User-Centric Modelling For Data Warehouses

Resmi Nair

Thesis

Submitted by Resmi Nair

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Faculty of Information Technology, Monash University

June 2007
AMENDMENTS

ERRATA

p 25, line 1 ”its” for it’s
p 201, line 3 ”its” for it’s
p 57, line 7, ”latter” for ”later”
p 174, line 8, ”latter” for ”later”
p 194, line 4, ”latter” for ”later”

ADDENDUM

p27, At the end of para 2: A UML based design that spans all the three phases of the design could be found in [NJI06]. Meta models based on UML and transformations using Object Constraint Language (OCL) are used here for the design. The application of OCL can again found in [SJS06] for multidimensional modeling.
p 27, Section 2.2, line 11: Read ”Different from above approaches.”
p 93, Section 3.2, line 4: Add ”and [VRTR05]” and read ”analysis processes such as [RB03], [FJ03], [M.D04] and [VRTR05]” ...
p 137, Section 3.7, line 3: delete ”suggested”, add ”introduced” and read ”reduction algorithm were introduced in the context of.”
A reduction/optimization algorithm has been proposed to optimize the proposed schema. Evaluation of the effectiveness and precision of the algorithm is an area that requires further study.

Add new references in the reference section:


DEDICATION

This thesis is dedicated to our loving father
Declaration

I hereby declare that this thesis contains no material which has been accepted for
the award of any other degree or diploma at any university or equivalent institution
and that to the best of my knowledge and belief, this thesis contains no material
previously published or written by another person, except where due reference has
been made.

______________________________

Resmi Nair

June 2007
Publications


Acknowledgement

A research project of this magnitude would not have been possible without the support of many individuals.

My first and foremost gratitude goes to my main supervisor Prof. Bala Srinivasan for his support and guidance. At times when I was at crossroads, wondering which way to go, Srini guided me on the right path. His valuable suggestions and feedback immensely helped me to complete the thesis. I am also grateful to my other supervisor Dr. Campbell Wilson for his contributions during research discussions. Those meetings helped me to formalise ideas and writing up of the thesis.

I thank my fellow researchers Guy, Samar and Viranga for their help during these years. I also thank Oshadi, my room-mate; she was there for me during some difficult periods in my research.

A “big thank you” to Gerry for correcting my initial drafts and assisting me with mathematical notations. I would like to thank CSSE administration and tech-
nical staff for their support during my candidature.

I must acknowledge the love and support from my friends, who helped me to pass through the research pressure. It is hard to mention all of them but Gayika is special. She is there always, wholeheartedly, offering her helping hands.

I thank my loving parents, whose aspirations inspired me to set higher goals. They have been supporting whatever I decided to do. For them, this is a dream come true. Special thanks to my sister, I owe her a lot. My niece, nephew, grand mother, brother-in-law and other family members loved me and supported me in all my endeavors. I take this opportunity to remember my brother, who left us nine years ago. He would have been proud of his sister achieving this great feat.

I need to thank my little girl, Anita, for her unconditional love and understanding. She would have wished her mum to be with her during moments of glory at school. She is very proud of her mum.

Finally, I thank my wonderful husband for his support. He shared my sorrows, and taught me to be positive in life. He is my best friend and critic.
Abstract

Data warehouses are the primary source for a consolidated view of enterprise wide data which aids organisations in their decision making process. The data in a data warehouse is populated from the instances of operational systems and is made available to the users for their analytical needs. A number of back-end and front-end tools are used for data acquisition and analysis. Hence the data warehouse schema design is primarily driven by the functionalities of those tools as well as a model for representation of data in data warehouses. So the schema design is characterised more by the implementation models rather than by conceptual requirements. In principle, a data warehouse can be viewed as an aggregated database and hence its schema design does need to take into account the users’ needs and requirements. Generally the users requirements are expressed in terms of queries. Hence there is a need to formally define classes of queries and formulate a systematic and iterative approach to synthesising the schema from the user specifications. The design should be general and independent so that it can
be translated to logical and physical phases.

This thesis proposes a model for capturing user queries and a method of synthesising data warehouse schemas. In order to provide a query model, a taxonomy of data warehouse queries is studied and has been classified so that this synthesis is correct and complete. The proposed design allows query specification in natural language form. To translate these queries into a schema, a formal representation is required. Hence a query representation is proposed which aids the schema generation process. Formal discussion on the design is adequately supported by real life examples and a case study is provided for this purpose.

Recognising the advantage of an information model at the conceptual phase, we have introduced such a model in our design. This serves as a common platform for various data warehouse schemas with specific properties and the designer is able to derive schema based on demand.

The generality of our model is described in terms of subsuming other models. Hence mapping techniques are detailed and particular implementation issues are discussed. The proposed design follows the guidelines of traditional conceptual design where by the conceptual phase is independent of the logical and physical levels. The notion of independence is further extended towards the conceptual level, providing a method for schema refinement with respect to relevant optimization criteria for data warehouses. The design is comprehensive yet practical, in the sense that it provides query specific schema for implementation.
## Contents

1 Introduction 1

1.1 Data Warehousing: A Business Intelligent System 3

1.1.1 Architecture of Data Warehousing 5

1.2 Data Warehousing Issues 12

1.2.1 Design Issues 15

1.3 Motivation 16

1.4 Objectives and Contributions 20

1.5 Thesis Structure 21

1.6 Summary 23

2 Data Warehouse Design 24

2.1 Introduction 24

2.2 Data Warehouse Design: An Overview 26

2.2.1 Top-down and Bottom-up Designs 28
2.3 Design Considerations ............................................ 30
  2.3.1 Multidimensional Representation .......................... 31
  2.3.2 Aggregation .................................................. 32
  2.3.3 Business Measures ......................................... 36
  2.3.4 Hierarchies .................................................. 38
  2.3.5 Aggregation Along a Hierarchy ............................ 44
  2.3.6 Source Data ................................................. 46
2.4 Data Warehouse and Queries .................................... 47
  2.4.1 User-Centered Schema Design .............................. 48
  2.4.2 Query Based Approaches .................................. 49
  2.4.3 Query Types ................................................. 52
2.5 Top-down Designs ............................................... 55
  2.5.1 Cube Models ................................................. 56
  2.5.2 Extended ER Models ....................................... 60
  2.5.3 Other Approaches ......................................... 64
2.6 Bottom-Up Designs .............................................. 69
  2.6.1 ER to Dimensional Fact Model ............................. 70
  2.6.2 ER to Multidimensional Model ............................ 72
  2.6.3 ER to Star .................................................. 74
  2.6.4 ER to Multidimensional ER ................................. 77
  2.6.5 A Hybrid Approach ....................................... 79
3 User-Centric Schema Design Framework

3.1 Introduction

3.2 Formalisation of Requirements
  3.2.1 Definition: Measure Type
  3.2.2 Derived Measure Type
  3.2.3 Similarity between Measure Types
  3.2.4 Classification

3.3 The Knowledge Base Graph

3.4 Query Representation

3.5 Query Taxonomy
  3.5.1 Single Measure Type-Single Classification (Type SS)
  3.5.2 Single Measure Type-Multiple Classification (Type SM)
  3.5.3 Multiple Measure Type (Type MM)
  3.5.4 Query with only one Measure Type (Type M)
  3.5.5 Query with Specialized Class (Type S)
  3.5.6 Query with Exception (Type E)

3.6 The User-Centered Schema Design Framework

3.7 Summary
4 Translation of Queries for Schema Synthesis

4.1 Introduction ............................................. 141

4.2 Intermediate Schema Using Queries ............... 143

4.3 Mapping Query Trees .................................. 148

4.3.1 Validation and Classification of Query Trees . 152

4.3.2 Similarity Function ................................. 154

4.4 TypeSS Queries and the Mapping .................... 158

4.4.1 Mapping TypeSS1 Queries ...................... 158

4.4.2 Mapping TypeSS2 Queries ...................... 164

4.4.3 Mapping TypeSS3 Queries ...................... 167

4.5 TypeSM Queries and the Mapping .................... 168

4.5.1 Mapping TypeSM1 Queries ...................... 168

4.5.2 Mapping TypeSM2 Queries ...................... 170

4.5.3 Mapping TypeSM3 Queries ...................... 171

4.6 Mapping Other Query Types ......................... 172

4.6.1 Mapping TypeMM queries ...................... 172

4.6.2 Mapping Exception Queries .................... 173

4.6.3 Reconsidering Rejected Queries ............... 175

4.7 The Intermediate Schema ............................. 176

4.8 Summary ................................................. 181
5 Synthesis of Intermediate Schema

5.1 Introduction .................................................. 182

5.2 Mapping of Intermediate Schema to Existing DW Models ........ 184

5.2.1 Cube Design Using Intermediate schema .................. 184

5.2.2 Mapping to Star-schema ................................. 198

5.2.3 Discussion ............................................... 204

5.3 Conceptual Reduction ....................................... 205

5.3.1 The Reduction Algorithm ............................... 208

5.4 Summary ..................................................... 215

6 Conclusion and Future Work ................................. 217

6.1 Conclusion ................................................... 217

6.2 Future Research ............................................. 220

References ....................................................... 223

Appendix A Proof: Classification Lattice ........................ 243

Appendix B TypeSS_2 Mapping ................................ 252

Appendix C TypeSS_3 Mapping ................................ 255

Appendix D TypeSM_1 Mapping ................................ 257

Appendix E TypeSM_2 Mapping ................................ 259
Appendix F  TypeSM3 Mapping  262

Appendix G  Intermediate Schema : An Example  264
## List of Figures

1.1 Data warehousing architecture. ........................................... 6

2.1 A data cube: multidimensional view. ................................. 33

2.2 Requirements: different point of views. ............................. 36

2.3 Multiple hierarchies: an example. ................................. 42

3.1 Classification schema: an example ................................. 104

3.2 A grouping function .................................................. 105

3.3 An example classification lattice ..................................... 108

3.4 Representation of descriptive attributes ........................... 110

3.5 The KB Graph: an example ........................................... 116

3.6 A KB graph: from Case Study ........................................ 117

3.7 Edges in KB graph ................................................... 118

3.8 A Query tree from Case Study. ...................................... 122

3.9 A query taxonomy. ................................................... 124

3.10 The user-centered schema design framework ..................... 136
4.1 TypeSS₁ query tree ........................................... 163
4.2 TypeSS₂ query tree ........................................... 167
4.3 TypeSS₃ query tree ........................................... 168
4.4 TypeSM₁ query tree ........................................... 169
4.5 TypeSM₂ query tree ........................................... 171
4.6 TypeSM₃ query tree ........................................... 172
4.7 Intermediate schema: an example. ......................... 177

5.1 Star constellation: an example. ............................. 202
5.2 Star galaxy: an example. .................................. 203
5.3 Minimal DW schema: an example. ........................ 212

G.1 An intermediate schema from case study ................. 266
List of Algorithms

4.1 Intermediate Schema Algorithm ........................................... 148
4.2 Mapping $TypeSS_1$ Queries ............................................. 162
5.1 Dimension Selection Algorithm ......................................... 196
5.2 Reduction Algorithm ...................................................... 211
Chapter 1

Introduction

Businesses around the world are undergoing a transformation caused mainly by globalisation and economic changes. They are forced to change their conventional business processes to become more agile and efficient. A new breed of managers called *multidimensional managers* ([RNR99]) have emerged to tackle the current challenges. These multidimensional managers operate in a business world that is complex and possess a multidimensional view of their business. They are changing the traditional way of hierarchical management to a horizontal network where the interests of managers span the whole business itself.

Most organisations will have multiple business applications that improve the efficiency of managerial and operational processes of various business units. These business applications create enormous amount of data which in most cases are purged out after a certain period. The strategic value of the data is being recog-
nised and efforts are focused towards extracting the value to gain competitive advantage. A new generation of systems called *business intelligent (BI) systems* or *analytical systems* have come into existence as a result of this. Organisations benefit from improved customer service, effective marketing programs, accurate demand forecasting and an overall cost savings due to operational efficiencies bestowed from BI systems.

BI systems are primarily focussed on processing the existing enterprise data to provide high-quality information to people throughout the organisation. In other words, BI systems gather, manage and analyse data to provide insight into trends and opportunities. It is assumed that any knowledge worker is able to access information from these systems.

In this thesis we analyse one such BI system called *Data Warehousing*; which is also used for decision making processes. This is an environment rather than a product [K.J98], in which various components are involved and in the following sections we will describe data warehousing and its functionalities in detail.

This chapter is organised as follows. In Section 1.1, the data warehousing environment is introduced. In subsequent subsections, different stages associated with this system are explained. Various issues associated with the system development are discussed in Section 1.2. From these addressed issues, one particular aspect; data warehouse design, which is the primary focus of the thesis, is considered in detail. Thesis motivations and the contributions are explained in Section
1.3 and Section 1.4 respectively. The thesis organisation is explained in Section 1.5 and the chapter is concluded in Section 1.6

1.1 Data Warehousing: A Business Intelligent System

“Data warehousing is defined as a process for assembling and managing data from various sources for the purpose of gaining a single, detailed view of a part or all of a business” [S.R98]. A centralised repository called data warehouse is used to provide the integrated view of data. It can also be described as a dedicated database which provides relevant information to decision makers.

The demand for this technology is stronger ever since the concept emerged in the early nineties. According to the Gartner group survey in 2004, ([Rel06]), data warehousing came into the top ten priorities among CEOs. Organisations are investing in data warehousing as they consider data as their asset.

Case studies from different business areas illustrate success stories of data warehousing. An American retail company [LC04], implemented a data warehouse in 1995 and captured many customer interactions. This enabled the company to gain customer intimacy and thereby segmenting their customer portfolio.
In addition, the data warehouse allowed the company to improve their customer relationship management as well as marketing opportunities.

*Mercy Care Plan* ([Cor04]) is the second largest health care plan in Arizona. By implementing a data warehouse they improved their customer analysis, controlled their expenses and thus made an over all improvement in the business. In the case of *First American Corporation* [BHBD00], strategies were shifted from traditional banking to a more customer oriented approach with the help of a data warehouse and this implementation helped them to emerge as a leader in financial industry.

In the past, it has been very difficult to get an integrated view of activities that occurred in an enterprise. Previously, the information has been gained from various source systems or production systems called *On-line Transaction Processing (OLTP)* systems, only after someone requested for information. This is a time consuming process and termed as a *lazy* approach [J.W95]. Data warehousing provided a solution to this problem in which data from different sources is integrated into a single, centralized repository ready for querying. Compared to the lazy approach mentioned earlier this one is an *eager* approach because data is already there to answer queries.

Different types of tools are necessary in order to integrate data from various sources as well as for the presentation of data in a warehousing environment. These tools can be classified as *back-end* and *front-end* tools. Back-end tools are
used to integrate data from heterogenous sources whereas front-end tools help to access data from data warehouses. A more detailed discussion of tools associated with data warehousing can be found in the following section. It also discusses different stages associated with data warehousing in detail.

1.1.1 Architecture of Data Warehousing

The figure 1.1, divides data warehousing into three stages which can be described as follows.

- **Data acquisition** stage: During this stage data is extracted from sources, transformed and loaded into a database using various tools.

- **Data storage**: At this stage, data is stored in a database.

- **Data analysis**: This is the final stage in which data is accessed through analysis tools.

Each stage is explained in detail in the following sections.

**Data Acquisition**

A data warehouse contains data from multiple operational systems and external sources. Since this data is used for decision making, it is important that the data should be correct and consistent. Inconsistent field lengths and inconsistent de-
scriptions are some of the common data errors. A common example of an inconsistent description is the gender specification in various systems. It can be male and female in certain systems or M and F in some other systems. But in a data warehouse every piece of data can have only one meaning and specification. A variety of tools are utilised for data cleaning, transforming and loading in order to provide correct data in the data warehouse. These tools are collectively known as Extract, Transform and Load (ETL) tools.

After extraction, cleaned and transformed data is loaded into a data warehouse. Before data loading, an additional requirement such as summarisation is required. Summarisation refers to processing of raw input data or detail data for more compact storage in a form useful for analysis in a particular application. This involves selection, filtration, reorganization and manipulation etc, of atomic data to produce predetermined totals [S.R98].
A data warehouse stores historical data hence the volume of data is high. Because of this summarization is necessary. This also improves the responsiveness of a data warehouse by improving the query response time. Two main factors involved with summarisation are; the selection of data to summarise and the unit of time for summarization. These factors are decided by a designer based on the nature of the queries that need to be answered.

As we mentioned earlier, data from a data warehouse comes from operational systems. This means the changes happening at this level need to be propagated to the data warehouse after the data warehouse is populated. Sending updates to a data warehouse is called *refreshing* and the tools used for this purpose are called refresh tools. The refreshing of a data warehouse is an important process which determines the effective usability of data collected and aggregated from the sources [BFMB99]. The refresh frequency is normally set by an administrator based on user needs and source systems.

**Data Storage**

As mentioned in Section 1.1, a data warehouse is a dedicated database used for decision making processes. The characteristics of a data warehouse are considered here in detail. A well cited reference for data warehouses comes from [W.H92]. Inmon defined data warehouse (DW) as a subject oriented, integrated, time variant and non-volatile collection of data. The definition is further detailed as:
• **Subject orientation**: In a data warehouse, data is organised in subject areas like finance, sales etc across the enterprise whereas operational systems are typically designed in the context of an enterprise’s applications such as payroll, order entry, etc.

• **Integration**: Data from various heterogenous sources are integrated in a data warehouse. The sources of a data warehouse normally are the underlying operational systems, external data sources etc. Data in a data warehouse is homogenized and cleaned which allow querying from a common repository. Data warehouse data varies in **granularity**. That is, data in a data warehouse ranges from the most detailed level to a highly summarized level. Raw data from the operational systems is considered to be the most detailed level.

• **Time-invariant**: Generally data in a data warehouse are stored as snapshots and not on current status. This historical data can then be analysed in order to ascertain business trends.

• **Non-volatility**: Data in data warehouse is read only; users are not allowed to change the data. Modifications in data warehouse data takes place only when the modifications of the source data are loaded into the data warehouse.

Even though data warehouse is derived from operational data, it is different from the operational databases. The design of a data warehouse is more chal-
lenging than that of operational databases because of the nature of the system and the volume of data involved. Also the functionalities of these two databases are different. Maximizing transaction through-put and minimizing conflicts are the main criteria for operational databases whereas data warehouses aim to support a smaller number of users in their decision making. Since data warehouses are used for analysis purposes they are also known as analytical databases.

The following Table 1.1 shows the main differences between a data warehouse and an operational database.

<table>
<thead>
<tr>
<th><strong>Operational database</strong></th>
<th><strong>Data warehouse</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction driven</td>
<td>Analysis driven</td>
</tr>
<tr>
<td>Support day-to-day operation</td>
<td>Support decision making</td>
</tr>
<tr>
<td>Constant updates</td>
<td>Updates are rare</td>
</tr>
<tr>
<td>Detailed data</td>
<td>Summarized data</td>
</tr>
<tr>
<td>Current data</td>
<td>Historical data</td>
</tr>
<tr>
<td>Deals with few records</td>
<td>Deals with millions of records</td>
</tr>
<tr>
<td>Large number of users</td>
<td>Smaller number of users</td>
</tr>
</tbody>
</table>

Table 1.1: Operational database vs Data warehouse

**Data Analysis**

Data analysis refers to access data from data warehouses for decision making processes. Two major application areas of a data warehouse are Online Analytical Processing (OLAP) and Data Mining.
OLAP: In 1993 E.F. Codd introduced the term OLAP for intuitive, interactive and multidimensional analysis of data [ESC93]. That is, OLAP is characterized by the multidimensional representation (presenting data as a multidimensional space) and analysis of consolidated enterprise data which supports end users analytical and navigational needs. As pointed out in [BCGM98], during the last decade there was a rapid growth in this area. OLAP has become very popular primarily because of the features it offers to the users. The features are: visualization, navigation and sophisticated analysis [N.G03]. Visualization allows the user to interact with the system in a user friendly manner. Navigation refers to operations by which users are able to view data with different granularity. Sophisticated analysis allows users to perform different types of analysis such as statistical profiling, moving averages, exception conditions etc.

OLAP tools are mainly classified in terms of the underlying data warehouse implementation. They are called Relational OLAP and Multi dimensional OLAP.

- ROLAP: Data warehouses might be implemented on standard or extended relational databases called Relational OLAP or ROLAP ([SU97]). One major issue with this implementation is expressing the operations. OLAP operations can not map efficiently into relational operations. In other words, using standard SQL it is difficult to express an OLAP query. Many commercial products extended SQL to incorporate OLAP operations. However
this drastically reduces system performance [LJ00]. Even with this limitation, the relational implementation is considered as good for very large data warehouses.

MicroStrategy’s DSS server ([Mic98]) and Informix’s Meta Cube ([IBM98]) are examples of commercially available ROLAP.

- **MOLAP**: If data warehouses use a specific data management system in terms of array structure it is called *Multi dimensional OLAP* or MOLAP. Since arrays are used for data storage, empty cells are possible resulting in sparse data. Storing sparse data is inefficient and additional compression techniques are necessary to overcome this limitation [BCGM98]. Nonetheless MOLAP efficiently supports OLAP operations and is suitable for data warehouses where the amount of data is considerably low.

Hyperion Essbase from Hyperion ([Hyp01]) is an example of a commercially available MOLAP product.

**Data Mining**: The OLAP application, that we have discussed previously, deals with users on-line analysis requirements. In this case users’ involvement is mandatory. When dealing with large amounts of data, this manual analysis becomes slow and expensive. Businesses consider data as their asset and it is used to increase their efficiency and success. Hence computational techniques are necessary to get valuable information from the data. Data mining is one such
technique which is used to identify patterns from mass volume of data. Data mining is a generic term used for the application of specific algorithms for extracting patterns from data. This is considered as a step in Knowledge Discovery in Databases (KDD). KDD is defined as the nontrivial process of identifying valid, novel, potentially useful, ultimately understandable patterns in data [UGSP96a]. Commonly used data mining algorithms for this purpose are: classification, regression and clustering [UGSP96b]. Future trends and interesting patterns of a business can be extracted from a data warehouse using these mining techniques.

In this section we have studied data warehousing, in particular the stages associated with the system. Based on this we will discuss various research areas associated with data warehouse system development in the next section.

1.2 Data Warehousing Issues

In the previous section we have seen different stages of data warehousing in terms of data acquisition, storage and analysis. Each of these stages present various challenges. In order to tackle this, different streams of research areas have been evolved. They are mainly focussed on issues related to design, implementation, operation and maintenance. The main goal of all these areas is to provide necessary information to the user correctly and quickly.
Based on the general architecture discussed in Section 1.1.1, different research problems are illustrated here. The main issue associated with the first stage can be seen as data integration and loading [J.W95], [J.S99]. Data integration is a difficult process because of the inconsistencies in source databases. Since different types of data sources are involved, schema integration is the major issue. Tools development related to data integration and translation is also challenging.

Query processing and optimization has been of keen interest to researchers [SU97]. Since a data warehouse is primarily used as a source for querying, queries are important in this environment. Compared to the update and deletion queries in traditional databases, data warehouse queries are complex. Queries are complex in the sense that they consist of common functions such as \textit{sum}, \textit{count}, \textit{minimum}, \textit{maximum etc} as well as complex statistical functions like \textit{median} or moving average \textit{etc}. Moreover, as the size of the database is in gigabyte to terabyte range, the \textit{join} operation becomes a more costly procedure than the update and deletion queries. The well known query language, SQL does not allow the concise expression of queries involving aggregation and group-by [CR96]. In order to express such queries, extension of SQL is required. Because of these reasons query processing and optimization need more sophisticated techniques and approaches like [AVD95], [JWWL00], [SS96] and [DA01] contributed towards this area.

With respect to maintenance and operation of a data warehouse, view maintenance is one area to consider [DASY97], [RAX00] [DM00] and [RC03]. As we
mentioned earlier, a data warehouse stores integrated information from multiple
data sources. The integrated information is normally in a pre-computed form in
order to improve the query response time. This pre-computation is called *mathe-
ricalising views* over source data. Hence in an abstract level, data in a data ware-
house can be seen as a set of materialised views. The challenges associated with
this technique are:

- *selection of views to materialise*: A materialised view is normally treated
  as a pre-computed query or a partial query. That means view selection de-
  pends on those queries which need to be answered. While the view selec-
  tion, generally minimum query evaluation cost will be the main criterion
  and storage space could be another consideration. Pre-computing all the
  expected queries is ideal but space constraint has to be considered. The
  challenge here is to select views with minimum query evaluation cost for
  given system constraints.

- *efficient use of the materialised views*: One view will be able to answer
  different queries. In that respect techniques are necessary for the efficient
  use of the views.

- *efficient update of materialised views according to the changes in source
  data*: The main issue associated with the view technology is that views
  need to be updated according to the source.
If the frequency of updates are high the maintenance cost will increase which is not desirable.

Various approaches studied the above mentioned issues and a few examples are [CMJ01], [RHD02], [HM99], [SKSK95] and [DASY97].

Since data warehouse is considered as the main component of the system, its design is very important. This is considered in detail in the next section.

1.2.1 Design Issues

The design of data warehouse itself is challenging in the second stage. This includes all the phases of design, from conceptual to physical implementation. Even though data warehouse design is similar to operational databases, the techniques used in the design of these databases are not suitable for a data warehouse. This is mainly due to the differences in functionalities that we have summarised earlier in Section 1.1.1. Different from OLTP databases, a data warehouse needs to support OLAP applications which include navigation and complex analysis.

In the past few years research in this area has flourished. Approaches such as [WWAL02] and [SMKK98] identified issues associated with conceptual modeling in terms of representation and summarisation. However we could find more focussed designs in [TC99], [AKS01] and [CMGD98] where the interest is the conceptualisation itself. Another set of works; for example [D.l00], [MDS98b],
known as methodologies, studied the transformation of an ER model to a data warehouse model. These designs assume the existence of an ER model and guidelines are provided to identify the data warehouse constructs from this.

Issues related to implementation such as efficient data structures and new indexing techniques are the focus of physical design. Here we avoid a detailed discussion on physical design to concentrate on the conceptual side. However, readers could find studies specific to physical aspects in [VAJ96], [MDA01], [LJY03], [MZ00], [HVAJ97], and [WDB97].

1.3 Motivation

We have addressed various issues associated with data warehousing in the previous section. Here we further investigate issues related to conceptual design which form the basis of this thesis.

At the beginning research in conceptual design was focussed on operators/algebra. Approaches such as [RGS97], [LW96] [PV98], [AH97] and [ML97] are example of this. In [AJS01] these models are called formalisms due to the lack of semantic richness.

More semantically rich models; [CMGD98], [JMJS01], came later and these approaches extended the existing ER or object-oriented concepts in order to capture data warehouse requirements. A common feature of all these models is the
multidimensional data representation (discussed in detail in Chapter 2), which is the natural way the user perceives the problem at hand.

The current models addressed a set of analysis requirements related to navigation and data summarization. These approaches provide their own formalisations and graphical notations which means there is no standardisation in this regard.

Also, these designs terminate with the conceptualisation of analysis requirements. A design process is not complete without schema derivation and evaluation. As pointed out in [HLV00], the existing data models were developed without an associated design process and thus without guidelines on how to use them or what to do with the resulting schema.

When comparing data warehouse design with traditional database design, the transition of a model to a schema is not seamless. In traditional database design there are pre-defined objectives associated with each stage of a design. For example, completeness with respect to application domain, minimality of the schema, freedom of redundancy etc are the objectives associated with conceptual design [HLV00]. The dependency theory ([C.J00]), further explains the need for redundancies and normal forms are suggested to eliminate them. A similar systematic approach could not be found in data warehouse design. Normal forms are studied in the context of data warehouse data models in [WJH98]. However this approach applied normal forms only to avoid specific issues related to data summarisation. Functional dependencies are applied in [HLV00] to derive a data warehouse
schema from an ER model. Taking all these aspects into account, the motivations of this thesis are as follows.

- Even though the existing designs such as [TC99], [AKS01] and [Leh98], addressed the issues related to visualization and navigation along the data, a formal foundation is still missing. A model that can act as a basis for generating schema with particular characteristics is lacking. This is noted in [A.A02] that “it is difficult to identify a standard data model, because there is neither a model encompassing the semantic constructs of the rest, nor a consensus or standard for what should be represented in a schema ”. To this extent a generalisation is necessary.

- Another set of designs called methodologies or bottom-up designs could be found in this area. These methodologies ([MDS98a], [LR98], [D.l00]) provide design guidelines to construct a data model from an existing global ER model of the operational systems. In these bottom-up design approaches the modeling constructs are pre-defined and they are identified from an ER model. The main drawback of these methodologies is that, a data warehouse is derived based only on the operational systems. Data warehouses need external data like web data and market survey data. External data representation is not taken into consideration in current methodologies. Discussion on user requirements and a method to translate those requirements to a schema
are also lacking here. So, to summarise, bottom-up designs are data driven and a user oriented approach is necessary.

- Complex query support is the main function of a data warehouse. The analysis requirements presented in existing models originate from the queries but real queries are given very little importance during the conceptualisation phase. As per current philosophy ([CMGD98], [MDS98a]), a data model captures certain analysis requirements which are only a higher level abstraction of real user requests. The user then formulates queries based on this model and the query support is left out as a post design problem. What we argue here, is that user queries should be taken into consideration while designing the schema even though not all queries are known at the early stage. By doing so the schema becomes more user oriented and is easier to support those types of queries that are considered during the design. This approach also allows easy translation of the schema to logical and physical levels.

In order to provide such a technique, data warehouse queries need to be studied from a conceptual perspective. Currently the query discussions presented are mainly related to optimization and processing. So far, a study on queries at the conceptual level and impact of different types of data warehouse queries in schema
1.4 Objectives and Contributions

This thesis addresses the previously discussed problems, by providing a modelling framework which supports a conceptual schema design for a data warehouse. The framework is general enough to support a wide range of business applications. The concepts and the techniques are presented formally in this framework. This formalisation will map user requirements into a conceptual schema.

A discussion on schema properties and the derivation of a data house schema can be seen as an optimisation approach at the conceptual level. But the optimisation part does not deviate from conceptual end to physical end. We emphasize here that the term optimisation refers only to conceptual restructuring and all the optimisation properties are discussed purely at the conceptual level. Any assumptions or considerations of physical properties cannot be seen in this approach. Based on the above discussion, the main contributions of this thesis are as follows.

- A modeling framework is proposed based on requirements that support a data warehouse conceptual schema design. This proposed design uses queries for schema synthesis, so a query efficient schema can be generated. In addition to this the design fulfills generality and independence properties.
• Data warehouse queries are studied from a conceptual perspective. A taxonomy and a conceptual representation for queries are developed based on this study and are used in the design for schema derivation.

• We propose a generalisation of existing data warehouse data models. While the existing models use specific modelling techniques such as ER and Object-oriented, we choose a graph theory approach for the formalisation. This serves the purpose of generality of the data structure and can be fitted into the proposed framework. Graph reduction techniques are applied to produce the final data warehouse schema.

1.5 Thesis Structure

In order to investigate more about data warehouse design, in Chapter 2, the existing designs are reviewed. Since our focus is on requirement oriented schemas, current models have been approached from a requirements point of view. Another highlight of this chapter is a discussion on data warehouse queries. This study leads to a formal query taxonomy and a conceptual representation.

Chapter 3 begins with formal definitions and formalisation of concepts that are necessary for our design. For the formalisation, a graph theory approach is selected to keep the framework as general as possible. This formalisation is termed a knowledge base, which is treated as the basic knowledge required for the schema.
Other than this, a conceptual perspective on queries is presented which consists of a taxonomy and a conceptual representation.

Chapter 4 describes a schema generation method using queries. An algorithm is developed for this purpose which maps different types of queries in the knowledge base. Mapping methods associated with each query type are described in detail. A special type of function called similarity function is suggested to aid the mapping process, the main functionality being to test the similarity that exists between the nodes in the graph.

Chapter 5 is presented as two parts. The first part discusses the mapping of intermediate schema to the existing models. This mapping shows the completeness and generality of the proposed framework. Algorithms are developed to map the graph based schema to the cube and star models. In the mapping, various implementation issues are also considered.

The second part of this chapter discusses schema refinement. This is proposed as a conceptual reduction where a graph reduction method is suggested to derive a data warehouse schema. The proposed reduction algorithm finds the shortest path for a query which is will be the schema property.

A retail case study is adopted to describe the design and is used as a running example of this thesis.

Chapter 6 concludes the thesis and future works are detailed.
1.6 Summary

Data Warehousing has been introduced as a business intelligent system in this chapter. The architecture of the system was detailed as a three step process and each step studied separately. Various issues associated with data warehouse schema design were considered. Specific problems existing in conceptual design were identified and presented as motivations. In particular, the lack of a general foundation for schema derivation as well as the necessity of a query driven approach was illustrated.
Chapter 2

Data Warehouse Design

2.1 Introduction

Database is defined as a collection of related data [RS00]. It plays an important role in all areas of our day-to-day life such as education, health, business etc. As per the database systems time-line given in [RS00], databases have a very long history which goes back to 1906. In more recent years, we have seen the evolution of different types of databases such as statistical databases, spatio-temporal databases, multimedia databases etc. A data warehouse is one of these and is specifically used for analysis purposes. Hence it comes under the category of analytical databases.
Even though a data warehouse is a different type of database, its design can relate to traditional databases [C.J00]. So here, before considering data warehouse design in detail, a brief discussion on traditional database design is presented.

Formally, database design distinguishes a database and its description. The description of a database is called a database schema. Generally a schema is described on the basis of a paradigm known as a data model, which is a set of concepts used to describe a database structure [RS00].

A data model may describe a database at different levels. This reflects the level of abstraction by the model. Depending upon the abstraction, data models can be categorized as high-level or conceptual model, logical or implementation model and physical model. The corresponding designs are termed as conceptual, logical and physical designs respectively.

During the conceptual design of a database a conceptual schema is produced, which is a concise description of user’s requirements. The schema does not include implementation details and for this reason it is easier to understand and is used to communicate with the non-technical users. A conceptual schema can also be used as a reference for user requirements.

A slightly different approach in conceptual design could be found in [Jr79]; where a two level approach is suggested. That is, two types of models are proposed called the conceptual information model and the conceptual database model. The conceptual information model is a reference model developed from the in-
formation requirements. The conceptual database model is a derivative of the conceptual information model and is the database model. In other words, from the reference conceptual information model, different database models can be derived.

The conceptual schema is translated into a logical schema during the second stage. Logical design produces a logical schema that can be processed by a database management system software. The physical design phase describes the storage structures and access methods in order to implement the database in secondary memory.

Unlike traditional databases, data warehouses flourished only in the last decade. Even a design methodology similar to the traditional database ([BCN92]) was lacking until recently. Researchers made their attempt to solve this issue and design methodologies such as [O.H00] and [HLV00] came as a result of this. These approaches suggested design methodologies for data warehouses in line with traditional approach which we will see in the following section. This also covers all relevant discussions related to data warehouse design.

2.2 Data Warehouse Design: An Overview

As mentioned previously, data warehouse design follows a methodology as that of a traditional database. Based on those methodologies, the design starts with
requirements specification followed by conceptual and logical design; then finally the physical design. Each of the existing methodologies are considered here in detail.

A three level modeling framework proposed in [O.H00] starts with the conceptual level where a modeling language called MML (Multidimensional Modeling Language) is introduced. MML is an object-oriented, implementation independent language which is specified by UML diagrams. The UML diagrams are transformed to a relational schema in the logical design phase. At the final stage, physical level, the relational schema is translated to a working database schema by means of an algorithm.

Different from the above approach, a process model proposed in [HLV00] included a requirement specification step in the design. At this stage, data from operational schema are selected in discussion with data warehouse users and business experts. Then a conceptual design is suggested based on functional dependencies.

Research in data warehouse data modeling was started even before the existence of the above discussed design methods. A few examples to this are [R.K96], [LW96], [LR98]. These approaches addressed various modeling issues and proposed models for data representation. At the beginning, the focus was more on the representation and performance. Hence, conceptual design was left out or conceptual and logical designs were coincided [O.H00], [MDS98a]. Even today the significance of conceptual design for data warehouses is high. A recent keynote
address ([S.R03]) in one of the major data warehousing conferences stressed the significance of conceptual design in this area. So, in this thesis we restrict the scope only to conceptual design and issues relating to logical and physical levels are left out from further consideration but could be found in [MZ00], [WDB97], [DKT02].

### 2.2.1 Top-down and Bottom-up Designs

Two main design principles in data warehouse design are: top-down and bottom-up approaches. In top-down designs, for example, [CMGD98], [MDS98b], and [AKS01], requirements related to data representation and summarisation are conceptualised and presented as a new data model for data warehouses. In this case, the modeling process terminates with the definition of the model. On the other hand, bottom-up designs translate an operational ER model, which is the source, to a data warehouse model (target) through algorithms. Approaches like [D.100], [MDS98a] and [LR98] are examples to bottom-up design. Normally, bottom-up designs do not discuss requirements explicitly. It only provides transformation from one model to another.

The above mentioned design techniques mainly deal with conceptualisation and transformation. So these designs can also be described as *conceptualisation* and *transformation* respectively. A data warehouse design which integrates both
the top-down and bottom-up approaches could be found in [AFS+01]. In this case, the top-down design phase insists on a user oriented schema design while the bottom-up design is similar to any other transformation methods. At the end, an integration of top-down and bottom-up design is suggested. We will discuss the existing top-down and bottom-up designs in Section 2.5 and Section 2.6 respectively.

As we stated earlier, a conceptual schema is an abstraction of user requirements which means the design should start from requirements. To that extent in this chapter, requirements are considered in detail. In other words, requirements are given utmost priority in our approach considering their relevance in schema design. On the other hand, existing designs are also discussed with respect to traditional conceptual design goals. Based on the above discussions the main objectives of this chapter are:

- identify a set of requirements that are important in data warehouse design;
- discuss existing approaches from a requirements point-of-view. That is, address the requirements considered and describe the formalisation of requirements in each model and
- compare current data warehouse designs against traditional conceptual design.

The following sections of this chapter are organised as follows. The business
requirements that are important for data warehouse design are discussed in Section 2.3. In Section 2.4, queries have been studied with a view to incorporate them in the design. Different top-down and bottom-up designs are thoroughly studied in Section 2.5 and Section 2.6. A comparison of existing data warehouse design has been made against traditional data base design methodology in Section 2.7. Finally, in Section 2.8 the chapter is summarised.

### 2.3 Design Considerations

The existing top-down designs, address the requirements from the database point of view. That is, considering a data warehouse as a database for analytical purposes, requirements are stated in terms of data representation and summarisation which are specifically known as *multidimensional representation and aggregation* [TC99]. But in conceptual modeling the users point of view as well as business perspective plays an important role because a conceptual model is considered as an abstraction of these views. In that respect, requirements have to be talked in terms of users and business. A thorough discussion of these matters is presented in this section.
2.3.1 Multidimensional Representation

Multidimensionality is defined as conceiving the subject of analysis as a multidimensional space ([AJS01]). The primitive concepts used in a data warehouse for constructing a multidimensional space are facts and dimensions. A fact is defined as an item or subject of interest for an enterprise and which is the subject of analysis in a multidimensional space. Fact is described through a set of attributes (typically numerical but not necessarily) called measures [MDS98b].

Dimensions represent the different points of view of data for analysis or it can be defined as the context for analysing the facts. Facts and dimensions are also known as quantifying and qualifying data respectively.

Multidimensionality is normally visualized through the cube metaphor [SU97]. Dimensions act as the coordinates of the cube and different combinations of dimensions define a cell in a cube. Each cell represents a unit of data which is a measure value. Since in practice there can be more than three dimensions, the cube is called a Hypercube. These multidimensional concepts and terminologies are explained through the following case study.

Case Study

An application that heavily uses data warehousing is retail business. So in this case study we consider a hypothetical company which has outlets through out
a nation. Each outlet store records every sales transaction with respect to each customer. But at the top level, management is not interested in individual customer transactions. They would like to receive data in a more summarised form, for instance, on a daily or weekly basis and it is hoped that the data warehouse can provide the summarised data. More specifically, the management is interested in looking at their sales figures and the purchases made in order to study their business growth. So sales and purchases are the subject of interest and hence these are the facts in this application. Sales can have attributes like sales amount and number of items sold etc, which are the measures. Also, sales make sense only in the context such as when it has happened, where it has happened, what is being sold etc. Then the context for sales are time, store and product which are the dimensions. Based on sales and its dimensions, a data cube is shown in figure 2.1. The cell values in the cube represent values for the measures.

2.3.2 Aggregation

While presenting the processes associated with data warehousing in Chapter 1 we have mentioned summarisation. There we stated that data warehouse users are normally decision makers and they are only interested in high level data rather than the most detailed data such as a single customer transaction. Such queries are considered to be a summarisation of detailed data. Summarisation is also known
as aggregation.

Aggregation is defined as computation of a function or combination of functions on one or more measures. Aggregation functions are a class of generic functions which must be usable in any database application [HJA01]. Aggregation functions are not limited to common functions such as COUNT, MIN, MAX, SUM, AVG in a data warehouse but can also include other functions like ranking, percentile, comparison, as well as complex statistical functions such as median, moving average etc.
The aggregation functionalities can be achieved by performing well-defined operations. The common operators that are proposed in data warehouse are:

- **Roll-Up / Drill-Down**: Roll-Up is an operation whereby the user navigates among the levels of data ranging from the most detailed to the most summarised in one or more dimensions. Drill-down is a similar type of operation by which a user navigates from the most summarised to the most detailed level.

- **Slice**: Slicing is the operation used to select dimension for viewing data. This operation is equivalent to the selection operation in relational algebra.

- **Dice**: Dice is the operation that specifies the range from the selected dimension.

- **Pivot**: The operation pivot switches one selected dimension over to the other. For example, in the cube shown in figure 2.1, there are three dimensions. Suppose a user is looking at data with respect to time and product, then using the operator pivot, he/she can view the data with respect to product and store or store and time.

As we mentioned earlier in this section, multidimensional representation and aggregation could only be seen as a macro level consideration. In fact there are other micro level requirements, which directly come from users and business peo-
ple. Specifically these requirements dictate the content and scope of multidimensionality and aggregation which in turn define the contents of the schema. The micro level requirements are as follows.

- **Business structure**: Management structure their business in a hierarchical form for smooth operation. These business structures are accomplished by classifying the organisation, products and/ or services etc. Classification is defined as a grouping of objects based on certain criteria [C.S01]. The rules or criteria that are used for the grouping are called business rules.

- **Business measures**: A company’s growth depends on its strategies. The strategic success can be measured through a set of standard gauges called *business measures*. Every business has its own business measures to measure the success.

- **User queries**: User queries are the requests from users which specify the information demands of various data warehouse users. The ultimate goal of any data warehouse is to support user requests in an efficient manner.

- **Source data**: We have seen in the previous chapter that a data warehouse is derived from different operational systems and other external data sources. In that respect, the underlying data structures which are the sources of a data warehouse are helpful in its design.
As shown in figure 2.2, user queries, business structures, business measures and source data, define multidimensional representation/ aggregation. In other words, the later one can be seen as an abstraction of the former ones and this is explained in the following sections.

2.3.3 Business Measures

We have defined measures as attributes of facts in Section 2.3.1. However, the concept of measures came as a result of another type of measure which is important in business and these business related measures are called business measures.
The performance of a business is analysed based on many factors such as profitability, organisational efficiency, customer perspective etc. In earlier days the main focus was on improving end of year figures and the business strategies were underpinned and influenced by financial measures. But financial measures are in adequate to formulate business strategies in a competitive environment [RD96]. Managers realised that the growth is not sustainable with those strategies based on financial measures. They were forced to look for other indicators of business, which allow them to infer how the organisation is going to work in the future. These indicators or gauges are called business measures. Formally, business measures are defined as quantifiable standards used to evaluate performance against expected results [P.R02]. That is, the strategic success of an organisation is measured through a set of standards. These standards determine whether the objectives are met and the strategies are implemented successfully. Examples of business measures from retail are annual sales, customers lost, gross margin, total assets, etc.

At the time of requirement specification, business people specify their business measures and that gives an indication of the subjects of analysis. This means business measures define the multidimensional space, and, thus fact. Also, there are business measures that are derived from other business measures which are called derived business measures and all other business measures are basic. Specific rules are necessary in order to derive a business measure from other business
measures. These rules are specified by the business people and are known as business rules. The business rules suggest which business measures are related and how a business measure can be derived.

In the literature the discussion on derived measure is presented in a slightly different manner. That is, if a measure is derived from other measures of a fact then it is called a derived measure.

An example of a derived measure from our case study is explained here. Amount sold and Quantity are described as two measures of sales. Another measure called Total amount can be derived using Amount sold and Quantity such that

\[
Total\ amount = Amount\ sold \times Quantity
\]

Then Total amount is considered as another attribute of sales and is a derived measure.

2.3.4 Hierarchies

Hierarchies are originated from business structures and are used for analysing a measure at different levels of details [MA01]. That is, aggregation functionality is provided by means of hierarchies. In that respect aggregation depends on the definition of a hierarchy.

Hierarchy in data warehousing is defined as an ordered structure. Specifically, it defines the structure of a dimension and the most general type is a direct acyclic
Other structures such as tree and lattice could also be found in this regard [JLS99], [TC99]. The levels in a hierarchy are called dimension levels or aggregation levels and each level represents a level of detail of data (granularity) required by some desired analysis [TC01]. For any two consecutive levels in a hierarchy, the higher level is called parent and the lower level is called a child. Each level in a hierarchy is composed of different members called *dimensional members*. A dimensional member has a set of attributes and those attributes are used for classification as well as describing the properties of that member. For example, in product dimension, *product* is one level and the level is composed of various individual products. Each individual product has a set of attributes such as *productID*, *colour*, *weight*, *size* etc. In this case the attribute *productID* can be used for classification; that is, to create a higher level in the hierarchy. In general, one or more attributes can be used for this purpose. The remaining attributes such as *colour*, *weight*, *size* etc describe the properties of each product.

Members at different dimension levels are connected by means of mappings. If each member of a lower level is mapped to only one member of a higher level and each member of a higher level is mapped to at least one member of a lower level, then it is called *full mapping*. Otherwise the mapping is *partial* [EM99]. The mapping depends on the criteria used for classification as well as different practical considerations in classification. In general terms, full or partial mapping depends on either the criterion for classification, practical considerations or both.
The structure of a hierarchy depends on the mapping between the levels which is discussed in terms of *strict hierarchy*, *multiple hierarchy* and *unbalanced hierarchy*.

**Strict and non-strict hierarchies**

We have defined full and partial mapping in the previous section. Depending on the mapping between the levels hierarchies are further described as *strict* or *non-strict hierarchies*. If the mapping between any two levels in a hierarchy is full, then it is called a *strict hierarchy* or *balanced hierarchy*. Otherwise the hierarchy is *non-strict*. It is also useful to describe the strictness of a hierarchy in terms of *cardinality*. The cardinality defines the minimum and maximum numbers of members in one level that can be related to a member of another level [MZ04]. That is, in strict hierarchy the mapping between any two levels is *one-to-many* where as in non-strict hierarchy the mapping between the levels can be *many-to-many*. The one-to-many or many-to-many conditions arise due to the criterion used for classification or practical considerations.

For example, let *QC5678* be a mobile phone which has functionalities of a PDA and MP3 player and is a member of a level called *product*. Then *QC5678* can be related to any of the members *mobile phone*, *PDA*, *MP3 player* in another level, *product family*. If so, the mapping between the levels *product* and *product family* becomes many-to-many, hence hierarchy becomes non-strict. If *QC5678* is only
related to either one of the three members of the other level, then the mapping is one-to-many and hierarchy will be strict. Practical considerations allows \textit{many-to-many} mapping. Nonetheless the business people could decide which family the particular product belongs to and hence allow the mapping to be \textit{one-to-many}.

\textbf{Multiple hierarchies}

In an ordered structure, if there is more than one path and those paths share common leaf levels, those paths are called multiple hierarchies [TC01]. For example, \textit{stores} in the case study can be classified according to the geographical location. Then a hierarchy can be constructed as:

\textit{store} \rightarrow \textit{city} \rightarrow \textit{state} \rightarrow \textit{region} \rightarrow \textit{country}

In this hierarchy \textit{store} is the lowest level which is classified as the \textit{city} where each store belongs to and \textit{state} and so on. However in real situations, some \textit{cities} may not belong to any \textit{state}, it might be related to \textit{Provinces}. Then another hierarchy of store can be written as:

\textit{store} \rightarrow \textit{city} \rightarrow \textit{province} \rightarrow \textit{region} \rightarrow \textit{country}

Since the hierarchies share common levels like \textit{store} and \textit{city}, these two hierarchies can be represented as in figure 2.3.

In this figure each path is a hierarchy and they share common levels. Other than geographic location, organisational structure can be used as another criterion for classifying stores. In this case, \textit{stores} are grouped by mangers, departments
and so on. This classification creates a different hierarchy for stores. Generally, one or more criteria can be used for classification and in both cases multiple hierarchies are possible.

**Homogeneous and Heterogenous Hierarchy**

In a *homogeneous hierarchy* for every pair of adjacent levels $l_1$ and $l_2$, level $l_1$ is related to level $l_2$ and the mapping between the levels is a full mapping. If a hierarchy is not homogeneous then it is *heterogeneous* [CA01]. That is, in a heterogenous hierarchy the mapping between the adjacent levels need not be full. Also the members of a level can have ancestors in different levels. This scenario
is termed as *Unbalanced hierarchies* in [WWAL02].

For example, in figure 2.3, the level *city* is related to both *state* and *province*. However, it is not necessary that all *cities* are either related to *state* or *province*. Some *cities* can be related to *state* and some can to *province*. This creates partial mapping and the hierarchy is heterogeneous.

**Heterogeneous dimension levels**

We have mentioned earlier in Section 2.3.4 that members of a level have attributes and are used for classification as well as describing the properties of members. Normally the attributes of members are treated as attributes of that particular dimension level to which the members belong. The attributes that describe the properties of members are called *descriptive attributes* [LR98]. If all the members in a level have similar descriptive attributes, then the level is called *homogeneous dimension level*. If the members have different descriptive attributes, the level is called *heterogeneous dimension level*.

We have stated that the attributes *colour*, *weight*, *size* etc describe the properties of different *products* and hence they are descriptive attributes. These attributes are common to all products. However, some products may have attributes that are specific to them.

For example, *clothing size* for clothes, *power consumption* for electrical products etc. are attributes that are not common to all products. If those attributes are
included in the level, the members of that level have different descriptive attributes and hence the level becomes heterogeneous.

### 2.3.5 Aggregation Along a Hierarchy

When designing a hierarchy the essential goal is to provide aggregated data along the hierarchy. It is also important that the aggregation function should return correct results. This depends on the mapping that exists in the hierarchy. An example of this case can be illustrated through a non-strict hierarchy. By definition, in a non-strict hierarchy, a child member can have more than one parent. In the discussed mobile phone example (cf:Section 2.3.4), if the phone QC5678 is mapped to two other members, PDA and MP3 player, it returns an incorrect result for the function \( \text{SUM} \). That is, while computing Total Sales of product family with respect to the level product, the value could be three times greater than the expected result because the same product QC5678 is counted three times instead of once due to the mapping.

Similar to non-strict hierarchies, unbalanced hierarchies can also create problems due to the partial mapping between the members of different levels. The example hierarchy shown in figure 2.3 is an unbalanced hierarchy consisting of two different paths such as:

\[
\text{store} \rightarrow \text{city} \rightarrow \text{state} \rightarrow \text{region} \rightarrow \text{country}
\]
While computing an aggregation function with respect to region, these two paths will return different results. This happens due to the partial mapping between city to state and city to province.

The following conditions need to be satisfied in order to achieve correct aggregation along a path in a hierarchy. These conditions are known as summarisability conditions [A.S97].

- The disjointness condition states that a dimension level forms disjoint subsets over the members. This can further be described by means of mapping. If a member from a child level is related to more than one member of a parent, then that parent is not formed with disjoint members. For example, if the product mobile phone QC5678 is mapped to both the members mobile phone and PDA in the other level, then the members mobile phone and PDA are not disjoint. This is because QC5678 is common to both those members.

- The completeness condition tests the grouping of objects into different dimension levels which is addressed in two fold. First, all members should exist in different levels. The second one is that all members should participate in the grouping, which forms different dimension levels. The first condition is taken into consideration by making an assumption that the leaf
level in a hierarchy is complete and there is no missing member in that level.

For the group participation, an artificial node is normally introduced in the hierarchy, which is called the root node. We will discuss this later in Section 2.5.1 while reviewing the existing models.

- The type of measure as well as the statistical function being applied. In order to get correct results for aggregation, the correct statistical function must be applied depending on the type of measure. Types of measures are detailed in Section 2.5.2 based on the model [NFC99].

2.3.6 Source Data

As mentioned in Chapter 1, a data warehouse is derived from external data sources such as operational systems, web data, survey data etc. However the bottom-up designs consider only one source which is the ER model of the operational systems. But this is not always true. Generally data warehouse sources are heterogeneous and that is the reason why data integration is very challenging in data warehousing.

It is not necessary that source data should be known for data warehouse design. However it helps to get an idea of the existing data and their relationships so that any information demands related to unavailable source data can be eliminated at the initial stage of the design.
Since data warehouse data constitutes snapshots of source data, temporal aspects of data is an interesting consideration in design. Normally temporal aspects are addressed in data warehouse design by incorporating time as a default dimension. Most of the modelling techniques follow this as a rule of thumb. However certain approaches such as [TC99], [BBSA01], favour the inclusion of temporal aspects like time stamping in the design. If this happens a data warehouse becomes a real time data warehouse which conflicts with the definition of a data warehouse itself. As per Inmon definition [W.H92] (cf: Chapter 1) data warehouse data is time invariant. Hence in this thesis we follow Inmon definition and consider time as a default dimension. Any other discussion on temporal aspects could not be seen in this thesis but could be found in [SCE98], [N.L99].

We have introduced user’s queries as the main consideration for data warehouse design. So it is important to discuss queries in detail and in the next section we will describe data warehouse queries.

2.4 Data Warehouse and Queries

As we know, a data warehouse is a database specifically used for querying and reporting. In that respect queries are the best indicators of users’ information needs and thus can directly relate to data warehouse design. Currently data warehouse literature addresses queries mainly at the physical level and any relevant
conceptual approach is hard to identify as we will see later in this section. Before moving specific to queries and data warehouse design, we look at user-centered design and its significance. This serves as a motivating factor for our query oriented design framework. Later in this section, queries have been discussed with a view to support the schema design.

2.4.1 User-Centered Schema Design

In the context of this thesis, we define user-centered schema design as a design approach based on active involvement of users for a clear understanding of user requirements [MK01]. The popularity of this type of design method is increasing now days and can be found in various streams of applications. A survey on user-centered design methods, [KMST02], indicates that these methods have significantly improved the usefulness and usability of the products developed.

Generally users are involved in database design during the requirement analysis stage. At this stage all the users, those who are supposed to use the database are interviewed and their requirements are collected. In the case of a data warehouse, these users are administrators, executives, managers, and data analysts. The administrators are mainly in charge of the maintenance and performance of the data warehouse. In this respect they are interested in the hardware side as well as the database support, data loading and updates. Users other than administrators re-
retrieve information from a data warehouse. These users provide their requirements during the conceptual design stage so that a designer can produce a schema based on these requirements. Therefore the user requirements dictate the schema.

The discussion on users informal requirements specification in a data warehouse is limited to the operations which users wish to perform and business requirements such as business measures and business structures. Real user queries have been given very little importance during the design. However considering the purpose of a data warehouse, queries can play a major role in its design. Rather than assuming queries, it is better to study general data warehouse queries and suggest a schema based on those queries. This leads to the possibility of a user-centered design for data warehouses. In that case, the user involvement which is insisted in user-centered design methods, can be fulfilled by the queries. Alternately we could say that user requirements are represented by means of queries.

To identify the importance of queries in existing designs, we first review the query oriented approaches and later in Section 2.5, other designs are considered in detail.

### 2.4.2 Query Based Approaches

Even though there are informal discussions relating to data warehouse queries, none of them are comprehensive enough to contribute towards a conceptual schema
design. As per current designs (considered in detail in Section 2.5 and Section 2.6), a user can formulate queries based on a multidimensional data warehouse schema. That is, a schema determines what types of queries a user can ask. However, we feel that this is too restrictive and any practical schema design technique should consider queries during design even though we do not have all queries beforehand. We could find a similar argument in [TJP01] and [DBB+99]. Both these approaches insist upon query efficient schema design.

A method, suggested in [TJP01], uses queries to find a set of data cubes (sub cubes) from a base cube. Two algorithms; one for query similarity and another for cube schemas are proposed. Here the queries are assumed to be in Multidimensional expression (MDX) format. MDX is a declarative query language for multidimensional databases designed by Microsoft [Cor05]. The similarity algorithm places MDX queries into equivalence classes if the queries have directly or transitively have common dimensions. Then, for these equivalence classes of queries, a normalization algorithm is applied in the second part which ensures the correct aggregation and produces a set of cube schemas. Even though this work provides a practical design method, this is not a conceptual approach in the sense that the underlying model is relational and query similarity is tested by means of dimension keys.

Requirement based data cube design is proposed in [DBB+99]. This approach identifies data cubes that need to be pre-computed in order to improve query per-
formance. This work is an optimisation technique rather than a schema design method.

An XML data warehouse called X-Warehouse is proposed in [JWHS05] based on frequent query patterns. In this method, historical queries are transformed to query path transactions. From a set of query path transactions a database is created, then, by applying a mining technique significant query patterns are discovered. After the mining process, a set of query patterns are achieved and are used to construct a schema using the clustering technique. The main focus of this work is the mining algorithm to find frequent query patterns instead of a complete schema design methodology.

Most of the other studies related to queries are on query processing and optimisation ([AKS01], [JWWL00], [AVD95], [CCSC01], [WMJ00] and [D.T03]). These approaches classify queries based on different operations present in a query. As a part of analysing queries, here the nature of data warehouse queries and their classes are investigated. The query classification is mainly presented using terminologies from physical design literature due to the unavailability of relevant discussions at the conceptual level.
2.4.3 Query Types

Complex queries constitute the basis for any decision support systems [D.C97]. The complexity of queries can be addressed mainly in terms of data volume and the involved aggregations. Data warehouse queries are based on the data that is kept over a long period of time. In particular, a single query may require access to gigabytes of data to achieve an answer. This process is expected to complete in a reasonable period of time. Similarly the aggregation functions present in queries vary in its nature.

Queries in this environment are not simply querying, rather discovering information regarding the underlying business. For this reason queries inherit certain characteristics such as multidimensionality and aggregation and are generally known as multidimensional aggregate queries [D.T03] [AVD95]. We have introduced these concepts as data warehouse design considerations in the previous section. In fact these requirements were originated from the queries since they are multidimensional in nature. In order to capture these characteristics, the schema is also constructed as a multidimensional space. This shows the ability of queries to provide the constructs of a schema so it is relevant to investigate more about data warehouse queries and their usefulness towards schema design.

A data warehouse query can be described as an analytical process by which information is obtained for successful decision making. There are mainly two
aspects; static and dynamic, that can be extracted from a query. The static aspect indicates the object for analysis which in turn defines the multidimensional space. On the other hand the dynamic aspect provides aggregation and operations required by the user. It is further detailed below.

In data warehousing users are interested in trends rather than looking at individual records. This means aggregated data is required. Aggregation in data warehouse queries can range from computing simple totals to complex statistical functions such as moving average at different granularities. Most of the discussions on aggregation functions are related to functions such as \( SUM, AVG, MIN, MAX, \) and \( COUNT \) ([MDS98b], [Leh98]). Another study in this area can be found in [CT99]. This work studied different aggregation functions in the context of database query languages while as in [MFW03], classes of aggregation functions such as distributive and algebraic functions have been studied in the context of aggregating multidimensional data.

Other than aggregation functions multidimensional constructs are also present in a query. These include the measures that need to be summarised and its granularity. The granularity is normally indicated through dimension levels.

Informally queries can be grouped based on the dimension levels present in them. If a query has more than one dimension level and there is a grouping on these dimension levels then it is called a group-by-query ([WMJ00]). In group-by queries, the dimension levels may be from different dimensions. Literature
related to physical designs refer to these queries as join queries [CCSC01]. An example of a grouping query is sales of home brand products by year. In this query, grouping on sales is requested based on dimension levels brand and year. This query can also be described as a join query because, tables corresponding to the dimension levels and measure need to be joined in order to answer the query.

Sometimes queries place a restriction on dimension levels. A query that restricts dimension levels to certain intervals is called a range query [DA01]. An example of such a query is show the sales in southern region during 1995-2000. In this query, there are restrictions on both the dimension levels region and year. Region is restricted to southern region and year is limited to 1995-2000.

Another type of query that are frequently referred in literature are roll-up/drill-down queries. As the name suggests these queries are with respect to the operators roll-up/drill-down. Here queries utilise the hierarchical nature of dimensions to navigate from the most detailed level to the highly summarised level and vice-versa. Each rolling-up or drilling down from one level to another level can be seen as a roll-up or drill-down query. Querying from weekly sales to monthly sales is an example of a roll-up whereas looking at weekly sales from monthly sales is a drill-down.

Based on the discussion presented in this section, it can be concluded that

- queries define the schema constructs;
• a conceptual study on queries which influence schema design is missing and
• in particular, a query classification could found only at the physical level
and it is informal.

Considering the significance of queries in data warehouses, in this thesis queries
has been given utmost priority in the design. For this purpose, a conceptual per-
spective of queries have been presented and this can be found in Chapter 3.

2.5 Top-down Designs

In this section we discuss different top-down designs which were proposed as data
models for data warehouses. These data models define multidimensional space
through facts and dimensions. Aggregation is addressed by hierarchy definition
as well as conditions for summarisability which were discussed in Section 2.3.5.
Since various approaches define models with respect to the representation, mod-
els are grouped as cube models, extended ER models and other approaches. Cube
models organise data in the form of a multidimensional cube whereas extended
ER models provide graphical notations based on ER concepts for data representa-
tion. Remaining approaches that do not fit into the above categories are grouped
as other approaches. Each of these approaches are discussed in detail in the sub-
sequent sections.
2.5.1 Cube Models

Cube models, such as [LW96], [RGS97], [AH97], [P.V98], proposed in early stages of the research were focussed on operators and/or algebra rather than requirements. However like other cube models, [TJP98], [TC99], [NA01], [Leh98] and [AKS01], these models also include hierarchy definition in the schema.

All the above models with the exception of [AKS01], define hierarchy as a dimension schema by means of partially ordered sets. That is, a dimension schema is defined as a partially ordered set of dimension levels with specific domains. The mapping between the levels is achieved by means of functions. These functions are generally called roll-up functions [MA01] and operate from a higher domain to a smaller domain. Since the mappings between levels are considered as functions, a child level can participate with only one parent level and hence the hierarchy is strict. Another constraint such as domain disjointness can also be found in the schema definition which satisfies the summarisability condition addressed in Section 2.3.5 and ensures correct aggregation along the hierarchy.

The description of aggregation functions is considered in [NA01]. The allowed functions in this case are $SUM$, $COUNT$, $MAX$, and $MIN$. Additionally two more operations $COMPOSITE$ and $NONE$, are suggested.
Composite is used where measures cannot be utilised in order to automatically derive higher aggregation from a lower one. The operation None is applied to measures that are not aggregated.

The cube definition in [TC99] explained the partial ordering through greater than relationship. This relationship is defined as follows.

one level is greater than another level if the member of the former level logically contains the members of the latter. This logical containment allows a level to participate with any level above it thus the schema captures multiple paths in a dimension. This approach also distinguishes aggregation functions as type $SUM(\Sigma)$, $AVE(\Phi)$, and $Constant(C)$ in such a way that, if it is $SUM$, the data can be added together. In case of $AVE$, average is applicable and $Constant$ is used for counting. This model satisfies all the requirements related to hierarchies such as multiple/ shared path as well as non-strict hierarchies. But the approach lacks derived measures and specific consideration of queries.

In Section 2.3.4, we have stated heterogeneous attributes of a dimension level. A cube model that support heterogeneous attributes is [Leh98]. Normal cube definition is extended in this case to provide attribute based analysis. The basic cube is proposed as a primary multidimensional object (PMO) in which the business structure is defined as a balanced tree. This is termed as a classification hierarchy and each node in the hierarchy is defined as an instance of an attribute called classification attribute. The nodes in the hierarchy are termed as classification nodes.
For example, if *product family* is a classification attribute in product structure, it can have instances like *VCR, Camcorder, Washers* etc. Similarly all classification attributes have instances and each one of those form a node in the hierarchy. Since the hierarchy is a tree; it is strict, hence unbalanced and non-strict conditions are not applicable in this case.

As in [TC99], aggregation here is addressed as aggregation type \(\{\Sigma, \Phi, c\}\). The first two aggregation types are the same as the definition in [TC99] whereas the constant \(c\) is interpreted as no aggregation in this case.

As a second level, the cube extension is proposed which includes the definition of a *secondary multidimensional object (SMO)*. An SMO is defined as a set of classification nodes and a set of attributes called dimensional attributes that are applicable to the classification nodes. In the given *product family* example, *VCR, Washers* are two classification nodes. According to the SMO definition, these nodes can have different attributes that are local to each node. That is, the definition allows one to model heterogeneous attributes for the nodes in a classification hierarchy and thus able to undertake analysis based on features of classification nodes.

By introducing PMO and SMO, this work shows data cubes are nested. This is done by defining dimensions in two levels; one is classification-oriented and another is feature-oriented. The aggregation type in the definition of PMO shows the allowed aggregation operations applicable to the measure. But this is not
enough to capture whether the measure is additive along different dimensions or not. Attributes are defined at the instance level in this approach. That is, the structure and contents are not separated. A formal data model should take this into consideration and needs to separate structure from the contents [MSHD98].

A different modelling technique suggested in [AKS01] captures the semantics of hierarchies by means of dimension paths. A dimension path is introduced as an $n \leftrightarrow way$ relationship which relates a member of a parent level to one or more members of the children levels. Note that, relationship is not strict, which means, any two members at the parent level can be related to the same child-level member and non-strict hierarchy is allowed. Formally, a dimension path is defined as a non-empty set of $n \leftrightarrow way$ relationships called drilling relationships with certain properties. These properties are:

- in the graph of the defined set of relationships there is exactly one node that has no incoming edges.
- the graph is a direct acyclic graph (DAG).
- there are no two relationships in the graph having the same parent.

Since the graph is a DAG, multiple hierarchy is possible here but the last property restricts the possibility of a parent level participating with two children levels. So redundant levels are necessary to capture unbalanced hierarchies. Also,
this model does not consider aggregation functions and derived measures. In fact, none of the cube models support derived measures.

While designing the hierarchies, the models prevented the possibility of having cycles. This is mainly due to the fact that in data warehouse applications such an operation, that is starting from one level and coming back to the same level (cyclic operations), is meaningless. Also business structures do not allow cycles. Cycles are prevented in models either by defining transitive closure of the partial order relation or by defining the graph as acyclic like in [AKS01].

Another aspect related to hierarchies and aggregation addressed in models is the completeness from summarisability conditions. The completeness condition, stated in Section 2.3.5, insists the participation of all members of a lower level to one of the higher levels. For this purpose, models such as [NA01], [TC99], [Leh98] and [AKS01] introduced a new artificial level in the hierarchy schema which contains a unique member called All. It is assumed that all the members of the leaf levels are related to this unique member directly or indirectly and thus completeness is achieved.

### 2.5.2 Extended ER Models

The ER model is widely recognised in enterprises as a conceptual model for transaction systems. However, the concepts in ER are not enough to separate the se-
mantics of quantifying and qualifying data in a data warehouse [MDS98b]. So researchers have extended ER concepts in order to address data warehouse requirements. Modelling approaches that need to be discussed in this context are [NFC99], [FS99] and [CMGD98]. The starER ([NFC99]) uses ER concepts and the star representation proposed by [R.K96]. The Multidimensional ER model (ME/R) by [CMGD98] also uses ER concepts along with new graphical notations.

Similar to the cube models, extended ER models also explicitly consider hierarchies and are modeled through entity sets and relationship sets. In [NFC99] the relationship set between entity sets are described using ER concepts such as specialisation/generalisation, aggregation and membership. Specifically, hierarchies in dimension are defined by the membership relationship. The allowed cardinalities of the membership are many-to-many, many-to-one and one-to-many. The strictness or the completeness of the membership is shown by the cardinality of the membership and an accompanying constraint.

This approach provides a detailed discussion on measures that we have mentioned as one of the summarisability conditions in Section 2.3.5. That is, measures are classified as flow, stock and value-per-unit. Stock represents the state of something at a specific point in time. For example, the amount sold for the fact sales. Flow represents the changes or commutative effect over a period of time for some parameter in an environment monitored by a data warehouse. Flow can also be summarised along the time dimension. Value-per-unit is similar to stock
but it is measured for a fixed time. The units of the measurements are different in these two cases. That is, value-per-unit describes the recording of the parameter in relation to some unit in the environment monitored. The distinction of these two types can be made on *what* is recorded rather than *when* it is recorded. An example of this type is *tax* per the amount sold. Tax makes sense only in the context of amount sold. Summarisability of measure types value-per-unit and stock depends on the statistical functions applied. This work mainly focussed on the extension of ER concepts to capture data warehouse requirements. Through the membership definition and the classification of measures, this starER model supports non-strict hierarchies and multiple hierarchies as well as correct aggregation along the hierarchies. But similar to other top-down designs, only conceptualisation is addressed here. Any other discussion on schema selection/ refinement can not be found here.

The other ER approach, [CMGD98], define hierarchies in terms of dimension levels and a binary relationship set. The binary relationship relates one dimension level to other. Formally, dimensions are defined as a direct acyclic graph with a finite set of levels and a set of binary relationships. An integrity constraint such as transitive closure of the relation is applied in order to avoid cycles in the graph. Multiple hierarchies and alternate paths are possible in this case as the graph is a DAG.
A detailed discussion of the above approach can be found in [C.S01]. Instead of dimension, the concept of classification is used in this case, for modelling business structure. That is, initially, dimension and classification are considered as two concepts. Later, at the schema level the two concepts are connected by considering each level in the classification as a dimension. Compared to other dimension definitions this is more pragmatic because, initially, only classifications are known, not dimensions. Then it is better to define classification first and select dimensions later based on that classification. However, derived measure is intentionally left out in this approach as a functional requirement.

Aggregation has been prioritised in [FS99] where the ER concepts are extended in order to include the description of aggregation. The main focus of this approach is a graphical representation which includes aggregation of individual entities. A new entity called \textit{aggregated entity} is proposed for this purpose which represents the aggregation of basic entities.

The remaining models proposed for data warehouses are compared against the requirements in Section 2.5.3. Other than design techniques, approaches which address any of the presented requirements are also considered.
2.5.3 Other Approaches

Like the cube definition, another type of model definition is in the form of *tables*. Even though the models are defined in terms of tables they do not assume any implementation. Examples of these type of models are [LR98] and [ML97]. In [ML97], hierarchy is modeled as a function from a set of dimension names to a set of attributes. In [LR98], hierarchy is defined by means of partial ordering similar to that of [TJP98], [TC99] discussed in Section 2.5.1.

Another well known table schema definition is *star schema* or *Dimension Model* by [R.K96]. In contrast to [LR98] and [ML97], this approach assumes relational implementation so this is a logical model rather than a conceptual one. A star representation includes a central *fact table* and a number of surrounding *dimension tables*. Dimension tables are connected to a fact table using keys. Due to the simplified star like data representation, this model is widely accepted in industry. Star representation addresses the need of hierarchies through dimension tables. Each dimension table in a star contains attributes that are necessary for a hierarchy. None of the requirements other than a strict hierarchy can be seen in these table definitions.

The *dimensional fact model (DFM)*, proposed by [MDS98a], considers hierarchy as a tree with a many-to-one relationship between its levels. The aggregation statement in a schema declares that a measure can be aggregated along a dimen-
sion by means of an aggregation operator. The aggregation operators considered in this context are \(\{\text{SUM, AVG, COUNT, MIN, MAX, AND, OR, ...}\}\).

If there are no aggregation statements for a given measure and dimension, then the measure cannot be aggregated along that dimension. By the tree definition itself it is clear that the hierarchy is balanced and strict. That means requirements related to hierarchies are not fully met in this case. A query language called \textit{fact instance expression} is another feature of this model and is used to measure the workload on the data warehouse. This language is proposed only in the context of this particular model.

The object-oriented approach; [JMJS01], can be distinguished from other works due to the inclusion of derived measure in the model. This is achieved by placing a constraint next to a measure which is derived. The derivation rules for measures are also included in the graphical representation. Hierarchies are defined in terms of association of classes and form a DAG. The DAG structure represents both alternate and multiple paths. Strictness and completeness of a hierarchy is achieved by defining constraints such as \textit{completeness, non-strict} etc. in the classes in a hierarchy. The main focus of this object-oriented approach is to capture the requirements in a graphical representation. For this purpose UML class diagrams are used along with object-oriented concepts. Even though the conceptualisation captures all the requirements, this design is more focussed on representation rather than a complete design methodology.
Extension of description logic has been applied in [MU97] to define cube operations. In this case we can find only the formalisation of hierarchies. No other requirements are considered and much of the work is devoted to operators.

Specific Considerations

As we mentioned earlier, there are approaches like [AJF01], [EM99], [JLS99] and [M.S03] which address specific requirements. The main requirements addressed in these approaches are hierarchies and aggregation along the hierarchy. These approaches studied different aspects related to hierarchical structure and its ramifications in data aggregation. Since these approaches address only individual requirements they can not be considered as complete design techniques.

Discussions on hierarchy and aggregation are presented in [AJF01]. This work studied the usefulness of the part-whole relationship to solve issues related to hierarchies such as non-strict, multiple hierarchy etc. Hierarchy definition and characterisation could be found in [EM99] and [JLS99]. At the design level these works are less significant since they only define hierarchy and then the focus is shifted to operators and query language.

Non-strict hierarchy is presented as exceptions in [MA01]. A new language called IRAH (standing for Intensional Redefinition of Aggregation Hierarchies) is proposed to handle exceptions that occur during hierarchy design.
A study, related to hierarchies [TJP01], demonstrated the relevance of dependency theory in hierarchy design. Dependencies such as anti-closure dependency, functional dependencies and boolean dependencies are applied in various hierarchical structure and showed how correct aggregation can be achieved in each case.

Aggregation in the context of heterogeneous dimensions were addressed in [WJH98], [CA01] and [CA02]. Lehner et al. ([WJH98]) suggested normal forms called dimensional normal forms for cases of dimensions causing heterogeneity. Heterogeneity is addressed both in terms of heterogenous hierarchy and heterogenous levels. In the case of heterogenous hierarchy, the hierarchy is made homogenous by applying functional dependency between members in the participating levels. A constraint is applied in such a way that, in a hierarchy for any two members from two different levels, there is a strong functional dependency. This means that every member in one level is related to exactly one member in the other level. If this condition is not satisfied, the functional dependency becomes weak and violates the normal form.

Heterogenous level is handled in this approach by placing the attributes that cause heterogeneity outside of the level in the hierarchy. These attributes are considered as separate levels of that hierarchy. This proposed transformation facilitates the ability to model heterogeneous attributes that are specific to certain members.
Summarisation in heterogenous hierarchies were studied in [CA01]. Hierarchy is defined as a set of parent/child relationships. If such a relation exists between two members from two levels, those levels are directly connected. The definition of a path in the hierarchy and constraints for valid paths, called *dimension constraints*, distinguish this work from other hierarchy definitions. The proposed constraints specify legal paths in a dimension and thus ensure correct aggregation.

Attempts have been made in [AJS01] and [M.S03] to study *facts* in detail. The object-oriented approach ([AJS01]) defines fact in terms of a group of measures. The grouping is done depending on the level of detail of a *measure*. In general, a fact is defined as a connected directed graph. The vertex of the graph represents a cube containing cells and an edge corresponds to part-whole relationships between cells in the cube in the fact. The part-whole relationship says that every cell at the target cube can be decomposed as a collection of cells in the source cube.

A graph based study related to fact can be found in [M.S03]. The proposed graph is known as *data warehouse graph*. In a data warehouse graph, each node represents either a fact or a dimension level. A node is connected to another node by a directed edge which represents a *reference*. This means that if there is a reference between two nodes, then those two nodes are related through their attributes. This proposed graph structure allows to model relationships between facts that could not be found in other models.
In this section, we have discussed different top-down designs from the requirement point of view. Regardless of the representation, all models agreed upon the hierarchy requirement needed for aggregated data. However, the generality of a hierarchy varies as it can be strict like a tree or non-strict and unbalanced. A number of models include the types of allowed aggregation in the schema, along with measures and dimensions. In summary, similar to representation, schema constructs are also vary from model to model. There is no common consensus about what should be represented in a schema. Moreover none of the designs, except [JMJS01], considered derived measures. Another important aspect that emerged from the discussions is the role of queries. None of the presented top-down designs considered queries during the design.

2.6 Bottom-Up Designs

In this section, bottom-up designs are explained in detail. As mentioned in Section 2.2.1, bottom-up designs provide transformation methods to construct data warehouse model from existing ER models of operational systems. Main approaches in this area are [MDS98a], [LR98], [D.GL00] and [CK02]. All these approaches derive a data warehouse structure from an ER model and the selection of the data warehouse model varies in each approach. Unlike top-down designs, any discussion on identification of requirements and the formalisation can not be found in
these designs. The main focus in this case is the identification of facts, dimensions and associated hierarchies from an ER model. Algorithms or transformation guidelines can be found in this regard which map one model to the other. So here we discuss the transformation algorithms in detail.

2.6.1 ER to Dimensional Fact Model

A semi-automated methodology, proposed in [MDS98a], uses the dimensional fact model (DFM) (proposed by the same authors elsewhere ([MDS98b]) as the target model. The derivation of the DFM from an ER model consists of the following steps.

- **Identifying the fact**: A fact on an ER model is either an entity or an $n$-ary relationship between entities. Entities or relationships representing frequently updated archives are considered as good candidates for facts. Each fact identified from the ER model is represented as the root of different fact schemas.

- **Building the attribute tree**: An attribute tree is selected from a portion of the ER schema that contains the identified fact. An attribute tree is defined as a tree in such a way that:

  - each vertex corresponds to an attribute,
  - the root corresponds to the identified fact, and
– for each vertex $v$, the corresponding attribute functionally determines all the attributes corresponding to the descendants of $v$.

For a chosen fact, the attribute tree is created automatically by calling a recursive procedure.

• *Pruning and grafting the attribute tree:* Not all the attributes in the attribute tree will be needed for a data warehouse. Unnecessary attributes can be eliminated by pruning and grafting. Pruning is an operation used to drop subtrees from the attribute trees and that dropped attributes will not be in the fact schema. Similar to pruning, grafting is used for eliminating a vertex that carries uninteresting information. In this case the descendants will be preserved.

• *Defining dimensions:* Dimensions are identified among the children vertices of the root. Normally, dimensions are discrete attributes or ranges of discrete or continuous attributes.

• *Defining fact attributes:* Fact attributes are either counts of the number of instances of the fact or the numerical attributes of the attribute tree.

• *Defining hierarchies:* This is the last step of defining the fact schema. At this stage, hierarchy on the dimension is defined. Along the hierarchy, attributes are arranged into a tree such that a many-to-one relationship holds
between each node and its descendants. During this step it is also possible to prune and graft the tree in order to eliminate unwanted information. At this stage, attributes used for informative purpose are identified as non-dimensional attributes.

2.6.2 ER to Multidimensional Model

The methodology proposed by [LR98] uses a model called the multidimensional model (MD) for the multidimensional representation. The proposed methodology consists of the following steps.

- **Identification of facts and dimension**: During this step, the identification of facts, measures and dimensions are done from the ER model. Similar to the previous approach [MDS98a], facts are selected from entities, attributes or relationships. A dimension is selected as a subschema of the given ER schema and identified as entities related by one-to-many relationships.

- **Restructuring ER schema**: In this step, the original ER schema is reorganized in order to describe the facts and dimensions in a better way. After this step, a new ER schema is produced and this can directly translate into the MD model. Restructuring of the ER model consists of the following steps.
  
  - **Representing facts as entities**: facts normally correspond to entities but
it is possible that fact can be described by attributes or relationships. In that case, the attributes and relationship need to be translated into entities.

- **Adding dimensions**: During this step additional dimensions can be added if necessary.

- **Refining the levels of each dimensions**: Various dimension levels of a dimension are identified at this stage. This is done by replacing a many-to-many relationship with a one-to many relationship. Add new entities for new dimension levels (if any) with identifiers. Adding new entities depends on additional levels needed in the dimension.

- **Derivation of a dimensional graph**: From the restructured ER schema a graph is derived which is called the dimensional graph. Each node in a dimensional graph corresponds to either an entity that represents the domain of that entity, or an attribute that represents the domain of that attribute. The arc between two nodes represents a function between the corresponding domains. Four kinds of nodes can be identified in a dimensional graph. A fact node originates from a fact entity, level nodes from dimension levels, descriptive nodes originate from descriptive attributes and measure nodes originate from measures. The dimensions are the sub-graph of the dimensional graph.
• Translation to \textit{MD} model: The dimensional graph is translated to the MD model in this step. The dimensions of the MD model are the dimensions of the dimensional graph, and for each dimension there are dimension levels. The subgraphs of the dimensional graphs denotes the partial ordering among the dimension levels of the \textit{MD} model.

2.6.3 \textbf{ER to Star}

The method proposed by [D.l00] describes developing a star model from the ER model. A three step transformation is suggested and those steps are as follows:

• \textit{Classify entities}: Entities in an ER model are classified into \textit{transaction entities}, \textit{component entities} and \textit{classification entities}. Transaction entities form the basis for constructing the fact table in a star schema and they are characterized by an event that happened at some point in time. They also contain measures or quantities that may be summarised. A component entity in an ER model is that entity which is directly related to the transaction entity via a one-to-many relationship. Another type of entity called classification entity is also identified. Entities that are related to component entities directly or transitively are called classification entities.

• \textit{Identification of hierarchies}: During this step a hierarchical representation from the ER model is identified. A hierarchy in an ER model is considered
as a sequence of entities joined together by one-to-many relationships and all aligned in the same direction.

- **Produce Dimensional model**: Two operators; collapse hierarchy and aggregation are suggested to produce a star model from an ER model.

  A *collapse hierarchy* is used to collapse higher level entities into lower level entities. After this operation, the lower level entity contains its original attributes as well as the attribute of the collapsed entity and thus introduces redundancy. This operator is applicable till the bottom of a hierarchy.

  Aggregation operator is used to create new entities containing summarized data. For example, this operator can be used to create an entity sale item to get a new aggregated entity called *product summary*, which shows the total sales for each product.

  A star schema is derived from the identified entities in the following way.

  - A fact table is formed for each transaction entity.
  - A dimension table is formed for each identified component entity.

  Another derivation process could be found in [MA99], in which multidimensional structures are obtained from the underlying conceptual data model of the operational systems. The conceptual data model under consideration is the Structured Entity Relationship Model (SERM) ([E.J88]) which is an extension of the
conventional entity relationship model. The derivation of multidimensional structure from an SERM diagram consists of three steps. They are as follows:

- **Identification of business measures**: Business measures are identified through the argumentation chain goals-services-measures. Goal describes the nature and purpose of the service production. The service performance and the set goals are evaluated through quantitative measures. The adequate measures are identified using business events and the identified measures are then assigned to the data objects in SERM.

- **Identification of dimensions and dimensions hierarchy**: Dimensions and dimensions hierarchies are identified from the SERM by using closure of existency. The SERM structure provides the dependency and from these dependencies closure can be identified. The starting point for the closure identification is the data object type chosen as the measure. Then, the other data object types, along the edges of the graph from the right to the left, are examined. The data objects with existency pre-requisites to the measure are selected and the necessary dimensions are chosen using domain knowledge. Hierarchies in the dimensions are identified using the zero-to-many or one-to-many relationship between two or more adjacent data objects.

- **Mapping to star schema**: The structures identified in the previous steps are transformed as measures in a fact table and dimension tables in a star
schema. Since star schemas do not offer any support for the summarisation rules or the consolidation rules along hierarchies, it is left out for analysis tools to deal with subsequently.

### 2.6.4 ER to Multidimensional ER

An automated conceptual schema design is proposed in [CK02]. The input of the proposed algorithm is an ER schema in tabular form and output is a set of candidate ME/R schema in tabular form (We have discussed the ME/R model earlier in Section 2.5.2 as an extended ER approach).

The steps involved in the algorithm can be generalized as follows:

- Find entities with numerical fields and create a fact node for each entity identified.
- Based on numeric fields in the entities, create numerical attributes of each fact node.
- Create date/time levels with any date/time type fields per fact node.
- Create a dimension level containing the remaining entity attributes other than numeric, key and date fields.
- Recursively examine the relationships of the entities to add additional levels in a hierarchical manner.
That is, recursive traversal of relationships is conducted and dimensions of
fact are created.

The output of this algorithm produces a set of candidate ME/R schemas in tab-
ular form. These candidate schemas are then evaluated against a set of queries.
Mainly two aspects from the queries are tested: one tests for tables in the FROM
clause of the query and the other is numeric fields in the SELECT clause. If the
candidate schema does not contain the tables in the FROM clause then it can not
answer the query. The numerical fields stand for the measures that need to be the
attributes of the fact node to answer the query. An evaluation algorithm identifies
whether various candidate schemas meet the requirements of the queries or not.
Finally, a manual refinement is suggested for the conceptual schema refinement.
The suggested steps for schema refinement are:

- If user queries are known, eliminate the unnecessary candidate schemas.

- Check the credibility of the measures in each fact node. That is check
  whether the measures in the fact nodes are really measures. There is a pos-
sibility of having descriptive attributes on the fact node that are not really
measures.

- Decide the granularity for date/time information.
• Check the necessity of calculated fields and provide derived measures if necessary.

• Check whether schemas can be merged.

• Eliminate fields if they are not necessary.

• Check for data required that did not exist in the operational database.

2.6.5 A Hybrid Approach

A different schema design method which integrates both top-down and bottom-up approaches can be found in [AFS+01]. The top-down analysis puts emphasis on user requirements whereas the bottom-up design is similar to approaches such as [MDS98b], [LR98] etc.

In this case, the top-down design presents a totally new approach. The design starts with goal identification, which identifies different goals of various users. This is achieved by interviewing users iteratively. Finally the goals are determined and integrated. A table called abstraction sheet is created for each goal which further describes the nature of each goal in terms of purpose, quality focus, users involvement etc. At the end of the design, the abstraction sheet is used for creating star schemas which satisfy the goals in various abstraction sheets. All these star schemas may not be implementable due to the lack of information in an
operational database. In order to compliment the schemas from the top-down approach, a subsequent bottom-up design phase is suggested. The main purpose of this bottom-up design phase is to identify data from the operational schema. Similar to other methodologies discussed in Section 2.6, this phase starts from the ER schema of the operational systems. Using transformation rules the ER schema is transformed into a graph known as connectivity graph. In the connectivity graph, nodes represents facts and dimensions. The edges represent relationships between them. Through an algorithmic procedure, star schemas are generated from the connectivity graph.

As a final stage of the design, integration of top-down and bottom-up designs are suggested. At this stage, the schemas generated through the two different phases are compared. A schema matching technique is proposed for this purpose which compare and rank the schemas based on the goals that need to be satisfied.

In this section, we have studied different transformation methods proposed in a data warehouse context. These methods provide guidelines to identify candidate facts and dimensions from entities and relationships. On top of that, transformation algorithms can also be found in most cases. While providing the transformation, the methodologies only considered ER model; any source data, other than ER representation is beyond the scope. If the source data is not in ER form, additional techniques are required for the data transformation.
Generally in bottom-up designs, the design requirements are not well discussed. This is left out as part of conceptualisation. These designs assume that the target data model satisfies the necessary data warehouse requirements. But in certain approaches we could find relevant discussion towards schema refinement. For example, schema selection is considered in [CK02]. The query-based schema selection proposed here can be seen as a consideration of queries in schema design. However, the design is limited to SQL queries since the algorithm could only handle queries in SQL form.

In [MDS98a] schema refinement is addressed through the operators, Pruning and grafting. These operators eliminate unwanted information from the schema. Similarly the collapse operator in [D.100] can be used to collapse a hierarchy in a schema. Nonetheless none of these approaches provide a formal method to eliminate schema constructs from the initially selected constructs. As a matter of fact, the current methods suggest how information can be eliminated, whereas other aspects, such as what should be eliminated and which criteria should be used for elimination, also need to be addressed.

### 2.7 Towards a Query Oriented Schema Design

In Section 2.5 and Section 2.6, we have seen existing top-down and bottom-up approaches. In top-down designs we could find models based on cube and ER con-
cepts as well as other representations such as dimensional fact model. These designs formally define modelling constructs and present a conceptualisation of requirements. Different from top-down designs, bottom-up designs are data driven and provide transformation methods to convert an ER model to a multidimensional model. Based on the study presented in the previous sections, we are now in a position to describe the properties which form the basis for the proposed schema design framework. From our observations and own hypothesis, the properties that are critical for a conceptual design methodology are detailed here.

- **Independence**: The independence property could be addressed mainly in two levels; one is system design as a whole and the other is each phase of the design. The system design should provide independence between different phases of a design. This means, there is a need for the distinction between the conceptual, logical and physical levels which allows possible changes at the conceptual level without changing anything at the physical level and vice-versa.

  Independence can be further described in terms of individual design phases. Each phase should be independent in such a way that the phase under consideration should discuss schema properties/ refinement that leads to other phases of the design as well as schema evolution. Additionally, proper guidelines are necessary to reach a schema from the model so that the de-
signer has the flexibility to develop schema on demand. In another words a systematic approach is necessary for the design.

- **Clear and unambiguous formalisation**: The conceptual formalisations used in a design should be clear and unambiguous. That is, a rigorous, expressive and mathematically sound framework should be developed.

- **Generality**: The design should be general enough to subsume other models. Then it is easy to switch from one model to another.

Bottom-up designs; provide the transformation methods, are not focussed on conceptualisation. In that respect it is difficult to discuss those approaches in terms of the above mentioned properties as they need not strictly follow the properties of a methodology. The only requirement that is being taken into consideration is the operational data. Nonetheless, as stated in Section 2.6, one particular aspect associated with a design can be found here which is the schema refinement.

Top-down designs formalise requirements, however, they do not satisfy all the properties. Independence and generality are the properties that are mainly misinterpreted in many approaches. Especially in the case of cube models it is difficult to draw a fine line between conceptual and logical levels. There is a chance of overlap between the levels which is clear in approaches such as [LR98], [AH97], [RGS97].
Extended ER concepts and other approaches satisfy independence property. But each model has its own graphical notations and there is no standardisation in this regard. This means those models are general with respect to implementation, but it is difficult to compare the generality between the models. Also any discussion on schema evolution and refinement are left out from the design.

Considering the formalisation property, it varies in each model. In almost all models, the concept of a dimension is presented using the definition of a hierarchy. That means a dimension is treated as a hierarchy of levels. In cube models this consideration makes a conflict in the concept. Cube models allow only one level from a hierarchy as its dimension; then any levels from a hierarchy become dimensions. There is no formal method for the identification of schema constructs and this is the general case in all other designs. A systematic approach, starting from formal definition to the identification of schema constructs, is still lacking.

While reviewing the existing designs with respect to the requirements, we have seen that none of the approaches considered real queries during the design process. This could be seen as a lack of dynamic aspect in design. A conceptual design should consider all relevant static and dynamic aspects. Summarising the discussion, the following points are valid with respect to current designs.

- Bottom-up designs cannot be seen as a complete methodology due to the lack of conceptualisation phase in the design.
• Top-down designs vary in their formalisation but provide independence and
generality to an extent. However, they do not include guidelines to reach a
schema from the model and schema evolution.

• Considering the requirements, the majority of the designs fail to consider
user queries during the design process. Those approaches which include
queries either deviate from the conceptual level or the importance of queries
are relatively very low.

To reinstate the validity of our conclusions, a comparison table is presented in
Table 2.1. The comparison is made on the following criteria.

• Formal procedure: This check identifies whether the design is systematic.
In other words, if the requirements are stated clearly and the final schema is
reached through a formal procedure that design satisfies this property.

• Conceptualisation: Examined the formalisation presented especially hier-
archies and derived measures.

• Independence: This is tested in terms of all phases of design as well as
conceptual phase itself. In the conceptualisation phase any discussion on
schema refinement/ transformation and evolution were evaluated. In current
designs, independence is discussed only in terms of different phases of the
design. That is conceptual design is independent of logical and physical
levels and vice-versa. However we feel that within each individual phase, the design should address possible schemas or guidelines to translate the proposed model to various schemas otherwise the design will be incomplete and additional efforts are required to translate the schema to other phases of the designs.

- **Generality**: Generality is introduced as an evaluation on the model quality. This check identifies whether the design offers provision to translate to other models or it addresses this aspect.

- **Queries**: Identified the importance of queries in the design process. More specifically, investigated if the query consideration is implicit or explicit. As we know the existing designs are developed based on the assumption that queries are multidimensional and aggregation is required. Also these designs focus on ad-hoc queries. But our interest is more than these considerations. We like to know whether there is any specific discussion on queries with respect to that particular design. If it is only multidimensionality / aggregation we assume the relevance of queries is nil in those designs.

For the comparison, models have been categorised based on the representation or the concepts used for modeling. Each property is tested against a category of models and the result is presented. If most of the models in one particular category satisfies a property, it is indicated by ✓ otherwise it is a ×. For example,
Table 2.1: Current designs vs Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cube [TC99], [AKS01]</td>
</tr>
<tr>
<td></td>
<td>Extended ER [CMGD98], [NFC99]</td>
</tr>
<tr>
<td></td>
<td>OO concepts [JMJS01]</td>
</tr>
<tr>
<td></td>
<td>Other approaches [MDS98b], [LR98]</td>
</tr>
<tr>
<td></td>
<td>MD from ER [D.1999], [MDS98a]</td>
</tr>
<tr>
<td>Formal procedure</td>
<td>×</td>
</tr>
<tr>
<td>Conceptualisation</td>
<td>-</td>
</tr>
<tr>
<td>Generality (Transformation)</td>
<td>×</td>
</tr>
<tr>
<td>Independence</td>
<td></td>
</tr>
<tr>
<td>a) Separation between different</td>
<td></td>
</tr>
<tr>
<td>phases of design</td>
<td>×</td>
</tr>
<tr>
<td>b) Schema refinement/evolution</td>
<td>×</td>
</tr>
<tr>
<td>Queries</td>
<td>×</td>
</tr>
</tbody>
</table>

the object-oriented approach [JMJS01] fulfills the conceptualisation property by addressing all the requirements related to hierarchies as well as derived measures. Similarly cube designs do not provide a systematic design procedure hence property is not valid for cubes. There are designs which address certain aspect of a property but not completely. Those cases are given consideration and marked by a *hyphen* (−). That means if a design presents some relevant discussion on any of the properties it is treated as a partial consideration of that property and is not enough for the data warehouse design. For example, cube designs conceptualise hierarchies but all the requirements are not satisfied.
The comparison Table 2.1 shows that the current designs do not necessarily follow a formal procedure for schema design. Conceptualisation is addressed in all designs except the bottom-up designs where a multidimensional model is derived from an ER model. But the majority of the designs fail to formalise all the requirements. Most of the designs claim implementation independent models nonetheless relevant discussion on logical mapping is missing in many cases which is necessary to prove the generality of the model. Similarly, independence is addressed only in terms of various phases of the design with not much consideration on schema refinement or evolution. In that sense conceptual design is not fully independent. Another main problem is the lack of consideration of queries. None of the designs treat queries as a design consideration except for the fact that they assume certain types of queries need to be answered.

From the study presented in this chapter it can be concluded that the current situation demands a complete design framework which satisfies all the above addressed properties. In this thesis we propose such a framework based on real queries. The proposed framework is very general and is independent of any implementation.
2.8 Summary

In this chapter, we have illustrated a set of requirements that are necessary for data warehouse schema design. These requirements were presented from a business and users’ perspective. Current approaches have been reviewed with respect to the requirements and their formalisation detailed. It is identified from this review that none of the current designs address queries adequately. Moreover, we could not find a general model which acts as a basis for data warehouse schemas with specific properties. With respect to completeness of a design methodology, existing designs lack discussion on schema properties, refinement, and schema evolution. In other words currently there exists a gap between the model and a final implementation schema. We consider these issues in the following chapters by means of a general user-centric schema design framework. This framework is requirement driven and considers all aspects of a schema design. Later chapters of this thesis will explain the concepts used in the framework as well the generation of the schema itself in detail.
Chapter 3

User-Centric Schema Design Framework

3.1 Introduction

In the previous chapter we have seen various main data warehouse designs in terms of top-down and bottom-up techniques. Top-down designs conceptualised requirements and presented models in the form of starER, MultidimensionalER and cube whereas the bottom-up designs provided transformation algorithms to transform ER model to another data warehouse model. In these designs the main focus is on transformation algorithms rather than conceptualisation. It also stated that real users queries have not been given consideration at all in both top-down and bottom-up designs to offer maximum flexibility to ad-hoc queries. While the
importance of user queries in data warehousing is high, in this chapter we propose a query driven approach. More specifically a design framework is proposed in which queries are given utmost priority.

The proposed framework provides a systematic design approach in such a way that it starts with requirement specification and finally produces a schema from the formalised requirements. That means the framework suggests systematic identification of schema constructs from the requirements; which is lacking in current techniques.

We have summarized in Chapter 2, that except queries, other requirements related to aggregation such as hierarchies and conditions for correct aggregation were already addressed in existing top-down designs. So in this framework existing conceptualised requirements are used as a starting point for the schema design. Since every modeling technique has its own modeling constructs, we avoid the use of any specific model in our framework. On the contrary a generalised formalisation is suggested.

While adopting a query oriented schema design, the following issues need to be addressed.
• users may not have sufficient knowledge about the information that a data
warehouse could supply in which case, users may not be able to provide
queries before hand. This means completeness with respect to queries is
not guaranteed.

• users may not be capable of presenting business structures, business rules
and data dependency because their view is subjective to specific business
units.

The first issue, completeness with respect to queries, is taken into considera-
tion by classifying queries into different types. This classification is called a query
taxonomy and is developed based on the semantic constructs present in a query.
The proposed schema derivation algorithm utilises this taxonomy which ensures
query participation in the design.

To overcome users’ lack of business knowledge, as a first phase of the de-
sign, business requirements are formalised. This formalisation acts as an initial
knowledge, called knowledge base, for the schema design. More precisely, the
knowledge base provides the business rules and measures that are important in a
business. It acts as a reference for the queries and thus validity of queries can be
tested.

Following the above discussion, a query driven schema design framework
needs to consider the business knowledge as well as users queries. So in this chap-
ter before presenting the framework, the formalisation of business knowledge and the query taxonomy are presented. The rest of the chapter is organised as follows.

In Section 3.2 the formalisation of requirements is discussed where the concepts and notations used in this thesis are defined. A generalised conceptual representation is the main focus of this section. In Sections 3.4 and Section 3.5 queries have been studied from a conceptual perspective. This resulted in a conceptual representation for queries and a query taxonomy. Incorporating the proposed conceptual representation and the query taxonomy, a schema design framework is formally presented in Section 3.6.

### 3.2 Formalisation of Requirements

We have discussed the design requirements in Chapter 2 that are important for the schema design. In this chapter, the discussion on requirements collection is restricted assuming we have all the necessary requirements as a result of one of the existing requirement analysis processes such as [RB03], [FJ03], [M.D04]. As the initial step in the design, the business requirements; business measures and business structures, are identified and formalised. For this formalisation a graph theory approach is adopted here. Graph theory has been successfully applied in many areas of computer science such as modeling, query languages, and programming. A few examples to this end are [MG95], [JPL95], [MA90] and [V.N01].
The main advantage of graph theory is that it has only two constructs; nodes and edges. A broad spectrum of concepts can be represented using these constructs. That means, with lesser constructs the requirements can be represented efficiently so that the representation becomes more flexible.

We have seen different representations such as cube, multidimensional ER and dimensional fact model, in Chapter 2 but these representations failed to capture one or more requirements. Our aim is to capture as many requirements as possible with minimum constructs. Graph theory is the tool best suited for this purpose as it avoids the use of any specific graphic notations thus a superior representation, in terms of generality, than the existing ones, can be achieved.

Since business structures are hierarchical, graph is the natural way to represent this. Also a formal graph definition, with a set of nodes and edges, allows to define the necessary design requirements mathematically and eliminates the need for any specific concepts and their notations from ER and object-oriented models. This approach can achieve a notable feature for the framework which is the generality of the framework itself.

In the following subsections, the basic constructs necessary for the proposed framework are explained in detail.
3.2.1 Definition: Measure Type

Earlier in Section 2.3.3 we have defined business measures as quantifiable standards used to evaluate the business performance. We have also mentioned that certain business measures need to be derived from other business measures. In order to capture these requirements we define Measure types.

A Measure type is defined as a class of business objects. Each instance or a business object in a measure type is called a measure. It is assumed that the state of each business object in the class is of interest to the analyst and it can be quantified. A Measure type is denoted by the symbol $M_i$ where $i = 1, 2, ..., n$.

A measure type has a structure which can be written as:

$$< \text{Name}, \text{list of identity attributes}, \text{list of analysis attributes} >$$

where:

- Name is the name of the measure type;
- list of identity attributes are a list of attributes and these attributes identifies each instance of the measure type;
- list of analysis attributes are attributes which are used for quantifying the business objects.

Example from case study: In retail, sales is of interest to the analysts and it is a measure type. That is, sales represents sales occurred at a particular store at a particular time for some products. Then the structure of that measure type is:
< Sales, Stid, Pid, Tid, Amt, Qty > where

Sales is the name of the measure type. Stid, Pid, Tid are identity attributes representing store identifier, product identifier and time identifier respectively. Amt and Qty are the analysis attributes for dollar amount, quantity sold respectively.

**Domain of a Measure type**

The domain of a measure type, $M_i$, is defined as the union of all individual attribute domains.

$dom(M_i) = \bigcup_{k=1}^{x} dom(a_k)$ where each $a_k$ represents either an identity or analysis attribute and $x$ is a natural number.

**3.2.2 Derived Measure Type**

A measure type may be related to any other measure types through its attributes. For example, $M_i$ is related to $M_j$ if they share common identity attributes, and at least one of the analysis attributes of $M_j$ is a function of analysis attributes of $M_i$. If such an $M_j$ exists, then it is a derived measure type and the other measure type $M_i$ is a called basic measure type.

The above definition can be formally represented as follows:

A derived measure type $M_d$ has the same structure as that of a basic measure type. That is

$M_d : < Name, list of identity attributes, list of analysis attributes >$
Where *Name* is the name of the derived measure type. The list of identity attributes is represented as $I_j$ which is a subset constructed from all other identity attributes of the basic measure types from which $M_d$ is derived. $A_k$ represents the list of all analysis attributes of the derived measure type. An analysis attribute $a_i \in A_k$ can be further described as a function of analysis attributes of the basic measure types and it can be formally written as follows.

Let $P$ be the set of all participating measure types and $P_A$ be the set of all participating analysis attributes, then;

$$
\exists a_i \subseteq A_k \text{ and } a_i = \mathcal{F}(P_A) \text{ and } \cap_{j=1}^{y} I_j \neq \emptyset \quad (3.1)
$$

In Equation 3.1, $\mathcal{F}$ is a function which operates from the cross-product of the analysis attribute sets $A_1, \ldots, A_k$. The function may also act as an aggregation function such as *SUM*, *AVE* etc. This function is further described as a composite function which has the functionalities of mapping as well as aggregation functions.

Derived measure type can be explained through the *sales* example. The measure type *Sales* has two analysis attributes *Amt* and *Qty*. Another measure type called *Price* can be derived from *Sales* in such way that the analysis attribute;
\( T\text{Amt} \) in \( Price \) is represented as a function of \( Amt \) and \( Qty \). That is:

\[
\mathcal{F} : (Amt \times Qty) \rightarrow T\text{Amt}; \quad \mathcal{F}(Amt, Qty) = Amt \times Qty
\]  

(3.2)

In Equation 3.2, the function \( \mathcal{F} \) takes values from \( Amt \) and \( Qty \), then applies the aggregation function and maps to \( Total\text{Amt} \).

At this stage it is interesting to present a discussion on derived measure type and derived measures presented in Chapter 2. We have seen in Section 2.5 that most of the existing models do not support derived measures. In current models measures are defined as attributes of a fact and derived measure is one such attribute which is derived from other basic measures of the same fact. Normally derived measures are represented in the fact itself. Our approach is slightly different and the definition allows to capture derived measure types and is general enough to relate a measure type to any number of measure types. In this case if an analysis attribute (which is equivalent to a measure in other approaches), is derived from other analysis attributes that is treated as a new measure type. So the derived measures discussed in existing approaches is equivalent to representing an analysis attribute as a function of other analysis attributes of another measure type. A consideration like this enable us to introduce the concept of derived measure type at the schema level without any difficulty. The derived measure type \( Price \) is an example to this. In existing approaches the analysis attribute \( T\text{Amt} \)
should be in Sales, where as here TAm

is defined as an analysis attribute of a new measure type called Price.

As we mentioned earlier, in existing approaches derived measures are discussed only with respect to a single fact. That is, derived measures are presented as measures derived from basic measures of the same fact. However it is possible that measures can be derived using basic measures of different facts. This case is also taken into consideration in the derived measure type definition and is explained with an example.

Stock-turn is a measure type and is related to two other measure types called sales and purchases. That is stock-turn is defined by the business rule as:

\[
Stock - turn = \frac{Average \ sales \ amount}{Total \ purchase \ amount}
\]

So the two measure types; \(< Sales, Stid, Pid, Tid, Amt, Qty >\) and \(< Purchases, Stid, Pid, Tid, Pamt, Pqty >\) are required for representing Stock-turn.

In Stock-turn an analysis attribute Stamt is represented as a function of two other analysis attributes Amt and Pamt from Sales and purchases respectively. The Stock-turn can be written as:

\(< Stock - turn, Stid, Pid, Tid, Stamt >\)

where \(\mathcal{F} : (Amt \times Pamt) \rightarrow Stamt\) and

\(\mathcal{F}(Amt, Pamt) = Ave(Amt)/Sum(Pamt)\)
The function $F$ takes values from the analysis attributes $Amt$ and $Pamt$ and applies $AVE$ and $SUM$ to $Amt$ and $Pamt$ respectively. Finally $Ave(Amt)/Sum(Pamt)$ is computed and maps to $Stamt$.

### 3.2.3 Similarity between Measure Types

At a very high level, similarity is defined in terms of attribute similarity. That is, a measure type is said to be similar to another measure type if the following conditions are satisfied.

- Both measure types share the same identity attributes and
- they have at least one analysis attribute in common.

We establish similarity between measure types with respect to similarity between its attributes. A function could be defined to identify similarity between attributes and thus check similarity between measure types. These types of functions are called *Similarity functions*.

Another way to look at similarity is in terms of participating attribute domains. It can be written as:

- **First condition**:
  $$\text{Dom}(i_1), \text{Dom}(i_2), ..., \text{Dom}(i_x) \cap \text{Dom}(j_1), \text{Dom}(j_2), ..., \text{Dom}(j_y) \neq \emptyset;$$
  where $x, y$ are natural numbers. Each $i_x$ and $j_y$ represent identity attribute of measure types $M_i, M_j$ respectively.
• Second condition:

\[
\text{Dom}(a_k) \cap \text{Dom}(a_l) \neq \emptyset \text{ where } a_k \in M_i \text{ and } a_l \in M_j \text{ are analysis attributes; } k, l \text{ are natural numbers.}
\]

If the above conditions are true then the measure types \( M_i \) and \( M_j \) are said to be similar. In the Sales example given in Section 3.2.1, two analysis attributes given are: \( \text{Amt} \) and \( \text{Qty} \). Let \( \text{Sales}' \) be another measure type with an analysis attribute \( \text{Amt} \) and all other identity attributes similar to \( \text{Sales} \). Then we say, \( \text{Sales} \) is similar to \( \text{Sales}' \).

Since our definitions are purely conceptual, there is no indication of the type of data involved. So it is difficult to discuss the actual implementation of the similarity test at this stage. Hence we suggested the domain intersection as one possible direction towards the similarity identification.

### 3.2.4 Classification

We have introduced the term classification along with business structures in Chapter 2. Classification is used to construct business structures and typically it is hierarchical in nature. A formal definition of classification is presented in this section.

As we mentioned in Section 2.5.2, formalisation of classification is already proposed in literature and can be found in [C.S01]. Since it offers a graph based formalisation we follow that in our framework and is described here in detail.
Classification is defined as a grouping on a set of objects based on some criterion and the criterion is given by the business. Each group in a classification is called a class and any two classes are related by means of a relation called classification relation. Classification can be explained through a classification schema and it is defined as follows.

**Classification Schema**

A classification schema is defined as a set of distinct classes and a classification relation. It can be represented as a graph with classes as nodes and members of classification relation as edges.

Each class \( c \) represents a set of objects from a domain \( \text{dom}(c) \), which can take values that are valid in the respective applications. Also the domains of any two classes in a given classification schema are pair-wise disjoint. That is:

\[
\text{dom}(c_i) \cap \text{dom}(c_j) = \emptyset
\]

Business structures are typically hierarchical, hence the classification relation is reflexive, transitive and antisymmetric. These properties imply a partial order among the classes and the partial ordering forms a hierarchy.
A classification schema $CS$ is defined as a tuple $(C, R)$ with

- $C = \{c_1, \ldots, c_k\}$ where $c_1, \ldots, c_k$ are distinct classes of classification $C$.

- $R \subseteq C \times C$ defines the classification relation between different classes and it has the following properties:
  - reflexive: $\forall c_i \in C$, $(c_i, c_i) \in R$
  - transitive: $\forall c_i, c_j, c_k \in C$, $(c_i, c_j) \in R \text{ and } (c_j, c_k) \in R \Rightarrow (c_i, c_k) \in R \text{ and } i \neq j \neq k$
  - antisymmetry: $\forall c_i, c_j \in C$, $(c_i, c_j) \in R \Rightarrow (c_j, c_i) \notin R \text{ and } i \neq j$

An example of a classification is shown in figure 3.1 based on two classes customer and store. The nodes in the classification schema graph represent the classes and the edges represent members of classification relation. For better visualization, reflexive and transitive edges are omitted from figure 3.1.

The definition of classification relation is general and it does not impose any constraint on the relation between two classes in a classification. That means, a lower level class can participate with any class above it and thus the hierarchy need not be balanced. Since there is no restriction on class participation, the definition allows multiple/shared paths. Also, classification schema is a direct acyclic graph which allows to capture all requirements related to business structures such as multiple and unbalanced hierarchies, shared paths. However the above schema
definition only specifies which classes are related. In order to achieve the correct aggregation condition, discussed in Section 2.3.5, the mapping between two classes needs to be defined. In this circumstance, defining instances of classification schema is important, which further defines the relation between two classes.

The instances of the classification schema can be defined by means of a set of grouping functions. A grouping function maps one member of a lower class (child class) to another member in a higher class (parent class).
That is, a grouping function between classes $c_1$ and $c_2$ can be defined as:

$$group_{c_1,c_2} : dom(c_1) \rightarrow dom(c_2).$$

The following figure 3.2 shows an example grouping function. As shown in this figure the class stores, has members Malvern, Chadstone, Burke st, Newcastle. Similarly the class City has members Melbourne, Sydney. The grouping function called $group_{stores,city}$ shows the mapping between the classes stores and city and is given in figure 3.2.

![Figure 3.2: A grouping function](image)

**Fundamental Class**

A class node in a classification schema is a called a *fundamental class* ($fc$), when there is no incoming classification relation to it. This is a class with the lowest level of granularity in the classification schema. That is, it is not possible to represent a class with lower granularity than a fundamental class and there always exists at least one fundamental class in a classification schema.
A classification schema can have more than one fundamental class depending upon the classification given by the business people. For example, in the classification schema shown in figure 3.1, stores are classified according to geographical location. Similar criterion is used for customers as well. Then, in that schema store and customers are two fundamental classes.

By definition classification schema allows shared paths. Hence it is important to characterise the path with respect to the participating fundamental classes otherwise aggregation along the path will produce incorrect results. For instance, the class region in the example classification schema can be derived based on both store and customer. So it is necessary to distinguish in the schema which fundamental class is used for aggregation. Similarly, completeness is another requirement for aggregation. As in other existing techniques, for example [AKS01], [TC99], in order to satisfy the completeness condition for summarisability (cf: Section 2.3.5), an artificial node called All is required in the schema. The fundamental classes and the artificial nodes are introduced in the classification schema to get correct aggregation of data which results in specialised graphs called classification lattices.

Classification Lattice

In a classification schema \((C,R)\), classification lattice\((CL)\) is defined as \((C',R')\) [given a fundamental class \(c \in C\)] where:
\[ C' = \{ x | x \in C \land c \leq x \in R \} \cup \{ \text{All}_c \} \] (3.3)

where \( \leq \) shows the partial ordering

\[ R' = (R \cap (C' \times C')) \cup \{(y, \text{All}_c) | y \in C'\} \] (3.4)

Mathematically a direct acyclic graph is said to be a lattice when there is a partial ordering among the participating nodes and there is a guaranteed lower level node called greatest lower bound and a highest level called least upper bound in the graph.

These conditions will be met by the introduction of a fundamental class and the artificial node. That is, \((C', R')\) form a lattice because, there exists a partial order among the classes of the classification schema and preserved those conditions while defining \(R'\). Moreover there is a greatest lower bound, which is the fundamental class \(c\) and always there exists a fundamental class. The least upper bound in this case is \(\text{All}_c\) which is the artificial node added in the schema in order to satisfy the summarizability condition. Generally classification lattices are subgraphs that exist in a classification schema defined with respect to fundamental classes. A detailed mathematical proof of lattice structure based on the classification schema definition is included in Appendix A.
Two classification lattices can be defined with respect to the classification schema shown in figure 3.1; one with respect to customer and another with respect to store. In the following figure 3.3, the shaded area shows one lattice based on store. Similarly another lattice is there with customer as the lower bound and All\(_c\) as the greatest bound. The dotted arrow indicates that any class node can directly relate to the root node if it cannot relate to any other existing classes.

![An example classification lattice](image)

**Figure 3.3: An example classification lattice**

Since there exists different classification lattices in a classification schema, classes need specific notations to identify them. Here we use class notation with
respect to its fundamental class. That is, a class $c_{ij}$ in a classification schema represents the $j$th class of $i$th fundamental class $fc$. Meanwhile in the case of shared paths, the class notation will follow either one of its fundamental classes. For example, assume fundamental classes $fc_i$ and $fc_j$ share the same classification. Then the classes in that classification can be represented either by $c_{ip}, c_{iq}, ..., All_e$ or $c_{jp}, c_{jq}, ..., All_e$.

**Attributes**

We have introduced heterogenous levels in Section 2.3.4 as levels having heterogenous attributes for their respective members. In this formalisation we allow heterogeneity by representing the attributes as separate nodes in the classification. That is, all the attributes, that are not part of the classification, are taken out from that class node and treated as independent nodes. These attributes are called *descriptive attributes* and a descriptive attribute is denoted by $Da_k$.

The attributes that are used for classification are called *classification attributes*. In other words classification attributes are used for constructing a higher class in a hierarchy.

The descriptive attributes in this formalisation represent all the attributes, whether it is homogeneous or heterogenous with respect to a class. However, as shown in [YL03], a different notation (nodes are indicated by dashed lines), has been used in order to represent attributes that are heterogenous to a particular class. This
distinction establishes the idea that heterogeneous attributes are applicable only to certain members of a class and queries based on these attributes are valid with respect to those members.

In figure 3.4 the class \textit{product} is shown with its descriptive attributes. Among those attributes, \textit{colour} and \textit{weight} are homogenous with respect to product which means they are applicable to all products in the class. Where as \textit{clothing size} is applicable only to certain products such as \textit{clothes} and hence it is represented as a node with dashed lines.

**Special Classes**

Special classes are classes that are not used in any classification, but are directly related to a measure type. These special classes may be present in the source data if so can be included in the formalisation. For example, if \textit{Order} be a measure type, it can have a class such as \textit{order date}.  

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{descriptive_attributes.png}
\caption{Representation of descriptive attributes}
\end{figure}
3.3 The Knowledge Base Graph

While proposing a query oriented schema design earlier in Section 3.1, we have mentioned that user’s business knowledge is subjective and they may not be able to provide all the necessary information for a schema design. To that extent, the business knowledge required for a schema should be introduced in the design framework. Otherwise users have to formally indicate all the underlying semantic relationships as well as business rules through queries which means formal query language representation is expected instead of informal natural language queries. This scenario places users in a difficult position. To avoid such consequences, in our framework a knowledge base is introduced which captures the underlying business knowledge. The knowledge base represents formalised business knowledge which mainly includes business measures and business structures and acts as a reference for the queries. The knowledge base helps to test the validity of query constructs and the associated relationships in a query.

We have seen a graph oriented approach in this regard in Section 3.2. The graph theory approach forms the basis of our design which is appropriate for a generalised schema. That means, the graph is general enough to translate to other existing models and thereby achieve an implementation independent schema at the conceptual level. A more detailed discussion on this will be presented in Chapter 5.
The knowledge base graph \((G_k)\), in short KB graph, is formally defined as a graph with a set of nodes and edges. The nodes and edges have varied semantics depending upon the concept or relationship they represent. The nodes in the graph can be a measure type, fundamental class, class and/or descriptive attributes. Similarly the edges show the relationship that exits between the two participating nodes. As we know, the nodes are of different types; hence the edges need to be defined according to the nodes involved. This characterisation of edges according to varied semantics is later useful in the schema synthesis as this helps to traverse along the graph structure.

The edges of KB graph are defined as follows.

- **Classification relation**: We have discussed the relevance of business structure in a data warehousing environment, earlier in Chapter 2. In Section 3.2.4, the formalisation of business structure is also presented. Based on that, in the knowledge base, an edge between any two classes represents a member of the classification relation. At the instance level, this can be replaced by a grouping function because the instances are defined in terms of grouping functions. During the schema generation, the classification relation helps to derive a higher class from a lower class so that aggregation along a path can be supported.

- **Relation between a measure type and a fundamental class**: A measure type
and a fundamental class are related through the members present in them. That is, one or more classification attributes of the objects from a fundamental class are related to one or more objects in a measure type. The relation can be defined as:

\[ \alpha_{ij} = \{(f, m)\} \text{ where } f \in fc_i \text{ and } m \in M_j \]

\( f \) represents classification attributes of an object in \( fc_i \) and \( m \) represents an instance of \( M_j \). Note that each \( f \) can be considered as a set of classification attributes.

For example, let \( Product \) be a fundamental class with members \( \{p_1, p_2, ..., p_n\} \) and \( Sales \) be the measure type with instances

\( \{< p_1, s_1, 10, 4 >, < p_2, s_2, 15, 3 >, ..., < p_x, s_y, 25, 5 >\} \)

then the relation

\[ \alpha_{productsales} = \{(p_1, < P_1, s_1, 10, 4 >), (p_2, < p_2, s_2, 15, 3 >), \ldots\} \]

The relation is written as a set of ordered pairs with values from the fundamental class and measure type respectively. This type of relation integrates the two concepts; measure types and classification schema, which in turn establishes the required multidimensional nature for the schema.

Relation between measure type and fundamental class is represented by bold dotted directed arrows in our graphical representation.

- **Relation between Measure types (Derived measure type):** We have defined
the relationship between measure types, earlier in Section 3.2.2, as a function from one measure type to other measure types. A directed edge between measure type nodes indicates that the target measure type can be derived from the other measure types.

If the measure in a query is derived, the query may not provide any indication about the participating measures. In that context the definition of relation between two measures is important for schema generation.

- **Relation between a class and an attribute**: For each fundamental class \(fc_i\) and each descriptive attribute \(Da_j\), a many-to-one relationship is defined as:

\[ \beta': A \subseteq fc_i \rightarrow B \subseteq Da_j \]

Similarly for each class \(c_{ij}\) and each descriptive attribute \(Da_k\) a many-to-one relationship is defined as: \( \beta'': C \subseteq c_{ij} \rightarrow D \subseteq Da_k \)

\(\beta'\) and \(\beta''\) are represented by thin dotted directed arrows.

As in the case of derived measure types, if an attribute is present in a query, it does not necessarily provide the associated class. In order to support this type of query the relation between a class and attribute is required in the knowledge base.

- **Relation between fundamental classes**: Earlier we have seen that a classification relation relates two classes in a classification schema. However this relation does not relate two fundamental classes. Here we introduce such
Fundamental classes in the classification schema are identified from the operational schema according to the given classifications. In that respect the relation between these classes also need to be identified from the operational schema. Generally the relation exists between two fundamental classes is many-to-many. A thin double edged arrow is used for the representation.

The relation is defined as:

$$\gamma_{kl} : D \subseteq f_{c_k} \rightarrow E \subseteq f_{c_l} \text{ and } D \cap E = \emptyset$$

This relation connects two classification lattices through a measure type and allows us to perform an operation between two classes at the schema level.

Using the definitions, the KB graph can be formally described as:

$$G_K = (N_k, E_k)$$

where

$$N_k = \{M_1, M_2, \ldots, M_n\} \cup \{c_{11}, c_{12}, \ldots, c_{pq}\} \cup \{f_{c_1}, f_{c_2}, \ldots, f_{c_p}\} \cup \{D_{a_1}, D_{a_2}, \ldots, D_{a_z}\} \text{ and}$$

$$E_k = \{\alpha_{i1}, \ldots, \alpha_{ij}\} \cup \{\beta\} \cup \{\beta'\} \cup R \cup \{\gamma_{11}, \ldots, \gamma_{st}\}$$

An example KB graph with respect to a measure type $M_1$ is shown in figure 3.5. Three classification lattices based on fundamental classes $f_{c_1}, f_{c_2}$ and $f_{c_3}$ are also shown separately in this figure. By relating these lattices to the measure type, through the relation $\alpha_{ij}$ (which is indicated by bold, dashed, directed ar-
rows), the knowledge base graph is constructed. The descriptive attributes of the fundamental class; $f c_2$, are shown as $D a_1$ and $D a_2$.

Figure 3.5: The KB Graph: an example
The formal symbols, used in figure 3.5, are replaced by equivalent names from the case study in figure 3.6. In the case study, there is a measure type called \textit{sales}. Also there are classification lattices with respect to \textit{product}, \textit{store}, \textit{day} and \textit{customer}. These classification lattices are attached to \textit{sales} using the measure-fundamental class relations and the knowledge base is constructed accordingly. The presence of transitive edges in the classification schema is indicated by the edge; \textit{(store, region)}. It may also be possible that another edge \textit{(store, region)} in the knowledge base shows the mapping between the two nodes so distinction between transitive edges and members of the classification relation is required during the implementation of the graph.

Figure 3.6: A KB graph: from Case Study
The following Table 3.1 and figure 3.7 detail the notations used in the graph structure.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_j$</td>
<td>$j$th Measure Type</td>
</tr>
<tr>
<td>$f_{c_i}$</td>
<td>$i$th Fundamental class</td>
</tr>
<tr>
<td>$c_{ik}$</td>
<td>$k$th Class, of $i$th Fundamental class</td>
</tr>
<tr>
<td>$D_{an}$</td>
<td>$m$th Descriptive Attribute</td>
</tr>
</tbody>
</table>

Table 3.1: Nodes in KB graph

<table>
<thead>
<tr>
<th>Edges</th>
<th>Notations</th>
</tr>
</thead>
<tbody>
<tr>
<td>----&gt;</td>
<td>Member of classification relation</td>
</tr>
<tr>
<td>----&gt;</td>
<td>Relation between class/ descriptive attribute</td>
</tr>
<tr>
<td>----&gt;</td>
<td>Relation between fundamental class and measure type</td>
</tr>
<tr>
<td>----&gt;</td>
<td>Relation between fundamental classes</td>
</tr>
</tbody>
</table>

Figure 3.7: Edges in KB graph

In this section, by integrating the formal concepts and their relationships, a graphical representation has been developed. This graph is called the knowledge base graph and is used as a starting point for the data warehouse schema generation process. It is assumed that the initial knowledge required for the schema is present in the knowledge base. This knowledge, along with the queries collected
from the users are used to generate a query oriented schema. Readers could find a
detailed description of this in Chapter 4. In the next sections, the other necessary
component for the proposed framework; queries, is studied in detail.

3.4 Query Representation

When considering queries in a schema design process, we can express queries
either in natural language form or in any other query language; for example by
extending SQL to be suitable for data warehouses. The main issue associated with
natural language queries is the processing of the query itself. Natural language
query processing is challenging and it complicates the schema design process.
Another option is to express queries in a standard query format. However this is
a restriction on representation and we do not intend to restrict the design only to
certain query languages. One possible solution to tackle this issue is to propose a
formal query representation which is suitable for our schema design framework.

As mentioned in Section 3.1, to meet the generality property, our approach
employs graphs. Query representation is not exempted from this rule and each
query is treated as a graph in this design. However, specific treatment on the
graph structure is required due to the inherent properties of queries. The proper-
ties are as follows.
• Generally data warehouse queries rely on measure types. This means measure type is the root of the graph.

• Cyclic operations are highly unlikely, so cycles are eliminated from the representation.

• Direction of the edges in the graph is irrelevant as the semantic relationships are already defined in the knowledge base.

Because of the above mentioned properties, a natural language query is treated as a tree called a query tree. In a query tree the root node is always a measure type and successive nodes can be a class, fundamental class or a descriptive attribute. Additionally, a tree is allowed to have only one measure type. Then the leaf nodes in the tree are always nodes other than a measure type.

An edge in a query tree represents a function called requirement function. The requirement function \( R_q \) can be written as:

\[ R_q \subset \{ f_1, f_2, ..., f_i \} \]

where each \( f_i \) can be any valid function including statistical functions such as \( SUM, AVE, MIN, MAX \) and functions that apply constraints. They may also represent combinations of more elementary functions or functions it is meaningful to compute. These functions may be used to represent aggregate operations as well as offering a generalised representation of the operators that we have discussed in Chapter 2. Since a query tree is a conceptual representation, any specification
of operators is purposely omitted from the tree definition. Later, at the physical design level these functions can be translated into operators.

Using the notations previously detailed in Section 3.3, a query tree $G_{q_i}$ is formally defined as follows.

$$G_{q_i} = (N_{q_i}, E_{q_i})$$

where $N_{q_i}$ is a set of nodes and $E_{q_i}$ is a set of edges.

$$N_{q_i} = \{M_j \cup f c_j \cup c k_l \cup D a_i\}$$ and

$$E_{q_i} \subset \{f_1, f_2, \ldots, f_n\}$$

Note that the tree definition does not enforce any order on the class nodes and attribute nodes. This leads to multiple representations for a given natural language query. In general, a query will have more than one possible representation depending upon the classes or descriptive attributes in a query. An example query tree from the case study can be show the sales by region by year. In this query, sales is identified as a measure type; region and year are classes. Then the root node of the query tree will be sales followed by region or year or year and region. The possible representations with root being the measure node are shown in figure 3.8.

The tree representation does not limit queries having sub-queries in it. For example a query like, list the stores where the sales exceeds average sales can be represented in the same way as in the above figure. Even though the query under consideration here is nested, it is based on the same measure type and class. That means the semantic constructs used in the representation is enough to handle the
query at the conceptual level. Computation of these types of queries are purely an implementation issue so we leave that out from our discussion.

### 3.5 Query Taxonomy

Following an informal query classification in Chapter 2, in this chapter we revisit queries and present a taxonomy that can be used in schema design. This taxonomy is proposed as an integral part of our design framework and it allows us to classify
general data warehouse queries into different classes. These query classes serve as the basis for further schema generation processes (which we will see in Chapter 4) and ensure the completeness of the synthesis. Thus a query oriented schema is offered at the end of the design.

From the formal representation introduced in Section 3.4, a query can be seen as a collection of semantic constructs and functions. This suggests a possible classification based on the existing constructs, functions or both. From a conceptual perspective, the functions associated with the queries are not taken into consideration in the first instance for the classification. However, it is assumed that a set of functions are associated with each group and which is going to be applicable to all groups. This allows us to consider the functions associated with a query later in schema synthesis.

As seen in the figure, the taxonomy is developed based on the semantic concepts of measure type and classification lattice. The measure type in a query can be basic or derived which creates two classes of queries; queries based on basic measure type and derived measure types.

The classes/attributes in a query are from one or more classification lattices. This property leads to two other classes of queries which are queries with respect to a single classification lattice and multiple classification lattices. These types of queries are further classified according to the presence of class/attributes or both. If a query shows any special character related to the measure type or classes or
attributes present in them, they are considered differently and termed as *exception queries*.

Taking all these factors into account a formal taxonomy is detailed here. Common cases, for instance, queries based on a single measure type and single classification are considered first and then multiple measure types and multiple classifications are discussed. After that categories with special considerations are described. Example queries are presented from the case study to show the given taxonomy.
3.5.1 Single Measure Type-Single Classification ($Type SS$)

$Type SS$ is a common class of queries in data warehouse. In this type of query there is a measure type and a fundamental class or classes from a single classification lattice. This type can further be divided into three sub-categories based on presence of class and descriptive attributes in them.

**Queries with Classes ($Type SS_1$)**

A $Type SS_1$ query has one measure type and at least one class. These are the most common form of queries in a data warehouse and are generally known as roll-up and drill down queries. $Type SS_1$ queries define an aggregation of a measure type terms of one or more classes.

Typical examples are:

- Sales by Region

- Total sales in quarter 1 of year 1997

In the first example, region is a class and sales is a measure type. This is a roll-up query with respect to store. Similarly the second one is another roll-up query based on day and there are two classes, quarter and year specified in the query.

Examples of drill-down queries with respect to sales by region are: sales by state and sales by city. All roll-up/drill-down queries, with respect to each defined
classification lattice, are members of this category and each roll-up/ drill-down operation is treated as an individual query.

**Queries with Classes and Attributes** \( (TypeSS_2) \)

Queries with a measure type, a fundamental class or classes and descriptive attributes come in this category. The descriptive attributes present in the queries can be further explained in terms of their parents. That is, it need not be necessary that the fundamental class or class in the query be the parent of the given descriptive attributes. It is possible that a query may contain descriptive attributes from any classes in that given classification lattice and those classes may not be present in the query. Some examples are:

- Sales by product by package size
- Purchases in southern region by managers
- Sales of washers from category by water usage

The classes *product* and *region* in the first two queries are associated with their own descriptive attributes *package size*, *manager* respectively. While in the third query the attribute *water usage* is not directly related to *washers*, where *washers* is an instance of the class *category*. The descriptive attribute *water usage* is related to the fundamental class *products*. 

126
Different from *TypeSS*$_1$, these types of queries require a result based on members of a class with respect to certain attributes. This means an operation has to be performed with respect to specified descriptive attributes.

**Queries with Attribute (TypeSS$_3$)**

It is possible that users may present queries with descriptive attributes and a measure type. They may not specify any class at all in the query. Queries satisfying the above criterion fall into this particular category.

While discussing the formalisation of requirements in Section 3.2, we have seen that descriptive attributes are always attached to a class. Taking this into account, in order to answer queries with attributes, corresponding classes are required. In this respect these types of queries can be seen as a special case of the category, queries with classes and attributes; *TypeSS*$_2$. By introducing this category in the taxonomy we have tried to accommodate those users, they are not able to provide complete information for answering queries.

A few examples of this query type are:

- Sales by shop type
- Sales by age.

In these example queries, *shop type* and *age* are descriptive attributes and are associated with the classes *store* and *customer* respectively.
3.5.2 Single Measure Type-Multiple Classification (TypeSM)

In TypeSM queries the classes are not from a single classification lattice. The subcategories are similar to TypeSS; the only difference is that classes and attributes are from multiple classification lattices.

Queries with Multiple Classes (TypeSM₁)

In this type of query there will be a measure type and two or more classes. These types of queries are common data warehouse queries since users are interested in analysing data with respect to various classifications. In query optimisation techniques these queries are normally called join queries. Example of multiple class queries are given below.

- Sales by region by year
- Sales by week by product

In the first query, sales is requested based on region and year, which are the classes. The class region is from the classification lattice store and year is based on day. In the case of the second query, product and week are the classes and they belong to two different lattices.
Queries with Multiple Classes and Attributes ($TypeSM_2$)

Similar to $TypeSS_2$ queries discussed in Section 3.5.1, in this case also measure type, classes and descriptive attributes are present in the query. The attributes in the query may or may not be local to the specified classes in the query. Sometimes the descriptive attribute may be from a different classification lattice itself. Like $TypeSS_2$, in this case also users like to perform operations with respect to descriptive attributes. Examples are:

- Sales of home brand product by shop type,
- Average sales by supplier by week.

In the first query, the descriptive attribute is shop type and it is directly related to the class store. The other class present in the query is brand which is based on the classification lattice products. That is, the attribute and the classes are from two different classifications lattices. Similarly the descriptive attribute supplier from the other query is from the lattice products and the class week is from the classification lattice Day.

Queries with Multiple Attributes ($TypeSM_3$)

$TypeSM_3$ queries have a measure type and descriptive attributes in them. The descriptive attributes are from classes of different classification lattices. As in the
case of $TypeSS_3$, the respective classes are required to answer these queries and is a special case of $TypeSM_2$. Examples of such queries are:

- Sales by clothing size by age,

- Sales by retail shop by supplier.

The descriptive attributes present in the above queries are size, age, shop type and validity period. In the case of the first query the attributes clothing size and age are not from the same classification lattice; these attributes are related to product and customer respectively. Similarly retail store is an instance of shop type which is related to store and the other descriptive supplier is related to product.

### 3.5.3 Multiple Measure Type ($TypeMM$)

In all the above discussed categories, the associated measure type is assumed to be a basic measure type. The user may not clearly specify whether the measure type is a basic or derived. We recommend the application of business knowledge in this case in order to identify the measure type and categorize the queries accordingly.

If the measure type in the query is not a basic and is derived from other measure types, those queries come into this category. For instance if a measure type such as stock-turn is present in a query, then it comes under the multiple-measure category since it is related to other measure types sales and purchases.
The sub-categories associated with this type of query can be done in a similar manner as in the case of single measure type. So all the sub-categories with respect to single measure type are applicable in this case and these are:

- **Multiple measure-Single classification** (*TypeMS*)
  - queries with classes *TypeMS*$_1$
  - queries with classes and attributes *TypeMS*$_2$
  - queries with attributes *TypeMS*$_3$

- **Multiple measure-Multiple classification** (*TypeMM* )
  - queries with multiple classes *TypeMM*$_1$
  - queries with multiple classes and attributes *TypeMM*$_2$
  - queries with multiple attributes *TypeMM*$_3$

The participation of classes and/or attributes in these categories are similar to the subclassifications of *TypeSS* and *TypeSM* queries. So redundant discussions are omitted.

### 3.5.4 Query with only one Measure Type (Type M)

In this case we consider the simplest form of a query, that is, a query with only one construct namely - a measure type. A query with a class or attribute is meaningless in a data warehouse context so we avoid such cases. Queries with only one
measure type are trivial but the user is not restricted from presenting the query in his/her own way. A user can ask for a measure type and those queries come in this group. Since identity attributes are associated with a measure type which in turn directly related to fundamental classes, these queries can be treated as a special case of either $TypeSS_1$ or $TypeSM_1$. Examples $TypeM$ queries are:

- Show the average sales;
- List the items sold.

### 3.5.5 Query with Specialized Class (Type S)

While defining the KB graph in Section 3.2, we have mentioned that there are classes which are not part of any classification and are directly related to measure types. Queries that contain this special type of class and measure type come under this category. The measure type can be a basic or derived measure type in this case. These queries are similar to queries with a measure type and a fundamental class as in both cases the classes do have a direct relationship with the measure type.

### 3.5.6 Query with Exception (Type E)

Queries with exceptions are discussed in this category. All the queries presented previously follow the fact that the semantics of the constructs in the query and the business are the same. For example, constructs such as a measure type or class
possess the same meaning both in a query and in the business. But this is not true always, especially in the case of a measure type. According to the definition of a measure type, any object that is of interest to the analyst can be considered as a measure type. Following the above definition it need not be necessary that the measure type in the query be a measure type according to the business. For instance, a measure type in the query can be an attribute or a class.

Queries having such exceptions are grouped together and called *queries with exception*. This group can again be subdivided into two depending on the measure type in the query. They are as follows.

**Attribute as Measure type** ($TypeE_A$)

In this case, the measure type in the query can be a descriptive attribute according to the application. In order to answer these queries, computation based on the corresponding descriptive attribute is required. Since the concepts measure types and attributes have specific meaning, queries like this need to be treated differently compared to $TypeSS$ and $TypeMM$. A few examples of such queries are:

- Average age of customers those who did a transaction for more than $100
- Total tax paid during the year 2000
In the above queries, *age and tax* are requested by the user; so according to the query those are the measure types. But *age and tax* are attributes according to the business knowledge.

**Class as Measure type** (*TypeE_c*)

Similar to the above discussed type (*TypeE_A*), in this case the measure type in the query will be a class/fundamental class. Common ranking queries, for instance, a query such as *best selling product during last week*, can be considered in this group. Since the ranking is associated with a measure type these queries can be supported in the same way as *TypeSS_1*.

Here we presented a taxonomy of data warehouse queries on the basis of the semantic concepts that we have proposed in the context of our design. Different users perspectives on information retrieval have been considered whilst addressing the queries. The proposed taxonomy is very formal compared to the existing informal query discussion that we have presented in Chapter 2. Generally we could say that the taxonomy is complete with respect to common queries that need to be supported through a data warehouse. Normal queries fit into one or more classes in the taxonomy.
3.6 The User-Centered Schema Design Framework

We have seen a classification of general data warehouse queries and formalisation of business requirements in the previous sections. In this section by integrating the taxonomy and the formalisation, a formal schema design framework is presented.

In Chapter 2, we have seen that the main issue associated with current designs is the lack of consideration of queries in schema design. As stated in Section 2.4, data warehouse queries have inherent characteristics such as multidimensionality and aggregation. Queries provide the type of aggregations, measures and dimensions required in a schema. This proposed framework utilises these information from queries and thus decides what should be included in the schema. Even though queries in data warehouses have specific characteristics, they differ in constructs and operators present in them. In that respect the classification suggested in Section 3.5 can be used to distinguish different queries that come across a data warehouse.

Due to the lack of business knowledge users’ may not be able to construct queries accurately. Moreover in order to support queries, underlying relationships that exist in a business should be known. To address these issues a knowledge base is incorporated in the framework. This knowledge base serves as a means to check the validity of queries.
Validation of query refers to a process by which the query constructs as well as the correctness of the user defined operations are tested with respect to a particular application.

Following the above discussion, the conceptual framework is shown in figure 3.10.

![User-Centered Schema Design Framework](image)

Figure 3.10: The user-centered schema design framework

Like any traditional conceptual design, [BCN92], this framework also starts with requirements specification. During this step, the important design requirements that we have discussed in Chapter 2 are collected. This includes the business measures, business structures, source and external data structures (if any) and users queries. Since queries are given emphasis in this framework, they are treated separately. The requirements other than queries are formalised during the
next step. This graph based formalisation is labeled as knowledge base in the figure. The knowledge base is considered as the initial knowledge required for schema design.

As stated earlier in this section, a knowledge base is necessary in the framework to provide the business knowledge required to support queries. In the absence of a knowledge base, the user has to be aware of all the existing relationships to formally represent those through queries. To eliminate such a scenario from the design as well as allow the users to present queries in an informal manner, knowledge base is introduced in our approach. The two steps; requirements gathering and creation of knowledge base, together constitute the conceptualisation phase of this framework.

The next phase after conceptualisation is the schema derivation phase. The main steps involved in this phase are:

- create an intermediate schema;
- develop a data warehouse schema from intermediate schema.

The schema derivation is a two-level approach which allows a provision for alternate data warehouse schemas. We have discussed a traditional conceptual design, [Jr79] (cf: Chapter 2), which recommends a two-level approach in schema design. Motivated from that design our framework provides an information model which is the intermediate schema. This schema is more query specific and thus the user
needs are established. In the absence of an intermediate schema the design be-
comes arbitrary like the current designs and the need of additional post design
 technique such as view selection will be required to implement the schema. Var-
ious designs that we have discussed in Chapter 2 provide only a single schema
whereas in our framework the intermediate schema allows different data ware-
house schemas. Moreover query based implementation models are possible. A
more detailed discussion on this is presented in Chapter 5.

The second phase, schema derivation starts with an intermediate schema gen-
eration. The intermediate schema is also a graphical representation where all the
user queries are captured. This means, the queries collected during the require-
ments specification are used at this level. To do so the natural language queries
are transformed as query trees. We have seen query trees in Section 3.4, as a con-
ceptual representation of queries. This representation is useful for intermediate
schema generation. More specifically, each query tree corresponding to a natural
language query is identified in the knowledge base. An algorithm called inter-
mediate schema algorithm is suggested for this purpose. The algorithm takes the
knowledge base graph and a set of query trees as input and generates another graph
representation which is the intermediate schema. This algorithm is described in
detail in Chapter 4.
The intermediate schema acts as a common platform for various data warehouse schemas. This schema is general which subsumes existing models. We show the mapping of intermediate schema to other models in Chapter 5.

Since queries are indicated in the graph, it is easier to identify queries in the intermediate schema. This leads to one possible final schema, which is a minimal schema that support those initial queries. Derivation of such a schema is the final step in the framework which means derive a data warehouse schema from the intermediate schema. The derivation can be seen as graph reduction process and a reduction algorithm is proposed in this regard. The final schema derivation is further described as an optimisation at the conceptual level. By addressing optimisation this framework provides a practical approach to schema design. The final schema is directly implementable and issues related query support can be avoided.

### 3.7 Summary

In this chapter, we have introduced a new framework for conceptual schema design. The proposed framework subsumes the existing data models and provides a generalised and practical approach to schema design.

The framework consists of two design stages. The first stage is the conceptualisation of business requirements and the second stage is the schema derivation.
stage. Conceptualised business requirements are used as an initial knowledge for schema derivation. Two algorithms; an intermediate schema algorithm and a reduction algorithm were suggested in the context of schema generation. Another significant contribution, other than the proposed framework, is the query taxonomy and query representation. To the best of our knowledge this is the first approach which studied data warehouse queries from a conceptual perspective.
Chapter 4

Translation of Queries for Schema Synthesis

4.1 Introduction

The relevance of queries in data warehouse design lies in the fact that they define the schema that needs to be stored. We have discussed various designs in Chapter 2 and concluded that neither of these designs fully incorporate queries. To that extent an attempt has been made to analyse queries conceptually resulting in a query representation and a taxonomy (cf:Chapter 3). The main motivation behind this analysis was to investigate the usefulness of queries in schema design. In this chapter different query types; discussed in Chapter 3, are revisited and a schema is developed using the queries.
The current designs address only one schema with a view to offering maximum flexibility to ad-hoc queries. The addition of the intermediate schema in the proposed framework overcomes this limitation by offering an opportunity to derive various data warehouse schemas for given properties. If such a derivation is not required, the schema can be translated to a desired model such as cube or star. In particular, query specific cube/star selection is possible with the intermediate schema. We will see this in detail in Chapter 5.

The KB graph provides only the semantic relations in a business however, along with the semantic relations, the intermediate schema carries information about the types of queries that are asked, required query constructs and the operations. Since it carries the information about the required queries this schema acts as a platform for synthesising different data warehouse schemas. In other words this schema can be transformed to get schemas with specific properties.

The intermediate schema generation is considered in detail in this chapter and is presented in terms of query mapping. Application of the queries at this stage ensures query participation in schema design. The chapter is organised in the following manner.

In Section 4.2 the intermediate schema is discussed in detail along with an algorithm for its generation. The algorithm maps different query trees to the KB graph so that the intermediate schema can be generated. The process associated with the mapping is detailed in Section 4.3. Prior to the actual mapping, classi-
fication of query trees is required. This could be found in Section 4.3.1. Since two graphs; KB graph and query tree, are involved in the mapping process, the similarity that exists between nodes in these graphs needs to be tested. A special similarity function is suggested in Section 4.3.2 for this purpose. Later, Section 4.4, Section 4.5, and Section 4.6, details the mapping procedure associated with different query types. Finally, the intermediate schema of our running example is presented in Section 4.7 followed by the chapter summary in Section 4.8.

4.2 Intermediate Schema Using Queries

The intermediate schema and its purpose was defined earlier in Section 3.6. A more detailed discussion on this regard is presented in this section.

As mentioned in Chapter 3, the intermediate schema is an information model in this framework. Therefore various data warehouse schemas with specific properties are possible. The intermediate schema thus avoids the possibility of a direct optimised schema, which allows only one schema at the data warehouse implementation level. The involvement of queries at this level gives an understanding of queries that need to be supported through a schema hence a query oriented schema can be provided at the final stage. Unlike other approaches, in this design, real users queries are considered whereas in existing approaches a schema is designed giving emphasis on roll-up/drill-down operations ([MDS98b] [AKS01]).
The intermediate schema \((G'_k)\) is also a graph similar to the KB graph. The nodes in the graph have the same semantics as that of a knowledge base. However, other than dependency, the edges carry requirements functions from the query trees. More specifically, an edge in an intermediate schema shows the dependency between two nodes as well as user defined operations, if any. That is, if a user specifies an operation between two given nodes, through a query tree, this will be captured in the intermediate schema. This means query semantics is also taken into consideration for the schema design, which is a positive aspect of this framework. In addition to data requirements, user defined operations are also accountable for schema design.

Considering the nature of the intermediate schema, it satisfies the following properties.

- Intermediate schema is a graph with directed edges and the direction of an edge indicates a semantic relation or a user defined operation.

- A query, that is considered during the design, is indicated as a path in the graph (a more detailed explanation in this regard could be found in Section 4.3). This means, the required constructs for a query as well as its type are easily identifiable from the intermediate schema.

- More than one path might be possible for a query in the schema.
• Queries that require different kinds of aggregations (queries with specialisation), are differentiated by means of special edges.

For the intermediate schema generation an algorithm is proposed which requires the KB graph and query trees as inputs. Hence query trees are created from the natural language queries; collected during the requirement specification. The transformation of natural language queries to query trees follows the definition given in Section 3.4. That is, the root of a query tree is a measure type node and the other nodes are fundamental classes / classes and /or descriptive attributes.

The algorithm is designed in such a way that a path corresponding to a query tree is identified in the knowledge base. A path in the knowledge base is equivalent to a query tree if,

• the start node and end node are the same;

• all the intermediate nodes in the query tree should be present; and

• additional nodes that are not in the query tree can be present. However these nodes should have a direct relationship with at least one of the nodes in the query trees.

The path identification can be further described as a mapping process by which a query tree is mapped to the knowledge base. This process begins with initialising the intermediate schema as the KB graph. The initialisation enables us to check
the validity of query trees under consideration. After initialisation each query
tree is mapped to the existing graph. The mapping involves selection of edges
in the knowledge base graph according to a query tree and is done sequentially.
The edges in a query tree varies depending upon the query type it belongs. This
illustrates the need of identification of query type as well as different mapping
techniques.

While trying to map a query in the existing graph, the following possibilities arise.
The corresponding outcome in each case is also illustrated.

- All nodes, that are present in a query tree, exist in the KB graph: the query
can be mapped without any difficulty.

- The measure node and another node which is directly connected to the mea-
sure node are present: the query is partially mapped. Specifically, only an
edge from the query is mapped to the graph and the rest of the tree is dis-
carded.

- Could not find a node that is similar to the measure node: the query will
be rejected initially. It is assumed that the mapping will take place only if
there is a node in the KB graph that is required by the query. However we
will consider those queries separately and will identify whether it is possible
to add a new measure node in the existing graph based on the other nodes
present in the query tree. (See Section 4.6.3)
• The measure node and another node that is not directly connected to the measure is identified: As per normal procedure, this type of query will also be rejected. Similar to the above case a separate treatment is necessary here which may allows us to map the query partially or completely. (Discussed in Section 4.6.3)

The above discussed possibilities urge the importance of identifying similar nodes corresponding to the nodes in a query tree and this will consider in detail in Section 4.3.2. Straightforward mapping is addressed through algorithms in the following sections.

Query trees do not necessarily indicate the dependency among the participating nodes however the knowledge base provides that dependency so query trees may require modification. The modification is mainly suggested on edges. For example, if there is an edge from one node to another node in a query tree, and there is no such direct edge in the KB graph; the given query edge can be mapped in terms of more than one edge depending upon the underlying knowledge base structure. Similarly the knowledge base may also require modification. This is mainly done to include the query semantics in the intermediate schema. In order to capture the requirements functions from query trees, it is sometimes necessary to add new edges in the KB graph. These modifications are discussed in detail in Section 4.3.
Based on the above discussions the main intermediate schema algorithm is presented here and the different mapping process, with respect to query types, will follow the algorithm.

**Algorithm 4.1 Intermediate Schema Algorithm**

1: Input: $G_k$, $Q$; where $Q$ is the set of all query trees
2: Initialise $G'_k$ as $G_k$
3: For each $G_{q_i} \in Q$
4: do
5: Set query type
6: Modify as per query type
7: done

In the first step, the algorithm initialises the intermediate schema graph ($G'_k$) to that of KB graph, $G_k$. Later, according to the query type, modification has been suggested to the initial graph. The final modified graph is the intermediate schema; a common platform for data warehouse schemas. The modification of the KB graph or the query trees can be explained in terms of query types. The following section considers this aspect in detail.

### 4.3 Mapping Query Trees

As mentioned in Section 4.2 mapping a query tree means identifying a path, equivalent to a query tree, in the knowledge base. To do such a process, a query tree is treated as a sequence of edges. We have seen in Section 3.5 that the semantic constructs existing in a query vary depending on its type. Hence the edges in a
query tree also vary from one type to another type. The edges associated with query types are explained in detail before the mapping process.

- **TypeSS**: As discussed in Section 4.4.1, TypeSS queries are mainly based on a measure and classes. In that respect, the edges can be described as a combination of these types of nodes. Since measure is the root in all query trees, an edge satisfying the root node can be either one of the following:

  - (measure, fundamental class) \[ (M_i, f_{c_j}) \]: A fundamental class \( f_{c_j} \) follows a measure node \( M_i \). An example to this type of edge is an edge from sales to customer; where sales is the measure and customer is the fundamental class.

  - (measure, class) \[ (M_i, c_{jl}) \]: In this case, a class node \( c_{jl} \) follows measure \( M_i \). An edge \( (sales, week) \) is an example to this type of edge where week is a class node.

Since there is only one measure node, the other edges that exist in query trees can be combinations of classes.

  - (fundamental class, class) \[ (f_{c_j}, c_{jk}) \]: In this case, the edge starts from a fundamental class and ends in a class node.

  Example: An edge from day to quarter comes under this category.
- (class, class) \([\langle c_{jk}, c_{jm} \rangle]\): An edge with two class nodes can be found in query trees; \((\text{quarter, year})\) is an example to this.

- \(\text{TypeSM}_1\): In \(\text{TypeSM}_1\) queries, the edges with a measure node are similar to \(\text{TypeSS}_1\). That is, a fundamental class or a class follows a measure. The remaining edges are also similar to that of \(\text{TypeSS}_1\) except for the fact that the class nodes participating in an edge may be from two different classification lattices. For example, an edge like \((\text{region, year})\) could be found in \(\text{TypeSM}_1\) query tree. In addition to the edges that are discussed in the context of \(\text{TypeSS}_1\), one more type namely \((\text{fundamental class, fundamental class})\) could be found in \(\text{TypeSM}_1\).

- (fundamental class, fundamental class) \([\langle f_{cj}, f_{cl} \rangle]\):

  This type of edge has two fundamental class nodes which show the participation of two different classification lattices in a query. An edge \((\text{customer, product})\) is an example to this type.

- \(\text{TypeSS}_2\): Descriptive attributes are present in \(\text{TypeSS}_2\) queries. In that sense, an attribute node \((Da_t)\) can follow a measure node \((M_i)\) and an edge of type \((M_i, Da_t)\) is possible in these queries. The other two types of edges with a measure node are the same as that of \(\text{TypeSS}_1\). Remaining edges are combinations of classes and attributes. Edges that involves only classes are similar to \(\text{TypeSS}_1\) and edges with attributes are:
– (fundamental class, descriptive attribute) \[ (fc_j, Da_t) \]: In the case of edges with attributes, an attribute follows a fundamental class or vice versa. That means both types of edges; \( (fc_j, Da_t) \) and \( (Da_t, fc_j) \) are valid in a query. An example is \( (product, colour) \).

– (class, descriptive attribute) \[ (c_jl, Da_t) \]: As in the case of fundamental class, attribute with a class also forms two types of edges such as (class, attribute) and (attribute, class). Edge like \( (region, population) \) is an example to this where \( region \) is a class and \( population \) is its descriptive attribute.

– (descriptive attribute, descriptive attribute) \[ (Da_t, Da_s) \]: A combination of two descriptive attributes can form a valid edge in a query. An example for such an edge is \( (manager, product size) \). In this edge \( manager \) is an attribute of the class \( category \) and \( product size \) is an attribute of the fundamental class \( product \).

• \( TypeSM_2 \): All edges discussed in the case of \( TypeSS_2 \) are also possible for \( TypeSM_2 \) and the classes participating in the query can be from different classification lattices. Other than that, an edge consisting of two fundamental classes is also valid.

• \( TypeSS_3 \) and \( TypeSM_3 \): These types of queries contain a measure and one or more attributes hence only two types of edges are possible; a measure
with an attribute, \((M_i, Da_t)\) and the other is an edge with two attributes \((Da_t, Da_s)\).

- Other query types: The edges present in other query types such as \(TypeMM\) and \(TypeE\) can be described as a combination of different types of edges discussed previously.

Various types of edges that could be found in different query trees are presented above. The edges are described with respect to the participating nodes, which leads to the possible classification of query trees. For the mapping process it is necessary to classify the query trees as per the taxonomy provided in Chapter 3. Similarly the query trees produced from the natural languages queries need to satisfy the definition of a query tree given in Section 3.4. The classification rules and the validation of query trees are discussed in the following section.

### 4.3.1 Validation and Classification of Query Trees

As per query tree definition, the root of the tree must be a measure node and there is only one measure in a tree. So for each query tree the following conditions need to be tested.

- The degree of a measure node in every tree should be one.

- If there is more than one measure node in a query tree, divide the tree into sub-trees having different measure nodes.
If a query tree does not satisfy the above conditions, then that tree can not be considered for the mapping.

Classifying query trees to one of the categories such as $TypeSS_1$, $TypeSS_2$, etc. requires identification of nodes in the query trees. The following guidelines are applied while classifying query trees into different categories.

- Identify the measure node in the query tree
- Check, if the measure is a basic or derived measure type
- For the remaining nodes
  - Check if the nodes are from single or multiple classification lattice.
  - Check if there are any attributes in the tree and do the classification accordingly.

Mapping of a query tree requires identification of similarity between the nodes in a query tree with that of nodes in the KB graph. Similarity checking is needed for all edges regardless of their type and for this purpose edges in a query tree are considered one at a time. The similarity function suggested in Section 3.2.3 is useful for this purpose. We have introduced similarity function as a means to check similarity between measure types however, that function requires extension in the context of this mapping process and is detailed as follows.
4.3.2 Similarity Function

As stated in Section 3.2.3, domain intersection is one possible way to test the similarity. That is, if the domain intersection of the participating nodes is not null, nodes are similar. Since we do not impose a particular quantification on node types at this stage, a domain is treated as a set of values and set similarity is taken as a similarity measure. This case is extended to all types of nodes such as classes and descriptive attributes. The function is represented by the notation, \( SIM \) and its functionality is described as follows.

The function takes parameters from a maximum of two nodes. These nodes correspond to the edge that connects the two nodes. The nodes are then compared against the nodes that exist in \( G_k \). A simple example of such a comparison would be the domain intersection. That is, if the domain intersection of one of the nodes from a query tree and a node from the KB graph is not null, then those nodes are considered to be similar. After identifying similar nodes in KB graph, the function returns values. These values can be:

- a node: which is similar to one of the input nodes. This case happens when the function could not find similar nodes for both the input nodes. If the function returns such a single node, the mapping process has to exit at that point for all query types except TypeM queries. TypeM queries have only one node (cf: Chapter 3) so the similarity function will take parameters from
that node and return a similar node. Then the function can be written as:

\[ SIM(M_i, c_{jl}) = M_i \text{ or } SIM(M_i) = M_i \]

• a pair of nodes: If there is no direct edge between the given input nodes the function returns a pair of unconnected nodes that are similar to the input nodes. This occurs when:

  – both the input nodes are of type class and they belong to two different classification lattices
  
  – both the input nodes are of type attribute and their parents belong to two different classification lattices

For example, in the case of an edge \((\text{region}, \text{year})\) from a query tree, the similarity function returns two nodes; \(\text{region, year}\). As per the KB graph there is no direct edge between these two nodes so the function produces only the nodes as its output.

• an edge: For two given nodes, the function can return an edge. That edge consists of nodes which are similar to the input nodes. Generally the edge is a directed one which represents an existing semantic relationship in the knowledge base.

Example: If the parameters of the function are of type measure and fundamental class, then \(SIM(M_i, f_{c_j})\) returns the edge \((f_{c_j}, M_i)\).
Similarly if there exists an edge in the knowledge base corresponding to the input nodes, the function returns that edge irrespective of the node type.

- a set of edges: In certain cases, the function produces a set of edges depending upon the input nodes. For example, if the input nodes are of type measure and class; there is no direct relation between them in the knowledge base, then the function identifies a fundamental class that is related to the measure and the class. The function is written in the form:

\[
SIM(M_i, c_{jl}) = \{ (fc_j, M_i), (fc_j, c_{jl}) \}
\]

Similarly if both the input nodes are of type descriptive attribute, the function identifies the parent of those attributes and thus produces two edges. The attributes can be from

- a single parent; either fundamental class or class; and

- two different parents; a fundamental class and a class or two distinct classes.

That is:

\[
SIM(Da_t, Da_s) = \{ (fc_j, Da_t), (fc_j, Da_s) \} \text{ or }
SIM(Da_t, Da_s) = \{ (c_{jl}, Da_t), (c_{jl}, Da_s) \} \text{ or }
SIM(Da_t, Da_s) = \{ (c_{jk}, Da_t), (c_{jk}, Da_s) \} \text{ or }
SIM(Da_t, Da_s) = \{ (fc_j, Da_t), (c_{km}, Da_s) \}
\]

are valid function outputs. In the first two cases the \( SIM \) function produces
edges based on a single parent whereas in the last two cases the attributes have different parents.

There is a possibility that the function could produce more than two edges. This happens when the input nodes are of type measure and attribute and the attribute belongs to a class. Then the function returns three types of edges such as:

- (fundamental class, measure)
- (fundamental class, class)
- (class, attribute)

The function is formally written as:

$$SIM(M_i, D_{at}) = \{(f_{c_j}, M_i), (f_{c_j}, c_{jl}), (c_{jl}, D_{at})\}$$

The presence of derived measure type in a query also produces more than one edge for $SIM$ function. From the definition of a derived measure type (Section 3.2.2), it is related to other measure types. If the measure type under consideration is derived, the function will return the relationships between the given measure type and the associated measure types in the form of edges.
4.4 TypeSS Queries and the Mapping

We have seen different sub-classes of TypeSS query in Section 3.5. Mapping of each query type, such as TypeSS₁, TypeSS₂ and TypeSS₃, are considered here in detail.

4.4.1 Mapping TypeSS₁ Queries

Different types of edges associated with various query trees have been presented earlier in Section 4.3. Based on that discussion, a TypeSS₁ query tree can have edges like \((\text{measure}, \text{fundamental class})\), \((\text{measure}, \text{class})\), \((\text{fundamental class}, \text{class})\), \((\text{class}, \text{class})\). The mapping process with respect to different edges is as follows.

Since a measure node is the root of a query tree, the mapping starts from that node and is first identified from a query tree. In the next step, neighbour nodes, which form an edge with the measure node, are selected and each edge is mapped in the knowledge base. The similarity function is called at this stage to get the corresponding edges in the KB graph and as per the function output, edges are selected.

The possible edges with a measure type in this case are \((\text{measure}, \text{fundamental class})\) \(\{(M_i, fc_j)\}\) and \((\text{measure}, \text{class})\) \(\{(M_i, c_{jl})\}\). Both of these edge types are possible depending upon a query tree. As stated in Section 3.4, a query can be
represented in more than one way and any one of those representations can be considered for mapping. This means, different types of tree representations such as linear and a tree with branches will come across a particular mapping process. The proposed process should be general enough to accommodate these types of representations. In that respect, the process checks for both the types of edges; \((M_i, fc_j)\) and \((M_i, c_{jl})\). For every testing condition, similarity function is used and the edges are selected in the knowledge base accordingly. If the two testing conditions are not met, the process will exit. Failure of testing conditions may occur when the similarity function returns null values and this happens when the similarity function can not find nodes corresponding to the input nodes. If the function returns null values, the process will exist stating invalid query type. The algorithm is devised on an assumption that all the required query constructs are present in the KB graph. However, we relax this assumption, and these cases are later reconsidered in order to accommodate those queries in the schema.

After processing edges with a measure node, the remaining parts of the tree are processed in a similar manner. In a TypeSS\textsubscript{1} query tree, the other edges can be \((fundamental\ class,\ class)\ [(fc_j, c_{jl})]\ and \((class,\ class)\ [(c_{jl}, c_{jk})]\). These types of edges are pre-defined in the knowledge base so the similarity function identifies those edges and can be selected.

When selecting an edge of type \((fundamental\ class,\ class)\ [(fc_j, c_{jl})]\ or \((class,\ class)\ [(c_{jl}, c_{jk})]\) in the knowledge base it is important that the selected
edge should be a transitive edge, if there are any intermediate nodes between those nodes that form the edge. The reason is that there might be a direct edge in the KB graph between two given classes which shows the mapping between the members of those classes. However as detailed in Section 3.2.4, the mapping may not be a full mapping and that leads to an incorrect query result. For example, in the case of a query edge \((sales, region)\), the similarity function returns two edges; \((store, sales), (store, region)\). While doing the edge selection, the edge \((store, region)\) should be the transitive edge. It is possible that an edge \((store, region)\) in the KB graph which shows the participation of certain stores to region. Those stores may not be part of any city. Taking this into account, if we select that edge, the query result becomes partial; store sales that are mapped to city and state will be lost. To avoid such an error, the mapping insists on transitive edges so that every member in a class is accountable for the query.

The mapping process involves the following steps and the formal algorithm can be found in Algorithm 4.2.

- **Identify the measure node:** A query tree is processed from the root node. So the measure node \((M_i)\) is identified first and set as the root of the query tree.

- **Identify all the neighbours of the measure node:** In order to handle various query representations (see Section 3.4 for different query representations), the algorithm initially considers edges having a measure node. To do so,
the neighbour nodes that are directly connected to the measure node are identified and those edges are selected for mapping.

- **Call the similarity function**: The similarity function is used to check the validity of the edges that are selected in the previous step. The function takes parameters from one edge at a time and returns values.

- **Select edges in knowledge base**: If the function returns edges corresponding to the input nodes those edges are selected in the KB graph. Otherwise another possible edge is tested. If the function returns null value or a single node the process will exit and accepts next query tree.

- **Process the remaining edges** After processing all the edges with a measure node the remaining edges are considered. For this purpose a node which is directly connected to the measure node is taken and its neighbours are identified. The similarity function is used to test possible edge types and as per function output the corresponding edges are selected in the KB graph or the process exit in the case of invalid edges. This is an iterative process and continues till the bottom of the query tree. In other words this loop will go on until it processes the last edge in a query tree.
Algorithm 4.2 Mapping TypeSS\(_1\) Queries

1: Identify \(M_i\)
2: Let \(N_{M_i}\) be the set of neighbors of \(M_i\) \{N\}eighbour nodes that are directly connected to \(M_i\) are identified
3: \(m = \emptyset\)
4: For all \(x \in N_{M_i}\) \{x\} is a variable and is assigned to one of neighbour nodes
5: \textbf{if} \(x\) is of Type fundamental class \textbf{then}
6: \(SIM(M_i, x) = (f_{c_j}, M_i)\) \textbf{then}
7: \(e_m = (f_{c_j}, M_i)\)
8: \textbf{else}
9: Exit (Invalid edge)
\textbf{else if} \(x\) is of type class \textbf{then}
10: \(SIM(M_i, x) = \{(f_{c_j}, M_i), (f_{c_j}, c_{jl})\}\) \textbf{then}
11: \(e_m = (f_{c_j}, M_i), e_{m+1} = (f_{c_j}, c_{jl})\)
12: \(m = m + 1\)
13: \textbf{else}
14: Exit (Invalid edge)
15: \textbf{else}
16: Exit (Invalid query type) \{E\}xit process if the function returns null values
17: \(p = x\)
18: \(m = m + 1\)
19: Let \(N_p\) be the set of neighbors of \(p\)
20: \textbf{while} \((N_p \neq \emptyset)\) \textbf{do}
21: \textbf{if} \(x\) is of Type class \textbf{then}
22: \textbf{if} \(p\) is of Type fundamental class \textbf{then}
23: \(K = f_{c_j}\)
24: \textbf{else}
25: \(K = c_{jl}\)
26: \textbf{if} \(SIM(p, x) = (K, c_{jk})\) \textbf{then}
27: \(e_m = (K, c_{jk})\)
28: \textbf{else}
29: Exit (Invalid query type)
30: \(m = m + 1\)
31: \(p = x\)
32: Let \(N_p\) be the set of neighbors of \(p\) \{P\}rocess is continued until all the nodes are considered.
Example: An example query tree of TypeSS₁ is considered here in detail. A query tree equivalent to the natural language query *sales in quarter 1 of year 1999* is shown in figure 4.1. The structure of the tree is linear in this case.

![Figure 4.1: TypeSS₁ query tree](image)

For this query tree *sales* is the measure node hence the condition *(measure, class)* is true. That is, the edge *(sales, quarter)* has a measure node followed by a class node; *quarter*. The similarity function can be written as:

\[
SIM(sales, quarter) = \{(day, sales), (day, quarter)\}
\]

Since *quarter* is a class, its fundamental class; which is *day* is identified and the function returns two edges; *(day, sales),(day, quarter)*. These two edges are selected in the KB graph. Note that the edge *(day, quarter)* is transitive.

The other edge present in query tree is *(quarter, year)*. Since this edge is present in the KB graph, the similarity function returns that edge and it can be selected as part of the query path and the mapping is completed.
4.4.2 Mapping $TypeSS_2$ Queries

Unlike $TypeSS_1$, $TypeSS_2$ queries have edges with descriptive attributes. Edges with a measure node that need to be tested are:

$(\text{measure, fundamental class}) [(M_i, f_{c_j})]$, $(\text{measure, class}) [(M_i, c_{jl})]$ and $(\text{measure, descriptive attribute}) [(M_i, Da_t)]$. Similar to the $TypeSS_1$ mapping, these edges are tested by calling the similarity function and depending upon the outcome of the similarity function respective edges are selected from the KB graph.

Branches of the tree with edge type such as $(\text{fundamental class, class}) [(f_{c_j}, c_{jl})]$ and $(\text{class, class}) [(c_{jl}, c_{jk})]$ can also be identified in the graph using the similarity function. However edges with descriptive attributes require special treatment. This is mainly due to the fact that during the mapping, the parent of the descriptive attribute needs to be identified. It is possible that users may specify attributes without the parent and also the fundamental class/class specified in the query may not be the parent of the given attribute. In that case the similarity function has to identify the parents of the given attributes and will produce edges like $(\text{fundamental class, descriptive attribute}) [(f_{c_k}, Da_t)]$, $(\text{class, descriptive attribute}) [(c_{kl}, Da_t)]$.

As mentioned in Section 4.3.2, in the case of an edge with two attributes, the attributes can be either from a single parent or from two different parents. If both attributes belong to the same parent, the edges provided by the similarity function...
can be selected. That is, the user requested operation between the two attributes can be performed with respect to that parent. But if the parents are different, the knowledge base can only provide the semantic relation between the attribute and its parent. In this case, addition of user requested operation is suggested in the KB graph. The operation is captured as an edge and it is constructed between the two attributes under consideration. The newly added edge in the graph is clearly distinguished from the pre-defined edges in the knowledge base. The edges that are defined in the knowledge base represent the semantic relationship between two nodes. When a query is mapped to the KB graph, these edges also carry the operations (which is defined as requirement functions) from the query trees. However the newly added edge is purely operational in the sense that it represents an operation a user wishes to perform between the two nodes. A different notation, double headed arrow, is used to represents this type of edge.

Adding new edges in the existing graph is not restricted to edges with attributes. Addition of new edges in the knowledge base graph can be generalised in the following manner.

*If there exists a pre-defined dependency between two fundamental classes through the same measure type, new edges can be added between descriptive attributes, from one class to another class, from an attribute to a class or vice-versa.*

Our rationale for this generalisation is that adding such edges allows us to capture user defined operations which are useful during the implementation of the schema.
TypeSS₂ mapping can be written in the same way as that of TypeSS₁.

- Identify the measure node and its neighbours

- Call similarity function to test three types of edges such as $(M_i, f c_j)$, $(M_i, c_{jl})$ and $(M_i, D a_t)$

- Select the edges based on the function output.

- For the remaining edges, call similarity function.
  - if the similarity function produces edges with attributes having different parents, select the resultant edges and create an edge between the attributes.
  - if the similarity function produces an edge with an attribute and a node of type class, create an edge between the class and attribute
  - if the similarity function produces edges with attributes having the same parent, select those edges.
  - if the function output is of type (fundamental class, class)$[(f c_j, c_{jl})]$ and (class, class)$[(c_{jl}, c_{jk})]$, select those edges.

For better readability of this chapter, the formal algorithm in this case as well as all other query types discussed hereafter is included in the appendix.

Example: The TypeSS₂ query considered here is sales of washers by water usage and the query tree is shown in figure 4.2. The edge (sales, washers) is of
4.4.3 Mapping TypeSS₃ Queries

Generally TypeSS₃ queries are only a special case of TypeSS₂ queries. That is, there are only two types of edges, \(\text{measure, attribute } [(M_i, D_{a_i})]\) and \(\text{attribute, attribute } [(D_{a_j}, D_{a_k})]\) are possible in this case and these edges are present in TypeSS₂ as well. So the same mapping process discussed above in Section 4.4.2 can be used in this case. However test cases involving fundamental class/class can
be avoided. The example query that is being considered for mapping is *sales by package size* and the query tree is shown in figure 4.3.

![Figure 4.3: TypeSS3 query tree](image)

The edge in this case is of type *(measure, descriptive attribute)*. So the similarity function needs to locate the parent of the attribute. The function in this case is

\[
SIM(sales, package size) = \{(product, sales), (product, package size)\}.
\]

The edges that are produced by the *SIM* function are selected and the query is mapped.

### 4.5 *TypeSM* Queries and the Mapping

As per the proposed taxonomy, *TypeSM* queries can be classified as *TypeSM*₁, *TypeSM*₂, and *TypeSM*₃. The following sub-sections detail the mapping of these query types.

#### 4.5.1 Mapping *TypeSM*₁ Queries

In *TypeSM*₁ query mapping, all types of edges that are discussed in the context of *TypeSS*₁ need to be considered and the edges can be mapped with a similar
procedure. Since the classes that form an edge may be from different classification lattices, the similarity function may return only nodes. In that case an additional edge can be constructed between the two class nodes as in the case of descriptive attributes. If there is an edge with two fundamental classes in the query tree, the function returns the pre-existing edge so that it can be selected. A $TypeSM_1$ query tree is discussed here with an example. A query tree corresponding to the query $sales$ by region by year is considered. As shown in figure 4.4 the query tree

![Figure 4.4: $TypeSM_1$ query tree](image)

has two edges, both of them of type (measure, class). so the similarity function is:

$SIM(sales, region) = \{ (store, sales), (store, region) \}$  

$SIM(sales, year) = \{ (day, sales), (day, year) \}$. The edges produced by the similarity function are selected in the KB graph. Since the classes in the query are from two different classification lattices, in order to show the operation between the classes, an additional edge is required. That means an edge (region, year) is added in the graph.
4.5.2 Mapping $TypeSM_2$ Queries

Mapping a $TypeSM_2$ query tree is similar to a $TypeSS_2$ query tree because the edges in both types are similar. All testing conditions with respect to the measure node as well as other edges with fundamental class/class or descriptive attribute can be applied in this case as well. As with $TypeSS_2$ for an edge with attributes, the selection depends on the parent of the attributes. If both the attributes have the same parent the existing edges are selected, otherwise an edge between the two attributes is created. Similarly if there exists an edge, consisting of classes from two different classification lattices in the query tree, an edge is created in the KB graph. The formal algorithm in this case is written in the same way as $TypeSS_2$ and could be found in Appendix D.

Example: The example query used in this mapping is sales by brand by customer age. The query tree corresponding to this query is shown in figure 4.5. Similar to the example discussed in Section 4.5.1, the edges which need to be considered in this case are: $(sales, brand), (sales, customer), (customer, age)$. The edges $(sales, brand)$ and $(sales, customer)$ are of type $(measure, class)$. Therefore the similarity function returns two edges such as $(product, sales), (product, brand), (customer, sales)$.

In the case of the other edge $(customer, age)$, it is of type $(measure, attribute)$ and the function returns the existing edge. These edges are selected. Since the
knowledge base could not provide the relation between \((\text{brand}, \text{age})\) an edge is added between these two nodes which shows the user defined operation.

### 4.5.3 Mapping \(T_{ypeSM_3}\) queries

As in the case of \(T_{ypeSS_3}\), \(T_{ypeSM_3}\) queries are a special case of \(T_{ypeSM_2}\). In that sense the mapping is suggested for edges \(\text{measure, attribute } [(M_i, Da_t)]\) and \(\text{attribute, attribute } [(Da_s, Da_t)]\).

The example query considered here for mapping is \(\text{sales by clothing size by age}\) and the query tree is shown in figure 4.6. The two types of edges present in the query tree are: \((\text{sales, clothing size}), (\text{clothing size, age})\). Since both the edges have attributes the similarity function produce results that contain parents of those attributes. For the edge \((\text{sales, clothing size})\), edges such as \((\text{product, sales}),\)
(product, clothing size) can be selected as per the similarity function. However in the case of (clothing size, age), a new edge is necessary because these two attributes belong to two different parents.

Figure 4.6: TypeSM query tree

4.6 Mapping Other Query Types

TypeMM queries and exception queries are re-visited in this section to discuss the corresponding mapping.

4.6.1 Mapping TypeMM queries

The main difference between TypeMM queries and the other query types discussed in Section 4.4, and Section 4.5 is that TypeMM queries contain derived measure type whereas in TypeSS and TypeSM the measure type is basic. This
means, in order to support TypeMM queries, basic measures associated with the derived measure need to be identified during the mapping. This can be achieved in the same way as that of TypeSS and TypeSM queries which means the similarity function has to return all related measures of the given measure.

The subcategories of TypeMM queries have the same structure as that of TypeSS and TypeSM so, the same mapping techniques are applicable in this case. An example for this type of query is the stock-turn during the year 2000.

Here the measure is stock-turn and year is a class. The similarity function is called and takes parameters from these nodes. It can be written as:

\[ \text{SIM}(\text{stock-turn, year}) = \{(\text{sales, stock-turn}), (\text{purchases, stock-turn}), (\text{day, sales}), (\text{day, year}), (\text{day, purchases})\} \]

These edges are selected and the query is mapped.

### 4.6.2 Mapping Exception Queries

Exception queries are defined in Section 3.5.6 and according to that definition the query constructs have different semantics to that of nodes in KB graph. That is, a class or descriptive attribute in the KB graph can be a measure as per query. Since the constructs are already present, the query can be mapped. However, these queries require different computation compared to TypeSS and TypeSM queries. This means that those operational aspects need to be indicated in the
schema. In order to achieve this existing semantic edges are modified. Nonetheless, the pre-defined dependency remains the same and the modification shows a possible aggregation along that edge. Special notations are used in the graph for these modified edges. This can be explained with an example from our case study.

An exception query presented in Chapter 3 is *Average age of customers those who made transaction above $100*. This query has the same constructs as that of the TypeSS2 query *Sales by customer by age*. However in the former query the measure type is *age* whereas in the later one the measure type is *sales*. For this TypeEA exception query, aggregation has to be done with respect to a descriptive attribute. In order to show the aggregation with respect to the descriptive attribute the edge which carries the descriptive attribute in KB graph is modified. This modification states that an aggregation of the descriptive attribute is possible. So in the KB graph the edge \((customer, age)\) is changed from a *thin dotted directed arrow* to a *circled end* edge. A similar case is applicable to the edge \((customer, sales)\) because in the query the operation defined is *grouping*. This is different from the normal aggregation required by TypeSS or TypeSM queries. The mapping process in this case is similar to TypeSS2. Edges are modified if the testing conditions are true.
4.6.3 Reconsidering Rejected Queries

In the above discussion, a query tree will be rejected if the process cannot find a measure type node or any intermediate nodes. Here we reconsider these queries to check whether the queries can be mapped.

Sometimes users might be interested in new measure types that are not in the knowledge base. If that measure type is presented through a query, the $SIM$ function will not identify that measure type although it may be a valid one. This gives rise to the possibility of adding a new measure type in the existing graph, however this can be achieved only if the measure type provides the associated fundamental classes/classes and their relationships through the query. If it happens, the new measure type is added into the graph. The classes in that particular query might be already present in the graph, then the $SIM$ function can be utilised to identify them. If the similarity function returns only a fundamental class/class for an edge with a measure type, instead of rejecting that edge, the remaining edges are processed. If those edges are identified, the measure type is added using the other given edges. The addition of new nodes in the KB graph can also be interpreted as schema evolution. More specifically, by allowing new nodes in the schema, this approach adds provision to new knowledge in the schema that is not available during the creation of the knowledge base.

In certain cases, a measure node and another node which is not directly con-
nected to the measure node might be present in a query. Normally this query will be rejected. If the measure in the query is true, instead of rejecting the query, the remaining edges are processed and similar nodes are identified. In such cases there is a possibility of partial mapping of the query in a way that the measure node and the other intermediate class node can relate directly. If that edge is available it can be mapped in the graph.

4.7 The Intermediate Schema

We have seen different mapping processes associated with various types of queries in the previous sections. The intermediate schema algorithm, presented in Section 4.2, utilises these techniques to map a query tree to the existing graph, $G_k$. During the mapping different edges associated with a query tree are selected in $G_k$. That is, each selected path in $G_k$ represents a query tree.

The final graph that is achieved after the completion of the mapping process is called the intermediate schema; $G'_k$. Different from $G_k$, query trees can easily be identified in $G'_k$ since the paths are already selected during the mapping. An example intermediate schema is shown in figure 4.7 and a detailed version of this schema can be found in Appendix G. The paths shown in the schema correspond to queries *Sales in quarter 1, year 1999*, *sales by region by year*, *Average age of customers who made a purchase for $100*. These queries belong to $TypeSS_1$,
Type\textit{SM}_1, and Type\textit{EA} respectively. Using the respective mapping algorithms the query trees are mapped in the KB graph and the intermediate schema is created.

![Intermediate schema: an example.](image)

The intermediate schema, corresponding to the above mentioned example queries, is shown in figure 4.7. Different colour codes has been used to mark the selected nodes and edges. If an edge, required by a query tree, is already
present in the KB, it is marked as red indicating that the edge is selected as a part of a query tree. Similarly different notations are used to distinguish the types of edges. The notation; an edge with a circled end represents an existing edge which carries additional information such as different types of aggregation. Edges such as \( \text{(customer, sales)} \) \( \text{(customer, age)} \) are examples of this. Operational edges are defined in the case of \( TypeSS_2 \), \( TypeSM \) etc. An edge with a double arrow head represents this newly added edge in the schema and an example is \( \text{(year, region)} \).

In certain cases, there might be more than one path for a query. The query sales by region by year is an example of this. The path indicated in figure 4.7 is \( \text{(day, sales)} \), \( \text{(day, year)} \), \( \text{(year, region)} \). However another path, \( \text{(store, sales)} \), \( \text{(store, region)} \), \( \text{(region, year)} \) is also possible in this case. For this path, \( \text{(region, year)} \), is the operational edge. The selection of the path depends on the optimisation criterion used for the final schema.

In this section, we have presented an intermediate schema from the case study and discussed the queries. This schema satisfies the properties that are defined in Section 4.2 and these properties were revisited while addressing the mapping. In addition, this representation also carries all the semantics of the KB graph which includes the classification schema definition (cf: Section 3.2.4). In that respect the requirements such as multiple paths/shared paths and unbalanced hierarchies are satisfied through this definition. The discussion on descriptive attributes presented in Section 3.2.4 was useful to capture heterogenous attributes for a class in the
schema. Similarly the definition of derived measure types (cf: Section 3.2.2) includes derived measure types in the representation. In summary, the intermediate schema satisfies all the design considerations that we have addressed in Chapter 2.

Since the paths are known in the intermediate schema, it is easier to reduce a graph from the intermediate schema which answers all the given queries. If the designer is not interested in the reduction, but would prefer to directly reach an implementation model that is also possible from the intermediate schema. For example, cubes can be derived from this schema without any difficulty. Specifically dimensions are selected in such a way that the class nodes selected for the queries in the graph act as the dimensions of the cubes. This cube derivation is more straightforward than the existing designs that we have discussed in Chapter 2 in such a way that in those designs, queries are anticipated and thus all dimensions are kept in the schema. In this case, as the queries are mapped in the intermediate schema, dimensionality of the cubes can be reduced and a better implementation is achieved. A more detailed discussion on this matter is presented in Chapter 5.

As mentioned in Section 3.6, the intermediate schema has been suggested as an information model for various data warehouse schemas. In other words this schema is an enterprise wide schema which spans different areas of a business. So it is worth considering, dividing the schema with respect to corresponding measure types so that smaller data warehouses called Data marts are possible. In terms of data security, this technique is worth to considering. That means users
access can be restricted depending upon specific business units or eligibility.

Following the discussion regarding the intermediate schema, here we summarise the characteristics and usefulness of this schema.

- Intermediate schema is a generalised representation as it is described through a set of nodes and edges.
- It offers semantic as well as operational aspects of data.
- Different types of edges have been used to distinguish semantic edges from operational edges and which helps to process the queries.
- Each query that is collected from the users are indicated by means of a path in the schema.
- Since users queries are known from the schema, a reduced graph which is more optimised for the queries is possible.
- An intermediate schema also allows direct transformation to implementation models (see Chapter 5 for more detailed discussion).
- It is possible to divide the schema based on the measure types and this creates smaller data warehouse schemas for easy storage and maintenance.
4.8 Summary

In this chapter we proposed a graph oriented mapping technique through which query trees are mapped to an existing KB graph. Since the similarity function is general enough to handle pre-defined nodes this technique is applicable to any graph structures. The final graph achieved after the mapping process is called the intermediate schema. On top of the defined semantic relationships in a KB graph, the intermediate schema has operational aspects such as user defined operators. This level is considered as an information model in our framework which offers immense possibilities towards implementation. The intermediate schema itself supports queries however, to follow a complete design process, we look at schema properties. This is detailed in the next chapter along with schema mapping.
Chapter 5

Synthesis of Intermediate Schema

5.1 Introduction

We have seen the importance of user-centered schema design in Chapter 2. Motivated from that, an intermediate schema has been proposed which is synthesised based on a set of user queries. This schema carries semantic aspects from the knowledge base as well as operational details from the queries. The graph representation makes the schema very general and allows translation to other existing models. To prove the generality of our work, a mapping of intermediate schema to the existing models is presented in Section 5.2. Cube and star models are selected as the target models for mapping. The popularity of these models at implementation level justify their selection for this approach.
Even though the queries can be supported through the intermediate schema, it offers further scope for optimisation. The data Warehouse schema can be tailored for a set of queries that are considered during the intermediate schema generation. The optimisation of the intermediate schema is addressed as a schema refinement in the framework. The schema refinement method reduces the intermediate schema such that it satisfies certain properties. So depending upon the properties, different data warehouse schemas can be achieved. In general terms, an intermediate schema can serve as a platform for deriving a range of data warehouse schemas having different properties. On the other hand, it also offers the flexibility to directly select cubes or stars for implementation.

As the final step in the proposed framework we also discuss a data warehouse schema with the \textit{minimality} property. This schema derivation is explained in Section 5.3.1. The main reason behind such a step in the framework is that, the framework is proposed strictly in line with traditional conceptual design. Hence it is important to discuss schema properties that lead to logical and physical designs which are extensively mentioned in data warehouse literature. Since the intermediate schema is a graph, the particular schema derivation can be described as a graph reduction process. A reduction algorithm is presented to achieve this goal.

The chapter is organised as follows: the generality of the framework is shown by means of mapping to existing models in Section 5.2. The target models selected are: cube and star models. Each mapping is explained in detail in Section 5.2.1,
and Section 5.2.2 respectively. Following the mapping, a semantic reduction is presented in Section 5.3. This covers an algorithm for reduction as well discussion related to various schema properties. Finally the chapter is concluded in Section 5.4.

5.2 Mapping of Intermediate Schema to Existing DW Models

In Chapter 2 we have seen two popular models; cube model and star-schema. Due to the acceptance of these models among the practitioners, this section shows the mapping of intermediate schema to those models. The discussions are restricted to standard cube and star models. Extended approaches that are proposed to overcome the limitations of these models are not taken into consideration during the mapping due to the non-standardisation of those models. Through the mapping we show that our design is general, thereby subsuming the existing models.

5.2.1 Cube Design Using Intermediate schema

Recollect the cube concept from Chapter 2 where a cube is defined in terms of dimensions and measures. Dimensions act as the coordinates and different combinations of dimensions define a cell in a cube. Each cell represents one or more
measures of a fact. Empty cells are also possible in a cube which shows null values for a measure for particular combinations of dimensions.

In order to transform the intermediate schema to a cube schema, the nodes that exist in the intermediate schema need to be interpreted as facts and dimensions. That is, nodes such as measure types, fundamental classes/classes and descriptive attributes are transformed as cube constructs. The general transformation of an intermediate schema is considered first and implementation specific discussion is presented later in this section.

Each measure type in the intermediate schema can be seen as a fact with respect to the cube model. The analysis attributes present in measure types become the measures of the cube and the classes present in different classification lattices form the dimension levels. The operational aspects that we have captured from the queries can be included in the cube definition as operators. This includes the defined aggregation functions. Since the operational aspect in the intermediate schema is addressed on a conceptual basis, we do not suggest any particular cube operations. For example, the requirement functions, defined in the context of query representation (cf:Section 3.4), are high-level abstraction of operators and aggregation functions. These requirement functions will be present in the intermediate schema and can be translated to corresponding cube operations. Operations can be selected according to the implementation model.
Initially the intermediate schema is transformed as a general cube schema where the cube schema constructs are defined. Taking the intermediate schema presented in Appendix G as an example the transformation is further explained.

Formally a cube schema with respect to the intermediate schema is written as:

\[ C = \langle M, CL, \alpha, O \rangle \]

where:

- **M**: is a set of measure types and each measure type acts as a fact in the cube schema. The analysis attributes in a measure type represents the measures of that cube. For example, from the intermediate schema shown in figure G.1 the measure types such as *sales, purchases* become the facts in the cube definition.

- **CL**: is a set of classification lattices acting as the dimension hierarchies of the cube. In the above example, we have four classification lattices with respect to *store, day, customer* and *product*.

- **\( \alpha \)**: a set of fundamental class-measure type relations \( (\alpha_{ij}) \). These relations relate a fundamental class to a measure type. While constructing a cube, this relation maps a dimension level to a measure or helps to define a cell in a cube. In the intermediate schema, each fundamental class-measure type relation is indicated by a bold, dotted directed edge. The edge like \( (store, sales) \) is an example of this type.
- O can be aggregation functions such as \textit{SUM, AVE, MAX, MIN} as well as cube operations such as \textit{slice, dice} etc.

A multidimensional space based on the measure type \textit{sales} from the intermediate schema is considered here. We have seen a data cube in figure 2.1 with three dimensions. Similar to that using the intermediate schema we can construct a hypercube having four dimensions. These dimensions corresponds to the fundamental classes \textit{store, day, customer} and \textit{product}. The lattices based on these classes become the dimension hierarchies and can be represented as axes of the multidimensional space. Note that in a standard cube schema, hierarchies are implicit in the dimension definition. That means every classification lattice in the intermediate schema is a candidate dimension.

However cube models insist that hierarchy should be a balanced tree which is very restrictive. So the definition of the classification lattice (cf: Section 3.2.4) as a direct acyclic graph needs to be restricted to a balanced tree. Then the requirement such as unbalanced hierarchy will be compromised during the mapping. For example, in the intermediate schema there is a defined lattice with respect to \textit{store}. This lattice allows direct participation of certain stores to \textit{region}. When the lattice is restricted to a tree, such direct participation will not be allowed and this information will be lost.
It is seen in Chapter 2 that cube models enforce restriction on descriptive attributes. That is, all the members of a dimension level need to have the same descriptive attributes. In other words descriptive attributes should be homogeneous. However in the intermediate schema the descriptive attributes may not be homogeneous. A class in the intermediate schema allows different descriptive attributes for its members. So while translating intermediate schema to a cube, the descriptive attributes corresponding to a class have to be homogenized by avoiding all the heterogeneous attributes. This action eliminates the information related to the rest of the descriptive attributes and those queries cannot be supported. Similarly cube models do not allow derived measures. The derived measure type nodes defined in the intermediate schema become irrelevant during the transformation. The function defined for the purpose of derived measure types can be seen as a fact-to-fact relationship in existing designs. Other than our definition (cf: Section 3.2.2) only a few approaches such as [AJF01], [M.S03] studied this type of relationship. However in standard cube, representation of fact-to-fact relationship is not possible. So our measure-to-measure relation in the intermediate schema can not be translated directly to the cube.

The cube definition addressed above resembles the existing cube designs. The designs, that we have discussed in Chapter 2, conceptualise the requirements and only define the schema constructs.
In other words this is an arbitrary way of designing cubes. With the help of the intermediate schema a more practical cube construction is explained.

**Selecting Dimensions for Cubes**

As mentioned in the previous section, a cube can have empty cells. These empty cells indicate that for certain combinations of dimensions there is no measure. If the proportion of empty cells in a cube are high, then the cube is called a *sparse cube*. Such cubes are not good from an implementation point because they create storage problems [MDS98a], [LJ00]. Sparse cubes are common in data warehouses and result when the number of dimensions are high. To overcome the limitation of sparse cubes and to improve query performance, sub-cubes are suggested over the sparse cubes. This would require guidelines or techniques for selecting sub-cubes from an existing cube schema. Approaches such as [VAJ96], [LJJ02], [LJ00] address this issue and suggest different methods for selecting cubes for implementation. All these techniques assume queries in a data warehouse are ad-hoc so the cube selection problem is addressed as NP-hard.

The proposed design framework addresses the cube design problem more practically than the existing approaches. In this case, rather than assuming the nature of queries, a query taxonomy has been developed which covers general data warehouse queries. Even if some queries are missing initially, the pre-defined mapping techniques allow capturing of queries without any difficulty and the in-
termediate schema can be modified. Since the queries are already indicated in the schema, cubes that are required to answer those queries can easily be derived.

As in [TJP01] (cf:Chapter 2), cubes based on a set of queries are suggested here. In [TJP01], queries should be in Multidimensional Expression; MDX ([Cor05]), format to test their similarity and subsequently a set of cubes are generated which are suitable for those queries whereas our approach does not assume any particular query language and a set of queries are captured in the intermediate schema.

While selecting the cubes suitable for queries the following issues need to be considered.

- Standard cubes do not allow two classes from the same classification lattice as dimensions since cubes insist on orthogonality among dimensions. For example, the classes quarter, year cannot act as dimensions of the cube which shows sales. But as per a query, it may be necessary; a $TypeSS_1$ query like sales during quarter one in 1999 is common in a data warehouse.

- Constructing cubes based on descriptive attributes and their parent class is also an issue. Similar to two classes (from the same classification lattice) a cube restricts classes and attributes from becoming the dimensions. That is, queries of types $TypeSS_2$, $TypeSS_3$, $TypeSM_2$ and $TypeSM_3$ will be difficult to represent as cubes.
• Constructing a single cube equivalent to a query having a derived measure will cause problems. The business rules required for deriving measure types from basic measure types can not be included in cube representation hence basic measure types need to be represented as independent cubes. In that case, queries of TypeMM will not be accountable in cube representation.

A cube could answer one or more queries depending upon its dimensions. That is, if two queries have dimensions that are related directly or transitively, a single cube is enough to answer those queries. Taking this aspect into account, it need not be necessary to construct cubes for all queries. Therefore a minimal set of cubes for queries will become another design consideration.

To avoid conflict with the existing standard cube definition here attributes and derived measures are avoided during the cube construction. Nonetheless this is not a restriction on our framework, if the target model allows attributes and derived measures, they can be mapped. For example, the object-oriented approaches such as [Leh98] and [AJF02] support descriptive attributes and derived measures and may be useful for the mapping.

Based on the design considerations illustrated above, the cube construction process is further explained. The relations such as classification relation and fundamental class-fundamental class relations (γkl), exist in the intermediate schema, are mainly used here for the selection of dimensions. The classification relation
is employed to choose orthogonal dimensions. This could be described as a functional dependency test where two classes are prohibited from becoming the dimensions of a cube if one class is functionally related to the other. The formal definition of functional dependency is:

*two classes, \( c_{ji} \) and \( c_{mn} \) are said to be functionally dependent if and only if for each member in the class \( c_{ji} \), there exists exactly only one member in the other class \( c_{mn} \) [WJH98].*

The above stated condition was taken into consideration as one-to-one mapping between classes during the formalisation of classification in Section 3.2.4. Hence it is true that there exists a functional dependency between the classes in the classification schema and the edges that are members of the classification relation can be used to test the dependency between classes. This dependency test will guarantee orthogonal dimensions.

Along with classification relation, fundamental class-fundamental class relations also aids to choose valid dimensions. This could be interpreted as another type of dependency test with respect to the intermediate schema. The condition is stated as:

*if two lattices are related to a measure type through their fundamental classes, the classes in those lattices can act as the dimensions of a cube otherwise separate cubes are constructed with those classes.*

192
In the example intermediate schema shown in figure 4.7, the fundamental classes *store, product* are related to each other hence any classes from those lattices can be the dimensions of the cube.

The selection of dimensions is formally presented as an algorithm in Algorithm 5.1 and is detailed in the following steps.

1. Identify the measure type: The measure type (used in a query path) determines the required cubes except that the same measure type may form different cubes.

2. Identifying a class as candidate dimension: Selection of a class as a dimension is done with respect to the corresponding fundamental classes. A class \( c_{jk} \) from a classification lattice will become a dimension if and only if:

   - it should be referred in a query path either by means of a semantic or an operational edge and

   - there is no \( c_{jl} \) (required by a query) in that lattice such that \( c_{jl} \leq c_{jk} \) (where \( \leq \) represents the partial ordering). This condition insists that the class selected as a dimension will be the lowest one in the partial ordering and this ensures the required minimum granularity of the class from a lattice.

While selecting the class, both semantic and operational edges are taken into
consideration to ensure the required granularity. For example, let *year* be a class involved in a semantic relationship and *week* be another class participated in an operational edge. If *year* is selected over *week* based only on the semantic relation, the latter class will be lost from the schema and the query based on that class can not be answered. Hence the presence of semantic as well as operational edges are tested with respect to a class.

The selected class; $c_{jk}$ will become a member of $X_j$ where $X_j$ is a set corresponding to the fundamental class $f_{cj}$ and the members of this set represent candidate dimensions. That is, $X_j = \{c_{jk}\}$ and $c_{jk}$ is a dimension.

3. Identification of the remaining dimensions: The relation between the fundamental classes are used for the identification of the remaining dimensions.

For example, let $c_{mn}$ be a class in a different lattice based on the fundamental class $f_{cm}$ and is related to $f_{cj}$, the class $c_{mn}$ can be added to $X_j$. This dependency test is carried out for all lattices that are related to the measure type and the members of $X_j$ are identified accordingly. Similarly even if a class is related to one of the members of a dimension set through an operational edge that class can also be treated as a possible dimension.

For instance, if the class $c_{xz}$ forms an operational edge with $c_{jk}$, $c_{xz}$ can be added to $X_j$ as a possible dimension.

4. Repeat step 2 for all fundamental classes connected to the measure type.
5. Eliminating redundancy: Depending upon the number of fundamental classes, the above described selection process creates \( n \) dimension sets. Generally the fundamental classes are related to each other hence there is a possibility of redundant sets of dimensions. To avoid such a scenario, the following constraint is applied.

*If a dimension set \( X_j \) is a subset of another set \( X_m \), the set \( X_j \) is eliminated.*

Another type of redundancy occurs as a result of operational edges in the intermediate schema. A class may occur in two different dimension sets due to an operational edge. This means, redundant dimensions will be created. Therefore a class which is repeated in two different sets should be eliminated from one of the sets. This elimination depends on the number of classes in the corresponding sets which means, when eliminating a class from a set due to redundancy, the set with more number of classes is chosen over the other. Such a consideration allows the creation of cubes with minimum dimensions.

Using the intermediate schema given in the Appendix G, the working of the cube design algorithm is illustrated.

The process starts with the identification of a measure type which is *sales* in this case. In the next step (line 4 of the algorithm), for each fundamental classes; *day, product, store*, the lowest class, referred by a semantic or operational edge, is
Algorithm 5.1 Dimension Selection Algorithm

1: Input: $G'_K$
2: For every $M_i \in G'_k$
3: do
4: For every $f c_j$ connected to $M_i$
5: Reach $c_{jl}$ such that $c_{jl}$ is referred by a semantic or operational edge and there is no $c_{jp} \leq c_{jl}$ in that lattice.
6: $X_j = \{c_{jl}\}$
7: $\forall f c_k, j \neq k$
8: if there is an edge of type $(f c_j, f c_k)$ then
9: $X_j = \{c_{jl}, c_{kp}\}$
10: if there is an operational edge between $(c_{kp}, c_{nq})$ then
11: $X_j = \{c_{jl}, c_{kp}, c_{nq}\}$
12: else
13: Go to step 4
14: else
15: Goto step 4
16: if $X_g \subseteq X_j$ then
17: Eliminate $X_g$
18: else
19: $c_{kp} \in X_k$ and $c_{jl} \in X_j$ and there is an operational edge $(c_{kp}, c_{jl})$ and the number of elements in $X_k > X_j$
   Eliminate $c_{kp}$ from $X_j$
20: done
identified. These classes are *quarter*, *region* and *brand*. Then, the dimension set in each case is written as:

\[ X_d = \{\text{quarter}\}, \quad X_p = \{\text{brand}\}, \quad X_s = \{\text{region}\} \]  

where:

- \( X_d \): set of dimensions with respect to the fundamental class *day*
- \( X_p \): set of dimensions with respect to the fundamental class *product* and
- \( X_s \): set of dimensions with respect to the fundamental class *store*

Then the remaining classes are added in each of these sets sequentially, as per the existing dependency. Since the three fundamental classes; *product*, *day* and *store*, are related to each other the members of the sets will be:

\[ X_d = \{\text{quarter, region, brand,}\} \]  
\[ X_p = \{\text{brand, quarter, region}\} \]  
\[ X_s = \{\text{region, brand, quarter}\} \]

Here, the sets are identical so any one of these sets can be taken as the dimension set and the cube is constructed using these dimensions.

Compared to the normal cube design, the above discussed approach is more practical in the way that using the information from the intermediate schema, query specific cubes can be created. Nonetheless, every query in the intermediate schema may not be captured in the cube schema. For example, queries like *Sales by clothing size by age*, *Sales by product by package size* and *Show the stock-turn during the year 1999* are a few of them that are not represented in the synthesised cube schema. Generally, cubes are suitable for queries which involve
classes. However, the cube representation is not expressive enough for queries with attributes and derived measure types.

In addition, the lattices with respect to customer and store share paths. In the case of cubes these shared paths need to be separated and classes will be repeated. This means the dimensions customer and store have redundant dimension levels in the cube which is not a desirable property as it cause storage problems.

5.2.2 Mapping to Star-schema

As seen in Chapter 2, the main constructs of a star-schema are; a central fact table and a number of dimension tables. In this section transformation steps are suggested to transform our graph based schema to a star-schema. These steps can be written as:

- each measure type node corresponds to a fact table. The identity attributes and the analysis attributes become the attributes of the fact table.

- every classification lattice is equivalent to a dimension table. Each class including the fundamental class in a lattice become attributes of the dimension table.

- any descriptive attributes, exist in the lattice, that are common to the members of a class, are included in the respective dimension tables.
Using the above transformation steps, the intermediate schema can be transformed as a star-schema. The number of fact tables in the star-schema depends on the number of measure type nodes in the intermediate schema. So a data warehouse may have more than one fact table. Star schema allows sharing of the same dimension tables to different fact tables. Then, the fundamental classes that are related to different measure type nodes can be mapped without any difficulty.

Even though a star schema is considered as a logical model, it assumes relational implementation. Since our graphical representation is proposed purely in conceptual terms, here we avoid any discussion on attributes that become the primary and foreign keys of dimension and fact tables respectively.

Another major issue with a star schema is that it does not model hierarchies very well [JLS99]. In a star schema, the hierarchical structure is collapsed and represented as one table. This is called de-normalisation and the schema is called a de-normalised schema.

Due to the lack of consideration of hierarchies, a star-schema does not address requirements such as multiple paths, shared paths and unbalanced hierarchies. In other words, the table representation is not enough to capture various requirements related to the hierarchical nature of business structures.

As stated in Chapter 2, a variation of a star schema is called a snow-flake schema in which each dimension level is represented as a separate table. For this reason, the schema is said to be a normalised schema. Representationally, an in-
Intermediate schema resembles a snow-flake schema as each class in a classification lattice is represented as an independent node. This is similar to the dimension level tables in a snow-flake schema. However, through the classification schema definition (cf: Section 3.2.4) an intermediate schema covers the limitations related to hierarchies in star/snow-flake schemas.

As in the case of cube models, star schemas restrict descriptive attributes that are not common to all members of a dimension level which means heterogeneous descriptive attributes are problematic in star representation and queries of this type cannot be answered. The query *sales by clothing size* is an example of this.

Not all business rules can be incorporated in a star. So the derivation functions that are used in the intermediate schema for defining derived measure types, cannot be transformed. Since fact-to-fact relationships are not defined in a star, derived measure types need to be avoided.

Similarly any forms of operational aspects cannot be included in a star so the requirements functions that exist in the intermediate schema are not useful in the case of a star schema.

Following the general discussion related to the transformation of an intermediate schema to a star schema, this matter is further discussed here. This is mainly related to the enhancements that are possible in the star model which allow us to explain more user requirements. For this purpose the specific characteristics of an intermediate schema have been utilised.
The intermediate schema has been developed on the assumption that there is more than one measure type in the schema. When it is transformed to the star model, the schema contains several fact tables. Each fact and its associated dimension tables create a star. If the stars are isolated, having several facts in the schema become useless [AJF02]. The stars need to be related in some way so that the information present in individual stars can be used for analysis.

The measure-to-measure relation defined in the intermediate schema relates fact tables together. This means hierarchically linked fact tables may be possible by means of measure-to-measure relations. As mentioned in [D.100] (cf: Section 2.6), this constellation provides the ability to drill-across from one fact table to another fact table. For example, assume we have two measure types *sales* and *total price* in the intermediate schema related by a measure-to-measure relation. These measure types create two connected fact tables with shared dimension tables which is shown in figure 5.1. In this star constellation, drill-across from *sales* to *total price* is possible so that the user gets a simultaneous view of certain *products* sold and the *total amount sold*.

Similarly fact tables that are not directly related are used to construct *galaxy*. In both constellation and galaxy the dimensions are shared. The galaxy corresponding to the measure types *sales* and *purchases* can be found in figure 5.2. Galaxy schemas are treated as a generalisation of star and constellation schemas with a view to supporting multiple facts in a schema.
The classification schema suggested in the context of an intermediate schema allows shared paths. This could be interpreted in a star as dimensions coinciding their aggregation levels. For example customer and store share the same hierarchy. In order to represent this hierarchy in a star schema two dimension tables corresponding to customer and store are necessary. These dimension tables will have redundant attributes. Redundancy causes problems at the physical level especially during schema evolution. Any changes in one of the attributes in a hierarchy will affect the entire hierarchical structure. In other words if the duplicated data changes, the changes need to be propagated to all attributes in the dimension ta-
bles. Snow-flake schema avoids problems with redundancy but create more joins and complicate the schema [R.K96]. A more general dimension incorporating the independent dimensions may be a solution for this issue [AJF02].

Another interesting factor that relates to a star schema is that the query paths consist of modified edges. During the mapping process (cf:Section 4.6.2) we have modified the existing edges in order to differentiate $Type E$ queries. This modification has been suggested to indicate that the measure in that query may be a class/descriptive attribute in intermediate schema. Then as per the star model, a dimension becomes fact and fact becomes dimension. The general considerations
treat all measure types as fact and classes as dimensions. Nonetheless, the inter-
mediate schema does not restrict a measure type as a dimension and class as fact
or vice versa.

5.2.3 Discussion

In this section we have addressed the implementation of the proposed graph rep-
resentation. Selecting cubes for implementation is the main achievement in this
regard. Selection of sub-cubes for better query performance is a well-known tech-
nique in this area and in the past various approaches addressed this issue. However
in these approaches dimension selection is not systematic whereas in this design,
the algorithm identifies classes based on queries and dimensions are selected ac-
cordingly. Moreover the algorithm produces only minimal cubes for the queries.
The proposed query taxonomy plays an important role in this regard which al-
lowed to cover general data warehouse queries.

The existing cube designs (cf:Chapter 2), proposed models with only defini-
tions of schema constructs. Ad-hoc queries were the main consideration behind
such approaches. This consideration restricted those designs from any discussion
on schema refinement and properties. In our approach, all aspects of a design
have been addressed including the derivation of a schema for implementation.
Similarly modifications on star schemas have also been suggested. Existing con-
cepts such as constellation and galaxy schemas have been adopted for this purpose. These concepts allow the alteration of star representation in a such way that connected facts are possible and hence analysis can be improved.

5.3 Conceptual Reduction

Different steps associated with the design were discussed in Chapter 3 and Chapter 4. The systematic design approach begins with requirement identification. From the collected business knowledge, a knowledge base is created which serves as an initial knowledge for a data warehouse schema. Then, based on a set of queries (collected during the requirement specification stage), a query oriented schema called intermediate schema is generated. Finally the intermediate schema is refined to obtain a data warehouse schema; which will be detailed in this section.

The derivation of the data warehouse schema is presented as the last step in our design and it depends on schema properties. Completeness and minimality are two main properties that a conceptual schema should satisfy. Hence these properties are defined in the context of the proposed framework. Based on this a reduction algorithm is proposed to get the final schema.

In this framework, the properties such as completeness and minimality are defined with respect to queries. The schema is said to be complete if it contains
all the information required to answer the queries under consideration. In this re-
spect, an intermediate schema satisfies the completeness property since it has been
developed based on a set of queries. We have seen this in Section 4.2 that a set
of queries have been considered for the intermediate schema generation. If com-
pleteness is the only property that we are looking for a schema, the intermediate
schema can be treated as the data warehouse schema. Considering schema refine-
ment as one of the steps involved in design, intermediate schema could be used
for further optimisation. This suggests possible reductions and we will introduce
the minimality property for this purpose.

As mentioned earlier in this section, minimality of the schema is addressed
with respect to the queries. In this framework we describe minimality in terms of:

- minimum constructs to support queries. This property defines the minimum
  number of nodes and edges in the schema. The rest of the intermediate
  schema can be eliminated. This means, the intermediate schema is refined
  in such a way that there is no redundancy considering the queries that need
to be supported.

- minimum path length or shortest path for a query. It is also helpful to de-
  scribe the minimum construct property in terms of minimum path length.

We know that in the intermediate schema, every query is indicated by means
of a path. In this path the root node will be the measure node and the end
node be a fundamental class/class or a descriptive attribute. If the path constructs are minimum (as per the above property) the length of the path will also be minimum. This leads to another property for the schema; minimal path length for queries. In order to satisfy the properties; minimum constructs and path length, a path which involves a direct edge between two nodes needs to be selected over other existing paths. This will be detailed in Section 5.3.1.

- minimum number of operations. Similar to the number of constructs, the number of operation required for a query can also be a criterion. More precisely, the minimum number of operations can also be another schema consideration. While introducing the intermediate schema in Chapter 4 we have discussed that, other than semantic relationships the edges carry requirement functions from the queries. These requirement functions have been suggested as a generalisation of operators involved in the queries. That is, each edge in the intermediate schema has one or more operations associated with it. In order to compute a query based on a query path, the operations corresponding to each edge in the query path need to be performed. In that circumstance, the operations are directly related to the number of edges in the query path. Then if there are multiple paths for a query, operationally the path with minimum edges may be a good choice over the
others. However, another interesting aspect that can relate to an operation is the number of tuples corresponding to each node. Since data is not involved at this stage of the design, it is difficult to take this aspect into consideration. As the number of tuples are known only after the data loading, at that time further physical optimisation is suggested to improve the efficiency of the operations.

Based on the above presented discussion, in Section 5.3.1 a conceptual reduction is proposed. This reduction takes the minimum path length for queries into consideration to produce a schema accordingly.

5.3.1 The Reduction Algorithm

The reduction of the intermediate schema to a query based data warehouse schema is explained here using an algorithm called reduction algorithm. The algorithm takes the intermediate schema ($G'_k$) as input and produces another graph, $G''_k$, which is the minimal data warehouse schema. The reduction has been attempted from a graph reduction point of view. However, the context of a shortest path algorithm is slightly different in our case from the shortest path algorithms in graph theory. In this case the start node and end node of a path are known whereas the graph theory algorithms are supposed to identify the nearest neighbours for a given node. So instead of using one of the existing graph theory algorithms, we
suggest a reduction algorithm which takes the semantics of our graph representation.

As the queries are already mapped in the intermediate schema, it is easier to reduce this graph according to the queries. The algorithm operates in such a way that it selects only that part of the graph which is required for a query tree. That is, the nodes and the edges that are identified during the mapping process are selected as schema constructs and the rest is eliminated.

In order to meet the shortest path requirement the selected path should have only minimum length compared to other alternate paths. For this purpose, we use the transitive edges. The condition is written as: *if there are any intermediate nodes, between two selected nodes in a query path the transitive edge is selected instead of other existing edges.* Selection of the transitive edges preserves the semantic as well as operational relationship in the schema. Thus guaranteeing correct results for the queries.

As per the intermediate schema, there are certain basic relations that are required for deriving other relations. These are the measure-to-measure, fundamental class-to-fundamental class, and fundamental class-measure relations. The measure-to-measure shows the relation between two or more measures. Similarly fundamental class-measure relations are necessary to relate a class to a measure. Fundamental class-to-fundamental class is another type of relation that is required in the schema. So these basic relations are kept in the schema as per queries.
The steps involved in the algorithm are:

- selecting basic relationships: Keep all the selected edges of type \((f_{c_j}, M_i)\) and \((M_i, M_k)\). **Fundamental class, measure** \([\{(f_{c_j}, M_i)\}]\) is necessary in the schema to derive higher classes in a classification lattice because all other classes in the lattice are related to the measure through the fundamental class. Similarly **measure-to-measure** \([\{(M_i, M_k)\}]\) relation defines a basic relation that is required to derive a measure type from another measure type.

- collapse classification lattices as required: As per the formalisation of lattices presented in Section 3.2.4, a lattice can have any number of classes. However all these classes may not be required for queries under consideration. Taking this into account, only those classes which are required for queries are kept in the data warehouse schema.

While selecting classes from a lattice, the higher class required in that lattice is identified first. If there are any intermediate classes between the fundamental class and a selected class, those classes are also considered. Then the transitive edges connecting the selected nodes are chosen in the case of classes. If an attribute is involved, the corresponding edge which shows the class-attribute relation is selected as part of the schema. For example, let *year* be the highest selected class in the lattice corresponding to the fundamental class *day*; if there is no other class is required from that lattice, the
transitive edge \((\text{day, year})\) is selected. If another class \(\text{quarter}\) is needed for a query, then the edges; \((\text{day, quarter}),(\text{quarter, year})\) are selected as schema constructs.

- Selecting operational edges: Operational edges indicate user defined operations so these edges are helpful during physical design. That is, the operational edges provide information such as tables that need to be joined and intermediate tables, if the implementation is relational. In order to keep track of the operations all the newly created edges are kept in the data warehouse schema.

The above mentioned steps are formally written in Algorithm 5.2.

**Algorithm 5.2 Reduction Algorithm**

1: Input: \(G'_{k}\); Output: \(G''_{k}\)
2: For a given \(G'_{k}\)
3: do
4: Retain all selected \((f_{c_j}, M_i)\) and \((M_i, M_k)\)
5: For each fundamental class \(f_{c_j}\)
6: Identify the highest visited class \(c_{jl}\) in the lattice
7: if \(c_{jl}\) be the only visited class then
8: Retain the transitive edge \((f_{c_j}c_{jl})\)
9: else
10: Identify all the intermediate nodes visited
11: Retain the edges with visited nodes
12: Keep all operational edges
13: done

A minimal data warehouse schema with respect to the intermediate schema presented in Chapter 4 is shown in figure 5.3. Four types of queries considered
for this reduction are yearly sales by region, Total sales in quarter1, year 1999, Sales of home brand products by age, and average age of customers who made transaction over $100. These queries belong to categories $TypeSS_1$, $TypeSM_1$, $TypeSM_2$, and $TypeEA$ respectively.

![Figure 5.3: Minimal DW schema: an example.](image)

After the reduction process, the schema contains only the paths required for the queries. Each path can be used to support one or more queries. That is, the schema is tailored for a set of queries. If queries change over time, the framework can accommodate those changes. This is possible because there is a taxonomy that incorporates general data warehouse queries as well as pre-defined mapping
processes. This mapping allows the creation of a new intermediate schema as well as a data warehouse schema.

The discussion presented here justifies our point about the intermediate schema. That is, the intermediate schema serves as an information model from where different data warehouse schemas can be derived. Even though the reduction algorithm presented here aims only for minimality, the framework does not place any restriction on deriving other schemas.

The schema reduction presented here is proposed as a part of our design and as claimed this process delivers a schema with specific properties. We have seen operators such as *pruning, grafting, collapse*, in Chapter 2, for eliminating unwanted constructs from the schema. However in those cases there is no clear criteria for elimination. Particularly, a basis on which schema can be refined was missing until now. In this thesis we have addressed this problem by refining the schema for a given property; minimality. Rather than proposing operators, here we have treated schema refinement as a step in the design and an algorithm is developed for this. The main advantage of having an algorithm is that, it can used for deriving different schemas with various properties. Whereas operators have been suggested for only removing the dimension levels and attributes from a schema. So in that case it is difficult to define a property for the schema.

The above discussed schema refinement can be further seen as an extension to the independence property. We have mentioned in Chapter 2 that current designs
interpret independence as the distinction between the logical and physical levels where as our approach adds a step further to this consideration which includes a method for schema selection/refinement. This could be described as guidelines employed to derive a schema for implementation.

At this stage we would like to revisit the properties of a methodology that we have introduced in Chapter 2 and show the properties.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Query Driven Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptualisation</td>
<td>Defined derived measure type and classification schema (See Chapter 3)</td>
</tr>
<tr>
<td>Generality</td>
<td>Implementation independent and subsumes other models (Section 5.2)</td>
</tr>
<tr>
<td>Independence</td>
<td>The design addressed all aspects of the conceptual design (Section 3.6, Section 5.3)</td>
</tr>
<tr>
<td>Queries</td>
<td>Query based schema is proposed (Chapter 4)</td>
</tr>
<tr>
<td>Formal procedure</td>
<td>Design starts with requirement specification and systematically identified schema constructs (Section 3.6)</td>
</tr>
</tbody>
</table>

Table 5.1: Proposed design vs properties

As presented in Table 5.1, our framework satisfies all the important properties of a methodology. Compared to the other approaches presented in Table 2.1, this work offers a better conceptual design due to the following reasons.

- The framework provided a step-by-step procedure to the schema design:

  In contrast to the existing designs, our approach clearly stated the design
considerations and based on these a data warehouse schema is developed at the end. There is a formal procedure for the selection of the schema constructs.

- The model is very general and subsumes other models: A general representation which acts as a basis for further schema derivation was lacking. The introduction of the intermediate schema in the framework resolved this existing issue and served as a platform for various data warehouse schemas.

- Offered direct translation to implementation models as well as provision to graph based schema derivation: Intermediate schema can be directly translated to an implementation model like a cube or it can be reconstructed as per given properties. These considerations can be treated as guidelines to streamline a model. Thus this approach filled the gap that currently exits between a model and a final schema.

- Queries have been given emphasis while developing the schema so it is more user oriented.

5.4 Summary

One of the arguments put forward for the user-centered schema design framework was its generality. In this chapter we have proved this argument by showing a
transformation of intermediate schema to existing models such as cube and star. Separate transformation as well as specific model based considerations were also addressed in each case.

Apart from the mapping, a semantic reduction has been presented as a final step involved in the design. The reduction process is included in the framework as a means of schema refinement which is lacking in current designs. For this purpose, minimality property is addressed with respect to queries and a reduction algorithm is proposed. The algorithm reduces the intermediate schema to a data warehouse schema where query paths have minimum path length. In general, the addressed schema refinement could be described as design independence which provides a method for schema refinement with respect to relevant data warehouse properties. Thus we conclude our discussion on the framework with this chapter and the thesis is concluded along with future works in the following chapter.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis investigated data warehouse conceptual design from a requirements point of view. The focus was not only on conceptualisation, but also tried to fill the gap in the existing design process. To achieve that goal, a query based design method has been adopted. Even though queries were indirectly addressed in existing designs, a query driven conceptual design was never attempted. Likewise a general model, which could subsume other models, was missing. The research presented in this thesis has addressed these issues and a general design method is suggested. The contributions of this thesis can be summarised as follows.
Query Driven Approach

A user-centered design framework is presented. User involvement is taken into consideration by means of queries and a schema is synthesised from a set of queries. In order to guarantee the correctness and completeness of the synthesis, queries have been studied. This study produced a formal query taxonomy and a query representation. The proposed taxonomy covers general data warehouse queries and suggests a classification based on the query constructs. For every query type in the taxonomy, there are pre-defined mapping processes and these mapping processes translate queries into a schema. The proposed query representation called query tree is used in the mapping which ensures query participation in the design. By providing a query specific schema this design provides a solution to the cube selection/ view selection problem discussed in Chapter 5.

Systematic Design

The proposed design presents a systematic and iterative approach to data warehouse design. The design starts with requirements collection hence the necessary design requirements are detailed. Based on these requirements, a step by step synthesis process is illustrated and a final schema is presented.
As in existing designs, the selection of schema constructs are not arbitrary in this design; proper guidelines are available to select the schema constructs which mainly rely on the queries that need to be supported.

**Introduction of Information Model**

An information model in conceptual design is very useful in the sense that it allows different schemas for a database [Jr79]. This concept is applied in our design and an intermediate schema is introduced in the design as an information model. Therefore the design offers various data warehouse schemas with specific properties and the designer has the flexibility to choose a schema based on demand.

**General and Independent Design**

The graph based approach presented in this thesis is general and this property is shown by means of mapping to cube/star models. The discussion on the transformation covered general aspects as well as dimension selection specific to implementation. The algorithm proposed for dimension selection identifies a set of dimensions required in the schema and hence query specific cubes can be constructed.

Independence of the design is addressed in two fold. The design is proposed on conceptual basis therefore it is distinguished from logical and physical designs. On the other hand, conceptual phase is extended in order to provide implementa-
tion specific schema. This aspect is considered as a conceptual reduction and a method is presented for schema refinement/selection. The refinement method has the ability to deduce a data warehouse schema with respect to required optimisation criteria.

A Function for Testing Similarity

In the context of the schema synthesis process, a function called similarity function is proposed. The main functionality of this similarity function is to test the similarity that exists between two nodes in the graphs. Even though the implementation of this function is not discussed in detail due to lack of knowledge of real data, the function could be applied to any graphs with formally defined nodes.

6.2 Future Research

Further investigations that are possible in the context of the proposed schema design are as follows.

- How well a methodology is adaptable in an organisation decides the success of that methodology. This thesis addressed most of the theoretical aspects. Nonetheless, the challenges that arise during the implementation of our methodology for data warehouse design is one area that requires further discussion. We would like to examine in future, the influence of organisa-
tional factors in the application of our methodology. For instance, organisational structure and people involved in the project do have an affect on the methodology. In particular, users participation and style, top-management support are a few factors that influence this user-centered approach.

- Although mapping of the schema has been discussed, logical transformation can be extended further, especially, the operators. Our approach considered operators as functions. It is worth considering the transformation of functions to specific operators which may be particular to certain models. Such a transformation helps to associate cost functions for the operators and optimisation can be achieved accordingly.

- In the context of the schema generation, similarity function has been developed in this thesis. This function was suggested as a stand alone function to check the similarity between nodes in the graphs. Implementation of similarity function needs more research; perhaps in the areas of ontologies or semantic webs.

- We have developed our methodology as general as possible. In that circumstance, it is interesting to see how it can applied in various data types. For example, a medical database contains multimedia data such as video, x-ray images, MRI scans and other text data. Similarly GIS and genome data also have particular characteristics.
Testing and modification (if necessary) of the methodology in non-traditional environment offers challenges.

- As we know, natural language processing is a well established area. However, in the context of our framework, it is interesting to study the transformation of natural language queries to query graphs.

- Real time data warehouses are gaining momentum these days. In this work we have avoided the temporal aspects assuming data warehouse is time invariant. This assumption can be relaxed in the future and temporal elements as well as spatial data requirements can be considered. In other words the representation can extended to accommodate spatio-temporal requirements.
Bibliography


[BCGM98] B.Dinter, C.Sapia, G.Hofling, and M.Blaschka. The OLAP market:
State of the art and research issues. In *International Workshop on


[BFMB99] M. Bouzeghoub, F. Fabret, and M. Matulovic-Broqué. Modeling the
data warehouse refreshment process as a workflow application. In *
International Workshop on Design and Management of Data Ware-
houses*, page 6, 1999.

[BHBD00] B.L.Cooper, H.J.Watson, B.M.Wixom, and D.l.Goodhue. First

in heterogeneous multidimensional schemas. In *International Con-

In *Symposium on Principles of Database Systems*, pages 169–179,
2002.


[JLS99] H. V. Jagadish, Laks V. S. Lakshmanan, and D. Srivastava. What can hierarchies do for data warehouses? In *International Conference on*
Very Large Databases, pages 530–541, 1999.


Appendix A

Proof: Classification Lattice

The formalisation of classification schema is presented in Chapter 3 and discussed classification lattices as a specialisation of the classification schema. As stated in Section 3.2.4, the mathematical proof for a classification lattice is presented here.

Theorem

In a classification schema; $CS = (C, R)$, prove $(C', R')$ forms a lattice.

Proof

We need to prove

- $R'$ is:
  - reflexive (1a)
  - transitive (1b) and
anti-symmetric (1c)

- Allc is the least upper bound LUB and
- c is the greatest lower bound GLB.

We have the definitions of $C'$ and $R'$ as:

\[
C' = \{ x \mid x \in C \land ((c, x) \in R) \} : \cup \{ Allc \} \tag{A.1}
\]

\[
R' = (R \cap (C' \times C')) \cup \{(y, Allc) \mid y \in C' \} \tag{A.2}
\]

and $R' \subseteq C' \times C'$

**Proof 1a:**

Need to prove

\[
x \in C' \Rightarrow (x, x) \in R' \tag{A.3}
\]

Assume that there is an

\[
x_0 \in C' \text{ such that } (x_0, x_0) \notin R' \tag{A.4}
\]

Then, from the definition of $C'$ either

\[
x_0 \in C \land (c, x_0) \in R \tag{A.5}
\]

or

\[
x_0 = Allc \tag{A.6}
\]

If equation A.5 is true:

\[
(x_0, x_0) \in R \tag{A.7}
\]

because $R$ is reflexive. Also, from equation A.4, $x_0 \in C'$, and hence

\[
(x_0, x_0) \in C' \times C' \tag{A.8}
\]

From equations A.7 and A.8

\[
(x_0, x_0) \in R \cap (C' \times C') \tag{A.9}
\]

From equation A.9 and definition of $R'$

\[
(x_0, x_0) \in R' \tag{A.10}
\]
Hence A.5 can not be true.
If A.6 is true:
From equation A.4
\[ x_0 \in C' \]  \hspace{1cm} \text{(A.11)}

Therefore from A.11 and A.6
\[ (x_0, x_0) \in \{(y, All_c) \mid y \in C'\} \]  \hspace{1cm} \text{(A.12)}

From A.12 and definition of \( R' \),
\[ (x_0, x_0) \in R' \]  \hspace{1cm} \text{(A.13)}

Hence equation A.6 cannot be true. Therefore equation A.4 leads to a contradiction and proved that \( R' \) is reflexive.

**Proof 1b:**

Need to prove:
\[ x_1, x_2, x_3 \in C' \land (x_1, x_2) \in R' \land (x_2, x_3) \in R' \Rightarrow (x_1, x_3) \in R' \]  \hspace{1cm} \text{(A.14)}

Assume that there are \( x_1, x_2, x_3 \in C' \) such that
\[ (x_1, x_2) \in R' \land (x_2, x_3) \in R' \land (x_1, x_3) \notin R' \]  \hspace{1cm} \text{(A.15)}

Then either:

- none of \( x_1, x_2, x_3 \) equals to \( All_c \)
  \hspace{1cm} \text{(A.16)}

- or:
  \[ x_1, x_2, x_3 \] equals to \( All_c \)
  \hspace{1cm} \text{(A.17)}

If A.16 is true:
from A.15
\[ (x_1, x_2) \in R' \]  \hspace{1cm} \text{(A.18)}

From A.16
\[ (x_1, x_2) \notin \{(y, All_c) \mid y \in C'\} \]  \hspace{1cm} \text{(A.19)}

because \( x_2 \neq All_c \)
From A.18, A.19 and definition of $R'$

$$(x_1, x_2) \in R \cap (C' \times C')$$  \hspace{2cm} (A.20)

Therefore

$$(x_1, x_2) \in R$$  \hspace{2cm} (A.21)

Similarly

$$(x_2, x_3) \in R$$  \hspace{2cm} (A.22)

Since $R$ is transitive,

$$(x_1, x_3) \in R$$  \hspace{2cm} (A.23)

Also $x_1, x_3 \in C'$, from A.15 and hence

$$(x_1, x_3) \in C' \times C'$$  \hspace{2cm} (A.24)

From A.23 and A.24

$$(x_1, x_3) \in R \cap C' \times C'$$  \hspace{2cm} (A.25)

From A.25 and definition of $R'$

$$(x_1, x_3) \in R'$$  \hspace{2cm} (A.26)

Hence A.16 cannot be true.

If A.17 is true then:

Either

$$x_1 = \text{All}_c$$  \hspace{2cm} (A.27)

or:

$$x_2 = \text{All}_c$$  \hspace{2cm} (A.28)

or:

$$x_3 = \text{All}_c$$  \hspace{2cm} (A.29)

or more than one of them equals to $\text{All}_c$.

If A.27 is true, then from A.15,

$$x_1 \in C'$$  \hspace{2cm} (A.30)

and

$$(x_1, x_2) \in R'$$  \hspace{2cm} (A.31)

Since $x_1 = \text{All}_c$ and $\text{All}_c \notin C$ we have

$$(x_1, x_2) \notin R$$  \hspace{2cm} (A.32)
Therefore
\[(x_1, x_2) \notin R \cap (C' \times C') \quad (A.33)\]

From A.15, A.33 and definition of \(R'\)
\[(x_1, x_2) \in \{(y, \text{All}_c) \mid y \in C'\} \quad (A.34)\]

From A.34
\[x_2 = \text{All}_c \quad (A.35)\]

Hence, if A.27 is true, then so is A.28.
Similarly, we can show that if A.28 is true then so is A.29.
If A.29 is true, then from A.15
\[x_1 \in C' \quad (A.36)\]

Therefore from A.36 and A.29
\[(x_1, x_3) \in \{(y, \text{All}_c) \mid y \in C'\} \quad (A.37)\]

From A.37 and definition of \(R'\)
\[(x_1, x_3) \in R' \quad (A.38)\]

Hence A.29 and A.17 cannot be true. So A.15 cannot be true; that means A.14 is true and \(R'\) is transitive.

**Proof 1c:**

Need to prove
\[x_1, x_2 \in C' \land (x_1, x_2) \in R' \land (x_1 \neq x_2) \Rightarrow (x_2, x_1) \notin R' \quad (A.39)\]

Assume that there are \(x_1, x_2 \in C'\) such that
\[(x_1, x_2) \in R' \land (x_1 \neq x_2) \land (x_2, x_1) \in R' \quad (A.40)\]

Either none of
\[x_1, x_2 \text{ equals to All}_c \quad (A.41)\]
or at least one of
\[x_1, x_2 \text{ equals to All}_c \quad (A.42)\]

If A.41 is true:
From A.40
\[(x_1, x_2) \in R' \quad (A.43)\]
and

\[(x_1, x_2) \notin \{(y, All_c) \mid y \in C'\}\]  \hspace{1cm} (A.44)

because \(x_2 \neq All_c\).

From A.43, A.44 and definition of \(R'\)

\[(x_1, x_2) \in R \cap (C' \times C')\]  \hspace{1cm} (A.45)

Therefore

\[(x_1, x_2) \in R\]  \hspace{1cm} (A.46)

Since \(R\) is antisymmetric and \((x_1 \neq x_2)\)

\[(x_2, x_1) \notin R\]  \hspace{1cm} (A.47)

Therefore

\[(x_2, x_1) \notin R \cap (C' \times C')\]  \hspace{1cm} (A.48)

From A.40, A.48 and and definition of \(R'\)

\[(x_2, x_1) \in \{(y, All_c) \mid y \in C'\}\]  \hspace{1cm} (A.49)

This contradicts A.41 because \(x_1 \neq All_c\). Hence A.41 cannot be true.

If A.42 is true:

either

\[x_1 = All_c\]  \hspace{1cm} (A.50)

or

\[x_2 = All_c\]  \hspace{1cm} (A.51)

Both cannot be true because \(x_1 \neq x_2\)

If A.50 is true:

From A.40

\[x_2 \in C'\]  \hspace{1cm} (A.52)

and

\[(x_1, x_2) \in R'\]  \hspace{1cm} (A.53)

Since \(x_1 = All_c\) and \(All_c \notin C\) we have

\[(x_1, x_2) \notin R\]  \hspace{1cm} (A.54)

because \(R \subseteq C \times C\) Therefore

\[(x_1, x_2) \notin R \cap (C' \times C')\]  \hspace{1cm} (A.55)
From A.55, A.40 and definition of $R'$

$$(x_1, x_2) \in \{(y, All_c) \mid y \in C'\} \quad (A.56)$$

From A.56,

$$x_2 = All_c \quad (A.57)$$

This contradicts $x_1 \neq x_2$ therefore A.50 cannot be true.
If A.51 is true:
Since $x_2 = All_c$ and $All_c \not\in C'$

$$(x_2, x_1) \notin R \quad (A.58)$$

because $R \subseteq C \times C$.
Therefore

$$(x_2, x_1) \notin R \cap (C' \times C') \quad (A.59)$$

From A.59, A.40 and definition of $R'$

$$(x_2, x_1) \in \{(y, All_c) \mid y \in C'\} \quad (A.60)$$

From A.60

$$x_1 = All_c \quad (A.61)$$

From A.61, A.51

$x_1 = x_2$

which contradicts $x_1 \neq x_2$.
Therefore A.51 cannot be true. Therefore A.42 cannot be true. Therefore A.40 cannot be true. Hence anti-symmetry is proved.

**Proof: Greatest lower bound**

We need to show $c$ is a lower bound of $(C', R')$.
Since $c \in C$ and $R$ is reflexive

$$(c, c) \in R \quad (A.62)$$

From equation A.62

$$c \in \{x \mid x \in C \land ((c, x) \in R)\} \quad (A.63)$$

From A.63 and definition of $C'$

$$c \in C' \quad (A.64)$$

We need to show

$$x \in C' \Rightarrow (c, x) \in R' \quad (A.65)$$
From the definition of $C'$

$$x \in C' \Rightarrow (x \in C \land (c, x) \in R) \lor x = \text{All}_c$$  \hspace{1cm} (A.66)

So, either

$$x \in C \land (c, x) \in R$$  \hspace{1cm} (A.67)

or

$$x = \text{All}_c$$  \hspace{1cm} (A.68)

If A.67 is true

$$(c, x) \in R$$  \hspace{1cm} (A.69)

and

$$x \in C$$  \hspace{1cm} (A.70)

From A.70, A.69 and definition of $C'$

$$x \in C'$$  \hspace{1cm} (A.71)

From A.71, A.69 and A.64

$$(c, x) \in R \cap (C' \times C')$$  \hspace{1cm} (A.72)

From A.72 and definition of $R'$

$$(c, x) \in R'$$  \hspace{1cm} (A.73)

If equation A.68 is true

$$(c, x) \in \{(y, \text{All}_c) \mid y \in C'\}$$  \hspace{1cm} (A.74)

From A.74, and definition of $R'$,

$$(c, x) \in R'$$  \hspace{1cm} (A.75)

Therefore equation A.65 is proven.

Therefore $c$ is a lower bound of $(C', R')$.

We need to show $c$ is the greatest lower bound. Assume it is not. Then there is an:

$$x_0 \in C' \text{ such that } x_0 \text{ is a lower bound}$$  \hspace{1cm} (A.76)

and

$$x_0 \neq c$$  \hspace{1cm} (A.77)

and

250
because $c$ is not the greatest lower bound. From equation A.76, since $x_0$ is a lower bound
\[ \{(x_0, y) \mid y \in C'\} \subseteq R' \]  
(A.79)

In particular
\[ (x_0, c) \in R' \]  
(A.80)

Equation A.80 contradicts equation 68 hence $c$ is the greatest lower bound.

**Proof: Least Upper bound**

By the definition of $R'$
\[ \{(y, All_c) \mid y \in C'\} \subseteq R' \]

Therefore $All_c$ is an upper bound of $(C', R')$.

We need to show it is the least upper bound. Assume that $All_c$ is not the least upper bound.

Then there is an
\[ x_0 \in C' \]  
(A.81)

such that $x_0$ is an upper bound and
\[ x_0 \neq All_c \]  
(A.82)

and
\[ (x_0, All_c) \in R' \]  
(A.83)

from A.81, since $x_0$ is an upper bound
\[ \{y, x_0\} \mid y \in C'\} \subseteq R' \]  
(A.84)

In particular,
\[ (All_c, x_0) \in R' \]  
(A.85)

Equation A.85 contradicts equation A.83 because $R'$ is anti-symmetric.
Appendix B

TypeSS$S_2$ Mapping

Mapping processes associated with various query types were detailed in Section 4.4 and Section 4.5. Appendix B through Appendix F will describe the mapping algorithms for these query types.

1: Identify $M_i$
2: Let $N_{M_i}$ be the set of neighbors of $M_i$ {1}nitialising a variable to assign the neighbour nodes of $M_i$
3: $m = \emptyset$
4: For all $x \in N_{M_i}$
5: if $x$ is of Type fundamental class then
6: if $SIM(M_i, x) = (fc_j, M_i)$ then
7: $e_m = (fc_j, M_i)$
8: else
9: Exit (Invalid edge)
10: else if $x$ is of type class then
11: if $SIM(M_i, x) = \{(fc_j, M_i), (fc_j, c_{jl})\}$ then
12: $e_m = (fc_j, M_i)$, $e_{m+1} = (fc_j c_{jl})$
13: else if $x$ is of type descriptive attribute then
14: if $SIM(M_i, x) = \{(fc_j, M_i), (fc_j Da_i)\}$ then
\[ e_m = (f_{c_j}, M_i) f_{c_j} M_i \]

\[ \text{if } SIM(M_i, x) = \{(f_{c_j}, M_i), (f_{c_j}, c_{jl}), (c_{jl} M_i)\} \text{ then} \]
\[ e_m = (f_{c_j}, M_i), \ e_{m+1} = (f_{c_j} c_{jl}) \ e_{m+2} = (c_{jl} M_i) \]
\[ m = m + 1 \]

\[ \text{else} \]
\[ \text{Exit (Invalid edge)} \]

\[ \text{else} \]
\[ \text{Exit(Invalid query type) \{P\}revious steps test possible edges with a measure node and edges are selected as per SIM function output.} \]

\[ p = x \]

Let \( N_p \) be the set of neighbors of \( p \) \{P\}rocessing the remaining edges of the tree.

\[ \text{while } (N_p \neq \emptyset) \text{ do} \]

\[ \text{if } x \text{ is of type class then} \]
\[ \text{CASE } p \text{ is of type fundamental class} \]
\[ K = f_{c_j} \]
\[ \text{CASE } p \text{ is of type class} \]
\[ K = c_{jl} \]
\[ \text{CASE } p \text{ is of type descriptive attribute} \]
\[ K = Da_t \text{ Go to Step 52} \]
\[ \text{CASE } p \text{ none of the above three} \]
\[ \text{Exit(Invalid edge)} \]

\[ \text{if } K = Da_t \text{ then} \]
\[ e_m = SIM(x, p) \]

\[ \text{else if } SIM(p, x) = \{(x, p), y\} \text{ then} \]
\[ e_m = SIM(x, p), \ e_{m+1} = (x, y) \ e_{m+2} = (y, p) \]
\[ y \text{ is a class but } x \text{ and } y \text{ are not the same.} \]

\[ \text{else} \]
\[ e_m = SIM(K, c_{jk}) \text{ testing different possible combinations of edges} \]
\[ \text{if } x \text{ is of Type descriptive attribute then} \]
\[ \text{CASE } p \text{ is of type fundamental class} \]
\[ K = f_{c_j} \]
\[ \text{CASE } p \text{ is of type class} \]
\[ K = c_{jl} \]
\[ \text{CASE } p \text{ is of type descriptive attribute} \]
\[ K = Da_t \text{ Go to Step 68} \]
\[ \text{CASE } p \text{ is none of the above three type} \]
\[ \text{Exit(Invalid edge)} \]

\[ \text{if } K = Da_t \text{ then} \]
\[ \text{CASE } SIM(p, x) = \{(c_{jk}, x)(c_{jk}, K)\} \]
$e_m = (c_{jk}, K), e_{m+1} = (c_{jk}, x)$

CASE $SIM(p, x) = \{(fc_j, K), (fc_j, x)\}$

$e_m = (fc_j, K), e_{m+1} = (fc_j, x)$

CASE $SIM(p, x) = \{(c_{jk}, K), (c_{jn}, x)\}$

$e_m(c_{jk}, K) e_{m+1} = (c_{jn}, x) e_{m+2} = (x, k)$

else if $e_m = SIM(K, c_{jk})$ then

Where $K$ is a class or fundamental class.

else

Exit(Invalid query type)

$m = m + 1$

$p = x$

Let $N_p$ be the set of neighbors of $p$
Appendix C

_TypeSS3 Mapping_

1: Identify $M_i$
2: Let $N_{M_i}$ be the set of neighbors of $M_i$
3: $m = \emptyset$
4: For all $x \in N_{M_i}$
5: if $x$ is of Type Descriptive attribute then
6: if $SIM(M_i, x) = \{(fc_j, M_i), (fc_jDa_t)\}$ then
7: $e_m = (fc_j, M_i)(fc_jDa_t)$
8: if $SIM(M_i, x) = \{(fc_j, M_i), (fc_j, c_{jl}), (c_{jl}, Da_t)\}$ then
9: $e_m = (fc_j, M_i), e_{m+1} = (fc_jc_{jl}), e_{m+2} = (c_{jl}, Da_t)$
10: $m = m + 1$
11: else
12: Exit (Invalid edge)
13: else
14: Exit (Invalid query type)
15: Let $N_p$ be the set of neighbors of $p$
16: while $(N_p \neq \emptyset)$ do
17: if $x$ is of Type Descriptive attribute then
18: if $p$ is of Type Descriptive attribute then
19: $K = Da_t$
20: CASE $SIM(p, x) = \{(c_{jk}, x)(c_{jk}, K)\}$
21: $e_m = (c_{jk}, K), e_{m+1} = (c_{jk}, x)$
22: CASE $SIM(p, x) = \{(fc_j, K), (fc_j, x)\}$
23: \[ e_m = (f_{c_j}, K), e_{m+1} = (f_{c_j}, x) \]
24: \[
\text{CASE } SIM(p, x) = \{(c_{jk}, K), (c_{jn}, x)\}
\]
25: \[
e_m(c_{jk}, K) e_{m+1} = (c_{jn}, x) e_{m+2} = (x, k)
\]
26: \text{else}
27: \text{Exit(Invalid query type)}
Appendix D

TypeSM \( M_1 \) Mapping

1: Identify \( M_i \)

2: Let \( N_{M_i} \) be the set of neighbors of \( M_i \) initialising a variable to assign the neighbour nodes of \( M_i \)

3: \( m = \emptyset \)

4: For all \( x \in N_{M_i} \)

5: if \( x \) is of Type fundamental class then

6: \( \text{if } SIM(M_i, x) = (f_{c_j}, M_i) \text{ then} \)

7: \( e_m = (f_{c_j}, M_i) \)

8: else

9: Exit (Invalid edge)

10: else if \( x \) is of Type class then

11: \( \text{if } SIM(M_i, x) = \{(f_{c_j}, M_i), (f_{c_j}, c_{jl})\} \text{ then} \)

12: \( e_m = (f_{c_j}, M_i), e_{m+1} = (f_{c_j}, c_{jl}) \)

13: \( m = m + 1 \)

14: else

15: Exit (Invalid edge)

16: else

17: Exit (Invalid query type) \{P\}revous steps test possible edges with a measure node and edges are selected as per \( SIM \) function output.

18: \( p = x \)

19: \( m = m + 1 \)
20: Let $N_p$ be the set of neighbors of $p$.

21: while ($N_p \neq \emptyset$) do

22: if $x$ is of type class then

23: if $p$ is of type fundamental class then

24: $K = f_{c_j}$

25: else

26: $K = c_{kl}$ or $K = c_{jl}$

27: if $SIM(p, x) = (K, c_{jk})$ then

28: $e_m = (K, c_{jk})$ if $K$ and $c_{jk}$ are from different lattices, $e_m$ will be a new edge that is added to the KB graph.

29: else

30: Exit(Invalid query type)

31: else

32: Exit(Invalid query type)

33: $m = m + 1$

34: $p = x$

35: Let $N_p$ be the set of neighbors of $p$. 

258
Appendix E

TypeSM\textsubscript{2} Mapping

1. Identify $M_i$
2. Let $N_{M_i}$ be the set of neighbors of $M_i$ 
3. Initialising a variable to assign the neighbour nodes of $M_i$
4. $m = \emptyset$
5. For all $x \in N_{M_i}$
6. if $x$ is of Type fundamental class then
7. $e_m = (f_{c_j}, M_i)$
8. else
9. Exit (Invalid edge)
10. else if $x$ is of type class then
11. if $SIM(M_i, x) = \{(f_{c_j}, M_i), (f_{c_j}, c_{jl})\}$ then
12. $e_m = (f_{c_j}, M_i), e_{m+1} = (f_{c_j}c_{jl})$
13. else if $x$ is of type descriptive attribute then
14. if $SIM(M_i, x) = \{(f_{c_j}, M_i)(f_{c_j}Da_i)\}$ then
15. $e_m = (f_{c_j}, M_i)(f_{c_j}Da_i)$
16. if $SIM(M_i, x) = \{(f_{c_j}, M_i), (f_{c_j}, c_{jl}), (c_{jl}Da_i)\}$ then
17. $e_m = (f_{c_j}, M_i), e_{m+1} = (f_{c_j}c_{jl}) e_{m+2} = (c_{jl}Da_i)$
18. $m = m + 1$
19. else
20. Exit (Invalid edge)
21. else
Exit(Invalid query type) The previous steps test possible edges with a measure node and edges are selected as per SIM function output.

\[ p = x \]
\[ m = m + 1 \]

Let \( N_p \) be the set of neighbors of \( p \) processing the remaining edges of the tree.

\[
\begin{align*}
26: & \text{ while } (N_p \neq \emptyset) \text{ do} \\
27: & \quad \text{ if } x \text{ is of type class then} \\
28: & \quad \quad \text{ CASE } p \text{ is of type fundamental class} \\
29: & \quad \quad \quad K = f c_j \\
30: & \quad \quad \text{ CASE } p \text{ is of type class} \\
31: & \quad \quad \quad K = c_{j'l} \\
32: & \quad \quad \text{ CASE } p \text{ is of type descriptive attribute} \\
33: & \quad \quad \quad K = D a_t \text{ Go to Step 52} \\
34: & \quad \quad \text{ CASE } p \text{ none of the above three} \\
35: & \quad \quad \text{ Exit(Invalid edge)} \\
36: & \quad \text{ if } K = D a_t \text{ then} \\
37: & \quad \quad e_m = SIM(x, p) \\
38: & \quad \quad \text{ else if } SIM(p, x) = \{(x, p), y\} \text{ then} \\
39: & \quad \quad \quad e_m = SIM(x, p), e_{m+1} = (x, y) e_{m+2} = (y, p) \\
40: & \quad \quad \quad y \text{ is a class but } x \text{ and } y \text{ are not the same.} \\
41: & \quad \quad \text{ else} \\
42: & \quad \quad \quad e_m = SIM(K, c_{j'k}) \\
43: & \quad \quad \quad \text{ if } x \text{ is of type fundamental class then} \\
44: & \quad \quad \quad \quad \text{ CASE } p \text{ is of type fundamental class} \\
45: & \quad \quad \quad \quad \quad K = f c_{j'} \{x\}, p \text{ are not necessarily from the same classification} \\
46: & \quad \quad \quad \quad \text{ CASE } p \text{ is of type class} \\
47: & \quad \quad \quad \quad \quad K = c_{k'l} \\
48: & \quad \quad \quad \quad \text{ CASE } p \text{ is of type descriptive attribute} \\
49: & \quad \quad \quad \quad \quad K = D a_t \text{ Go to Step 52} \\
50: & \quad \quad \quad \text{ CASE } p \text{ none of the above three} \\
51: & \quad \quad \quad \text{ Exit(Invalid edge)} \\
52: & \quad \quad \text{ if } K = D a_t \text{ then} \\
53: & \quad \quad \quad e_m = SIM(x, p) \\
54: & \quad \quad \quad \text{ else if } SIM(p, x) = \{(x, p), y\} \text{ then} \\
55: & \quad \quad \quad \quad e_m = SIM(x, p), e_{m+1} = (x, y) e_{m+2} = (y, p) \\
56: & \quad \quad \quad \quad y \text{ is a class but } x \text{ and } y \text{ are not the same.} \\
57: & \quad \quad \quad \text{ else} \\
58: & \quad \quad \quad \quad e_m = SIM(K, c_{j'k}) \\
\end{align*}
\]

Testing different possible combinations of edges
if $x$ is of Type descriptive attribute then 
CASE $p$ is of type fundamental class

$K = fc_j$

CASE $p$ is of type class

$K = c_{jl}$

CASE $p$ is of type descriptive attribute

$K = Da_t$, Go to Step 68

CASE $p$ is none of the above three type
Exit(Invalid edge)

if $K = Da_t$ then
CASE $SIM(p, x) = \{(c_{jk}, x), (c_{jk}, K)\}$

$e_m = (c_{jk}, K), e_{m+1} = (c_{jk}, x)$

CASE $SIM(p, x) = \{(fc_j, K), (fc_j, x)\}$

$e_m = (fc_j, K), e_{m+1} = (fc_j, x)$

CASE $SIM(p, x) = \{(c_{jk}, K), (c_{jn}, x)\}$

$e_m(c_{jk}, K), e_{m+1} = (c_{jn}, x), e_{m+2} = (x, k)$

else if $e_m = SIM(K, c_{jk})$ then
Where $K$ is a class or fundamental class.

else
Exit(Invalid query type)

$m = m + 1$

$p = x$

Let $N_p$ be the set of neighbors of $p$
Appendix F

TypeSM3 Mapping

1: Identify \( M_i \)
2: Let \( N_{M_i} \) be the set of neighbors of \( M_i \) \{I\}nitialising a variable to assign the neighbour nodes of \( M_i \)
3: \( m = \emptyset \)
4: For all \( x \in N_{M_i} \)
5: if \( x \) is of Type Descriptive attribute then
6: if \( SIM(M_i, x) = \{(fc_j, M_i)(fc_j, Da_t)\} \) then
7: \( e_{m} = (fc_j, M_i)(fc_j, Da_t) \)
8: if \( SIM(M_i, x) = \{(fc_j, M_i), (fc_j, c_{jl}), (c_{jl}, Da_t)\} \) then
9: \( e_{m} = (fc_j, M_i), e_{m+1} = (fc_j, c_{jl}), e_{m+2} = (c_{jl}, Da_t) \)
10: \( m = m + 1 \)
11: else
12: Exit (Invalid edge)
13: else
14: Exit(Invalid query type)
15: Let \( N_{p} \) be the set of neighbors of \( p \)
16: while \((N_p \neq \emptyset)\) do
17: if \( x \) is of Type descriptive attribute then
18: if \( p \) is of Type descriptive attribute then
19: \( K = Da_t \)
20: CASE \( SIM(p, x) = \{(c_{jk}, x)(c_{jk}, K)\} \)
21: \( e_{m} = (c_{jk}, K), e_{m+1} = (c_{jk}, x) \)
CASE SIM(p, x) = \{(fc_j, K), (fc_j, x)\}

e_m = (fc_j, K), e_{m+1} = (fc_j, x)

CASE SIM(p, x) = \{(c_{jk}, K), (c_{jn}, x)\}

e_m(c_{jk}, K) e_{m+1} = (c_{jn}, x) e_{m+2} = (x, k)

CASE SIM(p, x) = \{(c_{jk}, x)(c_{pq}, K)\}

e_m = (c_{pq}, K), e_{m+1} = (c_{jk}, x), e_{m+2} = (x, k) \{(x, k)\} (x, k) be the newly added operational edge.

CASE SIM(p, x) = \{(fc_j, K), (fc_p, x)\}

e_m = (fc_j, K), e_{m+1} = (fc_p, x), e_{m+2} = (x, k)

else

Exit(Invalid query type)
Appendix G

Intermediate Schema : An Example

A set of queries belong to various query types are illustrated here. Based on those queries the intermediate schema is generated and shown in figure G.1.

$Q_1$. Sales by Region

$Q_2$. Total sales in quarter 1 of year 1997

$Q_3$. Sales by product by package size

$Q_4$. Sales by shop type

$Q_5$. Sales by region by year

$Q_6$. Average sales by supplier by week

$Q_7$. Sales by clothing size by age
$Q_8$ Average age of customers those who did a transaction for more than $\$100$

$Q_9$ Show the average sales

$Q_{10}$ Show the stock-turn during the year 1999
Figure G.1: An intermediate schema from case study