A Practical Approach To Merging Multidimensional Data Models

by

Michael Mireku Kwakye

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfilment of the requirements
For the Masters of Science degree in
Computer Science at the
Ottawa-Carleton Institute for Computer Science

uOttawa

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Schema merging is the process of incorporating data models into an integrated, consistent schema from which query solutions satisfying all incorporated models can be derived. The efficiency of such a process is reliant on the effective semantic representation of the chosen data models, as well as the mapping relationships between the elements of the source data models.

Consider a scenario where, as a result of company mergers or acquisitions, a number of related, but possible disparate data marts need to be integrated into a global data warehouse. The ability to retrieve data across these disparate, but related, data marts poses an important challenge. Intuitively, forming an all-inclusive data warehouse includes the tedious tasks of identifying related fact and dimension table attributes, as well as the design of a schema merge algorithm for the integration. Additionally, the evaluation of the combined set of correct answers to queries, likely to be independently posed to such data marts, becomes difficult to achieve.

Model management refers to a high-level, abstract programming language designed to efficiently manipulate schemas and mappings. Particularly, model management operations such as *match*, *compose mappings*, *apply functions* and *merge*, offer a way to handle the above-mentioned data integration problem within the domain of data warehousing.

In this research, we introduce a methodology for the integration of star schema source data marts into a single consolidated data warehouse based on model management. In our methodology, we discuss the development of three (3) main streamlined steps to facilitate the generation of a global data warehouse. That is, we adopt techniques for deriving attribute correspondences, and for schema mapping discovery. Finally, we formulate and design a merge algorithm, based on multidimensional star schemas; which is primarily the core contribution of this research. Our approach focuses on delivering a polynomial time solution needed for the expected volume of data and its associated large-scale query processing.

The experimental evaluation shows that an integrated schema, alongside instance data, can be derived based on the type of mappings adopted in the mapping discovery step. The adoption of Global-And-Local-As-View (GLAV) mapping models delivered a *maximally-contained* or *exact* representation of all fact and dimensional instance data tuples needed in query processing on the integrated data warehouse. Additionally, different forms of conflicts, such as semantic conflicts for related or unrelated dimension entities, and descriptive conflicts for differing attribute data types, were encountered and resolved in the developed solution. Finally, this research has highlighted some critical and inherent issues regarding functional dependencies in mapping models, integrity constraints at the source data marts,

and multi-valued dimension attributes. These issues were encountered during the integration of the source data marts, as it has been the case of evaluating the queries processed on the merged data warehouse as against that on the independent data marts.

Acknowledgements

I greatly thank God for helping me come this far in my education. His grace, favour and faithfulness have been unceasing in my entire life.

I also express my profound appreciation to my supervisors - Dr. Iluju C. Kiringa and Dr. Herna L. Viktor - who have been pivotal in exposing and capacitating me in the different aspects of computer science and research at the graduate level. I thank Dr. Kiringa for his untiring support, careful supervision, and guidance in my overall research. His patience in introducing me to the theoretical aspects of computer science, and helping me to address my deficient areas of formal languages and computational complexities cannot go unmentioned. I express my sincere gratitude to Dr. Viktor for her insightful discussions, discerning feedback and unwavering support in my graduate studies. Her ardent intuition in teaching me the basics of data warehousing and dimensional modelling, as well as research documentation is indelible. I sincerely acknowledge the financial support I received from the NSERC Stategic Network on Business Intelligence (BI) through my supervisors.

I cannot forget my research lab mates - Dela De Youngster, Daniel Antwi, Sepideh Ghanavati, Fatemeh Nargesian, Mana Azarm, Mohammed Al Shammeri, Salvador Valencia Rodriguez - and others who have been very informative and helpful in discussions.

Finally, I thank my mum and siblings, as well as other family and friends for their love, encouragement and caring support.

Many thanks to all others who helped in varied ways.

Dedication

To the memory of Martin Yaw Kwakye Addo.

Daddy, may your soul be soothed in your eternal rest by this academic accomplishment.

Contents

List of Tables

List of Figures

List of Algorithms

Part I

INTRODUCTION

Chapter 1

Introduction

The concept of schema merging is important in databases as it has both academic and industrial implications. Schema merging involves integrating disparate models of related data using methods of element matching, mapping discovery, schema merging, and consolidation. These procedures, as well as the identification of prime meta-models and the articulation of semantic representation of the meta-models, make the overall procedures of data and schema integration very difficult.

Most of the procedures that go into schema merging, have been focused on traditionally identifying the independent data sources and the associated mapping correspondence of its elements to the elements of other integrating data sources. Further processes involve the development of transformations for mapping relationships and the combination of the elements from different data sources to form a global mediated schema. Recent studies have focused on the inference of semantic meaning of the elements of the data sources in integration [90].

Data integration, as defined by *Lenzerini* in [55], is the problem of combining data residing at different sources, and providing the user with a unified view of these data. Most of the processes that go into generating the final output of data integration stem from the fundamental operations of model management [9]. Model management in the field of databases refers to a high-level, abstract programming language designed to efficiently manipulate *schemas* and *mappings*. It is therefore, a generic approach to solving problems of data programmability and heterogeneity where concise and clear-cut mappings are manipulated to deliver desired output of an engine that supports robust operations related to certain metadata-oriented problems [9], [8]. Some of these operations are to *match schemas*, *compose mappings*, *difference schemas*, *merge schemas*, *apply function*, *translate schemas* into different data models, and *generate data transformations* from mappings.

The main abstractions that are needed in expressing model management operations are

schemas and mappings, of which the choice of a *language* to express these schemas and mappings is vital. A *model* is described in [9], as a formal description of a complex application artefact such as database schema, an application interface, a Unified Modelling Language (UML) model, or an ontology. A *schema* is an expression that defines a set of possible instances, for example, database states, and a *meta-model* is the language needed to express the schemas. These schemas could be Structured Query Language (SQL), Extensible Markup Language (XML) Schema, Web Ontology Language (OWL), or Multidimensional Schema.

There have been varied applications of model management which include data management, e-commerce, object-to-relational wrappers, enterprise information integration, report generators, database portals, and data integration [9, 11]. The application area of data integration is evident in various domains. For instance, in the scientific domain where research results from different bioinformatics repositories are combined, data integration makes the analyses and knowledge discovery process of these results much more important [90]. In the financial services domain, for example, banking, insurance, investments or credit risk assessment, the need of data integration in processes cannot be overemphasized. On the one hand, data from different departments are summarized and then combined to form a uniform material for reporting. On the other hand, data from different subsidiaries of a company, or different companies coming together in a merger or acquisition will need to be consolidated into a uniform fashion, so as to depict the true representation of each of the underlying data sources from the different subsidiaries or companies. In the healthcare domain, data integration is also vital in the sense that, the history data of patients - in line with their diverse diagnoses - from different departments or healthcare centres are combined together to give an informed overview of the data on patients. This enables better healthcare reporting and analytics on the part of healthcare administrators.

A typical case of model management application in the area of data integration is *data warehouses*. Data warehouses are defined as a collection of information storage data derived from disparate operational and/or transactional data sources, and transformed into a central repository for analysis and decision-support within an organization.

In this research, we introduce a novel methodology for schema merging where apply model management operations into generating a single consolidated star schema data warehouse from multidimensional star schema data marts. Based on the literature reviews we conducted in Chapter 3, this problem of data integration has received very little attention. In our approach, we combine various data marts to form a uniform data warehouse capable of providing *exact* or *maximally-contained* answers to solutions as it were posed to the independent data marts. We choose multidimensional star schemas where we consider issues of

integration in terms of schema matching, mapping discovery and the merge algorithm.

1.1 Problem Definition

The procedural steps in delivering a data warehouse for an entire organization leads to the production of snippets of disparate data marts or "stovepipes" at scheduled times which are independent, but related to one another in some semantic form. The need to retrieve a full data set across these disparate snippets of data marts highlights a drawback in the existence of the independent scattered data marts in the organization.

Furthermore, the dynamics of company mergers and acquisitions prevalent in the business world today presents the consequent need of pulling required information across these data marts, in addition to performing analysis or decision support in relation to these scattered data marts.

There is, therefore, the need to incorporate all these multidimensional star schemas into a single global data warehouse, without resorting to the independent multidimensional star schemas for query processing. In our approach, we want to integrate of these independent, but related, multidimensional star schemas into a data warehouse from which all intended answers to queries can be derived from without resorting to any source data mart. This will enable a uniform medium where efficient data analysis can be conducted on the underlying data, and avoid the tedious task of comparing data multiple media.

1.2 Motivation and Research Objective

Past studies on model management and its operations have tried to highlight engineered ways of addressing information processing problems pertaining to data models [47, 9, 62]. In trying to offer users that flexibility and efficiency in data processing, model management operations in the form of *schema matching*, *schema mappings*, *schema merging*, amongst others, have been generally attempted by *Melnik* in [63], *Bernstein et al.* in [8], and lately by *Gubanov et al.* in [35].

To efficiently integrate different data sources, the model management *match* operation most expectedly serves as basis to other major operations [9]. *Schema matching* is a fundamental operation in the manipulation of schema information, which takes two schemas as input and produces a mapping between elements of the two schemas that correspond semantically to each other [78]. Various surveys and studies have been conducted in [78, 87, 31, 86] in this direction of schema matching of which incremental and new results have been used

to effectively deliver mapping correspondences. Out of these studies and surveys conducted, some concrete results some of which are tailored to a specific domain have been developed to produce very high precisions. Some of these algorithms are *Similarity Flooding* (SF) in [66], *COMA* in [26], *Cupid* in [59], *SEMINT* in [56], *iMAP* in [24], and the *Clio* Project in [42, 68].

Some of these other algorithms have been represented in one form or the other in industrial or commercial products such as in [45, 37] where a business user can combine and tweak the set of algorithms to generate expected outcomes. It will be noted that schema matching operations continue to be enhanced from fields such as *Knowledge Representation* [40], *Machine Learning* [56, 5], and *Natural Language Processing* [48], where techniques are used to deliver near-automatic and semantically correct solutions.

Another operation of model management, that is fundamental in delivering an efficient integration procedure, is in the form of *compose mappings*. This operation is normally an outgrowth of a schema matching operation and therefore evaluates better when the preceding schema matching operation is accurate and precise. *Schema mapping* is the fundamental operation in metadata management that takes as input elements of instances from a source and target schemas and produces a semantic relationship between these associated elements [52, 43, 42, 44, 28, 51]. Recent studies conducted in generating schema mappings have shown that the strength of mapping relationships that exists between schema elements largely goes to determine how best the overall data integration procedure will be. It therefore follows that the schema mapping step is an integral component of a formalized *data integration system*, I, defined by *Lenzerini* in [55] as a triple $I =< G, S, M >$ where G, is the *Global Schema*, S, are the *Sources Schemas*, and M is the *Mapping*.

Kensche et al. in [52, 51] define that an *extensional mapping* can be represented as two queries which are related by some operator (such as equivalent or subset), which can be expressed as Local-As-View (LAV), Global-As-View (GAV), Source-To-Target Tuple Generating Dependencies (S-T tgds), Second-Order Tuple Generating Dependencies (SO tgds), or similar formalisms. The first two (2) approaches are chosen as a basic form of specifying mappings in our context of data integration for multidimensional schemas. More intuitively, a hybrid approach of the LAV and GAV mappings termed as Global-and-Local-As-View (GLAV) mappings which has been formalized to merit on the strengths of both mappings, and suppressing of the weakness of both mappings; has received much studies and has been generally accepted to deliver efficient and expressive mapping relationship between schema elements. In our research work on data integration, we make use of the GLAV mappings which has been enhanced by *Hernández et al.* in [42, 43, 44] and being implemented in [45].

The final model management operation adopted in our line of research and which has been handled domain-wise in different ways is the *merge* operation, expressed as *schema merging*. Schema merging is the operation which takes as input two (2) meta-models and a set of mapping correspondences, and produces as output a merged meta-model capable of representing all the elements and semantics of the input meta-models. In the generic sense, a number of studies have been conducted and some results are highlighted in [74, 77, 63]. In the area of data warehousing, some work has been done by *Bernstein and Rahm* in [11], *Pottinger* in [73], and by *Calvanese et al.* in [17]. Additionally, *Pottinger and Bernstein* in [75] attempted to derive some results on schema merging in relation to relational data sources, while merging based on semantic mappings have also been addressed by the authors in [81].

Schema merging is supposed to be the summit of the overall data integration process, where the outputs of other preceding processes are utilized. As part of the merging process, various architectures and algorithms are adopted to form a uniform platform for users to access the underlying data sources. A typical architecture of a merge system as denoted by *Calvanese et al.* in [17] is described in terms of two (2) types of modules: *Wrappers* and *Mediators*. In terms of algorithms for merging, *Pottinger and Bernstein* in [75] have proposed an algorithm for relational sources that works on a Mediated Schema Normal Form (MSNF), and conjunctive queries and mappings. For generic merging as in [74, 77, 63], the algorithms proposed tend to present a procedure independent of the domain of the metadata model, and additionally a proposition of some requirements that the merged data model must satisfy and also an exposition of some likely conflicts and their resolution measures.

In this research, we introduce a new merge algorithm which subsumes the prior work of *Batini et al.* in [3], *Buneman et al.* in [13], and *Pottinger and Bernstein* in [74]. Our method is explained further in Chapters 4 and 5. Much more specifically, we draw on some of the significant propositions by *Pottinger and Bernstein* in [74] and extend it in formalizing our algorithm as a more practical solution for multidimensional data models.

In arriving at a motivation for this research, the work of *Rizzolo et al.* in [82] present a background activity to incorporate our process of integration of multidimensional data models into their Framework II of the Conceptual Integration Model (CIM), as depicted in Figure 1.1.

Our research seeks to deliver a solution in a streamlined approach in which the source data marts have been modelled as star schemas. This solution will then offer a single consolidated star schema data warehouse into their next stage of their framework.

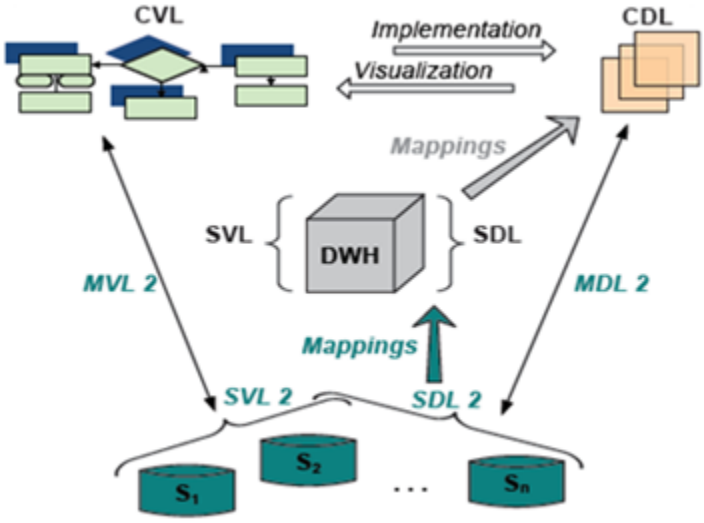

Figure 1.1: *Conceptual Integration Model (CIM) Proposed Framework II*

1.3 Thesis Goals and Assumptions

As discussed earlier in the previous subsection, this thesis introduces a novel approach to deliver a data integration solution for multidimensional data models. Much more specifically, we focus on *star schemas* as the modelling approach for the underlying data sources. We choose star schemas because of the following; *One*, this form of dimensional modelling presents the simplest form of models, in which some of the complexities associated with managing data as relating to snowflaking are avoided. *Two*, it also offers a platform for effective query processing, as compared to Snowflake and Fact Constellation schemas. This feature makes this form of modelling much more preferred in most organizational data marts, where a higher rate of query processing is sought for.

We present a solution where business users are presented with a single medium of a global data warehouse for query processing. The merits of our methodology are to:

1. Eliminate redundant dimensions and/or attributes across all integrating data marts after the merge procedure; and

2. Offer an integrated and efficient medium of query processing for the expected volume of data;

A summary of some other assumptions needed to validate the success of this research thesis is enumerated as follows: *Firstly*, the existence of *one-to-one* mappings and possible

one-to-many mappings between the multidimensional schema and the instance data values. *Secondly*, the existence of *quality* and *clean* data at the independent multidimensional star schemas, i.e. free of *inconsistencies* and *noisy* data. The presence of dirty data and schema structural defects inherent in the star schema multidimensional star schemas tend affect the generation of correct mapping correspondences and the discovery of efficient mapping models. This will later affect the output generated from of the merge algorithm. Finally, the expectation of queries and their solutions from the global data warehouse, which are *maximally-contained* or *exact* to that when expressed on the independent multidimensional star schemas. Maximally-contained query solutions are expected in some cases because of the existence of similarity mapping correspondences between different attributes in related dimension or fact tables.

1.4 Thesis Contributions

As part of outlining our novel methodology for integration, we itemize our main contributions in this thesis as follows:

1. We formulate and design a merge algorithm to integrate the multidimensional star schemas. This algorithm accepts as inputs, the Fact and Dimension tables of the multidimensional star schemas, a modelled GLAV mapping formalisms, and a set of predefined attribute descriptions.

2. We specify and describe a set of qualitative technical requirements that ensures the validation and correctness of the formulated merge algorithm. These requirements are to ensure the generation of tuples that satisfy the correct answers to posed queries.

3. We outline and describe of some possible conflicts that arises when merging multidimensional star schemas. The resolutions of these conflicts are also explained in each of the contexts expressed.

4. We highlight some open issues that are encountered during integration of multidimensional schema models. These issues are discussed as; multi-cardinality relationships that exist between the schema structure of the multidimensional star schemas and the instance data, and the presence and likely effect of integrity constraints on the multidimensional star schemas.

1.5 Thesis Outline

This thesis is organized into seven (7) chapters and the remainder of the chapters are described as follows. Chapter 2 presents a detailed overview of the major procedures that go into a data integration system. It outlines an exposition of current studies in line with the concept of data warehousing and the various techniques. Chapter 3 discusses the concept of data integration and details regarding *schema matching* approaches, *schema mapping* discoveries and all its flavours, *schema merge* algorithms, as well as integration for data marts. Chapter 4 presents our approach of data integration and an overview of the techniques adopted in our schema matching procedure. The chapter also details the proposed mapping discovery procedure and a discussion of the proposed multidimensional schema merge algorithm.

In Chapter 5, we present a summary of the implementation and experimental setup, a description of the data sets used, as well as the procedural steps that are involved in various phases of the research project. In Chapter 6, we present an evaluation analysis of the results out of the implementation procedures, where we explain the criteria in terms of; correctness of the data values, dimensionality hierarchy, rate of query processing, and Slowing Changing Dimensions. In Chapter 7, we conclude by summarizing the contributions of the research conducted, and the vital areas of applications in academia or industry. We also reflect on some of the consequent open issues and likely areas of future work.

Part II

LITERATURE REVIEW

Chapter 2

Data Warehousing

The dynamics of organizational needs from various stakeholders such as customers, management and other business users continue to pose varied challenges to decision-making applications which are supported by data warehouse repositories [54, 79].

This Chapter presents a general overview of the processes involved in dimensional modelling and in the overall development of data warehouses. We discuss a background to data warehousing in Section 2.1, and highlight some of the approaches of modelling multidimensional schemas in Section 2.2. We then finally summarize the discussion in Section 2.3.

2.1 Background To Data Warehousing

Data Warehouse (DW) are necessary to the success of an organization as most companies anticipate its benefits and are now devoting more attention and resources to the design and development. The functionality of data warehouses to provide efficient decision making analysis have now been extended into the development of *Business Intelligence* (BI) Systems [22, 38].

The basic lifecycle of a DW, as defined by *Kimball et al.* in [54, 53] and by *Dell'Aquila et al.* in [22], is displayed in Figure 2.1. It comprises of the following: *Analysis and Business Requirements Definition*, *Conceptual Design and Dimensional Modelling*, *Physical Design and Implementation*, *ETL Design and Development*, *Deployment and Refreshing*, and *BI Applications and Development*. A brief description of these procedures is explained in the next subsections.

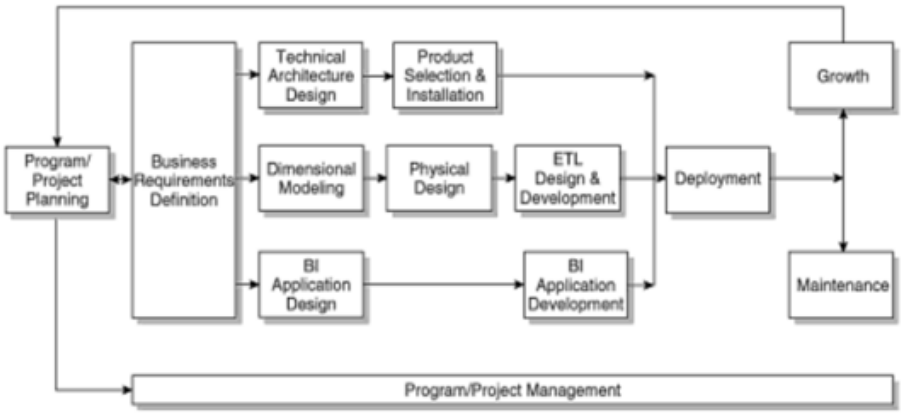

Figure 2.1: *The Kimball Data Warehouse Lifecycle* [54]

2.1.1 Analysis and Business Requirements Definition

The initial step in the development of a DW/BI system is the need to conduct a thorough analysis and business requirements, as the likelihood of the success is greatly increased by a sound understanding of the business users and their requirements [32]. A general methodological approach demands that the DW/BI architect must first define a preliminary workload that consists of a set of frequent queries that are the key factors driving the business. This step of the overall project management of the DW design occurs at two (2) distinct levels. The first level is at a *micro level*, where the DW/BI architect needs to understand the businesss needs and priorities relative to the program perspective. The second level is at the *macro level*, where the DW/BI architect enquires into the business users needs in the context of a streamlined system [54].

2.1.2 Conceptual Design and Dimensional Modelling

This next step, after the initial step of requirement gathering, focuses on the design of the target dimensional model based on the user requirements. This step centres on the design of a logical model to support businesss reporting and analytical needs. The dimensional modelling process divides the organizational entity data into *measurements* (facts) and *contexts* (dimensions). Particularly, the useful identification of all measurements which are necessary in producing business information and all its well-structured hierarchies have to be streamlined to deliver data aggregation queries. A four (4) dimensional design process stipulated by *Kimball et al.* in [54] is catalogued as follows; *choose the business process*, *declare the grain*, *identify the dimensions*, and *identify the facts*.

2.1.3 Physical Design and Implementation

The physical design phase focuses on defining the physical structures, which incorporates the setting up of the database environment and instituting appropriate security. This phase involves the implementation of the logical conceptual (dimensional) model, represented in ROLAP or MOLAP technology, and supported by the DBMS. Additional issues that have to be considered during this implementation phase are the need to address preliminary performance tuning strategies in line with *indexing*, *partitioning*, *aggregations*, *tablespaces*, and *disk layout*. Some of these tasks are continuously tweaked throughout the overall lifecycle to offer a continual upward performance for the DW.

2.1.4 ETL Design and Development

The ETL design phase presents the bulk of the tasks involved in the developmental lifecycle of the DW. The ETL architecture system - which produces a plan to feed and to periodically update the DW - is made up of a comprehensive set of subsystems which work together to provide an extraction, cleansing and conforming, delivery and management capabilities. These subsystems together make the ETL architecture system the foundation of the DW/BI project and as result its success helps in determining the overall success of the data warehouse. In line with automation processes for ETL, *Jörg and Dessloch* in [49] present an approach for an automated derivation of incremental load jobs based on equational reasoning. This and other related studies aim to offer a semi-automatic or fully automatic system platform for ETL. The ETL system also presents a virtual view of data integration in line with our approach in this research.

2.1.5 Deployment and Refreshing

The deployment and refreshing phase outlines an overview of convoluted tasks which are directed at technology, data, and BI applications. It also integrates the execution of the ETL repeated at regular intervals, and testing procedures such as system testing, data quality assurance testing, operations process testing, live testing, performance testing, and usability testing, amongst others. Other deployment procedures include database deployment, and report deployment. Some issues of documentation, training and overall administration are also looked at this phase of the data warehouse development.

2.1.6 BI Applications and Development

The BI Applications and Development step provides a platform for intertwining the back-end work of the data warehouse and the front-end work of BI applications usage by business users. These BI applications offer business users the medium to address their needs and capabilities in the form of appropriate navigation interfaces and parameter-driven analytical reporting. Other tasks of application development include configuring the business metadata and tool infrastructure, construction and validation of analytic and operational BI applications.

2.2 Dimensional Modelling

As stated in Section 2.1, an important phase of DW design is *dimensional modelling* 2.1.2, where the conceptual and logical design is formulated. The authors in [54] define dimensional modelling as a logical design technique for structuring data so that it is intuitive to business users and delivers fast query performance. An organizational entity data is segregated into two (2) forms - *measurements* and *context* - based on their *content* and their ability to infer on semantics.

Measurements portray an organizations business process in line with transactions that are processed in the OLTP. They are usually numeric values and are referred to as *facts*. The *contexts*, on the other hand, are the independent perspectives which surround the facts and give meaning to the numeric values. They are referred as *dimensions* and are normally represented in textual forms. The *dimensions* describe the who, what, when, where, why, and how context of the *measurement* (fact) [54]. Some of the main merits and propositions for dimensional modelling are *understandability of data*, *query performance*, and the *graceful accommodation of unexpected new data*; just to mention a few.

The end-product of a dimensional modelling is a *multidimensional data model* which can be implemented as a ROLAP, MOLAP, or the recent hybrid form of HOLAP. A multidimensional data model forms the building blocks for a DW and enables the data to be viewed in terms of a *cube* [41]. *Han and Kamber* in [41] define a *data cube* as a framework that allows data to be modelled and viewed in multiple *n-dimensions*. A data cube can be view from different dimensions which can represent different degree of summarization or aggregation of *facts* for semantic analysis.

Depending on the type of modelling approach adopted, which can be inferred in the manner in which the *dimensions* are made to associate each *fact* in the multidimensional data model paradigm, different forms of schema can be modelled to facilitate this concept of DW modelling. The four (4) main types of schemas that are generally employed in data

warehousing are *Star*, *Snowflake*, *Fact Constellation*, and *Data Vault*.

2.2.1 Star Schema

This schema type illustrates a large central table (fact table) which contains the bulk of the data and contains no redundant data, and a set of smaller attendant table (dimension tables), one for each dimension with a lot of redundant attribute data [41, 69]. It is the most common and simplest schema modelling with the graphical schema outline showing a starburst, with the dimension tables displayed in a radial pattern around the central fact table. The star schema model offers a prototype where queries are never complex as a result of the schema joins and conditions involving a fact table and a single level of dimension tables. In this architecture, there exist only direct dependencies from the dimensions to the fact tables and no existence of any *normalized* dimensions.

2.2.2 Snowflake Schema

This schema is represented by a centralized fact table which is connected to multiple dimension tables either directly or indirectly; with most of the dimension tables *normalized* into multiple related tables. This presents the complex snowflake shape with the dimensions more elaborate, having multiple levels of relationships, and the *child* tables have multiple *parent* tables. The schema type offers a merit where a redundancy in a dimension table is eliminated and offers an ease to maintain and saves disk storage space. On the other hand, the snowflake model structure demerits on the effectiveness of query processing where since more joins will be needed to execute a single query. Additionally, this *snowflaking* effect in this model affects query processing much more with the data attributes in the dimension tables but not with the fact table.

2.2.3 Fact Constellation Schema

This schema model displays an architecture that shows multiple fact tables *sharing* many dimension tables. This architecture of dimension modelling is much more complex to construct and handle, and exposes some critical shortcomings; as many variants for particular kinds of aggregation must be considered and selected. Moreover, the dimension tables associated with these convoluted set of fact tables are also large in size. This makes the schema model an undesirable one.

2.2.4 Data Vault Schema

This is a method of modelling DW where there is a detailed oriented, historical tracking and uniquely linked set of normalized tables that support one or more functional areas of business. It is the next generation of an evolving dimensional modelling and a hybrid approach which encompasses the best of breed between *3rd Normal Form* (3NF) and *star schema*. The design is flexible, scalable, consistent and adaptable to the needs of the enterprise [57]. The schema is designed to avoid or minimize the impact of issues that deal with changes in the systems feeding the DW and for cases of conformed dimensions, where the data have to be cleansed before loading during ETL, in conformance to the enterprise databus architecture. This form of modelling is therefore patterned as a neural network with simplistic view of *neurons*, *dendrites*, and *synapses* – where neurons are associated with *Hubs* and *Hub Satellites*.

2.2.5 Discussion of Dimensional Modelling Approaches

In this subsection, we discuss the various approaches of dimensional modelling. We compare their ability to offer a sound repository base and as a platform for analytical reporting and decision-making tool in an organization. A summary of the discussions is described in *Table* 2.1.

2.2.6 Data Marts and Data Warehouses

In data warehousing architectures, two (2) forms of deliverables are presented as the final product; namely, data warehouse and data marts.

A *Data warehouse* collects and stores data regarding the entire organization or company with its data and query processing from an *enterprise-wide* viewpoint. A *Data mart*, on the other hand, is a *departmental-wide* and always a subset of the data warehouse; that focuses and is oriented on a particular domain or business line of the organization. Data marts are developed based on the merits such as easy access to frequently needed data, improves business user query time, lower cost of implementation, amongst others.

2.3 Summary

In this chapter, we presented a general overview of data warehousing and we introduced basic approaches for dimensional modelling. We first discussed the various steps involved in the methodology of generating a data warehouse. In the later pages of the chapter, we discussed

Table 2.1: *Summarized Comparison of Dimensional Modelling Approaches*

Criterion / Modelling Approach	**Star Schema**	**Snowflake Schema**	**Fact Constellation Schema**	**Data Vault Schema**
Type of Architecture	Simple model with a centralized Fact Table connected directly by multiple Dimension Tables	A complex model with a centralized Fact Table connected directly or indirectly by multiple Dimension Tables	A complex model with multiple Fact Tables each connected directly or indirectly by shared Dimension Tables	A hybrid model of a breed between *3NF* and star schema model structure
Normalized Dimensions	No - Does not allow normalized dimensions	Yes - Allows any level of normalization in the dimensions	Yes - Allows normalization to an appreciable level in the dimensions as may be required by the mode of sharing	Yes - Allows normalization in the dimensions to the 3rd Normal Form
Rate of Query Processing	Offers the best and fast model for query processing	Experiences a reduction in the effectiveness of query processing as a result of more joins	Query processing is affected by volume of normalization and sharing between the dimensions	Experiences some form of reduction in the rate of query processing, but better than other complex Snowflake or Fact Constellation
Presence of Multiple Fact Tables	No - Does not allow multiple Fact Tables	No - Does not allow multiple Fact Tables	Yes - Could have one or more Fact Tables being connected by shared Dimension Tables	Yes - Allows multiple Fact Tables because of its adaptability for different operation systems
Adaptation to Operational Systems	Flexible and most scalable to operational system changes	A bit rigid to changes coming from operational systems because of different levels of normalization	Experiences a fair complexity in the changes coming from operational systems	Flexible, scalable consistent and most adaptable to changes coming from operational systems

the various approaches of *star*, *snowflake*, *fact constellation*, and *data vault schemas* that can be adopted in the modelling of multidimensional schemas. We compared the strengths and weakness of each of these approaches of modelling in terms of the query processing expected, the redundancy level expected in the dimension, the adaptation to changes in the operational systems, amongst others.

In the next chapter, we address the concept of data integration. We discuss each of the steps for schema matching, mapping models discovery, and schema merging that are involved in integration. We first compare the various approaches for each step. We also discuss some studies that have been conducted in the area of data integration for data marts.

Chapter 3

Data Integration

The concept of data integration has been studied by many research groups and from different perspectives. We discuss *schema matching* procedures in Section 3.1, *schema mapping* discovery procedures in Section 3.2, and *schema merge* procedures in Section 3.3. In Section 3.4, we examine some other related data integration work for data marts. We study the work by *Cabibbo and Torlone* in [16, 15, 14] and *Riazati et al.* in [80], similar to our approach of multidimensional data models (data marts). We carefully expound on some critical areas of their work, and how our work differs from theirs and comparatively efficiently answer the need for data integration for data marts. We finally summarize the discussion of this background work in Section 3.5.

3.1 Schema Matching

Different techniques of *schema matching* that have been studied so far may be categorized as *schema-level matchers*, *instance-level matchers*, and *hybrid* or *composite matchers*; with the last one being a combination of various matchers [78, 87, 10]. It can be inferred that the suitability of applying a set of *matcher(s)* to the set of data models would be based on the semantic schema information, instance data or model applicability. The hybrid or composite matchers are usually applied in cases where the schema- or instance-level matchers fail to deliver a good match result.

Rahm and Bernstein in [78] and *Shvaiko and Euzenat* in [87] state that the use of schema matching approaches are vital in many database application domains such as, schema integration, data warehouse, e-commerce, semantic query processing, P2P databases, and web services integration. These applications domains are dependent on, and become efficient, based on one technique or combination of techniques used.

3.1.1 The *Matching* Problem and General Techniques

Shvaiko and *Euzenat* in [87] describe a *matching element* as *Five-uple*, which establishes a correspondence between two (2) or more elements or entities. This matching element is defined in Equation 3.1.

$$< id, e, e', n, R > \tag{3.1}$$

where;

- id is a unique *Identifier* of a given matching element;
- e and e' are the *Entities* (table elements, properties) of the first and the second schema/ontology (e.g. fact or dimension tables), respectively;
- n is the *Confidence Measure* in some mathematical structure (typical in the $[0, 1]$ range) holding for the correspondence between the entities e and e';
- R is a *Relation* (e.g. *Equivalence*, *More General*, *Disjointness*, *Overlapping*) holding between the entities e and e'.

The authors in [78] further summarized the various schema matching approaches and classify the approaches, as illustrated in Figure 3.1.

Schema-level Matching

In terms of *schema-based* matching a consideration of the schema information is mainly used with available information of schema structure and properties of schema elements, such as *name*, *description*, *data type*, *constraints* [27, 70].

In this type of matching, the *granularity* or *level of matching* scales down to either *structure-level* or *element-level* matching. In the case of *element-level*, only elements in each of the schemas are observed, with elements at the finest level of granularity having the highest consideration. However, in the case of a *structure-level* matching, there is the reference to a combination of elements that appear together in a structure with a sought for precision of all components of the structures in the two schemas matching. Additionally, a known equivalence pattern or referential relationships from the data dictionary aid this form of matching. This arises in either a *full* or *partial* structural matching.

Another perspective of schema-level matching is the *cardinality* of the match, in which an element can participate in *zero*, *one*, or *many* mapping elements of the match result between two input schemas. Furthermore, *language-based* or *linguistic* matching which uses *names*

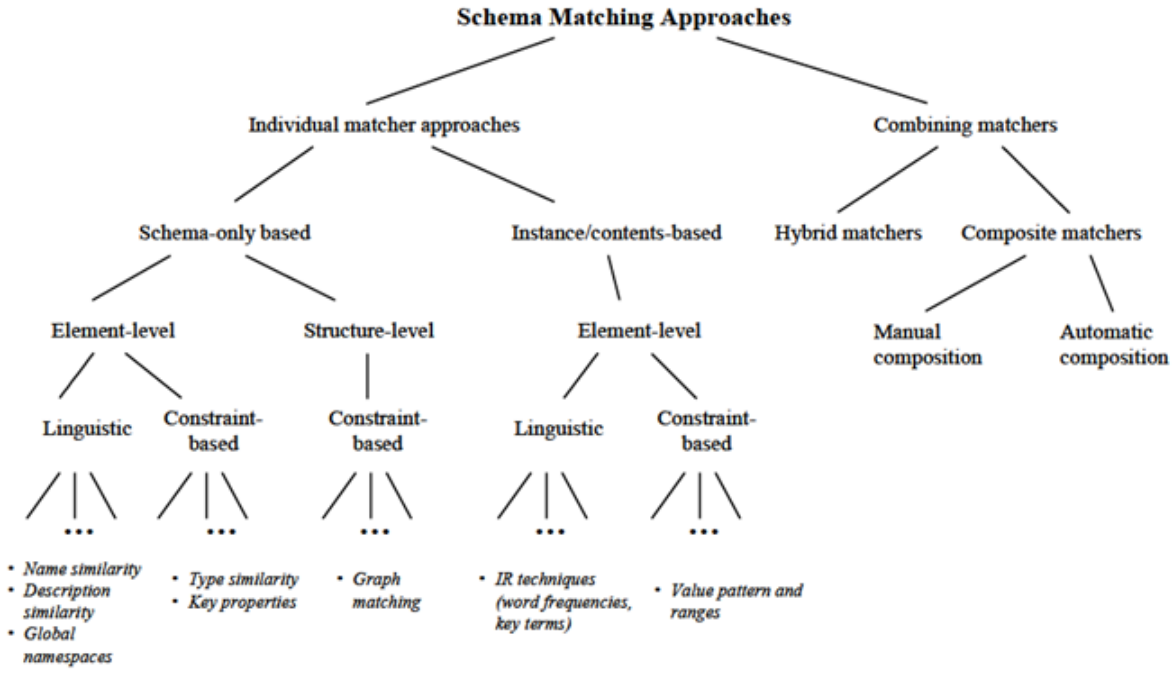

Figure 3.1: *Classification of Schema Matching Approaches* [78]

and *text* (that is, words or sentences) to semantically find similar schema elements can be applied. In [78], the authors state that instances of using a *name-based* schema-level matching could be in the equality of names, equality of canonical name representations, equality of synonyms, similarity based on common substrings, amongst others. *Name* matching can use domain- or enterprise-specific dictionaries containing common names and descriptions of schema elements, abbreviations to aid the similarity match.

The use of *constraint-based* approaches to schema matching is also one of the useful methods in finding correspondences. *Constraints* that define data types and value ranges, uniqueness, optionality, relationship types and cardinalities, amongst others, often serve to provide critical information which can be used by a matching algorithm to determine the similarity of schema elements. Using the constraint information of schemas can sometimes be misleading. That is, the result can sometimes generate imperfect match clusters, because of several other elements in a schema with comparable constraints. However, the approach could help limit the number of match candidates and may be combined with other matching algorithms for a desired perfection.

Instance-level Matching

In situations where schema information is limited or unavailable, instance-level data seems to provide important insight into the contents and meaning of schema elements. In contrast, cases where substantial schema information is available, the use of instance-level matching can be valuable to uncover incorrect interpretations of schema information, by sometimes choosing to match elements whose instances are more similar in an equally reasonable schema-level matches.

Some of the criteria used in evaluating this form of matching are value distribution, regular expression of data values, and similarity in data values. Consequently, other similarity matches can be deduced based on constraint-based characterization such as numerical value ranges and averages or character patterns which would allow recognizing phone numbers, postal codes, addresses, Social Security Numbers, dates, or money-related entries.

Hybrid or Composite Matching

The Hybrid or Composite matching presents another approach of deriving a good match result from different matching algorithms. It utilizes different information by combining several approaches. This type of matching would be most suitable for our specific domain of multidimensional data models. Hybrid matching algorithms determine match candidates based on multiple criteria or information sources. One method of applying such a hybrid approach is to generate a partial mapping with one algorithm and complete the mapping with another, as this offers a better performance of mapping models generation.

For example, in our *star schema* models in Figure 4.1, a hybrid approach of combined algorithms can be applied. Suppose we want to *match* the elements of the data marts - e.g., for the attributes of the fact tables - in both schemas S_1 and S_2, with the information of the data dictionary and value distributions offering good information for the existence of a better match. First, a schema-level *structural matching* can be applied which would distinctively or most-partially underline a similarity match between the policy fact tables in both star schemas S_1 and S_2, from other fact tables. Secondly, a constraint-based schema-level matching can be applied - using *data types* and *value ranges*, *uniqueness*, *relationship type* and *cardinalities* - which would offer best matches for almost all the attributes, and leaving a few ambiguities. Furthermore, on using an instance-level matching, considering the instance data *date entries* and *string character patterns* give a better picture of similarity matches.

A *composite matching* is implemented where the results of several independently processed

matchings are combined and presented as one single match result. A composite matching would allow for a set of modular *matchings* and would offer flexible ordering of matchings, so that each of them is executed simultaneously or sequentially. Hence, in the sequential mode the match result of a first matching is consumed and extended by a second matching to achieve an iterative improvement of the match result.

In summary, there have been quite a number of algorithm prototypes based on the techniques described above, and *Do et al.* in [25] carefully summarize the major generic ones; namely, *Similarity Flooding* (SF) in [66], *COMA* in [26], *Cupid* in [59], and *Clio* in [42]. The authors in [25] further state that these algorithms go a long way in delivering effective correspondence relationships between elements of schema models from different domains or ontologies. We discuss these algorithms in the following subsections.

3.1.2 *Similarity Flooding* (SF) Algorithm

The *Similarity Flooding* (SF) algorithm, by *Melnik et al.* in [66], for finding mapping correspondences proposes one of the generic methodologies for executing *schema matching* procedure. The algorithm - which works based on *fixpoint computation* - takes as input, schemas or other data model, which is transformed into graphs. It then produces as output a mapping corresponding between nodes of the graphs, being the entities in the schemas or data model.

This algorithm adopts a structural type of schema matching as discussed in Section 3.1.1 where there is the comparison of elements that appear together within a structure. A sequential execution of the procedural steps as outlined by the authors of this algorithm is enumerated in *Equation* 3.2, as follows:

$$\begin{aligned} &1.\quad G_1 = SQL2Graph(S_1); \quad G_2 = SQL2Graph(S_2); \\ &2.\quad initialMap \quad = \quad StringMatch(G_1, G_2); \\ &3.\quad product \quad = \quad SFJoin(G_1, G_2, initialMap); \\ &4.\quad result \quad = \quad SelectThreshold(product); \end{aligned} \tag{3.2}$$

The *first* step of the algorithmic procedure involves the translation of the schemas from their native formats into directed labelled graphs. Their approach of translating the schemas - in native format $ASCII$ files containing table definitions in $SQL\ DDL$ - into graphs, say G_1 and G_2, is done by using an import filter $SQL2Graph$ implemented in an *Open Information Model* (OIM) in [7].

The *second* step focuses on obtaining an initial mapping, coded *initialMap*, between the graphs G_1 and G_2 using an operator *StringMap*. This step involves an imprecise mapping

forming a correspondence of nodes from the graphs using a simple string matching that compares prefixes and suffixes of literal names based on their textual content.

In the *third* step, an operator *SFJoin*, is used to produce a refined mapping, coded *Product*, between the graphs G_1 and G_2. This step involves an iterative process based on a *fixpoint computation* to output a set of best possible similarity matches for elements from both set of graphs. In the final step, an operator *SelectThreshold* selects a subset of nodes pairs - from the third step output of *Product* - that corresponds to the most reasonable matching entry value.

3.1.3 *COMA* Matching System

The *COMA* schema matching system, by *Do et al.* in [26], is another kind of generic matching framework that provides a platform for combining different multiple *matchers*. It works in a much flexible way where the subsequent results from previous match operations are reused, in addition to several mechanisms that combine the results of *matcher* executions. This algorithmic platform also works on the idea of *structural matching* and makes use of a DBMS-based repository for storing schemas, intermediate similarity results of individual *matchers*, and a complete match result for later reuse.

The *first* step in the matching procedure of *COMA* is the representation of the schemas by rooted directed acyclic graphs where the schema elements are represented by graph nodes connected by directed links of different types. The *second* step of this schema matching system is the match processing phase. At this step, the translated elements of the schemas are made available to the set of match algorithms to be operated upon. The outcome of this process is a determination of a mapping indicating which elements of the input schemas logically correspond to each other. During this match processing step, one or more iterations are executed of which each iteration will consists of three (3) phases, namely; an optional user feedback phase, the execution of different matchers, and the combination of the individual match results.

In the *third* and final step, the concluding step in a match iteration leads to the derivation of combined match result from the individual match results stored in a *similarity cube*. At this stage, the matcher-specific similarity values are aggregated into a combined similarity value for each combination of schema elements, and secondly, a selection strategy is applied to choose the match candidates for the schema element.

3.1.4 *Cupid* Matching System

The *Cupid* schema matching platform, by *Madhavan et al.* in [59], is a generic matching system that discovers mapping correspondences based on purely *schema-based* technique and does not employ any form of *instance-based matching*.

The procedural steps involved in achieving match pairs are described as follows. In the *first* step, the interconnected elements of a schema or the data model are modelled as a *schema tree*. These schemas are later encoded as graphs where the nodes represent schema elements. In the *second* step, the coefficient similarity between elements of the two (2) schemas is computed and then a mapping is deduced from the coefficients. This step is executed in two (2) phases, namely; the *linguistic matching* and the *structural matching* phases. In the first phase, individual schema elements are matched based on names, data types, domains, amongst others. using a thesaurus, acronyms and synonyms. In the second phase, the schema elements are matched based on the similarity of their *contexts* or *vicinities*.

The *third* and final step of the schema matching is the computation of *weighted similarity* (W_{sim}) - computed as in *Equation* 3.3 - out of the processes in the *second* step, from which a matching is created from the pairs of schema elements with maximal similarity coefficient.

$$W_{sim} = W_{struct} \times S_{sim} + (1 - W_{struct}) \times L_{sim} \tag{3.3}$$

where;

- W_{struct} is a *Coefficient* in the range 0 to 1;
- L_{sim} is the *Linguistic Similarity Coefficient*;
- S_{sim} is the *Structural Similarity Coefficient*.

3.1.5 *Clio* Project Schema Matching

The *Clio* schema matching system, by *Hernández et al.* in [42, 43] and *Miller et al.* in [68], is a joint project between the industry (IBM Almaden Research Centre) and academia (University of Toronto) that is engineered to manage and facilitate complex tasks of heterogeneous data transformation and integration [68]. The two (2) main components that form the core processing hub of the schema matching module of the project are the *Schema engine* and the *Correspondence engine*.

In the *Schema engine*, schemas are first loaded into the system in a manner where they are read from their underlying schema format, be it relational, XML, object-relational, object-oriented or any legacy source. The schema engine then augments the loaded schema with

additional constraint information such as the use of metadata, view definitions and the mining of possible keys and foreign keys. There is also an optional user interaction where schemas are verified by a user to ensure the validity of the generated information for necessary correction if required. This step in the overall matching process is facilitated by a GUI for user interaction and represented in the form of *Schema view* mode and *Data view* mode.

The output of the Schema engine processing step is the generated pair of schemas. This output is then feed into the *Correspondence engine*; where candidate value correspondences between the attributes of the schemas are generated and managed. At this stage of the matching process, attribute classifiers are used to learn the correspondences. These are also augmented with dictionaries, thesauri, and other matching techniques. The processing at the correspondence engine is facilitated by a user interactive GUI where the generated correspondences can be augment, changed or rejected by a user. This GUI interaction is represented in the form of *Schema view* mode and *Data view* mode.

The Clio system offers various features which makes it highly suitable generic schema matching for any form of data integration procedure. Some of these features are the ability to work on most generic metadata models, the ability to express many multi-cardinality correspondences between the attributes of the matching elements, and also the ability to script or easily transform the match results into mapping models [43, 42, 28, 68]. We discuss our novel adaptation of this schema matching system in Chapters 4 and 5.

3.1.6 Discussion of Schema Matching Approaches

In addressing the significant need of a schema matching phase in a data integration framework, as in this research paradigm, we compare the major most likely generic form of schema matching systems briefly described in the previous sections. We review these matching systems, so as to address some important techniques and comparatively analyze their efficiency [78, 87].

Table 3.1 highlights the strengths and weakness of each matching algorithmic system and their suitability for any schema matching procedure in terms of the usage of schema-level constraint or semantic information, and the cardinality of the element attributes. The usage of instance-level such as linguistic representation, and auxiliary information are also compared for each match approach, where it is applicable for Cupid, COMA and Clio. The use of hybrid or composite approach is also compared where it applies for all with the exception of Similarity Flooding. All of the approaches enable user interaction in their matching. The application usability of the match approaches are also highlighted for different data models.

Table 3.1: *Summarized Classification of some Generic Schema Matching Approaches*

Criterion / Schema Matcher	**Similarity Flooding**	**Cupid**	**COMA**	**Clio**
Instance-level Matching – *Text*	Not Applicable	Not Applicable	Uses additional Hybrid-level matching; of which currently, based on literature, no indication of any instance data	Makes use of instance data value distribution
Schema-level Matching – *Name*	Yes - Performs string-based matching of name equality	Yes - Performs string-based and linguistic matching	Yes - Performs string-based and linguistic matching	Yes - Uses a host of embedded algorithms, some of which perform string-based and linguistic matching
Schema-level Matching – *Constraint (data types, Keys, Foreign Keys)*	Yes - Uses data types and key properties in matching	Yes - Uses data types and key properties in matching	Yes - Uses data types and key properties in matching	Yes - Uses data types and key properties in matching
Use of auxiliary Information	No – Does not use any external information	Yes - Uses thesauri; acronyms, abbreviations, hypernyms, synonyms, etc.	Yes - Uses thesauri; acronyms, abbreviations, hypernyms, synonyms, etc.	Yes - Uses auxiliary thesauri information

Continued on next page

Table 3.1 – continued from previous page

Criterion / Schema Matcher	**Similarity Flooding**	**Cupid**	**COMA**	**Clio**
Syntactic Structural Match	Yes – In the case of iterative fix-point computation	Yes - In terms of graph tree matching weighted by leaves	Yes - Directed Acyclic Graphs tree matching using the leaves as the lead	Yes - the metadata translation of schemas presents a tree-view structure to match child leaves
Hybrid or Composite Matching	No	Yes – Hybrid	Yes - Hybrid and Composite matching of different matchers	Yes - Hybrid and Composite matching of different matchers
User Interaction	Yes – User validation of generated schema match candidate pairs	Yes - User can adjust threshold weights	Yes - An optional user feedback phase in each match iteration	Yes - User can validate generated schemas and value correspondences
Match Cardinality	One-to-one matching	One-to-one and many-to-one matching	One-to-one and many-to-one matching	Many-to-many matching
Usability	Useful in schema integration; but more practical with XML schemas	Useful in data translation applications; but intended to be for generic models	Useful in data integration applications	Useful in data exchange and data integration applications; but more practical for relational and XML schemas

3.2 Schema Mapping Discovery

Schema mapping approaches have been studied in metadata management based, on different content and expected results. Most of these generally focus on either, the discovery and manipulation of the mappings [9, 67, 65, 34], the tractability of query processing [61, 85], the composition for heterogeneous data support and functionalities for complete restructuring of data [51, 60], the compilation of mappings to bridge applications [64], the synthesis of the mappings [19], the holistic approach to resolving both schema and data mappings [36], the validation of the generated mappings [83], as well as the needed prerequisites for their formal specification [39], amongst others.

In addressing the need of integrating the heterogeneous data sources several formalisms, properties and requirements that are used to define mappings are expressed [83, 88]. These formalisms are therefore used to translate the data between the schemas. The expression of these mapping formalisms requires the careful creation and maintenance, so as to preserve the correlation of data translation and transformation between the schema and data from the sources and their intended targets [68]. We discuss these afore mentioned issues, within the context of schema mappings from the host of these studies.

3.2.1 Schema Mappings

The formulation of mapping relationships is needed in the modelling of schema mappings for metadata models in metadata management and operations. These mapping relationships are required to express the components or elements of the metadata models so as to uniquely define the relationships between the elements of the models. Schema mappings in this paradigm of metadata management are supposed to satisfy the *monotonic* and *deterministic* semantics of all source and target metadata models [61].

Bernstein et al. in [8] and *Kensche et al.* in [51] state that each mapping language, or formalism, should exhibit a list of requirements which should address the strengths and weaknesses of each of the mapping representations in the chosen mapping language. A summary of the key requirements that outlines the modelling of mappings is catalogued as follows:

- Mappings should be able to connect models, as well as the instances, of different modelling languages. This requirement might lead to an increase in the complexity of expressing the mappings on the data models.

- The mapping language should support complex expressions between sets of model elements in a manner of relating a set of elements in one model to a set of elements in another model. This could further be extended to any one of the models expressing an associated language for building the complex expressions over elements in the model, such as a query language or arithmetic expression.

- Mapping models must be able to support the nesting of mappings - to avoid redundant mapping specifications - and the provision of nested data structures to enable the reuse of mappings.

- Mapping models should exhibit the expressive richness of being generic enough across different modelling languages. In this case we avoid the need of defining separate elementary operations on mappings and have the flexibility of gaining mappings between mappings. This requirement in a way will enable the varied operations like copying a mapping, deleting a mapping, or selecting from a mapping, amongst others.

- Mapping models should support diverse data translation between the instances of the connected models. This requirement will enable the encoding of different instances in the wake of expressing more than one mapping between the given set of connected models.

Bernstein et al. in [8] further state that there are several fundamental issues to consider with regards to representation of mappings, and as such these issues are to be critically looked at when modelling mappings for any set of metadata models. These issues are briefly described as follows:

Interpretation of Mappings

The need to clearly interpret the mapping representations hinges on the magnitude of specification that goes into modelling mappings. There usually exists a spectrum of levels at which one can specify the mappings, and these could be done in a manner where at one extreme the mapping could specify the full semantic relationships between the two (2) metadata models. On the other hand, the mapping can be purely structural specifying only the elements in the two (2) metadata models that are related to each other and no mapping semantics. Addition-

ally, more semantic information can be attached to the mappings in an application-specific way, of which these semantics are not interpreted by the host model management system.

Directionality of Mappings

There is the need to check on the *directionality* of the mappings where a purely directional mapping will specify the transformation of data from its *domain* to its *range*. The issue of directionality evidently depicts how well the execution of any mapping results in a transformation function or complex expression for the elements of the metadata models.

Partial Mappings

The issue of partiality in the modelling of a mapping is most cases highlighted when a mapping does not fully connect or establish a relationship to all elements in the domain metadata model. This may be as a result of constraints on some of the corresponding elements in the two (2) metadata models. These constraints could represent a form of partial mappings and would need to be considered in the modelling of the overall mapping between the two (2) metadata models.

Ten Cate and *Kolaitis* in their recent work in [88] on schema mappings also highlighted some structural properties that schema mappings should exhibit. In their work, they state that schema mappings should be characterized by properties such as, *closure under target homomorphisms*, *admitting universal solutions*, *allowing for conjunctive query rewriting*, *closure under target intersection*, *closure under union*, *n-modularity*, and *reflecting source homomorphisms*. Their work outlines the intuitive proofs and complexity issues associated with modelling of schema mappings for any form of operation such as data integration, data sharing, or data exchange.

The process of modelling mappings for metadata models most often requires a significant amount of work in ensuring a high degree of validation, which should portray the semantic intention of the correspondence relationship between the elements of the metadata models. *Rull et al.* in [83] and *Madhavan et al.* in [58], attempt to propose some approaches for validating schema mappings and the definition of some important properties that these validated mappings must satisfy. In their work, they define a distinguished derived predicate (a query) that describes the fulfilment of any of the chosen mapping property. This definition is done over a new schema which would integrate the two (2) mapped schemas and a set of integrity constraints that explicitly expresses the relationship modelled by the mapping. In their assessment, the distinguished predicate is sustainable over the new schema if and only

the chosen property holds, and a derived property is also sustainable if the schema admits at least one fact or knowledge about it.

The authors in [83] and [58] therefore attempt to define and describe the four (4) forms of properties that *first-order* mapping models - as in the case of GLAV mappings - must satisfy; namely, *mapping inference*, *query answerability*, *mapping satisfiability*, and *mapping losslessness*.

Mapping Inference

Mapping inference consists in checking whether a mapping entails a given mapping formula, and that whether or not, the given formula adds new mapping information. This property can be used to check for redundancies that exist in the mapping or to check the equivalence of two (2) different mappings. It can also be used to check whether a given mapping is *minimal*, where removing any formula from the mapping causes a resultant loss of information. The results from the work of the authors in [83] and [58] showed that in the context of conjunctive queries and schemas - with or without integrity constraints - the checking of this property involves finding a maximally contained rewriting and checking two equivalences of conjunctive queries.

Query Answerability

Query answerability involves checking whether the mapping enables the correct answering of a certain set of queries, possibly infinite, over the schemas on which they are mapped. This property evolves from a reasoning given that mappings are typically required to enable a certain task and that a mapping that is partial or incomplete may be unsuccessfully used for the certain tasks. Once again, the results of the work of the authors in [83] and [58] showed that in the context of conjunctive queries, with or without integrity constraints on the schemas, this property can be checked by means of the existence of an equivalent rewriting.

Mapping Satisfiability

Mapping satisfiability aims to check whether there is at least one case in which the mapping and the constraints are satisfied simultaneously. This may be the outcome of possible incompatibilities between the constraints and the mapping, or even between the mapping formulas; whenever there is a mapping between schemas that have constraints. The issue of

constraints arises when the data retrieved from the sources cannot be reconciled in the global schema where the schema and the mapping are satisfied.

Mapping Losslessness

Mapping losslessness seeks to check whether all pieces of data from computed tuples that are needed to answer a given query over a schema involved in an integration procedure are captured by the mapping. This property may be required as a result of exposing hitherto sensitive data from the computation of a query over a global schema, in such contexts answering a query becomes too restrictive. In this case, such sensitive local data are always represented as the mapping will seek to fulfil this *losslessness* property.

In the formulations of mapping representations for integration systems, two (2) forms of categorizations are noted; namely, *Intensional* and *Extensional mappings*. These categorizations are based on the type of semantic intention on the models on which they are expressed [51]. *Intensional mappings* articulate on the intended semantics of the model and they interrelate model elements by set relationships such as equality and subset relationships. Since these *intensional mappings* infer only on the semantic constraints of a model, they are unable to explicitly refer unto the instances of models. This fact makes them very much unhelpful in cases of data translation.

Extensional mappings on the other hand, define inter-schema constraints that must be satisfied and therefore validate all the instances of the related schemas. Such extensional mappings are usually thought off being executable mappings which are represented as instances and expressed as a *tuple* of states one for each of the models involved in the mapping. Some of these mappings can further be denoted using morphisms such as SQL views, XQuery, relational algebra, Datalog, or an expression in a concrete language deployed in scripts such as SQL DML, XSLT, GLAV, amongst others [65]. As earlier stated in Section 1.2, extensional mappings can be represented as two (2) queries which are related by some operator, possibly equivalent or subset relationship [51]. Most executable and formal mapping representations rely much on the domain of the data model and these can be expressed in *first-order* logic assertions of source-to-target Tuple Generating Dependencies (s-t tgds), also known as GLAV mappings; or *second-order* logic Tuple Generating Dependencies (SO tgds). In our research context of the expressing executable mappings, we will solely focus on *first-order* logic extensional mappings.

3.2.2 LAV Schema Mappings

Local-As-View (LAV) mappings are a set of mapping models in which there is an assertion of mapping elements that associates to each element of the source schema, a query over the global (mediated) schema. In this case elements in a source schema are expressed as views over the global schema, since the source queries in the assertions are constituted by one atom and exactly one assertion appears for each relation symbol in the source schema [55, 18, 2, 60]. The LAV mapping approach is generally adopted in the case where the data integration system is based on an enterprise model or an ontology. This idea that is drawn out of an assertion that global schema is stable and well established in an organization, and addition of a new source just goes in enhancing the mapping with new assertions without any change [55].

Arocena et al. in [2] recently explain that the composition of LAV mappings is not only *first-order* logical assertion, but can now be characterized by a much more general definition of being a *second-order* source-to-target Tuple Generating Dependency (tgd) such that; it has exactly one literal from the source schema atom and every variable must be distinct. In their work, they further state that a LAV mapping is made up of a source, target, and a set of LAV tgds where the LAV mappings are composed from all these parameters.

This intuition behind the specification, characterization and subsequent modelling of LAV mappings scales down to the kind of views expected to be expressed in the mappings from the source schema to the global schema [55]. Three (3) different kinds of views are explained in the literature; namely, *sound*, *complete* and *exact* views. These views are normally conjectured based on the composition of tuple extensions and go a long way in underlining the logical modelling of LAV mappings. We present an example of a LAV mapping in *Example* 3.2.1.

Example 3.2.1 *We use the schema diagram Figure 4.1 to describe the LAV mapping model. The LAV datalog query for the Fact Table in the Claims Transactions schema in relation to the Fact Table in the Global DW schema is scripted as follows:*

Fact_ClaimTransaction *(ClaimTransactionDateKey, ClaimProcessingDateKey, ClaimReceivedDateKey, InsuredPartyKey, InsuredPolicyEmployeeKey, InsuredPolicyKey, InsuredPolicyItemKey, ClaimTransactionTypeKey, ClaimantKey, ClaimThirdPartyKey, ClaimKey, PolicyNumber, ClaimTransactionAmount) :=*
Fact_GlobalSchema *(TransactionDateKey, ProcessingDateKey, ClaimReceivedDateKey, In-*

suredPartyKey, InsuredPolicyEmployeeKey, InsuredPolicyKey, InsuredPolicyItemKey, TransactionTypeKey, ClaimantKey, ClaimThirdPartyKey, ClaimKey, PolicyNumber, TransactionAmount). ∎

Query Processing In LAV

Query processing in LAV mappings is based on incomplete information in the global schema, as a result of partial views from the source schemas. This concept of incomplete and open sources as evident in LAV mappings makes query answering in the global schema difficult. This, as a result, opens up a wide spectrum of high combined complexity in terms of data complexity and expression complexity. The comprehensive work on LAV mappings in [55] further state two (2) approaches to view-based query processing; namely, *view-based query rewriting* and *view-based query answering*. It will be noted that these approaches provide a medium in expressing queries over LAV mappings.

3.2.3 GAV Schema Mappings

Global-As-View (GAV) mappings are a set of mapping models in which there is an assertion of mapping elements that associates to each element in the global (mediated) schema, a query over each of the source schemas. The GAV mapping modelling presents an architecture where the global (mediated) schema is expressed as views over each of the source schemas; and as a result the mappings uniquely articulates how well to retrieve information from the global schema, and in assessing the overall constitution of global schema elements [55, 18, 60]. In the GAV mapping approach, there is a straightforward well-defined association between the global schema and the sources and the burden of complexity only falls on designing the global mediated schema [90].

GAV mapping models generally favour a data integration system where the set of local sources are very stable and less susceptible to changes; and as a result, enabling the efficient processing of queries posed to it [55]. A drawback to this form of architecture is the addition of new sources to the existing framework, which presents a likely problem to the existing structure of the model. This is because the new source may require the redefinition of various elements of the global mediated schema from a resultant change in the associated views being expressed in the mappings.

The logical intuition that underpins the formulation of GAV mappings stems from its characterization and the expression of views. From the studies conducted so far, GAV mappings are characteristically expressed as exact under a *Closed World Assumption* (CWA) and

sound under an *Open World Assumption* (OWA) [55, 50]. We present an example of a GAV mapping in *Example* 3.2.2.

Example 3.2.2 *We use the schema diagram Figure 4.1 to describe the GAV mapping model. The GAV datalog query for the Fact Tables in the Claims Transactions and the Policy Transactions schemas in relation to the Fact Table in the Global DW schema is scripted as follows:*

Fact_GlobalSchema *(TransactionDateKey, ProcessingDateKey, InsuredPartyKey, InsuredPolicyEmployeeKey, InsuredPolicyKey, InsuredPolicyItemKey, TransactionTypeKey, PolicyNumber, TransactionAmount) :=*
Fact_PolicyTransactions *(PolicyTransactionDateKey, PolicyEffectiveDateKey, PolicyHolderKey, PolicyEmployeeKey, PolicyCoverageKey, PolicyCoveredItemKey, PolicyTransactionTypeKey, PolicyNumber, PolicyTransactionAmount),*
Fact_ClaimTransactions *(ClaimTransactionDateKey, ClaimProcessingDateKey, ClaimReceivedDateKey, InsuredPartyKey, InsuredPolicyEmployeeKey, InsuredPolicyKey, InsuredPolicyItemKey, ClaimTransactionTypeKey, ClaimantKey, ClaimThirdPartyKey, ClaimKey, PolicyNumber, ClaimTransactionAmount).* ∎

Query Processing In GAV

The expression of relations in the global schema are described as views of the relations in the union of the local schemas, and this feature normally lead to the non-existence of integrity constraints on most GAV mediated schemas. Hence, the mappings lead to the expression of *exact views* under a CWA in the global mediated schema. This, in turn, allows for the processing of queries basically reliant on a simple *view unfolding* [55]. However, in the presence of integrity constraints in the global mediated schema, the views expressed are *sound*, and which makes query processing more difficult.

3.2.4 GLAV Schema Mappings

Global-And-Local-As-View (GLAV) mappings are a set of mapping models in which an assertion of mapping elements expresses the relationships between the global schema and the sources. This establishes an association by making use of both LAV and GAV assertions [55]. It presents a view of a modelling framework for a data integration system where every mapping assertion that has a query over the source schema uniquely corresponds to a query over the global mediated schema. This feature makes the GLAV mapping model express

mapping views where the sources are sound and an equivalent *arity* of both queries - from LAV and GAV - is established in the mapping model.

The concept of GLAV mappings was first introduced in [29], where *Friedman et al.* proposed a mapping language that combines the expressive power of both LAV and GAV and that will allow flexible schema definitions independent of the particular details of the sources [75]. The motivation for the authors in [29] in line with this proposition was to address inherent difficulties of global mediated schemas and their source schemas.

In the first place, they addressed the issue that the source schemas often contain differing levels of detail from each other, and from the global mediated schema. Secondly, the modelling of the same information by seemingly different source schemas may most likely result in the splitting of attributes into relations in different ways - of normalization in the database schema. This disadvantage of undesirable consequences of using either a pure GAV or pure LAV mapping model makes the GLAV mapping model, being a hybrid of the two (2), a preferred model with enhanced expressive capabilities. We present an example of a GLAV mapping in *Example* 3.2.3.

Example 3.2.3 *We use the schema diagram Figure 4.1 to describe the GLAV mapping model. The GLAV datalog query for the Fact Table in the Global DW schema in relation to the Fact Tables in the Claims Transactions and the Policy Transactions schemas is scripted as follows:*

Fact_GlobalSchema *(TransactionDateKey, ProcessingDateKey, ClaimReceivedDateKey, InsuredPartyKey, InsuredPolicyEmployeeKey, InsuredPolicyKey, InsuredPolicyItemKey, TransactionTypeKey, ClaimantKey, ClaimThirdPartyKey, ClaimKey, PolicyNumber, TransactionAmount) :=*
Fact_PolicyTransactions *(PolicyTransactionDateKey, PolicyEffectiveDateKey, PolicyHolderKey, PolicyEmployeeKey, PolicyCoverageKey, PolicyCoveredItemKey, PolicyTransactionTypeKey, PolicyNumber, PolicyTransactionAmount),*
Fact_ClaimTransactions *(ClaimTransactionDateKey, ClaimProcessingDateKey, ClaimReceivedDateKey, InsuredPartyKey, InsuredPolicyEmployeeKey, InsuredPolicyKey, InsuredPolicyItemKey, ClaimTransactionTypeKey, ClaimantKey, ClaimThirdPartyKey, ClaimKey, PolicyNumber, ClaimTransactionAmount).* ∎

3.2.5 Discussion of Schema Mapping Approaches

In analyzing the features and characteristics of the individual mapping models discussed so far within the framework of data integration system, we compare and highlight on the strengths and weaknesses of each of the LAV, GAV and GLAV mapping models, as described and studied in the literatures in [55, 18, 50, 29]. We address a summary of their characteristics in *Table* 3.2, discussing on various criteria. In terms of query processing, the GAV mapping model performs better, because of the higher number of overlapping elements. The GLAV also performs well, but the inclusion of local sources impacts on its query processing.

For each of the models, the introduction of new sources is handled differently because of the need of changes in the schema structure. The LAV mapping model offers a better platform because of the stability of its source elements. In terms of the type of query processing, the GAV mapping model adopts a view unfolding approach which extends the query expressions unto the source elements, and offers a better medium of querying data.

3.2.6 Clio Project - Schema Mappings Generation Platform

The logical assertions applied in the modelling of schema mappings, in line with GLAV mapping models have received some study in various literature, and added knowledge and techniques have incrementally aided in improved mappings. In order to explain on the functional components of the mapping for various metadata management operations such as data exchange, data sharing, data integration, data warehousing, amongst others, we discuss the *Clio* Project [67, 1, 43, 42, 28, 68, 30, 37] as a schema mapping platform. We choose this schema mapping platform because of its near-generic handling of schemas or data models. It expresses semantics and runtime executables for the practical implementation of the GLAV logical formalism.

In the Clio project, with emphasis on the schema mapping aspect, we consider the methodologies of schema and model translation, semantic value inference, query discovery techniques, and algorithms for automatically generating queries for data transformation, and some other procedures. The preliminary work in schema mapping which concerns schema matching have been discussed in Section 3.1.5 and is used as a background here. The mapping formalism design is discussed in this subsection.

The more advanced and expressive methodologies applied in the schema mapping process with *Clio* focus on the *mapping language and schema constraints*, the *mapping generation approach*, and the *query generation and transformation rules* for metadata operations. We briefly describe each of these methodologies and point out the main perspectives that make

Table 3.2: *Comparison of Schema Mapping Modelling Approaches*

Criterion / Mapping Model	**LAV**	**GAV**	**GLAV**
Logical Assertion	Associates each element in the source schema as a query over the global mediated schema	Associates each element in the global schema as a query over each of the source schemas	Associates each element in the global schema as a query over identical elements in each of the source schemas
Complexity of Query Processing	Query processing is difficult which could lead to appreciable level of undecidability	Query processing is quite easy, but could be difficult in the face of integrity constraints	The rate of query processing is appreciable, better than the LAV because of the incorporation of overlapping elements
Introduction of new sources or source elements	Very easy to incorporate new sources to the global schema; since nominally all source elements are always represented in the global schema	Very difficult and impracticable to add a new source to the global schema; since new source may require the redefinition of various elements in the global mediated schema and a rewriting of the views	New sources or source elements can be added with less difficulty, but the new source elements must first satisfy the constraints and source definition on the global schema
Stability of Sources	Used often when the global mediated schema is very stable	Used often when the set of local sources are very stable and less susceptible to changes	Can be used where either the global schema or local source schemas is stable, but more efficient when the global schema is stable
Form of Query processing	View-based query rewriting and view-based query answering	View unfolding	View unfolding and view-based query rewriting
Modelling Specification	Declarative approach in specifying the content of the local sources in terms of the global schema	Procedural approach in specifying the content of the local sources in terms of the global schema	Combines both declarative and procedural approaches in the datalog query specification

the *Clio* project a major mapping tool for GLAV schema mappings.

Mapping Language and Schema Constraints

The authors in [67, 28] address an overview of the general mapping development by addressing the schemas and the associated instances the mapping tool handles. In their work, they describe that though *Clio* is multifaceted with handling of schemas, they primarily dealt with relational and XML schemas; with an approach of using nested relation model to model both types of schemas where they make no assumption about the relationship between the schemas and how they are created.

In terms of the type and form of mapping model adopted, the authors rely on the formal sound GLAV mapping models. Here, they interpret earlier established correspondences and expressed is an inter-schema inclusion dependency or a more general source-to-target *tuple generating dependency* (tgd). These *tgds* are expressed as containment relationships that do not restrict the kind of data that can be in the target. The mapping approach also deals with the forms of schema constraints; namely, *primary paths* and *relative paths* which correspond to the tables in the two (2) schemas and the associations between the data elements, as well as the manipulation of relational foreign key and referential constraints, as needed in a later mapping algorithm.

Mapping Generation Approach

The mapping generation approach in *Clio* makes use of an algorithm where associations between atomic elements within the source and target elements are utilized. The semantic associations conveyed here specify how individual data values should be connected in the target with a depiction of some real-world association. Different forms of semantic associations, which are outlined and explained in [28] are *structural associations*, *user associations*, and *logical associations*. It will be noted that these associations are based on different semantics and logical implications and are combined in a mapping algorithm. The authors further state that, since there may be several ways of associating elements within a schema, they devise an algorithm that uses logical inference to find all associations represented by referential constraints and a schemas relational and nesting structure [67, 28].

The algorithm for generating schema mappings in *Clio* makes use of a logical assertion of correspondences that are meaningfully combined and then discover the maximal sets of these correspondences by testing whether the elements they match belong to the same logical association. In cases where there is a representation of multiple pairs of logical associations,

of which not all of the pairs will generate mappings, some pairs of associations are *subsumed* by other pairs and later discarded in an activity of minimization in the algorithm. This heuristic phase of the algorithm tries to eliminate a large number of unlikely mappings as it occurs in practice.

Query Generation and Transformation Rules

One unique feature of the *Clio* mapping platform is the ability to generate executable queries based on the schema mappings and these codes become priceless tools for data exchange and data integration operations. The queries are generated in the form of SQL, XQuery, or XSLT where in the case of purely relational source and target schemas these queries generate a *universal solution*. The algorithm used in generating the queries makes use of *Skolem functions* (one-to-one functions) that generate values based on a set of source values [28]. However, in the case of nested target schema, *Clio* applies additional grouping and nesting to produce a target instance that is in *partitioned normal form*. This is done to reduce the redundancy in the target instance and producing single *tuple* for each entity, and grouping all entity elements that belong to the same entity under a single entity grouping [30].

3.2.7 Generic Schema Mappings

The study on the generic formulation and generation of schema mappings has received attention in literatures and has been attempted by authors in [52, 51, 61]. In this section, we highlight the major contributions from these studies and the merits they offer for most model and metadata operations and applications.

Schema mappings are generally expressed in some logical formalism that is typically a fragment of *first-order* source-to-target *tgds* or a fragment of *second-order* source-to-target *tgds*. The exhibition of some properties of these fragments, such as the ability to generate *universal solutions* or a closure under target homomorphisms, make the said mapping formalism prime and likely candidate for the relationship between models in a data exchange or data integration application [88]. The need to support data translation between heterogeneous models in the form of entity-relationship models such as relational schemas, object-oriented and nested data structures such as XML schemas, or semantic web models such as OWL ontologies, has driven the edge to choose a logical formalism that is capable of complete data restructuring and query answering against a global mediated schema. To this end, a proposition of generic schema mappings is upheld to deliver answers in this vein of heterogeneity and data programmability.

Kensche et al. in [52, 51] propose a generic framework of defining a mapping representation across several modelling languages and has the capability of fulfilling mapping requirements of *expressiveness* and *executability*. In their work, they attempt to underscore such a representation that addresses the *composability*, *invertability*, *decidability*, and *executability*of mappings using a composition algorithm based on *second-order tgds*. The generic mapping language that they devise in their work also offers the translation of the mappings into a specific data manipulation language (DML) in the form of generated executable queries and update statements for SQL and XML.

Furthermore, *Marnette* in [61] also attempt to introduce a notion of generic framework that enriches the standard GLAV mappings with more expressive power and with an intuitive notion of semantics that addresses different criteria of *soundness*, *completeness*, and *laconicity* (non-redundancy and minimal size). This study also tries to address the identification of tractable generalized schema mappings among the class of *tuple generating dependencies* (tgds) based on a polynomial-time algorithm. In assessing such an approach of generalized mapping, the tractability results obtained for tuple generating schema mappings from the polynomial-time algorithm are used in some other *simulation procedure* to further strengthen the generation of an output of schema mappings which is highly tractable and much more generalized.

3.3 Schema Merging

The increasing rate in the amount of data in businesses and organizations results in the heightened need in drawing semantic knowledge, the support of decision-making, and the ability to draw tangible information from these myriad of disparate data sources. This need motivates the initiative of providing a general platform where these needs are addressed. The consolidation of most of these data into a singular module serves as a stimulus for this general platform of schema merge sought for - where other unattended to problems are also solved.

As earlier stated in Section 1.2, there have been numerous studies in this area of schema merging. Some of these studies have focused on generic models [74, 8], global mediated schema [75], data warehousing [11, 17], schema - view and database - integration [3], and whiles others have rather concentrated in the generic sense of schema merging [77, 63, 13].

In this section, we discuss some of the contents and results of the these studies in relation to the properties and technical requirements, the semantics to consider, the formulated algorithms, as well as the discussion of a few methodologies applied in schema merging or

data integration procedures in the studies so far.

3.3.1 Schema Merging

The concept of schema merging relies on a variety of procedures and transformations on the elements of models (or schemas), and the associated mapping relationships that exist between the elements of these models. The success of the merging process is highly dependent on the expressiveness and efficiency of the mapping models in the overall merging. This is so because of the need for the merge procedure to satisfy some semantic representation, technical requirements, merge properties, as well as the resolution of conflicts associated with the elements of the integrating models. One distinctive feature of the merged model is to possess non-redundant elements and with their characteristics that satisfy all the integrating models and fulfils the properties of those elements in the models.

Batini et al. in [3] in their assessment and opinion, point out some qualitative criteria that that a global conceptual (mediated) schema should depict. In their work they state that, when schemas go through a merge procedure, there is a superimposition and restructuring of elements in the global mediated schema. As a result these elements should therefore satisfy the stipulated criteria of a maximum containment of the properties in a duplicate-free element mediated schema.

Pottinger and Bernstein in [75] further enhanced the work of the authors in [3] by emphasizing on the earlier requirements and adding some other new ones. In summarizing the combination of the set of technical requirements which have been stated by the authors in [3] and [75], we briefly describe these requirements and their expediency in the face of schema merging. We briefly discuss each of these requirements.

Completeness and Correctness

The *completeness* criterion ensures that there is no information loss in the mediated schema and make certain that each source relation is accessible by a query over the mediated schema. This criterion is achieved and made executable in a form where for each source relation there is a query over the mediated schema that is equivalent to the identity source query. The adopted mapping models which exist to establish a relationship between the source schemas and the global mediated schema make this criterion possible by enabling some expressions and transformations where structured data from different sources are distinguished and information represented in the component schema are exposed.

Overlap Preservation

This criterion requires that each of the overlapping elements specified in the input mapping is exposed in the mediated schema relation. In every schema merge procedure, there exists, most often, an overlap of elements from both sides of the integrating source schemas. This criterion seeks to ensure that the input mapping offers a medium where these overlapping elements are uniquely expressed in the form of queries in the mediated schema. In executable forms, the criterion is expressed as; for each overlap of elements, there exists a query over one relation in the mediated schema that is equivalent to the overlap specification.

Extended Overlap Preservation

This criterion becomes needful in the wake of satisfying the *completeness* requirement, where in terms of exposing source elements attributes in the global mediated schema, there is the addition of attributes that go beyond the normal overlap of attributes but are needed for convenience. This addition of attributes might not be necessary from a purely *completeness* requirement perspective, but might be desirable to avoid the representation of joins of redundant attributes in the global mediated schema. This criterion is explained in formal executable terms as; for each overlap query that is paddled with an existential variable, there exists a query over one relation in the mediated schema that is equivalent to this query.

Normalization

The *normalization* criterion seeks to address the limit to the inclusion of overlapping attributes in source schema relations as being exposed in the global mediated schema relations. This requirement is proposed primarily to avoid the element attribute redundancy violations beyond those introduced by the overlapping element specifications. Formally stating the requirement as; for each global mediated schema element relation that corresponds to an overlapping set of attributes from the source schemas, there exists a single element from the mediated schema that represents each overlap.

Minimality

The *minimality* criterion seeks to summarize all the afore-mentioned requirements. Here, we discover and eliminate redundancies in the global mediated schema, and still present a streamlined schema satisfying all the element attribute queries of the source schemas.

In satisfying most of these technical requirements and criteria for schema merging, there arise some conflicts whether in the semantics of the elements, the structure of the models, or the description of the elements of the models [74, 76, 77, 3, 81, 6]. These conflicts emerge as a result of the diversity of the representations in the same real-world entity or semantic constraints in the elements coming from different entities. There is therefore the need to address these conflicts during the restructuring and modelling of the global mediated schema.

Different authors in their way of addressing these conflicts outline and describe them based on the semantics of the models that they deal with. As a result, these authors also propose some set of rules or procedures that can be followed in the resolution of these conflicts, of which some propose the use of the constraints, element expressions and transformations in the mapping models. *Batini et al.* in their study in [3] draw attention to some set of conflicts likely to be encountered, where as *Pottinger and Bernstein* in their study in [74] also describe some other forms of conflicts in a more generic way that can be applied to most specific models. *Quix* in [76] and *Quix et al.* in [77] also highlight some forms of likely conflicts and how they are resolved in their study of conflict management and resolution as part of the process of schema merging. To summarize these forms of conflicts from the various studies conducted by different researchers, we outline and briefly describe a set of frequent conflicts that runs through most studies on schema merging, either in the generic or specific forms and how they are resolved.

Schematic or Structural Conflicts

Batini et al. in [3] and *Quix* in [76] classify these conflicts as *Structural Conflicts* where as *Pottinger and Bernstein* in [74] also classify these types of conflicts as *Representation Conflicts*. These types of conflicts arise as a result of different representations of the same real-world concept, and may be due to the different choice of modelling constructs and integrity constraints and the specific characteristics of the different data models adopted in the methodologies. These conflicts are further distinguished into different kinds; namely, *type*, *dependency*, *key*, and *behavioural*. These *structural* conflicts are resolved by using the input mapping during merge, where there is the specification of the elements from all the integrating models or schemas, as well as the properties and the semantic relationships between these elements.

Heterogeneity Conflicts

These forms of conflicts are classified as *Heterogeneity Conflicts* by *Quix* in [76] and *Meta-model Conflicts* by *Pottinger and Bernstein* in [74]. Their occurrence is as a result of the representation of models that are described in different modelling languages and there arises a host of inconsistencies in the constraints of the models. An illustration of such conflict could be the representation of a real-world entity such as *customer*, as an *SQL table* in a model, say *A*, and an *XML DTD* in a model, say *B*, and the merged model has to be represented in an *SQL table*. These forms of conflicts are usually resolved outside the mainstream merging procedure, where the models are independently conformed to a laid out set format of constraints and enforcing models constraints by declaratively specifying them. This makes the overall conflict resolution process a non-generic procedure for most model merging processes.

Descriptive Conflicts

These forms of conflicts are classified as *Descriptive Conflicts* by *Quix* in [76] and *Fundamental Conflicts* by *Pottinger and Bernstein* in [74]. They are also partially classified as *Naming Conflicts* by *Batini et al.* in [3]. They occur as a result of the same elements being described by different set of properties; hence, the evolution of a possible inconsistency among the elements. Another representation of this kind of conflict is where an element possesses a *one-type* constraint and another element rather possesses a *two-type* constraint. An example could be the case where an element, say *ZipCode*, in one model, say *A*, possesses a one-type constraint of *integer* data type; whiles its corresponding element in another model, say *B*, possesses a two-type constraint of *varchar* (string, integer) data type. During merge, there arises a conflict in the particular constraint in which to represent the elements property. *Descriptive conflicts* are most times resolved in the input mapping based on the choice properties and constraints specified in the mapping, as well as the constraints on the relationships of the elements. A clear definition of the properties of elements of each of the models also aids in making these semantic modelling constructs more expressive during merge.

Semantic Conflicts

Semantic conflicts occur when model elements describe overlapping sets of objects. This leads to multiple properties or roles of the same type for one model element. An illustration of such a conflict could be the representation of a real-world entity, such as *employee*, with differing properties of *social_security_number* and *employee_number* being the respective keys in two (2) different component schemas. These forms of conflicts are normally resolved by

keeping the more general property among the set of properties for a particular model element. For instance, in the case where there are multiple roles of the same type for an element, the more general role is preserved; as in a typical example where if a key reference is in conflict with an association, the association property is preserved.

In general terms, it would be noted conflict resolution strategies are varied and based on the kind of model structure and also the elements and their properties, there can be a multi-level procedure or an ad hoc measure in resolving these conflicts [74, 77]. In the case of a multi-level procedure, the resolution process could start from the input models and mappings, by the parameters in the merge algorithm, or by the metadata on the model.

3.3.2 Generic Schema Merge Approach – *Quix et al.*

Quix et al. in their study in [77] propose an approach to schema merging based on the generic role-based meta-model and *intensional* mappings based on the real-world states of model elements. In their work, they point out the perspective of schema merging where there is the need to identify the candidate meta-models as well as the input mappings.

From their viewpoint, schema mapping models are supposed to exhibit a complex structure capable of answering the structural heterogeneities and semantic knowledge inference in the expression of relationship, and the transformation of schema elements and instance data of the meta-models during merge. In terms of the kind of meta-models, their study reveals that for a generic merge the native meta-models should employ some generic schema representation. This generic schema representation is sometimes done outside a model management system requiring some operators to be implemented for different combination of meta-models. Schema merging procedures are always inherent with the resolution of conflicts and as a result these conflicts are dealt better when there is enough information about the meta-models and also an expressive input mapping model. Their work points out some of these conflicts which have been discussed in Section 3.3.1; namely, *structural heterogeneities*, *semantic conflicts*, *descriptive conflicts*, and *heterogeneity conflicts*. The authors further reflect on how these conflicts are resolved in line with their generic meta-models.

In their study, the semantics of model elements have to be defined in relation to the real-world representation of the objects they describe. The formal semantics for these role-based meta-models characterizing the structure of their instances are described in four (4) different functionalities; namely, *Domain*, *ObjectSet*, *Aggregate*, and *Association*. These definitions, which have been clearly described in the literature, play a essential role in the transition of real-world semantics and in the implementation of the model merge procedure.

3.3.3 Generic Model Merge Approach – *Pottinger & Bernstein*

Another form of generic approach in model merging is studied by *Pottinger and Bernstein* in [74]. In their study, they used generic models which expressed semantics of object features of element *Name*, *ID*, and *History*, and also binary element relationships with cardinality constraints.

The approach adopted by the authors in [74], mainly examine the problem of merging models using given mapping correspondences. They propose a schema merge algorithm that will enforce such a merge procedure. In their study, the authors introduce a set of technical requirements that the merged model must satisfy, and also the handling of conflicts and how they are resolved. The authors further attempt to highlight on some of the representations or properties that models can assume, and describe the conventional meta-data terminologies of *model*, *meta-model*, and *meta-meta-model*; where a *model* is symbolized by an *element* with *relationships* between the elements.

One unique feature of their approach is the proposition of a *preferred model* as part of the merge procedure, and the use of a *first-class* mapping model mainly based on *equality* and *similarity* constraints. Based on the semantics adopted in the overall merge approach, the authors address a set of criteria, termed *Generic Merge Requirements* (GMRs), that the new merged model must satisfy. The GMRs that were outlined and described in their study were; *Element Preservation*, *Equality Preservation*, *Relationship Preservation*, *Similarity Preservation*, *Meta-meta-model Constraint Satisfaction*, *Extraneous Item Prohibition*, *Property Preservation*, and *Value Preference*. It will be noted that the satisfaction of these GMRs leads to a duplicate-free union and a vivid representation of the elements of all integrating models.

As part of deriving a merged model satisfying all or at most the GMRs, the authors categorize the likely conflicts to be encountered and which have been discussed in Section 3.3.1 as *representation conflicts*, *meta-model conflicts*, and *fundamental conflicts*.

3.3.4 Discussion of Generic Schema Merge Approaches

The two (2) approaches of the schema merge that we have discussed present a generic methodology of merging ontologies or data models within the context of *model management*. Each of the approaches uses a unique way of either expressing the input mappings, the input mapping models, the identification and resolution of conflicts, or the algorithmic methodology. In this section, we attempt to comparatively underscore the strengths and weaknesses of the two (2) approaches of generic merge as studied in [74, 77], amongst a host of others which were not

discussed in this thesis document.

We address these comparisons based on the expression of model type, the input mapping model adopted, the expression of mapping correspondences, the conflicts and their resolution, the technical requirement satisfaction, and the overall methodology adopted in *Table* 3.3.

In terms of the type of mapping correspondence, the *Quix et al.* [77] approach presents a more expressive set of mappings aside equality and similarity presented by *Pottinger and Bernstein* [74]. In the area of conflicts, similar forms of conflicts are outlined by both approaches, but these are categorized differently and also a proposition of different resolution measures. The merge algorithm formulated by *Pottinger and Bernstein* [74], present a unique feature of preferred model. On the other hand, *Quix et al.* [77] also utilizes the real-world states of the elements and mappings. In terms of mapping models adopted, where as *Pottinger and Bernstein* [74] uses first-class mapping models of elements and relationships, *Quix et al.* [77] on the other hand, use intensional and nested mappings because of the state semantics of the elements. We describe each of the comparisons in the Table.

3.4 Integration of Multidimensional Data Models

The study of data integration in relation to multidimensional data models has received minimal research. In this sub-section, we review some of the studies that have been conducted where independent and heterogeneous multidimensional databases (data marts) are merged, on the basis of their schema and instance data.

3.4.1 Concept of Multidimensional Data Models Integration

Multidimensional data models are models that exhibit special features of different perspectives - in terms of *dimensions* - and possibly numeric data *measurements* - in terms of *facts* - for every set of data record residing in a schema. These are normally the end product of dimensional modelling and data warehousing, as discussed in Section 2.1. Data integration in this domain normally refers to the merging of multidimensional databases, of both schema structure and instance data, where the various *dimension* and *fact* tables in the independent schemas are incorporated into a single module.

Cabibbo et al. in their series of studies on *dimension compatibility* and data integration in [16, 15], and [14] address the problem of data integration in relation to multidimensional databases (data marts). In their work in [16] and [14], they introduce fundamental assertions of *dimension algebra* and *dimension or fact compatibility*. Different forms of heterogeneities

Table 3.3: *Comparison of Generic Schema Merge Approaches*

Criterion / Merge Approach	**Generic Merge by *Pottinger and Bernstein***	**Generic Merge by *Quix et al.***
Type of Model	Uses a generalized meta-model with object-oriented capabilities	Uses generic role-based meta-model that is semantically very expressive
Mapping Model Adopted	*First-class* mapping models consisting of elements and relationships	*Intensional* and nested mappings based on real-world states of model elements
Type of Mapping Correspondence	Applies only *equality* and *similarity* mapping elements in the mapping model	Aside *equality* and *similarity*, applies more assertions of *disjointness*, *overlap*, *subset relationships*
Technical Requirements Satisfaction	Proposes *GMRs* for the algorithm	Satisfies all the *GMRs* proposed by *Pottinger et al.*, but *Extraneous item prohibition* and *Property preservation* are adapted in the input mappings
Conflicts Resolution	Handles and proposes resolution of *representation conflicts*, *meta-model conflicts*, *fundamental conflicts*	Handles and proposes resolution of *structural heterogeneities*, *semantic conflicts*, *descriptive conflicts*, *heterogeneity conflicts*
Merge Algorithm Methodology	Applies an optional designation of a *preferred model* to aid unspecified choice in the mapping model	Uses real-world semantic states in the intensional mappings; that answers all forms of ambiguities

are existent in dimensions. The addressing of these needs made them to introduce a novel theoretical concept of *dimension algebra*, which enables the selection of relevant portions of a dimension for integration. This dimension algebra is basically based on three (3) main operators; *selection*, *projection*, *aggregation*.

The authors in [16] and [14] also introduce the concept of *dimension compatibility*. This is described as the assertion where two (2) dimensions or facts - supposedly belonging to different data marts - are respectively compatible when their common information is consistent and there is a characterization of their general properties. These general properties outlined as; *level equivalence*, *dimension equivalence*, *dimension comparability*, and *dimension intersection*, tend to emphasize the notion of dimension compatibility and makes the claim much more expressive.

The *compatibility* property of dimensions is then used as a platform to perform *drill-across queries* over the autonomous data marts, where common information residing in the respective dimensions is used in merging these dimensions. This form of queries also aid in the hierarchical aggregation of instance data during query processing. Their work concludes in illustrating an integration methodology where; *firstly*, data marts are analyzed to identify the compatibility of dimensions; and *secondly*, the checking of semantic matching of compatible dimensions.

In the study in [15], the authors use the work done in [16] and [14] as background work and fundamental intuitions in proposing two (2) different approaches to the problem of integration of multidimensional databases; namely, *loosely coupled integration* and *tightly coupled integration*. They introduce a number of notions and algorithms that are useful in multidimensional integration. Moreover, they stipulate a number of desirable properties that a *matching* between dimensions should satisfy; such as *coherence*, *soundness*, and *consistency*. The algorithms that the authors propose are basically used in identifying common information residing in dimensions of independent data marts, and for deriving a conformed dimension from the merging of the separate dimensions.

Riazati et al. in [80] also propose a solution for integration of data marts where they infer aggregations in the hierarchies of the dimension tables existent in the multidimensional databases. In their work, they attempt to formulate the problem of inferring aggregation hierarchies as computing a minimal directed graph from data, of which these inferred hierarchies are used for roll-up relationships between levels and to ensure the *summarizability* of data. They further use the assertion of *dimension compatibility* introduced in [16, 15, 14] to develop algorithms which in turn are used for the integration of data marts.

3.4.2 Discussion on Approaches of Multidimensional Data Models Integration

The existing approaches to multidimensional schema data integration addressed in [16, 15, 14, 80] explain some important notions that need to be discussed when incorporating several data marts. Their work addresses some of the techniques needed to solving the problem of merging data marts, but fails to handle it from a *model management* perspective.

In this subsection, we address some of the failings of these approaches. In the first place, the previous approaches by the authors in [16, 15, 14] fail to address the issue of *first-order* mapping models. Although some general properties regarding the characterization of *dimension compatibility* seems to handle this concept. As a result, issues of data transformation for dissimilar or general mapping correspondences between attributes of different dimensions across data marts are unable to be expressed during integration.

Secondly, the previous approaches do not lay out a precise schema merge algorithm, which expresses in executable form the *merge* operator in *model management*. This merge algorithm, which is always definitive in finalizing the overall data integration procedure, is non-existent in the literatures studied so far, although descriptions of algorithms for deriving the common information between two (2) dimensions and for merging two (2) dimensions were put forward in [15].

Thirdly, issues of conflict management - in terms of identification and resolution - which are major occurrences during integration are not addressed by the authors in their approach. In [16], some properties that underlie and establish the *dimension compatibility* criteria seem to partially solve the likely conflicts that could be encountered in the dimensions. But these properties in their entirety fail to totally resolve such prominent conflicts during integration.

Fourthly, some technical qualitative requirements that were addressed by the authors in [3] and [75], and which serve to highlight some properties that the global mediated schema should possess seems to be non-existent in the specific approaches for multidimensional data integration attempted by the authors in [16, 15, 14, 80]. These requirements which serve as technical checklists during integration were attempted by the authors in [15], where they proposed of *coherence*, *soundness* and *consistency* as measures for compatible dimension matching. Though these properties seem to partially solve the problem, they are inconclusive in the larger scale of integrating schema and data from fact and dimension tables of data marts, and hence, present a genuine case for our approach of data marts integration.

In summary, our research which uses some major propositions from the work of [75] and [74] seek to handle better the varied issues in relation to integration of multidimensional data

models.

3.5 Summary

In this chapter, we introduced the concept of data integration and explained each of the methodologies of *schema matching*, *schema mapping discovery*, and *schema merge* operations. In the *schema matching* methodology, we discussed the various techniques of *schema-level*, *instance-level*, and *hybrid or composite* form of matching that can be adopted in generating mapping correspondences. We highlighted on and compared some of the generic approaches to schema matching emphasizing on their strengths and weaknesses. In the *schema mapping* methodology, we discussed some of the technical requirements that are needed in guaranteeing the generation of mapping models, and this lead us to discuss some of the structural properties that are necessary in validating mapping models. We discussed various approaches of LAV, GAV, and GLAV mapping models and compared the former two (2) approaches. This discussion also lead us to consider the *Clio* Project, which is a mapping generation platform based on the GLAV mappings. The mapping generation methodology ended with a discussion of a generic mapping model. In the *schema merge* methodology, we discussed some of the technical requirements that must be satisfied for a successful merge operation in the data integration framework. In addressing these requirements, we were lead understand some of the conflicts that are likely to occur in satisfying these conflicts. We discussed these conflicts and how some of them can be resolved in achieving merge data meta-models. We introduced some of the generic merge algorithms that have been formulated, and analyzed them side by side by comparing each of their semantics and method of execution. We discussed some of the recent works that have been studied in the area of multidimensional data models. We address some approaches and the methods as they apply to achieve such integration.

In the next chapter, we will discuss our approach of merging multidimensional data models. We give a general overview and discuss each of the schema matching, mapping models discovery, and our main focus of a merge algorithm. We also address some technical merge correctness requirements and some conflict resolution measures, as part of our integration methodology.

Part III

MERGING MULTIDIMENSIONAL DATA MODELS

Chapter 4

Merge Methodology

Database research in the area of integration continues to receive substantial interest and study through various approaches and methodologies, and based on the various forms of meta-data models that are adopted. In relation to our research methodology for star schema multidimensional data models, and to the level of our knowledge based on the literatures that we reviewed, no attempt has been made in designing a complete merge algorithm for integrating multidimensional star schemas into a single consolidated star schema data warehouse. Furthermore, the proposition of correctness requirements that such an algorithm must satisfy in providing a platform for efficient query processing is non-existent, so far based on the review of research literature we have conducted. In line with these weaknesses, our methodology primarily formulates an merge algorithm which will integrated both the schema structure and instance data into a consolidated data warehouse. This generated data warehouse seeks to answer the correctness requirements for query processing.

In this Chapter, we discuss our novel methodology of schema merging in line with our adopted meta-data model, the *multidimensional star schemas*. We initially address the general overview of schema merging in Section 4.1, and explain the hybrid procedure of finding mapping correspondences in Section 4.2. We discuss the mapping model discovery procedure in Section 4.3 and describe the merge algorithm procedure in Section 4.4. We also describe our new set of qualitative technical requirements and specify conflict resolution measures as part of formulating the merge algorithm. In Section 4.5, we explain some details regarding query processing on multidimensional data models, and we finally summarize the general discussion in Section 4.6.

4.1 Overview of Merge Methodology

Our approach for generating a global data warehouse from independent, but related, multidimensional star schemas extends from the concept of *model management* as earlier introduced in Section 1.2. In line with this meta-data conceptual assertion, we present an overview of our novel integration methodology in three (3) main streamlined procedures; namely, the adoption of *hybrid schema matching*, the adoption of *GLAV mapping models*, and the formulation of *multidimensional merge algorithm*. It will be observed that each of these procedural steps produces an output that serves as an input in the succeeding procedural step, so as to produce the final output of a complete data warehouse in the overall methodology.

4.1.1 Motivating Scenario

We address our methodology for merging the multidimensional data models using *Example* 4.1.1.

Example 4.1.1 *Suppose we have two (2) star schema data marts from an Insurance domain - Policy Transactions data mart, and Claims Transactions data mart - and we have to integrate these data marts into a global enterprise-wide data warehouse, as depicted in Figure 4.1.*

The existence of overlapping attributes will enable the possibility of schema matching, as well as mapping discovery procedures to be performed on the attributes of the fact and dimension tables of these data marts. A schema merge algorithm can then be applied to the mappings to generate the global data warehouse. ∎

In addressing our problem of schema merging for multidimensional data models, we make reference to the scenario in Example 4.1.1, where we have two (2) or more data marts, modelled in *star schemas*, and which are independent but exhibits semantic relationship between the dimensions and facts tables. It can be inferred that though the schema, and maybe instance data representation, in these separate data marts are different, the overlapping sets of real-world entity representations in the dimensions of the data marts seems to present a similarity in that line. Hence, a proposition of integration for the real-world entities in each of the data marts into a single entity in a complete data warehouse is not difficult to achieve.

Using the description in the example, a *Policy Holder* who applies for an insurance policy of a *Policy Coverage* entity, and with a unique *Natural Key* of a *Policy Number* in the *Policy Transactions* data mart, will be the same entity who in the event of a damage to a *Policy Covered Item*, such as a 3-bedroom house, will apply for an insurance claim in the *Claims*

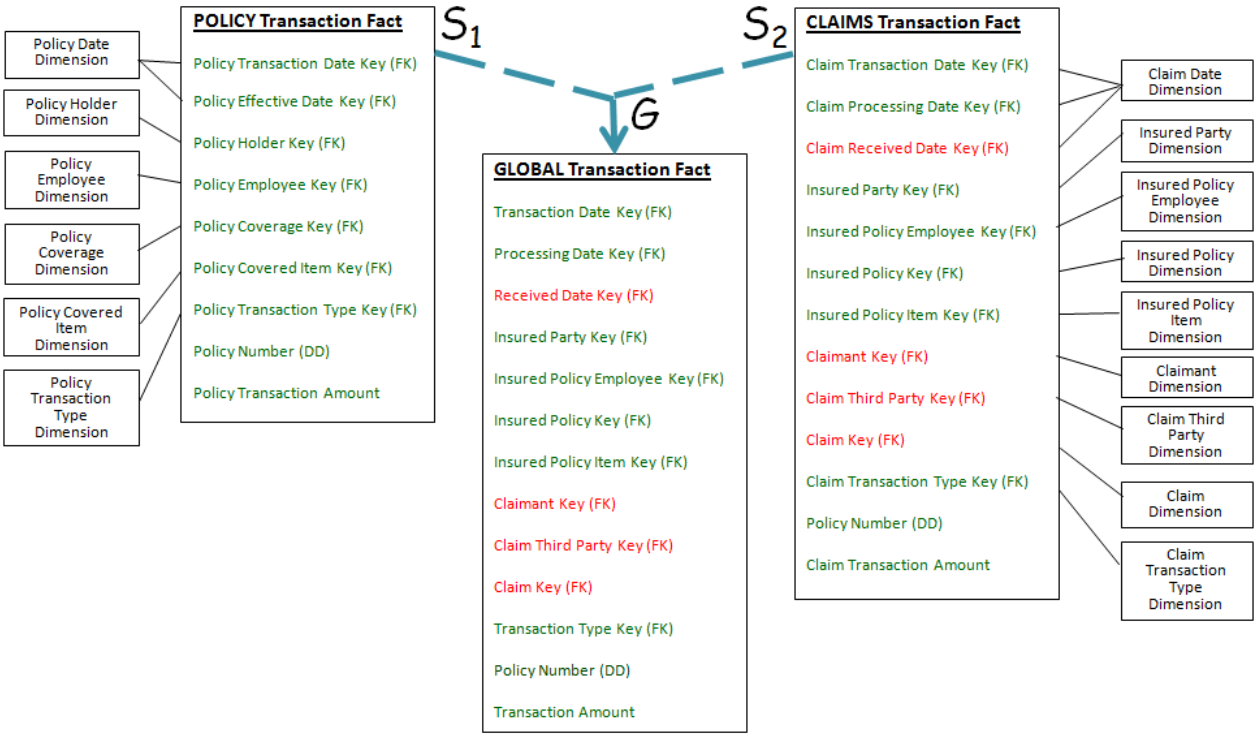

Figure 4.1: *Merging Multidimensional Data Models*

Transactions data mart. In the *Claims Transactions* data mart, this same policy client could be represented as an entity in the *Insured Party* dimension, using the same *Policy Number* for the same insurance coverage entity in the *Insured Policy* dimension. More importantly, there will be the need for processing of claims for the same 3-bedroom house entity being represented in the *Insured Policy Item* dimension.

In the *Claims Transactions* data mart, there could be the presence of some non-corresponding dimensional entities such as *Claim*, *Claim Third Party*, *Claim Received Date*, and *Claimant*; which make up for the data representation in the *Claims Fact Table*. It will be noted that, though these entities will not have any representation in the *Policy Transactions* data mart, the greater amount of overlapping entity representations in both data marts presents a fruitful platform to integrate both the schema structure the instance data into a complete data warehouse.

4.1.2 Description of Merge Methodology

We describe our methodology for merging multidimensional schemas and instance data in a work-plan schedule, being represented in Figure 4.2.

We address an assertion that the overall procedure is not fully automatic, but rather with some form of human interaction in the stages of the Hybrid Schema Matching and GLAV Mapping Models Discovery. This is necessary in terms of validating the results generated at each of these steps, and making these results as vital inputs to the running of the merge algorithm to generate the final single consolidated data warehouse. For instance, at the Hybrid Schema Matching step the user is presented with a set of possible matching candidates for a dimension or fact attribute. Based on the highest mapping correspondence rating and/or the available schema meta-data, the user selects one pair of the mapping correspondence to represent the correct mapping correspondence. Furthermore, on the mapping models discovery the user inputs complex transformation expressions into all forms of *similarity* mappings existing between multi-cardinality mappings, on one hand. On the other hand of *equality* mappings, a complex expression is formulated to aid in data transformation during the execution of the merge algorithm.

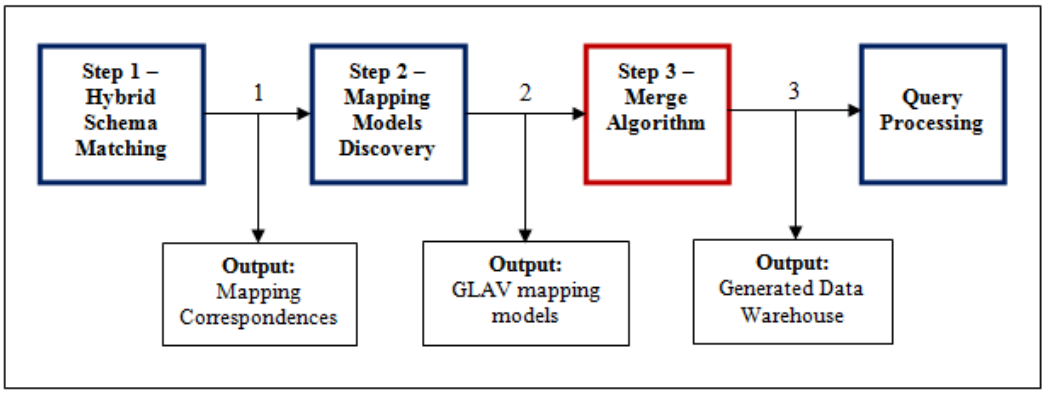

Figure 4.2: *Merge Methodology Procedural Steps*

In arriving at a solution for the core aspect of the merge algorithm for the research methodology, we use earlier algorithm by *Pottinger and Bernstein* in [74]. Their work focused on the theoritical aspect of merging generic models. Furthermore, their work focused on generic models; with elements having semi-structured properties such as, *name*, *id*, *history*; and element relationships in the form of *binary*, *directed*, and *kinded* with cardinality constraints. Our work which subsumes the work in [74], rather delivers a merged solution in a more practical approach by specifically using multidimensional star schema models, together with their associated data, into a single consolidated star schema data warehouse.

As part of our research approach, the concept of *merge* is explicitly explained and differentiated from *union*. On one hand of the high-level schema structure, the single consolidated data warehouse that is generated is free from redundant tables or attributes of the fact and di-

mension tables. On the other hand of low-level instance data, multiple representations of the entities existent in different dimensions are resolved into a unique representation satisfying all corresponding dimensional entities.

This methodology of running query solutions from a single consolidated data warehouse could be achieved alternatively from the approach of *federated data warehousing* [4, 84]. In this approach, a uniform, virtual and logically integrated medium is created for business users to pose their queries, with the underlying data sources scattered all over. This approach presents a drawback where the inefficiencies in network resources connecting these data sources could affect the rate of query processing. Additionally, the need to optimize query processing is affected when data from a number of fact and dimension tables had to be joined or combined in order to present a single solution to a posed query. The form of queries that are generally posed to the integrated medium are usually broken down into sub-queries to be processed on the source data marts, as part of the query execution plan. Hence, query processing is very inefficient as compared to our approach of integrated data warehousing. These issues highlighted and other inherent problems make this federated approach unsuitable in the provision of a uniform platform for the integration of a number of disparate, but corresponding, data marts.

In the next three (3) sections, we discuss the activities that are performed in each procedural step. Under each step, we outline the techniques, the logical intuition, the meta-data element manipulations, or the formulated algorithms and the format of the output expected.

4.2 Step 1 - Schema Matching Procedure

In the *schema matching* step, we use techniques to find mapping correspondences between the attributes of the *fact* and *dimension tables* of the multidimensional star schemas. The approach that we use here is that of a *hybrid* or *composite* methodology, as discussed in Section 3.1.1, where we adopt both the structure of the schemas (fact or dimension tables and their attributes), in a *schema-level matching*; and the instance data contained in the schemas, in an *instance-level matching*. For each of these forms of matching, we adopt various algorithms that understand the semantics of the items - whether schema structure or instance data - used in the matching technique.

We also enforce an ordering for the set of matching algorithms that we adopt. Our adoption of algorithms makes it imperative that one form of matching procedure result, say application of only schema-level matching, becomes an important output as it will be used as input into the other form of matching. This makes the subsequent matching easier, and

rather establishing the results from the previous match.

The ordering technique that we adopt in our schema matching methodology is in the form of first applying the schema-level matching, and then applying the instance-level matching. For each of these individual level matchings, we adopt different algorithms, and also enforce an ordering for the execution of these algorithms. Another feature of our adopted schema matching procedure is the evaluation of the matching candidates prior to their selection. In this regard, we adopt a semi-automatic evaluation where a human is invited to validate the proposed matching candidates for a single correspondence. This step in our procedure is added to avoid any error that the automatic phase of determining matching candidates might propose.

As part of applying these algorithmic techniques to deliver mapping correspondences, we utilize the prior work of the *Clio* Project [67, 1, 43, 42, 28, 68, 30, 37]. We use their proposed algorithms in the schema matching module as our set of algorithms in our paradigm of finding mapping correspondences for multidimensional star schemas. We choose the propositions from this research work because of the following reasons: the ability of the architecture to handle multidimensional data models efficiently, by uniquely identifying fact and dimension tables, as well as their attributes during finding mapping correspondences.

Furthermore, the architecture of the Clio methodology offers a seamless introduction of generated mapping correspondences easily into mapping models. Since this feature is efficiently handled in the architecture, it makes the procedure in the manipulation and translation of correspondences into mapping models easy. Finally, the ability of generating the output of mapping models in executables forms makes their research work very important. This feature in their architecture offers a solution for presenting mapping models in query or view definitions and scripts, as well as executable output files. This makes the feature very important in our methodology, because of the need of our mapping models output to be fed into the next step of running our merge algorithm.

We also enhance the technical manipulations and validations of the output from the generated mapping correspondences in order to achieve better matches for dimension attributes. We discuss the details of each form of matching, together with the algorithms implemented, in the subsequent sub-subsections.

4.2.1 Schema-level Matching

In terms of schema-level matching, the algorithms that we employ are *Lexical Similarity* and *Semantic Name*. These algorithms present a rejection threshold which defines the distance value representing the degree of similarity and offer the specification of a value whereby an

attribute match is accepted or rejected [46].

It will be noted that as part of generating efficient mapping correspondences, the rejection threshold is manipulated appropriately to enable the nomination of possible candidates for element matching. The determination of the rejection threshold and the manipulation of the mapping correspondences are done by the process of inspection. This kind of routine procedure is in the form of an iterative procedure where various adjustments are made. This inspection procedure is done whereby an adjustment of the rejection threshold is not set too high, thereby missing some possible attribute matches in the generated mapping correspondences. On the other hand, the rejection threshold is adjusted not too low which might result in the inclusion of many false positives in the generated mapping correspondences.

We further explain the functionality of each of the algorithms with regard to their techniques in delivering mapping correspondences between attributes of fact and dimension tables.

Lexical Similarity

The lexical similarity algorithm is a technique based on the linguistic form of schema matching in which string *names* and *text* (that is, words or sentences) are used to semantically find similar schema elements [46]. It defines a measure of the degree to which the word sets of two (2) given strings - in terms of *names* or *text* descriptions - are similar, and discovers the maximum length or the maximum weight subsequence of two (2) or more strings that are common to each other. A lexical similarity of 1 (or 100%) would mean a total overlap between the *name* or *text* sets, whereas 0 means there are no common words. Some of the criteria used in determining the similarity based on this algorithm are; the equality of names, equality of synonyms, equality of hypernyms, homonyms, abbreviations, similarity of common substrings, amongst others. The efficiency of this form of algorithmic technique is enhanced with the infusion of domain- or enterprise-specific dictionaries, thesauruses and glossaries which aid the similarity match for the above mentioned criteria.

Semantic Name

The semantic name algorithm is a technique based on the semantic deduction of the schemas and their characteristics. This form of algorithmic technique is reliant on the schema structure and the properties of the elements, and enforces on the varied forms of constraint-based matching where criterion such as; type similarity in relation to data types, value ranges, precision, uniqueness, optionality, relationship types and cardinalities are adopted [46].

Other forms of criterion such as; key properties in relation to key definitions, and ref-

erential constraints are also utilized. The algorithm is made efficient when the metadata of the schema and its constituent elements are clearly defined and are more expressive in the manipulation of data. For example, in terms of relational schemas, the table name, attribute names, attribute key properties and referential integrity, amongst others should be well define as part of the metadata information. Furthermore, the domain knowledge of the type of metadata models being used cannot be overlooked, as these also add to the semantic knowledge needed for matching.

Example 4.2.1 *Following up on Example 4.1.1, we illustrate how a schema-level matching can be achieved. Suppose we want to merge the dimensions of Policy Holder dimension in the Policy Transactions data mart and the InsuredParty dimension in the Claims Transactions data mart.* ∎

We implement the procedural ordering where we first apply the *Lexical Similarity* algorithm to the dimension schemas. This algorithm will deliver initial mapping correspondences in a single- or multiple-cardinality mapping to some of the attributes of the dimensions, as follows:

$$\begin{aligned}
&1.\quad PolicyHolder.PolicyHolderKey \approx InsuredParty.InsuredPartyKey;\\
&2.\quad PolicyHolder.FullName \approx InsuredParty.FamilyName,\\
&\quad\ InsuredParty.GivenName, InsuredParty.CityName, InsuredParty.DistrictName;\\
&3.\quad PolicyHolder.Address \approx\\
&\quad\ InsuredParty.StreetAddress, InsuredParty.EmailAddress;\\
&4.\quad PolicyHolder.PhoneNo \approx\\
&\quad\ InsuredParty.LicenseNo, InsuredParty.PhoneNumber;
\end{aligned} \tag{4.1}$$

These will be based on either, equality of names, equality of canonical name representations, equality of synonyms, similarity based on common substrings, or abbreviations.

On the other hand, the application of the Semantic Name algorithm in the next ordering for the schema-level matching will further refine the set of matches from the initial mapping correspondences. This algorithm will use techniques in the form of data types, constraints, value ranges, relationship types, amongst others to match attributes of the dimensions, as follows:

$$\begin{aligned}
&1.\quad PolicyHolder.PolicyHolderKey[int, PrimaryKey] \approx\\
&InsuredParty.InsuredPartyKey[int, PrimaryKey];
\end{aligned} \tag{4.2}$$

This matching is achieved because of the semantic representations of *Int* data types and *Primary Key* constraints for both attributes on either side of the dimensions of *Policy Transactions* and *Claims Transactions*.

2. $PolicyHolder.FullName[nvarchar(60)] \approx$
$InsuredParty.FamilyName[nvarchar(30)], InsuredParty.GivenName[nvarchar(30)];$ (4.3)

This matching is also achieved because of the semantic representations of *nvarchar(60)* data type for *PolicyHolder.FullName* corresponding better to *nvarchar(30)* for both *InsuredParty.FamilyName* and *InsuredParty.GivenName*. On the other hand, the *PolicyHolder.FamilyName* does not correspond to *InsuredParty.CityName* and *InsuredParty.DistrictName* with semantic representations *char(18)* and *char(20)*, respectively due to the differences in the data types schema constraints.

In the case of the mapping correspondence in (3) and (4), there will be no change in the matching because of the similar semantic representations exhibited in the attributes of both dimensions. These constraints of *nvarchar(80)* for *PolicyHolder.Address* in the *Policy Transactions* data mart and *nvarchar(80)*, *nvarchar(50)* for the *InsuredParty.StreetAddress* and *InsuredParty.EmailAddress*, respectively, in the *Claims Transactions* data mart did not affect the mapping correspondences generated initially. Moreover, constraints of *nvarchar(10)* for *PolicyHolder.PhoneNo* and *nvarchar(15)* and *nvarchar(10)* for *InsuredParty.PhoneNumber* and *InsuredParty.LicenseNo*, respectively, in mapping correspondence (4) did not change the previous outcome.

In addressing and correcting these incorrect mapping correspondences, the instance-level matchings are employed to correct and validate already established correspondences.

4.2.2 Instance-level Matching

The algorithms that we employ in the instance-level matching are *Signature*, *Distribution*, and *Regular Expression*. These algorithms which are based on the instance data contained in the schemas infer on the characteristics, meaning and similarity in the data, as well as the relationship to other data set contained in the schema. Moreover, just as the schema-level matching, the instance-level matching offers an adjustment for the rejection threshold as needed in matching, as well as parametrically adjusting the sampling size (in terms of number of rows) and sampling rate (in terms of percentage) in order to nominate better

candidates for the schema matching. We explain, in detail, the functionality of each of these algorithms in the next subsections.

Signature

The signature algorithm is based on the similarity in the data contained in the schemas and their signature to that effect. The algorithm uses sampled data from the permanent repository configured to the matching system to find relationships where a weighting value is assigned to certain classes of words in the data [46]. This sampling of data is based on the valid values of sampling size and also the rate of the sampling. The determination of match signature is done by clustering according to their distance measure, either by Euclidean distance [23] or Manhattan distance [20].

Distribution

The distribution algorithm discovers mapping correspondences based on the common values in the data contained in the schemas. This algorithm, in comparison to the Signature algorithm, also uses data sampling to aid the discovery function find relationships between attribute data values where the frequent occurrence of most data values for a particular attribute in relation to another attribute the candidacy of matching correspondence [46]. There has been quite an amount of study in this area of attribute matching, ranging from methods such as *A-priori* and *Laplacian* within the domain of machine learning [21].

Regular Expression

The regular expression algorithm is a technique based on the textual or string searches that use regular expressions or pattern matching. A simple regular expression will be an exact character match of attribute data values or of the common substrings contained in the instance data. This algorithm also uses data sampling to aid the discovery function of finding relationships between attribute data values [46].

Example 4.2.2 *Following up on Example 4.2.1, we illustrate how the instance-based matching further achieves a correct and validated matching as the final form of mapping correspondences for the set of attributes in each of the Fact and Dimension tables in the multidimensional star schemas.*

Suppose we orderly apply each of the algorithms of Signature, Distribution and Regular Expression, a better set of mapping correspondences can be attained for matches (3) and (4),

as follows: ∎

$$\begin{aligned} &3.\ \ PolicyHolder.Address \approx InsuredParty.StreetAddress; \\ &4.\ \ PolicyHolder.PhoneNo \approx InsuredParty.PhoneNumber; \end{aligned} \tag{4.4}$$

These matching are achieved because of the instance data values contained in the attributes of the dimensions. For example, in the mapping correspondence (3), the *PolicyHolder.Address* attribute will contain data values, such as; *3938 Baywood Drive*, *1178 Flora Ave.*, and *7179 Golden Rain St.* These data values will correspond better to that of *InsuredParty.StreetAddress*, such as; *4082 Roslyn St.*, *6481 Hastings Drive*, and *748 Whitehall Drive*. On the other hand, data values from *InsuredParty.EmailAddress* will not suit such a correspondence, and as a result validate discarding that correspondence. Examples of such data values will be; *amartens@cybserv19.com*, *dpetterson@vipce2k.com*, and *jtausig@fitexes.com*.

With regard to the mapping correspondence (4), data values contained in the *PolicyHolder.PhoneNo* attribute will be; *+1 (514) 747-4481* and *+1 (604) 688-9745*. These data values will correspond better to that of *InsuredParty.PhoneNumber*, which will also contain values, such as; *688-555-0130* and *908-555-0190*. Correspondence from *InsuredParty.LicenseNo* will be discarded based on the data values contained in the attributes. Examples such data values will be; *HJEK 253*, *MKED 457* and *JKSW 452*.

4.2.3 Schema Matching – Methodology Procedural Step Output

The output of this procedural step is the generation of a set of mapping correspondences that exist between the attributes of facts and dimension tables, and establishes a similarity relationship between these attributes. These mapping correspondences are represented in a single or multiple cardinality associations on either side of the set of attribute(s), and form the basis for the formulation of logical mapping assertions in the next procedural step of mapping model discovery in Step (2).

4.3 Step 2 – Mapping Model Discovery Procedure

In the mapping model discovery step, we adopt a set of mapping formalisms that expresses assertions on the elements of the schemas. The fundamental platform for the formulation of

logical assertions in the mapping model discovery is dependent on the prior work of finding mapping correspondences. It can be referred from Section 3.2.4 that GLAV mapping models are a combination of LAV and GAV mapping models, where it enforces on the strengths of both mapping models and suppresses on the weakness of the both mapping models. We discuss much of the expressiveness of this model in Section 4.3.1.

In this subsection, we explain the major definitions of this methodology procedural step. Firstly, we adopt GLAV mapping models and describe the features that are useful in achieving the intended results. Secondly, the various forms of manipulations available in these GLAV mappings and the capabilities of conveying their output forms in, for example, executable formats of view definitions, query scripts, amongst others.

4.3.1 GLAV Mapping Model

The GLAV mapping model combines the expressive power of both the LAV and GAV mapping models. Some of the processes that are undertaken in this mapping model involve the definition of complex transformation formulas for multiple attributes on one side of the integrating data mart corresponding to a single attribute on the other side. Moreover, the inclusion of non-corresponding attributes in the global schema attribute set is an important characteristic of this mapping model. Other forms of expressiveness is the ability to define the type of mapping relationship in terms of cardinality (i.e. equality or similarity), and definition of a general attribute and data type representation for the attributes involved in the mapping relationship.

Additionally, enforcing the ability of the mapping model to generate executable queries in the form of view definitions or query scripts in the form of native SQL. This makes the mapping model well defined, flexible and expresses the ability to describe the relationships between the source elements during the merge algorithm procedure. It also aids in the generation of metadata definition as part of the execution of the merge algorithm.

4.3.2 Capabilities and Manipulations of GLAV Mapping Models

There are various manipulations that the GLAV mapping model offers; we summarize a few of them:

1. It is a mapping language that facilitates the (semi-)automatic generation of schema mappings;

2. The composition of sequential mappings that enables the re-use of mappings when the schemas are different or change;

3. The semantics of such a mapping and its data exchange capabilities offers a data translation from one schema to another based on the mapping specifications;

4. The mapping language expresses the capabilities for runtime executables, for example, to generate view definitions, query answering, and generation of XSLT transformations, amongst others;

5. Its semantics makes it to be easily understood and manipulated by mapping tools, for example, the InfoSphere Data Architect, BizTalk Mapper, amongst others;

6. The mapping language offers a platform where there can be generation of codes based on the mappings; as in the case of efficient queries or transformations in various languages (e.g. native SQL) can implement the formulated mappings;

Example 4.3.1 *We follow up from Examples 4.1.1, 4.2.1 and 4.2.2, where the correct and validated mapping correspondences have already been generated and established. Suppose we want to model the mapping relationships between the attributes of both PolicyHolder and InsuredParty dimensions based on the GLAV mapping formalism. The following datalog query is generated which is later expressed in executable forms.*

Dim_GlobalDimension *(InsuredPartyKey, InsuredPartyID, City, District, PostZipCode, Province, Country, Occupation, OccupationCategory, AgeRange, DateOfBirth, MaritalStatus, Gender, IncomeBand, AnnualIncome, HomeSize, EmailAddress, HomeOwnerFlag, InsuredPartyName, HomeAddress, PhoneNumber, FaxNumber, Region, OccupationForm, CarOwnerIndicator) :=*
Dim_PolicyHolder *(PolicyHolderKey, PolicyHolderID, FamilyName, GivenName, Address, CityName, DistrictName, PostCode, ProvinceState, Country, Employment, EmploymentType, AgeBand, BirthDate, MaritalStatus, Sex, IncomeBand, YearlyIncome, HouseholdSize, DayPhoneNumber, EveningPhoneNumber, FacsimileNumber, Email, HouseOwnerFlag),*

Dim_InsuredParty *(InsuredPartyKey, InsuredPartyID, FullName, ApartmentSuite, StreetAddress, City, District, Region, PostZipCode, Province, Country, Occupation, OccupationForm, OccupationCategory, AgeRange, DateOfBirth, MaritalStatus, Gender, IncomeBand, AnnualIncome, HomeSize, PhoneNumber, FaxNumber, EmailAddress, HomeOwnerFlag, CarOwnerIndicator).* ∎

4.3.3 Mapping Discovery – Methodology Procedural Step Output

The output of this procedural step is the generation of GLAV mapping models in executable forms, where there is the definition of complex transformation expressions. The output, which also offers the generation of query scripts that can be used in translating data from the multidimensional star schemas into the global data warehouse, is used as one of the inputs the schema merge procedure in Step (3).

4.4 Step 3 – Multidimensional Data Model Merge

In the schema merge procedural stage, we formulate an algorithm to generate our expected global data warehouse. This step involves the incorporation of the mapping model and the multidimensional star schemas, together with their semantic metadata, and these in line with other processes conflict resolution and satisfaction of correctness requirements, finalizes the overall integration procedure.

In this subsection, we outline and describe some qualitative technical correctness requirements that the merge output should satisfy in Section 4.4.1, and then follow-up with a description of some likely conflicts that can arise within our framework of integration with multidimensional star schemas. Consequent to that, we propose some measures of resolving these conflicts in Section 4.4.2. We also describe our proposed formulated merge algorithm designed to integrate multidimensional star schemas into a global data warehouse in Section 4.4.3. We summarize the overall algorithm in line with the satisfaction of the merge correctness requirements in Section 4.4.4 and describe the computational complexity of the merge algorithm in Section 4.4.6.

4.4.1 Qualitative Merge Correctness Requirements

The global data warehouse that is generated as a result of the implementation of the merge algorithm needs to satisfy some requirements to ensure the correctness of the queries that

would be posed to it. These qualitative technical requirements give acceptance to the validation of properties that the global data warehouse schema should exhibit and seeks to underscore some of the standpoints to note with regard to the merging of multidimensional star data models.

In this sub-subsection, we outline some of these correctness requirements that will serve as guidelines during the formulation of the merge algorithm, and validate the accuracy of the output in the algorithm. *Pottinger and Bernstein* in [74] outline and describe some set of technical requirements that generic meta-models should satisfy during merging of their elements into a global schema. These requirements served as validation criteria which enforces on generic meta-models as part of a merge algorithm design and implementation.

Drawing on the major propositions in these defined requirements by the authors in [74], we performed a gap analysis on these propositions, and describe our set of correctness requirements in relation to merging of multidimensional star schemas. These technical requirements may have comparable similarities to that of the requirements already proposed in [74], but we will substantiate them better in terms of star schemas. Moreover, we address that these requirements specifically characterize the properties of the elements of our chosen meta-models schema, and also the instance data it contains.

It will be noted that these technical requirements have been validated to represent the set of criteria for merging multidimensional data models, especially in terms of star schemas, based on the experimental results as will be discussed in Chapter 6. The formulated set of queries pose to the global data warehouse delivered tuple answers that represented the correct set of answers as the same queries would have been posed to the independent multidimensional star schemas.

Outlined below are the *Merge Correctness Requirements* (MCR) stipulated for the formulated algorithm which has been elaborately described in Section 4.4.3;

Dimensionality Preservation

For each kind of dimension table attached to any of the integrating fact tables, there is a corresponding or representative dimension in the merged Fact table. This is made possible because of the non-redundant and all-inclusive attribute values giving rise to the *Foreign Key constraint* satisfiability in the merged Fact table.

Measure and Attribute Entity Preservation

All *facts* or *measures* of the attribute values in either of the integrating *fact tables* are represented in the *merged fact table*. Additionally, all other attribute values in each of the *dimension tables* are represented through an *Equality* or *Similarity* mapping. Where the mapping correspondence is an *Equality mapping*, there is an attribute in the merged dimension table that uniquely represents the integrating dimension attribute. In the case of a *Similarity mapping*, there is a set of attributes or a general attribute - based on a complex transformation expression - to represent that attribute from the integrating dimension table. Finally, there is an automatic inclusion for non-corresponding attributes in the merged fact or dimension tables for their necessity of not introducing any redundancy in the final merge data warehouse.

Slowly Changing Dimension Preservation

For *Slowing Changing Dimensions* (SCDs) where a dimensional entity has multiple representations, the merged dimension for such an entity should offer an inclusion of all the instances of the dimensional changes in their right order of changes. Hence, any attribute that makes up for the dimensional changed should be included in the merged dimension. Furthermore, all tuples from the resultant dimension changes should be uniquely represented in the merged data warehouse for fact and dimension tables.

Attribute Property Value Preservation

The merged attribute should preserve the value properties of the integrating attributes, whether the mapping correspondence is an *Equality* or *Similarity* mapping. *Equality mapping* should be trivially satisfied by the *UNION* property for all equal attributes. For a *Similarity mapping*, the transformation or complex representation should have properties encompassing enough to satisfy the attribute property value of each dimension attribute.

Tuple Containment Preservation

The merged data warehouse should offer the containment of all unique tuples as they are valuable in returning correct answers to queries posed. This makes the preservation of all *Surrogate Keys* to dimensional entities. In cases where there are conflicts in key representation, the merge algorithm enforces a modification of the based on the chosen *Preferred Model* and reassignment to the conflicting tuple.

4.4.2 Conflict Resolution – Surrogate Keys, Entity De-duplication

The integration of meta-data models are generally coupled with different forms of conflicts, and these are resolved through different propositions based on the semantic inference of the chosen meta-data models.

In our integration approach of using multidimensional data models as our chosen meta-meta model, we identify some conflicts that are the likely to be encountered and propose some measures of resolving these conflicts;

Semantic Conflicts for Same Identifier Dimensional Entities

These conflicts arise as a result of the multiple semantically unrelated representations of the same real-world entity in the merged dimension by the *same* identifier. This occurrence could be from the scenario where we have different data marts that are very much semantically unrelated, as in the case of company mergers and acquisitions. In this perspective, there could be the possibility of different entities of the same kind having the same *surrogate key identifier* in their individual dimensions. This calls for a resolution of the multiple representations of same *surrogate key* identifiers for these dissimilar real-world entities, as explained in the example illustration.

Example 4.4.1 *Suppose we want to merge the employee dimensions from the multidimensional star schemas, as in the case of dissimilar real-world entity representations, into a single dimension in the data warehouse. The first data mart, say Policy Transactions, has the dimension Policy Employee; whilst the second data mart, say Claims Transactions, has the dimension Insured Policy Employee.*

In such an integration procedure, if it happens that different entities have the same identifiers of a surrogate key in both dimensions, there is the need to resolve such a conflict before incorporating both representations in the merged dimension. A resolution measure outlined in our merge algorithm in Section 4.4.3 will be; to preserve the surrogate key identifier in the preferred data mart and reassign a new surrogate key identifier for the other data mart(s). ∎

Semantic Conflicts for Different Identifier Dimensional Entities

The second perspective of likely *Semantic Conflicts* arises as a result of the multiple semantically related representations of the same real-world entity in the merged dimension by

the *different* identifiers. The occurrence will be illustrated in the scenario where we have different data marts that are semantically closely related, as in the case of the merger of different data marts into a data warehouse for a single company or organization. This form of merging leads to different representations of *surrogate key identifiers* for the same real-world entity from different dimensions in the merged dimension. Following up from the scenario in the illustration above of employee dimension merging, a proposed resolution measure, as described in the merge algorithm, will be to perform a straightforward *de-duplication* of the conflicting entities of employee by preserving the entity from the preferred data mart, say the *Insured Policy Employee*, as the sole representation of the real-world entity in the merged dimension.

Descriptive Conflicts for Differing Attribute Data Types

Another form of conflicts that we deal with is that of *Descriptive Conflicts* which occur as a result of existence of different attribute property values, from the integrating attributes, for the merged attribute. We explain this form of conflict in the Example 4.4.2:

Example 4.4.2 *Suppose we have an instance where the HouseOwnerIndicator attribute in the Policy Holder dimension table in the Policy Transactions data mart possesses a nchar(1) data type, whilst the HomeOwnerIndicator attribute in the Insured Party dimension table in the Claims Transactions data mart also possesses a Bit data type. Combining these attributes into a merged attribute of, say HomeOwnerIndicator, will force the merged attribute to possess a data type property value being the UNION of both integrating attributes.*

We resolve these forms of conflict also in the merge algorithm, where most often we use the predefined set of attribute property values. In this scenario, we resolve the conflict by representing the merged data type for the merged attribute by nvarchar(10) to represent both attribute property values from the integrating attributes. ∎

4.4.3 Merge Algorithm

In this sub-subsection, we present our algorithm based on the multidimensional star schemas and the proposed mapping model, to generate a global data warehouse. The algorithm is basically formulated to merge the schema structure and the instance data of the supposed star schema data warehouse.

The general procedure of the algorithm starts with an initialization of table and attributes of the start schema data warehouse. The next step, in Step (2), is the design of the schema structure of the fact and dimension tables, together with their corresponding attributes set. In Step (3), we define the set of attributes for the merged tables, which come from the mapping relationships in the form of *Equality* and *Similarity* mappings. All other non-corresponding attributes are added to the merged table at this stage of the algorithm. In Step (7), the attribute properties (data type, field size, amongst others) are determined to complete the overall schema structure of the merged data warehouse. In Step (10), the generated dimension schemas are populated with instance data from the incorporating dimension tables from the source star schema data marts. Conflicts of surrogate keys and dimensional entity de-duplication are resolved. In Step (11), the instance data from the incorporating fact tables are populated into the merged fact table.

This algorithm is designed to run in a polynomial time in the worst-case, and its computational complexity is analyzed in Section 4.4.6. Running the algorithm terminates and generates an output in the order of seconds, and this is analyzed in terms of a low data complexity. In cases of a large number of tables and attributes contained in the schema structure of the multidimensional star schemas, and/or also a huge amount of data contained in each of the fact and dimension tables, then an appreciable increase in the level of data complexity is attained leading to an overall increase in the complexity of running the algorithm. This might lead to the order of minutes or hours in generating the merged star schema data warehouse. The details of the algorithm are displayed in Figures 4.3, 4.4, 4.5.

4.4.4 Merge Algorithm Summary

The formulated merge algorithm described here satisfies the technical *Merge Correctness Requirements* (MCRs) stipulated in Section 4.4.1. We summarize the adherence of these requirements in line with the step-wise lay out of the algorithm, as follows:

- Step (2) satisfies *Dimensionality Preservation*. As Fact Tables represent the base tables of data marts, there is the iteration of Fact Tables from each of the integrating data marts to form the Merged Fact Table.

- Step (3) satisfies *Measure and Attribute Entity Preservation*, where all the attributes contained in the Fact or Dimension Tables are represented in the Merged Table (Fact or Dimension) through *Equality* or *Similarity* mapping.

Algorithm 1: *Multidimensional Schema Merge Algorithm*

- **Input**:

(a) A set of *star schema* data marts; A and B;

(b) A set of *first-order* logic mapping model;

(c) An optional designation of one of data marts, *A* or *B* as the *preferred data mart*;

(d) A predefined set of suitable *Attribute Data Types* for Merged Attributes.

- **Output**:

(a) A global multidimensional star schema free of *duplicate* and *redundant* data.

(b) A meta-data constituting a combined data definition of the data marts and the global data warehouse.

- **Notational Conventions**:

(a) DW – Global Data Warehouse;

(b) F_A and F_B – Fact Tables of data marts A and B, respectively;

(c) D_A and D_B – Individual Dimension Tables of data marts A and B, respectively;

(d) $FAtt_A$ and $FAtt_B$ – Attributes of Fact Tables in data marts A and B, respectively;

(e) $DAtt_A$ and $DAtt_B$ – Attributes of Dimension Tables in data marts A and B, respectively;

(f) MAP_{FactAB} – Mapping model correspondences in between Fact Tables A and B;

(g) MAP_{DimAB} – Mapping model correspondences in between Dimension Tables A and B;

(h) F_{AB} – Merged Fact Table;

(i) D_{AB} – Merged Dimension Table;

(j) Met_{AB} – Metadata containing data definition of the data marts and the global data warehouse.

- **Procedure**:

(1) ***Initialization***: Initialize the merged DW, G to *NULL* with empty set of Fact and Dimension Table attributes.

(2) ***Generate Merged Table***: Generate the attribute composition of the Fact Table, as follows:

a. Using MAP_{FactAB} and the Correspondences between the attributes; iterate through all the Fact Tables, F_A and F_B to find common attributes and combine them to generate the merged F_{AB}.

b. If there is no correspondence, exit the iteration, exit the procedure and output G.

Figure 4.3: *MultiDimensional Merge Algorithm – Part 1*

(3) ***Merged Table Attribute Relationships***: For each set of corresponding attributes in F_A and F_B, check for the type of mapping [*Equality*, *Similarity*] existing between them.

a. Represent the Fact Table Name by an *Element* from the *Table Name*;

i. Defined in the mapping correspondence, MAP_{FactAB}; if not *NULL*

ii. From the *Preferred Data Mart*.

b. If *Equality Mapping*, represent the merged attribute by the;

i. Attribute Name defined in the mapping definition; if not *NULL*

ii. Preferred Attribute Name from F_A or F_B being the *Preferred Data Mart*; if not *NULL*

c. If *Similarity Mapping*, represent both attributes in the merged Fact Table by a general Attribute Name, as defined in MAP_{FactAB}, with a similarity correspondence or computed function to each attribute in $FAtt_A$ and $FAtt_B$.

d. For non-corresponding attributes in either Fact Tables, include each of them in the merged Fact Table, F_{AB} maintaining their individuality, *if and only if they will not result in a duplicated and redundant attribute*.

e. The Fact Table Name and Attribute Names for the merged F_{AB} is thus created.

(4) For each merged F_{AB}, iterate through all the set of Dimension, D_A and D_B belonging to the constituent Fact Tables, F_A and F_B using the *Foreign Key* constraint linking each Dimension Table to a Fact Table.

(5) Repeat Step (2.a) for all correspondences between Dimension tables for their merging, using MAP_{DimAB}.

(6) Repeat Step (3) for all correspondences between Dimension tables for their attribute relationships, using MAP_{DimAB}, $DAtt_A$, $DAtt_B$.

(7) ***Merged Table Attribute Properties***: For each merged element (attribute) in F_{AB}, $FAtt_{AB}$; the value of the properties [*Data Type, Field Size, Precision, Scale*] is a *UNION* of the combining properties; and determined as:

a. defined in the mapping correspondence; if not *NULL*

b. selected from the Predefined Attribute Data Type; if not *NULL*

c. defined from a Preferred attribute data type (Data mart) property value; if not *NULL*

Figure 4.4: *MultiDimensional Merge Algorithm – Part 2*

d. If in any of the definitions (a) - (c) there is an existence of more than one property value, then data type is chosen as the ordering above.

(8) Repeat Step (7) for all attribute properties of Dimension tables, using D_{AB}, $DAtt_{AB}$

(9) The *Logical* and *Physical* merged DW, G of a merged tables, is thus created.

(10) ***Dimension Tables Data Population***: Populate each of the dimensions in the merged data warehouse taking note of;

a. *Surrogate Keys* – where conflicts arise for seemingly very much unrelated data marts, retain the *Surrogate Keys* for the chosen *Preferred Data Mart* and reassign the conflicting attribute for the other data mart using the *Natural Key* as an identifier.

b. *Dimensional Entity De-duplication* – where there exists more than one representation for a dimensional entity for seemingly very much related data marts. Perform a naïve de-duplication *using the Natural Key as an identifier*, where an update is performed on the *Surrogate Key* of all *Non-Preferred Data Marts* in relation to the *Surrogate Key* of the *Preferred Data Mart*; after loading the Fact Table records in Step (11).

c. *Slowing Changing Dimensions* (SCDs) – If there exists SCDs for multiple representations for a dimensional entity, use the *Natural Key* and other additional dimensional attributes to uniquely identify the order of changes.

(11) ***Fact Table Data Population***: Populate the Fact table using the *New Surrogate Key, Old Surrogate Key* and *Natural Key* of the entities in the Dimension tables to load the individual Fact Table Records.

Figure 4.5: *MultiDimensional Merge Algorithm – Part 3*

- Step (4) satisfies *Dimensionality Preservation*, where each of the Dimension Tables linked to each Fact Table already merged iterated for merging.

- Step (7) satisfies *Attribute Property Value Preservation*, where there is a representation of the value properties of attributes (Data Type, Field Size, amongst others) of each of the Fact or Dimension Tables from the integrating data marts.

- Step (10) satisfies *Slowly Changing Dimension Preservation* and *Tuple Containment Preservation*, where all multiple entity representations from the different data marts are included in the merged dimensions. Subsequently, the different representations of a single entity in a particular integrating dimension are also represented in the merged dimension.

- Step (11) satisfies *Tuple Containment Preservation*, where tuple data values from each of the data marts are populated in merged data warehouse as a representation of each entity either in the Fact or Dimension Table for query processing.

4.4.5 Schema Merge – Methodology Procedural Step Output

The output of this procedural step is the creation of a global data warehouse that combines both the schema structure and the instance of the integrating multidimensional star schemas. This global data warehouse provides the stage where answers to intended queries that are separately processed on each of the independent data marts, are computed correctly with the same or similar kind of queries being posed to it. This procedural step, which also summarizes the overall integration methodology, produces a metadata definition for the mapping relationships between the attributes of the global data warehouse and that of the multidimensional star schemas.

4.4.6 Computational Complexity of the Merge Algorithm

The algorithm presented in the previous sub-subsection, Section 4.4.3, is projected to run with a low worst-case time complexity in a polynomial time as earlier stated.

In the initialization step in Step (1), a running time of $O(n)$ is needed to initialize the global fact table and its constituent dimension tables. In the Step (2), a derivation of the

merged fact table involves the iteration through each of the fact tables from the individual data marts, as well as the iteration through each of the attributes of each fact table to find common correspondences, using the mapping. These iterations will require a computation time of $O(n^2 \log m)$ for the number of fact tables n and the number of attributes m, contained in each fact table.

Taking into consideration Step (4) and Step (5) - being a repetition of Step (2) - for each of the dimension tables, there is an overall time complexity of $O(k + n^2 \log m)$ for both fact and dimension tables iterations. With regards to the executions in Step (3) and its repetition for the dimension tables in Step (6), the derivation of attribute relationships will require a complexity of $O(k + n)$ for the set of corresponding attributes n and the set of non-corresponding attributes k. In finding the attribute properties for each of the generated merged tables in Steps (7) and (8), a running time of $O(k + n)$ is required for both fact and dimension tables. Similar iterations are performed in Steps (10) and (11), which require worst case running times just as in previous steps.

In general, an overall worst case complexity of $O(n) + O(k + m) + O(k + n^2 \log m)$ is required in executing the merged algorithm to generate a global data warehouse.

4.5 Semantics of Query Processing on Multidimensional Data Models

The type of queries that are processed on multidimensional data models are the category based on Online-Analytical Processing (OLAP). OLAP queries generally focus on fast answers to ad hoc queries in the form of aggregating the warehouse data. The use of OLAP tools for such query processing has primarily been based on performance issues where large static and historical data are made available to business users for analytical decision-making.

There are a few problems that are inherent with OLAP query processing, and these are addressed as follows. On the one hand, is the problem of deficient data that arises from missing data values and also imprecise data values of varying extents. It will be noted that in our approach of merging different schema, as well as data, the possibility of having missing data values from any of the star schemas is highly probable. This resultant effect of missing data will impact on some of the data values generated from the queries posed relating to the affected dimensional attributes. The varying granularities caused by the different degrees of precision in the data values from the combination of data from different *star schemas* also exposes a non-uniform representation of the combined data values needed for analytical

reporting.

On the other hand, the problem of imperfections innate in the hierarchies of dimensional tables also places an overhead cost on query processing for multidimensional data models. Hierarchies enable *drill-down* and *roll-up* in the aggregate data, and as a result multiple hierarchies in a particular dimensional entity are supported for different aggregation paths within the dimension. Different forms of *strict* and *non-strict* hierarchies are exhibited in the dimensional entities of multidimensional data models. *Strict* hierarchies exhibit a phenomenon where a dimension item or child level element has only one parent level element enforcing a constraint restriction on the data values that are rolled-up during aggregation. *Non-strict* hierarchies also exhibit a phenomenon where a dimension item or child level element has several elements at the parent levels, thus allowing flexibility in the kinds of data values aggregation based on the data analysis conducted.

Pedersen et al. in [72], propose some requirements that a multidimensional data model should satisfy in order to fully support OLAP queries. These are outlined as; explicit hierarchies in dimensions, multiple hierarchies, support for aggregation semantics, non-strict hierarchies, non-onto hierarchies, non-covering hierarchies, symmetric treatment of dimensions measures, many-to-many relationships between facts and dimensions, handling change and time, handling different levels of granularity, and handling imprecision. These requirements give insights into how OLAP tools manage the raw data values retrieved from the permanent repository, and how they express the data values in a more analytical format as required by business users.

In our approach of query processing, we align our mode of query processing in relation to the proposition in [72]. These forms of queries will be efficient enough because of the adoption of star schema as the multidimensional data model, which will offer a platform for basic SQL *star-join optimization* - in the fact and dimension tables - during the pulling of data values for analytical representation. The ability of structured cube modelling of each of the dimension elements by OLAP representations offers the medium for the individual hierarchies in the dimensional entities to be captured explicitly, and consequently enables the flexible control of business users in navigating through the cubes. These hierarchies and their data manipulations are captured using either, *grouping relations and functions*, *dimension merging functions*, *roll-up functions*, *level lattices*, *hierarchy schemas and instances*, or an *explicit tree-structured hierarchy* as part of the cube.

Different forms of aggregations are computed in the approach of query processing on the generated data warehouses. These aggregations are made possible because of the defined hierarchies established in the dimensional entities. The aggregations are represented in func-

tions such as addition computations, average calculations, and constant functions through an OLAP operation of *summarizability*. *Summarizability* is a conceptual property of multidimensional data models where individual aggregate results can be combined directly to produce new aggregate results. This property enhances processes of easily *drilling-down* and *rolling-up* data values without much cost in data transaction processing from the permanent repository.

In summary, an assertion is established that query processing in the generated data warehouse is primarily based on OLAP technology. This mode of query processing highlights issues such as, the imperfections in missing and imprecise data values as a result of the merging of different hierarchies of different dimensional entities. Additionally, different forms of - strict and non-strict - hierarchical representations in the merged dimensions are also addressed. Other issues of *aggregations* and *summarizability* also expose the ability to present query solutions to business users in a much more uniform, flexible and user-controlled manner.

4.5.1 Computational Complexity & *Correctness* of Query Processing

In terms of deriving correct answers to queries posed to the generated data warehouse, the complexity of computing the query result is the same as the complexity of recognizing the tuples in the query result. A low amount of computational time is considered as the *combined complexity*, which is also in *polynomial-time*; and follows a worst-case complexity just as in the case of the running of the algorithm. This polynomial time complexity of running query processing on the generated data warehouse is depicted in the evaluation results in Chapter 6, where data values to posed queries are generated in the least amount of computational time.

The combined complexity takes into account the *data complexity* and *query complexity* in the evaluation of a query answering where both the query and the instance data are marked as part of the input, and as a result can be considered as variables in the function. The *data complexity* of query answering is the complexity of evaluating a query on the database instance, *when the query is fixed*, and hence we express the complexity as a function of the size of the database instance; supposedly of the large volume of instance data contained in the multidimensional star schema global data warehouse. The *query* (expression) *complexity*, on the other hand, is the complexity of evaluating a query, *when the size of the database is fixed*, and we express the complexity as a function on the size of the query definitions. Since query complexity is highly sensitive to the syntax of queries, we generally would rather refer

to it as *expression complexity* [71, 89].

Formally, we will express the combined complexity formally in mathematical terms, as in Equation 4.5:

$$\{D, Q, t \quad | \quad Q \in C(L), \quad t \in Q(D)\} \tag{4.5}$$

where;

- Q is the *Query* to be evaluated;
- $C(L)$ is the *Type of Query Class*;
- D is the *Multidimensional Database*;
- t is the set of *Tuples* for the generated query solution.

With regards to the algorithm enabling the generation of correct data values to queries posed to the data warehouse, we explain the *correctness* of the algorithm and substantiates on the worst-case polynomial-time complexity of computing correct answers to posed queries. We give a detailed proof in Appendix A, where we provide a sketch outlining the *soundness* and *completeness* properties of the formulated merge algorithm.

4.6 Summary

In this Chapter, we presented a general overview of the merge methodology; which had its steps broken down into three (3) main procedural methods of schema matching, mapping model discovery and schema merge. We further discussed the activities that are performed in each of the procedures separately; cutting across techniques and manipulations of processes, algorithm formulation, specification of technical requirements, and specification and resolution of some likely conflicts. Other discussions focused on the computational complexity of the merge algorithm, and of query processing on the data warehouse. We discussed the semantics of OLAP query processing that are performed on the generated data warehouse, and also discussed issues of *dimensional hierarchy*, *data aggregation* and *summarizability* which are necessary in handling multidimensional data.

In the next Chapter, we discuss the implementation of the merge methodology. We first describe the experimental setup, covering the data sets and their composition, and the necessary tools used in the implementation; describing their manipulations and configurations. We also describe how we implement the afore-mentioned streamlined procedures in this Chapter which will lead us to the outlined expected outputs of mapping correspondences, discovered mapping models, and the generated merged data warehouse.

Chapter 5

Experimental Setup and Implementation

In line with our novel integration methodology discussed in Chapter 4, which detailed the various techniques, algorithms and processes needed in producing the global data warehouse output in the theoretical sense, we describe the practical methods and activities that we performed. These implemental activities and procedures lead to the achievement of the output sought for. With regard to this assertion, we explain our implementation corresponding to the proposed methodology.

In this Chapter, we discuss the experimental data set we used in the implementation in Section 5.1, and describe a graphical representation of the overall experimental implementation in Section 5.2. In Section 5.3, we explain how we performed the schema matching and mapping discovery methodologies. In Section 5.4, we discuss the implementation of the merge algorithm; taking note of the entity classes, business logic classes, programme control, as well as, other database procedures that were scripted and applied. We then discuss the query processing tasks that were performed in Section 5.5, and a summary of the overall discussion in this Chapter in Section 5.6.

5.1 Experimental Data Sets

In this subsection, we describe the data sets that were used during the experiments in the implementation phase of the methodology. It will be noted that in our paradigm of research, the methodology proposed could either work with very independent data marts, as in the case of different companies merging, or semantically related data marts, such as the in Figure 4.1, where both data marts are modelled for specific departments in the same company.

Each of these data marts had their schema well structured with key constraints and referential integrity, and with their accompanying instance data; making the data sources free of inconsistencies or noisy data. A critical note of caution had to be adhered to, in that, the existence of inconsistency or noisy data in schemas prime for integration tend to bring an overhead cost delivering a final global data warehouse inherent with these structural errors or noisy data. These defects could impact on and affect the processing of queries and the presentation of correct results to business users. In cases where there are inconsistencies or noisy data, a data cleaning process or data quality procedure will have to be performed to eliminate all such anomalies from the multidimensional star schemas [12, 33].

We implemented our methodology using data sets from two (2) different domains; namely, *Insurance* and *Transportation Services*. We give a graphical representation of these data sets in Appendix C.

With regards to the Insurance data set, we used two (2) multidimensional star schemas. These were *Policy Transactions* and *Claims Transactions* data marts. We describe the content of these data marts briefly:

Insurance Policy Transactions Data Mart

The *Policy Transactions* data mart contained seven (7) Dimension Table schemas which had their key constraints referentially connected to a single Fact Table schema. This fact table schema had a *Degenerate Dimension* (DD) attribute of a *Policy Number* and a fact or measure of *Policy Transaction Amount*, aside the foreign key representation of each of the attached dimension tables. The fact table schema contained instance data of *3,070* tuple rows of data, where as each of the dimension tables contained adequate rows to make the experiment and its results much more definite; with the *Policy Holder* dimension containing the highest amount of tuple rows of *18,485* alongside *24* set of attributes to describe it.

Insurance Claims Transactions Data Mart

The *Claims Transactions* data mart also contained ten (*10*) Dimension Table schemas with each of their key constraints referentially connected to a Fact Table schema. The similarity in content with these data marts is also depicted in the fact table here also containing *Policy Number* as a degenerate dimension attribute, as well as a *Claims Transaction Amount* as a fact or measure. The Claims fact table contained *1,144* tuple rows of data, with the corresponding *Insured Party* dimension - similar to that in the *Policy Transactions* data mart - also containing *26* set of attributes description, and tuple rows of data of *848*.

Both data sets had overlapping dimensional entity representation of six (*6*) dimension tables, whilst the *Claims Transactions* data mart had three (*3*) other non-corresponding dimensions. One other feature that characterized the data marts was the existence of multiple representations of entities in the dimension tables. This depicted the concept of *Slowly Changing Dimensions* (SCDs) in the dimensional entity tables. Additionally, the dimension tables were free from *Multivalued Dimension Attributes*; where there exist the associations of different number of entities to a different number of accounts.

With regards to the Transportation Services domain, we had three (*3*) multidimensional star schemas. These data sets were *Frequent Flyer Transactions*, *Hotel Stays Transactions*, and *Car Rental Transactions* data marts. All the data marts had three (*3*) conformed or overlapping dimensions; namely, *Customer*, *Date*, and *Sales Channel*. These dimensions were complemented with a number of non-corresponding and unique dimensions in each of the data marts. We further briefly describe the contents of each of the data marts.

Frequent Flyer Transactions Data Mart

The *Frequent Flyer Transactions* data mart was made up of nine (*9*) dimension table schemas and a single fact table. These dimensions were *Customer*, *Fare Class*, *Flight*, *Flight Status*, *Flyer Date*, *Flyer Time*, *Sales Channel*, and *Segment Airport*. The fact table had degenerate dimension attributes of *Ticket Number*, *Segment Sequence Number* and *Itinerary Number*. The facts or measures that made up the numeric data representation were *Segment Flight Duration*, *Segment Miles Earned*, *Segment Miles Flown*, *Gross Segment Fare*, *Minutes Late At Departure*, *Minutes Late At Arrival*, and *Net Minutes Late*. All these fact table attributes together represented a total instance data of *7257* tuples of rows.

Hotel Stays Transactions Data Mart

The *Hotel Stays Transactions* data mart was made up of five (*5*) dimension tables, each linking the fact table by referential key constraints. These dimensions were, namely; *Customer*, *Hotel*, *Hotel Reservation Date*, *Hotel Status*, and *Sales Channel*. The attributes that constituted the degenerate dimension in the fact table were *Itinerary Number*, *Ticket Number*, and *Segment Number*. The fact table was made up of measures, of which together with the degenerate dimension and other dimension attributes, contributed to a total of *2449* tuples of rows. The facts or measures of the fact table were *Number Of Days*, *Room Dollar Charge*, *Meals Dollar Charge*, *Phone Dollar Charge*, *Miscellaneous Charge*, and *Tax Charge*.

Car Rental Transactions Data Mart

The *Car Rental Transactions* data mart was also constituted by a single central fact table and a set of five (*5*) dimension tables. These dimensions were *Customer*, *Car Rental Date*, *Car Rental Status*, *Rental Car Service*, and *Sales Channel*. The degenerate dimensions that formed part of the attributes of the fact table were *Itinerary Number*, *Segment Number*, and *Ticket Number*. The total number of tuple rows that made up the fact table were *2449*, with a set of measures making up for the overall set of attributes in the fact table. These measures were *Rental Amount*, *Rental Number Of Days*, *Miscellaneous Amount*, *Rental Tax Charge*, and *Rental Charge Rate*.

All the multidimensional star schemas had their permanent repository stored in Microsoft SQL Server DBMS, with each entity representation in the dimensions identified by a unique *surrogate key* of an incremental identity specification, and based on clustered indexing.

5.2 Description of Our Experimental Implementation

We describe the experimental implementation for our merge methodology in workflow chain. We use the diagram represented in Figure 5.1 to illustrate our methodology for the integration. Each of the steps in the workflow consists of a series of processes which we describe into detail in the next sections.

5.3 Schema Matching and Mapping Discovery Methodologies

The schema matching and mapping models discovery procedural steps were implemented using IBM Infosphere Data Architect application software [45]. In enabling the accessibility of the application software to automatically infer on instance data so as to find mapping correspondences or generate the mapping models, a data connection to the data sources where our data mart repositories are stored was created. These data sources were then incorporated into the *Data Model* module through a reverse engineering approach of using the *Physical Data Model with Dimensional Notation* option. This option was chosen because of the multidimensional characteristics of the *star schema* data models used as data sets. This enabled an automatic identification of the loaded schema tables into categories of *fact table* and *dimension tables* by the application tool.

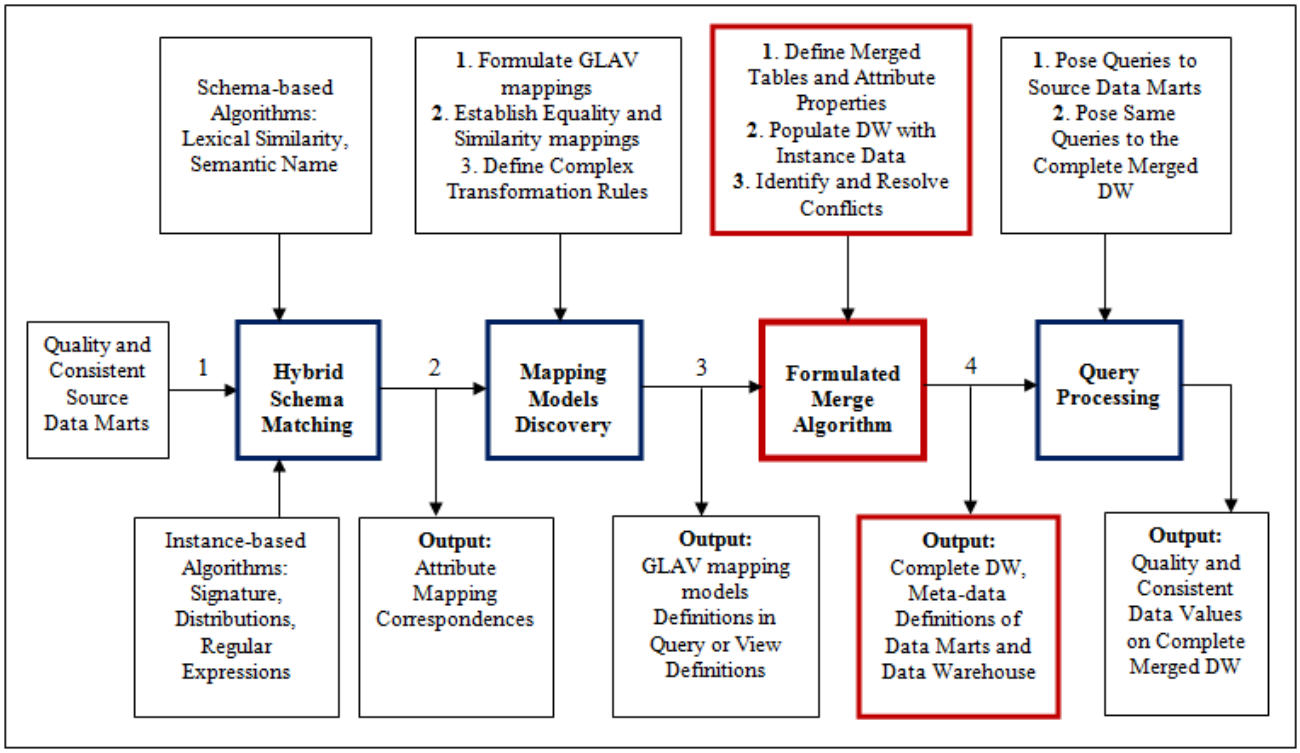

Figure 5.1: *Procedural Steps in the Experimental Implementation*

Based on the implementation architecture of the application tool in line with the rudimentary tasks for schema matching, one or more *physical data models* had to be designated as *source(s)*, whilst at least one of the incorporated physical data models had to be designated as a *target*. In terms of the Insurance data set, the *Policy Transactions* data mart was designated as *source* and the *Claims Transactions* data mart designated as *target*. In the Transportation data set, the *Hotel Stays Transactions* and the *Car Rental Transactions* data marts were designated as the *source schemas*, whiles the *Frequent Flyer Transaction* data mart was assigned as the *target* schema.

The rest of the subsections discuss the implementation processes with regards to the *schema matching* in subsection 5.3.1 and the generation of mapping models in subsection 5.3.2.

5.3.1 Manipulation of Schema Matching Algorithms

The implementation of the schema matching procedure was based on the laid out methodology as described in Section 4.2, where both schema-level and instance-level algorithms were manipulated in a hybrid approach to generate mapping correspondences between the

attributes of the fact and dimension tables.

In finding mapping correspondences between the schema attributes in terms of using the discovery function in the application software tool, two (2) methods used in generating the set of candidate attribute match(es) are defined: *Find Best Fit* and *Find Similar* [46]. The *Find Best Fit* method finds the best overall score of all potential element pairings or matching in all of the elements within the scope of the schema or model. Since there is a potential for a probabilistic attribute matching, this automatic method produces the most satisfactory matches in the set of attributes of the entire model and returns at most one match for one target and one source. Because of its automatic nature, there is a possibility of having no matches after the execution of the discovery function for finding mapping correspondences, or matching wrong attribute(s) in the source schemas to attribute(s) in the target schemas.

The *Find Similar* method, on the other hand, is a semi-automatic method of finding mapping correspondences with the option of a human interaction in the schema matching procedure, where the generation of possible attribute match results is presented to the user. In this method, a predefined number of match pairings for each target attribute element within the scope of the schema are produced. This method then offers the user the ability to validate and choose the satisfied match pairing among the host of produced match candidates.

In our schema matching procedure, in the overall methodology, we adopt the *Find Similar* method, where we choose the most semantically correct match for a set of schema attribute mappings generated from running the system. Our motivation for such a choice is to be able to control the generation of the semantically correct mapping correspondences. We therefore introduce a user input in the generation of match pairings, which is usually one of the characteristics that make a hybrid schema match model a better choice amongst others. The *Find Best Fit* alternative has the tendency of generating semantically wrong matches for the schema attributes, with no option of user validation and correcting such semantic errors.

This makes the *Find Similar* matching routine a better option where there we implement the processes of attribute matching by inspection, and necessarily adjust the configuration for better semantic correspondences. The processes of user validation of the attribute match results also lead to generating semantic correct attribute correspondences in the schema matching procedural step.

An example of choosing a semantically correct match candidate from the generated mapping correspondences of *PolicyTransactionTypeKey*, *PolicyTransactionID*, and *TransactionCodeName* attributes in the *Dim_PolicyTransactionType* dimension to the *ClaimTransactionCode* attribute of the *Dim_ClaimTransactionType* dimension, is displayed in Figure 5.2.

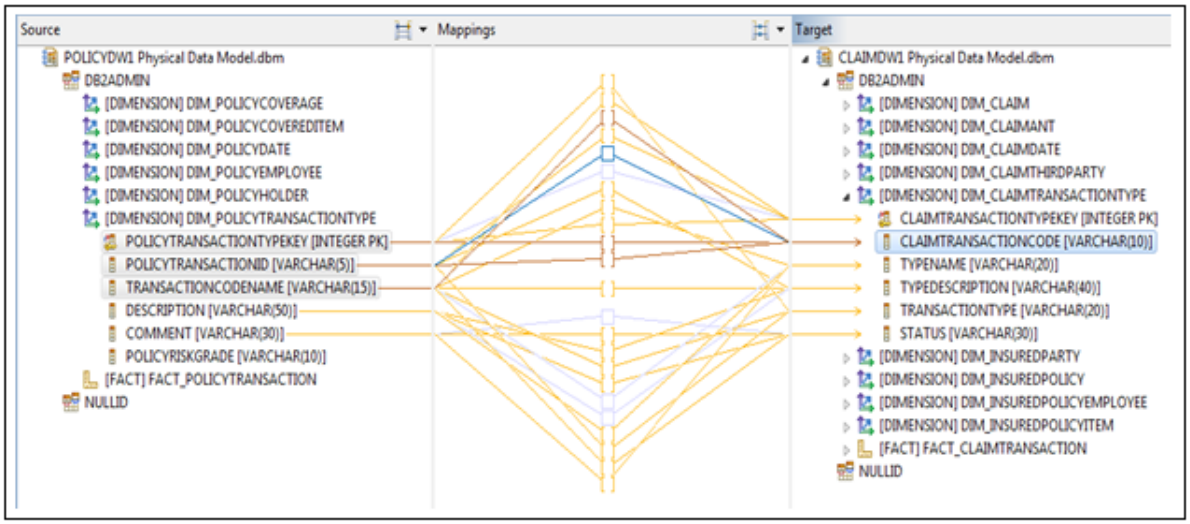

Figure 5.2: *Finding Attribute Mapping Correspondences*

In Figure 5.2, the *blue-coloured* mapping represents the chosen semantically correct matching candidate, where *PolicyTransactionID* attribute corresponds to the *ClaimTransactionCode* attribute. On the other hand, the *red-coloured* mappings represent the semantically incorrect matching candidates of *PolicyTransactionTypeKey* and *TransactionCodeName* which are ignored, as part of user validation by inspection.

When generating mapping correspondences for the fact and dimension table attributes, various configuration manipulations of algorithms are performed on the discovery function, in terms of adjusting the parameters for efficient matching results. As earlier stated in Section 4.2, the execution of the algorithms are ordered with the schema-level algorithms coming first and the instance-level algorithms following up second. The ordering for the schema-level algorithms is *Lexical Similarity*, and *Semantic Name*; whilst the ordering for the instance-level algorithms is *Signature*, *Distributions*, and *Regular Expressions*.

Different configurations were specified for each of the algorithms implemented in the schema matching procedure. The parameters used in configuring the algorithms were *Rejection Threshold*, *Thesaurus Option*, *Sampling Size*, and *Sample Rate*. The *Rejection Threshold* parameter was configured with different adjustments for both the schema- and instance-based algorithms. The *Thesaurus Option* parameter was only applicable to the *Semantic Name* algorithm, but there was no external glossary or thesaurus configuration for the algorithm. The *Sampling Size* and *Sampling Rate* parameters were not applicable to the schema-based algorithms, but rather for instance-based algorithms. These parameters were configured appropriately to aid the efficient generation of matching candidates.

We summarize the parameterized configuration of the algorithms adopted in the schema matching procedure for finding mapping correspondences in *Table* 5.1. It will be noted that these configurations where based on an iterative procedure of inspection, where different parameter values were experimentally tweaked as by observing the generated mapping correspondence results. These configurations were also based on the initial default configurations that have been specified in [46].

Table 5.1: *Summary of Manipulation Configurations for Schema Matching Algorithms*

Matching Algorithm / Configuration Option	**Rejection Threshold**	**Thesaurus Option**	**Sampling Size (Rows)**	**Sampling Rate (%)**
1. Lexical Similarity	0.6	Not Applicable	Not Applicable	Not Applicable
2. Semantic Name	0.5	Is Applicable; But not configured for the schema matching	Not Applicable	Not Applicable
3. Signature	0.8	Not Applicable	150	30
4. Distributions	0.8	Not Applicable	100	20
5. Regular Expressions	0.9	Not Applicable	100	30

5.3.2 Mapping Models Generation

The implementation of the mapping models generation was based on the adopted GLAV mapping models, where we had the definition of overlapping attributes being represented by a single merged attribute and also the incorporation of non-corresponding local attributes into the merged table schemas. The GLAV mapping models also offered the definition and enforcement of some complex transformation expressions on multiple cardinality mapping relationships.

As part of making the mapping model more expressive, we enclosed the complex transformation expressions in the generated mapping relationships for any pair of corresponding attributes. For instance, in Figure 5.3, there is a multiple cardinality mapping relationship between *FullName* attribute in *Dim_InsuredParty* dimension schema and two (2) other attributes in the *Dim_PolicyHolder* dimension; namely, *FamilyName* and *GivenName*. We therefore, defined a complex transformation expression, as in Equation 5.1, in the mapping relationship already established between these dimension attributes.

$$FULLNAME = FAMILYNAME \quad + \quad ', \quad ' \quad + \quad GIVENNAME \qquad (5.1)$$

These forms of complex tranformation expressions are generally derived based on the examination of the instance data contained in the schema of each of the source star schema data marts. It will be emphasized that the complex transformation expressions or formulas aid in the data population activity as part of the merge algorithm.

Other forms of mapping properties that were defined in the established mapping correspondence relationships were the expressive characterization of relationship cardinality, the attribute semantic representation, and attribute data type representation, amongst others. In terms of the relationship cardinality, an equality or similarity mapping cardinality type was defined. For the attribute semantic representation, a definition of the supposed merged attribute name was specified where possible. This merge attribute name will represent both attributes involved in a particular mapping relationship. The supposed merge attribute data type which will serve as a union data type for the merging attributes was also defined.

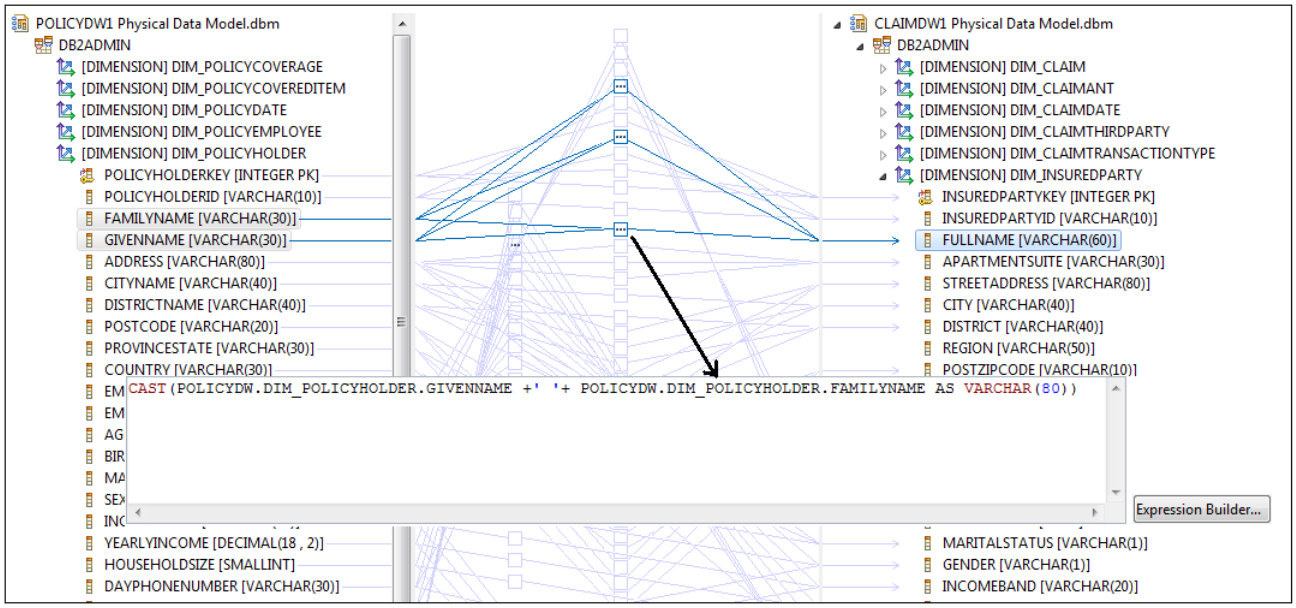

Figure 5.3: *Discovering and Establishing Mapping Relationships*

After the establishment of mapping formalisms between the attributes of the fact and dimension tables and the subsequent definition of all complex transformation formulas, which together formed our supposed mapping model, we generated this mapping model in executable forms. The application tool offered two (2) options of either query scripts or an output file, of which we did our implementation using the latter. The output file, which was

formatted in *.csv*, offered the entire needed attribute columns needed in a mapping model and necessary in the algorithmic programming for the schema merge procedural step in the overall integration methodology.

5.4 Merge Algorithm Implementation

The merge algorithm was implemented by using a programming control, which was scripted in Microsoft Visual C# .Net Integrated Development Environment (IDE). The organization of the program design initially involved establishing connection string as an entity class to serve as communication link between the source data mart repositories and the programme control, and to ensure a transaction processing medium between the object classes and the data repositories. To that effect, the main inputs to the algorithm were mapping model - which had been loaded into the database as a schema table, and comprising of necessary attribute columns - as well as, the multidimensional star schemas.

The main content of the program control involved the design of the Graphical User Interfaces (GUIs) to serve as an interactive medium for a user-friendly application, and offer a flexible usability to users. Moreover, various forms of object classes were developed and scripted to comply with the object-oriented architecture of the scripting environment. The object classes that were scripted were composed of *Utility*, *Entity*, *Data Access*, and *Business Logic* classes.

The Utility classes were scripted for basic operations of temporary hold of variable data values, database and SQL helper classes, amongst others. The Entity classes were scripted to hold the property information of mapping models, data mart schemas, table and column definitions, amongst others. The Data Access classes were also scripted to contain the transaction processing operations of select, insert, delete, and update. Finally, the Business Logic classes were scripted to contain the processing and execution control of the DataAccess classes.

To sum it up, a total of *8029* lines of code were scripted for the merge algorithm implementation; which is composed of *2909* lines of code from programme control, *876* lines of code from Business Logic classes, *656* lines of code from Entity classes, *1595* lines of code from DataAccess classes, and *1993* lines of code from Utility classes.

As part of ensuring a transaction processing workload to be placed on the server side of the application, stored procedures were employed. This was adopted because of the expected amount of data usually contained in data marts, and which will avoid an unnecessary delay in processing data transactions. These stored procedures were scripted to perform normal

data transactions - between the program control and server side database - and logic based transactions as well, based on the input variables fed to them.

5.5 Query Processing – Star Schema Data Marts and Single Consolidated Data Warehouse

The query processing and analytics were implemented using a Business Intelligence (BI) tool, being IBM Cognos BI application software. This tool enabled the possibility of performing query processing - in the form of report generation - on the data sources and easily analyzing the results, for comparison and evaluation. The software has various components that lead to creating reports and trend analysis of charts, extreme data highlighting, amongst others. There is the *Framework Manager* - for a conceptual modelling and setup of data sources - and a host of analytic capabilities - *Querying and Reporting*, *Analysis*, *Dashboarding*, *Scorecarding*.

The procedural steps used in creating query processing reports are outlined as follows:

Step 1 – Creating a Project (Framework Manager)

A project was created and this contained all the configurations needed for the BI application. It is made up of the *Models*, *Data Sources*, *Parameter Maps*, and *Packages*.

Step 2 – Creating the Metadata Source (Framework Manager)

A *Metadata Wizard* was run which created all the needed components in establishing the BI application. A *Metadata Source* was first created, and this connected the BI application to the data repository. The objects of fact and dimension tables, and referential constraints existent in these star schemas are used in creating the conceptual model for query proccessing.

Step 3 – Creating the Model (Framework Manager)

The next step is the importation of the data warehouse objects. This formed the basis for the creation of the *Model*. The *Model* presents a conceptual representation consisting of different panes - *Explorer*, *Diagram*, *Dimension Map* - for managing the BI application. The *Explorer* pane displays all the objects that can be used to establish the referential integrity of the dimension tables to the fact table and also to demonstrate the relationships between them, as well as displaying hierarchical dimensions - *Regular* and *Measure*. At the *Dimension Map*, various hierarchies, in each of the imported dimensions from the *Metadata Sources*,

were created. For example, an *Insured Party* (*Geography*) hierarchy was created for the *Insured Party Dimension* in the global data warehouse, where we had *Country*, *Province*, *Region*, *City*, *District*. Another example for the *Date Dimension* was a Season hierarchy as in; *Calendar Year*, *Calendar Season*, *Calendar Month*. Due to different hierarchical order per dimension, different hierarchies were created for each dimension as needed in report querying and analysis. Hence, a *Season*, *Quarter*, and *Month* hierarchies were created for the *Date Dimension* in the global data warehouse.

In summary, these relationships and hierarchies were created in the *Model* framework to offer the best of querying and analysis in general or aggregated reports, and also to present the BI user functionalities of *drill-down* and *roll-up*, and *dicing* and *slicing*.

Step 4 – Creating and Publishing the Package (Framework Manager)

At this step, a *Package* is created. This served as a container for accommodating all the objects that had been created in the *Model* framework and for onward use in the *Reporting Studio*.

Step 5 – Creating the Query Processing Reports (Report Studio)

At this step, reports are generated which answers the set of queries posed by business users to the data warehouse. The *Report Studio* offers a variety of reporting options and visual representation to business users as part of data warehousing querying and analysis. The *Package* already created for the *Model* at *Framework Manager* is selected and used for all forms of query report processing.

5.6 Summary

This Chapter presented the overall experimental setup and the implementation procedures for our approach of integration methodology for multidimensional data models. We first described the experimental data sets that we used; being multidimensional star schemas from Insurance and Transportation Services domains. We discussed their schema structure and instance data content. We also discussed about the implementation procedures for the schema matching methodology in terms of the manipulation and configuration processes of the available algorithms - both schema-level and instance-level - to deliver efficient mapping correspondences between schema attributes.

Regarding the mapping model discovery methodology, we discussed the enrichment implementation procedures of defining complex transformation expressions, and also the definition of other expressive characteristics that the mapping model can exhibit to make it very resourceful in the merge algorithm. We also discussed the details as regard to the merge algorithm, where we explained the programming environment involving the object classes and stored procedures. The last section of the Chapter was dedicated to the implementation procedures that involved the query processing tasks. In this segment, we discussed about the BI tool we used and also the generation analytical reports which depicted our query processing execution.

In the next Chapter, we will discuss about the evaluation results based on the implementation tasks we performed in the query processing activities. We will first of all outline the criteria for evaluation. These criteria will focus discussions about dimensionality hierarchy representation in the merged dimensional entities, general query processing for correct data values, and aggregate query processing for OLAP operations. We will then summarize in that Chapter by addressing the rate of query processing, where we will compare the rate of generating data values to similar queries posed to the generated data warehouse in comparison to that posed to the individual multidimensional star schemas.

Chapter 6

Experimental Results Evaluation

As way of keeping track and critically follow the main content of this thesis, we briefly recap the discussions so far. In Chapter 1, we discussed the motivation and the main propositions for this thesis, where we outline some objectives. In Chapter 4, we then followed up with some propositions - in terms of methods of generating efficient mapping correspondences, efficient mapping models discovery methods, technical merge correctness requirements, merge algorithm, and likely conflicts and their resolutions - and discussions on the overall integration methodology needed in achieving such prior objectives discussed in Chapter 1. In Chapter 5, we discussed the implementation of the proposed methodology in Chapter 4, where we had to explain the practical procedures and their involved activities of schema matching, mapping model discovery, programming of the merge algorithm, and the query processing setup and implementation.

In this Chapter, we address the analysis of our evaluation results based on the various experiments we conducted in Chapter 5; as part of the query processing implementation. This evaluation analysis is carried out as a measure in determining and verifying the correctness of the merge methodology. It will be noted that the evaluation analyses are primarily based on the output from the formulated merge algorithm - that is, the generated global data warehouse - in relation to the independent multidimensional star schemas. We compared the output of the query processing on the multidimensional star schemas and the generated data warehouse, by formulating a query that has similar semantics in both multidimensional star schemas. We then consecutively run this query on the independent data marts, and afterwards run the same form of query on the generated data warehouse. With these orderly activities, we are able to effectively compare the results, first from the multidimensional star schemas and then from the global data warehouse.

The main content of this Chapter as follows. We outline our propositions of a set of

evaluation criteria to strengthen the determination of the success for the results in Section 6.1. In Section 6.2, we perform a series of query processing experiments on, first, the independent multidimensional star schemas and then on the generated global data warehouse. In Section 6.3, we discuss query processing response rate for some of the experiments we performed so as to evaluate the processing time for query execution. We conclude on the overall discussions in this Chapter in Section 6.4.

6.1 Evaluation Criteria

The first point of call in these evaluation analyses of comparing query results was to outline the criteria for evaluation, and then based on these perspectives we appraise the results in the right direction. From the discussion of the semantics of query processing on multidimensional data models in Section 4.5, we follow-up with some of these standpoints that queries posed to the multidimensional star schemas and global data warehouse should conform to, and which can be used in accessing the validity of generated data values.

Pedersen et al. in [72] outlined an elaborate set of characteristics and requirements that multidimensional data models should satisfy in order to fully support OLAP query processing. It will be inferred that these requirements and characteristics serve as vital guidelines for query processing on multidimensional data models. Consequent to these summarized characteristics and requirements explained and discussed in [72], our methodology also focuses on the semantics of this form of query processing. In line with this notion of running such queries on the adopted star schemas, we performed a gap analysis on the study in [72], based on their proposed requirements, and establish our outlined set of evaluation criteria.
Our criteria for query processing evaluation are outlined as follows:

1. Dimensionality Hierarchy;

2. Correctness of Data Values; and

3. Rate of Query Processing.

We explain these criteria in the next subsection under the experiments that we conducted.

6.2 Query Processing & Analysis of Star Schema Data Marts and Single Consolidated Data Warehouse

In this section, we conducted some experiments that express the validation of the evaluation criteria specified in the previous section. As earlier pointed out in the introductory discussion in this Chapter, we had to run similar queries on both independent multidimensional star schemas and the global data warehouse and compared their results. Some of the processes that we were performed as part of the query processing were the comparison of total and subtotal summaries of data values in line with different query items of interest. Additionally, the execution of OLAP activities on the query results to draw out meaning to the data values being displayed. These OLAP activities were *drill-down* and *roll-up*, and *dicing* and *slicing*.

6.2.1 Experiment 1 (General Query Processing)

In terms of *Correctness of Data Values*, we had the same data values for similar queries that were run. This was made possible as a result of the satisfaction of the MCR of *Tuple Preservation*.

We perform a query processing experiment on the Insurance data set, where we first run the query on the multidimensional star schemas and later on the global data warehouse. We use the diagrammatic query results in Figure 6.1, 6.2, 6.3, and 6.4 to explain better the criteria of correctness of data values for *Query* 1, where the same data values were generated by the global data warehouse (Figure 6.3, and 6.4) in comparison to that of the multidimensional star schemas of Policy Transactions (Figure 6.1) and Claims Transactions (Figure 6.2). The query report in *Query* 1 analyses the volume of new transactions that were processed on *Policy Holders* on their *Insured Policies* in a specific *Calendar Month* as against the performance during the same *Calendar Season*.

Query 1 How do the new transactions during a specific *Calendar Month* {*February*, *July*, *September*} compare to that during a particular *Calendar Season* {*Winter*, *Spring*, *Summer*}?

We also posed similar set of queries on the data repositories from the Transportation Services data set, so as ascertain the accuracy of the generated data values from the queries posed in comparison to the merged global data warehouse. We illustrate our query (*Query* 2) and the results of the data values generated for *Car Rental* in Figure 6.5, *Hotel Stays* in

PolicyTransactionAmount	March	May	April	Total(CalenderMonthName)
Spring	182,802.60	245,351.15	182,639.05	**610,792.80**
Total(CalenderSeason)	**182,802.60**	**245,351.15**	**182,639.05**	**610,792.80**

Spring

Figure 6.1: *Data Values from Policy Transactions Data Mart for Query* 1 *– Dicing on the* ***'Spring'*** *Calendar Season Parameter*

ClaimTransactionAmount	April	May	March	Total(CalenderMonthName)
Spring	1,745,600.00	2,056,000.00	2,338,400.00	**6,140,000.00**
Total(CalenderSeason)	**1,745,600.00**	**2,056,000.00**	**2,338,400.00**	**6,140,000.00**

Spring

Figure 6.2: *Data Values from Claims Transactions Data Mart for Query* 1 *– Dicing on the* ***'Spring'*** *Calendar Season Parameter*

TRANSACTIONAMOUNT		April	March	May	July	June	February	January	Total (CALENDERMONTHNAME)
ClaimDW	Spring	1,745,600.00	2,338,400.00	2,056,000.00	0.00	0.00	0.00	0.00	**6,140,000.00**
	Summer	0.00	0.00	0.00	6,848,800.00	2,392,000.00	0.00	0.00	**9,240,800.00**
	Winter	0.00	0.00	0.00	0.00	0.00	1,470,400.00	130,400.00	**1,600,800.00**
	Total (CALENDERSEASON)	**1,745,600**	**2,338,400**	**2,056,000**	**6,848,800**	**2,392,000**	**1,470,400**	**130,400**	**16,981,600.00**
PolicyDW	Spring	182,639.05	182,802.60	245,351.15	0.00	0.00	0.00	0.00	**610,792.80**
	Summer	0.00	0.00	0.00	70,446.95	504,538.20	0.00	0.00	**574,985.15**
	Winter	0.00	0.00	0.00	0.00	0.00	226,894.50	181,281.70	**408,176.20**
	Total (CALENDERSEASON)	**182,639.05**	**182,802.6**	**245,351.15**	**70,446.95**	**504,538.2**	**226,894.5**	**181,281.7**	**1,593,954.15**
Total(DataMartName)		**1,928,239.05**	**2,521,202.6**	**2,301,351.15**	**6,919,246.95**	**2,896,538.2**	**1,697,294.5**	**311,681.7**	**18,575,554.15**

CALENDERSEASON

Figure 6.3: *Data Values from Global Data Warehouse for Query* 1 *– General*

TRANSACTIONAMOUNT		May	April	March	Total(CALENDERMONTHNAME)
PolicyDW	Spring	245,351.15	182,639.05	182,802.60	**610,792.80**
	Total(CALENDERSEASON)	**245,351.15**	**182,639.05**	**182,802.6**	**610,792.80**
ClaimDW	Spring	2,056,000.00	1,745,600.00	2,338,400.00	**6,140,000.00**
	Total(CALENDERSEASON)	**2,056,000**	**1,745,600**	**2,338,400**	**6,140,000.00**
Total(DataMartName)		**2,301,351.15**	**1,928,239.05**	**2,521,202.6**	**6,750,792.80**

Spring

Figure 6.4: *Data Values from Global Data Warehouse for Query* 1 *– Dicing on the* ***'Spring'*** *Calendar Season Parameter*

Figure 6.6, *Frequent Flyer* in Figure 6.7, and Global Data warehouse in Figure 6.8.

Query 2 How do the new transactions during a specific *Calendar Season* {*Winter*, *Spring*, *Summer*} compare to that during a particular *Calendar Month* {*February*, *July*, *September*}?

GrossRentalAmount		January	February	Total(CalenderMonthName)
2008	Winter	545,451.68	0.00	**545,451.68**
2009	Winter	0.00	77,516.00	**77,516.00**
Total(CalenderYear)		**545,451.68**	**77,516.00**	**622,967.68**

Winter

Figure 6.5: *Data Values from Car Rental Data Mart for Query* 2 - *Dicing on the* ***'Winter'*** *Calendar Season Parameter*

GrossDollarCharge		January	February	Total(CalenderMonthName)
2008	Winter	1,019,582.91	0.00	**1,019,582.91**
2009	Winter	0.00	164,503.01	**164,503.01**
Total(CalenderYear1)		**1,019,582.91**	**164,503.01**	**1,184,085.92**

Winter

Figure 6.6: *Data Values from Hotel Stays Data Mart for Query* 2 - *Dicing on the* ***'Winter'*** *Calendar Season Parameter*

GrossSegmentFare		February	January	Total(CalenderMonthName1)
2011	Winter	226,370.00	278,935.00	**505,305.00**
Total(CalenderYear)		**226,370.00**	**278,935.00**	**505,305.00**

Winter

Figure 6.7: *Data Values from Frequent Flyer Data Mart for Query* 2 - *Dicing on the* ***'Winter'*** *Calendar Season Parameter*

Discussion

It will be realized that the data values that were generated in the global data warehouse were the exact values from that of the multidimensional star schemas; even in the presence of OLAP operation of *dicing* and *slicing*. In *Query* 1, for instance, the query processing that was done on the Insurance data set showed that dicing of the **'Spring'** *Calendar Season*

GROSSFAREAMOUNT			January	February	Total(CALENDERMONTHNAME)
HotelStaysDW	2008	Winter	1,019,582.91	0.00	**1,019,582.91**
	2009	Winter	0.00	164,503.01	**164,503.01**
	Total(CALENDERYEAR1)		**1,019,582.91**	**164,503.01**	**1,184,085.92**
FrequentFlyerDW	2011	Winter	278,935.00	226,370.00	**505,305.00**
	Total(CALENDERYEAR1)		**278,935.00**	**226,370.00**	**505,305.00**
CarRentalDW	2008	Winter	545,451.68	0.00	**545,451.68**
	2009	Winter	0.00	77,516.00	**77,516.00**
	Total(CALENDERYEAR1)		**545,451.68**	**77,516.00**	**622,967.68**
Total(DataMartName)			**1,843,969.59**	**468,389.01**	**2,312,358.60**

Winter

Figure 6.8: *Data Values from Global Data Warehouse for Query* 2 *- Dicing on the* ***'Winter'*** *Calendar Season Parameter*

out of the general set of *Calendar Seasons* for each of the multidimensional star schemas still produced accurate data values in comparison to that on the generated data warehouse. This underscores the generation of data values on the generated global data warehouse being an accurate representation of the data values from the independent multidimensional star schemas.

6.2.2 Experiment 2 (Dimensional Hierarchy)

With regards to *Dimensionality Hierarchy*, we realized that there was either a *full-* or *partial-level* representation of any hierarchy in the merged dimension. For merged dimension tables that had the similar semantics and contents in the levels of the hierarchy in the independent multidimensional star schemas, a *full-level* hierarchy was represented in the merged dimension table. We describe this phenomenon in Example 6.2.1, where we use the multidimensional star schemas and global data warehouse from the Insurance data set.

Example 6.2.1 *Suppose we have a business clustering hierarchy in the Insured Policy dimension in the Claims Transactions data mart, corresponding to a similar hierarchy in the Policy Coverage dimension in the Policy Transactions data mart; Insured Policy Name, Business Type, Clientele Target as the hierarchy in the Insured Policy dimension, and Policy Coverage Name, Line Of Business, Market Segment also as a hierarchy in the Policy Coverage dimension. During merging, these two dimensional hierarchies have to be combined into one in the merged dimension.* ∎

With this kind of hierarchy representations in both dimensions, it will be realised that there is an equal representation in each of the levels of the respective dimensions. A merged dimension representing these two independent dimensions will subsequently assume the hierarchy from both of these hierarchies. This new hierarchy will, therefore, present a *full-level* hierarchy representation for any of the integrating hierarchies of their independent dimension.

In terms of *partial-level* hierarchy representation, there might be the case where the merged dimension will present a number of hierarchy levels identical to each of the integrating dimensions. We explain this phenomenon in the illustration in Example 6.2.2.

Example 6.2.2 *Suppose we have a business clustering hierarchy in the Insured Policy dimension in the Claims Transactions data mart, corresponding to a similar hierarchy in the Policy Coverage dimension in the Policy Transactions data mart; Insured Policy Name, Business Type, Clientele Target as the hierarchy in the Insured Policy dimension, and Policy Coverage Name, Line Of Business, Market Segment also as a hierarchy in the Policy Coverage dimension. During merging, these two dimensional hierarchies have to be combined into one in the merged dimension.* ∎

It will be noted that although the Region level in the hierarchy in *Policy Holder* dimension is non-existent, it is still represented in the merged dimension - because of the GLAV mapping model adopted. Consequent to this representation, the integration of data will not have any real-world data values coming from the *Policy Holder* dimension in the *Policy Transactions* data mart, but there will still be roll up of data into the *Region* level - as a result of seemingly *Region* level data value from the *Insured Party* dimension in the *Claims Transactions* data mart. This will therefore present a case in hand where a partial form of this hierarchy is exposed in the merged dimension, and OLAP operations of *dicing* and *slicing*, and *roll-up* and *drill-down* will not depict the real-world representation of data values contained in their hierarchical levels.

A description of this phenomenon is represented in the set of evaluation results from the experiments of formulated query *Query* 3 below.

Query 3 How do the new transactions during a specific *Calendar Month* {*February*, *July*, *September*} compare to that during a particular *Calendar Season* {*Winter*, *Spring*, *Summer*}?

We used the experimental results from the *Policy Transactions* data mart in comparison to the global data warehouse, of which we performed this experiment in various stages.

Step 1

Our initial queries on both the *Policy Transactions* data mart and the global data warehouse demonstrated data values representing each of the ***States***, as displayed in Figure 6.9 - *Policy Transactions* data mart and Figure 6.10 - Global data warehouse.

PolicyTransactionAmount ($)	2008	Total(CalenderYear1)
California	317,244.10	**317,244.10**
Oregon	46,268.50	**46,268.50**
Washington	202,938.45	**202,938.45**
Total(Country1)	**566,451.05**	**566,451.05**

Figure 6.9: *Data Values from Policy Transactions Data Mart for Query* 3

TRANSACTIONAMOUNT		2008	2011	Total(CALENDERYEAR1)
California	PolicyDW	317,244.10	0.00	**317,244.10**
	Total(DataMartName)	**317,244.10**	**0.00**	**317,244.10**
Oregon	PolicyDW	46,268.50	0.00	**46,268.50**
	Total(DataMartName)	**46,268.50**	**0.00**	**46,268.50**
Washington	PolicyDW	202,938.45	0.00	**202,938.45**
	Total(DataMartName)	**202,938.45**	**0.00**	**202,938.45**
Total(COUNTRY1)		**566,451.05**	**5,413,600.00**	**5,980,051.05**

Figure 6.10: *Data Values from Global Data Warehouse for Query* 3 *– Drilling-down on* ***'PolicyDW'*** *Data Mart*

Step 2

In the next step, we drilled down further onto the next level of the hierarchy, i.e. drilling-down to the *Region* level. We choose the ***Oregon*** *State* as the parameter to drill-down on. Our experimental results for the *Policy Transactions* data mart is displayed in Figure 6.11 whilst that of the global data warehouse is displayed in Figure 6.12.

Step 3

In the previous step, the query results displayed *Cities* for the *Policy Transactions* data mart, whilst the global data warehouse still displayed the *Regions*. This is because the *Policy Transactions* data mart had no *Region* level in its geographical hierarchy. In this step, we had to drill-down on a particular *Region* in the global data warehouse, so as to come to par with the level in the query results being displayed by the *Policy Transactions*

PolicyTransactionAmount ($)	2008	Total(CalenderYear1)
Lake Oswego	7,900.00	**7,900.00**
Milwaukie	11,396.00	**11,396.00**
Oregon City	5,377.50	**5,377.50**
W. Linn	15,907.50	**15,907.50**
Woodburn	5,687.50	**5,687.50**
Total(Country1)	**46,268.50**	**46,268.50**

Figure 6.11: *Data Values from Policy Transactions Data Mart for Query* 3 *– Drilling-down on the* ***'Oregon'*** *State*

TRANSACTIONAMOUNT		2008	2011	Total(CALENDERYEAR1)
Donovan's Sports	PolicyDW	7,175.00	0.00	**7,175.00**
	Total(DataMartName)	**7,175.00**	**0.00**	**7,175.00**
Holstein Golf	PolicyDW	15,907.50	0.00	**15,907.50**
	Total(DataMartName)	**15,907.50**	**0.00**	**15,907.50**
Le Golfeur	PolicyDW	5,687.50	0.00	**5,687.50**
	Total(DataMartName)	**5,687.50**	**0.00**	**5,687.50**
Maximum Sports	PolicyDW	5,377.50	0.00	**5,377.50**
	Total(DataMartName)	**5,377.50**	**0.00**	**5,377.50**
Sport & Freizeit	PolicyDW	11,396.00	0.00	**11,396.00**
	Total(DataMartName)	**11,396.00**	**0.00**	**11,396.00**
Tamarack Outfitter Rentals	PolicyDW	725.00	0.00	**725.00**
	Total(DataMartName)	**725.00**	**0.00**	**725.00**
Consumer Club	PolicyDW	0.00	0.00	**0.00**
	Total(DataMartName)	**0.00**	**0.00**	**0.00**
Total(COUNTRY1)		**46,268.50**	**1,133,200.00**	**1,179,468.50**

Figure 6.12: *Data Values from Global Data Warehouse for Query* 3 *– Drilling-down on the* ***'Oregon'*** *State*

data mart. We choose to drill-down on the ***'Maximum Sports'*** *Region*. Our experimental results for the drill-down on the global data warehouse are displayed in Figure 6.13.

TRANSACTIONAMOUNT		2008	2011	Total(CALENDERYEAR1)
Oregon City	PolicyDW	5,377.50	0.00	**5,377.50**
	Total(DataMartName)	**5,377.50**	**0.00**	**5,377.50**
Total(COUNTRY1)		**5,377.50**	**0.00**	**5,377.50**

Figure 6.13: *Data Values from Global Data Warehouse for Query* 3 *– Drilling-down on the* ***'Maximum Sports'*** *Region*

Step 4

In the step, we now have both geographical hierarchical levels being at same hierarchical level, at the *City* level. Query results from the *Policy Transactions* data mart and the global data warehouse showed similar representations of cities in the experimental display. The next step was to drill-down on a specific *City* level to track the data values that will be displayed, as *Districts*. We choose to drill-down on the ***'Oregon City'*** *City* on both platforms, of which our experimental results for the *Policy Transactions* data mart displayed query data values as in Figure 6.14, whilst the global data warehouse displayed query data values as in Figure 6.15.

PolicyTransactionAmount ($)	2008	Total(CalenderYear1)
Melissa	5,377.50	**5,377.50**
Total(Country1)	**5,377.50**	**5,377.50**

Figure 6.14: *Data Values from Policy Transactions Data Mart for Query* 3 *– Drilling-down on the* ***'Oregon City'*** *City*

TRANSACTIONAMOUNT		2008	2011	Total(CALENDERYEAR1)
Melissa	PolicyDW	5,377.50	0.00	**5,377.50**
	Total(DataMartName)	**5,377.50**	**0.00**	**5,377.50**
Total(COUNTRY1)		**5,377.50**	**0.00**	**5,377.50**

Figure 6.15: *Data Values from Global Data Warehouse for Query* 3 *– Drilling-down on the* ***'Oregon City'*** *City*

From the experimental results which displayed query data values from both the *Policy Transactions* data mart and the global data warehouse, we realize that the data values for

the *District* level hierarchy were the same for the ***'Melissa'*** *District*, which indicated a preservation of the data though the hierarchical levels had some changes in the individual cases.

Preamble To Aggregate Query Processing

In the experiments covering formulated *Queries* 4, 5 and 6, we performed aggregate queries on the global data warehouse and compared the generated query data values to that of the individual multidimensional star schemas. It will be noted that aggregate queries are the most common type of queries posed to data marts and data warehouses. Hence, the aggregate queries that we posed to the data marts and data warehouses were of the form of typical queries that are normally posed to data warehouses by business users - such as supervisory managers, middle management, or top executives - and those which cut across various levels of information need in the company or organization.

6.2.3 Experiment 3 (Aggregate Query Processing)

We performed the first experiment on the Insurance data set where we posed an aggregate query of all new transactions of *Policy Holders* in the *Policy Transactions* data mart and that of *Insured Parties* in the *Claims Transactions* data mart. Our attention for this experiment was to evaluate the performance of transactions based on the *Countries* from which these *Policy Holders* or *Insured Parties* reside or commercially do their business, and thereon business users could make decisions and analytics of strategizing on commercial activities from these query results.

Query 4 How do the Aggregated new transactions for *Country* {*Germany*, *Canada*, *France*} compare to that of a particular *Calendar Year* {*2008*, *2009*, *2011*}?

The query results that were generated from posing *Query* 4 to the data marts and the data warehouse are displayed as in; Figure 6.16 for the *Policy Transactions* data mart, Figure 6.17 for the *Claims Transactions* data mart, and Figure 6.18 for the global data warehouse.

6.2.4 Experiment 4 (Aggregate Query Processing)

The second experiment that we performed on aggregate query processing involved the formulated query in *Query* 5, in the Transportation Services data set. In this query processing

PolicyTransactionAmount ($)	2008	Total(CalenderYear1)
Australia	334,344.00	**334,344.00**
Canada	172,395.50	**172,395.50**
France	144,800.70	**144,800.70**
Germany	154,528.80	**154,528.80**
United Kingdom	221,434.10	**221,434.10**
United States	566,451.05	**566,451.05**
Total(Country1)	**1,593,954.15**	**1,593,954.15**

Figure 6.16: *Data Values from Policy Transactions Data Mart for Query* 4

ClaimTransactionAmount	2011	Total(CalenderYear1)
Germany	951,200.00	**951,200.00**
United States	5,413,600.00	**5,413,600.00**
Australia	5,959,600.00	**5,959,600.00**
United Kingdom	868,800.00	**868,800.00**
France	1,363,200.00	**1,363,200.00**
Canada	2,425,200.00	**2,425,200.00**
Total(Country1)	**16,981,600.00**	**16,981,600.00**

Figure 6.17: *Data Values from Claims Transactions Data Mart for Query* 4

TRANSACTIONAMOUNT		2008	2011	Total(CALENDERYEAR1)
United Kingdom	PolicyDW	221,434.10	0.00	**221,434.10**
	ClaimDW	0.00	868,800.00	**868,800.00**
	Total(DataMartName)	**221,434.10**	**868,800.00**	**1,090,234.10**
Canada	PolicyDW	172,395.50	0.00	**172,395.50**
	ClaimDW	0.00	2,425,200.00	**2,425,200.00**
	Total(DataMartName)	**172,395.50**	**2,425,200.00**	**2,597,595.50**
Australia	PolicyDW	334,344.00	0.00	**334,344.00**
	ClaimDW	0.00	5,959,600.00	**5,959,600.00**
	Total(DataMartName)	**334,344.00**	**5,959,600.00**	**6,293,944.00**
United States	PolicyDW	566,451.05	0.00	**566,451.05**
	ClaimDW	0.00	5,413,600.00	**5,413,600.00**
	Total(DataMartName)	**566,451.05**	**5,413,600.00**	**5,980,051.05**
Germany	PolicyDW	154,528.80	0.00	**154,528.80**
	ClaimDW	0.00	951,200.00	**951,200.00**
	Total(DataMartName)	**154,528.80**	**951,200.00**	**1,105,728.80**
France	PolicyDW	144,800.70	0.00	**144,800.70**
	ClaimDW	0.00	1,363,200.00	**1,363,200.00**
	Total(DataMartName)	**144,800.70**	**1,363,200.00**	**1,508,000.70**
Total(COUNTRY1)		**1,593,954.15**	**16,981,600.00**	**18,575,554.15**

Figure 6.18: *Data Values from Global Data Warehouse for Query* 4

task, we seek to analyze the total new transactions that were processed through different *Sales Channel* as compared to the different *Calendar Years*.

We first pose the query to the *Car Rental* data mart, the *Hotel Stays* data mart, the *Frequent Flyer* data mart, and then finally on the global data warehouse. The query data values that were generated for *Query* 5 are displayed in Figure 6.19 for the *Car Rental* data mart, Figure 6.20 for the *Hotel Stays* data mart, Figure 6.21 for the *Frequent Flyer* data mart, and Figure 6.22 for the Global data warehouse.

Query 5 How do the Aggregated new transactions for *Sales Channel* {*Internet*, *Fax*, *Travel Agent*} compare to that of a particular *Calendar Year* {*2008*, *2009*, *2011*}?

GrossRentalAmount	2008	2009	Total(CalenderYear)
Direct Counter	137,275.09	30,624.00	**167,899.09**
Travel Agent	128,702.83	33,248.00	**161,950.83**
Direct Telephone	133,656.19	31,006.00	**164,662.19**
Internet	129,120.93	33,220.00	**162,340.93**
Fax	127,717.64	32,014.00	**159,731.64**
Total(channel_name)	**656,472.68**	**160,112.00**	**816,584.68**

Figure 6.19: *Data Values from Car Rental Data Mart for Query* 5

GrossDollarCharge	2008	2009	Total(CalenderYear)
Direct Counter	335,133.81	78,176.61	**413,310.42**
Travel Agent	196,648.31	52,180.35	**248,828.66**
Direct Telephone	265,749.55	75,458.25	**341,207.80**
Internet	288,673.67	51,759.08	**340,432.75**
Fax	265,694.99	75,661.72	**341,356.71**
Total(channel_name)	**1,351,900.33**	**333,236.01**	**1,685,136.34**

Figure 6.20: *Data Values from Hotel Stays Data Mart for Query* 5

6.2.5 Experiment 5 (Aggregate Query Processing)

On the final form of aggregated query processing that was performed on the multidimensional star schemas and the global data warehouse in the Insurance data set, we formulated a query as in *Query* 6. We analyzed the performance of the total new transactions of all *Insured Policies* that have been signed on as *Policy Holders* on one hand in the *Policy Transactions*

GrossSegmentFare	2011	Total(CalenderYear)
Direct Telephone	349,426.00	**349,426.00**
Travel Agent	191,933.00	**191,933.00**
Fax	284,299.00	**284,299.00**
Direct Counter	257,707.00	**257,707.00**
Internet	285,597.00	**285,597.00**
Total(sales_channel_name)	**1,368,962.00**	**1,368,962.00**

Figure 6.21: *Data Values from Frequent Flyer Data Mart for Query* 5

GROSSFAREAMOUNT		2008	2009	2011	Total(CALENDERYEAR)
Travel Agent	HotelStaysDW	196,648.31	52,180.35	0.00	**248,828.66**
	CarRentalDW	128,702.83	33,248.00	0.00	**161,950.83**
	FrequentFlyerDW	0.00	0.00	191,933.00	**191,933.00**
	Total(DataMartName)	**325,351.14**	**85,428.35**	**191,933.00**	**602,712.49**
Internet	HotelStaysDW	288,673.67	51,759.08	0.00	**340,432.75**
	CarRentalDW	129,120.93	33,220.00	0.00	**162,340.93**
	FrequentFlyerDW	0.00	0.00	285,597.00	**285,597.00**
	Total(DataMartName)	**417,794.60**	**84,979.08**	**285,597.00**	**788,370.68**
Direct Counter	HotelStaysDW	335,133.81	78,176.61	0.00	**413,310.42**
	CarRentalDW	137,275.09	30,624.00	0.00	**167,899.09**
	FrequentFlyerDW	0.00	0.00	257,707.00	**257,707.00**
	Total(DataMartName)	**472,408.90**	**108,800.61**	**257,707.00**	**838,916.51**
Fax	HotelStaysDW	265,694.99	75,661.72	0.00	**341,356.71**
	CarRentalDW	127,717.64	32,014.00	0.00	**159,731.64**
	FrequentFlyerDW	0.00	0.00	284,299.00	**284,299.00**
	Total(DataMartName)	**393,412.63**	**107,675.72**	**284,299.00**	**785,387.35**
Direct Telephone	HotelStaysDW	265,749.55	75,458.25	0.00	**341,207.80**
	CarRentalDW	133,656.19	31,006.00	0.00	**164,662.19**
	FrequentFlyerDW	0.00	0.00	349,426.00	**349,426.00**
	Total(DataMartName)	**399,405.74**	**106,464.25**	**349,426.00**	**855,295.99**
Total(SALES_CHANNEL_NAME)		**2,008,373.01**	**493,348.01**	**1,368,962**	**3,870,683.02**

Figure 6.22: *Data Values from Global Data Warehouse for Query* 5

data mart, and those that had claims for the *Insured Parties* processed on them on the other hand in the *Claims Transactions* data mart, with a comparison across the various *Calendar Years*.

We also compared these query data values generated on the multidimensional star schemas to that on the global data warehouse. The query data values generated are displayed as in Figure 6.23 for *Policy Transactions* data mart, Figure 6.24 for *Claims Transactions* data mart, and Figure 6.25 for the global data warehouse.

Query 6 How do the Aggregated new transactions for *Insured Policies* {*Motor*, *Home*, *Travel*} compare to that of a particular *Calendar Year* 2008, 2009, 2011}?

PolicyTransactionAmount ($)	2008	Total(CalenderYear1)
Home Insurance	796,450.00	**796,450.00**
Motor Insurance	766,644.15	**766,644.15**
Travel Insurance	30,860.00	**30,860.00**
Total(PolicyCoverageName1)	**1,593,954.15**	**1,593,954.15**

Figure 6.23: *Data Values from Policy Transactions Data Mart for Query* 6

ClaimTransactionAmount	2011	Total(CalenderYear1)
Home Insurance	3,894,800.00	**3,894,800.00**
Travel Insurance	2,394,800.00	**2,394,800.00**
Motor Insurance	10,692,000.00	**10,692,000.00**
Total(InsuredPolicyName1)	**16,981,600.00**	**16,981,600.00**

Figure 6.24: *Data Values from Claims Transactions Data Mart for Query* 6

6.3 Rate of Query Processing

As part of ensuring that these evaluation criteria are satisfied in our query results from the queries posed to the data marts and data warehouse, we also had to observe the *rate of processing* of these queries. With the data coming from these independent data marts and being fused into a single data warehouse, an appreciable volume of expected data cannot be overemphasized. As can be expected, we observed that these aggregate queries either run at almost the same rate as being run on the multidimensional star schemas or at a slightly higher rate on the global data warehouse, as compared to that on the multidimensional star schemas.

TRANSACTIONAMOUNT		2008	2011	Total(CALENDERYEAR)
Home Insurance	ClaimDW	0.00	3,894,800.00	**3,894,800.00**
	PolicyDW	796,450.00	0.00	**796,450.00**
	Total(DataMartName)	**796,450.00**	**3,894,800.00**	**4,691,250.00**
Motor Insurance	ClaimDW	0.00	10,692,000.00	**10,692,000.00**
	PolicyDW	766,644.15	0.00	**766,644.15**
	Total(DataMartName)	**766,644.15**	**10,692,000.00**	**11,458,644.15**
Travel Insurance	ClaimDW	0.00	2,394,800.00	**2,394,800.00**
	PolicyDW	30,860.00	0.00	**30,860.00**
	Total(DataMartName)	**30,860.00**	**2,394,800.00**	**2,425,660.00**
Total(INSUREDPOLICYNAME)		**1,593,954.15**	**16,981,600.00**	**18,575,554.15**

Figure 6.25: *Data Values from Global Data Warehouse for Query* 6

We recorded the query response time for 20 query executions for *Queries* 5 (*Transportation Services* data set) and *Query* 6 (*Insurance data* set) that were posed to each of the multidimensional star schemas and the global data warehouse, on a 3.20 GHz single processor with a 2 GB of RAM. The query execution durations (in milliseconds) for the data marts and data warehouses are displayed in *Table* 6.1.

It can be deduced that the query response rate for the global data warehouse was good and very promising, as compared to the individual multidimensional star schemas. We present a summary of the variances in the average query response time (in milliseconds) for the multidimensional star schemas in comparison to the merged data warehouse in both the Insurance and Transportation data sets, as displayed in *Table* 6.2.

6.4 Summary

In this Chapter, we first discussed about the need to evaluate the work done in the preliminary introduction, and went on further to discuss the criteria for evaluation. These criteria were explained in the demonstration of experiments that were conducted, where we performed experiments based on general query processing, dimensional hierarchy, and aggregate query processing. Queries were first processed on the independent multidimensional star schemas and then on the generated global data warehouse. The data values generated from each of the data marts and data warehouse were compared side-by-side to check for the consistency in the specified criterion under consideration.

The rate of query processing was also critically considered as the methodology of integration will deal with an appreciable amount of data volumes from the integrating data marts. Consequently, we recorded the query response time for one of the experiments on both the

Table 6.1: *Summary of Query Response Time on multidimensional star schemas and Merged Data Warehouse*

Query Run No.	**Car Rental**	**Hotel Stays**	**Frequent Flyer**	**Transport DW**	**Policy**	**Claims**	**Insurance DW**
1	31	26	60	125	26	44	62
2	22	24	72	85	28	13	57
3	22	28	64	116	58	13	61
4	22	32	69	79	28	19	61
5	26	23	73	88	28	12	56
6	22	33	67	195	29	16	51
7	23	30	67	132	27	13	81
8	22	13	65	102	30	12	78
9	22	27	72	131	28	13	62
10	26	25	90	79	27	12	57
11	29	24	62	93	28	14	52
12	28	27	85	119	28	15	61
13	29	25	76	81	33	13	59
14	31	39	70	125	30	13	77
15	23	24	80	114	31	14	58
16	51	24	77	142	38	12	67
17	25	28	72	104	29	13	63
18	25	28	60	114	29	13	55
19	30	25	69	86	28	13	53
20	25	25	69	125	33	12	57
Total	**534**	**542**	**1419**	**2235**	**616**	**299**	**1228**
Average	**26.7**	**27.1**	**70.95**	**111.75**	**30.8**	**14.95**	**61.4**

Table 6.2: *Summary of Average Query Response Time & Variances*

Experimental Data Set	**Type of Data Mart / Data Warehouse**	**Average Query Response Time**	**Response Time Variance**
Transportation	Car Rental Data Mart	26.7	85.05
Transportation	Hotel Stays Data Mart	27.1	84.65
Transportation	Frequent Flyer Data Mart	70.95	40.8
Transportation	Merged Transportation Data Warehouse	**111.75**	**Not Applicable**
Insurance	Policy Transactions Data Mart	30.8	30.6
Insurance	Claims Transactions Data Mart	14.95	46.5
Insurance	Merged Insurance Data Warehouse	**61.4**	**Not Applicable**

individual multidimensional star schemas and the global data warehouse.

Chapter 7 concludes this thesis, where we summarize the major propositions. We detail the major contributions, areas of applications of the research thesis, as well as possible areas of open issues and future work.

Chapter 7

Conclusion

In presenting the concluding viewpoints on the paradigm of this thesis, we discuss the general summary of our work in this Chapter. To this end, we discuss the summary in Section 7.1, the main contributions and applications of the thesis in Sections 7.2 and 7.3, respectively. In Section 7.4, we address some areas of open issues and future work based on which the research may be pursued further.

7.1 Discussions

Schema merging is the procedure of combining both the schema and data from different - related or unrelated - independent metadata models into a single unified metadata model from which the necessary information - for example, correct data values from processed queries - can be derived. The meta-data models that we use in such integration procedures could have varied degree of element relationship levels. This fact makes processes for such integration procedures exhibit an appreciable level of complexity. This concept of integration has been performed in diverse ways in various studies, surveys and reviews conducted, and these have been handled in the generic sense or in specific cases where the metadata model is explicitly defined.

The general approach of data integration is always composed of the procedures that range from the fundamental work of finding mapping correspondences, discovery of mapping models, transforming of mapping model relationships into view definitions, implementation of a merge algorithm, amongst a few other intermediate procedures that might have to be performed; either depending on the semantics of the chosen metadata models or on some of the constraints likely to be encountered during the performance of any of the procedures.

In this thesis, we presented a methodology for the integration of a chosen metadata

model, which was a *star schema* multidimensional data model or in other terms, *star schema* data marts. The main idea behind the integration approach was to generate a global data warehouse that could independently represent any of the data marts, without referencing the source data mart. We presented three (3) main streamlined procedures for executing this methodology; where we discussed the procedural steps of *schema matching*, *mapping models discovery*, *schema merging* - with the merge operation coupled with instance data integration. We discuss these procedural steps as part of outlining our contributions in the next section. It will be noted that the success of our approach in integrating the multidimensional star schemas was largely dependent on the efficient processes we adopted in each of the procedural steps leading to the generation of expressive outputs at each stage. Moreover, such expressive outputs generated in each step became important ingredients in the set of inputs needed for the processes in the subsequent procedural step.

7.2 Contributions

In this Section, we discuss the main contributions of the thesis in line with the methodology adopted in Chapter 4, and the implementation procedures and processes in Chapter 5, and evaluation results based on the query processing and data analysis in Chapters 6. We summarize the technical contributions as follows:

- ***Multidimensional Star Schemas*** We adopted *star schemas* as candidates for our chosen multidimensional data model, highlighting on the *fact* and *dimension tables*, and *surrogate keys*. The star schema that we used offered a good platform in easily identifying the elements during the schema matching procedure of finding attribute correspondences, as there were no snowflakes in the schemas to make the correspondences difficult to be established. Additionally, the absence of snow-flaking in the schemas enabled the merge algorithm executed better, without any inherent join relationships in the attribute structures in the dimension tables and eliminating high running-time complexities that could be encountered.

- ***Hybrid Schema Matching*** We adopted a hybrid form of *schema matching* in which we used both schema-based and instance data algorithms to deliver correct attribute mapping correspondences. The hybrid approach that we adopted in this thesis made our integration methodology draw on both the schema structure and constraints, and also the instance data of the star schema data marts. Since the use of schema structure

and constrains alone could be misleading in finding attribute correspondences, we used the instance data as a sure way of validating correct mapping correspondences earlier generated by the schema algorithms or correcting earlier matching candidates. We also used different forms of schema matching algorithms, in either *schema-level* or *instance-level*, where we enforced an ordering to the execution of these algorithms as well as performing some manipulations and configurations on these algorithms. This made the schema matching procedural step very effective in determining matching candidates.

- ***First-Order GLAV Mapping Model*** We adopted *first-order* GLAV mapping models in the mapping discovery procedure, which expressed the transformation of complex expressions between attributes of the schema tables. The GLAV mapping models that we used offered us the opportunity to define complex transformation formulas for differing cardinalities between the element attributes of the star schemas. The composition of the mapping models also facilitated the expression of type of mapping relationship between the attributes, and the definition of a unique representation of either a merged attribute or data type for the supposed mapping relationship. The ability of processing these mapping relationships into executable forms - in either view definitions or output file formats - also presented a strong merit of the chosen mapping model, as it was a significant input in the merge algorithm.

- ***Conflicts Resolution*** We outlined some specific conflict resolution measures as a result of integration of the multidimensional star schemas. Integrating schemas and instance data are always inherent with some conflicts which arise due to the different representations of the same real-world entity and entity properties. In this integration methodology, our implementations lead us to deal with conflicts from different perspectives. *First*, we resolved the conflict that related to the same real-world entities from different dimensions that had the different identifiers of *surrogate keys*. Since these surrogate keys are the usual identifiers for the most data marts dimension table entities, we choose our representative identifier as that our *preferred* data mart, whilst reassigning the conflicting one. *Secondly*, for conflict that that to do deal with different real-world entities but with the same identifier of *surrogate keys*, we resolved it also by using the *preferred* data mart. *Thirdly*, for conflicts that had to deal with attribute value properties of data type, we resolved it by first using the mapping model, and then by using predefined set of attributes.

- ***Merge Correctness Requirements*** We defined some technical qualitative merge correctness requirements which served to validate the formulation of the merge algorithm. To enable the formulated merge algorithm generate a global data warehouse which satisfies all the query processing needs of the individual data marts and also to exhibit the characteristics of the these data marts, we outlined a set of technical correctness requirements for the merge algorithm. These requirements facilitated the validation of each of the statements in the algorithm, and lead to an efficient output of the expected data warehouse. These requirements were *Dimensionality Preservation*, *Measure and Attribute Entity Preservation*, *Slowing Changing Dimension Preservation*, *Attribute Property Value Preservation*, and *Tuple Containment Preservation*.

- ***Formulated Merge Algorithm*** We formulated a merge algorithm that specifically dealt with the integration of schema and instance data of the data marts. This merge algorithm was to demonstrate the model management operation of *merge*, in executable forms. It took as inputs the mapping models formalisms, as well as the schema and instance data of the data marts. The algorithm was designed to satisfy the technical *MCRs* and also resolve all conflicts.

These contributions enabled the evaluation of a successful integration approach for data marts, which have some applications in different scenarios. We discuss some of these application areas for this form of integration methodology for data marts in Section 7.3.

7.3 Applications

This thesis work and its implementation prototype provide a pedestal for some areas of applications in the commercial industry. In this Section, we discuss two (2) of these likely areas.

Suppose we have the scenario where two (2) or more companies are involved in mergers and acquisitions, and as a result their independent corporate data have to be merged into one complete data source, as it should be a single organization. This kind of development will force the integration of the data either from internal or external sources. An instance for the case of internal sources, data from the *Human Resources Department*; containing information

such as *Employee*, *Department Type*, amongst othes, from the *Procurement Department*; containing information such as *Vendor*, *Product*, *Contract Terms*, *Purchase Agent*, amongst others, or from the *Accounting Department*, containing information such as *General Ledger Book Name*, *General Ledger Chart of Accounts*, *General Ledger Organizations/Companies*, amongst others, will have to be merged into single of such data marts.

Consequently, forming an organizational-wide data warehouse from these departmental-based data marts is also not farfetched as these scattered data mart sources from the merging companies might continue to pose inherent query processing difficulties. This makes the research study an important background methodology for such forms of data integration, where identification of key dimensions and attributes relationships and conflicts resolution measures are essentially handled.

Another instance where this research can be applied is in the area of single organization or company which attempts to form an enterprise-wide data warehouse from multiple departmental-wide data marts. In such a scenario, the need to identify all related data in each of the data marts becomes critical. In this line, consider data from different data marts such as in an Insurance industry, *Policy* and *Claims Transactions*. A *Policy Holder* dimension in the *Policy Transactions* data mart will be the same real-world entity in the *Insured Party* dimension in the *Claims Transactions* data mart. Additionally, a typical *Policy Coverage* dimension could be the same representative real-world *Insured Policy* dimension in the *Policy Transactions* and *Claims Transactions* data marts, respectively.

This approach of forming a data warehouse is much less laborious, as the relationship between the dimensions and attributes on either side of the set of data marts are easily established, and merging these schemas, alongside their contained data, is achieved with fewer tasks. Though this form of merging is less tedious, a number of conflict resolution measures that will have to be addressed have been discussed in this literature.

7.4 Open Issues and Future Work

We envision some areas of open issues and future work, as part of this proposed integration methodology and the type of meta-data model adopted, that is, the multidimensional data model. In terms of some open issues, we deal with the enrichment of the mapping language to handle *Functional Dependencies* in between the attributes of fact and dimension tables. Moreover, other issues of dealing with the introduction, and handling, of *Integrity Constraints* at the multidimensional star schemas into the global data warehouse need to be addressed. For example, this is relevant in the case where there are active rules for data population in

the global data warehouse. This can be in the form of, say, enforcing on the basic limit on the insurance *Policy Coverage*, say homeowners fire protection, being the appraisal value of the insurance *Policy Covered Item*, say a 4-bedroom home.

There are a number of areas of future work to be pursued further, and we outline some of them briefly.

Firstly, we envision an approach of extending the techniques for the schema matching procedure where we apply machine learning techniques of *de-duplication* and *record linkage*. This, we believe will enhance the possibility of generating efficient attribute mapping correspondences.

Secondly, the extension of the integration methodology to handle *snowflaking* in the multidimensional data models. Hence, the capability our methodology to handle *Snowflake* and *Fact Constellation* schemas.

Thirdly, the optimization (speed) of the merge algorithm to handle extremely large number of data marts – in terms of large quantities of facts and dimension tables – and run in less time complexity of minutes or few hours. Additionally, the consideration of high volume of data contained in the data marts for integration. These issues are normally encountered in the case of institution (company) mergers and acquisitions.

Fourthly, the capability of the generated data warehouse to efficiently handle *Changing Dimensions*, in terms of Slowly, Medium, or Fast. This will enable the ability to analyze fact records relating to the multiple representations of a single entity in a dimensions at the data marts.

Fifthly, the integration methodology to handle *Multi-valued Dimension Attributes* in the dimension tables, in terms of two (2) associations.

One, the association of multiple entities with a single account in the dimension tables; where for e.g. in an Insurance industry, we have multiple *Policy Holders* sign on to one insurance *Policy Coverage* account, as in the case of group or family health insurance scheme, and each of the *Policy Holders* are representatively unique and might sign on for other *Policy Coverages* as well.

Two, the association of a single entity with multiple classifications in the dimension tables; where for e.g. in an Insurance industry, a single commercial *Policy Holder* may be associated with one or more *Standard Industrial Classification* (SIC) of insurance policies such as Fire, & Marine Insurance, Life Insurance, Home Owner Insurance, Accident & Health Insurance, amongst others. Another example could be in the health care industry, where a single *Patient* has one or more *Diagnoses*, of say Lung Cancer and Respiratory Disorders, all at the same time in the line of treatment or billing at a single attendance at the health care facility.

Appendix A

Merge Algorithm Complexity and Proof of *Correctness*

A.1 Preliminaries

For the merge algorithm formulated in Section 4.4.3, we present the following criteria to substantiate the worst-case polynomial time complexity and proof of correctness;

1. *Soundness*

2. *Completeness*

Definition A.1.1 *A **Query** is said to be **Certain** iff it is true in all instances of a Multidimensional Database, M and satisfies the properties and semantics of the elements of M.* ∎

Definition A.1.2 *A **Tuple** forming an **Answer** to a query is said to be **Certain** iff it is the intended, meaningful, and acceptable answer to a posed Certain Query in a Multidimensional Database, M and it is true for all instances of M.* ∎

The criteria of *Soundness* and *Completeness* are proven to clarify the validity of the algorithm in providing intended *Certain Answers* to its intended queries. For *Soundness*, we want to make sure that the answers to queries from the global data warehouse are in fact in the syntactical meaning of the algorithm, and therefore all computed answers to queries posed are *True*.

In other words, we want to state that the given answers from a given global data warehouse are true for all instances of the application of the algorithm. Additionally, the truth of the answers to queries posed on the global data warehouse means the answers are *Certain Answers*, and are valid for the global data warehouse and also valid for which ever set of local data marts it may be posed to.

For *Completeness*, we want to make sure that any *Certain Answer* to a query that can be attained for a given global data warehouse can be computed for that global data warehouse in comparison to its associated local data marts.

It means that for the *Completeness* criterion; any *Certain Answer* to a query posed to the global data warehouse should be proven or computed to exist, just as it exists in the local data mart. In other words, we want to make sure that our algorithm does not miss any *Certain Answer* to a posed *Certain Query*. The total *Completeness* criterion is trivially the converse of the *Soundness* criterion and partially contains or proves it.

Theorem A.1.1 *Let S and I, respectively, represent the Schema and Instance Data of MultiDimensional Star Schema, M; which contains a Fact Table, F and k Dimension Tables, D_i, $\{1 \leq i \leq k\}$. Then, a merge algorithm which accepts n Star Schemas, M_j, $\{2 \leq j \leq n\}$ and Mapping Model, MAP_{FD} as inputs, generates a Global Data Warehouse, DW in a worst-case polynomial time complexity.* ∎

PROOF: To outline the proof clearly we adopt some notational conventions which will better illustrate the sketch of the proofs.

Let A represent an expected *Tuple variable* ranging over a set of queries. Let X, Y, and Z represent a set of possible and certain *Queries* likely to be posed to the global data warehouse. For the *Tuple A* proving a *Query X* will mean the tuple computes answers to the query posed to the global data warehouse.

A.2 Proof of *Soundness*

Proof To prove the *Soundness* of the algorithm, we want to show that:

(SKETCH) If a *Tuple A* can be proven or computed as an answer to a posed *Query X*, then *Tuple A* will imply *Query X*. In other words the *Tuple A* that can be derived will form

the set of intended *Certain Answers* to the posed *Query X*.

($\Rightarrow$)

1. By use of inductive definition, we assume for an arbitrary *Tuple A* and *Query X* and that the *Tuple A* is computed in n number of steps; which is fewer as expected. Consequent to this assumption, the *Tuple A* will represent *Certain Answers* to the *Query X*; and for all instances of a global data warehouse from this algorithm the Tuples generated will imply the Queries posed.

2. For Step (2) in the algorithm, it can be inferred that once the mapping and the correspondence between the attributes of the local data marts Fact Tables are iterated through in finite steps (because of their finiteness of attributes), the global data warehouse will represent a Fact Table with attributes from which any query likely to be posed to it will generate set of attributes that represent the set of integrating Fact Table attributes from the local data marts. Hence, able to produce *Certain Answers* on *Tuple*, say A, for any *Query*, say X, posed to it.

3. For Step (3), an inference can be deduced that with only two (2) forms of mapping, all forms of mapping ambiguities (which might lead to undecidability) are not expected. Additionally, *Certain Answers* will be expected from a query in the sense that the *Equality* mapping will offer *Tuple* attributes that are the same to that from the local data marts. If on the contrary the exact answers for Tuples cannot be generated, similar answers are expected because of an alternative *Similarity* mapping which enforces a complex expression or transformation.

   Finally, if it happens that an expected *Tuple* will be unique to one kind of data mart and hence any query posed for such attribute is likely to pose a *Falsity* in the *Tuples* generated in the case of another data mart, the Step (d) offer a solution for all such non-corresponding attributes; where all such attributes augment earlier ones from the mapping. This makes all generated *Tuples* for queries posed in relation to the attributes of the Fact Table (and their associated Dimension Tables – because of same derivation of attributes) for the global data warehouse *True* for any instance application of the algorithm. As a result, by inductive proposition the correctness is trivially preserved.

4. For Step (7), the *tuples* that are generated from the global data warehouse will have attribute properties being the *UNION* of all integrating attributes. If a *Tuple*, say A, is generated for a *Query*, say Y, a truth validity can be ascertained in the sense that the tuple will represent a Certain answer to such a query having the unique property of being able to entirely represent any of the integrating attribute properties. This makes the inference and inductive claims from the earlier premise satisfy and preserve the correctness criteria. ∎

A.3 Proof of *Completeness*

We will adopt the same notational conventions from above.

Proof To prove the *Completeness* of the algorithm, we want to show that:

(SKETCH) If a *Tuple A* is a *Certain Answer* to a *Query Z* posed to the global data warehouse, then the *Tuple A* can be proven to exist.

In other words, for any *Query Z* posed we are sure not to miss any *Certain Answer* from the tuples that can be generated. In this prove, we will supposition that the global data warehouse might miss some *Certain Answers*, that can still be proven to exist. This supposition will become evident in the sense that the effect of *Similarity Mapping* between attributes and some missing hierarchy attributes from the merge process, the subsequent usual aggregate queries that will be posed to the global data warehouse will make it trivially possible to miss a few *Certain Answers*.

($\Leftarrow$)

1. We begin the proof by the hypothesis of contraposition, and show that: If a *Tuple*, say A, cannot be computed or generated for a *Query*, say Z, then the *Tuple A* cannot represent a *Certain Answer* to the *Query Z*.

2. Let us assume the aggregated *Tuple A* cannot be computed or generated for the *Query Z* in the strong sense.

3. If the *Tuple A* cannot be computed, then we can construct an infinite general set, S^* of aggregated Tuples with different combination of attributes (because of the *Equality* or *Similarity Mapping* from the Mapping output) for the particular query in question, *Query Z*, and which will still not form computed tuples needed enough to answer the intended *Query Z*.

   (a) A few definitions and inductions will then be made based on this construction.

   (b) We can generate a categorization of all forms of aggregations with different projections on attributes and aggregation types that can compute *tuples* for a *Certain answer*; we enumerate them as $E_1, E_2, \ldots$

   (c) We then will inductively define a series S_n of different sets of tuples (S_0,S_1,...);

      i. We then let the first of the series of *tuple* sets, S_0 represent the arbitrary *Tuple A*;

      ii. As part of the inductive construction, if the union of one set of a *tuple*, say S_k and a subsequent categorization, say E_{k+1} is a computed tuple to answer *Query Z*, then we have both the *initial tuple* and the *new tuple* having the same form of answer. This will mean that if we have any *subsequent tuple* with a bit more aggregation input or modification and attribute projections, and that addition still makes it a *Certain answer* to *Query Z*; then the additional projected attributes or aggregation constructs did not change the certainty of the answer for the *Query Z*.

      iii. On the other hand, if the union of one set of a tuple, say S_k and a subsequent categorization, say E_{k+1} does not form a computed tuple needed to answer *Query Z*, then the new tuple, S_{k+1} is definitely giving us a different form of answer from the initial one, S_k. This will mean that each set of *tuples* with an addition of projected attributes and also additional aggregation constructs makes the *tuples* different enough to give different answers to the same *Query Z* posed to the global data warehouse. Hence, able to change the validity of

the answer to the query.

4. We will then have the general set S^* representing the combination of all the aggregated tuples likely to give an answer to the query.

5. It will then be deduced that;

   (a) The general set S^* holds our supposed *Tuple A*.

   (b) The general set S^* does not provide enough computed tuples to form a *Certain answer* to the posed *Query Z*; because if the general computed tuples set formed a *Certain answer* then we should have some additional attribute projections as well as other added aggregations to any of the member tuple set, say S_k, make it a valid *Certain answer* the query.

   (c) The general set S^* is encompassing enough in relation to our supposed *Tuple A*, and in the sense that if we were able to add some projected attributes and aggregations to the general set, these additions should be well enough to compute tuples to form *Certain answers* to the query. And these additions could have been made during the construction of all the individual set phases of the general set.

6. For our general set S^* of computed tuples to be encompassing enough, then it has a satisfiability property where if some attribute projections and aggregations make such a computed tuple to become a certain answer to a query, it will always be true and never be false.

7. With such a satisfiability property, we can say that there is always a judgment on constitution of the general set of computed *tuples* making all its generated *tuples* true in the context of answering a particular query and anything outside it false. As a result, this will make our computed *Tuple A* always true and make the posted *Query Z* false.

8. This assertion of the *Tuple A* being true and the posted *Query Z* being false does not offer a good basis for the computed tuple validating as a *Certain answer* to the posted *Query Z*. Hence, our preceding proposition of contraposition is satisfied and valid. ∎

Appendix B

Glossary of Terms

B.1 Abbreviations

CIM Conceptual Integration Model

SQL Structured Query Language

XML Extensible Markup Language

XML DTD XML Document Type Definitions

LAV Local-As-View

GAV Global-As-View

GLAV Global-And-Local-As-View

MSNF Mediated Schema Normal Form

DW/BI Data Warehouse / Business Intelligence

OLTP Online Transaction Processing

OLAP Online Analytical Processing

ROLAP Relational Online Analytical Processing

MOLAP Multidimensional Online Analytical Processing

HOLAP Hybrid Online Analytical Processing

DBMS Database Management System

ETL Extract, Transformation, Load

P2P Peer-To-Peer

SF Similarity Flooding

GUI Graphical User Interface

SQL DML SQL Data Manipulation Language

SQL DDL SQL Data Definition Language

XSLT Extensible Stylesheet Language Transformations

TGD Tuple Generating Dependency

OWA Open World Assumption

CWA Closed World Assumption

OWL Web Ontology Language

ASCII American Standard Code for Information Interchange

GMR Generic Merge Requirement

MCR Merge Correctness Requirement

OODBMS Object-Oriented Database Management System

IDE Integrated Development Environment

OOP Object-Oriented Programming

RAM Random Access Memory

B.2 Acronyms and Technical Terms

COMA Combining Match Algorithms

XQuery A query and functional programming language that is designed to query collections of XML data

Appendix C

Experimental Data Sets (Star Schema Source Data Marts)

C.1 Insurance Data Set

C.2 Transportation Services Data Set

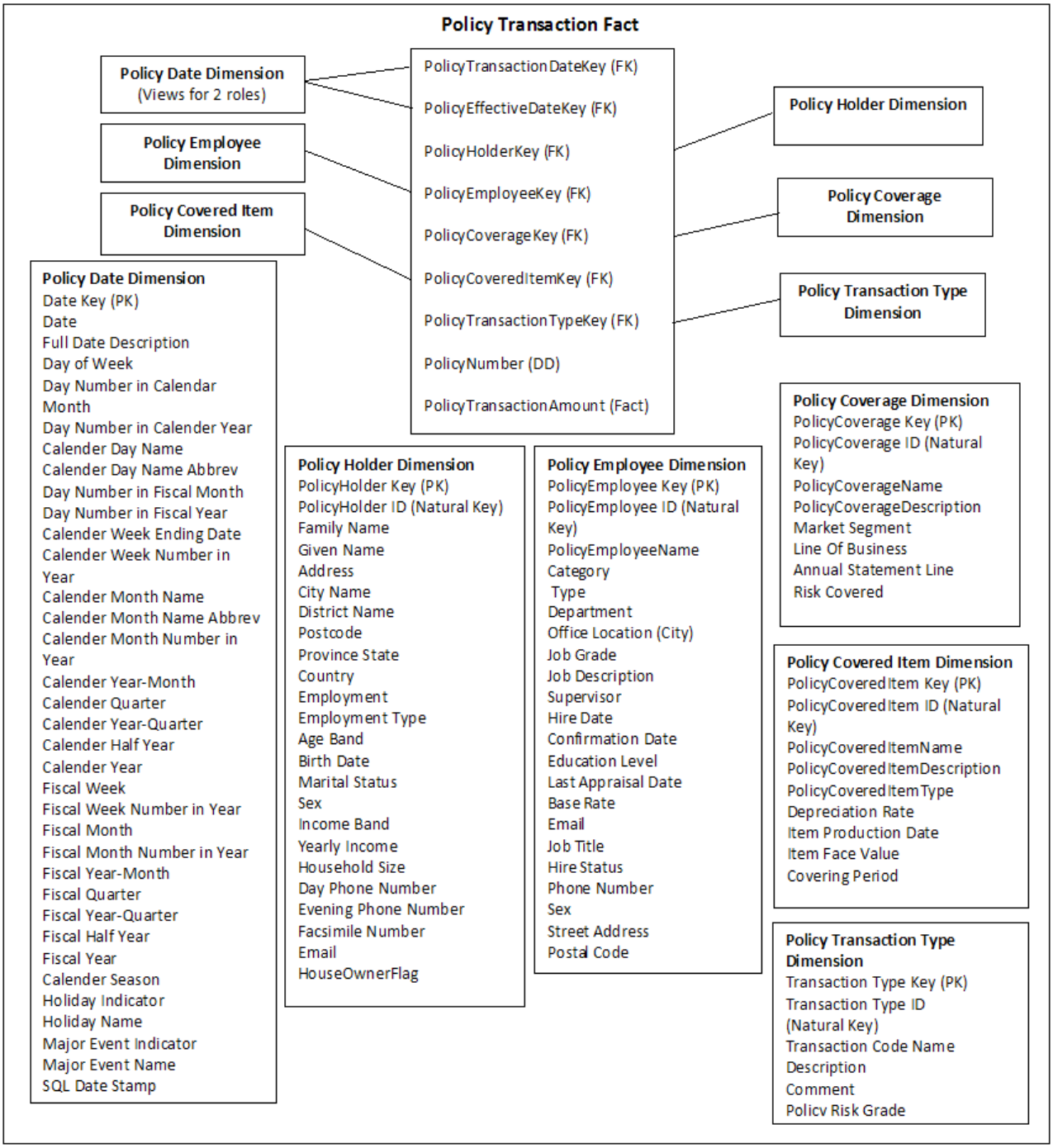

Figure C.1: *Policy Transactions Data Mart*

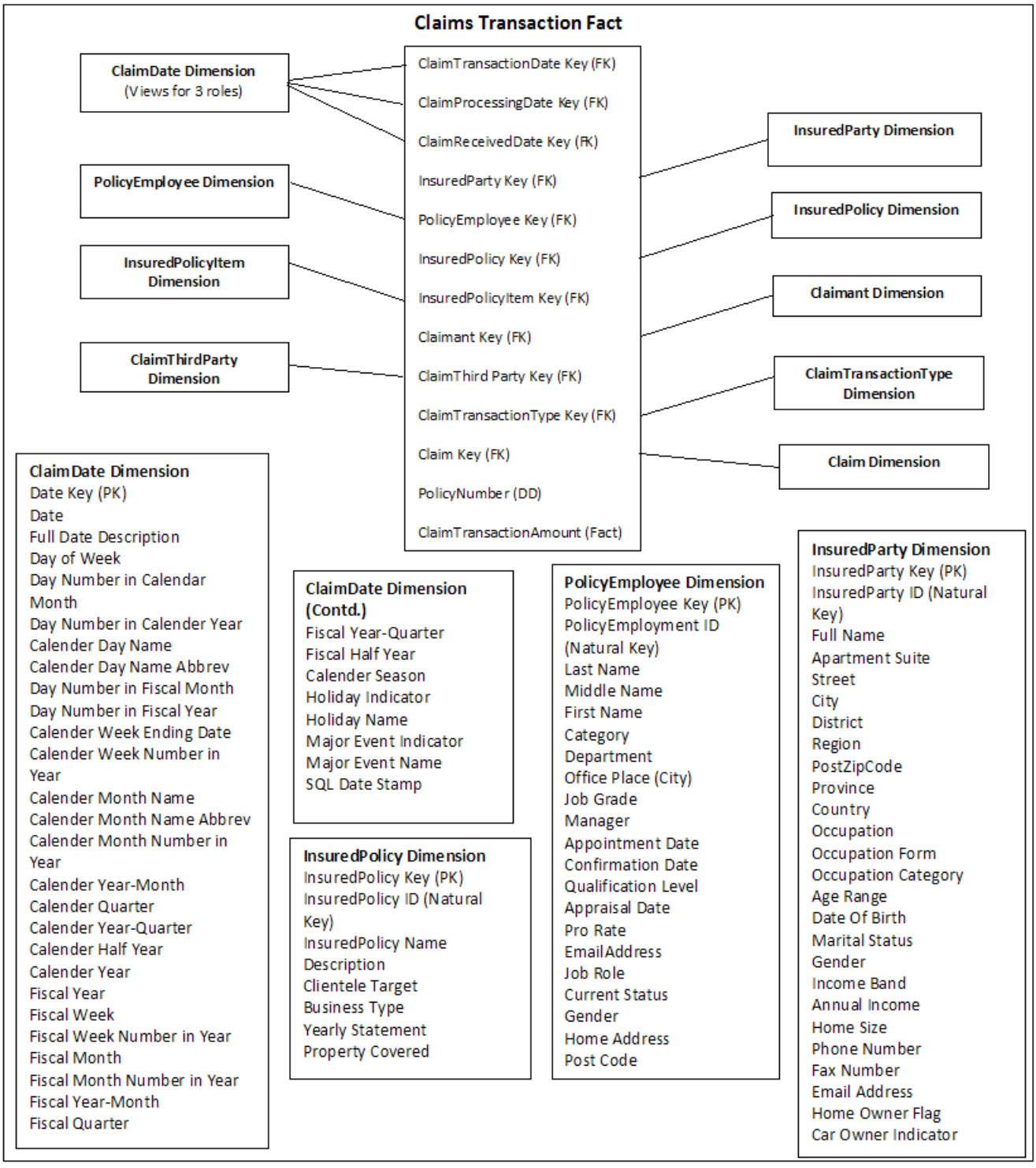

Figure C.2: *Claims Transactions Data Mart - Part 1*

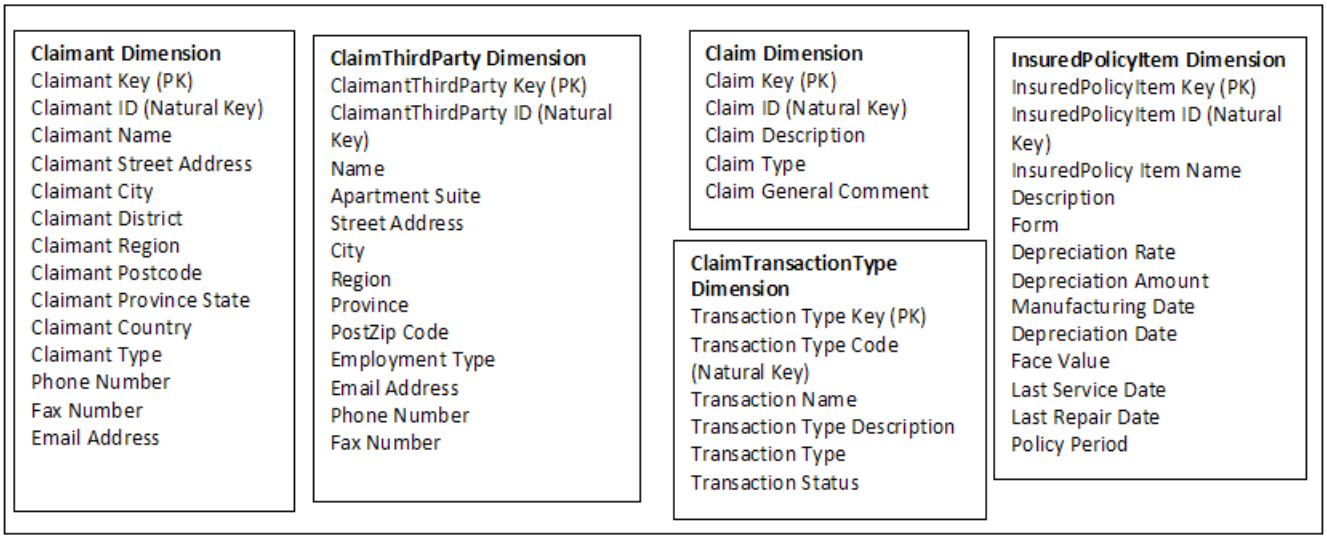

Figure C.3: *Claims Transactions Data Mart - Part 2*

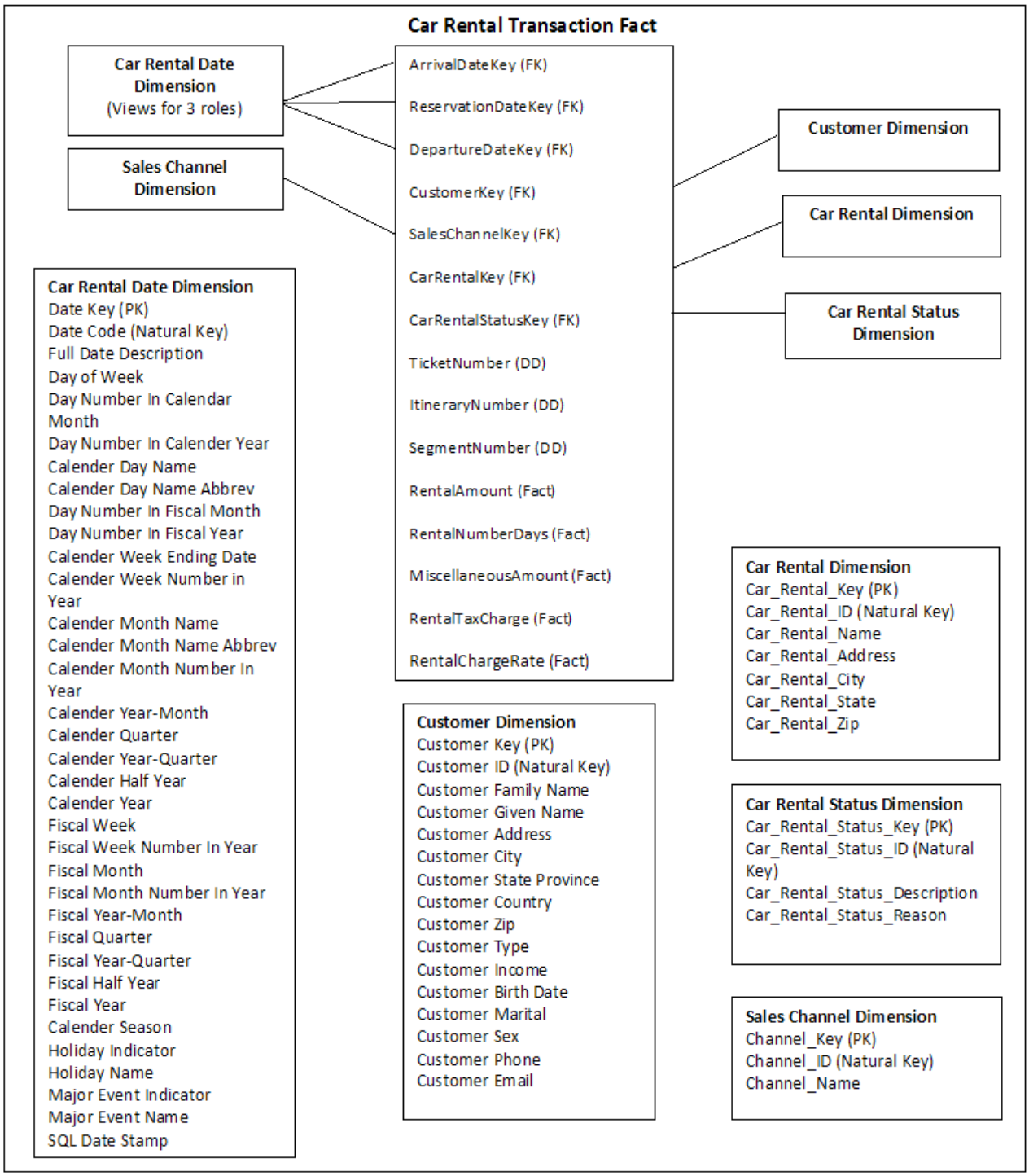

Figure C.4: *Car Rental Transactions Data Mart*

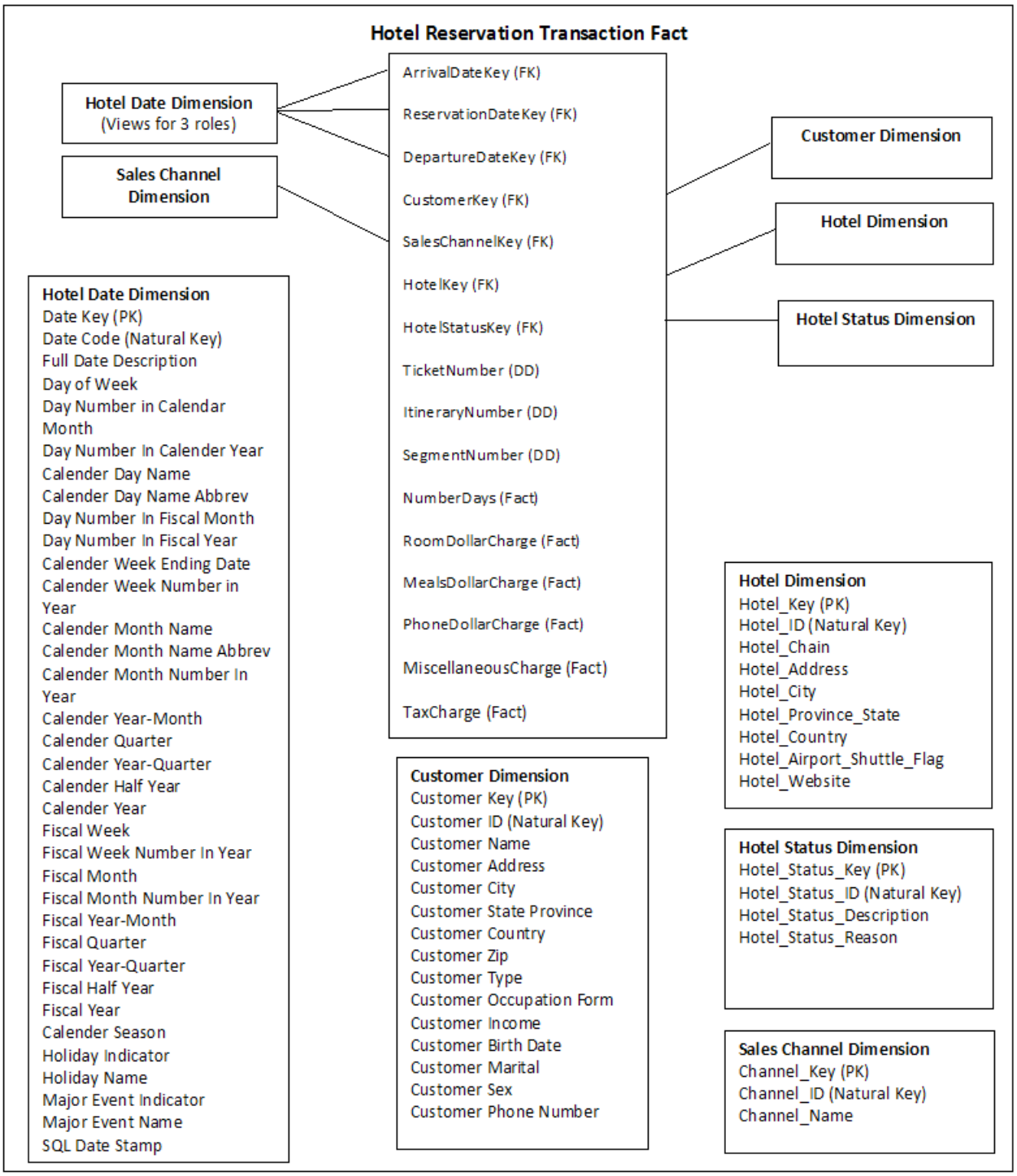

Figure C.5: *Hotel Reservations Transactions Data Mart*

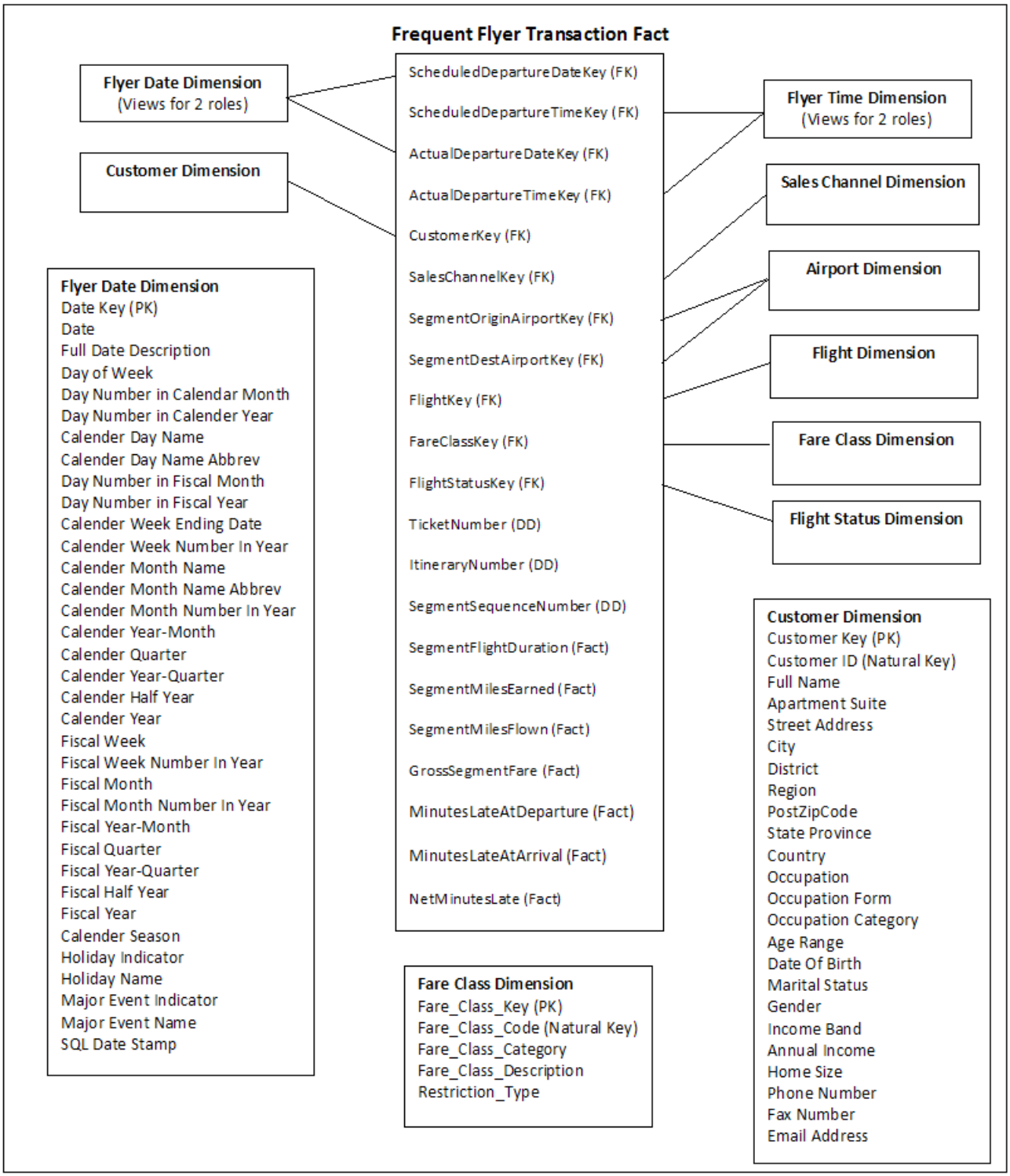

Figure C.6: *Frequent Flyer Transactions Data Mart - Part 1*

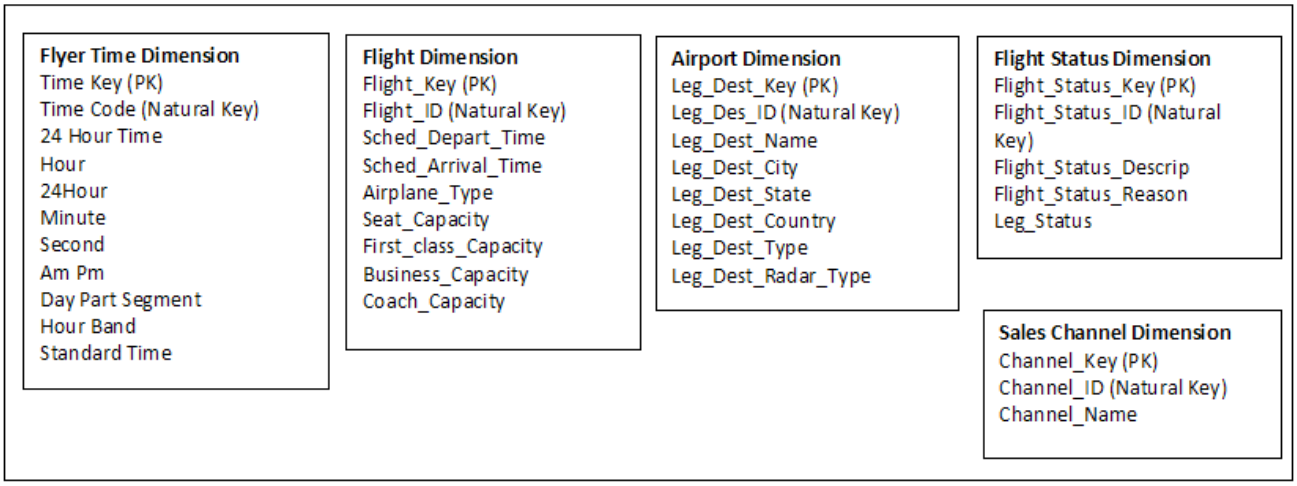

Figure C.7: *Frequent Flyer Transactions Data Mart - Part 2*

Appendix D

Bibliography

Bibliography

[1] P. Andritsos, R. Fagin, A. Fuxman, L. M. Haas, M. A. Hernández, C. T. H. Ho, A. Kementsietsidis, R. J. Miller, F. Naumann, L. Popa, Y. Velegrakis, C. Vilarem, and L-L Yan. Schema Management. *IEEE Data Engineering Bulletin (DEBU) 25(3):*, pages 32–38, (2002).

[2] P. C. Arocena, A. Fuxman, and R. J. Miller. Composing Local-As-View Mappings: Closure and Applications. In *Proceedings of the 13th International Conference on Database Theory*, pages 209–218. ICDT 2010.

[3] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, Volume 18:323–364, (1986).

[4] S. Berger and M. Schrefl. From Federated Databases To A Federated Data Warehouse System. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 394. HICSS 2008.

[5] J. Berlin and A. Motro. Database Schema Matching Using Machine Learning with Feature Selection. In *Proceedings of 14th International Conference on Advanced Information Systems Engineering*, pages 452–466. CAiSE 2002.

[6] P. A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research*. CIDR 2003.

[7] P. A. Bernstein, T. Bergstraesser, J. Carlson, S. Pal, P. Sanders, and D. Shutt. Microsoft Repository Version 2 and the Open Information Model. *Web, Web-Services, and Database Systems*, Volume 24(Number 2):71–98.

[8] P. A. Bernstein, A. Y. Halevy, and R. A. Pottinger. A Vision of Management of Complex Models. In *Proceedings of the 19th ACM SIGMOD International Conference*

on Management of Data, pages 55–63. SIGMOD Record (SIGMOD) 29(4) and Technical Report, (2000).

[9] P. A. Bernstein and S. Melnik. Model Management 2.0: Manipulating Richer Mappings. In *Proceedings of the 26th ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM SIGMOD 2007.

[10] P. A. Bernstein, S. Melnik, and J. Churchill. Incremental Schema Matching. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 1167–1170. VLDB 2006.

[11] P. A. Bernstein and E. Rahm. Data Warehouse Scenarios for Model Management. In *Proceedings of the 19th International Conference on Conceptual Modelling*, pages 1–15. ER 2000.

[12] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. In *Proceedings of the 14th International Conference on Database Theory*, pages 268–279. ICDT 2011.

[13] P. Buneman, S. B. Davidson, and A. Kosky. Theoretical Aspects of Schema Merging. In *Proceedings of the 3rd International Conference on Extending Database Technology*, pages 152–167. EDBT 1992.

[14] L. Cabibbo and R. Torlone. Dimension Compatibility for Data Mart Integration. In *Proceedings of the Twelfth Italian Symposium on Advanced Database Systems*, pages 6–17. SEBD 2004.

[15] L. Cabibbo and R. Torlone. Integrating Heterogeneous Multidimensional Databases. In *Proceedings of the 17th International Conference on Scientific and Statistical Database Management*, pages 205–214. SSDBM 2005.

[16] L. Cabibbo and R. Torlone. On the Integration of Autonomous Data Marts. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, page 223. SSDBM 2004.

[17] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data Integration in Data Warehousing. *International Journal of Cooperative Information Systems*, Volume 10(Number 3):237–271, (2001).

[18] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Simplifying Schema Mappings. In *Proceedings of the 14th ACM International Conference on Database Theory*, pages 114–125. ICDT 2011.

[19] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. View Synthesis from Schema Mappings. *The Computing Research Repository*, CoRR abs/1003.1179, 2010.

[20] S. Craw. Manhattan Distance. *Encyclopedia of Machine Learning*, page 639.

[21] M. Dash and H. Liu. Feature Selection for Classification. *Intelligent Data Analysis*, 1(3):131156.

[22] C. Dell'aquila, F. Di Tria, E. Lefons, and F. Tangorra. Logic Programming for Data Warehouse Conceptual Schema Validation. In *Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery*, pages 1–12. DaWak 2010.

[23] E. Deza and M. M. Deza. Euclidean Distance. *Encyclopedia of Distances*, page 94.

[24] R. Dhamankar, Y. Lee, A. Doan, A. Y. Halevy, and P. Domingos. iMAP: Discovering Complex Mappings between Database Schemas. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 383–394. ACM SIGMOD 2004.

[25] H. H. Do, S. Melnik, and E. Rahm. Comparison of Schema Matching Evaluations. *Web, Web-Services, and Database Systems*, pages 221–237, 2002:.

[26] 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 Data Bases*, pages 610–621. VLDB 2002.

[27] A. Doan, P. Domingos, and A. Y. Levy. Learning Source Description for Data Integration. In *Proceedings of the Third International Workshop on the Web and Databases*, pages 81–86. WebDB (Informal Proceedings) 2000.

[28] R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema Mapping Creation and Data Exchange. *Conceptual Modelling:*, pages 198–236.

[29] M. Friedman, A. Y. Levy, and T. D. Millstein. Navigational Plans For Data Integration. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, pages 67–73. AAAI/IAAI 1999.

[30] A. Fuxman, M. A. Hernández, C. T. H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested Mappings: Schema Mapping Reloaded. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 67–78. VLDB 2006.

[31] A. Gal. Managing Uncertainty in Schema Matching with Top-K Schema Mappings. *Journal on Data Semantics VI:*, pages 90–114.

[32] I. Gam and C. Salinesi. A Requirement-driven Approach for Designing Data Warehouses. In *Proceedings of the 12th International Working Conference on Requirements Engineering*. REFSQ 2006.

[33] V. Ganti. Data Cleaning. *Encyclopedia of Database Systems*, pages 561–564.

[34] G. Gottlob and P. Senellart. Schema Mapping Discovery from Data Instances. *Journal of the ACM*, Volume 57(Number 2).

[35] M. N. Gubanov, P. A. Bernstein, and A. Moshchuk. Model Management Engine for Data Integration with Reverse-Engineering Support. In *Proceedings of the 24th International Conference on Data Engineering*, pages 1319–1321. ICDE 2008.

[36] L. M. Haas, M. Hentschel, D. Kossmann, and R. J. Miller. Schema AND Data: A Holistic Approach to Mapping, Resolution and Fusion in Information Integration. In *Proceedings of the 28th International Conference on Conceptual Modelling*, pages 27–40. ER 2009.

[37] L. M. Haas, M. A. Hernández, C. T. H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 805–810. SIGMOD 2005.

[38] L. M. Haas and A. Soffer. New Challenges in Information Integration. In *Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery*, pages 1–8. DaWak 2009.

[39] A. Y. Halevy. Technical Perspective – Schema Mappings: Rules for Mixing Data. *Communications of the ACM*, Volume 53(Number 1):100.

[40] A. Y. Halevy and J. Madhavan. Corpus-Based Knowledge Representation. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1567–1572. IJCAI 2003.

[41] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, second edition edition, (2006).

[42] M. A. Hernández, R. J. Miller, and L. M. Haas. Clio: A Semi-Automatic Tool For Schema Mapping. In *A Workshop Presentation at ACM Conference*, page 607. ACM SIGMOD 2001:.

[43] M. A. Hernández, L. Popa, C. T. H. Ho, and F. Naumann. Clio: A Schema Mapping Tool for Information Integration. In *Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms, and Networks*, page 11. ISPAN 2005.

[44] M. A. Hernández, L. Popa, Y. Velegrakis, R. J. Miller, F. Naumann, and C-T. H. Ho. Mapping XML and Relational Schemas with Clio. In *Proceedings of the 18th International Conference on Data Engineering*, pages 498–499. ICDE 2002.

[45] Ibm. IBM Infosphere Data Architect 7.5.3.0:. *http://www-01.ibm.com/software/data/optim/data-architect/*, September(2011).

[46] Ibm. IBM Infosphere Data Architect 7.5.3.0: Finding Relationships. *http://publib.boulder.ibm.com/infocenter/idm/v2r1/index.jsp?topic=/com.ibm.datatools.metadata.mapping.ui.doc/topics/iiymdadconfiguring.html*, September(2011).

[47] Icde. Bulletin on the Technical Committee on Data Engineering. *International Conference on Data Engineering (ICDE)*, Volume 25(Number 3), September 2002.

[48] A. Islam, D. Z. Inkpen, and I. Kiringa. Applications of Corpus-based Semantic Similarity and Word Segmentation to Database Schema Matching. *The Very Large Data Base Journal*, Volume 17(Number 5):1293–1320.

[49] T. Jörg and S. Dessloch. Formalizing ETL Jobs for Incremental Loading of Data Warehouses. In *Proceedings of the 13th Conference on Database Systems in Business, Technology and Web*, pages 327–346. BTW 2009.

[50] G. Karvounarakis. Answering Queries Across Mappings:. *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.104.4864*, September(2011).

[51] D. Kensche, C. Quix, X. Li, Y. Li, and M. Jarke. Generic Schema Mappings for Composition and Query Answering. *Data & Knowledge Engineering*, Volume 68(Number 7):599–621.

[52] D. Kensche, C. Quix, Y. Li, and M. Jarke. Generic Schema Mappings. In *Proceedings of the 26th International Conference on Conceptual Modelling*, pages 132–148. ER 2007.

[53] R. Kimball and M. Ross. *The Data Warehouse Toolkit.* Second edition: edition, (2002).

[54] R. Kimball, M. Ross, W. Thornthwaite, J. Mundy, and B. Becker. *The Data Warehouse Lifecycle Toolkit.* John Wiley and Sons, second edition: edition, 2008.

[55] M. lenzerini. Data Integration: A Theoretical Perspective. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 233–246. ACM PODS 2002.

[56] W-S Li and C. Clifton. SEMINT: A Tool For Identifying Attribute Correspondences In Heterogeneous Databases Using Neural Networks. *Data & Knowledge Engineering*, Volume 33(Number 1):49–84.

[57] D. Linstedt, K. Graziano, and H. Hultgren. *The New Business Supermodel. The Business of Data Vault Modelling, 2nd edition.* Lulu.com, 2009.

[58] J. Madhavan, P. A. Bernstein, P. Domingos, and A. Y. Halevy. Representing and Reasoning About Mappings Between Domain Models. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence*, pages 80–86. AAAI/IAAI 2002.

[59] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 49–58. VLDB 2001.

[60] J. Madhavan and A. Y. Halevy. Composing Mappings among Data Sources. In *Proceedings of 29th International Conference on Very Large Data Bases*, pages 572–583. VLDB 2003.

[61] B. Marnette. Generalized Schema-Mappings: From Termination to Tractability. In *Proceedings of the Twenty-Eight ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 13–22. PODS 2009, 2009.

[62] S. Melnik. Model Management: First Steps and Beyond. In *Proceedings of the 11th Conference on Database Systems in Business, Technology and Web*, pages 455–464. BTW 2005.

[63] S. Melnik. *Generic Model Management: Concepts and Algorithms.* 2967, (2004).

[64] S. Melnik, A. Adya, and P. A. Bernstein. Compiling Mappings to Bridge Applications and Databases. In *Proceedings of the 26th ACM SIGMOD International Conference on Management of Data*, pages 461–472. SIGMOD 2007.

[65] S. Melnik, P. A. Bernstein, A. Y. Halevy, and E. Rahm. Supporting Executable Mappings in Model Management. In *Proceedings of the 24th ACM SIGMOD International Conference on Management of Data*, pages 167–178. SIGMOD 2005.

[66] S. Melnik, H. Garcia Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Proceedings of the 18th International Conference on Data Engineering*, pages 117–128. ICDE 2002.

[67] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema Mapping as Query Discovery. In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 77–88. VLDB 2000.

[68] R. J. Miller, M. A. Hernández, L. M. Haas, L-L Yan, C. T. H. Ho, R. Fagin, and L. Popa. The Clio Project: Managing Heterogeneity. *SIGMOD Record*, Volume 30(Number 1):78–83.

[69] K. Morfonios and Y. E. Ioannidis. Star Schema Modelling. *Encyclopedia of Database Systems:*, pages 2779–2780, 2009.

[70] L. Palopoli, D. Sacca, and D. Ursino. Semi-automatic, Semantic Discovery of Properties from Database Schemas. In *Proceedings of the International Database Engineering and Applications Symposium*, pages 244–253. IEEE Computing Society, 1998.

[71] C. H. Papadimitriou and M. Yannakakis. On the Complexity of Database Queries. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 12–19. ACM PODS 1997.

[72] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A Foundation for Capturing and Querying Complex Multidimensional Data. *Elsevier Science.*, Volume 26(Number 5):383–423.

[73] R. A. Pottinger. Database Schema Integration. *Encyclopedia of GIS:*, pages 226–231, 2008.

[74] R. A. Pottinger and P. A. Bernstein. Merging Models Based on Given Correspondences. In *Proceedings of 29th International Conference on Very Large Data Bases*, pages 826–873. VLDB 2003 and Technical Report MSR-TR-2000-53: Microsoft Research.

[75] R. A. Pottinger and P. A. Bernstein. Schema Merging and Mapping Creation for Relational Sources. In *Proceedings of the 11th International Conference on Extending Database Technology*, pages 73–84. EDBT 2008.

[76] C. Quix. Model Management. *Encyclopedia of Database Systems*, pages 1760–1764.

[77] C. Quix, D. Kensche, and X. Li. Generic Schema Merging. In *Proceedings of the 19th International Conference Advanced Information Systems Engineering*, pages 127–141. CAiSE 2007.

[78] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *Journal on International Conference on Very Large Data Bases*, Volume 10(Number 4):334–350, (2001).

[79] M. Reddy V and S. K. Jena. Active Datawarehouse Loading by Tool Based ETL Procedure. In *Proceedings of the 2010 International Conference on Information & Knowledge Engineering*, pages 196–201. IKE 2010.

[80] D. Riazati, J. A. Thom, and X. Zhang. Inferring Aggregation Hierarchies for Integration of Data Marts. In *Proceedings of the 21th International Conference on Database and Expert Systems Applications*, pages 96–110. DEXA 2010.

[81] N. Rizopoulos and P. McBrien. Schema Merging Based on Semantic Mappings. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 193–198. BNCOD 2009.

[82] F. Rizzolo, I. Kiringa, R. A. Pottinger, and K. Wong. The Conceptual Integration Modelling Framework: Abstracting from the Multidimensional Model. *The Computing Research Repository*, CoRR abs/1009.0255, 2010.

[83] G. Rull, C. Farré, E. Teniente, and T. Urpí. Validation of Mappings Between Schemas. *Data & Knowledge Engineering*, Volume 66(Number 3):414–437.

[84] M. Schneider. Integrated Vision of Federated Data Warehouses. In *Proceedings of the CAiSE-06 Workshop on Data Integration and the Semantic Web*, pages 336–347. DISWEB 2006.

[85] P. Senellart and G. Gottlob. On the Complexity of Deriving Schema Mappings from Database Instances. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 23–32. PODS 2008.

[86] P. Shvaiko. A Classification of Schema-based Matching Approaches. In *Proceedings of the Meaning Coordination and Negotiation Workshop at the International Semantic Web Conference*. (ISWC):2004.

[87] P. Shvaiko and J. Euzenat. A Survey of Schema-based Matching Approaches. *Journal on Data Semantics IV:*, pages 146–171.

[88] B. Ten Cate and P. G. Kolaitis. Structural Characterizations of Schema-Mapping Languages. In *Proceedings of the 12th International Conference on Database Theory*, pages 63–72. ICDT 2009.

[89] M. Y. Vardi. The Complexity of Relational Query Languages (Extended Abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pages 137–146. STOC 1982.

[90] Wikipedia. Data Integration:. *http://en.wikipedia.org/wiki/Data_integration*, September(2011).