  VARCHAR(16) ,
36    FIRST_OPENED_DATE  DATE ,
      PRIMARY KEY (STORE_ID)  ) ;

37   CREATE TABLE  T07B.TIME_BY_DAY      (
38    TIME_ID  INTEGER  NOT NULL,
39    THE_DATE  DATE  NOT NULL,
40    THE_DAY  SMALLINT  NOT NULL,
41    THE_MONTH  SMALLINT NOT NULL,
42    THE_YEAR  SMALLINT  NOT NULL,
43    DAY_OF_MONTH  SMALLINT NOT NULL,
44    WEEK_OF_YEAR  SMALLINT  NOT NULL,
45    MONTH_OF_YEAR  SMALLINT  NOT NULL,
46    QUARTER  CHAR(2)  NOT NULL,
47    FISCAL_PERIOD  DATE  NOT NULL ,
      PRIMARY KEY (TIME_ID)
     )  ;

48   CREATE TABLE  T07B.SALES      (
49    STORE_SALES  DOUBLE  NOT NULL,
50    STORE_COST  DOUBLE  NOT NULL,
51    UNIT_SALES  DOUBLE  NOT NULL,
52    STORE_ID  INTEGER ,
```

```
53    PRODUCT_ID  INTEGER ,
54    PROMOTION_ID  INTEGER ,
55    CUSTOMER_ID  INTEGER ,
56    TIME_ID  INTEGER ,
      FOREIGN KEY (STORE_ID) REFERENCES T07B.STORE(STORE_ID) ,
      FOREIGN KEY (PRODUCT_ID) REFERENCES T07B.PRODUCT(PRODUCT_ID) ,
      FOREIGN KEY (CUSTOMER_ID) REFERENCES T07B.CUSTOMER(CUSTOMER_ID) ,
      FOREIGN KEY (TIME_ID) REFERENCES T07B.TIME_BY_DAY(TIME_ID)  ) ;
```

Pair 18

Right Hand Schema Id: T11B



<u>DDL</u>

```
a   CREATE TABLE T11.GEOGRAPHIC_LOCATION (
b     SALES_LOCATION_ID      INTEGER NOT NULL,
c     SALES_LOCATION_NAME    VARCHAR(40),
d     SALES_LOCATION_NUMBER  SMALLINT,
e     SALES_LOCATION_STREET  VARCHAR(40),
f     SALES_LOCATION_CITY    VARCHAR(30),
      PRIMARY KEY (SALES_LOCATION_ID)
    );
```

```
g   CREATE TABLE T11.PRODUCT (
h     PRODUCT_CODE          INTEGER NOT NULL,
i     PRODUCT_NAME          VARCHAR(40),
j     PRODUCT_WEIGHT        DOUBLE,
      PRIMARY KEY (PRODUCT_CODE)
    );

k   CREATE TABLE T11.CUSTOMER (
l     CUSTOMER_ID           INTEGER NOT NULL,
m     CUSTOMER_FNAME        VARCHAR(40),
n     CUSTOMER_ADDRESS      CHAR(120),
o     CUSTOMER_TELEPHONE    CHAR(10),
p     CUSTOMER_BIRTHDATE    DATE,
q     CUSTOMER_LNAME        VARCHAR(40),
r     NUMBER_OF_CHILDREN    SMALLINT,
      PRIMARY KEY (CUSTOMER_ID)
    );


s   CREATE TABLE T11.DATE (
t     DAY_NUMBER            INTEGER NOT NULL,
u     DAY_DATE              DATE,
v     WEEK_NUMBER           SMALLINT,
w     MONTH_NUMBER          SMALLINT,
x     YEAR_NUMBER           SMALLINT,
      PRIMARY KEY (DAY_NUMBER)
    );

y   CREATE TABLE T11.PRODUCT_SALES (
z     SALES_LOCATION_ID     INTEGER NOT NULL,
aa    PRODUCT_CODE          INTEGER NOT NULL,
bb    CUSTOMER_ID           INTEGER NOT NULL,
cc    DAY_NUMBER            INTEGER NOT NULL,
dd    SALES_AMOUNT          DOUBLE PRECISION,
ee    SALES_VOLUME          SMALLINT,
ff    SALES_COST            DOUBLE PRECISION,
      FOREIGN KEY (SALES_LOCATION_ID) REFERENCES T11.GEOGRAPHIC_LOCATION,
      FOREIGN KEY (PRODUCT_CODE) REFERENCES T11.PRODUCT,
      FOREIGN KEY (CUSTOMER_ID)  REFERENCES T11.CUSTOMER,
      FOREIGN KEY (DAY_NUMBER)   REFERENCES T11.DATE
    );
```

Appendix C

Schema Match Results

This appendix contains match results for each pair of schemas described in appendix B. The results are in three parts:

i Match results returned from the Similarity Flooding algorithm when schemas were described using the relational model.

ii Match results returned from the Similarity Flooding algorithm when the same schemas were described using StarMod.

iii Match results returned from COMA algorithm when schemas were described using the relational model.

iv Match results returned from our participants in the evaluation. Only matches agree by at least two of the participants are included.

```
Matching Results between M7L and M7R using the Relational model:
Columns:
--------------------------------- ----------------------------         ------
PERSONNEL.BORN                    EMPLOYEE.BIRTHDATE                     0.1300
PERSONNEL.DEPT                    DEPARTMENT.DEPTNAME                    0.1274
PERSONNEL.PNAME                   EMPLOYEE.EMPNAME                       0.1356
PERSONNEL.PNO                     EMPLOYEE.EMPNO                         0.1648

Tables:
--------------------------------- ----------------------------         ------
PERSONNEL                         EMPLOYEE                               0.6174

Matching Results between M7L and M7R using the Star model:
Columns:
--------------------------------- ----------------------------         ------
PERSONNEL.BORN                    EMPLOYEE.BIRTHDATE                     0.0509
PERSONNEL.DEPT                    DEPARTMENT.DEPTNAME                    0.0550
PERSONNEL.PNAME                   EMPLOYEE.EMPNAME                       0.0567
PERSONNEL.PNO                     EMPLOYEE.EMPNO                         0.0531

Tables:
--------------------------------- ----------------------------         ------
PERSONNEL                         EMPLOYEE                               1

COMA Results between schemas M7L and M8R
----------------------------------------
PERSONNEL_TABLE.PERSONNEL.PNO <-> EMPLOYEE_TABLE.EMPLOYEE.EMPNO: 0.5140774
PERSONNEL_TABLE.PERSONNEL.PNAME <-> EMPLOYEE_TABLE.EMPLOYEE.EMPNAME: 0.6005358
PERSONNEL_TABLE.PERSONNEL.DEPT <-> DEPARTMENT_TABLE.DEPARTMENT.DEPTNO:0.5329773
PERSONNEL_TABLE <-> EMPLOYEE_TABLE: 0.5399605

Expected results agreed by at least 2 participants
-----------------------------------------------------------------------------
1,e     M07L.PERSONNEL                    M7R.EMPLOYEE
2,f     PERSONNEL.PNO                     EMPLOYEE.EMPNO
3,g     PERSONNEL.PNAME                   EMPLOYEE.EMPNAME
5,j     PERSONNEL.BORN                    EMPLOYEE.BIRTHDATE
4,c     PERSONNEL.DEPT                    DEPT.DEPTNAME


Matching Results between M8L and M9R using the Relational model:
Columns:
--------------------------------- ----------------------------         ------
PROFESSOR.ADDR                    PERSONNEL.ADDR                         0.1479
PROFESSOR.ID                      PERSONNEL.ID                           0.1270
PROFESSOR.NAME                    PERSONNEL.NAME                         0.0725
PROFESSOR.SAL                     PERSONNEL.SAL                          0.1806

Tables:
--------------------------------- ----------------------------         ------
PROFESSOR                         PERSONNEL                              0.3672

Matching Results between M8L and M9R using the Star model:
Columns:
--------------------------------- ----------------------------         ------
PROFESSOR.ADDR                    PERSONNEL.ADDR                         0.0683
PROFESSOR.ID                      PERSONNEL.ID                           0.0429
PROFESSOR.NAME                    PERSONNEL.NAME                         0.0363
PROFESSOR.SAL                     PERSONNEL.SAL                          0.1498
```

```
Tables:
--------------------------------- ----------------------------      ------
PROFESSOR                         PERSONNEL                           0.5272

COMA Results
----------------------------------------------------------------------------
WORKSON_TABLE.WORKSON.NAME <-> PERSONNEL_TABLE.PERSONNEL.NAME: 0.8220887
STUDENT_TABLE.STUDENT.NAME <-> PERSONNEL_TABLE.PERSONNEL.NAME: 0.8142762
PROFESSOR_TABLE.PROFESSOR.ID <-> PERSONNEL_TABLE.PERSONNEL.ID: 0.83748543
PROFESSOR_TABLE.PROFESSOR.NAME <-> PERSONNEL_TABLE.PERSONNEL.NAME: 0.8195635
PROFESSOR_TABLE.PROFESSOR.SAL <-> PERSONNEL_TABLE.PERSONNEL.SAL: 0.8802841
PROFESSOR_TABLE.PROFESSOR.ADDR <-> PERSONNEL_TABLE.PERSONNEL.ADDR: 0.8499777
PROFESSOR_TABLE <-> PERSONNEL_TABLE: 0.7577273

Expected results agreed by at least 2 participants
----------------------------------------------------------------------------
1,a     M8L.ADDRESS              M9R.PERSONNEL
3,e     ADDRESS.STREET           PERSONNEL.ADDR
4,e     ADDRESS.CITY             PERSONNEL.ADDR
5,e     ADDRESS.POSTALCODE       PERSONNEL.ADDR
6,a     M8L.PROFESSOR            M9R.PERSONNEL
7,b     PROFESSOR.ID             PERSONNEL.ID
8,c     PROFESSOR.NAME           PERSONNEL.NAME
9,d     PROFESSOR.SAL            PERSONNEL.SAL
12,c    STUDENT.NAME             PERSONNEL.NAME
19,c    WORKSON.NAME             PERSONNEL.NAME


Matching Results between R05 and R05A using the Relational model:
Columns:
----------------------------------------------------------------------------
ACCOUNT.ACCOUNT_ID ACCOUNT_HIST.ACC_ID 0.0463
ACCOUNT_BALANCE.AMT_BALANCE_HOLDS ACCOUNT_HIST.ACC_BALANCE_AMT 0.0771
ACCOUNT_BALANCE.DATE_BALANCE ACCOUNT_HIST.BALANCE_DATE 0.0808
REF_ACCOUNT_TYPES.ACCOUNT_TYPE_CODE ACCOUNT_HIST.ACCOUNT_TYPE_CODE 0.0776

Tables:
--------------------------------- ----------------------------      ------
ACCOUNT_BALANCE                   ACCOUNT_HIST                        0.4046

Matching Results between R05 and R05A using the Star model:
Columns:
----------------------------------------------------------------------------
ACCOUNT_BALANCE.AMT_BALANCE_HOLDS ACCOUNT_HIST.ACC_BALANCE_AMT 0.0607
ACCOUNT_BALANCE.DATE_BALANCE ACCOUNT_HIST.BALANCE_DATE 0.0451
ACCOUNT_TRANSACTIONS.ACCOUNT_ID ACCOUNT_HIST.ACC_ID 0.0395
REF_ACCOUNT_TYPES.ACCOUNT_TYPE_CODE ACCOUNT_HIST.ACCOUNT_TYPE_CODE 0.0557

Tables:
--------------------------------- ----------------------------      ------
ACCOUNT_TRANSACTIONS              ACCOUNT_HIST                        1

COMA Results
----------------------------------------------------------------------------
ACCOUNT_BALANCE_TABLE.ACCOUNT_BALANCE.DATE_BALANCE <->
ACCOUNT_HIST_TABLE.ACCOUNT_HIST.BALANCE_DATE: 0.86262786
```

```
ACCOUNT_BALANCE_TABLE.ACCOUNT_BALANCE.AMT_BALANCE_HOLDS <->
        ACCOUNT_HIST_TABLE.ACCOUNT_HIST.ACC_BALANCE_AMT: 0.72583765
ACCOUNT_BALANCE_TABLE.ACCOUNT_BALANCE.BALANCE_CLOSING <->
        ACCOUNT_HIST_TABLE.ACCOUNT_HIST.CLOSING_BAL_AMT: 0.65651226
ACCOUNT_TRANSACTIONS_TABLE.ACCOUNT_TRANSACTIONS.DATE_BALANCE <->
        ACCOUNT_HIST_TABLE.ACCOUNT_HIST.BALANCE_DATE: 0.86436397
ACCOUNT_TABLE.ACCOUNT.ACCOUNT_ID <->
        ACCOUNT_HIST_TABLE.ACCOUNT_HIST.ACC_ID: 0.7396621
ACCOUNT_TABLE.ACCOUNT.ACCOUNT_TYPE_CODE <->
ACCOUNT_HIST_TABLE.ACCOUNT_HIST.ACCOUNT_TYPE_CODE: 0.86590797
ACCOUNT_TABLE <-> ACCOUNT_HIST_TABLE: 0.74746954

Expected results agreed by at least 2 participants
-----------------------------------------------------------------------------
2,g     REF_ACCOUNT_TYPES.ACCOUNT_TYPE_CODE R05A.ACCOUNT_HIST.ACCOUNT_TYPE_CODE
5,b     ACCOUNT.ACCOUNT_ID                 ACCOUNT_HIST.ACC_ID
6,g     ACCOUNT.ACCOUNT_TYPE_CODE          ACCOUNT_HIST.ACCOUNT_TYPE_CODE
10,a    R05.ACCOUNT_BALANCE                R05A.ACCOUNT_HIST
12,c    ACCOUNT_BALANCE.DATE_BALANCE       ACCOUNT_HIST.BALANCE_DATE
13,d    ACCOUNT_BALANCE.AMT_BALANCE_HOLDS  ACCOUNT_HIST.ACC_BALANCE_AMT
15,f    ACCOUNT_BALANCE.BALANCE_ADJUSTED   ACCOUNT_HIST.ADJ_BAL_AMT
16,e    ACCOUNT_BALANCE.BALANCE_CLOSING    ACCOUNT_HIST.CLOSING_BAL_AMT


Matching Results between M8L and M8R using the Relational model:
Columns:
---------------------------------- ----------------------------         ------
PROFESSOR.ID                       PROFESSOR.ID                         0.1100
PROFESSOR.NAME                     PROFESSOR.NAME                       0.0375
PROFESSOR.SAL                      PROFESSOR.SALARY                     0.0509
STUDENT.GPA                        STUDENT.GRADEPOINTAVERAGE            0.0207
STUDENT.NAME                       STUDENT.NAME                         0.0384
STUDENT.YR                         STUDENT.YEAR                         0.0184
WORKSON.PROJ                       WORKSON.PROJECT                      0.0525
ADDRESS                            PROFESSOR.ADDRESS                    0.1257

Tables:
---------------------------------- ----------------------------         ------
PROFESSOR                          PROFESSOR                            0.3239
STUDENT                            STUDENT                              0.2049
WORKSON                            WORKSON                              0.2287

Matching Results between M8L and M8R using the Star model:
Columns:
---------------------------------- ----------------------------         ------
PROFESSOR.ID                       PROFESSOR.ID                         0.0281
PROFESSOR.NAME                     PROFESSOR.NAME                       0.0179
PROFESSOR.SAL                      PROFESSOR.SALARY                     0.0414
STUDENT.GPA                        STUDENT.GRADEPOINTAVERAGE            0.0269
STUDENT.NAME                       STUDENT.NAME                         0.0226
STUDENT.YR                         STUDENT.YEAR                         0.0242
WORKSON.HRS                        WORKSON.EXPENSES                     0.0190
WORKSON.NAME                       WORKSON.STUDENTNAME                  0.0168
WORKSON.PROJ                       WORKSON.PROJECT                      0.0214

Tables:
---------------------------------- ----------------------------         ------
PROFESSOR                          PROFESSOR                            0.2897
STUDENT                            STUDENT                              0.2408
```

```
WORKSON                           WORKSON                             0.2418

COMA Results
---------------------------------------------------------------------------
WORKSON_TABLE.WORKSON.PROJ <-> WORKSON_TABLE.WORKSON.PROJECT: 0.639833
WORKSON_TABLE.WORKSON.HRS <-> WORKSON_TABLE.WORKSON.EXPENSES: 0.4064996
WORKSON_TABLE <-> WORKSON_TABLE: 0.75411034
STUDENT_TABLE.STUDENT.NAME <-> STUDENT_TABLE.STUDENT.NAME: 0.91960126
STUDENT_TABLE.STUDENT.GPA <->
        STUDENT_TABLE.STUDENT.GRADEPOINTAVERAGE: 0.44814813
STUDENT_TABLE.STUDENT.YR <-> STUDENT_TABLE.STUDENT.YEAR: 0.631713
STUDENT_TABLE <-> STUDENT_TABLE: 0.82824075
PROFESSOR_TABLE.PROFESSOR.ID <-> PROFESSOR_TABLE.PROFESSOR.ID: 0.9234411
PROFESSOR_TABLE.PROFESSOR.NAME <-> PROFESSOR_TABLE.PROFESSOR.NAME: 0.91960126
PROFESSOR_TABLE.PROFESSOR.SAL <-> PROFESSOR_TABLE.PROFESSOR.SALARY: 0.67480767
PROFESSOR_TABLE.PROFESSOR.ADDR <->
        PROFESSOR_TABLE.PROFESSOR.ADDRESS: 0.6813034
PROFESSOR_TABLE <-> PROFESSOR_TABLE: 0.89705133

Expected results agreed by at least 2 participants
---------------------------------------------------------------------------
1,a     M8L.ADDRESS                       M8R.PROFESSOR
3,e     ADDRESS.STREET                    PROFESSOR.ADDRESS
4,e     ADDRESS.CITY                      ROFESSOR.ADDRESS
5,e     ADDRESS.POSTALCODE                PROFESSOR.ADDRESS
6,a     M8L.PROFESSOR                     M8R.PROFESSOR
7,b     PROFESSOR.ID                      PROFESSOR.ID
8,c     PROFESSOR.NAME                    PROFESSOR.NAME
9,d     PROFESSOR.SAL                     PROFESSOR.SALARY
11,f    M8L.STUDENT                       M8R.STUDENT
12,g    STUDENT.NAME                      STUDENT.NAME
12,k    STUDENT.NAME                      WORKSON.STUDENTNAME
13,h    STUDENT.GPA                       WORKSON.GRADEPOINTAVERAGE
14,i    STUDENT.YR                        STUDENT.YEAR
18,j    M8L.WORKSON                       M8R.WORKSON
19,k    WORKSON.NAME                      WORKSON.STUDENTNAME
20,l    WORKSON.PROJ                      WORKSON.PROJECT
21,m    WORKSON.HRS                       WORKSON.EXPENSES


Matching Results between R01 and R02 using the Relational model:
Columns:
--------------------------------- ----------------------------        ------
BOOKING.BOOKING_ID                ADDRESS.ADDRESS_ID                  0.0401
BOOKING.PAYMENT_RECEIVED          CARS_SOLD.MONTHLY_PAYMENT_AMOUNT    0.0405
CAR.CURRENT_MILEAGE               CARS_FOR_SALE.CURRENT_MILEAGE       0.0752
CUSTOMER.ADDRESS_LINE_1           ADDRESS.ADDRESS_LINE_1              0.0805
CUSTOMER.CITY                     ADDRESS.TOWN_CITY                   0.0514
CUSTOMER.EMAIL_ADDRESS            CUSTOMERS.EMAIL_ADDRESS             0.0761
CUSTOMER.STATE                    ADDRESS.STATE_COUNTY_PROVINCE       0.0386
MANUFACTURER.MANUFACTURER_NAME    CAR_MANUFACTURERS.MANUFACTURER_FULL_NAME 0.0647
MODEL.MODEL_CODE                  CAR_MODELS.MODEL_CODE               0.0252
MODEL.MODEL_NAME                  CAR_MODELS.MODEL_NAME               0.0776

Tables:
--------------------------------- ----------------------------        ------
BOOKING                           CARS_SOLD                           0.0508
CAR                               CAR_MODELS                          0.0805
CUSTOMER                          ADDRESS                             0.1121
```

```
Matching Results between R01 and R02 using the Star model:
Columns:
--------------------------------- ----------------------------        ------
BOOKING.CUSTOMER_ID               CARS_SOLD.CUSTOMER_ID                0.0166
BOOKING.DATE_OF_SERVICE           CARS_FOR_SALE.DATE_ACQUIRED          0.0181
BOOKING.PAYMENT_RECEIVED          CARS_SOLD.MONTHLY_PAYMENT_AMOUNT     0.0757
CAR.CURRENT_MILEAGE               CARS_FOR_SALE.CURRENT_MILEAGE        0.0634
CUSTOMER.CITY                     ADDRESS.TOWN_CITY                    0.0234
CUSTOMER.CUSTOMER_ID              CUSTOMERS.CUSTOMER_ID                0.0239
CUSTOMER.EMAIL_ADDRESS            CUSTOMERS.EMAIL_ADDRESS              0.0655
CUSTOMER.PHONE_NUMBER             CUSTOMERS.CAR_MOBILE_PHONE           0.0306
CUSTOMER.STATE                    ADDRESS.STATE_COUNTY_PROVINCE        0.0178
MANUFACTURER.MANUFACTURER_NAME    CAR_MANUFACTURERS.MANUFACTURER_FULL_NAME 0.0664
MODEL.MODEL_CODE                  CAR_MODELS.MODEL_CODE                0.0379
MODEL.MODEL_NAME                  CAR_MODELS.MODEL_NAME                0.0703

Tables:
--------------------------------- ----------------------------        ------
BOOKING                           CARS_SOLD                            0.3584
CAR                               CARS_FOR_SALE                        0.1325
CUSTOMER                          CUSTOMERS                            0.1845
MANUFACTURER                      CAR_MANUFACTURERS                    0.1259
MODEL                             CAR_MODELS                           0.1401

COMA Results
----------------------------------------------------------------------------
MODEL_TABLE.MODEL.MODEL_NAME <->
        CAR_MODELS_TABLE.CAR_MODELS.MODEL_NAME: 0.8650665
MODEL_TABLE <-> CAR_MODELS_TABLE: 0.69309604
CUSTOMER_TABLE.CUSTOMER.CUSTOMER_ID <->
        CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_ID: 0.88079137
CUSTOMER_TABLE.CUSTOMER.PHONE_NUMBER <->
CUSTOMERS_TABLE.CUSTOMERS.CAR_MOBILE_PHONE: 0.625968
CUSTOMER_TABLE.CUSTOMER.ADDRESS_LINE_1 <->
ADDRESS_TABLE.ADDRESS.ADDRESS_LINE_1: 0.8232952
CUSTOMER_TABLE.CUSTOMER.CITY <-> ADDRESS_TABLE.ADDRESS.TOWN_CITY: 0.62634003
CUSTOMER_TABLE.CUSTOMER.STATE <->
        ADDRESS_TABLE.ADDRESS.STATE_COUNTY_PROVINCE: 0.5606197
CUSTOMER_TABLE.CUSTOMER.EMAIL_ADDRESS <->
CUSTOMERS_TABLE.CUSTOMERS.EMAIL_ADDRESS: 0.88613033
CUSTOMER_TABLE <-> CUSTOMERS_TABLE: 0.77226067
MANUFACTURER_TABLE.MANUFACTURER.MANUFACTURER_NAME <->
CAR_MANUFACTURERS_TABLE.CAR_MANUFACTURERS.MANUFACTURER_FULL_NAME: 0.79365826
MANUFACTURER_TABLE <-> CAR_MANUFACTURERS_TABLE: 0.7768769
CAR_TABLE.CAR.MODEL_CODE <->
        CAR_MODELS_TABLE.CAR_MODELS.MODEL_CODE: 0.87084925
CAR_TABLE.CAR.CUSTOMER_ID <-> CARS_SOLD_TABLE.CARS_SOLD.CUSTOMER_ID: 0.8257048
CAR_TABLE.CAR.CURRENT_MILEAGE <->
CARS_FOR_SALE_TABLE.CARS_FOR_SALE.CURRENT_MILEAGE: 0.8133527
CAR_TABLE.CAR <-> CAR_MODELS_TABLE.CAR_MODELS: 0.6177288
BOOKING_TABLE.BOOKING.DATE_OF_SERVICE <->
        CARS_SOLD_TABLE.CARS_SOLD.DATE_SOLD: 0.5087609
BOOKING_TABLE.BOOKING.PAYMENT_RECEIVED <->
CARS_SOLD_TABLE.CARS_SOLD.MONTHLY_PAYMENT_AMOUNT: 0.49978667
```

```
Expected results agreed by at least 2 participants
----------------------------------------------------------------------------
1,p     R01.CUSTOMER                R02.CUSTOMERS
2,q     CUSTOMER.CUSTOMER_ID        CUSTOMERS.CUSTOMER_ID
6,s     CUSTOMER.EMAIL_ADDRESS      CUSTOMERS.EMAIL_ADDRESS
7,r     CUSTOMER.PHONE_NUMBER       CUSTOMERS.CAR_MOBILE_PHONE
8,cc    CUSTOMER.ADDRESS_LINE_1     ADDRESS.ADDRESS_LINE_1
9,dd    CUSTOMER.CITY               ADDRESS.TOWN_CITY
10,ee   CUSTOMER.STATE              ADDRESS.STATE_COUNTY_PROVINCE
11,a    R01.MODEL                   R02.CAR_MODELS
12,b    MODEL.MODEL_CODE            CAR_MODELS.MODEL_CODE
14,c    MODEL.MODEL_NAME            CAR_MODELS.MODEL_NAME
15,e    R01.MANUFACTURER            R02.CAR_MANUFACTURERS
16,f    MANUFACTURER.MANUFACTURER_CODE
                                    CAR_MANUFACTURERS.MANUFACTURER_SHORT_NAME
17,g    MANUFACTURER.MANUFACTURER_NAME
                                    CAR_MANUFACTURERS.MANUFACTURER_FULL_NAME
19,h    R01.CAR                     R02.CARS_FOR_SALE
22,k    CAR.MODEL_CODE              CARS_FOR_SALE.MODEL_CODE
23,l    CAR.CURRENT_MILEAGE         CARS_FOR_SALE.CURRENT_MILEAGE
25,m    CAR.MANUFACTURER_CODE       CARS_FOR_SALE.MANUFACTURER_SHORT_NAME
26,t    R01.BOOKING                 R02.CARS_SOLD
27,u    BOOKING.BOOKING_ID          CARS_SOLD.CAR_FOR_SALE_ID
28,v    BOOKING.CUSTOMER_ID         CARS_SOLD.CUSTOMER_ID
30,x    BOOKING.DATE_OF_SERVICE     CARS_SOLD.DATE_SOLD
207

Matching Results between R03 and R06 using the Relational model:
Columns:
--------------------------------- ----------------------------         ------
AUTHORS.AUTHOR_INITIALS           PRODUCTS.BOOK_AUTHOR                   0.0333
BOOKS.BOOK_PUBLICATION_DATE       PRODUCTS.PUBLICATION_DATE              0.0639
BOOKS.ISBN                        PRODUCTS.BOOK_ISBN                     0.0453
BOOKS.TITLE                       PRODUCTS.BOOK_TITLE                    0.0452
CONTACTS.CONTACT_FIRST_NAME       CUSTOMERS.FIRST_NAME                   0.0538
CONTACTS.CONTACT_LAST_NAME        CUSTOMERS.LAST_NAME                    0.0538
CUSTOMERS.CUSTOMER_EMAIL          CUSTOMERS.CUSTOMER_EMAIL               0.0670
CUSTOMERS.CUSTOMER_ID             CUSTOMERS.CUSTOMER_ID                  0.0253
CUSTOMERS.CUSTOMER_PHONE          CUSTOMERS.CUSTOMER_PHONE               0.0678
ORDER_ITEMS.AGREED_PRICE          PRODUCTS.PRODUCT_PRICE                 0.0462
ORDER_ITEMS.CUSTOMER_ID           CUSTOMER_ORDER.CUSTOMER_ID             0.0194
ORDER_ITEMS.ITEM_NUMBER           CUSTOMER_ORDERS_PRODUCT.QUANTITY       0.0205

Tables:
--------------------------------- ----------------------------         ------
BOOKS                             PRODUCTS                               0.0936
CUSTOMERS                         CUSTOMERS                              0.2386
ORDER_ITEMS                       CUSTOMER_ORDER                         0.0752

Matching Results between R03 and R06 using the Star model:
Columns:
--------------------------------- ----------------------------         ------
BOOKS.BOOK_ID                     PRODUCTS.PRODUCT_ID                    0.0377
BOOKS.BOOK_PUBLICATION_DATE       PRODUCTS.PUBLICATION_DATE              0.0643
BOOKS.ISBN                        PRODUCTS.BOOK_ISBN                     0.0458
BOOKS.TITLE                       PRODUCTS.BOOK_TITLE                    0.0457
CUSTOMERS.CUSTOMER_EMAIL          CUSTOMERS.CUSTOMER_EMAIL               0.0714
CUSTOMERS.CUSTOMER_ID             CUSTOMERS.CUSTOMER_ID                  0.0571
```

```
CUSTOMERS.CUSTOMER_PHONE          CUSTOMERS.CUSTOMER_PHONE            0.0719
ORDER_ITEMS.AGREED_PRICE          CUSTOMER_ORDERS_PRODUCT.QUANTITY    0.0830
ORDER_ITEMS.BOOK_ID               CUSTOMER_ORDERS_PRODUCT.PRODUCT_ID  0.0231

Tables:
--------------------------------- ----------------------------        ------
BOOKS                             PRODUCTS                            0.2834
CUSTOMERS                         CUSTOMERS                           0.5884
ORDER_ITEMS                       CUSTOMER_ORDERS_PRODUCT             0.2629

COMA Results
-----------------------------------------------------------------------------
ORDER_ITEMS_TABLE.ORDER_ITEMS.CUSTOMER_ID <->
        CUSTOMER_ORDER_TABLE.CUSTOMER_ORDER.CUSTOMER_ID: 0.87248325
ORDER_ITEMS_TABLE.ORDER_ITEMS.AGREED_PRICE <->
        CUSTOMER_ORDER_TABLE.CUSTOMER_ORDER.ORDER_PRICE: 0.5893995
ORDER_ITEMS_TABLE <-> CUSTOMER_ORDER_TABLE: 0.6626226
BOOKS_TABLE.BOOKS.ISBN <-> PRODUCTS_TABLE.PRODUCTS.BOOK_ISBN: 0.63474965
BOOKS_TABLE.BOOKS.TITLE <-> PRODUCTS_TABLE.PRODUCTS.BOOK_TITLE: 0.63474965
BOOKS_TABLE.BOOKS.BOOK_PUBLICATION_DATE <->
        PRODUCTS_TABLE.PRODUCTS.PUBLICATION_DATE: 0.6903354
BOOKS_TABLE <-> PRODUCTS_TABLE: 0.5840041
CONTACTS_TABLE.CONTACTS.CONTACT_ID <->
        PRODUCTS_TABLE.PRODUCTS.PRODUCT_ID: 0.5752502
CONTACTS_TABLE.CONTACTS.CONTACT_FIRST_NAME <->
        CUSTOMERS_TABLE.CUSTOMERS.FIRST_NAME: 0.73795813
CONTACTS_TABLE.CONTACTS.CONTACT_LAST_NAME <->
        CUSTOMERS_TABLE.CUSTOMERS.LAST_NAME: 0.73880535
AUTHORS_TABLE.AUTHORS.AUTHOR_INITIALS <->
        PRODUCTS_TABLE.PRODUCTS.BOOK_AUTHOR: 0.5216146
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_ID <->
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_ID: 0.9337578
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_PHONE <->
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_PHONE: 0.96267486
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_EMAIL <->
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_EMAIL: 0.96267486
CUSTOMERS_TABLE <-> CUSTOMERS_TABLE: 0.9253497

Expected results agreed by at least 2 participants
-----------------------------------------------------------------------------
1, a    R03.BOOKS                 R06.PRODUCTS
2, b    BOOKS.BOOK_ID             PRODUCTS.PRODUCT_ID
3, d    BOOKS.ISBN                PRODUCTS.BOOK_ISBN
4, g    BOOKS.TITLE               PRODUCTS.BOOK_TITLE
5, f    BOOKS.BOOK_PUBLICATION    PRODUCTS.PUBLICATION_DATE
6, k    R03.CUSTOMERS             R06.CUSTOMERS
7, l    CUSTOMERS.CUSTOMER_ID     CUSTOMERS.CUSTOMER_ID
9, m    CUSTOMERS.CUSTOMER_NAME   CUSTOMERS.FIRST_NAME
9, n    CUSTOMERS.CUSTOMER_NAME   CUSTOMERS.LAST_NAME
11, o   CUSTOMERS.CUSTOMER_PHONE  CUSTOMERS.CUSTOMER_PHONE
12, p   CUSTOMERS.CUSTOMER_EMAIL  CUSTOMERS.CUSTOMER_EMAIL
15, x   ORDER_ITEMS.BOOK_ID       CUSTOMER_ORDERS_PRODUCT.PRODUCT_ID
16, r   ORDER_ITEMS.CUSTOMER_ID   CUSTOMER_ORDER.CUSTOMER_ID

Matching Results between R06 and R07 using the Relational model:
Columns:
--------------------------------- ----------------------------        ------
CUSTOMERS.CUSTOMER_EMAIL          CLIENT.CUSTOMER_EMAIL               0.0693
```

```
CUSTOMERS.CUSTOMER_ID              CLIENT.CUSTOMER_ID                   0.0419
CUSTOMERS.CUSTOMER_PHONE           CLIENT.CUSTOMER_PHONE                0.0682
CUSTOMERS.FIRST_NAME               AUTHOR.AUTHOR_FIRST_NAME             0.0562
CUSTOMERS.LAST_NAME                AUTHOR.AUTHOR_LAST_NAME              0.0562
CUSTOMER_ORDERS_PRODUCT.QUANTITY   BOOK_AUTHOR.SEQ_NO                   0.0212
PRODUCTS.BOOK_AUTHOR               BOOK_AUTHOR                          0.0656
PRODUCTS.BOOK_TITLE                BOOK.BOOK_TITLE                      0.0693
PRODUCTS.PRODUCT_                  PRICE BOOK.BOOK_PRICE                0.0393
PRODUCTS.PUBLICATION_DATE          BOOK.PUBLICATION_DATE                0.0725

Tables:
---------------------------------- ----------------------------         ------
CUSTOMERS                          CLIENT                               0.1819
PRODUCTS                           BOOK                                 0.1313

Matching Results between R06 and R07 using the Star model:
Columns:
---------------------------------- ----------------------------         ------
CUSTOMERS.CUSTOMER_EMAIL           CLIENT.CUSTOMER_EMAIL                0.0397
CUSTOMERS.CUSTOMER_ID              CLIENT.CUSTOMER_ID                   0.0236
CUSTOMERS.CUSTOMER_PHONE           CLIENT.CUSTOMER_PHONE                0.0388
CUSTOMERS.FIRST_NAME               AUTHOR.AUTHOR_FIRST_NAME             0.0584
CUSTOMERS.LAST_NAME                AUTHOR.AUTHOR_LAST_NAME              0.0584
CUSTOMER_ORDERS_PRODUCT.PRODUCT_ID BOOK_AUTHOR.BOOK_ID                  0.0239
CUSTOMER_ORDERS_PRODUCT.QUANTITY   BOOK_AUTHOR.SEQ_NO                   0.0220
PRODUCTS.BOOK_TITLE                BOOK.BOOK_TITLE                      0.0713
PRODUCTS.PRODUCT_ID                BOOK.BOOK_ID                         0.0232
PRODUCTS.PRODUCT_PRICE             BOOK.BOOK_PRICE                      0.0426
PRODUCTS.PUBLICATION_DATE          BOOK.PUBLICATION_DATE                0.0722

Tables:
---------------------------------- ----------------------------         ------
CUSTOMER_ORDERS_PRODUCT            BOOK_AUTHOR                          0.2028
PRODUCTS                           BOOK                                 0.3913

COMA Results
----------------------------------------------------------------------------
CUSTOMER_ORDER_TABLE.CUSTOMER_ORDER.CUSTOMER_ID <->
        CLIENT_TABLE.CLIENT.CUSTOMER_ID: 0.8388858
CUSTOMER_ORDER_TABLE.CUSTOMER_ORDER.ORDER_ID <->
        AUTHOR_TABLE.AUTHOR.AUTHOR_ID: 0.55583584
PRODUCTS_TABLE.PRODUCTS.BOOK_ISBN <-> BOOK_TABLE.BOOK.BOOK_ID: 0.5712036
PRODUCTS_TABLE.PRODUCTS.PUBLICATION_DATE <->
        BOOK_TABLE.BOOK.PUBLICATION_DATE: 0.79273427
PRODUCTS_TABLE.PRODUCTS.BOOK_TITLE <->
        BOOK_TABLE.BOOK.BOOK_TITLE: 0.82702506
PRODUCTS_TABLE <-> BOOK_TABLE: 0.5806375
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_ID <->
        CLIENT_TABLE.CLIENT.CUSTOMER_ID: 0.847226
CUSTOMERS_TABLE.CUSTOMERS.FIRST_NAME <->
AUTHOR_TABLE.AUTHOR.AUTHOR_FIRST_NAME: 0.68796057
CUSTOMERS_TABLE.CUSTOMERS.LAST_NAME <->
        AUTHOR_TABLE.AUTHOR.AUTHOR_LAST_NAME: 0.68796057
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_PHONE <->
        CLIENT_TABLE.CLIENT.CUSTOMER_PHONE: 0.86897534
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_EMAIL <->
        CLIENT_TABLE.CLIENT.CUSTOMER_EMAIL: 0.86897534
CUSTOMERS_TABLE <-> CLIENT_TABLE: 0.70517427
```

```
CUSTOMER_ORDERS_PRODUCT_TABLE <-> BOOK_AUTHOR_TABLE: 0.4523942

Expected results agreed by at least 2 participants
----------------------------------------------------------------------------
1, p    R06.PRODUCTS                R07.BOOK
2, q    PRODUCTS.PRODUCT_ID         BOOK.BOOK_ID
3, t    PRODUCTS.PRODUCT_PRICE      BOOK.BOOK_PRICE
5, x    PRODUCTS.BOOK_AUTHOR        AUTHOR.AUTHOR_FIRST_NAME
5, y    PRODUCTS.BOOK_AUTHOR        AUTHOR.AUTHOR_INITIALS
5, z    PRODUCTS.BOOK_AUTHOR        AUTHOR.AUTHOR_LAST_NAME
6, u    PRODUCTS.PUBLICATION_DATE BOOK.PUBLICATION_DATE
7, s    PRODUCTS.BOOK_TITLE         BOOK.BOOK_TITLE
11, a   R06.CUSTOMERS               R07.CLIENT
12, b   CUSTOMERS.CUSTOMER_ID       CLIENT.CUSTOMER_ID
13, d   CUSTOMERS.FIRST_NAME        CLIENT.CUSTOMER_NAME
14, d   CUSTOMERS.LAST_NAME         CLIENT.CUSTOMER_NAME
15, f   CUSTOMERS.CUSTOMER_PHONE    CLIENT.CUSTOMER_PHONE
16, g   CUSTOMERS.CUSTOMER_EMAIL    CLIENT.CUSTOMER_EMAIL
209

Matching Results between R07 and R03 using the Relational model:
Columns:
--------------------------------- ----------------------------         ------
AUTHOR.AUTHOR_FIRST_NAME          AUTHORS.AUTHOR_FIRST_NAME              0.0578
AUTHOR.AUTHOR_INITIALS            AUTHORS.AUTHOR_INITIALS                0.0594
AUTHOR.AUTHOR_LAST_NAME           AUTHORS.AUTHOR_LAST_NAME               0.0578
BOOK.BOOK_PRICE                   ORDER_ITEMS.AGREED_PRICE               0.0300
BOOK.BOOK_TITLE                   BOOKS.TITLE                            0.0382
BOOK.PUBLICATION_DATE             BOOKS.BOOK_PUBLICATION_DATE            0.0502
BOOK_AUTHOR.SEQ_NO                ORDER_ITEMS.ITEM_NUMBER                0.0151
CLIENT.CUSTOMER_ADDRESS           CUSTOMERS.CUSTOMER_ADDRESS             0.0591
CLIENT.CUSTOMER_CODE              CUSTOMERS.CUSTOMER_CODE                0.0573
CLIENT.CUSTOMER_EMAIL             CUSTOMERS.CUSTOMER_EMAIL               0.0573
CLIENT.CUSTOMER_ID                CUSTOMERS.CUSTOMER_ID                  0.0346
CLIENT.CUSTOMER_NAME              CUSTOMERS.CUSTOMER_NAME                0.0561
CLIENT.CUSTOMER_PHONE             CUSTOMERS.CUSTOMER_PHONE               0.0561

Tables:
--------------------------------- ----------------------------         ------
AUTHOR                            AUTHORS                                0.1319
BOOK                              BOOKS                                  0.0703
CLIENT CUSTOMERS 0.2491

Matching Results between R07 and R03 using the Star model:
Columns:
--------------------------------- ----------------------------         ------
AUTHOR.AUTHOR_FIRST_NAME          AUTHORS.AUTHOR_FIRST_NAME              0.0398
AUTHOR.AUTHOR_ID                  AUTHORS.AUTHOR_ID                      0.0226
AUTHOR.AUTHOR_INITIALS            AUTHORS.AUTHOR_INITIALS                0.0405
AUTHOR.AUTHOR_LAST_NAME           AUTHORS.AUTHOR_LAST_NAME               0.0398
BOOK.BOOK_ID                      BOOKS.BOOK_ID                          0.0290
BOOK.BOOK_PRICE                   ORDER_ITEMS.AGREED_PRICE               0.0353
BOOK.BOOK_TITLE                   BOOKS.TITLE                            0.0484
BOOK.PUBLICATION_DATE             BOOKS.BOOK_PUBLICATION_DATE            0.0606
BOOK_AUTHOR.BOOK_ID               ORDER_ITEMS.BOOK_ID                    0.0210
BOOK_AUTHOR.SEQ_NO                ORDER_ITEMS.ITEM_NUMBER                0.0093
CLIENT.CUSTOMER_ADDRESS           CUSTOMERS.CUSTOMER_ADDRESS             0.0408
CLIENT.CUSTOMER_CODE              CUSTOMERS.CUSTOMER_CODE                0.0396
```

```
CLIENT.CUSTOMER_EMAIL              CUSTOMERS.CUSTOMER_EMAIL             0.0388
CLIENT.CUSTOMER_ID                 CUSTOMERS.CUSTOMER_ID                0.0230
CLIENT.CUSTOMER_NAME               CUSTOMERS.CUSTOMER_NAME              0.0380
CLIENT.CUSTOMER_PHONE              CUSTOMERS.CUSTOMER_PHONE             0.0382

Tables:
--------------------------------- ----------------------------         ------
BOOK                               BOOKS                                0.2292
BOOK_AUTHOR                        AUTHORS                              0.1146
CLIENT                             CUSTOMERS                            0.1820

COMA Results
----------------------------------------------------------------------------
AUTHOR_TABLE.AUTHOR.AUTHOR_ID <-> AUTHORS_TABLE.AUTHORS.AUTHOR_ID: 0.9300933
AUTHOR_TABLE.AUTHOR.AUTHOR_FIRST_NAME <->
        AUTHORS_TABLE.AUTHORS.AUTHOR_FIRST_NAME: 0.9632353
AUTHOR_TABLE.AUTHOR.AUTHOR_INITIALS <->
        AUTHORS_TABLE.AUTHORS.AUTHOR_INITIALS: 0.9632353
AUTHOR_TABLE.AUTHOR.AUTHOR_LAST_NAME <->
        AUTHORS_TABLE.AUTHORS.AUTHOR_LAST_NAME: 0.9632353
AUTHOR_TABLE <-> AUTHORS_TABLE: 0.9264706
CLIENT_TABLE.CLIENT.CUSTOMER_ID <->
        CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_ID: 0.87094575
CLIENT_TABLE.CLIENT.CUSTOMER_CODE <->
        CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_CODE: 0.90828025
CLIENT_TABLE.CLIENT.CUSTOMER_NAME <->
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_NAME: 0.9063005
CLIENT_TABLE.CLIENT.CUSTOMER_ADDRESS <->
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_ADDRESS: 0.90768933
CLIENT_TABLE.CLIENT.CUSTOMER_PHONE <->
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_PHONE: 0.9063005
CLIENT_TABLE.CLIENT.CUSTOMER_EMAIL <->
CUSTOMERS_TABLE.CUSTOMERS.CUSTOMER_EMAIL: 0.9063005
CLIENT_TABLE <-> CUSTOMERS_TABLE: 0.77982455
BOOK_TABLE.BOOK.BOOK_ID <-> BOOKS_TABLE.BOOKS.BOOK_ID: 0.87176013
BOOK_TABLE.BOOK.BOOK_TITLE <-> BOOKS_TABLE.BOOKS.TITLE: 0.71178776
BOOK_TABLE.BOOK.PUBLICATION_DATE <->
        BOOKS_TABLE.BOOKS.BOOK_PUBLICATION_DATE: 0.7795656
BOOK_TABLE <-> BOOKS_TABLE: 0.7624644

Expected results agreed by at least 2 participants
----------------------------------------------------------------------------
01, f   R07.CLIENT                 R03.CUSTOMERS
02, g   CLIENT.CUSTOMER_ID         CUSTOMERS.CUSTOMER_ID
03, h   CLIENT.CUSTOMER_CODE       CUSTOMERS.CUSTOMER_CODE
04, i   CLIENT.CUSTOMER_NAME       CUSTOMERS.CUSTOMER_NAME
05, j   CLIENT.CUSTOMER_ADDRESS    CUSTOMERS.CUSTOMER_ADDRESS
06, k   CLIENT.CUSTOMER_PHONE      CUSTOMERS.CUSTOMER_PHONE
07, l   CLIENT.CUSTOMER_EMAIL      CUSTOMERS.CUSTOMER_EMAIL
16, a   R07.BOOK                   R03.BOOKS
17, b   BOOK.BOOK_ID               BOOKS.BOOK_ID
19, d   BOOK.BOOK_TITLE            BOOKS.TITLE
21, e   BOOK.PUBLICATION_DATE      BOOKS.BOOK_PUBLICATION_DATE
22, x   R07.AUTHOR                 R03.AUTHORS
23, y   AUTHOR.AUTHOR_ID           AUTHORS.AUTHOR_ID
24, z   AUTHOR.AUTHOR_FIRST_NAME   AUTHORS.AUTHOR_FIRST_NAME
25, aa  AUTHOR.AUTHOR_INITIALS     AUTHORS.AUTHOR_INITALS
26, bb  AUTHOR.AUTHOR_LAST_NAME    AUTHORS.AUTHOR_LAST_NAME
```

```
28, b   BOOK_AUTHOR.BOOK_ID       BOOKS.BOOK_ID
28, o   BOOK_AUTHOR.BOOK_ID       ORDER_ITEMS.BOOK_ID

Matching Results between R08A and R08B using the Relational model:
Columns:
--------------------------------- ----------------------------        ------
ACCOUNT.ACCOUNT_NAME              APPLICANT.APPLICANT_NAME            0.0386
CUSTOMER.CUSTOMER_ADDRESS         APPLICANT.APPLICANT_ADDRESS         0.0389
CUSTOMER.CUSTOMER_PHONE           APPLICANT.APPLICANT_PHONE           0.0362
FINANCIAL_TRANSACTION.TRANSACTION_AMOUNT TRANSACTION.TRANSACTION_AMOUNT 0.0749
FINANCIAL_TRANSACTION.TRANSACTION_DATE TRANSACTION.TRANSACTION_DATE    0.0768

Tables:
--------------------------------- ----------------------------        ------
CUSTOMER                          APPLICANT                           0.0473
CUSTOMER_CARD                     CARD                                0.1536
FINANCIAL_TRANSACTION             TRANSACTION                         0.1593

Matching Results between R08A and R08B using the Star model:
Columns:
--------------------------------- ----------------------------        ------
ACCOUNT.ACCOUNT_NAME              APPLICANT.APPLICANT_NAME            0.0363
CUSTOMER.CUSTOMER_                ADDRESS APPLICANT.APPLICANT_ADDRESS 0.0364
CUSTOMER.CUSTOMER_PHONE           APPLICANT.APPLICANT_PHONE           0.0340
FINANCIAL_TRANSACTION.TRANSACTION_AMOUNT TRANSACTION.TRANSACTION_AMOUNT 0.1104
FINANCIAL_TRANSACTION.TRANSACTION_DATE TRANSACTION.TRANSACTION_DATE    0.0299
REF_CARD_TYPE.CARD_TYPE_CODE      CARD_TYPE.CARD_TYPE                 0.0452
REF_CARD_TYPE.CARD_TYPE_DESCRIPTION PAYMENT.PAYMENT_DESCRIPTION       0.0335

Tables:
--------------------------------- ----------------------------        ------
CUSTOMER                          APPLICANT                           0.1162
CUSTOMER_CARD                     CARD                                0.3277
FINANCIAL_TRANSACTION             TRANSACTION                         0.4860

COMA Results
-----------------------------------------------------------------------------
REF_CARD_TYPE_TABLE.REF_CARD_TYPE.CARD_TYPE_CODE <->
        CARD_TYPE_TABLE.CARD_TYPE.CARD_TYPE: 0.81683004
REF_CARD_TYPE_TABLE <-> CARD_TYPE_TABLE: 0.77969897
CUSTOMER_CARD_TABLE.CUSTOMER_CARD.CARD_TYPE_CODE <->
        CARD_TABLE.CARD.CARD_TYPE: 0.8051237
CUSTOMER_CARD_TABLE.CUSTOMER_CARD.CARD_ID <->
        CARD_TABLE.CARD.CARD_DEBT: 0.66209203
CUSTOMER_CARD_TABLE.CUSTOMER_CARD.CARD_NUMBER <->
        CARD_TABLE.CARD.CARD_NUMBER: 0.88136595
CUSTOMER_CARD_TABLE.CUSTOMER_CARD.DATE_VALID_FROM <->
        CARD_TABLE.CARD.CARD_EXPIRE_DATE: 0.596231
CUSTOMER_CARD_TABLE.CUSTOMER_CARD.DATE_VALID_TO <->
        CARD_TABLE.CARD.CARD_EXPIRE_DATE: 0.596231
CUSTOMER_CARD_TABLE <-> CARD_TABLE: 0.7763117
CUSTOMER_TABLE.CUSTOMER.CUSTOMER_PHONE <->
        APPLICANT_TABLE.APPLICANT.APPLICANT_PHONE: 0.5251432
CUSTOMER_TABLE.CUSTOMER.CUSTOMER_ADDRESS <->
        APPLICANT_TABLE.APPLICANT.APPLICANT_ADDRESS: 0.5378516
ACCOUNT_TABLE.ACCOUNT.ACCOUNT_ID <->
        APPLICANT_TABLE.APPLICANT.APPLICANT_ID: 0.6380226
ACCOUNT_TABLE.ACCOUNT.ACCOUNT_NAME <->
```

```
        APPLICANT_TABLE.APPLICANT.APPLICANT_NAME: 0.6395299
ACCOUNT_TABLE <-> APPLICANT_TABLE: 0.6348931
FINANCIAL_TRANSACTION_TABLE.FINANCIAL_TRANSACTION.TRANSACTION_ID <->
        TRANSACTION_TABLE.TRANSACTION.TRANSACTION_ID: 0.8932756
FINANCIAL_TRANSACTION_TABLE.FINANCIAL_TRANSACTION.TRANSACTION_AMOUNT <->
        TRANSACTION_TABLE.TRANSACTION.TRANSACTION_AMOUNT: 0.908187
FINANCIAL_TRANSACTION_TABLE.FINANCIAL_TRANSACTION.TRANSACTION_DATE <->
        TRANSACTION_TABLE.TRANSACTION.TRANSACTION_DATE: 0.9068642
FINANCIAL_TRANSACTION_TABLE <-> TRANSACTION_TABLE: 0.8026174

Expected results agreed by at least 2 participants
----------------------------------------------------------------------------
1,g     R08A.REF_CARD_TYPE                R08B.CARD_TYPE
2,h     REF_CARD_TYPE.CARD_TYPE_CODE      CARD_TYPE.CARD_TYPE
4,i     REF_CARD_TYPE.DEBIT_AMOUNT        CARD_TYPE.CARDTYPE_LIMIT
5,a     R08A.CUSTOMER                     R08B.APPLICANT
6,b     CUSTOMER.CUSTOMER_ID              APPLICANT.APPLICANT_ID
7,c     CUSTOMER.CUSTOMER_NAME            APPLICANT.APPLICANT_NAME
8,e     CUSTOMER.CUSTOMER_PHONE           APPLICANT.APPLICANT_PHONE
10,f    CUSTOMER.CUSTOMER_ADDRESS         APPLICANT.APPLICANT_ADDRESS
11,j    R08A.CUSTOMER_CARD                R08B.CARD
13,m    CUSTOMER_CARD.CUSTOMER_ID         CARD.APPLICANT_ID
14,k    CUSTOMER_CARD.CARD_NUMBER         CARD.CARD_NUMBER
16,l    CUSTOMER_CARD.DATE_VALID_TO       CARD.CARD_EXPIRE_DATE
17,o    CUSTOMER_CARD.CARD_TYPE_CODE      CARD.CARD_TYPE
22,t    R08A.FINANCIAL_TRANSACTION        R08B.TRANSACTION
23,u    FINANCIAL_TRANSACTION.TRANSACTION_ID
                                          TRANSACTION.TRANSACTION_ID
25,x    FINANCIAL_TRANSACTION.TRANSACTION_AMOUNT
                                          TRANSACTION.TRANSACTION_AMOUNT
27,w    FINANCIAL_TRANSACTION.TRANSACTION_DATE TRANSACTION.TRANSACTION_DATE

Matching Results between PP1 and PP2 using the Relational model:
Columns:
--------------------------------- ----------------------------        ------
DEALER.DELIVERY_FINAL_COST        CAR_SALES.FIN_COST                  0.0512
FISCAL_CAL.FISCAL_MONTH           FINANCIAL_CAL.FIN_MONTH             0.0592
FISCAL_CAL.FISCAL_YEAR            FINANCIAL_CAL.FIN_YEAR              0.0577
FISCAL_CAL.YEAR_MONTH             FINANCIAL_CAL.FIN_YEAR_MONTH        0.0339
MONTHLY_SALES.SALES_AMT           CAR_SALES.SALES_AMOUNT              0.0545
MONTHLY_SALES.SALES_QTY           CAR_DEALER.SALES_RNK                0.0495

Tables:
--------------------------------- ----------------------------        ------
DEALER                            CAR_DEALER                          0.1181
FISCAL_CAL                        FINANCIAL_CAL                       0.1541
MAKE                              CAR_MAKE                            0.0566
MODEL                             CAR_MODEL                           0.0566
MONTHLY_SALES                     CAR_SALES                           0.1761

Matching Results between PP1 and PP2 using the Star model:
Columns:
--------------------------------- ----------------------------        ------
DEALER.DEALER_ID                  CAR_DEALER.DEALER_KEY               0.1165
DEALER.DEALER_NAME                CAR_DEALER.DEALER_NM                0.0534
FISCAL_CAL.FISCAL_MONTH           FINANCIAL_CAL.FIN_MONTH             0.0564
FISCAL_CAL.FISCAL_YEAR            FINANCIAL_CAL.FIN_YEAR              0.0552
FISCAL_CAL.YEAR_MONTH             FINANCIAL_CAL.FIN_YEAR_MONTH        0.0459
```

```
MAKE.MAKE_ID                      CAR_MAKE.CAR_MAKE                    0.0343
MAKE.MAKE_NAME                    CAR_MAKE.CAR_MAKE_DESC               0.0458
MODEL.MODEL_ID                    CAR_MODEL.CAR_MODEL                  0.0343
MODEL.MODEL_NAME                  CAR_MODEL.CAR_MODEL_DESC             0.0458
MONTHLY_SALES.DEALER_ID           CAR_SALES.DEALER_KEY                 0.0221
MONTHLY_SALES.MAKE_ID             CAR_SALES.CAR_MAKE                   0.0177
MONTHLY_SALES.MODEL_ID            CAR_SALES.CAR_MODEL                  0.0177
MONTHLY_SALES.MONTHLY_ADS_COST    CAR_SALES.FIN_COST                   0.0415
MONTHLY_SALES.SALES_AMT           CAR_SALES.SALES_AMOUNT               0.0517
MONTHLY_SALES.SALES_QTY           CAR_SALES.SOLD_QTY                   0.0487
MONTHLY_SALES.YEAR_MONTH          CAR_SALES.FIN_YEAR_MONTH             0.0248

Tables:
--------------------------------- ----------------------------         ------
DEALER                            CAR_DEALER                           0.2725
FISCAL_CAL                        FINANCIAL_CAL                        0.2705
MAKE                              CAR_MAKE                             0.1141
MODEL                             CAR_MODEL                            0.1141
MONTHLY_SALES                     CAR_SALES                            0.9070

COMA Results
----------------------------------------------------------------------------
DEALER_TABLE.DEALER.DEALER_ID <->
        CAR_DEALER_TABLE.CAR_DEALER.DEALER_KEY: 0.6401282
DEALER_TABLE.DEALER.DEALER_NAME <->
        CAR_DEALER_TABLE.CAR_DEALER.DEALER_NM: 0.66036695
DEALER_TABLE.DEALER.DELIVERY_FINAL_COST <->
        CAR_SALES_TABLE.CAR_SALES.FIN_COST: 0.5586653
DEALER_TABLE <-> CAR_DEALER_TABLE: 0.702073
MAKE_TABLE.MAKE.MAKE_ID <-> CAR_MAKE_TABLE.CAR_MAKE.CAR_MAKE: 0.62163365
MAKE_TABLE.MAKE.MAKE_NAME <-> CAR_MAKE_TABLE.CAR_MAKE.CAR_MAKE: 0.6184272
MAKE_TABLE <-> CAR_MAKE_TABLE: 0.7313195
FISCAL_CAL_TABLE.FISCAL_CAL.YEAR_MONTH <->
        FINANCIAL_CAL_TABLE.FINANCIAL_CAL.FIN_YEAR_MONTH: 0.74298567
FISCAL_CAL_TABLE.FISCAL_CAL.FISCAL_YEAR <->
        FINANCIAL_CAL_TABLE.FINANCIAL_CAL.FIN_YEAR: 0.6262011
FISCAL_CAL_TABLE.FISCAL_CAL.FISCAL_MONTH <->
        FINANCIAL_CAL_TABLE.FINANCIAL_CAL.FIN_MONTH: 0.70120114
FISCAL_CAL_TABLE <-> FINANCIAL_CAL_TABLE: 0.7388057
MONTHLY_SALES_TABLE.MONTHLY_SALES.MODEL_ID <->
        CAR_SALES_TABLE.CAR_SALES.CAR_MODEL: 0.5878663
MONTHLY_SALES_TABLE.MONTHLY_SALES.DEALER_ID <->
        CAR_SALES_TABLE.CAR_SALES.DEALER_KEY: 0.60163754
MONTHLY_SALES_TABLE.MONTHLY_SALES.MAKE_ID <->
        CAR_SALES_TABLE.CAR_SALES.CAR_MAKE: 0.58968925
MONTHLY_SALES_TABLE.MONTHLY_SALES.YEAR_MONTH <->
        CAR_SALES_TABLE.CAR_SALES.FIN_YEAR_MONTH: 0.7123402
MONTHLY_SALES_TABLE.MONTHLY_SALES.SALES_AMT <->
        CAR_SALES_TABLE.CAR_SALES.SALES_AMOUNT: 0.7046375
MONTHLY_SALES_TABLE.MONTHLY_SALES.SALES_QTY <->
        CAR_SALES_TABLE.CAR_SALES.SOLD_QTY: 0.62969095
MONTHLY_SALES_TABLE <-> CAR_SALES_TABLE: 0.65521526
MODEL_TABLE.MODEL.MODEL_ID <->
        CAR_MODEL_TABLE.CAR_MODEL.CAR_MODEL: 0.61981076
MODEL_TABLE <-> CAR_MODEL_TABLE: 0.72402775
```

```
Expected results agreed by at least 2 participants
----------------------------------------------------------------------------
1,a     PP1.FISCAL_CAL             PP2.FINANCIAL_CAL
2,b     YEAR_MONTH                 FIN_YEAR_MONTH
3,c     FISCAL_YEAR                FIN_YEAR INTEGER
4,d     FISCAL_MONTH               FIN_MONTH
5,i     PP1.MAKE                   PP2.CAR_MAKE
6,j     MAKE.MAKE_ID               CAR_MAKE.CAR_MAKE
7,k     MAKE.MAKE_NAME             CAR_MAKE.CAR_MAKE_DESC
8,l     PP1.MODEL                  PP2.CAR_MODEL
9,m     MODEL.MODEL_ID             CAR_MODEL.CAR_MODEL
10,n    MODEL.MODEL_NAME           CAR_MODEL.CAR_MODEL_DESC++++
11,e    PP1.DEALER                 PP2.CAR_DEALER
12,f    PP1.DEALER.DEALER_ID       CAR_DEALER.DEALER_KEY
13,g    DEALER_NAME                DEALER_NM
15,o    PP1.MONTHLY_SALES          CAR_SALES
16,p    MONTHLY_SALES.YEAR_MONTH   CAR_SALES.FIN_YEAR_MONTH
17,s    PP1.MONTHLY_SALES.MAKE_ID CAR_SALES.CAR_MAKE
18,r    PP1.MONTHLY_SALES.MODEL_ID CAR_SALES.CAR_MODEL
19,q    PP1.MONTHLY_SALES.DEALER_ID CAR_SALES.DEALER_KEY
20,w    SALES_QTY                  SOLD_QTY
21,u    SALES_AMT                  SALES_AMOUNT
22,v    MONTHLY_ADS_COST           MONTH_AD
213

Matching Results between T01A and T01B using the Relational model:
Columns:
--------------------------------- ----------------------------         ------
AUTO_DEALER.DEALER_ID              DEALER.DEALER_ID 0.0194
AUTO_MAKE.MAKE_ID                  MMSC.MAKE_ID 0.0181
AUTO_MAKE.MAKE_NAME                MMSC.MAKE_NAME 0.0612
AUTO_MODEL.MODEL_ID                MMSC.MODEL_ID 0.0181
AUTO_MODEL.MODEL_NAME              MMSC.MODEL_NAME 0.0612
DATE.CALENDAR_YEAR                 DATE.CALENDAR_YEAR 0.0665
DATE.FISCAL_YEAR                   DATE.FISCAL_YEAR 0.0629
DATE.MONTH_NAME                    DATE.MONTH_NAME 0.0619
MONTHLY_AUTO_SALES.AUTO_SALES_AMOUNT
                                   MONTHLY_AUTO_SALES.AUTO_SALES_AMOUNT 0.0598
MONTHLY_AUTO_SALES.OBJECTIVE_SALES_AMOUNT
                                   MONTHLY_AUTO_SALES.OBJECTIVE_SALES_AMOUNT
                                                                       0.0598
MONTHLY_AUTO_SALES.OBJECTIVE_SALES_QUANTITY
                                   MONTHLY_AUTO_SALES.OBJECTIVE_SALES_QUANTITY
                                                                       0.0593

Tables:
--------------------------------- ----------------------------         ------
AUTO_DEALER                        DEALER                              0.0857
AUTO_MODEL                         MMSC                                0.0578
DATE                               DATE                                0.1527
MONTHLY_AUTO_SALES                 MONTHLY_AUTO_SALES                  0.2451

Matching Results between T01A and T01B using the Star model:
Columns:
--------------------------------- ----------------------------         ------
AUTO_DEALER.DEALER_ID              DEALER.DEALER_ID                    0.0302
AUTO_MAKE.MAKE_ID                  MMSC.MAKE_ID                        0.0263
AUTO_MAKE.MAKE_NAME                MMSC.MAKE_NAME                      0.0618
```

```
AUTO_MODEL.MODEL_ID                MMSC.MODEL_ID                        0.0263
AUTO_MODEL.MODEL_NAME              MMSC.MODEL_NAME                      0.0618
DATE.CALENDAR_YEAR                 DATE.CALENDAR_YEAR                   0.0698
DATE.FISCAL_YEAR                   DATE.FISCAL_YEAR                     0.0665
DATE.MONTH_NAME                    DATE.MONTH_NAME                      0.0658
DATE.MONTH_YEAR                    DATE.MONTH_YEAR                      0.0297
MONTHLY_AUTO_SALES.AUTO_SALES_AMOUNT
                                   MONTHLY_AUTO_SALES.AUTO_SALES_AMOUNT 0.0590
MONTHLY_AUTO_SALES.DEALER_ID       MONTHLY_AUTO_SALES.DEALER_ID         0.0217
MONTHLY_AUTO_SALES.MAKE_ID         MONTHLY_AUTO_SALES.MAKE_ID           0.0227
MONTHLY_AUTO_SALES.MODEL_ID        MONTHLY_AUTO_SALES.MODEL_ID          0.0227
MONTHLY_AUTO_SALES.MONTH_YEAR      MONTHLY_AUTO_SALES.MONTH_YEAR        0.0176
MONTHLY_AUTO_SALES.OBJECTIVE_SALES_AMOUNT
                                   MONTHLY_AUTO_SALES.OBJECTIVE_SALES_AMOUNT
                                                                        0.0590
MONTHLY_AUTO_SALES.OBJECTIVE_SALES_QUANTITY
                                   MONTHLY_AUTO_SALES.OBJECTIVE_SALES_QUANTITY
                                                                        0.0645

Tables:
---------------------------------- ----------------------------         ------
AUTO_DEALER                        DEALER                               0.2075
AUTO_MODEL                         MMSC                                 0.2095
DATE                               DATE                                 0.3413
MONTHLY_AUTO_SALES                 MONTHLY_AUTO_SALES                   1

COMA Results
----------------------------------------------------------------------------
AUTO_MAKE_TABLE.AUTO_MAKE.MAKE_NAME <-> MMSC_TABLE.MMSC.MAKE_NAME: 0.83674204
AUTO_MAKE_TABLE.AUTO_MAKE <-> MMSC_TABLE.MMSC: 0.47530934
AUTO_MODEL_TABLE.AUTO_MODEL.MODEL_NAME <->
        MMSC_TABLE.MMSC.MODEL_NAME: 0.83394575
AUTO_MODEL_TABLE <-> MMSC_TABLE: 0.6048572
DATE_TABLE.DATE.MONTH_YEAR <-> DATE_TABLE.DATE.MONTH_YEAR: 0.9361829
DATE_TABLE.DATE.FISCAL_YEAR <-> DATE_TABLE.DATE.FISCAL_YEAR: 0.9925
DATE_TABLE.DATE.CALENDAR_YEAR <-> DATE_TABLE.DATE.CALENDAR_YEAR: 0.9925
DATE_TABLE.DATE.MONTH_NAME <-> DATE_TABLE.DATE.MONTH_NAME: 0.9925
DATE_TABLE.DATE <-> DATE_TABLE.DATE: 0.985
MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.DEALER_ID <->
        MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.DEALER_ID: 0.84580815
MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.MAKE_ID <->
        MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.MAKE_ID: 0.93859327
MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.MODEL_ID <->
        MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.MODEL_ID: 0.93845487
MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.MONTH_YEAR <->
        MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.MONTH_YEAR: 0.9361829
MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.AUTO_SALES_AMOUNT <->
        MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.AUTO_SALES_AMOUNT:
        0.90986574
MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.OBJECTIVE_SALES_AMOUNT <->
        MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.OBJECTIVE_SALES_AMOUNT:
        0.90986574
MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.OBJECTIVE_SALES_QUANTITY <->
        MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES.OBJECTIVE_SALES_QUANTITY:
        0.8648658
MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES <->
        MONTHLY_AUTO_SALES_TABLE.MONTHLY_AUTO_SALES: 0.8797314
AUTO_DEALER_TABLE.AUTO_DEALER.DEALER_ID <->
```

```
        DEALER_TABLE.DEALER.DEALER_ID: 0.803889
AUTO_DEALER_TABLE.AUTO_DEALER.DEALER_INFORMATION <->
        DEALER_TABLE.DEALER.DEALER_NAME: 0.6346453
AUTO_DEALER_TABLE.AUTO_DEALER <->
        DEALER_TABLE.DEALER: 0.65350115

Expected results agreed by at least 2 participants
-----------------------------------------------------------------------------
1,f     T01A.AUTO_DEALER                  T01B.DEALER
2,g     AUTO_DEALER.DEALER_ID             DEALER.DEALER_ID
3,h     AUTO_DEALER.DEALER_INFORMATION    DEALER.DEALER_NAME
3,i     AUTO_DEALER.DEALER_INFORMATION    DEALER.DEALER_CITY
3,j     AUTO_DEALER.DEALER_INFORMATION    DEALER_STATE
4,m     T01A.AUTO_MAKE                    T01B.MMSC
5,n     AUTO_MAKE.MAKE_ID                 MMSC.MAKE_ID
6,r     AUTO_MAKE.MAKE_NAME               MMSC.MAKE_NAME
7,m     T01A.AUTO_MODEL                   T01B.MMSC
8,o     AUTO_MODEL.MODEL_ID               MMSC.MODEL_ID
9,s     AUTO_MODEL.MODEL_NAME             MMSC.MODEL_NAME
11,a    CREATE T01A.DATE                  T01B.DATE
12,b    DATE.MONTH_YEAR                   DATE.MONTH_YEAR
13,c    DATE.FISCAL_YEAR                  DATE.FISCAL_YEAR
14,d    DATE.CALENDAR_YEAR                DATE.CALENDAR_YEAR
15,e    DATE.MONTH_NAME                   DATE.MONTH_NAME
16,w    T01A.MONTHLY_AUTO_SALES           T01B.MONTHLY_AUTO_SALES
17,bb   MONTHLY_AUTO_SALES.MONTH_YEAR     MONTHLY_AUTO_SALES.MONTH_YEAR
18,cc   MONTHLY_AUTO_SALES.DEALER_ID      MONTHLY_AUTO_SALES.DEALER_ID
19,x    MONTHLY_AUTO_SALES.MAKE_ID        MONTHLY_AUTO_SALES.MAKE_ID
20,y    MONTHLY_AUTO_SALES.MODEL_ID       MONTHLY_AUTO_SALES.MODEL_ID
22,dd   MONTHLY_AUTO_SALES.AUTO_SALES_AMOUNT DATE.AUTO_SALES_AMOUNT
23,gg   MONTHLY_AUTO_SALES.OBJECTIVE_SALES_AMOUNT
        MONTHLY_AUTO_SALES.OBJECTIVE_SALES_AMOUNT
24,ee   MONTHLY_AUTO_SALES.OBJECTIVE_SALES_QUANTITY
        MONTHLY_AUTO_SALES.OBJECTIVE_SALES_QUANTITY

Matching Results between T07A and T09A using the Relational model:
Columns:
--------------------------------- ----------------------------         ------
CUSTOMER.EDUCATION                ON_SALE.DESCRIPTION                    0.0101
PRODUCT.BRAND_NAME                ON_SALE.BRAND                          0.0699
PRODUCT.SKU                       ON_SALE.SKU_NUMBER                     0.0677
PROMOTION.COST                    PRODUCT_SALES.UNIT_COST                0.0685
STORE.STORE_TYPE                  ON_SALE.PACKAGE_TYPE                   0.0569

Tables:
--------------------------------- ----------------------------         ------
SALES                             PRODUCT_SALES                          0.1753

Matching Results between T07A and T09A using the Star model:
Columns:
--------------------------------- ----------------------------         ------
PRODUCT.BRAND_NAME                ON_SALE.BRAND                          0.0507
PRODUCT.PRODUCT_ID                ON_SALE.PRODUCT_KEY                    0.0401
PRODUCT.SKU                       ON_SALE.SKU_NUMBER                     0.0494
PROMOTION.MEDIA_TYPE              ON_SALE.PACKAGE_TYPE                   0.0404
PROMOTION.PROMOTION_NAME          ON_SALE.DESCRIPTION                    0.0080
SALES.PRODUCT_ID                  PRODUCT_SALES.PRODUCT_KEY              0.0358
```

```
Tables:
---------------------------------  ----------------------------        ------
PRODUCT                            ON_SALE                               0.4005
SALES                              PRODUCT_SALES                         0.9202

COMA Results
----------------------------------------------------------------------------
SALES_TABLE.SALES.STORE_COST <->
        PRODUCT_SALES_TABLE.PRODUCT_SALES.UNIT_COST: 0.5818722
SALES_TABLE.SALES.PRODUCT_ID <->
        PRODUCT_SALES_TABLE.PRODUCT_SALES.PRODUCT_KEY: 0.55921656
SALES_TABLE <-> PRODUCT_SALES_TABLE: 0.61783636
PRODUCT_TABLE.PRODUCT.PRODUCT_ID <->
        PRODUCT_SALES_TABLE.PRODUCT_SALES.PRODUCT_KEY: 0.5640777
PRODUCT_TABLE.PRODUCT.BRAND_NAME <->
        PRODUCT_ON_SALE_TABLE.PRODUCT_ON_SALE.BRAND: 0.6251667
PRODUCT_TABLE.PRODUCT.SKU <->
        PRODUCT_ON_SALE_TABLE.PRODUCT_ON_SALE.SKU_NUMBER: 0.5951667
PRODUCT_TABLE.PRODUCT.RECYCLANLE_PACKAGE <->
        PRODUCT_ON_SALE_TABLE.PRODUCT_ON_SALE.PACKAGE_TYPE: 0.57150924

Expected results agreed by at least 2 participants
----------------------------------------------------------------------------
28,a    T07A.PRODUCT               T09A.ON_SALE
29,b    PRODUCT.PRODUCT_ID         ON_SALE.PRODUCT_KEY
31,e    PRODUCT.BRAND_NAME         ON_SALE.BRAND
32,d    PRODUCT.PRODUCT_NAME       ON_SALE.DESCRIPTION
33,c    PRODUCT.SKU                ON_SALE.SKU_NUMBER
89,h    T07A.SALES                 T09A.PRODUCT_SALES
216

Matching Results between A01 and A02 using the Relational model:
Columns:
---------------------------------  ----------------------------        ------
CALENDAR.CALENDAR_DAY              DATES.DAY                             0.0365
CALENDAR.CALENDAR_MONTH            DATES.MONTH                           0.0365
CALENDAR.CALENDAR_YEAR             DATES.YEAR                            0.0387
CALENDAR.QUARTER_NUMBER            DATES.QUARTER                         0.0382
CLAIM.CLAIM_NO                     CLM_DEMAND.NO_CLAIMS                  0.0387
CLAIM.DEMANDED_AMT                 CLAIM_HIST.DEMANDED_DTE               0.0275
CLAIM.INSERT_TIMESTAMP             CLM_DEMAND.LOAD_TIMESTAMP             0.0424
CLAIM.RECEIVED_AMT                 CLAIM_HIST.RECEIVED_DTE               0.0275
CLAIM.RECOVERED_AMT                CLAIM_HIST.RECOVERED_DTE              0.0275
CLAIM.SETTLEMENT_AMT               CLAIM_HIST.SETTLEMENT_DTE             0.0275
CLAIM_STAT.STATUS_CODE             CLAIM_ITEM_STATUS.CLAIM_ITEM_STATUS_CODE
                                                                         0.0380
CLAIM_STAT.STATUS_DATE_FROM        CLAIM_ITEM_STATUS.DATE_VALID_FROM     0.0377
CLAIM_STAT.STATUS_DATE_TO          CLAIM_ITEM_STATUS.DATE_VALID_TO       0.0377
LOCATION.STATE_NAME                LOCALITY.STATE_NAME                   0.0584
LOCATION.STATE_NO                  LOCALITY.STATE_NBR                    0.0282
PRODUCT.PRODUCT_NAME               PRODUCT_REFERENCE.PRODUCT_DESCRIPTIVE_NAME
                                                                         0.0462

Tables:
---------------------------------  ----------------------------        ------
CALENDAR                           DATES                                 0.0670
CLAIM                              CLM_DEMAND                            0.0960
CLAIM_STAT                         CLAIM_ITEM_STATUS                     0.1125
```

```
LOCATION                          LOCALITY                               0.0546
PRODUCT                           PRODUCT_REFERENCE                      0.1165

Matching Results between A01 and A02 using the Star model:
Columns:
--------------------------------- ----------------------------         ------
CALENDAR.CALENDAR_DAY             DATES.DAY                              0.0397
CALENDAR.CALENDAR_MONTH           DATES.MONTH                            0.0397
CALENDAR.CALENDAR_YEAR            DATES.YEAR                             0.0419
CALENDAR.QUARTER_NUMBER           DATES.QUARTER                          0.0414
CLAIM.DEMANDED_AMT                CLAIM_HIST.DEMANDED_DTE                0.0278
CLAIM.INSERT_TIMESTAMP            CLM_DEMAND.LOAD_TIMESTAMP              0.0156
CLAIM.PRODUCT_KEY                 CLM_DEMAND.PRODUCT_HIERARCHY_KEY       0.0158
CLAIM.RECEIVED_AMT                CLM_DEMAND.DEMAND_RECEIVED_AMOUNT      0.0348
CLAIM.RECOVERED_AMT               CLM_DEMAND.DEMAND_RECOVERED_AMOUNT     0.0348
CLAIM.SETTLEMENT_AMT              CLAIM_HIST.SETTLEMENT_DTE              0.0278
CLAIM_STAT.STATUS_CODE            CLAIM_ITEM_STATUS.CLAIM_ITEM_STATUS_CODE
                                                                         0.0440
CLAIM_STAT.STATUS_DATE_FROM       CLAIM_ITEM_STATUS.DATE_VALID_FROM 0.0435
CLAIM_STAT.STATUS_DATE_TO         CLAIM_ITEM_STATUS.DATE_VALID_TO        0.0435
CLAIM_STAT.STATUS_DESCRIPTTION    CLAIM_ITEM_STATUS.CLAIM_ITEM_STATUS_DESC
                                                                         0.0295
CLAIM_STAT.STATUS_KEY             CLAIM_ITEM_STATUS.CLAIM_ITEM_STATUS_KEY
                                                                         0.0246
LOCATION.LOCATION_KEY             LOCALITY.LOCALITY_KEY                  0.0214
LOCATION.STATE_NAME               LOCALITY.STATE_NAME                    0.0656
LOCATION.STATE_NO                 LOCALITY.STATE_NBR                     0.0342
PRODUCT.PRODUCT_CAT               PRODUCT_REFERENCE.PRODUCT_SUBCLASS     0.0403
PRODUCT.PRODUCT_KEY               PRODUCT_REFERENCE.PRODUCT_HIERARCHY_KEY
                                                                         0.0302
PRODUCT.PRODUCT_NAME              PRODUCT_REFERENCE.PRODUCT_DESCRIPTIVE_NAME
                                                                         0.0553

Tables:
--------------------------------- ----------------------------         ------
CALENDAR                          DATES                                  0.2099
CLAIM                             CLM_DEMAND                             0.7669
CLAIM_STAT                        CLAIM_ITEM_STATUS                      0.3055
LOCATION                          LOCALITY                               0.1574
PRODUCT                           PRODUCT_REFERENCE                      0.3123


COMA Results for matching A01 with A02
-----------------------------------------------------------------------------
PRODUCT_TABLE.PRODUCT.PRODUCT_KEY <->
        PRODUCT_REFERENCE_TABLE.PRODUCT_REFERENCE.PRODUCT_HIERARCHY_KEY:
        0.8030105
PRODUCT_TABLE.PRODUCT.PRODUCT_CAT <->
        PRODUCT_REFERENCE_TABLE.PRODUCT_REFERENCE.PRODUCT_CLASS: 0.69785964
PRODUCT_TABLE.PRODUCT.PRODUCT_SUB_CAT <->
        PRODUCT_REFERENCE_TABLE.PRODUCT_REFERENCE.PRODUCT_SUBCLASS: 0.7056376
PRODUCT_TABLE.PRODUCT.PRODUCT_NAME <->
        PRODUCT_REFERENCE_TABLE.PRODUCT_REFERENCE.PRODUCT_DESCRIPTIVE_NAME:
       0.8152281
PRODUCT_TABLE.PRODUCT <->
        PRODUCT_REFERENCE_TABLE.PRODUCT_REFERENCE: 0.75034904
DEMAND_TYPE_TABLE.DEMAND_TYPE.DEMAND_ID <->
        CLM_DEMAND_TABLE.CLM_DEMAND.DEMAND_SENT_DATE: 0.52169096
```

```
LOCATION_TABLE.LOCATION.LOCATION_KEY <->
        LOCALITY_TABLE.LOCALITY.LOCALITY_KEY: 0.71002436
LOCATION_TABLE.LOCATION.STATE_NO <->
        LOCALITY_TABLE.LOCALITY.STATE_NBR: 0.68908656
LOCATION_TABLE.LOCATION.STATE_NAME <->
        LOCALITY_TABLE.LOCALITY.STATE_NAME: 0.87971157
LOCATION_TABLE.LOCATION <-> LOCALITY_TABLE.LOCALITY: 0.6469231
CALENDAR_TABLE.CALENDAR.CALENDAR_KEY <->
        DATES_TABLE.DATES.DATE_KEY: 0.52051216
CALENDAR_TABLE.CALENDAR.CALENDAR_YEAR <->
        DATES_TABLE.DATES.YEAR: 0.69585377
CALENDAR_TABLE.CALENDAR.CALENDAR_MONTH <->
        DATES_TABLE.DATES.MONTH: 0.643076
CALENDAR_TABLE.CALENDAR.CALENDAR_DAY <->
        DATES_TABLE.DATES.DAY: 0.6523352
CALENDAR_TABLE.CALENDAR.QUARTER_NUMBER <->
DATES_TABLE.DATES.QUARTER: 0.68733126
CALENDAR_TABLE.CALENDAR <-> DATES_TABLE.DATES: 0.4375409
CLAIM_TABLE.CLAIM.PRODUCT_KEY <->
        CLM_DEMAND_TABLE.CLM_DEMAND.PRODUCT_HIERARCHY_KEY: 0.7567142
CLAIM_TABLE.CLAIM.LOCATION_KEY <->
        CLM_DEMAND_TABLE.CLM_DEMAND.LOCALITY_KEY: 0.69613546
CLAIM_TABLE.CLAIM.CALENDAR_KEY <->
        DATES_TABLE.DATES.DATE_KEY: 0.52051216
CLAIM_TABLE.CLAIM.DEMAND_ID <->
        CLAIM_HIST_TABLE.CLAIM_HIST.DEMANDED_DTE: 0.50176376
CLAIM_TABLE.CLAIM.CLAIM_NO <->
        CLM_DEMAND_TABLE.CLM_DEMAND.NO_CLAIMS: 0.6763557
CLAIM_TABLE.CLAIM.DEMANDED_AMT <->
        CLM_DEMAND_TABLE.CLM_DEMAND.DEMAND_SETTLED_AMOUNT: 0.5937898
CLAIM_TABLE.CLAIM.RECEIVED_AMT <->
        CLM_DEMAND_TABLE.CLM_DEMAND.DEMAND_RECEIVED_AMOUNT: 0.6322553
CLAIM_TABLE.CLAIM.RECOVERED_AMT <->
        CLM_DEMAND_TABLE.CLM_DEMAND.DEMAND_RECOVERED_AMOUNT: 0.63159084
CLAIM_TABLE.CLAIM.INSERT_TIMESTAMP <->
        CLM_DEMAND_TABLE.CLM_DEMAND.LOAD_TIMESTAMP: 0.5906589
CLAIM_TABLE.CLAIM <-> CLAIM_HIST_TABLE.CLAIM_HIST: 0.5492281
CLAIM_TABLE <-> CLM_DEMAND_TABLE: 0.6406745
CLAIM_STAT_TABLE.CLAIM_STAT.STATUS_KEY <->
        CLAIM_ITEM_STATUS_TABLE.CLAIM_ITEM_STATUS.CLAIM_ITEM_STATUS_KEY:
        0.7132797
CLAIM_STAT_TABLE.CLAIM_STAT.STATUS_CODE <->
CLAIM_ITEM_STATUS_TABLE.CLAIM_ITEM_STATUS.CLAIM_ITEM_STATUS_CODE: 0.7348547
CLAIM_STAT_TABLE.CLAIM_STAT.STATUS_DESCRIPTTION <->
        CLAIM_ITEM_STATUS_TABLE.CLAIM_ITEM_STATUS.CLAIM_ITEM_STATUS_DESC:
        0.62528205
CLAIM_STAT_TABLE.CLAIM_STAT.STATUS_DATE_FROM <->
        CLAIM_ITEM_STATUS_TABLE.CLAIM_ITEM_STATUS.DATE_VALID_FROM: 0.7348547
CLAIM_STAT_TABLE.CLAIM_STAT.STATUS_DATE_TO <->
        CLAIM_ITEM_STATUS_TABLE.CLAIM_ITEM_STATUS.DATE_VALID_TO: 0.7348547
CLAIM_STAT_TABLE.CLAIM_STAT <->
        CLAIM_ITEM_STATUS_TABLE.CLAIM_ITEM_STATUS: 0.72483766

Expected results agreed by at least 2 participants
-----------------------------------------------------------------------------
4,l     A01.CLAIM_STAT            A02.CLAIM_ITEM_STATUS
5,m     CLAIM_STAT.STATUS_KEY     CLAIM_ITEM_STATUS.CLAIM_ITEM_STATUS_KEY
6,n     CLAIM_STAT.STATUS_CODE    CLAIM_ITEM_STATUS.CLAIM_ITEM_STATUS_CODE
```

```
7,o      CLAIM_STAT.STATUS_DESCRIPTTION
                                   CLAIM_ITEM_STATUS.CLAIM_ITEM_STATUS_DESC
8,p      CLAIM_STAT.STATUS_DATE_FROM
                                   CLAIM_ITEM_STATUS.DATE_VALID_FROM
9,q      CLAIM_STAT.STATUS_DATE_TO CLAIM_ITEM_STATUS.DATE_VALID_TO
10,g     A01.PRODUCT               A02.PRODUCT_REFERENCE
11,h     PRODUCT.PRODUCT_KEY       PRODUCT_REFERENCE.PRODUCT_HIERARCHY_KEY
2,i      DEMAND_TYPEDEMAND_ID      PRODUCT_REFERENCE.PRODUCT_CLASS
13,j     PRODUCT.PRODUCT_CAT       PRODUCT_REFERENCE.PRODUCT_SUBCLASS
14,k     PRODUCT.PRODUCT_NAME      PRODUCT_REFERENCE.PRODUCT_DESCRIPTIVE_NAME
15,r     A01.LOCATION              A02.LOCALITY
16,s     LOCATION.LOCATION_KEY     LOCALITY.LOCALITY_KEY
17,t     LOCATION.STATE_NO         LOCALITY.STATE_NBR
18,u     LOCATION.STATE_NAME       LOCALITY.STATE_NAME
19,w     A01.CALENDAR              A02.DATES
20,x     CALENDAR.CALENDAR_KEY     DATES.DATE_KEY
21,y     CALENDAR.CALENDAR_YEAR    DATES.YEAR
22,z     CALENDAR.CALENDAR_MONTH   DATES.MONTH
23,aa    CALENDAR.CALENDAR_DAY     DATES.DAY
24,bb    CALENDAR.WEEK_OF_MONTH    DATES.MONTH_COUNT
25,cc    CALENDAR.QUARTER_NUMBER   DATES.QUARTER
26,dd    A01.CLAIM                 A02.CLM_DEMAND
27,ee    CLAIM.PRODUCT_KEY         CLM_DEMAND.PRODUCT_HIERARCHY_KEY
28,ff    CLAIM.LOCATION_KEY        CLM_DEMAND.LOCALITY_KEY
29,gg    CLAIM.CALENDAR_KEY        CLM_DEMAND.DEMAND_SENT_DATE
30,hh    CLAIM.CLAIM_NO            CLM_DEMAND.CLAIM_NUM
32,jj    CLAIM.STATUS_KEY          CLM_DEMAND.CLAIM_ITEM_STATUS_KEY
33,kk    CLAIM.SETTLEMENT_AMT      CLM_DEMAND.DEMAND_SETTLED_AMOUNT
35,ll    CLAIM.RECEIVED_AMT        CLM_DEMAND.DEMAND_RECEIVED_AMOUNT
36,mm    CLAIM.RECOVERED_AMT       CLM_DEMAND.DEMAND_RECOVERED_AMOUNT
37,oo    CLAIM.INSERT_TIMESTAMP    CLM_DEMAND.LOAD_TIMESTAMP


Matching Results between T02A and T02B using the Relational model:
Columns:
----------------------------------------------------------------------------
ORGANIZATION.REGION_NAME           REGION.REGION_DESC
TIME.DAY_OF_YEAR                   PERIOD.YEAR
TIME.QUARTER_NUMBER                PERIOD.QUARTER
TIME.TIME_IDENTIFIER               PERIOD.PERIOD_ID

Tables:
----------------------------------------------------------------------------
ORGANIZATION                       REGION
PRODUCT                            PRODUCT
SALES                              SALES_CURRENT
TIME                               PERIOD

Matching Results between T02A and T02B using the Star model:
Columns:
----------------------------------------------------------------------------
ORGANIZATION.REGION_NAME           REGION.REGION_DESC
PRODUCT.PRODUCT_IDENTIFIER         PRODUCT.PRODUCT_ID
SALES.PRODUCT_IDENTIFIER           SALES_CURRENT.PRODUCT_ID
SALES.SALES_DOLLAR                 SALES_CURRENT.DOLLARS
TIME.DAY_OF_YEAR                   PERIOD.YEAR
TIME.QUARTER_NUMBER                PERIOD.QUARTER
TIME.TIME_IDENTIFIER               PERIOD.PERIOD_ID
```

```
Tables:
---------------------------------------------------------------------------
ORGANIZATION                      REGION
PRODUCT                           PRODUCT
SALES                             SALES_CURRENT
TIME                              PERIOD

COMA Results
---------------------------------------------------------------------------
ORGANIZATION_TABLE.ORGANIZATION.REGION_NAME <->
DISTRICT_TABLE.DISTRICT.REGION: 0.608506
ORGANIZATION_TABLE <-> REGION_TABLE: 0.5459982
TIME_TABLE.TIME.DAY_OF_YEAR <-> PERIOD_TABLE.PERIOD.YEAR: 0.52207315
TIME_TABLE.TIME.QUARTER_NUMBER <-> PERIOD_TABLE.PERIOD.QUARTER: 0.62035346
TIME_TABLE <-> PERIOD_TABLE: 0.50840425
PRODUCT_TABLE.PRODUCT.PRODUCT_IDENTIFIER <->
        PRODUCT_TABLE.PRODUCT.PRODUCT_ID: 0.65886056
PRODUCT_TABLE <-> PRODUCT_TABLE: 0.7549077
SALES_TABLE.SALES.PRODUCT_IDENTIFIER <->
SALES_CURRENT_TABLE.SALES_CURRENT.PRODUCT_ID: 0.61372167
SALES_TABLE.SALES.SALES_DOLLAR <->
        SALES_CURRENT_TABLE.SALES_CURRENT.DOLLARS: 0.5786033
SALES_TABLE <-> SALES_CURRENT_TABLE: 0.64747894

Expected results agreed by at least 2 participants
---------------------------------------------------------------------------
1,a     T02A.Location             T02B.District
1,o     T02A.Location             T02B.Region
3,q     Location.Country_Name     Region.Region_Desc
5,c     Location.County_Name      Region.District_Desc
6,c     Location.City_Name        Region.District_Desc
15,j    T02A.Product              T02B.Product
16,k    Product.Product_Identifier Product.Product_Id
21,r    T02A.Sales                T02B.Sales_Current
22,s    Sales.Time_Identifier     Sales_Current.Period_Id
24,t    Sales.Product_Identifier  Sales_Current.Product_Id
25,u    Sales.Location_Identifier Sales_Current.District_Id
26,x    Sales.Sales_Dollar        Sales_Current.Dollars
28,f    Sales.Time_Identifier     Sales_Current.Period_Id


Matching Results between T10 and T11 using the Relational model:
Columns:
--------------------------------- ----------------------------        ------
CALENDAR.CALENDAR_WEEK            DATE.WEEK_NUMBER                    0.0316
CALENDAR.CALENDAR_YEAR            DATE.YEAR_NUMBER                    0.0335
CALENDAR.DAY                      DATE.DAY_DATE                       0.0482
CUSTOMER.CUSTOMER_ADDRESS         CUSTOMER.CUSTOMER_ADDRESS           0.0666
ITEM.PRODUCT_DESC                 PRODUCT.PRODUCT_DESC                0.0626

Tables:
--------------------------------- ----------------------------        ------
CALENDAR                          DATE                                0.1140
CUSTOMER                          CUSTOMER                            0.2033
ITEM                              PRODUCT                             0.0424
SALES                             PRODUCT_SALES                       0.1481
```

```
Matching Results between T10 and T11 using the Star model:
Columns:
--------------------------------- ----------------------------         ------
CALENDAR.CALENDAR_WEEK            DATE.WEEK_NUMBER                       0.0322
CALENDAR.CALENDAR_YEAR            DATE.YEAR_NUMBER                       0.0339
CALENDAR.DAY                      DATE.DAY_DATE                          0.0477
CUSTOMER.CUSTOMER_ADDRESS         CUSTOMER.CUSTOMER_ADDRESS              0.0672
CUSTOMER.CUSTOMER_KEY             CUSTOMER.CUSTOMER_ID                   0.0270
ITEM.PRODUCT_DESC                 PRODUCT.PRODUCT_DESC                   0.0612
ITEM.PRODUCT_KEY                  PRODUCT.PRODUCT_KEY                    0.0219
SALES.PRODUCT_KEY                 PRODUCT_SALES.PRODUCT_KEY              0.0247

Tables:
--------------------------------- ----------------------------         ------
CALENDAR                          DATE                                   0.3269
CUSTOMER                          CUSTOMER                               0.4820
ITEM                              PRODUCT                                0.1461
MARKET                            GEOGRAPHIC_LOCATION                    0.1040
SALES                             PRODUCT_SALES                          0.8023

COMA Results
---------------------------------=
MARKET_TABLE.MARKET.REGION_NAME <->
        GEOGRAPHIC_LOCATION_TABLE.GEOGRAPHIC_LOCATION.SALES_LOCATION_NAME:
        0.506793
ITEM_TABLE.ITEM.PRODUCT_KEY <-> PRODUCT_TABLE.PRODUCT.PRODUCT_KEY: 0.8731693
ITEM_TABLE.ITEM.PRODUCT_DESC <-> PRODUCT_TABLE.PRODUCT.PRODUCT_DESC: 0.8715706
ITEM_TABLE <-> PRODUCT_TABLE: 0.7431412
CUSTOMER_TABLE.CUSTOMER.CUSTOMER_KEY <->
        CUSTOMER_TABLE.CUSTOMER.CUSTOMER_ID: 0.689523
CUSTOMER_TABLE.CUSTOMER.CUSTOMER_NAME <->
CUSTOMER_TABLE.CUSTOMER.CUSTOMER_FNAME: 0.81839025
CUSTOMER_TABLE.CUSTOMER.CUSTOMER_NAME <->
CUSTOMER_TABLE.CUSTOMER.CUSTOMER_LNAME: 0.81839025
CUSTOMER_TABLE.CUSTOMER.CUSTOMER_ADDRESS <->
CUSTOMER_TABLE.CUSTOMER.CUSTOMER_ADDRESS: 0.9161467
CUSTOMER_TABLE <-> CUSTOMER_TABLE: 0.8322934
CALENDAR_TABLE.CALENDAR.DAY <-> DATE_TABLE.DATE.DAY_DATE: 0.6186527
CALENDAR_TABLE.CALENDAR.CALENDAR_WEEK <->
        DATE_TABLE.DATE.WEEK_NUMBER: 0.5677635
CALENDAR_TABLE.CALENDAR.CALENDAR_MONTH <->
        DATE_TABLE.DATE.MONTH_NUMBER: 0.5677635
CALENDAR_TABLE.CALENDAR.CALENDAR_YEAR <->
        DATE_TABLE.DATE.YEAR_NUMBER: 0.5948496
CALENDAR_TABLE <-> DATE_TABLE: 0.5854795
SALES_TABLE.SALES.PRODUCT_KEY <->
PRODUCT_SALES_TABLE.PRODUCT_SALES.PRODUCT_KEY: 0.85580826
SALES_TABLE.SALES.CUSTOMER_KEY <->
        PRODUCT_SALES_TABLE.PRODUCT_SALES.CUSTOMER_ID: 0.6345357
SALES_TABLE.SALES.UNIT_SALES <->
        PRODUCT_SALES_TABLE.PRODUCT_SALES.SALES_AMOUNT: 0.62851804
SALES_TABLE.SALES.UNIT_SALES <->
        PRODUCT_SALES_TABLE.PRODUCT_SALES.SALES_COST: 0.63407356
SALES_TABLE <-> PRODUCT_SALES_TABLE: 0.712036
```

```
Expected results agreed by at least 2 participants
-----------------------------------------------------------------------------
1,k     T10.CUSTOMER                T11.CUSTOMER
3,m     CUSTOMER.CUSTOMER_NAME      CUSTOMER.CUSTOMER_FNAME
3,n     CUSTOMER.CUSTOMER_NAME      CUSTOMER.CUSTOMER_LNAME
4,o     CUSTOMER.CUSTOMER_ADDRESS   CUSTOMER.CUSTOMER_ADDRESS
8,a     T10.MARKET                  T11.GEOGRAPHIC_LOCATION
9,b     MARKET.STORE_KEY            GEOGRAPHIC_LOCATION.SALES_LOCATION_ID
10,c    MARKET.STORE_NAME           GEOGRAPHIC_LOCATION.SALES_LOCATION_NAME
14,g    T10.ITEM                    T11.PRODUCT
15,h    ITEM.PRODUCT_KEY            PRODUCT.PRODUCT_CODE
16,i    ITEM.PRODUCT_DESC           PRODUCT.PRODUCT_NAME
27,s    T10.CALENDAR                T11.DATE
28,t    CALENDAR.DATE_KEY           DATE.DAY_NUMBER
32,v    CALENDAR.CALENDAR_WEEK      DATE.WEEK_NUMBER
33,w    CALENDAR.CALENDAR_MONTH     DATE.MONTH_NUMBER
35,x    CALENDAR.CALENDAR_YEAR      DATE.YEAR_NUMBER
40,y    T10.SALES                   T11.PRODUCT_SALES
41,bb   SALES.CUSTOMER_KEY          PRODUCT_SALES.CUSTOMER_ID
43,aa   SALES.PRODUCT_KEY           PRODUCT_SALES.PRODUCT_ID
46,dd   SALES.DOLLAR_SALES          PRODUCT_SALES.SALES_AMOUNT
47,ee   SALES.UNIT_SALES            PRODUCT_SALES.SALES_VOLUME
48,ff   SALES.RETAIL_SALES_PRICE    PRODUCT_SALES.SALES_COST


Matching Results between T05A and T05B using the Relational model:
Columns:
--------------------------------- ----------------------------         ------
CLAIMANT.ADDRESS                  CLIENT.CLIENT_ADDRESS                  0.0252
CLAIMANT.NAME                     CLIENT.CLIENT_NAME                     0.0251
CLAIMANT.STATE                    CLIENT.CLIENT_STATE                    0.0242
CLAIMS.CLAIM_DOLLARS              CLAIM_FINANCIAL.CLAIM_NO               0.0344
CLAIM_DESC.PROC_CODE              CLAIM_NATURE.NATURE_CODE               0.0195
CLAIM_TYPE.TYPE_DESC
        POLICY_AGREEMENT_TYPE.POLICY_AGREEMENT_TYPE_DESC                 0.0487
PERIOD.YEAR                       DATES.CALENDAR_YEAR                    0.0490
POLICY_HOLDER.NAME                POLICY_HADER.COMPANY_NAME              0.0247
POLICY_SALES.PREMIUM_DOLLARS      CLAIM_FINANCIAL.BASIC_PREMIUM_AMT      0.0340

Tables:
--------------------------------- ----------------------------         ------
CLAIMANT                          CLIENT                                 0.0328
CLAIMS                            CLAIM_FINANCIAL                        0.0601
PERIOD                            DATES                                  0.0663
POLICY                            POLICY_AGREEMENT_TYPE                  0.0818

Matching Results between T05A and T05B using the Star model:
Columns:
--------------------------------- ----------------------------         ------
CLAIMANT.ADDRESS                  CLIENT.CLIENT_ADDRESS                  0.0238
CLAIMANT.CLAIMANT_KEY             CLIENT.CLIENT_KEY                      0.0148
CLAIMANT.NAME                     CLIENT.CLIENT_NAME                     0.0238
CLAIMANT.STATE                    CLIENT.CLIENT_STATE                    0.0230
CLAIM_DESC.PROC_CODE              CLAIM_NATURE.NATURE_CODE               0.0145
CLAIM_TYPE.TYPE_DESC
        POLICY_AGREEMENT_TYPE.POLICY_AGREEMENT_TYPE_DESC                 0.0510
PERIOD.PER_KEY                    DATES.DATE_KEY                         0.0228
PERIOD.YEAR                       DATES.CALENDAR_YEAR                    0.0463
```

```
POLICY.POLICY_TYPE
        POLICY_AGREEMENT_TYPE.POLICY_AGREEMENT_TYPE_CODE              0.0475
POLICY_HOLDER.NAME                 POLICY_HADER.COMPANY_NAME          0.0242
POLICY_SALES.PREMIUM_DOLLARS       CLAIM_FINANCIAL.BASIC_PREMIUM_AMT  0.0368

Tables:
--------------------------------- ----------------------------      ------
CLAIMANT                           CLIENT                             0.0852
CLAIMS                             CLAIM_FINANCIAL                    0.3778
CLAIM_DESC                         CLAIM_NATURE                       0.0800
PERIOD                             DATES                              0.1660
POLICY                             POLICY_AGREEMENT_TYPE              0.1694

COMA Results
-----------------------------------------------------------------------------
CLAIMS_TABLE.CLAIMS.CLAIM_DOLLARS <->
        CLAIM_FINANCIAL_TABLE.CLAIM_FINANCIAL.CLAIM_NO: 0.5309347
CLAIMS_TABLE.CLAIMS.POLICY_HOLDER_KEY <->
        CLAIM_FINANCIAL_TABLE.CLAIM_FINANCIAL.POLICY_NO: 0.57706654
CLAIMS_TABLE.CLAIMS.CLAIMANT_KEY <->
        CLAIM_FINANCIAL_TABLE.CLAIM_FINANCIAL.CLIENT_KEY: 0.685828
CLAIMS_TABLE <-> CLAIM_FINANCIAL_TABLE: 0.6070546
POLICY_HOLDER_TABLE.POLICY_HOLDER.NAME <->
        POLICY_HADER_TABLE.POLICY_HADER.COMPANY_NAME: 0.68988
POLICY_HOLDER_TABLE <-> POLICY_HADER_TABLE: 0.7160417
CLAIM_TYPE_TABLE.CLAIM_TYPE.TYPE_KEY <->
        CLAIM_NATURE_TABLE.CLAIM_NATURE.NATURE_KEY: 0.62102175
CLAIM_TYPE_TABLE.CLAIM_TYPE <-> CLAIM_NATURE_TABLE.CLAIM_NATURE: 0.62718093
CLAIM_TYPE_TABLE <-> CLAIM_NATURE_TABLE: 0.6910546
CLAIMANT_TABLE.CLAIMANT.CLAIMANT_KEY <->
        CLIENT_TABLE.CLIENT.CLIENT_KEY: 0.6904576
CLAIMANT_TABLE.CLAIMANT.NAME <-> CLIENT_TABLE.CLIENT.CLIENT_NAME: 0.6900491
CLAIMANT_TABLE.CLAIMANT.ADDRESS <->
        CLIENT_TABLE.CLIENT.CLIENT_ADDRESS: 0.6900491
CLAIMANT_TABLE.CLAIMANT.STATE <-> CLIENT_TABLE.CLIENT.CLIENT_STATE: 0.6900491
CLAIMANT_TABLE <-> CLIENT_TABLE: 0.72654325
CLAIM_DESC_TABLE.CLAIM_DESC.TYPE_KEY <->
        CLAIM_NATURE_TABLE.CLAIM_NATURE.NATURE_KEY: 0.61595464
CLAIM_DESC_TABLE.CLAIM_DESC.CLAIMANT_KEY <->
        CLAIM_FINANCIAL_TABLE.CLAIM_FINANCIAL.CLIENT_KEY: 0.69045764
CLAIM_DESC_TABLE.CLAIM_DESC.PROC_CODE <->
        CLAIM_NATURE_TABLE.CLAIM_NATURE.NATURE_CODE: 0.6044005
PERIOD_TABLE.PERIOD.PER_KEY <-> DATES_TABLE.DATES.DATE_KEY: 0.50935185
PERIOD_TABLE.PERIOD.MONTH <-> DATES_TABLE.DATES.CALENDAR_MONTH: 0.6390741
PERIOD_TABLE.PERIOD.MONTH <-> DATES_TABLE.DATES.FINANCIAL_MONTH: 0.6390741
PERIOD_TABLE.PERIOD.YEAR <-> DATES_TABLE.DATES.CALENDAR_YEAR: 0.6814352
PERIOD_TABLE <-> DATES_TABLE: 0.61703706
POLICY_TABLE.POLICY.POLICY_KEY <->
        POLICY_HADER_TABLE.POLICY_HADER.POLICY_AGREEMENT_TYPE_KEY: 0.6866576
POLICY_TABLE.POLICY.POLICY_TYPE <->
        POLICY_AGREEMENT_TYPE_TABLE.POLICY_AGREEMENT_TYPE.
        POLICY_AGREEMENT_TYPE_CODE: 0.72884274
POLICY_TABLE <-> POLICY_AGREEMENT_TYPE_TABLE: 0.70771325
POLICY_SALES_TABLE.POLICY_SALES.PREMIUM_DOLLARS <->
        CLAIM_FINANCIAL_TABLE.CLAIM_FINANCIAL.BASIC_PREMIUM_AMT: 0.44146645
```

```
Expected results agreed by at least 2 participants
----------------------------------------------------------------------------
1,a     T05A.CLAIM_TYPE            CREATE T05B.CLAIM_NATURE
2,b     CLAIM_TYPE.TYPE_KEY        CLAIM_NATURE.NATURE_KEY
3,c     CLAIM_TYPE.TYPE_DESC       CLAIM_NATURE.NATURE_DESCRIPTION
4,s     T05A.CLAIMANT              T05B.CLIENT
5,t     CLAIMANT.CLAIMANT_KEY      CLIENT.CLIENT_KEY
6,u     CLAIMANT.NAME              CLIENT.CLIENT_NAME
7,v     CLAIMANT.ADDRESS           CLIENT.CLIENT_ADDRESS
9,w     CLAIMANT.STATE             CLIENT.CLIENT_STATE
10,i    T05A.PERIOD                T05B.DATES
11,j    PERIOD.PER_KEY             DATES.DATE_KEY
12,l    PERIOD.MONTH               DATES.CALENDAR_MONTH
13,k    PERIOD.YEAR                DATES.CALENDAR_YEAR
17,g    POLICY.POLICY_TYPE
        POLICY_AGREEMENT_TYPE.POLICY_AGREEMENT_TYPE_CODE
20,s    T05A.POLICY_HOLDER         T05B.CLIENT
21,t    POLICY_HOLDER.POLICY_HOLDER_KEY  CLIENT.CLIENT_KEY
22,u    POLICY_HOLDER.NAME         CLIENT.CLIENT_NAME
23,v    POLICY_HOLDER.ADDRESS      CLIENT.CLIENT_ADDRESS
25,w    POLICY_HOLDER.STATE        CLIENT.CLIENT_STATE
30,cc   CLAIM_DESC.TYPE_KEY        CLAIM_FINANCIAL.NATURE_KEY
43,ff   CLAIMS.CLAIMANT_KEY        CLAIM_FINANCIAL.CLIENT_KEY
48,aa   T05A.POLICY_SALES.POLICY_KEY     CLAIM_FINANCIAL.POLICY_NO


Matching Results between T06B and T04 using the Relational model:
Columns:
--------------------------------- ----------------------------      ------
CALENDAR.DAY_OF_YEAR              TIME.DAY_OF_WEEK                  0.0614
ORG_DETAIL.REGION_NAME            STORE.REGION                      0.0617
STATE_LOOKUP.STATE_NAME           STORE.STORE_NAME                  0.0552

Tables:
--------------------------------- ----------------------------      ------
CALENDAR TIME 0.1000
MONTH_LOOKUP TIME.MONTH 0.0599
ORG_DETAIL STORE 0.0999
PROD_LINE_CATEGORY_LOOKUP SALES_BY_CATEGORY 0.0455

Matching Results between T06B and T04 using the Star model:
Columns:
--------------------------------- ----------------------------      ------
CALENDAR.DAY_OF_YEAR              TIME.DAY_OF_WEEK                  0.0467
ORG_DETAIL.REGION_NAME            STORE.REGION                      0.0465
STATE_LOOKUP.STATE_CODE           STORE.STORE_KEY                   0.0152
STATE_LOOKUP.STATE_NAME           STORE.STORE_NAME                  0.0412
TRANSACTION.PRODUCT_IDENTIFIER    SALES_BY_CATEGORY.PRODUCT_KEY     0.0260
TRANSACTION.TIME_IDENTIFIER       SALES_BY_CATEGORY.TIME_KEY        0.0286

Tables:
--------------------------------- ----------------------------      ------
CALENDAR                          TIME                              0.2056
ORG_DETAIL                        STORE                             0.1930
PROD_LINE_CATEGORY_LOOKUP         CATEGORY                          0.0938
TRANSACTION                       SALES_BY_CATEGORY                 0.7051
```

```
COMA Results
----------------------------------------------------------------------------
STATE_LOOKUP_TABLE.STATE_LOOKUP.STATE_NAME <->
        STORE_TABLE.STORE.STORE_NAME: 0.62557685
STATE_LOOKUP_TABLE <-> STORE_TABLE: 0.5135469
ORG_DETAIL_TABLE.ORG_DETAIL.REGION_NAME <->
        STORE_TABLE.STORE.REGION: 0.59808177
MONTH_LOOKUP_TABLE.MONTH_LOOKUP.MONTH_NUMBER <->
TIME_TABLE.TIME.MONTH: 0.5025678
CALENDAR_TABLE.CALENDAR.DAY_OF_YEAR <->
        TIME_TABLE.TIME.DAY_OF_WEEK: 0.49160638
CALENDAR_TABLE <-> TIME_TABLE: 0.44984058
PROD_LINE_CATEGORY_LOOKUP_TABLE.
PROD_LINE_CATEGORY_LOOKUP.PRODUCT_CATEGORY_CODE <->
        CATEGORY_TABLE.CATEGORY.CATEGORY: 0.55901647
PROD_LINE_CATEGORY_LOOKUP_TABLE <-> CATEGORY_TABLE: 0.5512149
TRANSACTION_TABLE.TRANSACTION.TIME_IDENTIFIER <->
        TIME_TABLE.TIME.TIME_KEY: 0.42954636

Expected results agreed by at least 2 participants
----------------------------------------------------------------------------
26,t    T06B.CALENDAR                     T04.TIME
27,u    CALENDAR.TIME_IDENTIFIER          TIME.TIME_KEY
28,w    CALENDAR.MONTH_NUMBER             TIME.MONTH
35,a    PROD_LINE_CATEGORY_LOOKUP         T04.CATEGORY
36,c    PROD_LINE_CATEGORY_LOOKUP.PRODUCT_CATEGORY_CODE
                                          CATEGORY.CATEGORY
40,b    PROD_LINE.PRODUCT_IDENTIFIER      CATEGORY.PRODUCT_KEY
46,e    T06B.TRANSACTION                  T04.SALES_BY_CATEGORY
50,h    TRANSACTION.PRODUCT_IDENTIFIER    SALES_BY_CATEGORY.PRODUCT_KEY
51,i    TRANSACTION.SALES_DOLLAR          SALES_BY_CATEGORY.DOLLARS_SOLD
223

Matching Results between T07B and T11 using the Relational model:
Columns:
--------------------------------- ----------------------------         ------
CUSTOMER.ADDRESS1                  CUSTOMER.CUSTOMER_ADDRESS             0.0388
CUSTOMER.BIRTHDATE                 CUSTOMER.CUSTOMER_BIRTHDATE           0.0401
CUSTOMER.CITY                      GEOGRAPHIC_LOCATION.SALES_LOCATION_CITY
                                                                         0.0286
CUSTOMER.FNAME                     CUSTOMER.CUSTOMER_FNAME               0.0379
CUSTOMER.LNAME                     CUSTOMER.CUSTOMER_LNAME               0.0382
CUSTOMER.TOTAL_CHILDREN            CUSTOMER.NUMBER_OF_CHILDREN           0.0233
PRODUCT.PRODUCT_NAME               PRODUCT.PRODUCT_NAME                  0.0580
STORE.STORE_NUMBER                 DATE.WEEK_NUMBER                      0.0283
TIME_BY_DAY.THE_MONTH              DATE.MONTH_NUMBER                     0.0289
TIME_BY_DAY.THE_YEAR               DATE.YEAR_NUMBER                      0.0289

Tables:
--------------------------------- ----------------------------         ------
CUSTOMER                           CUSTOMER                              0.1449
PRODUCT PRODUCT                    0.1258
SALES   PRODUCT_SALES              0.1284
TIME_BY_DAY                        DATE   0.0912
```

```
Matching Results between T07B and T11 using the Star model:
Columns:
--------------------------------- ----------------------------         ------
CUSTOMER.ADDRESS1                 CUSTOMER.CUSTOMER_ADDRESS              0.0392
CUSTOMER.BIRTHDATE                CUSTOMER.CUSTOMER_BIRTHDATE            0.0405
CUSTOMER.CITY                     GEOGRAPHIC_LOCATION.SALES_LOCATION_CITY
                                                                         0.0285
CUSTOMER.CUSTOMER_ID              CUSTOMER.CUSTOMER_ID                   0.0275
CUSTOMER.FNAME                    CUSTOMER.CUSTOMER_FNAME                0.0382
CUSTOMER.LNAME                    CUSTOMER.CUSTOMER_LNAME                0.0386
CUSTOMER.TOTAL_CHILDREN           CUSTOMER.NUMBER_OF_CHILDREN            0.0241
PRODUCT.PRODUCT_NAME              PRODUCT.PRODUCT_NAME                   0.0608
SALES.CUSTOMER_ID                 PRODUCT_SALES.CUSTOMER_ID              0.0243
SALES.STORE_COST                  PRODUCT_SALES.SALES_COST               0.0404
STORE.STORE_NUMBER                DATE.WEEK_NUMBER                       0.0284
TIME_BY_DAY.THE_DATE              DATE.DAY_DATE                          0.0368
TIME_BY_DAY.THE_MONTH             DATE.MONTH_NUMBER                      0.0302
TIME_BY_DAY.THE_YEAR              DATE.YEAR_NUMBER                       0.0302

Tables:
--------------------------------- ----------------------------         ------
CUSTOMER                          CUSTOMER                               0.3568
PRODUCT                           PRODUCT                                0.2899
SALES                             PRODUCT_SALES                          0.7625
STORE                             GEOGRAPHIC_LOCATION                    0.1503
TIME_BY_DAY                       DATE                                   0.2689

COMA Results
----------------------------------------------------------------------------
PRODUCT_TABLE.PRODUCT.PRODUCT_ID <->
        PRODUCT_TABLE.PRODUCT.PRODUCT_CODE: 0.66858333
PRODUCT_TABLE.PRODUCT.PRODUCT_NAME <->
        PRODUCT_TABLE.PRODUCT.PRODUCT_NAME: 0.8943154
PRODUCT_TABLE <-> PRODUCT_TABLE: 0.78863084
STORE_TABLE.STORE.STORE_CITY <->
        GEOGRAPHIC_LOCATION_TABLE.GEOGRAPHIC_LOCATION.SALES_LOCATION_CITY:
        0.477647
CUSTOMER_TABLE.CUSTOMER.CUSTOMER_ID <->
        CUSTOMER_TABLE.CUSTOMER.CUSTOMER_ID: 0.91481817
CUSTOMER_TABLE.CUSTOMER.LNAME <->
        CUSTOMER_TABLE.CUSTOMER.CUSTOMER_LNAME: 0.73678505
CUSTOMER_TABLE.CUSTOMER.FNAME <->
        CUSTOMER_TABLE.CUSTOMER.CUSTOMER_FNAME: 0.73678505
CUSTOMER_TABLE.CUSTOMER.ADDRESS1 <->
        CUSTOMER_TABLE.CUSTOMER.CUSTOMER_ADDRESS: 0.73678505
CUSTOMER_TABLE.CUSTOMER.CITY <->
        GEOGRAPHIC_LOCATION_TABLE.GEOGRAPHIC_LOCATION.SALES_LOCATION_CITY:
        0.47915852
CUSTOMER_TABLE.CUSTOMER.BIRTHDATE <->
        CUSTOMER_TABLE.CUSTOMER.CUSTOMER_BIRTHDATE: 0.73678505
CUSTOMER_TABLE.CUSTOMER.TOTAL_CHILDREN <->
        CUSTOMER_TABLE.CUSTOMER.NUMBER_OF_CHILDREN: 0.6125258
CUSTOMER_TABLE <-> CUSTOMER_TABLE: 0.812459
TIME_BY_DAY_TABLE.TIME_BY_DAY.THE_DATE <->
        DATE_TABLE.DATE.DAY_DATE: 0.6427102
TIME_BY_DAY_TABLE.TIME_BY_DAY.THE_YEAR <->
        DATE_TABLE.DATE.YEAR_NUMBER: 0.54808575
TIME_BY_DAY_TABLE.TIME_BY_DAY.WEEK_OF_YEAR <->
```

Appendix D

StarMod Specification in OWL

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE rdf:RDF [
        <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
        <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
        <!ENTITY owl "http://www.w3.org/2002/07/owl#">
        <!ENTITY dbs "Relational.owl#">
        <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
        <owl:Ontology rdf:about="http://www.w3.org/2000/01/star-schema#"
                dc:title="The STAR Schema vocabulary (STAR)">
                <rdfs:comment xml:lang="en">An Ontology to describe the schema
                               information of Star schemas.</rdfs:comment>
        </owl:Ontology>

        <owl:Class rdf:ID="Star">
                <rdfs:subClassOf rdf:resource="&rdf;Bag"/>
                <rdfs:label xml:lang="en">Star Schema</rdfs:label>
                <rdfs:comment xml:lang="en">A star is an unordered set of things (i.e.
                               dimensions and facts).</rdfs:comment>
        </owl:Class>

        <owl:ObjectProperty rdf:ID="hasDimension">
                <rdfs:label xml:lang="en">Has Dimension</rdfs:label>
                <rdfs:comment xml:lang="en">Star schema can have zero or more dimension
                               tables.</rdfs:comment>
                <rdfs:subPropertyOf rdf:resource="&dbs;has"/>
                <rdfs:domain rdf:resource="#Star"/>
                <rdfs:range rdf:resource="#DimensionTable"/>
        </owl:ObjectProperty>

        <owl:Class rdf:ID="DimensionTable">
                <rdfs:subClassOf rdf:resource="&dbs;Table"/>
                <rdfs:label xml:lang="en">Dimension Table</rdfs:label>
                <rdfs:comment xml:lang="en">A dimension table is an ordered list (of
                               attributes) and a subtype of the relational
                               table.</rdfs:comment>
        </owl:Class>

        <owl:ObjectProperty rdf:ID="hasAttribute">
                <rdfs:label xml:lang="en">Has Attribute</rdfs:label>
                <rdfs:comment xml:lang="en">Dimesnion tables have one or more dimension
                        attributes.</rdfs:comment>
                <rdfs:subPropertyOf rdf:resource="&dbs;has"/>
                <rdfs:domain rdf:resource="#DimensionTable"/>
                <rdfs:range rdf:resource="#Attribute"/>
        </owl:ObjectProperty>

        <owl:Class rdf:ID="Attribute">
                <rdfs:subClassOf rdf:resource="&dbs;Column"/>
                <rdfs:label xml:lang="en">Dimension Attribute</rdfs:label>
                <rdfs:comment xml:lang="en">Attribute is a subtype of a
                        column.</rdfs:comment>
        </owl:Class>

        <owl:Class rdf:ID="SurrogateKeyAttribute">
                <rdfs:subClassOf rdf:resource="#Attribute"/>
                <rdfs:label xml:lang="en">SurrogateKeyAttribute</rdfs:label>
                <rdfs:comment xml:lang="en">Surrogate key attribute is a subtype of
                        attribute and is generated sequential number. It uniquely
                        identifies a row in the dimension table</rdfs:comment>
        </owl:Class>
```

```
<owl:Class rdf:ID="DataAttribute">
        <rdfs:subClassOf rdf:resource="#Attribute"/>
        <rdfs:label xml:lang="en">Data Attribute</rdfs:label>
        <rdfs:comment xml:lang="en">A data attribute is a subtype of attribute
                used for aggregation of measures</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="DegenerateFact">
        <rdfs:subClassOf rdf:resource="#Attribute"/>
        <rdfs:label xml:lang="en">Degenerate Fact</rdfs:label>
        <rdfs:comment xml:lang="en">A degenerate fact is a subtype of attribute.
                Although part of the dimension table can be aggregated. Use of
                degenerate facts are not recommended.</rdfs:comment>
</owl:Class>

<owl:ObjectProperty rdf:about="#hasAttribute">
        <rdfs:label xml:lang="en"> </rdfs:label>
        <rdfs:comment xml:lang="en">Keys have attributes.</rdfs:comment>
        <rdfs:subPropertyOf rdf:resource="&dbs;has"/>
        <rdfs:domain rdf:resource="&dbs;Key"/>
        <rdfs:range rdf:resource="#Attribute"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="DimUniqueKey">
        <rdfs:subClassOf rdf:resource="&dbs;Key"/>
        <rdfs:label xml:lang="en">Dimension Unique Key</rdfs:label>
        <rdfs:comment xml:lang="en">Dimension Unique Key (other than primary key)
                is a subtype of the key defined in Relational.owl</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="DimRegularKey">
        <rdfs:subClassOf rdf:resource="&dbs;Key"/>
        <rdfs:label xml:lang="en">Dimension Regular Key</rdfs:label>
        <rdfs:comment xml:lang="en">A regular key is a subtype of key that is not
                unique.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="PrimaryKey">
        <rdfs:subClassOf rdf:resource="&dbs;Key"/>
        <rdfs:label xml:lang="en">Primary Key</rdfs:label>
        <rdfs:comment xml:lang="en">A primary key is a subtype of
                key.</rdfs:comment>
</owl:Class>

<owl:ObjectProperty rdf:about="#refersTo">
        <rdfs:label xml:lang="en">Refers to</rdfs:label>
        <rdfs:comment xml:lang="en">This links a fact table to a dimension table
                through the surrogate key in the dimension and the same column as
                the foreign key in the fcat table.</rdfs:comment>
        <rdfs:domain rdf:resource="#SurrogateKeyReference"/>
        <rdfs:range rdf:resource="#SurrogateKeyAttribute"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#refersTo">
        <rdfs:label xml:lang="en">Refers to</rdfs:label>
        <rdfs:comment xml:lang="en">This link joins dimensions creating
                snowflaked dimensions through the surrogate key.</rdfs:comment>
        <rdfs:domain rdf:resource="#SurrogateKeyAttribute"/>
        <rdfs:range rdf:resource="#SurrogateKeyAttribute"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#refersTo">
        <rdfs:label xml:lang="en">Refers to</rdfs:label>
        <rdfs:comment xml:lang="en">This relation represents snowflaked
```

```
                dimensions which are linked through natural keys.</rdfs:comment>
        <rdfs:domain rdf:resource="#DataAttribute"/>
        <rdfs:range rdf:resource="#DataAttribute"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="dimUniquelyIdentifiedBy">
        <rdfs:label xml:lang="en">DimensionUniquely Identified by a primary
                key</rdfs:label>
        <rdfs:comment xml:lang="en">A primary key uniquely identifies a row in
                the dimension table.</rdfs:comment>
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:domain rdf:resource="#DimensionTable"/>
        <rdfs:range rdf:resource="#PrimaryKey"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#dimUniquelyIdentifiedBy">
        <rdfs:label xml:lang="en">Dimension Uniquely Identified By</rdfs:label>
        <rdfs:comment xml:lang="en">A unique key uniquely identifies a row in the
                dimension table. </rdfs:comment>
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:domain rdf:resource="#DimensionTable"/>
        <rdfs:range rdf:resource="#DimUniqueKey"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="dimIdentifiedBy">
        <rdfs:label xml:lang="en">Dimension Identified By a non unique
                key</rdfs:label>
        <rdfs:comment xml:lang="en">A regular key identfies instances from a
                dimension table</rdfs:comment>
        <rdfs:domain rdf:resource="#DimensionTable"/>
        <rdfs:range rdf:resource="#RegularKey"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="FactTable">
        <rdfs:subClassOf rdf:resource="&dbs;Table"/>
        <rdfs:label xml:lang="en">Fact Table</rdfs:label>
        <rdfs:comment xml:lang="en">A fact table is an ordered list of things
                (i.e. columns) and is a subtype of the table.</rdfs:comment>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasFact">
        <rdfs:label xml:lang="en">Has Fact</rdfs:label>
        <rdfs:comment xml:lang="en">A star schema must have at least one fact
                table.</rdfs:comment>
        <rdfs:subPropertyOf rdf:resource="&dbs;has"/>
        <rdfs:domain rdf:resource="#Star"/>
        <rdfs:range rdf:resource="#FactTable"/>
</owl:ObjectProperty>


<owl:ObjectProperty rdf:ID="hasElement">
        <rdfs:label xml:lang="en"> Has Element</rdfs:label>
        <rdfs:comment xml:lang="en">Fact tables have one or more
                elements</rdfs:comment>
        <rdfs:subPropertyOf rdf:resource="&dbs;has"/>
        <rdfs:domain rdf:resource="#FactTable"/>
        <rdfs:range rdf:resource="#Element"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasElement">
        <rdfs:label xml:lang="en"> </rdfs:label>
        <rdfs:comment xml:lang="en">Fact key containing elements.</rdfs:comment>
        <rdfs:subPropertyOf rdf:resource="&dbs;has"/>
        <rdfs:domain rdf:resource="&dbs;Key"/>
        <rdfs:range rdf:resource="#Element"/>
```

```
</owl:ObjectProperty>

<owl:Class rdf:ID="Measure">
        <rdfs:subClassOf rdf:resource="#Element"/>
        <rdfs:label xml:lang="en">Measure</rdfs:label>
        <rdfs:comment xml:lang="en">Measures are kinds of elements. Measures can
                be aggregated.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="DegenerateDimension">
        <rdfs:subClassOf rdf:resource="#Element"/>
        <rdfs:label xml:lang="en">Degenerate dimension</rdfs:label>
        <rdfs:comment xml:lang="en">Degenerate dimension is a subtype of element
                but is more of a dimension attribute than fact
                element.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="SurrogateKeyReference">
        <rdfs:subClassOf rdf:resource="#Element"/>
        <rdfs:label xml:lang="en">Surrogate key Reference</rdfs:label>
        <rdfs:comment xml:lang="en">Surrogate Key reference is a subtype of
                element. It refers to teh surrogate key in a dimension
                table.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="SurrogateKeyElement">
        <rdfs:subClassOf rdf:resource="#Element"/>
        <rdfs:label xml:lang="en">Surrogate key Element</rdfs:label>
        <rdfs:comment xml:lang="en">Surrogate key element is a subtype of
                element. It plays the same role as the surrogate key attribute
                except that it is not referred to.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="FactUniqueKey">
        <rdfs:subClassOf rdf:resource="&dbs;Key"/>
        <rdfs:label xml:lang="en">Unique Fact Key</rdfs:label>
        <rdfs:comment xml:lang="en">A Unique key for fact tables is subtype of
                key uniquely identifying a row in the fact table.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="FactRegularKey">
        <rdfs:subClassOf rdf:resource="&dbs;Key"/>
        <rdfs:label xml:lang="en">Regular Fact key</rdfs:label>
        <rdfs:comment xml:lang="en">A regular key for fact tables is subtype of
                key. This is not a unique key.</rdfs:comment>
</owl:Class>

<owl:ObjectProperty rdf:about="factIdentifiedBy">
        <rdfs:label xml:lang="en">Fact Identified by a non unique
                key</rdfs:label>
        <rdfs:comment xml:lang="en">A fact regular key identifies instances from
                fact table.</rdfs:comment>
        <rdfs:domain rdf:resource="#FactTable"/>
        <rdfs:range rdf:resource="#FactRegularKey"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="factUniquelyIdentifiedBy">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:label xml:lang="en">Fact Uniquely Identified By</rdfs:label>
        <rdfs:comment xml:lang="en">Fact  rows are uniquely identified using fact
                unique key.</rdfs:comment>
        <rdfs:domain rdf:resource="#FactTable"/>
        <rdfs:range rdf:resource="#FactUniqueKey"/>
</owl:ObjectProperty>
```

```
<owl:Class rdf:ID="Hierarchy">
        <rdfs:subClassOf rdf:resource="&rdf;Seq"/>
        <rdfs:label xml:lang="en">Dimension hierarchy</rdfs:label>
        <rdfs:comment xml:lang="en">Dimension hierarchies describe the roll-up
                relations.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Level">
        <rdfs:subClassOf rdf:resource="&rdf;Seq"/>
        <rdfs:label xml:lang="en">Level</rdfs:label>
        <rdfs:comment xml:lang="en">Aggregation of fact measures are done
                alongside the hierarchy levels.</rdfs:comment>
</owl:Class>

<owl:ObjectProperty rdf:about="hasLevel">
 <rdfs:subPropertyOf rdf:resource="&dbs;has"/>
        <rdfs:label xml:lang="en">Has Level</rdfs:label>
        <rdfs:comment xml:lang="en">A hierarchy has one or more
                levels.</rdfs:comment>
        <rdfs:domain rdf:resource="#Hierarchy"/>
        <rdfs:range rdf:resource="#Level"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="hasDataAttribute">
 <rdfs:subPropertyOf rdf:resource="&dbs;has"/>
        <rdfs:label xml:lang="en">Has Data Attribute</rdfs:label>
        <rdfs:comment xml:lang="en">A level may have one data
                attributes.</rdfs:comment>
        <rdfs:domain rdf:resource="#Level"/>
        <rdfs:range rdf:resource="#DataAttribute"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="rollsUpTo">
 <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:label xml:lang="en">Rolls up to</rdfs:label>
        <rdfs:comment xml:lang="en">A level rolls up to other parent
                levels.</rdfs:comment>
        <rdfs:domain rdf:resource="#Hierarchy"/>
        <rdfs:range rdf:resource="#Level"/>
</owl:ObjectProperty>

<owl:Restriction>
        <rdfs:comment xml:lang="en">All of the following relationships have a
                mandatory one to one relationship.</rdfs:comment>
        <owl:onProperty rdf:resource="#hasFact"/>
        <owl:onProperty rdf:resource="#hasPrimaryKey"/>
        <owl:minCardinality
                rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
        <owl:maxCardinality
                rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
</owl:Restriction>

<owl:Restriction>
        <owl:onProperty rdf:resource="#hasDimension"/>
        <rdfs:comment xml:lang="en">The following relationships have a one to
                        many optional relationship.</rdfs:comment>
        <owl:minCardinality
                rdf:datatype="&xsd;nonNegativeInteger">0</owl:minCardinality>
</owl:Restriction>

<owl:Restriction>
        <rdfs:comment xml:lang="en">The following relationships have a mandatory
                         one to many optional relationship.</rdfs:comment>
        <owl:onProperty rdf:resource="#hasAttribute"/>
        <owl:onProperty rdf:resource="#hasElement"/>
```

Bibliography

A. Abelló, J. Samos, and F. Saltor. On relationships offering new drill-across possibilities. In *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP*, DOLAP '02, pages 7–13. ACM, 2002.

A. Abelló, J. Samos, and F. Saltor. Implementing operations to navigate semantic star schemas. In *Proceedings of the 6th ACM International Workshop on Data Warehousing and OLAP*, DOLAP '03, pages 56–62. ACM, 2003.

A. Abelló, J. Samos, and F. Saltor. YAM^2: A multidimensional conceptual model extending UML. *Information Systems*, 31(6):541–567, 2005.

S. Agarwal, A. M. Keller, G. Wiederhold, and K. Saraswat. Flexible relation: An approach for integrating data from multiple, possibly inconsistent databases. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 495–504, 1995.

A. Aho, M. Garey, and J. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

J. Akoka, I. Comyn-Wattiau, and N. Prat. Dimension hierarchies design from UML generalizations and aggregations. In *Conceptual Modeling ER 2001*, volume 2224 of *LNCS*, pages 442–445. Springer-Verlag, 2001.

R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *28th International Conference on Very Large Data Bases*, VLDB'02, pages 586–597. VLDB Endowment, 2002.

P. Anokhin and A. Motro. Data integration: Inconsistency detection and resolution based on source properties. In *Proceedings of the 1st International Workshop on Foundations of Models for Information Integration*, FMII '01, pages 176–196. George Mason University, 2001.

M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the 18th Symposium on Principles of Database Systems*, PODS '99, pages 68–79. ACM, 1999.

M. Arenas, L. E. Bertossi, and J. Chomicki. Scalar aggregation in fd-inconsistent databases. In *The 8th International Conference on Database Theory*, volume 1973 of *LNCS*, pages 39–53. Springer-Verlag, 2001.

M. Banek, B. Vrdoljak, and M. Tjoa. Automating the schema matching process for hetrogeneous data warehouses. *International Journal of Data Warehousing and Mining*, 4(4):1–21, 2008.

C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

P. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-strength schema matching. *SIGMOD Record*, 33(4):38–43, 2004.

R. A. Best. Sharing law enforcement and intelligence information: The congressional role. Technical report, Foreign Affairs, Defense, and Trade Division, United States, 2007.

L. Betrossi, L. Bravo, and M. Caniupan. Consistent query answering in data warehouses. In *Alberto Mendelzon International Workshop on Foundations of Data Management*, AMW '2009, 2009.

S. Brockmans, R. Volz, A. Eberhart, and P. Lffler. Visual modeling of OWL DL ontologies using UML. In *Proceedings of the 3rd International Semantic Web Conference*, volume 3298 of *LNCS*, pages 198–213. Springer-Verlag, 2004.

D. Burdick, P. M. D. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. OLAP over uncertain and imprecise data. *The VLDB Journal*, 16:123–144, 2007.

Business Objects and Teradata. Data mart consolidation and business intelligence standardization: Getting the most out of information, 2007. URL www.businessobjects.com/pdf/investors/data_mart_consolidation.pdf.

L. Cabibbo and R. Torlone. A logical approach to multidimensional databases. In *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology*, volume 1377 of *LNCS*, pages 183–197. Springer-Verlag, 1998.

L. Cabibbo and R. Torlone. Dimension compatibility for data mart integration. In *Proceedings of The 12th Italian Symposium on Advanced Database Systems*, pages 6–17, 2004.

L. Cabibbo and R. Torlone. Integrating heterogeneous multidimensional databases. In *Proceedings of the 17th international conference on Scientific and Statistical Database Management*, SSDBM '2005, pages 205–214. Lawrence Berkeley Laboratory, 2005.

P. Calado, M. Herschel, and L. Leito. An overview of XML duplicate detection algorithms. In *Software Computing in XML Data Management*, volume 255/2010 of *Studies in Fuzziness and Soft Computing*, pages 193–224. Springer-Verlag, 2010.

A. Carmè, J.-N. Mazón, and S. Rizzi. A model-driven deuristic approach for detecting multidimensional facts in relational data sources. In *Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery*, volume 6263 of *LNCS*, pages 13–24. Springer-Verlag, 2010.

C. Carpineto, G. Romano, and P. d'Adamo. Inferring dependencies from relations: A conceptual clustering approach. *International Journal of Intelligent Systems*, 15(4):415–441, 2009.

S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26:65–74, 1997.

Y.-T. Chen and P.-Y. Hsu. A grain preservation translation algorithm: From ER diagram to multidimensional model. *Information Science*, 177(18):3679–3695, 2007.

J. Chomicki. Consistent query answering: Five easy pieces. In *Proceedings of the 11th International Conference on Database Theory*, volume 4353 of *LNCS*, pages 1–17. Springer-Verlag, 2006.

N. Conklin, S. Prabhakar, and C. North. Multiple foci drill-down through tuple and attribute aggregation polyarchies in tabular data. In *Proceedings of the IEEE Symposium on Information Visualization*, INFOVIS '02, pages 131–134. IEEE Computer Society, 2002.

C. P. de Laborda and S. Conrad. Relational OWL: A data and schema representation format based on OWL. In *Proceedings of the 2nd Asia-Pacific Conference on Conceptual Modelling*, pages 89–96, 2005.

H.-H. Do and E. Rahm. COMA: A system for flexible combination of schema matching approaches. In *Proceedings of the 28th International Conference on Very Large Databases*, VLDB '02, pages 610–621, 2002.

F. Duchateau, Z. Bellahsene, and R. Coletta. A flexible approach for planning schema matching algorithms. In *On the Move to Meaningful Internet Systems*, volume 5331 of *LNCS*, pages 249–264. Springer-Verlag, 2008.

Eclipse. UML2OWL, 2010. URL http://code.google.com/p/twouse/wiki/UML2OWL.

S. G. Eick. Visualizing multidimensional data. *SIGGRAPH Computational and Statistical Graphics*, 34(1):61–67, 2000.

A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, January 2007.

D. Engmann and S. Massmann. Instance matching with COMA++. In *BTW Workshops*, pages 28–37, Aachen, Germany, 2007.

C. Fellbaum, R. Tengi, and L. Jose. Wordnet: A Lexical Database For English, 2011. URL http://wordnet.princeton.edu/.

E. Franconi and U. Sattler. A datawarehouse conceptual datamodel for multidimensional aggregation: A preliminary report. In *Proceedings of the Workshop on Design and Management of Data Warehouses*, volume 1 of *(DMDW'99)*, pages 9–21. Italian Association for Artifcial Intelligence, 1999.

A. Gal. Why is schema matching tough and what can we do about it? *SIGMOD Record*, 35 (4):2–5, December 2006.

M. Gebhardt, M. Jarke, M. Jeusfeld, C. Quix, and S. Sklorz. Tools for data warehouse quality. In *Proceedings of 10th International Conference on Scientific and Statistical Database Management*, SSDM '1998, pages 229–232. IEEE Computer Society, 1998.

W. Giovinazzo. *Object Oriented Data Warehouse Design, Building a Star Schema*. Prentice Hall, Inc., 2000.

M. Golfarelli, S. Rizzi, and B. Vrdoljak. Data warehouse design from XML sources. In *Proceedings of the 4th ACM International Workshop on Data Warehousing and OLAP*, DOLAP '01, pages 40–47. ACM, 2001.

S. Greco and C. Molinaro. Towards relational inconsistent databases with functional dependencies. In *13th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, volume 5178 of *LNCS*, pages 695–702. Springer-Verlag, 2008.

W. Grossmann and M. Moschner. Knowledge integration from multidimensional data sources. In *Computer Aided Systems Theory*, volume 4739 of *LNCS*, pages 345–351. Springer-Verlag, 2007.

L. Haas, M. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: From research prototype to industrial tool. In *Proceedings of the 2005 ACM SIGMOD international*, pages 805–810, 2005.

M. S. Hacid and U. Sattler. An object-centered multi-dimensional data model with hierarchically structured dimensions. In *Proceedings of the 1997 IEEE Knowledge and Data Engineering Exchange Workshop*, pages 65–72. IEEE Computer Society, 1997.

J. Han and M. Kamber. *Data Mining - Concepts and Techniques*. Elsevier Inc., 2nd edition, 2006.

J. Horner and I.-Y. Song. A taxonomy of inaccurate summaries and their management in OLAP systems. In *24th International Conference on Conceptual Modeling*, volume 3716 of *LNCS*, pages 433–448. Springer-Verlag, 2005.

J. Horner, I.-Y. Song, and P. P. Chen. An analysis of additivity in olap systems. In *ACM 7th International Workshop on Data Warehousing and OLAP*, DOLAP '04, pages 83–91. ACM, 2004.

C. A. Hurtado and A. O. Mendelzon. Reasoning about summarizability in heterogeneous multidimensional schemas. In *Proceedings of the 8th International Conference on Database Theory*, volume 1973 of *LNCS*, pages 375–389. Springer-Verlag, 2001.

C. A. Hurtado and A. O. Mendelzon. OLAP dimension constraints. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 169–179. ACM, 2002.

C. A. Hurtado, A. O. Mendelzon, and A. A. Vaisman. Updating OLAP dimensions. In *ACM 2nd International Workshop on Data Warehousing and OLAP*, DOLAP '99, pages 60–66. ACM, 1999.

IBM. Websphere software, 2012. URL http://www-01.ibm.com/software/webspher.

IBM DB2. Multidimensional clustering, 2009. URL http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/admin/c0007201.htm.

C. Imhoff, N. Galemmo, and J. Geiger. *Mastering Data Warehouse Design*. Wiley Publishing, Inc., 2003.

W. H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, third edition, 2005.

M. Jensen, T. Holmgren, and T. Pedersen. Discovering multidimensional structure in relational data. In *Data Warehousing and Knowledge Discovery*, volume 3181 of *LNCS*, pages 138–148. Springer-Verlag, 2004.

M. R. Jensen, T. H. Møller, and T. B. Pedersen. Specifying OLAP cubes on XML data. *Journal of Intelligent Information Systems*, 17(2-3):255–280, December 2001.

M. R. Jensen, T. H. Møller, and T. B. Pedersen. Converting XML DTDs to UML diagrams for conceptual data integration. *Data Knowlege and Engineering*, 44(3):323–346, March 2003.

K. Kailing, H.-P. Kriegel, S. Schoenauer, and T. Seidl. Efficient similarity search for hierarchical data in large databases. In *Extending Database Technology*, EDBT'09, pages 676–693, 2004.

A. S. Kamble. A conceptual model for multidimensional data. In *Proceedings of the 5th Asia-Pacific Conference on Conceptual Modelling*, pages 29–38. Australian Computer Society, Inc., 2008.

M. Kantola, H. Mannila, K. Räihä, and H. Siirtola. Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligent Systems*, 7:591–607, 1992.

R. Kimball and M. Ross. *The Data Warehouse Toolkit*. Wiley Publishing, Inc., Indianapolis, USA, 2002.

E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of the 24th International Conference on Very Large Databases*, VLDB'98, pages 392–403. Morgan Kaufmann Publishers Inc., 1998.

C. Letz, E. T. Henn, and G. Vossen. Consistency in data warehouse dimensions. In *International Database Engineering and Applications*, IDEAS '02, pages 224–232. IEEE Computer Society, 2002.

L. Li and L. Yang. Automatic schema matching for data warehouses. In *5th World Congress on Intelligent Control and Automation*, pages 3939–3943. IEEE, 2004.

D. Linestedt. History of the data vault, 2011. URL http://danlinstedt.com/datavaultcat/dvhistory/.

D. Linstedt, K. Graziano, and H. Hultgren. *The Business Of Data Vault Modeling*. Lulu, Raleigh, North Carolina, 2011.

J. Liu, F. Huang, D. Ye, and T. Huang. Efficient consistent query answering based on attribute deletions. In *Proceedings of the International Symposium on Computer Science and its Applications*, CSA '08, pages 222–227. IEEE Computer Society, 2008.

S. Luján-Mora and M. Palomar. Reducing inconsistency in integrating data from different sources. In *International Database Engineering and Applications Symposium*, IDEAS '01, pages 209–218. IEEE Computer Society, 2001.

S. Luján-Mora, J. Trujillo, and I.-Y. Song. A UML profile for multidimensional modeling in data warehouses. *Data Knowledge and Engineering*, 59(3):725–769, 2006.

E. Malinowski and E. Zimányi. Hierarchies in a multidimensional model: From conceptual modeling to logical representation. *Data Knowledge and Engineering*, 59(2):348–377, November 2006.

H. Mannila and K. Räihä. Algorithms for inferring functional dependencies from relations. *Data Knowledge and Engineering*, 12(1):83–99, 1994.

B. Marshall, H. Chen, and T. Madhusudan. Matching knowledge elements in concept maps using a similarity flooding algorithm. *Decision Support Systems*, 42(3):1290–1306, 2006.

W. May. Information extraction and integration with FLORID: The MONDIAL case study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999. URL http://dbis.informatik.uni-goettingen.de/Mondial.

J.-N. Mazón and J. Trujillo. Enriching data warehouse dimension hierarchies by using semantic relations. In *23rd British National Conference on Databases*, volume 4042 of *LNCS*, pages 278–281. Springer-Verlag, 2006.

J.-N. Mazón, J. Lechtenbörger, and J. Trujillo. A survey on summarizability issues in multidimensional modeling. *Data Knowledge and Engineering*, 68(12):1452–1469, 2009.

S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm (extended technical report). Technical report, Stanford University, University of Leipzing, 2002a. URL http://ilpubs.stanford.edu:8090/497/1/2001-25.pdf.

S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of 18th International Conference on Data Engineering*, pages 117–118, 2002b.

D. Milano, M. Scannapieco, and T. Catarci. Structure-aware XML object identification. In *VLDB Workshop on Clean Databases (CleanDB)*, CleandDB '06, 2006.

J. Mundy, W. Thornwaite, and R. Kimball. *The Microsoft Data Warehouse Toolkit*. Wiley, 2006.

OLAP Council. OLAP council white paper, 1997. URL http://www.olapcouncil.org/research/whtpaply.htm.

B. O'Neil. The information model and the data vault, 2004. URL http://www.tdan.com/view-articles/5221.

Oracle. Oracle database SQL reference, 2005. URL http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_5006.htm.

Q. H. Pan, F. Hadzic, and T. S. Dillon. Discovering concept mappings by similarity propagation among substructures. In *International Database Engineering and Applications Symposium*, volume 6283 of *LNCS*, pages 324–333. Springer-Verlag, 2010.

S. Pavel and J. Euzenat. A survey of schema-based matching approaches. Technical Report DIT-04-087, Informaticae Telecomunicazioni, University of Trento, 2004. URL http://disi.unitn.it/~p2p/RelatedWork/Matching/JoDS-IV-2005_SurveyMatching-SE.pdf.

T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending practical pre-aggregation in on-line analytical processing. In *Proceedings of 25th International Conference on Very Large Data Bases*, VLDB '99, pages 663–674. Morgan Kaufmann Publishers Inc., 1999.

J. Pokorny. Modelling stars using XML. In *4th ACM International Workshop on Data Warehousing and OLAP*, DOLAP '01, pages 24–31, 2001.

S. Puhlman. Fehlerklassen in XML daten und erzeugung schmutziger XML daten (Error classes in XML data and generating dirty XML data). Technical report, Umboldt University of Berlin, 2004.

M. Rafanelli. Operators for multidimensional aggregate data. In M. Rafanelli, editor, *Multidimensional Databases*, pages 116–165. IGI Publishing, 2003.

M. Rafanelli and A. Shoshani. STORM: A statistical object representation model. In *Scientific and Statistical Database Management*, SSDBM'90, pages 14–29, 1990.

E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The Very Large Databases Journal*, 10(4):334–350, December 2001.

D. Riazati and J. A. Thom. Matching star schemas. In *Proceedings of the 22nd International Conference on Database and Expert Systems Applications - Volume Part II*, volume 6861 of *LNCS*, pages 428–438. Springer-Verlag, 2011.

D. Riazati, J. A. Thom, and X. Zhang. Drill across & visualization of cubes with non-conformed dimensions. In *Proceedings of the 19th Conference on Australasian Database*, volume 75 of *ADC '08*, pages 97–105. Australian Computer Society, Inc., 2008.

D. Riazati, J. A. Thom, and X. Zhang. Inferring aggregation hierarchies for integration of data marts. In *Proceedings of the 21st International Conference on Database and Expert Systems Applications: Part II*, volume 6262 of *LNCS*, pages 96–110. Springer-Verlag, 2010.

D. Riazati, J. A. Thom, and X. Zhang. Enforcing strictness in integration of dimensions: Beyond instance matching. In *Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP*, DOLAP '11, pages 9–16. ACM, 2011.

O. Romero, D. Calvanese, A. Abelló, and M. Rodríguez-Muro. Discovering functional dependencies for multidimensional design. In *Proceeding of the ACM 12th International Workshop on Data Warehousing and OLAP*, DOLAP '09, pages 1–8. ACM, 2009.

H. Samet. *The Design and Analysis of Spacial Data Structures*. Addison Wesley, 1990.

Satama. Different between knowledge and information, 2012. URL http://www.nairaland.com/702795/different-between-knowledge-information.

H. Shefrin. Ending the management illusion: Preventing another finnacial crisis, 2009. URL http://www.iveybusinessjournal.com/topics/innovation/ending-the-management-illusion-preventing-another-financial-crisis.

H. L. A. Shoshani. Summarizability in OLAP and statistical data bases. In *9th International Conference on Scientific and Statistical Database Management*, SSDBM 97, pages 132–143. IEEE Computer Society, 1997.

Y. Sismanis, L. Wang, A. Fuxman, P. J. Haas, and B. Reinwald. Resolution-aware query answering for business intelligence. In *25th International Conference on Data Engineering*, ICDE '09, pages 976–987. IEEE Computer Society, 2009.

I. Y. Song, R. Khare, and B. Dai. Samstar: A semi-automated lexical method for generating star schemas from an entity-relationship diagram. In *Proceedings of the ACM 10th International Workshop on Data Warehousing and OLAP*, DOLAP '07, pages 9–16. ACM, 2007.

C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional databases. *Communications of the ACM*, 51(11):75–84, November 2008.

R. Torlone. Two approaches to the integration of heterogeneous data warehouses. *Distributed Parallel Databases*, 23(1):69–97, 2008.

S. Vinnik and F. Mansmann. From analysis to interactive exploration: Building visual hierarchies from OLAP cubes. In *Proceedings of 10th International Conference on Extending Database Technology*, EDBT '2006, pages 496–514, 2006.

W3C. OWL web ontology language overview, 2009. URL http://www.w3.org/TR/owl-features/.

M. Weis and F. Naumann. Detecting duplicate objects in XML documents. In *Information Quality in Informational Systems*, IQIS '04, pages 10–19. ACM, 2004.

M. Weis and F. Naumann. Dogmatix tracks down duplicates in XML. In *Proceedings of the 2005 ACM SIGMOD International*, SIGMOD'05, pages 431–442. ACM, 2005.

J. Wijsen. Project-join-repair: An approach to consistent query answering under functional dependencies. In *Flexible Query Answering Systems*, volume 4027 of *LNCS*, pages 1–12. Springer-Verlag, 2006.