Semi-Automatic Support for the Design of Multi-Dimensional Databases

Ana María Giral Castillo

Kongens Lyngby 2007
IMM-MASTER-2007-23

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-MASTER: ISSN 0909-3192

Abstract

The context of this project is decision support on the basis of enterprise information systems. Typically decisions are based on information extracted from a collection of relational database systems. But it requires expert knowledge of the relational model to get reports which are well suited for decision support.

A multi-dimensional model of data supports decision making in a much better way, and the aim of the project is to build a tool that helps companies to create their own multi-dimensional model from a collection of relational databases, and to make a web-based environment supporting flexible views of the multi-dimensional model.

The project will involve the definition of requirements, analyse, design, implementation and validation in terms of small examples.

Resumé

The context of this project is decision support on the basis of enterprise information systems. The goal is to create a semi-automatic support for the design of multidimensional databases. The multidimensional databases are databases with an especial organization for the data. This organization helps users in concrete queries of huge amount of data.

This support has to extract de information, with the help of a non-expert user in multidimensional databases, from a relational database. With this information it has to make a new multidimensional database. The new multidimensional database is a reorganization of the data of the relational database.

The new multidimensional database has had a correspondence with the relational one, in order to extract data from the relational to the multidimensional.

At the end there is a system in which is possible to make queries for the decision support.

Preface

This thesis was prepared at Informatics Mathematical modelling at the Technical University of Denmark for acquiring a Master of Science in Engineering in the Erasmus program.

The project was completed in the period of 21st of September 2006 through the 6st of March 2007.

Ana María Giral Castillo

Kongens Lyngby, March 2007

Acknowledgements

I thank my supervisor at DTU, Michael R. Hansen, and my supervisor in my Spanish University J. Ignacio Pulido for their support and help. With their motivation they have helped me further on the academic level.

I would also like to thank my family that allows me to make this thesis in Denmark; my new Danish family, who make me feel at home; and my friends, who support me all this time.

Contents

Abstract **iii**

Resumé **v**

Preface **vii**

Acknowledgements **.ix**

Contents **xi**

1 Introduction **1**
1.1. Problem description 2
1.2. A solution....3
1.2.1 What is the structure of a decision support system? 3
1.2.2 How is the design of a decisions support system? 4
1.3 A simple Example....6
1.4 Thesis Overview 9

2 Multidimensional modelling **11**
2.1 Introduction....11
2.1.1 Data Warehouse....11
2.1.2 Data Presentation 12
2.2 Multidimensional modelling 13
2.2.1 Hypercube....14
2.3 Data Modelling 15
2.3.1 Dimension Table....15

2.3.2 Fact Table 16
2.5 Steps to make a model....18

3 Problem Analysis 21
3.1 Operational source 21
3.1.1 Meta-data of a relational model 22
3.1.2 Meta-data of tables....23
3.1.3 Meta data information about Attributes 23
3.1.4 Meta-data about relations between tables (foreign key relations)...25
3.2 Selection of information from the operational source....26
3.2.1 Measure selection....27
3.2.2 Grain attribute selection....27
3.2.3 Path between the grain attribute and the measure attribute....28
3.2.4 Hierarchies....30
3.3 Multidimensional model 32
3.4 Star Schema....32
3.5 Multidimensional Database creation....34
3.6 Transformation Schema 35
3.7 OLAP Schema....36
3.8 Query for the OLAP server....37
3.9 First load 38
3.10 Showing the results 39
3.11 System analysis....39
3.11.1 Functional system requirements....40
3.11.2 Non-functional system requirements 41
3.12 Activity diagram 42

4 Design of Multidimensional Databases from Relational Databases 43
4.1 Concepts Used....43
4.1.1 Relational model....44
4.1.2 Multidimensional Model....44
4.2 Multidimensional model 45
4.3 Path between grain attribute and measure attribute....47
4.4 Make dimensions 50
4.5 Hierarchy....51
4.6 Semantic information....54
4.7 Star schema 57
4.8 Transformation schema....58

5 System Implementation 63
5.1 Architecture of the system 63
5.2 Object design....64
5.3 The packages....65
5.3.1 Package Relational Model 65
5.1.1 Package Multidimensional Model....66

5.1.2 Package OLAP Model 67
5.1.3 Package Multidimensional Constructor 67
5.2 Implementation 69
5.2.1 Language used 69
5.2.2 Packages and classes implemented 69
5.3 Testing strategy 70

6 Conclusion 71
6.1 Status 72
6.2 Future 72

A Source Code 75
A.1 Package Relational Model 75
A.1.1 Class Attribute....75
A.1.2 Class Table....79
A.1.3 Class FKRelation 82
A.1.4 Class Path....83
A.1.5 Class Paths 87
A.1.6 Class Relational_Model 89
A.2 Package Multidimensional Model 99
A.2.1 Class AttributeMulti....99
A.2.2 Class Semantic 101
A.2.3 Class Dimension 103
A.2.4 Class Fact_Table....110
A.2.5 Class Multidimensional_Model 115
A.3 Package OLAP Model 121
A.3.1 Class OLAP_Schema....121
A.3.2 Class MDX_Query....125
A.4 Package Multidimensional_Construction 127
A.4.1 Class Relational_Schemas 127
A.4.2 Class Multidiemsnional_Schemas....135
A.5 Package databaseConnection....144
A.5.1 Class Metadata....145
A.5.2 Class Postgres 153
A.6 multidimensionalBuilder 157
A.4.1 Class userInteraction....157
A.4.1 Class Multidimensionalbuilder 162

CHAPTER 1

Introduction

Databases have been here for a long time. At the beginning they were just a collection of information stored in a computer. With the time the relational model born to solve some problems joins to the logical schema and to make easy the design of the databases. After that the Entity-Relationships model gave another high level vision for the database designs.

Just now the databases are not used only to control process. One of the most important assets of any organization is its information. This asset is almost always kept by an organization in two forms: the operational systems and the data warehouse systems.

The operational systems are databases for the common operations of an organization. In these databases the data may be modified. The stored information in these systems is frequently in real time. With these modifications an enterprise is losing a huge amount of information that it can not be recovered. Sometimes this information may be very useful in the future. For that reason the organization needs a data warehouse system to kept this information and use it in decisions support systems.

In this thesis we consider the generation of decision support system from relational database systems

1.1. Problem description

A decision support system (DSS) is a computer program application that analyses business data and presents it, so that users can make decisions more easily. It is an "informational application". Typical information that a decision support application might gather and present would be

- Comparative sales figures between one week and the next
- Projected revenue figures based on new product sales assumptions

A decision support system may present information graphically in order to make easy information view. The DSS never takes a decision. It only prepares the information to make it easy to users.

The operational systems are composed by relational model. A relational model is a logic database model that solves very efficiently the design of the operational databases with the inserts and update operations. On the other hand the data warehouse is a database that has to be prepared for the presentation of query's results. It organizes the information for the decisions support.

The requirements for the data warehouse are a multidimensional modelling with OLAP (On Line Analytical Processing) tools. The multidimensional modelling is a data model that it is very close to the way that the data is show. That model is design to be very easy to understand for the final user. OLAP is a solution used in business intelligence. It provides the answer to an analytical query that is based in a multidimensional data model. A typical application is the business reporting. These two technologies allow for complex analytical queries with a better execution time than a relational data model. It is prepared specifically for those analytical queries. An example of multidimensional modelling is shows in the Figure 1.1 and Figure 1.2. The Figure 1.1 shows a representation of a multidimensional model with 3 dimensions: time dimension to store the time of a purchase, market dimension to save the place of a purchase and product dimension to save the products of a purchase. This three dimensions are relate with the number of sales. The fugure 1.2 is shows a client OLAP tool, which permit users browse the information about products.

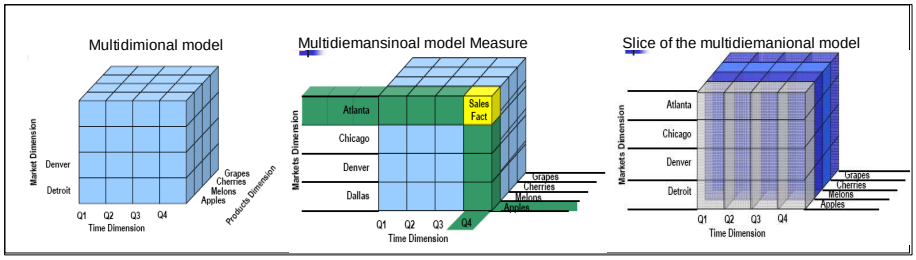
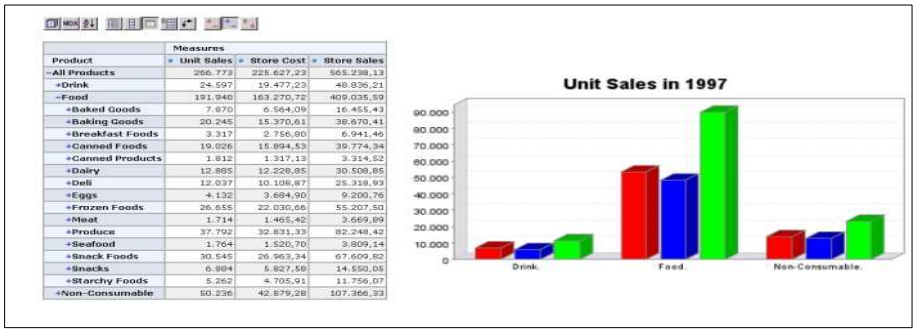


Figure 1.1: Multidimensional model with three dimensions



Product	Measures		
	Unit Sales	Store Cost	Store Sales
-All Products	266.773	225.627,23	565.238,13
+Drink	24.597	19.477,23	48.836,21
-Food	191.940	163.270,72	409.035,59
+Baked Goods	7.870	6.564,09	16.455,43
+Baking Goods	20.245	15.370,61	38.670,41
+Breakfast Foods	3.317	2.756,80	6.941,46
+Canned Foods	19.026	15.894,53	39.774,34
+Canned Products	1.812	1.317,13	3.314,52
+Dairy	12.885	12.228,85	30.508,85
+Deli	12.037	10.108,87	25.318,93
+Eggs	4.132	3.684,90	9.200,76
+Frozen Foods	26.655	22.030,66	55.207,50
+Meat	1.714	1.465,42	3.669,89
+Produce	37.792	32.831,33	82.248,42
+Seafood	1.764	1.520,70	3.809,14
+Snack Foods	30.545	26.963,34	67.609,82
+Snacks	6.884	5.827,59	14.550,05
+Starchy Foods	5.262	4.705,91	11.756,07
+Non-Consumable	50.236	42.879,28	107.366,33

Figure 1.2 View of a client application with OLAP

1.2. A solution

It is difficult for non-experts in Data warehouse the design of multidimensional models. But is the best way to design the data warehouse.

The aim of this project is to build a tool that helps users, who are not experts, to create their own multi-dimensional model from operational databases.

Furthermore, the tool should provide a web-based environment supporting flexible views of the multi-dimensional model with the OLAP tools.

1.2.1 What is the structure of a decision support system?

A *decision support system* (DSS) is more than a database, or a web environment or a simple tool. DSS involve a lot of important and complex elements with a final goal, structure the informal data of the enterprise as the best as possible, access to this data as quick as possible, and save only the necessary data.

The next figure (Figure 1.3) shows the architecture of a simple DSS.

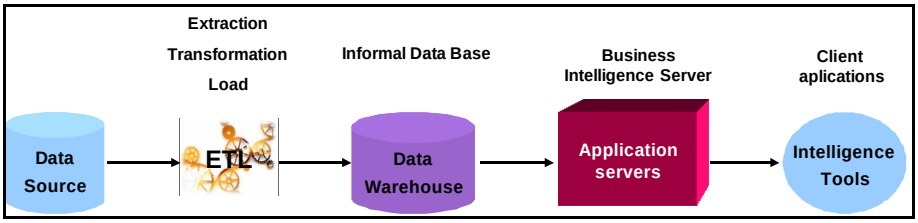


Figure 1.3: Architecture of a decision support system

The elements of the DSS architecture are the next ones:

- The operational database, which is an element older that the DSS. It is the data source of the system. All the necessary data of the informal database is generated from the operational database.
- The extraction, transformation and load (ETL) tool. It is between the Operational database and the informational database. It extract the information from the data source when is needed. The extracted information is transformed and homogenised into the model of the informal database and at the end the transformed data is stored into the informal database.
- The data warehouse. It is the main element of the system, in which is stored all the information needed for the decision support.
- The business intelligence server connects the data warehouse with the client application. It extracts the useful information for a single query and gives it to the client tool.
- The client tool is the graphic tool that shows the information to the user.

1.2.2 How is the design of a decisions support system?

The main activity in the design of a decisions support system is the design of the data model for the data warehouse. The user of the decision support system has to know what he wants to decide and which is the information that need, in order to take decision. This information is restructured into the multidimensional model (Figure 1.1).

It is supposed that the operational database is modelled with an Entity-relationship model and more concrete with a Relational model. The data warehouse is modelled with a multidimensional model.

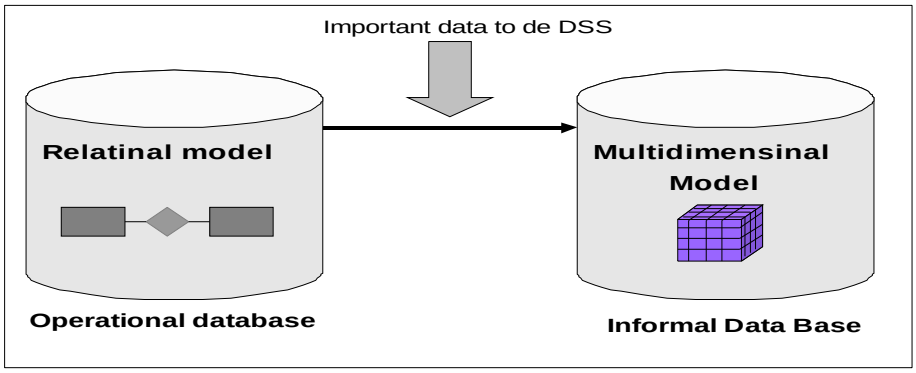


Figure 1.4: Correspondence between operational database and Data Warehouse

The final user has to know what is the important information. He has to select it, and with this information is built a multidimensional model.

Once the multidimensional model is created it has to be transformed into a physical model in order to store the information into a database manager. After that it is needed a transformation schema. It allows the ETL tool the load of data from the operational database to the new data warehouse.

With the multidimensional model, the physical model of the data warehouse, and the transformation model, the logical and physical correspondence between both databases is made it (Figure 1.5). The snowflake model that is show in the Figure 1.5 is a model used to represent the multidimensional model closer to the physical model. It physical model is design for a relational database.

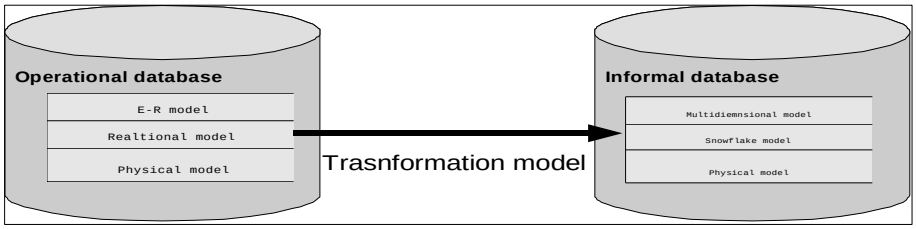


Figure 1.5: Design model for the databases

At this point is needed a schema for the business server that define the multidimensional model and the mapping of this model into the physical one. This is not a standard schema because each business intelligence server has one different for their own technology. With this model saved into the OLAP server, it is possible to see the data of the Data Warehouse like in the Figure 1.2. This figure shows an example. It is comparing the products classified with the hierarchy (drink, food, non-

consumable). This is a simple example with only one concept and different measures for that.

1.2 A simple Example

We have been seen that the information is a very expensive resort for the enterprises and it is very important for the strategic or evaluation decisions. This thesis wants to provide a simple way to convert the operational data into useful information for the decision process whit out an expert help.

A basic and real example could be the next one:
A company has a database in order to register the customer's purchases for the administration department. The manager of the company has decided that he wants to use that information in order to take some decisions about a new product. The schema of the database is in the Figure 1.6.
The company database has information about products, customers and their purchases.
A product is describe by a code, a name, a number of pieces on stock (Available units), and a textual description. It is related with the city that sells is. Each product only can be sold in a city.
A purchase of a customer is described by the code and a date. It is related with a list of products of the purchase with the quantity of each one.
A customer is described by a name, a social security number (ID_Number), an age, a sex and a city. It is related with a city.
A city is described by a name and a number of inhabitants. It is related with a state.
A state is described by a name, a number of inhabitants and a country.

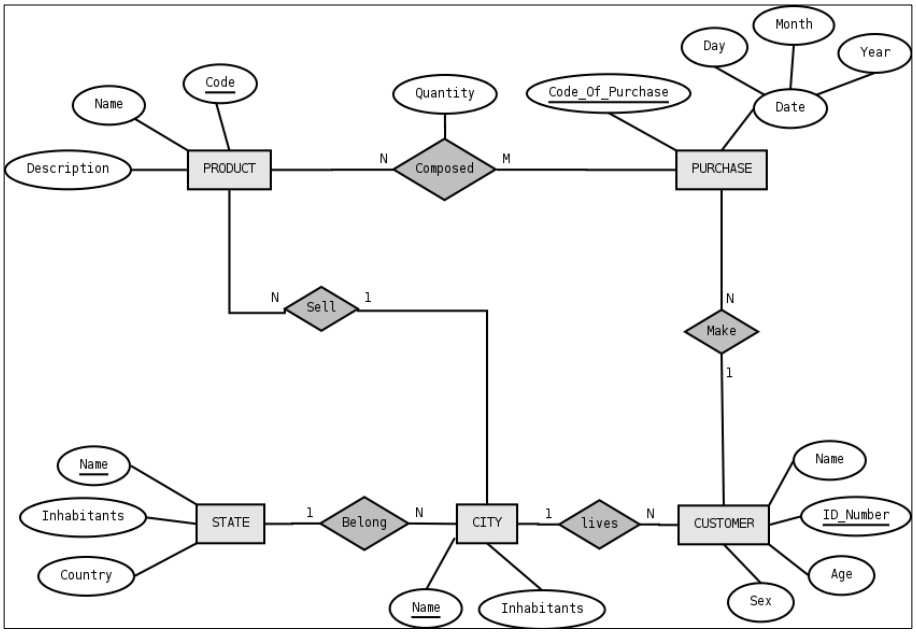


Figure 1.6: Entity-Relational model of the operational database.

Firstly the manager defines the data that he wants to know in order to help him in his decisions. The manager may have several different uses of the information store in the database. One could be to extract a daily and customer's city, based demand analysis of products, in order to find the best promotion of products.

Then with that information we know that the manager wants to know:

- Quantity of product of the purchase.
- Product of the purchases.
- City of the customers
- Date of the purchase.

He knows that if he wants to make a query of data into that operational database, he is going to spend a lot of time trying to understand the data. For that reason he tries this tool in order to make a new database. That database will be prepared for the especial query that he need.

The purchase is a tool supporting the construction of a multidimensional database from a given relational database. In the following i the more general steps of the construction of this basis example.

- Firstly the tool has to connect to the operational system. This operational system is a relational database. The tool extracts the relational model from the

operational database. The relational model consists in tables, attributes, foreign key relations between tables, and attributer's restrictions (PRIMARY KEY, UNIQUE, FOREIGN KEY).

```
PRODUCT (Code, name, description, available_units)
Primary_Key (Code)

PURCHASE (Code, date, customer)
Primary_key (Code)
Foreign_key (customer) references CUSTOMER.name

PRODUCT_PURCHASE (purchase_code, product_code, quantity)
Primary_key (purchase_code, product_code)
Foreign_key (purchase_code) references PURCHASE.code
Foreign_key (product_code) references PRODUCT.code

CUSTOMER (name, age, city)
Primary_key (name)
Foreign_key (city) references CITY.name

CITY (name, inhabitants, state)
Primary_key (name)
Foreign_key (state) references STATE.name

STATE(name, inhabitants, country)
Primary_key (name)
```

Figure 1.7: Relational model extracted from the operational database

- Once the relational model is extracted the tool has to guide the user in the selection of the important information in order to make the new database.
  For instance if the user wanted to know the product that he sell, the place of the client and the date. The multidimensional model represented by a hypercube is in the next figure (Figure 1. 8):

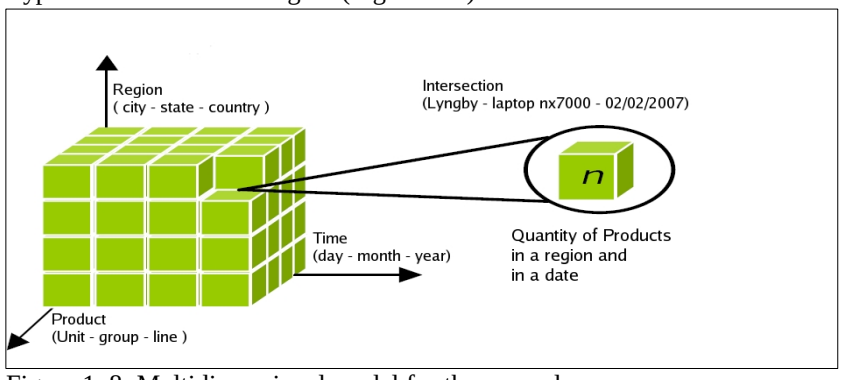


Figure 1. 8: Multidimensional model for the example.

- The information is selected. Then the tool has to make all the necessary data models and transformation model in order to create the new multidimensional database and the connexion with the business intelligence server.
- At the end it will be provide a web environment as a client tool (Figure 1.2).

1.4 Thesis Overview

The structure of the thesis is as follows:

Chapter 2, Multidimensional modelling. This chapter introduces the multidimensional modelling and concepts such as data warehouse, hypercube and schemas.

Chapter 3, Problem Analysis. This chapter explains how a multidimensional database is constructed from a relational one, on the basis of the example in section 1.3.

Chapter 4, Design of multidimensional databases from Relational Databases. This chapter develops the theory and algorithms for the construction o a multidimensional database from a relational database, which is the main contribution of this thesis.

Chapter 5, System Implementation. This chapter explains some important details about the object system design, the implementation of the tool and the testing methodology of the testing.

Chapter 6, Conclusion. This chapter makes a discussion about the multidimensional model, how this thesis resolves the problem, and the actual status of the solution.

CHAPTER 2

Multidimensional modelling

2.1 Introduction

Ralph Kimball is one of the two more important people in the Data Warehouse and multidimensional modeling. He says in book “The Datawarehouse Toolkit”:

“Dimensional modelling is a new name for an old technique for making data-bases simple and understandable. In case after case, beginning in the 1970s, IT organizations, consultants, end users, and vendors have gravitated to a simple dimensional structure to match the fundamental human need for simplicity”[1]

2.1.1 Data Warehouse

Bill Inmon is widely recognized as the father of data warehousing. In the book “Using the Data Warehouse” he defines the term *Data Warehouse* as:

“A Data Warehouse is a subject-oriented, integrated, time-variant, non-volatile

1 The Datawarehouse Toolkit The Complete Guide to Dimensional Modelling 2Nd Ed - Ralph Kimball - Margy Ross - John Wiley

collection of data in support of management's decision-making process."[2]

A data warehouse is a database that contains a collection of historical data. The term Subject-oriented means that the data is organized for themes. The term "time-variant" means that there is a time component in the data of the Data Warehouse. So that reports can be produced showing changes over time. The term "Non-volatile" means that data in the database is never over-written or deleted. It is retained for future reporting. The term "Integrated" means that the database contains data from most or all of an organization's operational applications, and that this data is made consistent.

The data is organized, integrated and prepared to a decision support system.

2.1.2 Data Presentation

The data presentation area is where data is organized, stored, and made available for direct querying by users, report writers, and other analytical applications.
All the users of a data warehouse see and touch the data via data access tools. The presentation area is typically referred as a series of integrated data marts. A *data mart* is a wedge of a data warehouse. It is the most simplistic form. A data mart presents the data from a single query while a data warehouse is the set of different data marts. In the Figure 2.1 it is possible to see that the data warehouse of an enterprise is composed by each single data mart. One of them may by the one how resolve the necessity that is exposing in the Charter 1 section 1.3. The manager of an enterprise wanted to make a multidimensional model with the information about products, cities of the customers, who makes the purchase, and date of the purchase. The other two data marts of the enterprise there solve different problems. They contain different information. The group of all these data mart of the enterprise is the Data Warehouse
The dimensional modelling is the most viable technique for delivering data to data warehouse users.

2 "Using the Data Warehouse" Inmon, W. H.

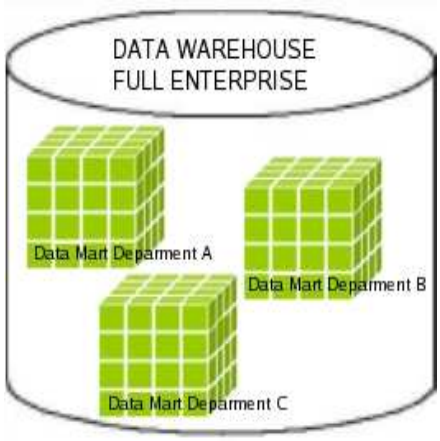


Figure 2.1: Data Marts of an organization

2.2 Multidimensional modelling

Ralph Kimball in the book “The Datawarehouse Toolkit” says:

"The central attraction of the dimensional model of a business is its simplicity.... that simplicity is the fundamental key that allows users to understand databases, and allows software to navigate databases efficiently."[3]

The multidimensional model is a logical data model. The necessity of the multidimensional model comes from the enterprises. They have a huge amount of data, but they can not access it. They use the same search patterns day by day and the data usually is not prepared for these queries.

The goals of the multidimensional modelling are:
- It has to make easy the access to the information.
- The content of the databases has to be understandable for the users.
- The information has to be consistently.

The Multidimensional model represents a measure that depends on a set of dimensions, which provides the context for the measure. It can be represented by hypercube, that are the graphic way and conceptual model, or by schemas in a logical model.

3 *“The Data Warehouse Toolkit” Ralph Kimball*

2.2.1 Hypercube

A *hypercube* is the graphic way to represent a multidimensional model. It is used for the conceptual design of the multidimensional model.
A hypercube is composed by dimensions. A dimension contains information. Each one has different levels to represent this information. The intersection of the dimension defines the measure of the cube. This is what the user wants to know. In the example that is introduce in the chapter 1 into the section 1.3. A manager of an enterprise wanted to make a multidimensional model with the next information:

- Quantity of product of the purchase.
- Product of the purchases.
- City of the customers.
- Date of the purchase.

The Figure 2.2 shows a hypercube for the example. In the axis is represented the Product. In the product dimension is store every single product with the information of the group and the line of the product.
In the x axis is represented the time where the product were sold. The time has the information about a day, a month and a year.
In the y axis is represented the Region of the Customer. The region of a customer is composed by a city, the state of the city and the country of the city.
The intersection of all these dimensions is a point of the hypercube where is save the quantity of a product that was purchased in a city and in a concrete date.

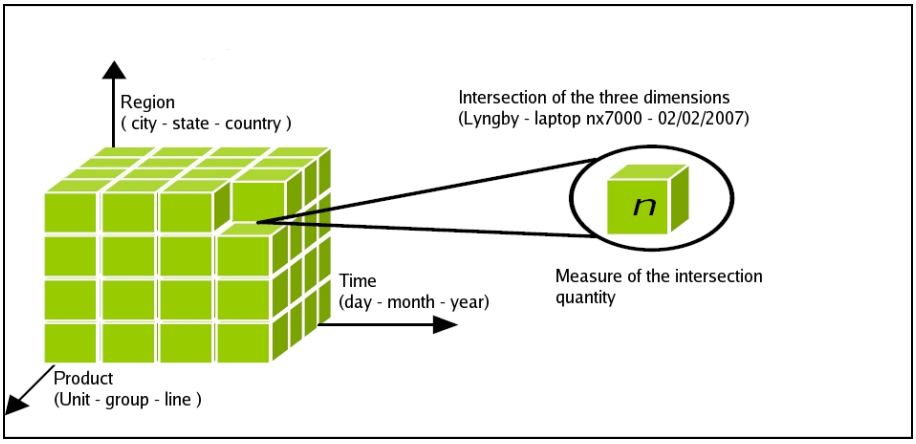


Figure 2.2: Hypercube for a multidimensional model

2.3 Data Modelling

The multidimensional model is defined by two elements: the dimension tables and the fact table.

2.3.1 Dimension Table

The dimensions define domains of information by attributes, it contains a set of unique values that identify and categorize data. An example of domains may be product, time, geography, customer or others. The members of a dimension are grouped in a hierarchy way.

A *hierarchy* is a way to organize data at different levels of aggregation. One advantage of the hierarchy is the capability of accessing the model at different levels of abstraction. An *abstraction* of an object denotes its essential characteristics which distinguish it from all other objects. It may be defined as a mental ordering imposed on the environmental model depending on the goal or task of the user. It may be understood as a selection of a set of attributes, objects, or actions from a much larger set of attributes, objects, or actions according to certain criteria determined by the above-mentioned goal or task.
For example in the multidimensional model present in the Figure 2.2 one of the dimensions is the region dimension. It has a city, a state, and a country. Its attributes may represent a hierarchy. The low level of the hierarchy is city. All the cities with the same country may be aggregated in a different level. Then there are different levels of abstraction for the data. The attribute country allows us to make a new level of abstraction.

In viewing data, analysts use dimension hierarchies to recognize trends at one level, drill down to lower levels to identify reasons for these trends, and roll up to higher levels to see what affect these trends have on a larger sector of the business.

- **Attributes**

  Attributes represent a single type of information in a dimension. For example, year, month or day are attributes in the Time dimension. Its may be related via a hierarchy.

- **Grain Attributes**

  The grain attribute is the most finely grained data in a dimension, the granularity of the fact table. The grain conveys the level of detail associated with the dimension. It is possible to define as the atomic data of a dimension. It provides the maximum analytical flexibility, because it can be constrained

and rolled up in every possible way.

Example grain declarations include:

- An individual line item on a customer's retail sales ticket as measured by a scanner device
- A line item on a bill received from a doctor
- An individual boarding pass to get on a flight
- Date Dimension

The date dimension is guaranteed to be in every data mart because virtually every data mart is a time series. There are a lot of attributes that can be interesting in the date dimension.

2.3.2 Fact Table

The fact table is the primary table in a dimensional model where the numerical performance measurements are stored. It is the information that is stored in the hypercube. For instance in the Figure 2.2 the quantity of a product that was purchased in a city and in a concrete date.

1.2 Schemas for store a multidimensional model

There are different ways to store the multidimensional model physically. In the context of this thesis we speak about the relational way the represent a multidimensional database. There are two different logical schemas in order to represent the multidimensional model in a relational way: the star schema and the snowflake schema.

2.4.1. Star schema

A star schema has a large "fact" table in the centre, with multiple "dimension" tables surrounding the fact table. There is a one-to-many relationship between each dimension table and the fact table.

In the Figure 2.3 there is an example of a star schema for the hypercube of the Figure 2.2. There is a table for the dimension Region. It contains all the information of this dimension. There is other table for the dimension Product. It contains all the information needed for the product. There one more table for the dimension time. It contains the attributes necessaries to represent the time.
In the centre of this dimension tables is the fact table. The fact table contains a referenced for each dimension and the attribute that is store in the intersection of the

hypercube. It is the quantity of a product that was purchased in a city and in a concrete date.

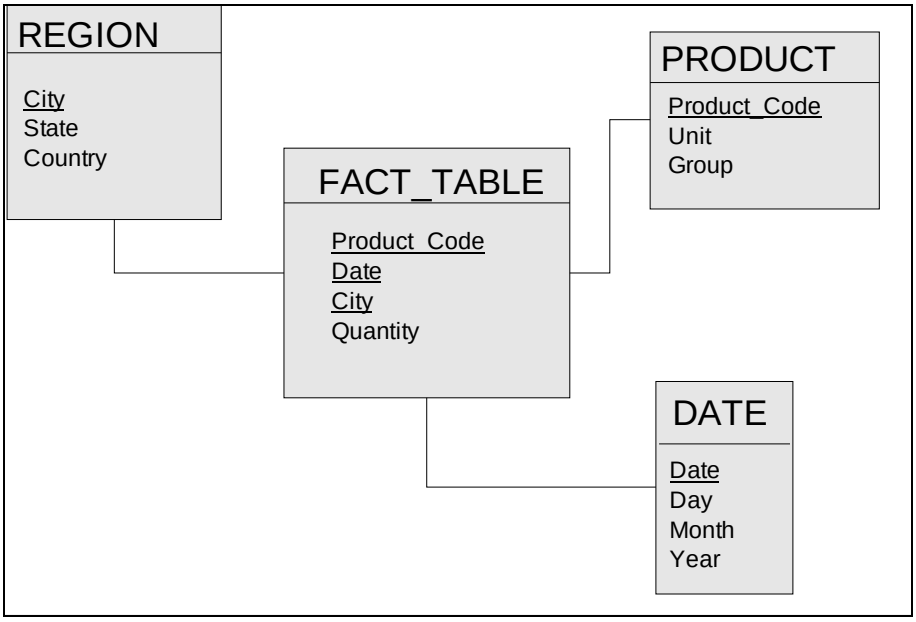


Figure 2.3: Star schema

2.4.2. Snowflake schema

The snowflake schema may be interpreted as an extension to a star schema. It is a start schema with the dimension tables normalized.
The start schema has redundancy because only is used a table to represent a dimension. Then the table is usually de-normalized. The redundancy is often a sign of bad relational design. In order to avoid the redundancy has to be used the snowflake model.
The snowflake model is used to represent hierarchies in a normalized way. Each dimension table has to be normalized with the information of the hierarchy that the dimension contains.
A hierarchy attribute is an attribute that is the grain attribute or an attribute that represent a level of abstraction. Then for each hierarchy attribute there is a level in the hierarchy. This level is represented as a table with the description attributes of each hierarchy attribute. The multitude of snowflake tables makes for a much more complex presentation.

The simplicity of the model is one of the primary objectives of a denormalized dimensional model.

In Figure 2.4 there is an example of a snowflake model for the hypercube of the

Figure 2.2. In we compare with the star schema, the dimension DATE continue being the same. The dimension product is divided into to tables in order to normalize this. Every product of the same group has the same line. Then the attribute line depend if the attribute group. In order to normalize it is necessary to create a new table. With the dimension Region it is happening the same. This dimension has to be normalized and then it is needed to tables to represent the dimension instead of one.

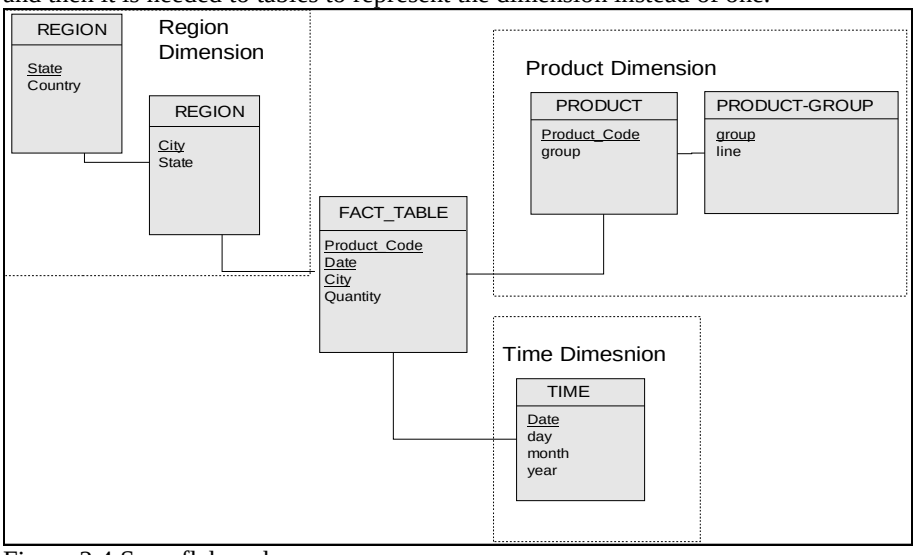


Figure 2.4:Snowflake schema

2.5 Steps to make a model

The traditional methodology to create a multidimensional model is composed by four steps.

Four-Step Dimensional Design Process:

- Requirements definition
  Firstly is needed the business requirements definition.
- Grain
  Declare the grain of the business process. It means specifying exactly what an individual fact table row represents. You can declare higher-level grains for business process that represent an aggregation of most atomic data. But, as soon as we select a higher-level grain, we are limiting ourselves to this detail level. The detail will disappear.
- Dimensions
  Once the grain has chosen the dimensions it determinate the dimensions of the model.

For each grain attribute selected a new dimension will be necessary to represent the hierarchy of that grain attribute.

- Facts
  Identify the numeric fact of the fact table. Are determinate by answering the question “What are we measuring?”

Identifying the requirements of the multidimensional model, the dimensions and the measure there are all the necessary elements to make the model.

2.6 OLAP (On Line Analytical Processing)

On Line Analytical Processing (OLAP) means analysing large quantities of data in real-time.
The term 'on-line' implies that even though huge quantities of data are involved, typically many millions of records, occupying several gigabytes. The system must respond to queries fast enough to allow an interactive exploration of the data.

The OLAP analytic engine supports the selection and rapid calculation of multidimensional data. It deals with data in bulk, and operations are generally read-only
The OLAP server understands how data is organized in the database. It has special functions for analysing the data.
The typical operations are aggregations and data reporting of multidimensional cubes. The most used are the next ones:

- Drill-up: It summarizes data by climbing up hierarchy.
- Drill down: It reverses of roll-up from higher level summary to lower level summary or detailed data, or introducing new dimensions. The example in the Figure 2.5 uses the data of the example. It takes all the data from the purchases during the year 2006. The figure shows how makes drill drown and drill up the hierarchy of the region dimension.
- Slice and dice: projection and selection of the attributes. In the Figure 2.6 there is an example of a slice in the hypercube of the example. It makes a slice of the all the data about a unique product. Then the result set y a subset of the hypercube.
- Pivot: It *r*eorient the cube, visualization, 3D to series of 2D planes. It changes the visualization of the dimensions, which dimension is in each axis.

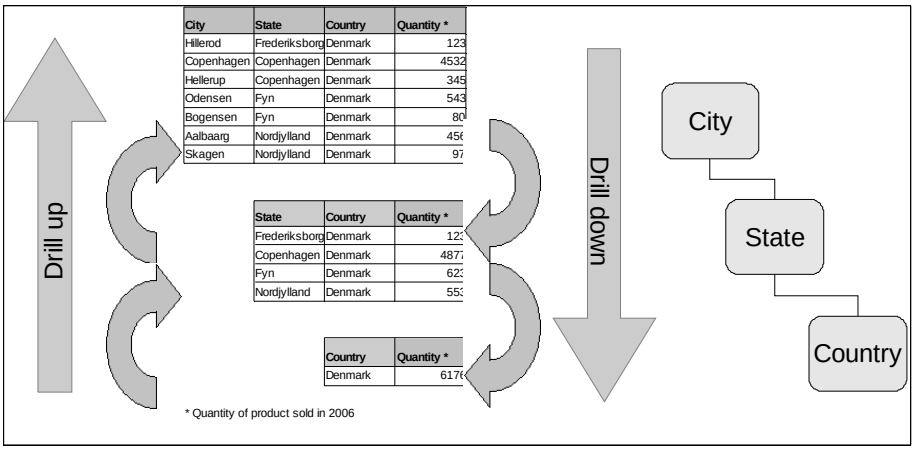


Figure 2.5: Drill up and drill down

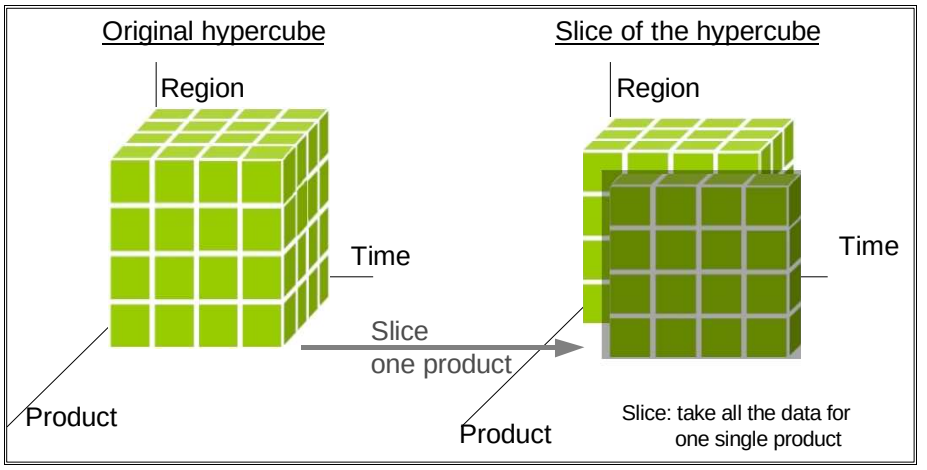


Figure 2.6: Slice of a Single product

CHAPTER 3

Problem Analysis

This chapter explains how a multidimensional database is constructed from a relational one, on the basis of the example explained in section 1.3.

It is going to be resolved the problem of the construction of a multidimensional database from a relational one. It is going to be explained all the necessaries steps in order to construct a multidimensional database. These steps describe a model that resolves the problem. The model has a few restrictions, for instance the operational database. These restrictions are explained along the chapter.

In each step is showed the result in the example explained in section 1.3.

At the end of the charter, after the analysis of the problem, is made a system analysis.

3.1 Operational source

Before start the process the user has to know what is the operational source that it is needed for the construction of the multidimensional database. The Operational Source contains the operational data of an enterprise.

There are two restrictions that the database of the operational source has to keep. It

has to be a relational database and it has to be in third normal form. It has to be relational because the model is prepared for a relational database. It has to be in Third Normal Form (3FN) because the model extracts information of the database supposing that it is in 3FN. A database is in 3FN when:
A database is First Normal Form if each column contains only a single value and each row contains the same columns.
A database is in Second Normal Form if it is in First Normal Form and non-key attribute in a table must be functionally dependent upon the primary key of the table.
A database is in Third Normal Form if it is in Second Normal Form and all attributes that are not dependent upon the primary key are eliminated.

These restrictions are necessary for the correct creation of multidimensional model. Along the chapter it will be explained its necessity.

Other question is that the tool needs to have access to the database manager that contains the relational database to extract the model. The user has to give this information to the tool. The required information is the name of the database, the IP address of the database server, the port that the server is using, a user of the database manager, and the password of the user.

In our example we have a database in a PostgreSQL management named "customersInformation", which allow users to make connexions in this server http://companyA.org:5432 with user "user" and password "pwdUser".

With this information the tool can connect to the specific database.

3.1.1 Meta-data of a relational model

The relational schema of the database is necessary in order to make the data extraction and the design of the multidimensional database.

To know the schema of the relational database it is needed the extraction of this information from the database manager. It is needed the information about how the data is organized, the meta-data. The *meta-data* is the information about the tables, the attributes and the relation between them. If the relational database does not have this information, the tool can not create the model.

The tables are the physical tables of the database, each table that belong to the relational model.

The attributes are the attributes that contains each table. The relations between tables

are the foreign key relation that joins two tables.

This process in which the meta-data is extracted is something transparent to the user

3.1.2 Meta-data of tables

The first meta-data that is extracted is the information about the tables. All the information that is needed is the name of each table of the relational database.

In the example for the PostgreSQL database the query will be the next one :

```
SELECT table_name FROM information_schema.tables
WHERE table_schema = 'public';
```

The result of the query:

```
table_name
"STATE"
"CITY"
"CUSTOMER"
"PURCHASE"
"PRODUCT_PURCHASE"
"PRODUCT"
```

All the tables of the database have been extracted, but is needed more information.

3.1.3 Meta data information about Attributes

Once the tables are extracted, is necessary to extract the information about the attributes that contains each table. For each attribute is necessary to know the name, the type and the restriction of the attribute.

The name of an attribute is the name of the column that contains this attribute in the relational database.

The type of an attribute is the SQL data type of the attribute in the relational database, the types contemplated in the tool are the SQL standards ones : CHAR, VARCHAR, LONGVARCHAR, NUMERIC , DECIMAL, BIT, SMALLINT, INTEGER, REAL, FLOAT ,DOUBLE, BINARY, VARBINARY, LONGVARBINARY, DATE, TIME, TIMESTAMP.

The restriction of an attribute is the restriction that it has into the relational database.

It restriction can be one of these four:
PRIMARY_KEY: The attribute is the primary key of the table where it is stored.
FOREIGN_KEY: The attribute is belonging in a foreign key relation. It means that this attribute is referencing other attribute that is in other table. This referenced table is different than its table.
UNIQUE: Two different rows of the table, that contains the attribute with the UNIQUE restriction, can not have the same value for this attribute.
NONE: The attribute do not have any of the previous restrictions.

At the beginning only is extracted the information of the name and the restriction of an attribute in order to make the model. The type of the attribute is extracted if this attribute has to be in the multidimensional database. Because otherwise it is not needed.

In the example the query to know the attributes of the table “CUSTOMER” the query is :

```
SELECT column_name FROM information_schema.columns
WHERE table_schema = 'public' AND table_name = 'CUSTOMER'
```

The result of the query

```
column_name
"Sex"
"City"
"Name"
"Age"
"ID_Number"
```

To now the restrictions of the attributes there are a single query for each attribute. For the attribute “ID_NUMBER” the query is :

```
SELECT constraint_type
FROM information_schema.table_constraints
WHERE table_schema = 'public' AND
(constraint_type = 'PRIMARY KEY' OR constraint_type = 'UNIQUE')
AND constraint_name IN ( SELECT constraint_name
FROM information_schema.constraint_column_usage
WHERE table_schema = 'public'
AND table_name = 'CUSTOMER'
AND column_name = 'ID_Number');
```

The result of the query is :

```
constraint_type
"PRIMARY KEY"
```

The information of the FOREIGN KEY is extracted when the Foreign key Relations are extracted.

In order to extract the type of an attribute IT is a single query for each attribute.

```
SELECT data_type FROM information_schema.columns
WHERE table_schema = 'public' AND table_name = 'CUSTOMER' AND
column_name = 'ID_Number';
```

The result of the query is :

```
data_type
"numeric"
```

3.1.4 Meta-data about relations between tables (foreign key relations)

A foreign key relation is a relation between two attributes of two different tables. One of the attributes references the other one. They are the same attribute. The difference between the two attributes is that the referenced has to be a primary key in the table where it is.

In the example the extraction of the foreign key relations is done with a SQL VIEW and a query:

```
Create view FK as (
SELECT constraint_name
FROM information_schema.table_constraints
WHERE table_schema = 'public' AND
        constraint_type = 'FOREIGN KEY' )

SELECT KCU.table_name, KCU.column_name,
        CCU.table_name, CCU.column_name
FROM information_schema.constraint_column_usage CCU,
        information_schema.key_column_usage KCU, FK
WHERE FK.constraint_name = CCU.constraint_name AND
        FK.constraint_name =KCU.constraint_name
```

The result for this query is the set of foreign key relations between table of the database:

table_name	column_name	table_name	column_name
CITY	State	STATE	Name
CUSTOMER	City	CITY	Name
PRODUCT	City	CITY	Name
PURCHASE	Customer	CUSTOMER	ID_Number
PRODUCT_PURCHASE	Purchase_Code	PURCHASE	Code
PRODUCT_PURCHASE	Product_Code	PRODUCT	Code

3.2 Selection of information from the operational source

At this point the user has to have a clear idea about what he wants to know. On the other hand he has to know the information that he needs, in order to create the multidimensional model.

There are two different methodologies to make the selection of the information from the relational database. Both of them have been studied.

The first methodology allow user to select all the attributes at the beginning of the process. Once the attributes are selected the tool asks user which attributes are the grain attributes (the atomic data of a dimension). The user has to select its. With this information the attributes are grouped into dimensions and into fact table.

During the process of grouping the attributes into dimensions, it is possible that for some attributes exist ambiguous information and the tool do not know how to do. In this case the user interaction is needed. In these cases is the user how choose the correct dimension for the attribute.

The selection of all the attributes at the begging has a problem; the problem is that the user has to know every thing at the begging. He has to know what he really needs to make the multidimensional database without help. On the other hand the user may not understand which information can be used. A common situation in this point is that the user is lost and then he does not continue with the process.

For that situation it has been proposed one other solution. A more guided selection of attributes. It makes easier the process to the user. The process is divided in different steps. Each step is explained next.

3.2.1 Measure selection

One thing that is very clear for the user is what he wants to save in the multidimensional database. It is the purpose of the multidimensional database, the measure. For this reason it will be the first thing that the user has to select. This selection is made form the relational model.

The user is asked “what you want know?”

In the example the user wants to know the quantity of product sell. This information is store in the table PRODUCT_PURCHASE in the column quantity. This is the attribute that the user has to choose as measure attribute.

Once the measure attribute is selected the user has to decide which is the aggregate operation for this measure attribute. It means that it is necessary to define what will be the operation to aggregate the measure attribute in the drill-up and down operations. There are different possibilities. The technologies used restrict these operations to the next ones: "sum", "count", "min", "max", and "avg".
The sum aggregator means that the measure attribute has sum in the aggregations of rows.
The count aggregator means that the rows with the same content have to be count.
The min aggregation means that for the rows with the same content, without measure, only is selected the one that has the less value in the measure.
The max aggregation means that for the rows with the same content, without measure, only is selected the one that has the high value in the measure.
The avg aggregation means that for the rows with the same content, without measure, the result is a row with the measure the arithmetic average of the column measure.

In the example the user wants to know the quantity of product. The aggregator operation is the sum.

3.2.2 Grain attribute selection

After that measure selection the user has to know which is the really important information to be stored in the model. What the user want to relate with the measure in order to make the comparison. The user is asked “What you want to relate with the measure attribute?” Now it has to choose a table. And with this table it has to choose

a grain attribute or more than one.

In this example the user wants to know the products of a purchase, the place where the products were purchased and the date of the purchase.

The selection of the tables will be the next one :

- Table Product for the product of the purchase. Inside this table the user has to select the attribute Name or the attribute Code. It selects the attribute Code
- Table City for the city where the customer lives. In this table the user has to select the attribute name.
- Table Purchase for the date of the purchase. In this table the user has to select the Attribute Date.

3.2.3 Path between the grain attribute and the measure attribute

The fundamental assumption of the relational model is that all data is represented as mathematical n-ary relations. An n-ary relation being a subset of Cartesian product of n domains. In order to related to different attributes is needed a path between them. Since now the term Path refers a set of relations that made this connection possible. There are some restrictions in this set of relations:

- A path star in an attribute and finish in other.
- A path has a direction.
- A path can not contain a cycle. It means that a new relation in the path can not finish in a table that exists in one of the relations of the path.

In the next chapter is explained more deeply the path concept.

Once a grain attribute and the measure attribute are selected is necessary to know with is path that join them. This path determinates the context of the grain attribute.

In the example we have the attribute Name of the table CITY selected. If we look for the path between this attribute and the measure (the attribute quantity of the table PRODUCT_PURCHASE) we found two different paths (Figure 3.1).
The path 1 starts in the table CITY, which contains the grain attribute. It goes to the table CUSTOMER. After it goes to table PURCHASE and it finish in the table PRODUCT_PURCHASE, which contains the measure attribute quantity. The city that is referred with this path is the city of the customer that makes the purchase.
The path 2 starts in the table CITY, which contains the grain attribute. It goes to the

table PRODUCT and it finishes in the table PRODUCT_PURCHASE, which contains the measure attribute quantity. The city that is referred with this path is the city where the product is sold.

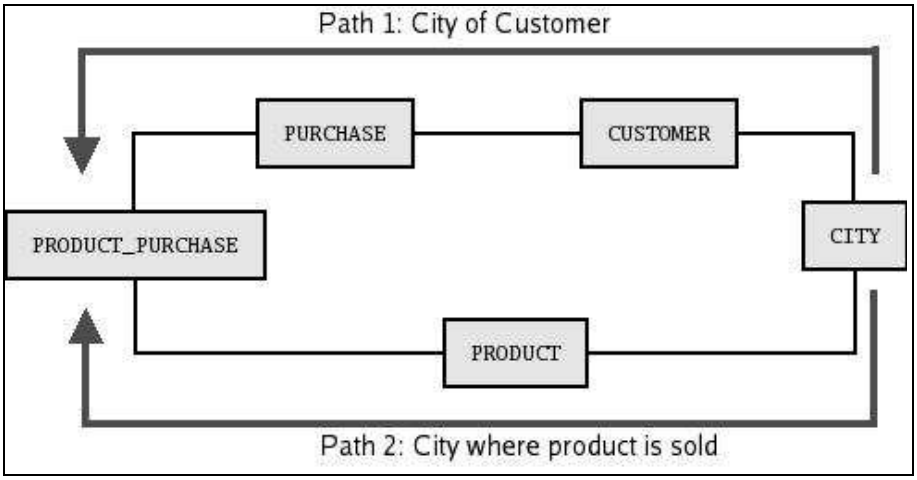


Figure 3.1 : Paths between the grain attribute Name in table CITY and the measure

There is a difference between the both paths. It is a context difference. For that reason the tool can not purpose a path between the grain and the measure to the user. He has to select between both paths. The decision depends of the requirements for the multidimensional model.

The tool made possible the selection of both paths. If the user takes both paths then both concepts of cities are selected, the city where the product is sold and the city of the Customer.

If there are only one path between the grain attribute and the measure attribute, this process is transparent for the user.

In the example the grain attributes are :

- Attribute Code of table PRODUCT. This attribute is a PRIMARY key of the table PRODUCT. For this attribute there is only a possible path from itself to the measure attribute.
- Attribute Name of Table City. For this attribute there are two possible paths. These are the paths explained before. Then the user has to choose the path 1, city of customer.

- Attribute Date of Table Purchase. For this attribute there is only a possible path from itself to the measure attribute.

3.2.4 Hierarchies

For each grain attribute the tool study the possibilities. These possibilities are the next ones:

If the grain attribute has a NONE restriction, this attribute has no hierarchies. The reason will be explained in the next chapter.

If grain attribute has a different attribute restriction than NONE. Then for this attribute the tool look for a hierarchy that start in himself.
A hierarchy is way to group the information in different levels of abstraction. The hierarchy concept is defined in the Chapter 2 Section 2.4.2 17
In the example there are the next ones hierarchies:

- A hierarchy for the product (Figure 3.2 Hierarchy product -> city -> Start) it shows the tables that are involve in this hierarchy. The tables that are involved in this hierarchy are product, city and state. The products that are in a database could e group in different levels of abstraction. The less level is with the atomic information of a product; the next one is grouping of the product with the same city into one other level, the city of product. At the end there is one other big grouping level, all the product with the same state.

- A hierarchy for the customer. (Figure 3.2 Hierarchy product -> city -> Start) It shows the tables that are involve in this hierarchy. They are the table customer, city and state. It means that it is possible to group the customers into different levels of abstraction. The less level is each customer, the next one is grouping of them by the city into one other level, customer by city. At the end there is one other big grouping level, all the customers with the same customers by state.

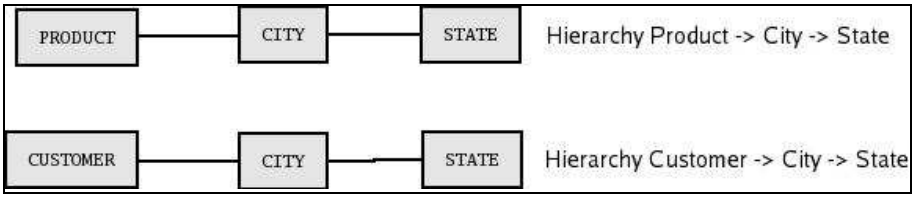
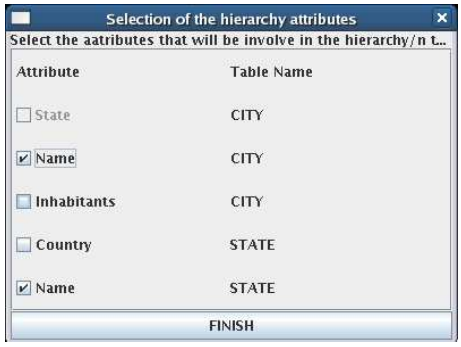


Figure 3.2: Hierarchies in the example

Then the tool looks for these hierarchies and suggest the ones that is involve the attribute, and if it is an attribute without possible hierarchy the program has to told to the user. The user has to select the hierarchies and the attributes for each one.

In the example the grain attributes are :

- Attribute Code of table PRODUCT. This attribute has a possible hierarchy. For our database we do not need this information and the hierarchy is not selected.
- Attribute Name of Table City. For this attribute there is a hierarchy. The hierarchy starts in the table CITY and finish in the table STATE. The user chooses this Hierarchy and the Attributes that are in the next figure :



- Attribute Date of Table Purchase. There are no possible hierarchies.

3.3 Multidimensional model

While the user has been selecting all the information, the multidimensional model was being built. The concrete building process that is needed to make the multidimensional model is explained in the next chapter.

After the user selection the tool present the multidimensional model to the user. The information that the user can see is:

- Fact Table: It only has the measure attribute
- Dimensions: Each dimension is show like a tree in order to show the hierarchies. The root of the row is the name of the hierarchy. The soon is the grain attribute.

In the example the user can see the result of the dimensions and the fact table as the next figure :

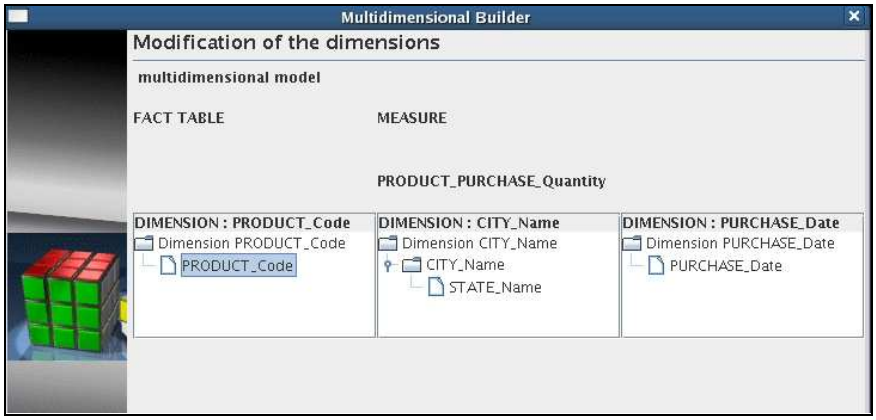


Figure 3.3: Multidimensional model created

3.4 Star Schema

Once the multidimensional model is created, is needed to representation of the model close to the physical model. In this thesis is used a star schema. In a star schema each dimension is in a single table and the fact table is in the centre. In a multidimensional database the huge amount of data is in the fact table.

The star schema for the multidimensional model is like in the Figure 3. **4**. In the centre is the fact table. Around the fact table there are a table for each dimension. The fact Table contains the measure attribute and the primary key of each dimension

table.
Each dimension table has a primary key and the rest of the attributes all in the same table.

Star schema for the example:

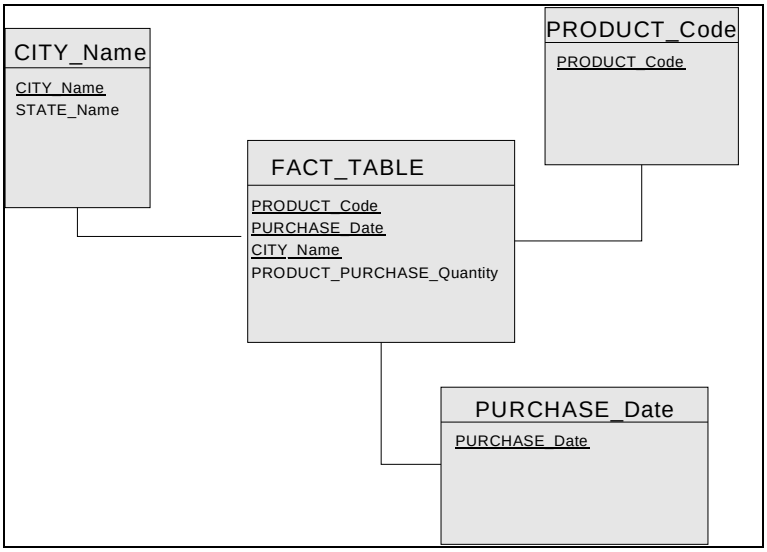


Figure 3. 4: Star Schema of the example

The star schema is a graphic representation, but it is needed a physical schema. For that reason this model is mapped into the standard SQL format.

The SQL schema for the multidimensional database of the example is composed by four tables. One table for the fact table and one for each dimension. The file is :

CREATE TABLE "PRODUCT_Code" (
"PRODUCT_Code" integer,
CONSTRAINT PK_PRODUCT_Code **PRIMARY KEY**("PRODUCT_Code"));

CREATE TABLE "CITY_Name" (
"CITY_Name" character varying,
"STATE_Name" character varying ,
CONSTRAINT PK_CITY_Name **PRIMARY KEY**("CITY_Name"));

```
CREATE TABLE "PURCHASE_Date" (
"PURCHASE_Date" date,
CONSTRAINT PK_PURCHASE_Date PRIMARY KEY("PURCHASE_Date")
);

CREATE TABLE "FACT_TABLE" (
"PRODUCT_Code" integer,
"CITY_Name" character varying,
"PURCHASE_Date" date,
"PRODUCT_PURCHASE_Quantity" numeric ,
CONSTRAINT PK_FACTTABLE PRIMARY KEY("PRODUCT_Code",
"CITY_Name",  "PURCHASE_Date"),
CONSTRAINT FK_FTPRODUCT_Code FOREIGN KEY
("PRODUCT_Code")
REFERENCES "PRODUCT_Code" ("PRODUCT_Code"),
CONSTRAINT FK_FTCITY_Name FOREIGN KEY ("CITY_Name")
REFERENCES "CITY_Name" ("CITY_Name"),
CONSTRAINT FK_FTPURCHASE_Date FOREIGN KEY
("PURCHASE_Date")
REFERENCES "PURCHASE_Date" ("PURCHASE_Date"));
```

3.5 Multidimensional Database creation

The physical schema has been created in SQL. The next step is to create the new multidimensional database into a database manager. The user has to introduce the information about the database manager to create the Multidimensional database and the name for the new Database.
The tool connects to the database and makes the update.

In the example only is needed an account that allow us to create a new database. Then we can create the new database is named “MultidimensionalCompanyA in the server http://companyA.org:3306 with the user “writerUser” and password “pwdwriteUser”.

3.6 Transformation Schema

The database is created but there are not data into it. Then it is necessary the definition of a schema that allow the load of data from the relational database to the multidimensional database.
This model is very simple. It defines a query for the relational database. The result of the query is the data of a dimension or a fact table. There are a query for each dimension and other one for the fact table.

The query for the Dimension PRODUCT_Code is :

```
SELECT DISTINCT PRODUCT."Code"
FROM "CITY" CITY ,"PRODUCT" PRODUCT ,"STATE" STATE
WHERE PRODUCT."City"=CITY."Name" AND
CITY."State"=STATE."Name"
```

The query for the dimension CITY_Name is :

```
SELECT DISTINCT CITY."Name", STATE."Name"
FROM "STATE" STATE ,"CITY" CITY
WHERE CITY."State"=STATE."Name"
```

The query for the dimension PURCHASE_Date is:

```
SELECT DISTINCT PURCHASE."Date"
FROM "PURCHASE" PURCHASE
```

The query for the fact table is :

```
SELECT PRODUCT1."Code" , CITY2."Name" , PURCHASE3."Date" ,
sum(PRODUCT_PURCHASE."Quantity" )
FROM "PRODUCT" PRODUCT1 ,"PRODUCT_PURCHASE"
PRODUCT_PURCHASE ,"CITY" CITY2 ,
"CUSTOMER" CUSTOMER ,"CUSTOMER" CUSTOMER2 ,"PURCHASE"
PURCHASE2 ,"PURCHASE" PURCHASE3
WHERE PRODUCT_PURCHASE."Product_Code"=PRODUCT1."Code" AND
CUSTOMER."City"=CITY2."Name" AND
PURCHASE."Customer"=CUSTOMER2."ID_Number" AND
PRODUCT_PURCHASE."Purchase_Code"=PURCHASE2."Code" AND
PRODUCT_PURCHASE."Purchase_Code"=PURCHASE3."Code"
GROUP BY PRODUCT1."Code" , CITY2."Name" , PURCHASE3."Date" ,
PRODUCT_PURCHASE."Quantity"
```

3.7 OLAP Schema

An OLAP schema defines a multi-dimensional database for a concrete OLAP Server. It contains a logical model consisting of cubes, hierarchies, and members; and a mapping of this model into the physical model.
The logical model consists of the constructs used to write queries in MDX language: cubes, dimensions, hierarchies, levels, and members.
The physical model is the source of the data which is presented through the logical model. It is typically a star schema, which is a set of tables in a relational database; later, we shall see examples of other kinds of mappings.
The OLAP server that is use in the model is Mondrian. The Mondrian schemas are represented in an XML file.
The most important components of a Mondrian schema are cubes, measures, and dimensions:

- A *cube* is a collection of dimensions and measures in a particular subject area.
- A *measure* is a quantity that you are interested in measuring.
- A *dimension* is our dimension previously defined.
- A *member* is a point within a dimension determined by a particular set of attribute values. The gender hierarchy has the two members 'M' and 'F'. 'San Francisco', 'California' and 'USA' are all members of the store hierarchy.
- A *hierarchy* is a set of members organized into a structure for convenient analysis. It is place into a dimension. It is our hierarchy.
- A *level* is a collection of members which have the same distance from the root of the hierarchy.

All the elements defined in a Mondrian Schema are mapped into the multidimensional database.

The OLAP schema for the example is the next one:

```
<?xml version="1.0" encoding="UTF-8"?>
  <Schema name="customersInfomationDW">
    <Cube name="customersInfomationDW">
      <Table name="FACT_TABLE" />
        <Dimension name="PRODUCT_Code" foreignKey="PRODUCT_Code">
          <Hierarchy hasAll="true" primaryKey="PRODUCT_Code"
           allMemberName="All PRODUCT_Code">
            <Table name="PRODUCT_Code" />
            <Level name="PRODUCT_Code" column="PRODUCT_Code"
            uniqueMembers="true" />
          </Hierarchy>
        </Dimension>
```

```
        <Dimension name="CITY_Name" foreignKey="CITY_Name">
          <Hierarchy hasAll="true" primaryKey="CITY_Name"
          allMemberName="All CITY_Name">
            <Table name="CITY_Name" />
            <Level name="STATE_Name" column="STATE_Name"
            uniqueMembers="false" />
            <Level name="CITY_Name" column="CITY_Name"
            uniqueMembers="true" />
          </Hierarchy>
        </Dimension>
        <Dimension name="PURCHASE_Date" foreignKey="PURCHASE_Date">
          <Hierarchy hasAll="true" primaryKey="PURCHASE_Date"
          allMemberName="All PURCHASE_Date">
            <Table name="PURCHASE_Date" />
            <Level name="PURCHASE_Date" column="PURCHASE_Date"
            uniqueMembers="true" />
          </Hierarchy>
        </Dimension>
        <Measure name="PRODUCT_PURCHASE_Quantity"
        column="PRODUCT_PURCHASE_Quantity" aggregator="sum" />
  </Cube>
</Schema>
```

3.8 Query for the OLAP server

In order to show the information we have to make a query to the OLAP server. This query is made in MDX. MDX is a language for querying multidimensional databases, in the same way that SQL is used to query relational databases. MDX was introduced by Microsoft with Microsoft SQL Server OLAP Services in around 1998, as the language component of the OLE DB for OLAP API. More recently, MDX has appeared as part of the XML for Analysis API. Microsoft proposed MDX as a standard, and its adoption among application writers and other OLAP providers is steadily increasing.

The query is going to be made in and web environment based in JPivot. JPivot is a web OLAP client. It is a JSP custom tag library that renders an OLAP table and chart. Users can perform typical OLAP navigations like drill down, slice and dice.
Then we have to create a special jsp (Java server page) with the query.

Example

```
<%@ page session="true" contentType="text/html; charset=ISO-8859-1" %>
<%@ taglib uri="http://www.tonbeller.com/jpivot" prefix="jp" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%-- uses a dataSource --%>
<jp:mondrianQuery id="query01" jdbcDriver="org.postgresql.Driver"
jdbcUrl="jdbc:postgresql://localhost:5432/customersInfomationDW"
jdbcUser="postgres" jdbcPassword="postgres" catalogUri="/WEB-
INF/queries/customersInfomationDW.xml">

  select
  {([PRODUCT_Code].[All PRODUCT_Code] ,
  [CITY_Name].[All CITY_Name])} ON COLUMNS ,
  {([PURCHASE_Date].[All PURCHASE_Date])} ON ROWS
  FROM customersInfomationDW

</jp:mondrianQuery>
<c:set var="title01" scope="session">Test Query uses Mondrian OLAP</c:set>
```

3.9 First load

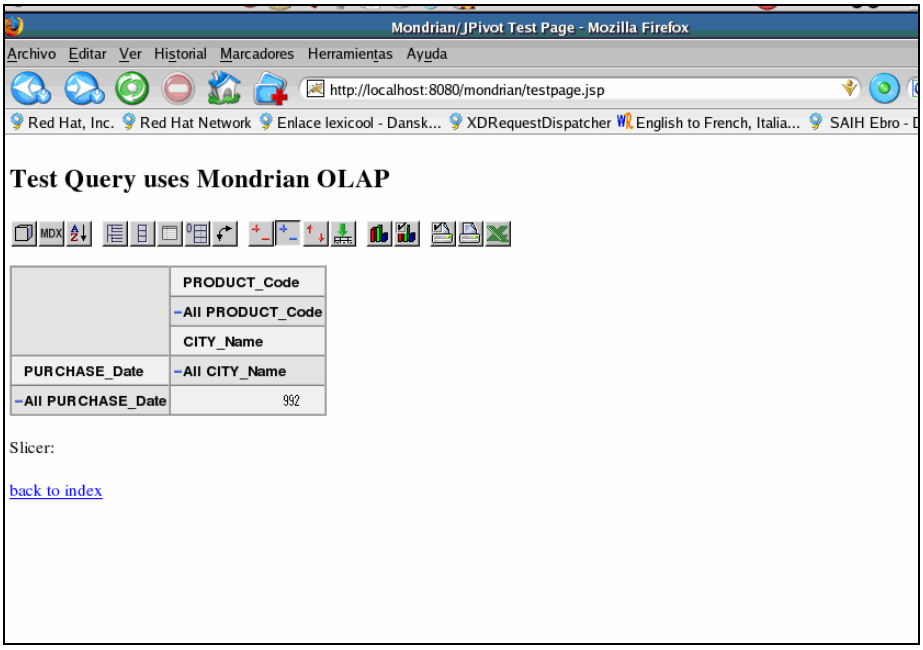
The model is not design to create a complete Decision Support System. In a complete DSS is needed an ETL tool that extracts information continuously from the operational database and it loads this data into the data warehouse. This ETL tool has to use the transformation schema to store more rows into the Multidimensional database. But the purpose of the model is not to create a ETL, is to create the multidimensional database. For that reason it is only involve in the first load of the multidimensional database.

In order to make the firs load the tool make a query in the relational database for each table of the multidimensional database. The query is defined in the transformation schema. The result of this query has to be insert into appropriated table of the multidimensional database.

3.10 Showing the results

At the end only is needed the visualization of the result in the web environment

In the example the result is the next one picture:



3.11 System analysis

During the charter has been described the problem that the thesis want to resolve and the steps that a tool has to make in order to make the multidimensional database. With this information the problem is completely describe. At the same way the necessities and requirements of the system that has to be created could be extracted from this in order to analyse the system that is going to be created.
System requirements

The requirements of the system will be presented into two categories. Functional requirements, that are the statements of service the system should provide. Non-functional requirements, which are the services or functions, offered by the system like for example timing, standards or technology.

3.11.1 Functional system requirements

Requirement Code	Requirement specification
FR-1	The system allows user to design a multidimensional database schema from a relational database in order to create a data mart.
FR-2	The system has to connect to a relational database to extract the information. The needed information to do that is provided by user.
FR-3	The system allows user to select the important information of the relational model that will be involved in the multidimensional model. This process is guide for the tool.
FR-4	The system provides user a especial way to introduce time attributes for the date dimension.
FR-5	The system extracts all the semantic information from the relational database, for the construction of the multidimensional model. When exist ambiguous situation the user has to be asked.
FR-6	The system has to create a multidimensional model from the selected data.
FR-7	The system will use a star schema to represent the multidimensional model.
FR-8	The system has to create the multidimensional database with a SQL specification since the multidimensional model.
FR-9	The system has to create the multidimensional database. It needed the information to connect with the database manager. This information is provided by user.
FR-10	The system creates the OLAP schema for the OLAP server, stored it and configure the server in order to the multidimensional database could show the multidimensional cube.
FR-11	The system has to create a correspondence model between the relational database and the multidimensional database in order to transfer the data from the relational database to the multidimensional database.
FR-12	The system has to make the first data load from the relational database to the multidimensional database.

Requirement Code	**Requirement specification**
FR-13	A web environment has to be created to allow user to connect to the new multidimensional database.

3.11.2 Non-functional system requirements

The system is going to be created to connect to a PostgreSQL database manager in order to extract the relational database and store the new one.

Requirement Code	***Requirement specification***
NFR-1	The system can extract information from a PostgreSQL manager
NFR-2	The system can create a multidimensional model in a PostgreSQL manager.
NFR-3	The OLAP Server used is Mondrian OLAP Server. The schema is a written for Mondrian.
NFR-4	The Mondrian OLAP Server will run over a tomcat web server.
NFR-5	The language used to write the wizard is JAVA

3.12 Activity diagram

The Figure 3. **5** shows a general vision of the actitivities that have to been done in order to make the multidimensional model. This diagram has been constructed from the problem analysis. It is a resume of the necessaries steps to make the model that are going to be used to make the design of the system.

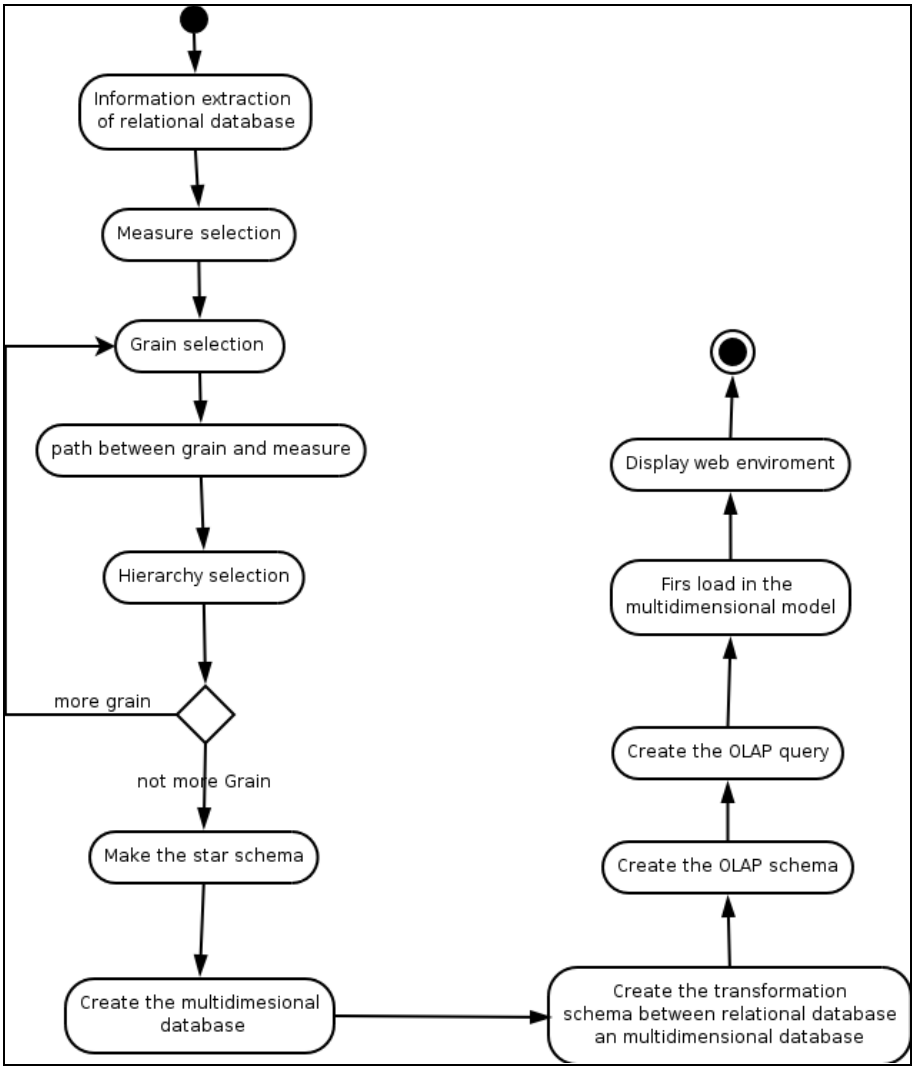


Figure 3. 5:Activity diagram

CHAPTER 4

Design of Multidimensional Databases from Relational Databases

This chapter develops the theory and algorithms for the construction o a multidimensional database from a relational one, which is the main contribution of this thesis.

4.1 Concepts Used

In this section is going to be presented the most important concepts, since a general point of view. These concepts are the ones that are present in the design of the multidimensional databases from the relational ones. This section only introduces the entire notation and the general concept. The deep definitions and explains are develop along the rest of the chapter.

4.1.1 Relational model

A *relational model* is going to be represented as a set of tables and a set of foreign key relations.

A *table* of the relational database is represented with: ti
A *set of tables* is represented as T: { ti } .

A table contains a set of attributes. An *attribute* has the information of the restriction that it has, the name, and the table where it is. The letter ´a´ is used to refer a general attribute. To refer to the restriction of an attribute is used a.restriction.
Other special attributes are the attributes from the relational model that are selected to be a grain attribute. This attributes are represented with the letter ´g´. And the attributes form the relational model that is selected to be a measure attribute for the multidimensional model is named with the letter m.

A *foreign key relation* is represented by two attributes (at, atRef) the attribute ´at´ is the attribute that has the foreign key restriction. It attribute referenced the attribute ´atRef´. The ´atRef´ attribute is referenced by the attribute ´at´.
The *set of foreign key relation* is represented as R: {(at, atRef)}.

4.1.1.1 Selected attributes from the relational model

The user has to select a set of attributes from the relational model to make the multidimensional model. The algorithms, that are needed to make a multidimensional database, need these attributes. They are represented at the next way:
AS: { a }

4.1.1.2 Path

The term path is introduced in the chapter 3. A path is composed by a start attribute, an end attribute and a set of foreign key relations that join them. It is represented as
p: (a , e, {fkr})
A set of path is represented with the letter P : { (a, e, {fkr})}.

4.1.2 Multidimensional Model

A *multidimensional model* is represented by a fact table and a set of dimension.

A *fact table* is the table of the multidimensional model where the measure attribute is store. It is represented by a set of grain attributes and a set of measure attributes. FT

= ({g}, {m}), in this thesis the measure attribute only can be one.

The *set of dimensions* is represented with the letter D: { d }
A *dimension* is represented by a name, a multidimensional grain attribute, and a list of multidimensional attributes. The notation for the algorithms is d: (name, gm , {am})

A *multidimensional attribute* is an attribute of the multidimensional model. This multidimensional attribute is represented by an attribute from the relational model, and the semantic information. The notation used is am: (a , s), a is the attribute from the relational model, and s is the semantic information of the multidimensional attribute am.

The *semantic information* of a multidimensional model is the information used in order to specify a hierarchy. This semantic information is defined an attribute, that it is the nearest attribute in a hierarchy and a type. The notation is s: (ah, HIERACHY | DESCRIPTIVE). The type may be HIERARCHY or DESCRIPTIVE. The type HIERARCHY is used for an attribute that represent a level in a hierarchy. The type DESCRIPTIVE is used for an attribute that is describing the attribute ah.

4.2 Multidimensional model

A multidimensional model is defined by a fact table and several dimension tables.

The first thing that the user has to select from the relational model is the measure attribute. As it has been explained, the measure attribute is the purpose of the cube. It is what the user wants to know, an attribute from the relational model that will be save into the fact table of the multidimensional model.

Once the measure attribute is selected the user has to be asked about what is the information the he want to relate with the measure attribute. This information is needed to define the dimensions. Then the grain attribute has to be selected. A grain attribute is selected from the relational model.

Once the grain attribute is selected, the tool has to find a path between the gain attribute and the measure. A grain attribute may have more than one path to the measure attribute. The user chooses the path or the paths that he wants for the multidimensional model. Each path represents a concept of information. Then each couple consisted of path and grain attribute represent different information that the user want to store in the multidimensional model. Therefore for each couple of grain attribute and path, that the user has selected, a new dimension is going to be created.

In order to create the fact table are needed the measure and the grain attributes of each selection. It is because the fact table is composed for this elements.

Algorithm Multidimensional_Modelling
Input:
T: { t_j } set of tables of the relational model
R: { (a_i , a_j) } set of foreign key relations

Output:

FT = ({g}, {m})
D = {d}
AS = { a }

Variables used in the algorithm
G : variable used to store a grain attribute
m : variable used to store the measure attribute.
P : {pi}: set of path in construction. They start in the grain attribute g.
p : a path.

Begin:
moreGrains : boolean := true;

AS = {}
m := select_measure_attribute_from_relational_model();

// Dimensions
while (moreGrains) **loop**
g := select_grain_attribute_from_relational_model();
// P : {p} set of path from the measure grain attribute to the measure attribute
P = makePathsGrainToMeasure(T, R, g, m);
for all p ∈ P **loop**
// d : new dimension
d := makeDimension(T, R, AS, g);
D := D ∪ { d }
P := P ∪ { p }
end loop;
moreGrain := user_answer();
end loop;

// Fact Table
for all d ϵ D **loop**
g' := grain_attribute_of(d);
FT := ({g}∪ {g'} , {m})
end loop

end algorithm;

4.3 Path between grain attribute and measure attribute

A multidimensional database relates the data of the dimensions whit the measure. This data is extracted from a relational database. This relational database has to content the way to relate them. It is possible through foreign key relations. More specifically a path is the structure that relates two attribute from a relational database.

As it is possible to see in the Chapter 3 Section 3.2. Selection of information from the Operational Source, it is possible to find more than one path that joins a grain attribute with the measure. This is because of the relational database is design to store the information in an optimal way, avoiding the redundancy of data. Then it is possible to find store data which has more than one meaning. The data has more than one interpretation. For example in the example of the chapter 3 section 3.2 between the table CITY and the table of the measure there are two possible path. Each one represents different information. The same data may be the city where a product is sold, if the table CITY is related with the table PRODUCT. The data of the table CITY may be other meaning if is related with the table CUSTOMER. It is the city of the customers.

One of the most important things in order to make a multidimensional model is to extract the relation of the data, and this relation has to contend the meaning that the user really wants. Then it is not possible to select one of the paths that the algorithm finds. The user has to understand that there are two different meaning for the same data. It has to choose the correct one.

The path between the measure attribute and the grain attribute is a path that starts in the grain attribute and it finishes in the measure attribute. It has a set of foreign key relation. The notation is: p: (g, m, {fkr})

Because of the relational database is in third normal form an attribute that is not a primary key, on of the table that contains its, depends only of the primary key of its table. For that reason the path has to follow the foreign key relations that the primary key of the table is referenced.

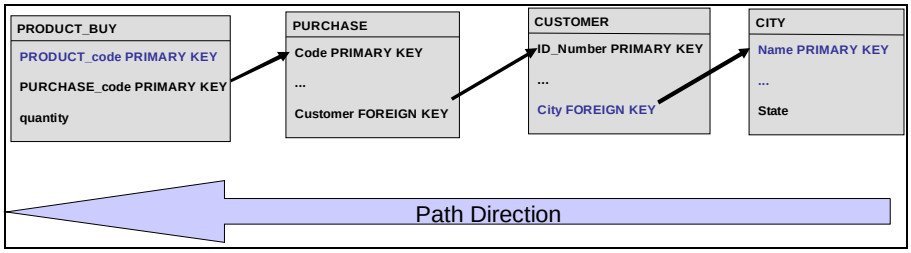


Figure 4.1: Path between measure attribute and the grain attribute city

In the Figure 4.1 there is an example of one of the paths between the grain attribute Name of the Table city, and the measure quantity of the table PRODUCT_BUY. The Path starts in the grain attribute. There are a foreign key relation between the table customer and the table city. The attribute City of the table CUSTOMER reference the attribute Name of the table CITY. It is indicated by the arrow. Once in the table CUSTOMER there is a foreign key relation with the table PURCHASE. In the table PURCHASE there is a foreign key relation with the table PRODUCT_PURCHASE that is the table that contains the measure attribute. Then is the end of the path.

The relational database may be considered like a graph. Each table is a node, and the foreign key relations are the edges of the graph. The direction of the edges is the contrary that the direction of the arrows in the Figure 4.1 because in that figure the arrow represents that and attribute is referencing a PRIMARY KEY. But in the graph of a relational database is needed to represent the direction of the path from the grain attribute to the measure attribute. The Figure 4. 2 shows the graph that represents the relational database of the example described in the chapter 1, section 1.3. In this figure is possible to see all the possible ways between a table and the measure attribute. As has been explained between the table city and the table state there are more than one possible path between these tables and the table PRODUCT_PURCHASE. The tool has to look for its and the user is who choose the correct one.

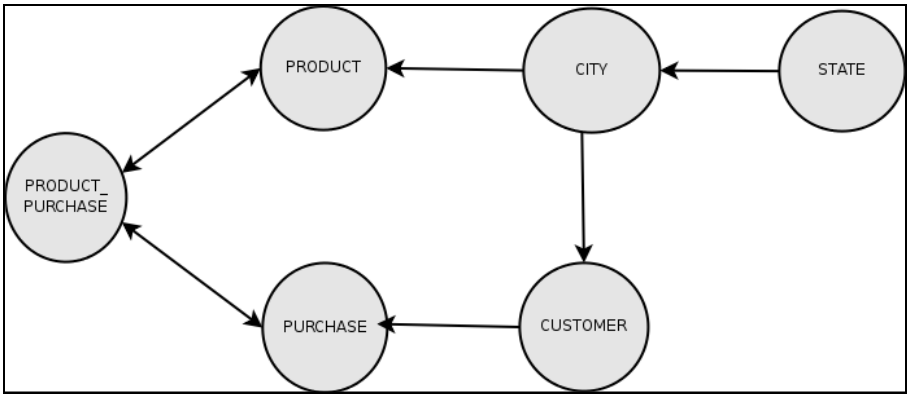


Figure 4. 2: Graph for a relational database

Between the table PRODUCT and the table PRODUCT_PURCHASE there are a bidirectional relation. It is because the key of the table PRODUCT_PUCHASE referenced the PRIMARY KEY of the table PRODUCT. It is a relation N : M in the model E-R. These relations are always bidirectional.

There are other types of paths that join the measure attribute with the grain attribute. This path contains only a relation and the direction is the same than the direction of

the relation. Other restriction is that the grain attribute has to be the attribute of the foreign key relation that is referencing the table of that contains the grain attribute. Then this attribute is the same attribute than the primary key of the measure's table.

The algorithm use to find the paths between a grain attribute and a measure attribute is:

Algorithm makePathsFromMeasure( ){

Input

T : { t_j } set of tables of the relational model
R : { (a_i , a_j) } set of foreign key relations
g : grain attribute
m : measure attribute

Output
P : {pi} : set of path from the grain attribute g to the measure attribute m

Variables used in the algorithm
Pwe : {pi}: set of path in construction. They start in the grain attribute g.
T : { ti } : tables inside a path
p : a path.

Begin

```
// create a path that start in g and finish in g
   p := ( g, g, {} ) ;
   // add the trivial path to the set of path to evaluate
   Pwe := Pwe ∪ p ;

   while ( Pwe /= Φ ) loop
      // p = ( g, a, {fk});
      p := getPath(Pwe);
      // The end of the path is the same table than the measure table.
                  // The path is complete
            if ( a ϵ ti . ti  ϵ tj . i = j ) then
            p :=  (g,m,{fk});
            P := P ∪ p ;
                  else
                  // get edges from the table that contains the attribute  of the end of the path
            EDGES = get_edges(a, R);
            // EDGES : { fkr }
                  for all edge  ϵ  EDGES loop
            // edge : (at , atRef)
            // The table of the attribute a is not in the set of tables of the paths
                  if ( at ϵ ti . ti not {tableOf(p)} ) then
```

```
                        p = (g, at ,{fkr} ∪ {edge});
                                Pwe = Pwe ∪ { p'}
                        end if;
                        end loop;
                end if;
        end loop;
        P := P ∪ Path_From_FK(A);
end makePathFromMeasure;
```

4.4 Make dimensions

A dimension is a set of attributes that is organized in a hierarchy. A grain attribute may have more than one hierarchy. For each one a dimension is going to be created.

A grain attribute is an attribute from a relational model that it has been selected to be a grain attribute of a dimension of a multidimensional model. The attributes from the relational database have a restriction. This restriction is that they are or PRIMARY KEY, or UNIQUE, or FOREIGN KEY, or NONE attributes. These restrictions help us to extract some information. Supposing that the relational model is in third normal form then:

- If the grain attribute has a PRIMARY KEY restriction, all the attributes of the table depend of the primary key. It is possible to find a hierarchy.
- If the grain attribute has a UNIQUE restriction, this attribute is a candidate key, that means that could a PK of the table, then is going to be process like a PRIMARY KEY.
- If the grain attribute has a FOREIGN KEY restriction, this attribute is referencing a PRIMARY KEY. Then the hierarchies of that attribute are the same than the hierarchies of the PRIMARY KEY that it is referencing.
- If the grain attribute has a NONE restriction, it means that it attribute depend only of the PRIMARY KEY of the table, the rest of the attributes of the tables has no relation with it. Then this attribute will be alone in the dimension.

Algorithm make_dimension(Path thePath){
Input :
T : set of tables of the relational model
R : set of foreign key relations of the relational model
AS : { a } set of selected Attributes
g : grain attribute

Output:
d : dimension created
AS : { a } set of selected attributes

```
Begin:
    P := lookForHierarchies(T, R, g);

    for all p ϵ P loop
        // p : ( g, e, {kfr} )
        // gm : ( g , s) The multidimensional grain attribute
        gm := ( g, addSemanticInfoGrain(AS, g, p));
        d := (name , gm , {gm});
        if g.restriction  is PRIMARY KEY | FOREIGN KEY | UNIQUE then
            A := selectHierachyAttributes(p);
            A := UserSelectAttributes(SelectedAttr);

            for all a  ϵ A loop
                am := (a, addSemanticInfo(AS, a, p));
                AM := AM ∪{ am };
                AS := AS ∪ A;
            end loop;
            d := (name , gm, AM);
        end if;
    end loop;
end algorithm;
```

4.5 Hierarchy

Each dimension has a grain attribute and a set of attributes organized in a hierarchy. A hierarchy has been defined as different levels of abstraction for the information.
A more concrete definition of a *hierarchy* is a set of attributes related by levels that start with the grain attribute. The less level of the hierarchy is the grain attribute. In a hierarchy there are two different types of attributes, the hierarchy attributes and the descriptive attributes. This information about the attributes of a hierarchy is called semantic information.

The hierarchy attributes define the levels. A descriptive attribute describe a property of a hierarchy attribute. Then the grain attribute is a hierarchy attribute of him. All the attributes of a hierarchy have to belong to a path. The path stars in the grain attribute and finish in primary key of the last table of the hierarchy.

The structure of a relational database allows us to find the hierarchies. The hierarchy attributes and the descriptive attribute. The extraction of this information is only possible if the relational database is in third normal form. Because of this it is assumed that the relational database is in third normal form.

There is a pattern to detect the path of a hierarchy in a relational database. This path

follows the direction of the foreign key relation. For example in the Figure 4. 3 there is represented a hierarchy. It is supposed that the attribute code of the table PRODUCT is selected as a grain attribute. Then the table PRODUCT is the beginning of the hierarchy. It is the more concrete level of information. The second table of the hierarchy is the table CITY, this table extends the information about the attribute city of the table PRODUCT. Then it is possible to group all the products by the same city. The table CITY contains the attribute state. This attribute has a foreign key relation with the attribute Name of the table STATE. Then the table STATE extends the information of the attribute state of the table PRODUCT. This table is the last in the hierarchy because of there are no attributes with foreign key relations. This hierarchy follows a path between the table PRODUCT and the table STATE. The direction of this path is the contrary than the direction of a path between the grain attribute and the measure. The reason is that when it is looking for a hierarchy, and the hierarchies trying to find information more general about an attribute.

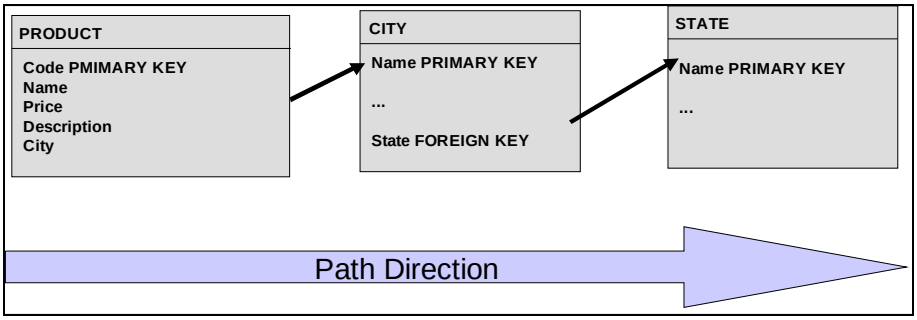


Figure 4. 3: Example of hierarchy

Then is possible to make a graph similar than the one of the path for the measure. The difference is that now the direction of the edges is the contrary than that one, is the same direction than the direction of the foreign key relations. It is possible to see in the Figure 4.3. Other difference is that the table that contains bidirectional relations can not be in the middle of a path. Its can be only in the begging. Then these nodes only have edges that go out. The reason is that it is a table with more than one attributes that are primary key, this attributes referenced other attributes in other tables. If one primary key is selected may be the same hierarchy than the hierarchies of the attribute that it is referencing. But can not be selected attribute of the table. Because of the attribute do not describe this selected attribute. The attribute describe de primary key of the table, because of the model is third normal form. The primary key of this table is composed by more than one attribute, and then there are more than one value for each grain attribute. That attribute do not describe the grain one. It is a measure attribute.

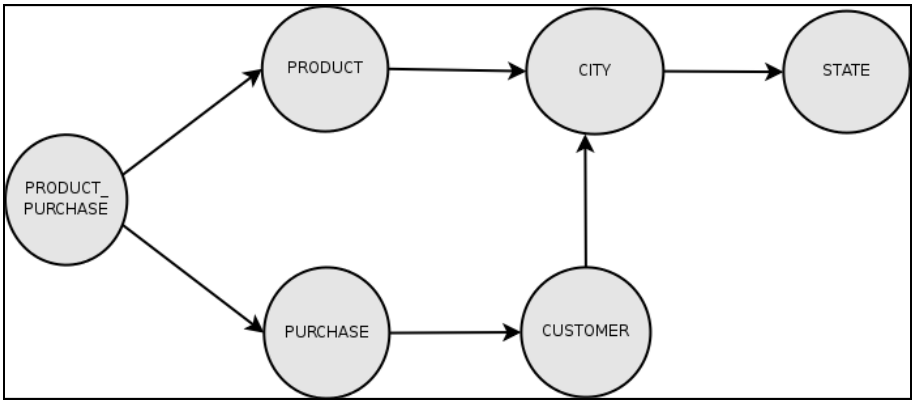


Figure 4.4: Graph for find the hierarchy in a relational model

Algorithm lookForHierarchies()

Input

T : { t_j } set of tables of the relational model
R : { (a_i , a_j) } set of foreign key relations
g : grain attribute
m : measure attribute

Output
P : {pi} : set of path of hierarchies

Variables used in the algorithm
Pwe : {pi}: set of path in construction. They start in the grain attribute g.
p : a path.

Begin

```
// create a path that start in g and finish in g
p := ( g, g, {} ) ;

// add the trivial path to the set of path to evaluate
Pwe := Pwe □ p ;

while ( Pwe /= Φ ) loop
        // p = ( g, a, {fk});
        p := getPath(Pwe);
        EDGES = get_edges(a, R);
        // EDGES : { edged} a edge is a foreign key relation
        if (EDGE == Φ ) then
                P := P □ { p };
```

```
                else
                        for all edge  ϵ  EDGES loop
                                //edge :  ( at , atRef )
                                if ( at ϵ ti . ti not {tableOf(p)} ) then
                                        p = (g, a ,{fkr}□ {edge});
                                        Pwe = Pwe □ { p }
                                end if;
                        end loop;
                end if;
        end loop;
end algorithm;
```

4.6 Semantic information

In the section of the hierarchy appears the concept of semantic information in order to make a hierarchy. The semantic information may be defined as the information that an attribute has in order to be related with other attribute inside a hierarchy. This information allows the construction of the hierarchies ordering the attributes and the relations between them.

The semantic information of an attribute is composed by two elements: an attribute, which is related, and the type of the relation. There are two different types of relations: hierarchy relation and descriptive relation.

An attribute has a hierarchy relation with other when this attribute represent a high level in a hierarchy. An attribute has a descriptive relation with other one when this attribute is describing a hierarchy attribute.

This relation are extracted from the relational model, it is supposed that the model i s in third normal form.

The semantic information that it is possible to extract of the relational database, which is in third normal form, is the next one:

- An attribute of a table that is not the primary key only depends on primary key.
- An attribute in a foreign key relation that is not the primary key represents a level. It represents a level because is referencing other attribute that is a different entity. It has meaning by himself.

With this information it is possible to say:

- The grain attribute is the less level of a hierarchy. The semantic information of it is a hierarchy attribute of him.
- An attribute that is a primary key or a foreign key is always a hierarchy attribute. The less level in the hierarchy for this attribute is the hierarchy

attributes nearest to him, between him and the grain attribute.

- An attribute that has a unique restriction is a descriptive attribute of the primary of its table if it is selected. If the primary key is not selected, it is a descriptive attribute of the foreign key that referenced the primary key of its table. If this one is not selected in hierarchies attribute. The less level in the hierarchy for this attribute is the hierarchy attribute nearest to him, between him and the grain attribute.
- The attributes that has the restriction none are descriptive attributes of nearness hierarchy attribute.

If there are two different selected attributes from the same table it has to be considered the next rule:

- The primary key of the table is the nearest attribute to the grain.
- A foreign key relation is the furthest attribute to the grain attribute.
- A descriptive attribute of a table can not be descriptive attribute of an attribute of the same table different than the primary key. It is because relational databases in third normal form have all the attributes of a table depending only of the primary key.

The Algorithm to extract the semantic information of an attribute is the next one:

Algorithm SemanticInformation
Input :

AS : {a} set of attributes selected for the hierarchy
a : attribute to add the semantic information
p : (g, e, {fkr}) path of the hierarchy of the attribute a. g is the grain attribute of the hierarchy.

Output:
s : (am, HIERARCHY | DESCRIPTIVE) semantic information of the attribute a.

Begin:

```
if (a ϵ ti  Λ g ϵ tj. i = j) then
    //attribute in the table of the grain attribute.
    if (a == g ) then
        s = ( g , HIERARCHY );
    else
        if ( a.restriction == FOREIGN_KEY  ) then
            s = ( g , HIERARCHY );
        else
            s = ( g , DESCRIPTIVE );
        end if;
else
    attributeInPath := false;
```

```
for all fk ϵ p:(g,e,{fkr}) loop
    // fk : (at , atRef)
    if ( a ϵ ti ∧ atRef ϵ tj. i = j) then
        attributeInPath = true
    end if;
    if attributeInPath then
        if ( a.restriction == PRIMARY_KEY ) then
            if at ϵ AS then
                s = ( at , HIERARCHY );
                break;
            elseif ( ∃ an ϵ ti ∧ g ϵ tj. an ϵ AS ∧ i = j ∧ an.restriction ==
            PRIMARY_KEY )
                s = ( at , HIERARCHY );
                break;
            end if;
        else
            if (at ϵ AS ) then
                if ( a.restriction == NONE ) then
                    s = ( at ,DESCRIPTIVE );
                    break;
                elsif ( a.restriction == UNIQUE ) then
                    if ( a ϵ ti ∧ at ϵ tj. i = j ) then
                        s = ( at ,DESCRIPTIVE );
                        break;
                    else
                        s = ( at , HIERARCHY );
                        break;
                    end if;
                end if;
            elsif (∃ au ϵ ti ∧ at ϵ tj. au ϵ AS ∧ i = j ∧ an.restriction == UNIQUE )
                if ( a.restriction == NONE ) then
                    s = ( au ,DESCRIPTIVE );
                    break;
                else
                    s = ( au ,HIERARCHY );
                    break;
                end if;
            elsif ( atRef ϵ AS) then
                if ( a.restriction == NONE ) then
                    s = ( au ,DESCRIPTIVE );
                    break;
                else
                    s = ( au ,HIERARCHY );
                    break;
                end if;
```

```
            end if;
          end if;
        end if;
    end loop;
end if;
```

4.7 Star schema

The multidimensional model is going to be stored in a relational database. Then this schema has to be represented as a relational database. The more common and standard way to represent a relational database is using the definition language SQL. Each dimension is represented as a table. And the fact table is other table. The algorithm to create that file of definition is the next one.

Algorithm makeStartSchema
Input:
D : $\{ d_i \}_{i=1..n}$ set of all dimension
FT : fact_ table
Output:
StartSchema: File with the schema.
Begin

```
    for all d ϵ D loop
        writeInFile( StartSchema, startSchemaDimension(d) );
    end loop;
    writeInFile ( StartSchema, startSchemaFactTable(FT));
end algorithm;
```

The algorithm to make the start schema of a single dimension:

Algorithm startSchemaDimension
Input:
d : (Name, gm, {am})
Output:
StartSchema: String;
Begin

```
    // gm : ( g , s)
    StartSchema = "CREATE TABLE "+ Name + " (";
     StartSchema = Start Schema +   gm + g.type + PRIMARY KEY ;
    // am : (a, s)
    for all am ϵ { am } . am /= gm  loop
        StartSchema = Start Schema +   am  + a.type ;
    end loop;
end algorithm;
```

The algorithm to make the start schema of the fact table:

```
Algorithm  startSchemaFactTable
Input:
    FT ({gm}, m)
Output:
        StartSchema: String;
Begin
        // gm : ( g , s)
        StartSchema = "CREATE TABLE FACT TABLE (";
        StartSchema = Start Schema +   gm + g.type + PRIMARY KEY ;
        // am : (a, s)
        for all gm ϵ { gm } . am /= gm  loop
                StartSchema = Start Schema +   gm  + g.type + PRIMARY KEY ;
        end loop;
         StartSchema = Start Schema +   mm  + m.type ;
        for all gm ϵ { gm } . am /= gm  loop
                // the function   foreignKeyConstrain return a string with the constrain of a
                //foreign key relation
                StartSchema = StartSchema + foreignKeyConstrain(gm) ;
        end loop;
end algorithm;
```

4.8 Transformation schema

The transformation schema has been explained in the chapter 3. It consists in a SQL query for each table of the multidimensional model, from the relational model. The result of this query can be inserted directly in the appropriate table of the multidimensional table.

The transformation schema used an algorithm to make this schema for the dimension tables, and other different for the fact table.
The algorithm to make all the schema is the next one:

```
Algorithm transformationSchema
Input:
    D   : { (d, p) }          set of all dimension, with the path of hierarchy
    P   : { p} set of all the path for each dimension

    FT : fact_ table
Output:
        TransformationSchema: File with the schema.
Begin
```

```
        for all dh ϵ D loop
                // dh : (d, p)
                writeInFile( TransformationSchema,
                TransformationSchemaDimension(d , p) );
        end loop;
        writeInFile ( TransformationSchema, TransformationSchemaFactTable(P,
FT));
end algorithm;
```

In order to make the transformation schema of the dimension tables is need the information about which are the attributes that are needed and what is the relation between these attributes. The attributes that are needed from the relational model are specify in the dimension. In order to relate these data is needed a set of tables and foreign key relations. This information is in the hierarchy path.
Then to make the transformation schema for a dimension table is the process is select the attributes from the dimension and related these attribute throw the relations of the hierarchy path.

The algorithm is the next one :

```
Algorithm  TransformationSchemaDimension()
input:
        d : (nombre, g, {a})
        p : (g, e, {fkr})
output:
        SELECT : String with the SQL query

Variable auxiliary to the process
        FROM : String with the clause FROM for the SQL query
        WHERE : String with the clause WHERE for the SQL query
Begin:
        SELECT = "SELECT DISTINCT " +  g;
        for all a ∈ A. a/=g loop
                SELECT = SELECT + g ;
        end loop;

        FROM ="FROM ";
        WHERE = "WHERE " ;
        if ( {fkr} == ø) then
                 FROM += g
        else
                for all fk ∈ {fkr} loop
                        // fk : (at, atRef)
                        if ( at ∈ ti . Ti  not ∈ T' ) then
```

```
                        FROM  = FROM + at
                    end if;
                    if ( atRef ∈ ti . Ti  not ∈ T' ) then
                        FROM  = FROM + atRef
                    end if;
                    WHERE = WHERE + at + '=' + atRef ;
                end loop;
                SELECT = SELECT + FROM+ WHERE;
        end if;
end algorithm;
```

The transformation schema of the measure table is more complex. They are needed the grain attribute of each dimensions, the measure attribute and the way to aggregate this one, and the path to relate the measure attribute with the grain attribute.

For each grain attribute has to create a path to the measure table. This path has to be different. It is not possible to make a query using the same JOIN for two different paths, because it is necessary that the information was related with the measure attribute and not with the other dimensions. Each dimension is independent.

Algorithms TransformationSchemaFactTable
Input:

P : { p } set of paths between the grain attribute and the measure
FT : ({gm}, mm)

Output:

SELECT : String with the SQL query

Variable auxiliary to the process

FROM : String with the clause FROM for the SQL query
WHERE : String with the clause WHERE for the SQL query
GROUPBY : String with the clause GROUPBY for the SQL query

Begin:

```
SELECT := "SELECT ";
WHERE := "WHERE ";
FROM := "FROM ";
GROUPBY := "GROUP BY ";

        for all p ∈ P  loop
            T := {};
            // p : (g, e, {fkr})
            SELECT := SELECT + g;
            GROUPBY := GROUPBY g;
            for all fk ∈ {fkr}  loop
                // fk : (at, atRef)
```

```
                                    if (at ∈ ti ∧ m ∈ tj . ti not T ∧  i/=j ) then
                                            FROM += ti + AliasAtTi;
                                    end if;
                                    if (atRef ∈ ti ∧ m ∈ tj . ti not T ∧  i/=j ) then
                                            FROM += ti   + Alias AtRefTi ;
                                    end if;
                            end loop;
                            //JoinFKRelation (fk) return a string with the join of the two
                            //attributes of  the table
                            WHERE +=  JoinFKRelation (fk, AliasAtTi,
                            AliasAtRefTi);
                    end loop;
            end loop;
            SELECT += mm.Aggregator + "(" + m + ")";
            GROUPBY += m ;
            SELECT  = SELECT + FROM + WHERE + GROUPBY;
end algorithm;
```

CHAPTER 5

System Implementation

5.1 Architecture of the system

The system design in this thesis in order to resolve the problem of the multidimensional database is the one who is show in the Figure 5. 1. This system is composed by different components: a relational database, the thesis tool, a Data Warehouse, an OLAP server, and a web environment.

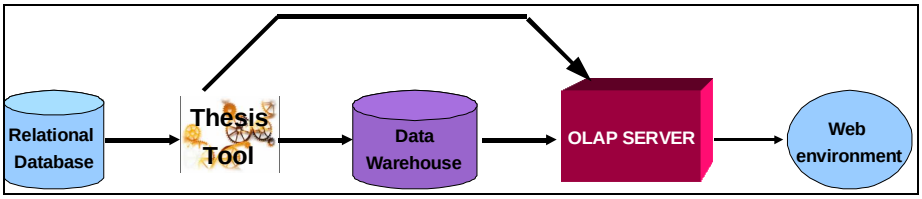


Figure 5. 1: System Architecture

The *relational database* is a database stored in a database manager. This database manager has to be a PostgreSQL manager.

The *Thesis tool* is a program built in JAVA. This tool has to connect to the relational database manager in order to read a database. It has to connect to the Data Warehouse manager in order to create a new database and introduce data. The Thesis tool has to access to the OLAP server directory, in order to store the OLAP schema and the MDX query.

The *Data Warehouse* is the database that the thesis tool has created. It is store in a relational database manager. This database manager has to be a PostgreSQL manager.

The OLAP Server is MONDRIAN OLAP Server. It is an open source OLAP server. This OLAP Server belongs to the platform Pentaho. Pentaho is a business intelligence Project that provides a complete suite to analyse business data. This OLAP server is a module that runs into a Tomcat web server.

The *web environment* is an access to the data through http channel.

5.2 Object design

The purpose of this section is specifying the objects design of the thesis tool. In order to make the design of the thesis tool has been used an UML specification.

It is going to be presented the collection of packages and the relations between those, which have been created for the object design of the tool. The Figure 5.2 is the diagram of the packages of the system. It is composed by the package MultidimesionalBuilder that is associated with the package Multidimensional_construction in order to build the multidimensional database. And with the package Olap_Access in order to make a connection with a multidimensional database already created.

The description of each package is the next one:

The *Relational Model* package represents a database relational model. It is the package used only to represent the data extracted from the operational source.

The *Multidimensional Model* package represents the multidimensional model. It contains all the information about the construction of the multidimensional database, and the information about the multidimensional model.

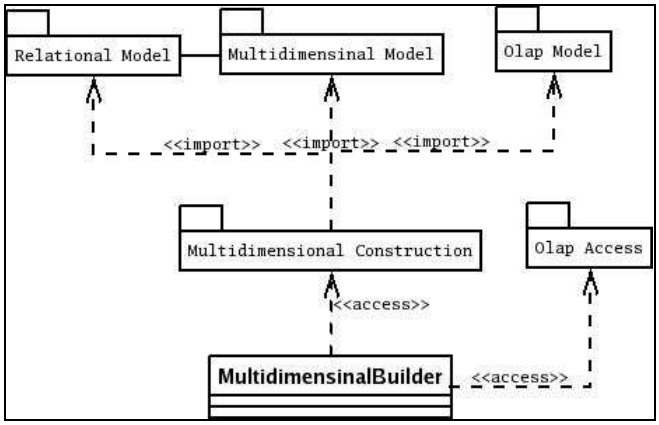


Figure 5.2 Package diagram

The *OLAP Model* package creates the OLAP schema for a multidimensional data model.

The *Multidimensional Construction* package has the intelligence to construct all the model.

The *MultidimensionalBuilder* class is the one that join all the activities that are describe in the multidimensional Construction package.

5.3 The packages

5.3.1 Package Relational Model

The package Relational model is composed for the classes that are represented in the Figure 5.3.

There is a class to represent each single element of a relational model. The class attribute, with the information about the relational attribute. The class table represents a table from a relational model. It contains a name and a set of attributes. The class FK_relation represents a foreign key relation of a relational model. It is compose by two attributes. The Class path is formed by a final attribute, a beginning attribute and a set of foreign key relations. The class Relational_Model is the one make the basic operation with a relational model like for instance the creation of path between two different attributes and the selection of the attributes from the relational model.

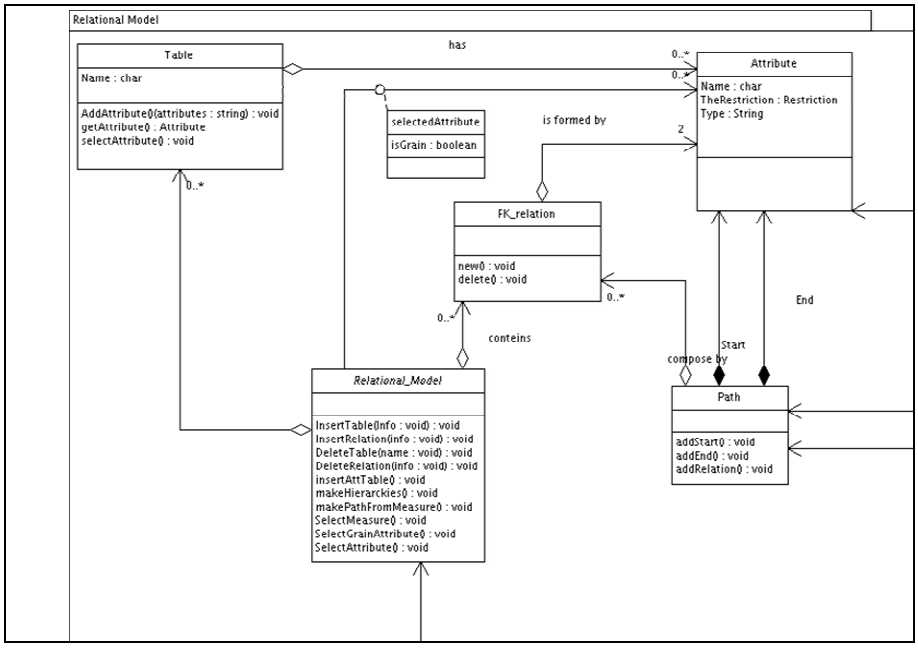


Figure 5. 3: : Package Relational_Model

5.1.1 Package Multidimensional Model

The package multidimensional model represents a multidimensional data model. As the Figure 5.4 shows, it is compose by the classes Dimension, Fact_table, AttributeMulti, Semantic and Multidimensional_Model. The class Dimension represents a dimension. It has a name and is compose for a grain attribute, a set of attributes and a path that defines the hierarchy of the dimension. The class Fact_Table is composed by a measure attribute. The class AttributeMulti represents an attribute from the multidimensional model. It references a attribute from the relational model. And has semantic information. The class Semantic represents the semantic information of an attribute. It is compose by other attribute, and is specify the type. At the end the class Multidimensional_Model represents the data model. It is compose by a set of dimensions a a Fact_Table organize the multidimensional model and allow to create a start schema and a transformation schema.

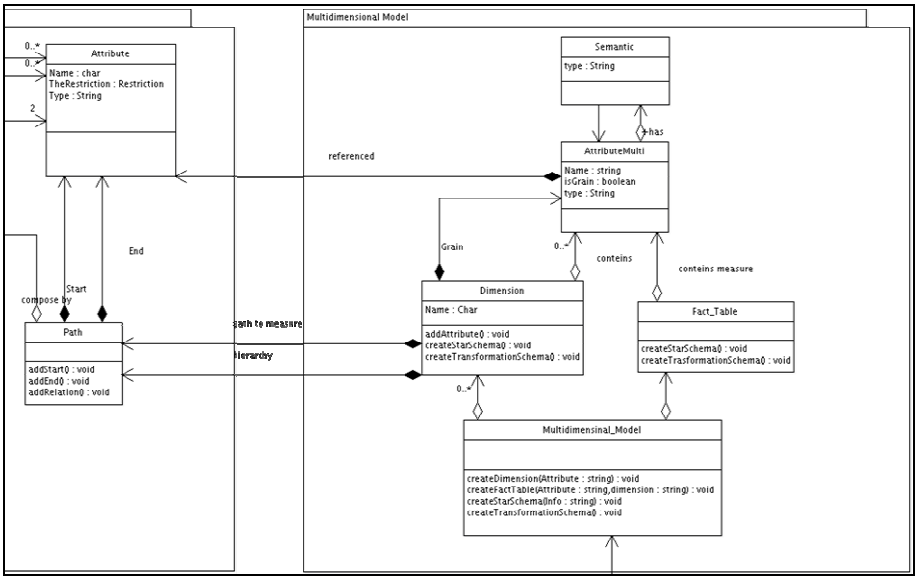


Figure 5.4:: Package Multidimensional_Model with the connexions with the package Relational_Model

5.1.2 Package OLAP Model

The package OLAP_Model is composed by the class OLAP_Schema. This class makes the mapping between the element of a multidimensional model and the element of an OLAP schema.

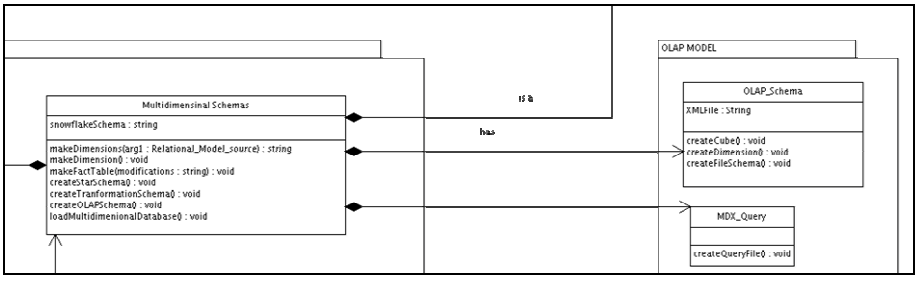


Figure 5. 5: Package OLAP_Model with the connecxion with Multidimensional Schemas

5.1.3 Package Multidimensional Constructor

The package Multidimensional_Construction is the one that has the intelligence to create a a multidimensional database from the relational one. Like it is show in the

Figure 5.6. It is composed by the class Relational Schemas and the class Multidimensional Schemas. The class relational schema makes all the typical operations that have to be done in a relational database in order to create a multidimensional one. It extracts the relational model, select the attributes or look for hierarchies. The class Multidimensional Schema has a relational Schema a multidimensional schema and an OLAP schema. The most important operation are makeDimensions, makeDimension, createStarSchema, CreateTransformationSchema, CreateOLAPSzchema and loadMultidiemsional database. The class multidimensional Builder is the one than control the sequence of the process, and make the interaction with the user.

The package Multidimensional_Construction is the one that has the intelligence to create a multidimensional database from the relational one. Like it is show in the Figure 5.6. It is composed by the class Relational Schemas and the class Multidimensional Schemas. The class relational schema makes all the typical operations that have to be done in a relational database in order to create a multidimensional one. It extracts the relational model, select the attributes or look for hierarchies. The class Multidimensional Schema has a relational Schema a multidimensional schema and an OLAP schema. The most important operation are makeDimensions, makeDimension, createStarSchema, CreateTransformationSchema, CreateOLAPSzchema and loadMultidiemsional database. The class multidimensional Builder is the one one thant control the secuence af the process, and make the interaction with the user.

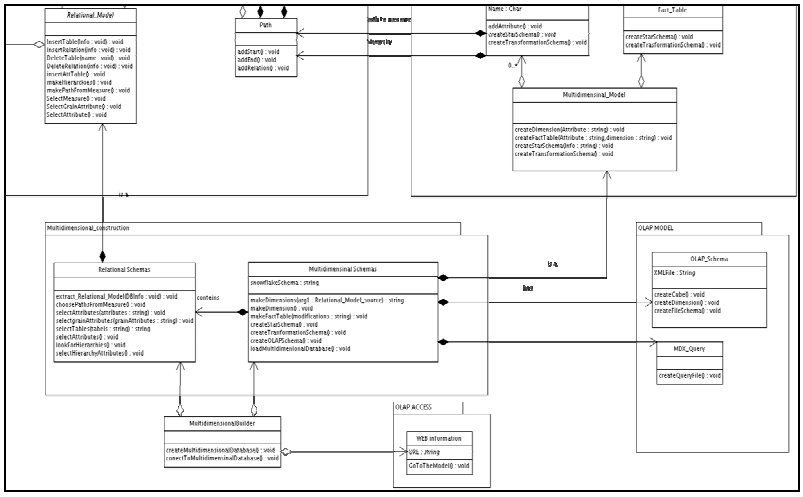


Figure 5.6: Package Multidimensional Construction

5.2 Implementation

The thesis has developed a model for the construction of a multidimensional database from a relational one. This model has been implemented in order to show the viability of it. The software that has been created is not commercial software. It is only a tool implements the model that has been created.

The algorithms that has been describe in the chapter 4 has been implemented trying to be as close as possible close to this description.

5.2.1 Language used

The system design has been made with an object oriented methodology. The implementation has to follow the same direction. For that reason has been used the Language JAVA. The reason of this selection is that JAVA is an object oriented and multi platform language. It means that the tool is able to run in different platforms without the necessity of modifications and new compilations.

5.2.2 Packages and classes implemented

The implementation of the tool use the next ones packages:

- Relational_Model (Source code in the Appendix A, section A.1 Package Relational Model, and page 75).
  The class Relational_Model implements the algoritm makePathFromMeasure explained in Chapter 4, section 4.3 Path between grain attribute and measure attribute, and page 47. This class implements too the algoritm lookforHierarchies explained in Chapter 4, section 4.5 Hierarchy, in page 51.

- Multidimensional_Model (Source code in the Appendix A, section A.2 Package Multidimensional Model, page 99).
  The class Multidimensional_Model implements the algorithm makeStarSchema explained in the Chapter 4, section 4.7 Star schema, and page 57. This class implements too the algoritm TransformationSchema explainded in Chapter 4, section 4.8 Transformation schema, and page 58.

  The class Dimension implements the algorithm StarSchemaDimension explained in the Chapter 4, section 4.7 Star schema, and page 57. This class implements too the algoritm TransformationSchemaDimnsion explainded in Chapter 4, section 4.8 Transformation schema, and page 58.

  The class Fact_Table implements the algorithm StarSchemaFactTable explained in the Chapter 4, section 4.7 Star schema, and page 57. This class implements too the algoritm TransformationSchema explainded in Chapter 4,

section 4.8 Transformation schema, and page 58.

- OLAP_Model (Source code in the Appendix A, section A.3 Package OLAP Model, and page 121).
- Multidimensional_Construction (Source code in the Appendix A, section A.4 Package Multidimensional_Construction, and page 127).
  The class Multidimensional_Schemas implements the algorithm make_Dimesnions explained in the Chapter 4, section 4.4 Make dimensions, and page 50. This class implements too the algoritm SemanticInformation explained in Chapter 4, section 4.6 Semantic information, and page 54.
- multidimensionalBuilder (Source code in the Appendix A, section A.6 multidimensionalBuilder, and page 157).
  This package implement the algoritm Multidimensional modelling explained in Chapter 4, section 4.2 Multidimensional model, and page 45.
- databaseConnection (Source code in the Appendix A, section A.5 Package databaseConnection, and page 144).

The implementation of the system contains all the packages of the design and it has been introduced one more. This package is databaseConnection. This one has the interface to make the connection with a PostgreSQL database. It extracts the meta-data and the data of a relational database and it allows to create a new database and insert data into its.

The rest of the packages are the same ones than in the design.

5.3 Testing strategy

The implementation testing has been done according with an ascending strategy.

Firstly has been testing the individual algorithms that have been explained in the chapter 4. For each algorithms has been tested all the possible inputs. For each single input has been tested that the output was the goal of the algorithms.

Once each single algorithm was testes, all the algorithms have been integrated following the correct sequence. For each message transmission between two algorithms has been tested that the message was the correct one, for each possibility. It has not been made data volumes testing. When there is a huge amount of data in the multidimensional database. The performance of the system has to be in the OLAP server. It is because the user is going to make a multidimensional model only once.

The goal of a multidimensional model is the data accessing. This data accessing is made trough the OLAP server. Then it is the critical part off the system. The creation of the model it is only once.

CHAPTER 6

Conclusion

The goal of the thesis was the creation simple model to create multidimensional database from relational database. It model allows the creation of the semi-automatic support for multidimensional databases. The model has been designed in the most simplify way. The simplicity of the model is for two reasons:

- The first one is that this model has been thought to help user that are non experts in multidimensional data model to create their own multidimensional database. For that reason if it is created a complicated model, the user has to know more things about multidimensional databases and this is not the goal of the thesis.

- The other reason is that this tool is not trying to replace the multidimensional experts. It is a simple model for simple databases. A construction of a complicated multidimensional model need more analysis, and this tool may be used to oriented.

The goal of the informatics is to organize the data in order to make it accessible for everyone. This thesis has tried to attain the same goal for the multidimensional databases. These databases organized the data to help users in a different way than the operational ones, but with the same data, the difference is the organization.

6.1 Status

The thesis has made and implemented a model to create a multidimensional database. During all the report has been explaining that this model makes the design of a multidimensional database. It creates the database, and makes the first load. That is not a complete decision support system. To be a complete decision support system it is needed an ETL tool. But this is not the purpose of the thesis. It is necessary if the user want to generate a complete decision support system. It would be a very interesting extension of the thesis in order to generate the complete decision support system.

The result of the thesis has been the simple model that was suggested at the begging. It is a model designed to help non expert users. The user is creating a multidimensional model from its data without been an expert. This is possible because of the simplicity of the tool and the simplicity of the process. The semi-automatic support that has been built is not a complete commercial software.

On the other hand there are other cuestion about the model that may be improve with more analysis. For instance the way to store the data in the relational database usingn a snowflake model, or the optimization of dimensions.

6.2 Future

As it has been mentioned before, the software created is not commercial software. To be commercial software the GUI has to be modified and some improvements have to be done.

One of the things that has to been done in the tool is implement the dimension optimization. A user can different grain attributes. This attributes is possible that belong to the same dimension but they are divided in two. It has to be notifying to the user and create only one if both dimensions may be organized into one hierarchy.

Other interesting improvement is related with the time. The Data Warehouse has been defined as a historical database. Most of the times the data has to be relate with the time. For that reason it has to be considered. It is easy to detect is one of the dimensions is a time dimension. It is only needed to check if one of the grain attributes is one of the data type. Then the user has chosen the time by him from the database. If the user has not choose the data thee tool may suggest that there are no time dimension and usually it is very important. The problem is that sometimes the time in an operational database is the real time. There is not a specific field with a time attribute. Then it has to be defined a time in the data warehouse and introduced in the load this time with criteria. For instance his criteria may be stored the time when the data is extracted from the database. This is something that has to be contemplated and it is not very

complicated.

The integration of an ELT in the model to make a complete decision support system could be a very interesting improvement of the model. Although is not necessary for the construction, but it is necessary for a real system

A commercial software has to follow the recommendation of the usability engineering. It means that it has to be design for the user. The document ISO 9241-11 (1998) Guidance on Usability, also issued by the International Organization for Standardization, defines usability as: “The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.”
The software has been design for the user. The model has been modelling thinking in the easiest way for the user. Only is necessary a redesign of the interface with that. This design has to make easy and more natural the selection of the tables and attributes or the representation of the path.

Once the improvements, which have been explained, would be made “The semi-automatic tool for the creation of a Multidimensional model” may be a commercial tool.

This thesis has made a simple model and a simple product. It allows a lot of users to use of their information. The goal of the thesis and the goal of the computer science have been achieved.

APPENDIX A

Source Code

This appendix includes the source code of the tool that has been created in order to test the model.

A.1 Package Relational Model

A.1.1 Class Attribute

```
/*
 * Attribute.java
 * Created on November 15, 2006, 4:23 PM
 */
/*********************************************************************
 * Package

*********************************************************************/
package Relational_Model;

/**
 * @author Ana M Giral
 */
public class Attribute {
```

```
/*****************************************************************
* FIELD DECLARATION

*****************************************************************/
    public String Name;
    public String TableName;
    private String Type;
    private int theRestriction;

/*****************************************************************
     * PUBLIC METHODS OF CLASS Attribute
*****************************************************************/

/*****************************************************************/
    /** public Attribute(String N, String TN, String tR)
     *       Creates a new instance of Attribute
     *  Input parameters :
     *      String N    : Name of the new attribute
     *      String TN   : Name of the table of the new attribute
     *      String tR    : Type of the restriction of the attribute
     */
    public Attribute(String N, String TN, String tR) {
        Restriction Restr = new Restriction();
        Name = N;
        Type = "";
        TableName = TN;
        this.setRestriction(tR);
    }

/*****************************************************************/
    /** public void setType(String T)
     *      Set the type of an attribute
     *  Input parameters :
     *      String T    : Name of the SQL Type for the Attribute
     */
    public void setType(String T){
        this.Type = T;
    }

/*****************************************************************/
    /** public void setRestriction(String R)
     *      Set the Restriction of the Attribute
     *  Input Parameter:
     *      String R :  Name of the restriction
     */
    public void setRestriction(String tR){
        Restriction Restr = new Restriction();
        if (tR.equals("UNIQUE"))
            theRestriction = Restr.UNIQUE;
        else{
```

```
            if (tR.equals("PRIMARY KEY"))
                theRestriction = Restr.PRIMARY_KEY;
            else {
                if (tR.equals("FOREIGN KEY"))
                    theRestriction = Restr.FOREIGN_KEY;
                else
                    theRestriction = Restr.NONE;
            }
        }
    }
/*****************************************************************/
    /** public String getType
     *      Return the type of the Attribute
     *  Output parameters :
     *  String : SQL Type of the attrbiute
     */
    public String getType(){
        return this.Type;
    }
/*****************************************************************/
    /** public int getRestriction(String R)
     *      Return the Restriction of the Attribute
     *  Output parameters :
     *  int : Restriction.type of the attribute
     */
    public int getRestriction(){
        return theRestriction;
    }
/*****************************************************************/
    /** public boolean isPK()
     *      Return if the attribute has a prymary key restriction
     *  Output Paramters:
     *      boolean :   True if the attribute is a primary key
     *                  False if the attribute is not a prymary key
     */
    public boolean isPK(){
        Restriction Restr = new Restriction();

        if (this.theRestriction == Restr.PRIMARY_KEY)
            return true;
        else{
            return false;
        }
    }
/*****************************************************************/
    /** public boolean isFK()
     *      Return if the attribute has a foreign key restriction
     *  Output Paramters:
     *      boolean :   True if the attribute is a foreign key
     *                  False if the attribute is not a foreign key
     */
```

```
    public boolean isFK() {
        Restriction Restr = new Restriction();
        if (this.theRestriction == Restr.FOREIGN_KEY)
            return true;
        else{
            return false;
        }
    }
/*****************************************************************/
    /** public boolean isUNIQUE()
     *      Return if the attribute has a UNIQUE restriction
     *  Output Paramters:
     *      boolean :   True if the attribute has a UNIQUE restriction
     *                  False if the attribute has a UNIQUE restriction
     */
    public boolean isUNIQUE(){
        Restriction Restr = new Restriction();

        if (this.theRestriction == Restr.UNIQUE)
            return true;
        else{
            return false;
        }
    }
/*****************************************************************/
    /** public boolean isNONE()
     *      Return if the attribute has a NONE restriction
     *  Output Paramters:
     *      boolean :   True if the attribute has a NONE restriction
     *                  False if the attribute has a NONE restriction
     */
    public boolean isNONE() {
        Restriction Restr = new Restriction();

        if (this.theRestriction == Restr.NONE)
            return true;
        else{
            return false;
        }
    }
  /*****************************************************************/
  //  Class Restriction
  /*****************************************************************/
    public class Restriction{
        public static final int UNIQUE = 0 ;
        public static final int PRIMARY_KEY = 1;
        public static final int FOREIGN_KEY = 2;
        public static final int NONE = 3;
    }
}
```

A.1.2 Class Table

```
//*
 * Table.java
 * Created on November 15, 2006, 4:23 PM
 */
/*********************************************************************
 * Package
*********************************************************************/
package Relational_Model;
/*********************************************************************
 * Import Class
*********************************************************************/
import java.util.*;
import Relational_Model.Attribute.Restriction;
/**
 * @author Ana M Giral
 */
public class Table {

/*********************************************************************
 * FIELD DECLARATION
*********************************************************************/
    public String Name;
    private LinkedList Attributes;

/*********************************************************************
     * PUBLIC METHODS OF CLASS Table
*********************************************************************/
     * public Table(String N)
     *       Creates a new instance of a Table
     *  Input parameters :
     *      String N    : Name of the new TAble
     */
    public Table(String N) {
        Name = N;
        Attributes = new LinkedList();
    }
/*********************************************************************
     * public void AddAttribute(String N, String tR)
     *       create and add a new attribute to the table
     *  Input parameters :
     *      String N    : Name of the new Attribute
     *      String tR    : Type of the restriction of the attribute
     */
    public void AddAttribute(String N, String tR) {
        Attributes.add(new Attribute(N, this.Name, tR));
    }
/*********************************************************************
     * public void setAttributeType(String N, String T) {
```

```
     *       set the type of an attribute
     *  Input parameters :
     *      String N    : Name of the Attribute
     *      String T    : Name of the SQL Type for the Attribute
     */
    public void setAttributeType(String N, String T) {
        ListIterator ListIt = Attributes.listIterator();
        Attribute A;
        // find the attribute
        A = getAttribute(N);
        //set the type
        if (A != null){
            A.setType(T);
        }
    }
/*****************************************************************/
    /** public Attribute getAttribute(String Name)
     *      Return an attribute of the table
     *  Input Parameters :
     *      String Name : Name of the Attribute
     *  Output PArameter :
     *      Attributed with Attribute.Name = Name
     */
    public Attribute getAttribute(String Name){

        ListIterator Iter;
        Attribute A;
        Iter = Attributes.listIterator();
        while (Iter.hasNext()){
            A = (Attribute)Iter.next();
            if (A.Name.equals( Name ))
                return  A;
        }
        return null;
    }
/*****************************************************************/
    /** public LinkedList getAttributes()
     *      Return all the attributes of the table
     *  Output Parameters:
     *      LinkedList : List of Attributes af the table
     */
    public LinkedList getAttributes(){
        return Attributes;
    }
/*****************************************************************/
    /** public Attribute getPKAttribute()
     *      Return the first attribute with primary key *retriction in
     *     the table
     *  Output Parameter:
     *      Attribute : Attribute that is primary key
     */
```

```
    public Attribute getPKAttribute(){

        ListIterator Iter;
        Attribute A;

        Iter = Attributes.listIterator();
        while (Iter.hasNext()){
            A = (Attribute)Iter.next();
            if ( A.isPK() )
                return  A;
        }
        return null;
    }
/*****************************************************************/
    /** public AttributegetPKAttribute()
     *      Return the first attribute with UNIQUE restriction in the
     *       table
     *      Attribute : Attribute that has a UNIQUE restriction
     */
    public Attribute getUNIQUEAttribute(){

        ListIterator Iter;
        boolean end = false;
        Attribute A;
        Restriction Restr;

        Iter = Attributes.listIterator();
        while (Iter.hasNext()){
            A = (Attribute)Iter.next();
            if ( A.isUNIQUE() )
                return  A;
        }
        return null;
    }
    /*************************************************************/
    /**  public LinkedList getattributesTableNotPK()
     *      Return all the attributes al a table that have not got
     *      a 'PRYMARY KEY' restriction
     *  Returned value:
     *      LinkedList : List of Attributes
     */
    public LinkedList getattributesTableNotPK(){

        ListIterator Iter;
        Attribute A;
        LinkedList L = new LinkedList();

        Iter = Attributes.listIterator();
        while (Iter.hasNext()){
            A = (Attribute)Iter.next();
            if ( ! A.isPK() )
```

```
                L.add(A);
            }
            return L;
        }
/*****************************************************************/
        /** public Attribute getNumberPK()
         *      Return the number of attributes which have a
         *      'PRIMARY KEY' restriction in the table
         */
        public int getNumberPK(){

            ListIterator Iter;
            Attribute A;
            int i = 0;

            Iter = Attributes.listIterator();
            while (Iter.hasNext()){
                A = (Attribute)Iter.next();
                if ( A.isPK() )
                    i++;
            }
            return i;
        }
}
```

A.1.3 Class FKRelation

```
/*
 * FKRelation.java
 * Created on November 15, 2006, 4:24 PM
 */
/*****************************************************************
 * Package
*****************************************************************/
package Relational_Model;
/**
 * @author Ana M Giral
 */
public class FKRelation {

/*****************************************************************
     * FIELD DECLARATION
*****************************************************************/
    public Attribute Attr;
    public Attribute referencedAttr;

/*****************************************************************
     * PUBLIC METHODS OF CLASS FKRelation
  ****************************************************************/
```

```
    /** Creates a new instance of FKRelation */
    public FKRelation( Attribute A, Attribute RA) {
        Attr = A;
        referencedAttr = RA;
    }
}
```

A.1.4 Class Path

```
/*
 * Path.java
 * Created on November 15, 2006, 4:28 PM
 */
/*****************************************************************
 * Package

*****************************************************************/
package Relational_Model;
/*****************************************************************
 * Import Class
*****************************************************************/
import java.util.*;

/**
 * @author Ana M Giral
 */
public class Path {

/*****************************************************************
 * FIELD DECLARATION
 *****************************************************************/
    private Attribute Start ;
    private Attribute End ;
    private LinkedList Relations;

/*****************************************************************
    * PUBLIC METHODS OF CLASS Path
  *****************************************************************/

/*****************************************************************/
    /** public Path(Attribute S)
     *       Creates a new instance of the class Path
     *  Input parameters :
     *      Attribute S : Start of the new path
     */
    public Path(Attribute S) {
        this.Start = S;
        this.End = null;
        this.Relations = new LinkedList();
```

```
    }

/*****************************************************************/
    /** public Path(Attribute S, Attribute E, LinkedList TheRelations )
     *       Creates a new instance of the class Path
     *  Input parameters :
     *      Attribute S : Start of the new path
     *      Attribute E : End of the new path
     *      LinkedList TheRelations : LinkedList with the FKRelations *
     * of the path
     */
    public Path(Attribute S, Attribute E, LinkedList TheRelations ) {
        this.Start = S;
        this.End = E;
        this.Relations = (LinkedList)TheRelations.clone();
    }

/*****************************************************************/
    /** public void addRelation(FKRelation FK)
     *      add a new relation at the end of the Path
     *  Input parameters :
     *      FKRelation FK : Relation to be added
     */
    public void addRelation(FKRelation FK){
        this.Relations.addLast(FK);
    }
/*****************************************************************/
    /** public void addEnd(Attribute Grain)
     *      modify the end of the path
     *  Input parameters :
     *     Attribute Grain : new end of the path
     */
    public void addEnd(Attribute Grain){
        this.End = Grain;
    }
/*****************************************************************/
    /**  public LinkedList getRelations()
     *      return the list of FKRelation of the path
     *  Output parameters :
     *     LinkedList : List of FKRealtion
     */
    public LinkedList getRelations(){
        return this.Relations;
    }
/*****************************************************************/
    /**  public  FKRelation getFirstRelation()
     *      return the first relation of the path
     *  Output parameters :
     *     FKRelation :  First FKRelation of the Path
     */
    public FKRelation getFirstRelation(){
```

```
        if ( Relations.size() == 0 ) return null;
        else return (FKRelation) this.Relations.getFirst();
    }
/*****************************************************************/
    /**  public FKRelation getLastRelation()
     *       return the last relation of the path
     *  Output parameters :
     *     FKRelation :  Last FKRelation of the Path
     */
    public FKRelation getLastRelation(){
        if ( Relations.size() == 0 ) return null;
        else return (FKRelation) this.Relations.getLast();
    }
/*****************************************************************/
    /**  public ListIterator getIteratorRelation()
     *       return a iterator of the FKRelation list of the path
     *  Output parameters :
     *     ListIterator :  Iterator of the list of FKRelations
     */
     public ListIterator getIteratorRelation(){
        if ( Relations.size() == 0 ) return null;
        else return this.Relations.listIterator();
    }
/*****************************************************************/
    /**  public Attribute getStart
     *       return the start Attribute of the path
     *  Output parameters :
     *    Attribute Start : start attribute.
     */
    public Attribute getStart(){
            return this.Start;
    }
/*****************************************************************/
    /**  public Attribute getStart
     *       return the start Attribute of the path
     *  Output parameters :
     *    Attribute Start : start attribute.
     */
    public Attribute getEnd(){
            return this.End;
    }
/*****************************************************************/
    /**  public int sizePath()
     *       return the number of FKRelations in the path
     *  Output parameters :
     *    int : number of FKRelations in the path
     */
    public int sizePath(){
        return this.Relations.size();
    }
/*****************************************************************/
```

```
    /**  public boolean containsTable(Attribute A)
     *      return true if the Path contains the table of an attribute
     *  Input Attributes :
     *      Attribute A : Attribute to look for the table
     *  Output parameters :
     *    boolean: True if the table of A is in the path
     */
    public boolean containsTable(Attribute A){

        ListIterator LIt = Relations.listIterator();
        while (LIt.hasNext()){
            FKRelation FKP = (FKRelation) LIt.next();
            if (!LIt.hasNext()){
                if (FKP.Attr.TableName.equals(this.End.TableName))
                    return false;
                else{
                    if(A.TableName.equals(FKP.referencedAttr.TableName)
                    )
                        return true;
                    if ( A.TableName.equals( FKP.Attr.TableName ))
                        return true;
                }
            }
            else{
                if (A.TableName.equals( FKP.referencedAttr.TableName ))
                    return true;
                if ( A.TableName.equals( FKP.Attr.TableName ))
                    return true;
            }
        }
        return false;
    }
/*****************************************************************/
    /**  public boolean containsHieraTable(Attribute A)
     *      return true if the of a hierarchy Path contains the table
of an attribute
     *  Input Attributes :
     *      Attribute A : Attribute to look for the table
     *  Output parameters :
     *    boolean: True if the table of A is in the path
     */
    public boolean containsHieraTable(FKRelation FKR){

        ListIterator LIt = Relations.listIterator();
        while (LIt.hasNext()){
             if ( FKR.Attr.TableName ==
             ((FKRelation)LIt.next()).Attr.TableName ) {
                return true;
            }
        }
        return false;
```

```
/*****************************************************************/
    /**  public String showPath()
     *      show a Path
     *  Output parameters :
     *    String: String with the representation of the path
     */
    public String showPath(){

        String StringPath = this.Start.TableName;
        ListIterator ListFK = this.getIteratorRelation();
        if (ListFK == null)
             return StringPath;
        else {
            StringPath = StringPath + " -> ";
        }
        while (ListFK.hasNext() ){
            FKRelation FKR = (FKRelation)ListFK.next();
            StringPath = StringPath + FKR.referencedAttr.TableName + "
                        . "+ FKR.Attr.TableName + "-> ";
        }
        StringPath = StringPath + this.getEnd().TableName;
        return StringPath;
    }
}
```

A.1.5 Class Paths

```
/*
 * Paths.java
 * Created on November 27, 2006, 1:11 PM
 */
/*********************************************************************
 * Package
*********************************************************************/
package Relational_Model;
/*********************************************************************
 * Import Class
*********************************************************************/
import java.util.*;
/**
 * @author Ana M Giral
 */
public class Paths{

/*****************************************************************
     * FIELD DECLARATION
*********************************************************************/
    private LinkedList PathList;
```

```
/*****************************************************************
     * PUBLIC METHODS OF CLASS Paths
*****************************************************************/

/*****************************************************************/
    /** public Path(Attribute S)
     *       Creates a new instance of the class Paths
     */
    public Paths(){
        this.PathList = new LinkedList();
    }
/*****************************************************************/
    /** public boolean  hasPath()
     *       return true if thre are element in the list
     */
    public boolean  hasPath(){
        if ( this.PathList.size() == 0 ) return false;
        else return true;
    }
/*****************************************************************/
    /**  public Path  nextPath()
     *      return and remove the first path of the list
     *  Output parameters:
     *      Path : PAth that has been remove for the list
     */
    public Path  nextPath(){

        if ( PathList.size() == 0 ) {
            return null;
        } else{
            return (Path)this.PathList.removeFirst();
        }
    }
/*****************************************************************/
    /**  public void addPath(Path P)
     *      Add a new Path to the List of Path
     *  Input parameters:
     *      Path P : Path to be added
     */
    public void addPath(Path P){
        if (! this.PathList.contains(P)){
            this.PathList.add(P);
        }
    }
/*****************************************************************/
    /**  public void addPath(Path P)
     *      Add a new Path to the List of Path
     *  Input parameters:
     *      Path P : Path to be added
     */
```

```
    public void addPaths(Paths P){
        ListIterator listIt = P.getIterator();
        while (listIt.hasNext())
            this.addPath((Path)listIt.next());

    }
/*****************************************************************/
    /**  public int size()
     *       return the number of Path in the list
     *  Output parameters:
     *       int : size of list
     */
    public int size(){
        return this.PathList.size();
    }
  /*****************************************************************/
    /**   public void clear()
     *     remove all the elements of the list
     */
    public void clear(){
        this.PathList.clear();
    }
  /*****************************************************************/
    /**  public Path getPath(int i)
     *       return the path in the position i of the list
     *  Input parameters :
     *       int i : Position of the path to be returned
     *  Output parameters:
     *       Path : the Path
     */
    public Path getPath(int i){
        return (Path)this.PathList.get(i);
    }
  /*****************************************************************/
    /**  public ListIterator getIteratorRelation()
     *       return a iterator of the Path's list
     *  Output parameters :
     *     ListIterator :  Iterator of the list
     */
    public ListIterator getIterator(){
        return this.PathList.listIterator();
    }
}
```

A.1.6 Class Relational_Model

```
/*
 * Relational_Model.java
 *
```

```
 * Created on November 15, 2006, 4:27 PM
 *
 */
/*********************************************************************
 * Package
*********************************************************************/
package Relational_Model;
/*********************************************************************
 * Import Class
*********************************************************************/
import java.util.*;

/**
 *
 * @author Ana M Giral
 */
public class Relational_Model {


/*********************************************************************
     * FIELD DECLARATION
*********************************************************************/
    private LinkedList Tables;
    private LinkedList FKRelations;
    private LinkedList SelectedAttr;
    private LinkedList GrainAttr;
    private Attribute Measure;

  /*********************************************************************
     * PUBLIC METHODS OF CLASS Relational_Model
*********************************************************************/
/*********************************************************************/
    /** Creates a new instance of Relational_Model */
    public Relational_Model() {
        Tables = new LinkedList();
        FKRelations = new LinkedList();
        SelectedAttr = new LinkedList();
        GrainAttr = new LinkedList();
    }
/*********************************************************************/
    /** public void InsertTable (String Name)
     *      Insert a new Table in the model
     *  Input Parameters :
     *      String Name -> Name of the Table in the Database
     */
    public void InsertTable(String Name){
        Tables.add(new Table(Name));
    }
/*********************************************************************/
    /** public void InsertAttTable(String TableName, String AttrName,
String AttrType)
```

```
     *      Insert Attributes in a table
     *  Input parameters :
     *     String TableName: Name of the Table that contains the
     *           attribute
     *      String AttrName : Name of the attribute
     *      String AttrRestriction : Restriction of the attribute
     *           ( PRYMARY KEY | UNIQUE )
     **/
    public int InsertAttTable(String TableName, String AttrName, String
        AttrRestriction){

        Table T = searchTable(TableName);
        if (T != null ){
            T.AddAttribute(AttrName, AttrRestriction);
            return 0;
        }else
            return -1;
    }
/*****************************************************************/
    /** public int InsertFKRelation(String TName, String ATName, String
        RefTName, String RefATName)
     *      Insert a relation in the model
     *  Input Parameters :
     *      String TName : Name of the table of foreign key attribute
     *      String ATName   : Name of the foreign key attribute
     *      String RefTName : Name of the table that is referenced by a
     *           foreign key attribute
     *      String RefATName: Name of the attribute that is reference
     *           by a foreign key attribute
     *  Output parameters :
     *      int ->   0 : The opetarion has been realized
     *              -1 : The operation has not been realized
     */
    public int InsertFKRelation(String TName, String ATName, String
        RefTName, String RefATName){

        Table T1 = searchTable(TName);
        Table T2 = searchTable(RefTName);
        // The tables exist
        if (T1 != null && T2!= null){
            if (T1 != null && T2 != null){
                FKRelations.add( new FKRelation(
                 T1.getAttribute(ATName),T2.getAttribute(RefATName)));
                // Insert the FOREIGN KEY RESTRICTION TO THE ATTRIBUTE
                if (T1.getAttribute(ATName).getRestriction() ==
                 Attribute.Restriction.NONE )
                    T1.getAttribute(ATName).setRestriction("FOREIGN
                                                           KEY");
                return 0;
            }else
                return -1;
```

```
        }
        return -1;
    }
/*****************************************************************/
    /** LinkedList getTables()
     *      Return all Names of the tables that are in the Tables list
     *  Output parameters :
     *      LinkedList  : String LinkedList with the names of the
     *           Tables in Tables list
     */
    public LinkedList getTables(){

        LinkedList L = new LinkedList();

        ListIterator ListIt = Tables.listIterator();
        while (ListIt.hasNext()){
            L.add( ((Table)ListIt.next()).Name );
        }
        return L;
    }
/*****************************************************************/
    /** public void SelectMeasure(String TableName, String AttrName)
     *      select an attribute as a measure attribute
     *  Input parameters :
     *      String TableName: Name of the Table that contains the
     *           attribute
     *      String AttrName : Name of the attribute
     */
    public void SelectMeasure(String TableName, String AttrName){
        this.Measure =
                this.searchTable(TableName).getAttribute(AttrName);
    }
/*****************************************************************/
    /**  public Attribute getMeasure()
     *      return the measure attribute
     *  Output parameters :
     *      Attribute : measure attribute
     */
    public Attribute getMeasure(){
        return this.Measure;
    }

/*****************************************************************/
    /** public void SelectAttribute(String TableName, String
     *  AttributeName)
     * Select an attribute to be in the multidimensional model
     *  Input parameters :
     *      String TableName: Name of the Table that contains the
     *           attribute
     *      String AttributeName : Name of the attribute
     */
```

```
    public void SelectAttribute(String TableName, String
        AttributeName){
        Table T = searchTable(TableName);
        SelectedAttr.add(T.getAttribute(AttributeName));
    }

/*****************************************************************/
    /** public void SelectAttribute(Attribute A)
     * Select an attribute to be in the multidimensional model
     *  Input parameters :
     *      Attribute A: Attribute to be selected
     */
    public void SelectAttribute(Attribute A){
        SelectedAttr.add(A);
    }

/*****************************************************************/
    /** public LinkedList SelectTable(String Name)
     * Select a table to show and allow the selection of the attributes
     * of this
     *  Input parameters :
     *      String Name: Name of the Table
     *  Output parameters:
     *      LinkedList : list with the attributes of the table
     */
    public LinkedList SelectTable(String Name){
        return  searchTable(Name).getAttributes();
    }

/*****************************************************************/
    /** public void SelectAttribute(Attribute A)
     *      Select an selected attribute as a Grain attribute
     *  Input parameters :
     *      Attribute A: Attribute to be selected as a Grain Attribute
     */
    public void SelectGrainAttribute(Attribute A){

        if (this.SelectedAttr.contains(A)){
            this.GrainAttr.add(A);
        }
    }


/*****************************************************************/
    /** protected Table searchTable (String Name){
     *      Return the table with the specified name
     *  Input parameters :
     *      String Name : Name of the table that has to be returned
     */
    public Table searchTable(String Name){
```

```
        // instanced variables
        ListIterator Iter;
        boolean end = false;
        Table T;

        Iter = Tables.listIterator();
        while (true){
            if (Iter.hasNext()) {
                T = (Table)Iter.next();
                if (T.Name.equals(Name)) {
                    return  T;
                }
            }else
                return null;
        }
    }
/*****************************************************************/
    /** public LinkedList getSelectedAttributes()
     *     Return all the selected attributes of the model
     *  Output Parameters:
     *      LinkedList : List of lists of Attributes
     */
    public LinkedList getSelectedAttributes(){
        return this.SelectedAttr;
    }
/*****************************************************************/
    /** public Attribute getPKAttribute(String TableName)
     *      Return the first attribute with a primary key restriction
     *   of a table
     * Input parameter:
     *      String Name : Name of the table
     *  Output Parameter:
     *      Attribute : Attribute that is primary key
     */
    public Attribute getPKAttribute(String TableName) {
        Table T = searchTable(TableName);
        return T.getPKAttribute();
    }
/*****************************************************************/
    /** public Attribute getUNIQUEAttribute(String TableName)
     *      Return the first attribute with a UNIQUE restriction of a
     *      table
     * Input parameter:
     *      String Name : Name of the table
     *  Output Parameter:
     *      Attribute : Attribute that has a UNIQUE restriction
     */
    public Attribute getUNIQUEAttribute(String TableName) {
        Table T = searchTable(TableName);
        return T.getUNIQUEAttribute();
    }
```

```
  /*****************************************************************/
    /** public LinkedList getRelationsFromPK(String Table)
     *     Return the relations that the primary key of the table is
     *     is referenced
     * Input parameter:
     *      String Table : Name of the table
     *  Output Parameter:
     *      LinkedList : list of FKRelation
     */
    public LinkedList getRelationsFromPK(String Table){

        LinkedList L = new LinkedList();
        ListIterator ListIt = this.FKRelations.listIterator();
        int numberPK = this.searchTable(Table).getNumberPK();

        while ( ListIt.hasNext()){
            FKRelation FKR = (FKRelation)ListIt.next();
            // normal direction of the relacion
             if ( numberPK > 1 ){ // more than one foreign key, then
             the //RElations fromPK are too
                if (FKR.Attr.TableName.equals(Table)&&FKR.Attr.isPK())
                    L.add(FKR);
            } else{
                if ( FKR.referencedAttr.TableName.equals(Table) ){
                    if (FKR.referencedAttr.isPK())
                        L.add(FKR);
                }
            }
        }
        return L;
    }
/*****************************************************************/
    /** private LinkedList getRelationsFromFK(Attribute A)
     *     Return the relations where an Attribute reference a primary
     *    key
     * Input parameter:
     *      Attribute A : Attribute that is a feriegn key
     *  Output Parameter:
     *      LinkedList : list of FKRelation
     */
    private LinkedList getRelationsFromFK(Attribute A) {
        LinkedList L = new LinkedList();
        ListIterator ListIt = this.FKRelations.listIterator();
        while ( ListIt.hasNext()){
            FKRelation FKR = (FKRelation)ListIt.next();
            // normal direction of the relacion
            if ( FKR.Attr == A  )
                L.add(FKR);
        }
        return L;
    }
```

```
/*****************************************************************/
    /** private LinkedList getRelationsToPK(String Table)
     *     Return the relations the primary key of a table is
     *           referenced
     * Input parameter:
     *      String Table : name of the table
     *  Output Parameter:
     *      LinkedList : list of FKRelation
     */
    public LinkedList getRelationsToPK(String Table){

        LinkedList L = new LinkedList();
        ListIterator ListIt = this.FKRelations.listIterator();
        while ( ListIt.hasNext()){
            FKRelation FKR = (FKRelation)ListIt.next();
            if ( FKR.Attr.TableName.equals(Table) )
                L.add(FKR);
        }
        return L;
    }
/*****************************************************************/
    /**  public Paths makePathsFromMeasure(Attribute A)
     *      return a set of Paths from a grain Attribute to
     *      the measure attribute
     *  Input parameters:
     *      Attribute A: Grain attribute
     *  Output parameters:
     *      Paths: List of path between the grain attribute and the
     *      measure
     */
    public Paths makePathsFromMeasure(Attribute A){

        // list of paths uncompleted
        Paths thePaths = new Paths();
        // list of paths that are completed
        Paths FinalPaths = new Paths();

        // Path inicialization
        Path thePath = new Path(A);
        thePath.addEnd(A);

        // add the first path, a path without relations
        thePaths.addPath(thePath);

        //while there were more possibles paths
        while ( thePaths.size() > 0 ){
            //Get and remove one path from the list of paths
            Path P = thePaths.nextPath();
            // if the table is the Measure add the path to the list of
            //final Paths
            if (P.getEnd().TableName.equals(this.Measure.TableName)){
```

```
                // Add the end of the path
                P.addEnd(Measure);
                FinalPaths.addPath(P);
                System.out.println(P.showPath());
            } else{ // The last Table in the path is not the measure
                LinkedList Rel=
                          getRelationsFromPK(P.getEnd().TableName);

                // There are more relations between the table an other
                while (Rel.size()> 0){

                    FKRelation FKR = (FKRelation)Rel.removeFirst() ;
                    Attribute AR;
                    Path P1 = new Path(P.getStart(), P.getEnd(),
                                       P.getRelations());
                   if (FKR.Attr.TableName.equals(
                               P1.getEnd().TableName)){
                        // TAble with bidirectional relation the
                         //relation end in the referended attribute
                        AR = FKR.referencedAttr;
                    } else {
                        AR = FKR.Attr;
                    }
                    // If the final table is not yet in the path add to
                    //the Path list
                    if ( ! P.containsTable(AR) ){  //no cycles
                        // Add a end to the path
                        P1.addEnd(AR);
                        // Add the relation to the path
                        P1.addRelation(FKR);
                        // Add the path to the list of path to evaluate
                        thePaths.addPath(P1);
                    }
                }
            }
        }
        if (A.isFK()){
            FinalPaths.addPaths( PathsFromFKWeightOne(A));
        }
        return FinalPaths;
    }
/*****************************************************************/
    /**  public Paths makePathsFromMeasure(Attribute A)
     *       return a set of Hierarchy Paths from a grain Attribute
     *  Input parameters:
     *       Attribute A: Grain attribute
     *  Output parameters:
     *       Paths: List of path between the grain attribute and the
     *            measure
     */
    public Paths makeHierarchies(Attribute A){
```

```
        //foreign key con la pk de la tabla
        String Table;
        FKRelation FK;
        // list of hieraquis uncompleted
        Paths thePaths = new Paths();
        // list of hierachies that are completed
        Paths FinalPaths = new Paths();

        // First path to evaluated
        Path thePath = new Path(A);
        thePath.addEnd(A);

        // add the first path, a path without relations
        thePaths.addPath(thePath);

        //while there were possibles paths
        while ( thePaths.size() > 0 ){

            //Get a path to evaluate
            Path P = thePaths.nextPath();
            // There are only a PK, then the hierarcky can follow
            LinkedList Rel= getRelationsToPK(P.getEnd().TableName);
            if (Rel.size() == 0 ){
                FinalPaths.addPath(P);
            } else {
                while (Rel.size()> 0){

                    FKRelation FKR = (FKRelation)Rel.removeFirst() ;
                    // If the final table is not yet in the path add to
                    //the Path list
                    if ( ! P.containsHieraTable(FKR) ){
                        if (this.searchTable(
                     FKR.referencedAttr.TableName).getNumberPK() == 1){
                                Path P1 = new Path(P.getStart(),
                                      P.getEnd(), P.getRelations());
                                P1.addRelation(FKR);
                                P1.addEnd(FKR.referencedAttr); // the
                              //referenced attribute is the end of the path
                                thePaths.addPath(P1);
                            }
                        }
                    }
                }
            }
        }
         return FinalPaths;
    }

/*****************************************************************/
    /**  public Paths makePathsFromMeasure(Attribute A)
     *      return a set PAths with weight 1 from and attribute that it
```

```
     *      is foreign key to the primary key
     *  Input parameters:
     *      Attribute A: Foreign key attribute
     *  Output parameters:
     *      Paths: List of path between the grain attribute and the
     *           measure
     */
    private Paths PathsFromFKWeightOne(Attribute A) {
        Paths FinalPaths = new Paths();
        ListIterator listIt =this.getRelationsFromFK(A).listIterator();
        while (listIt.hasNext()){
            FKRelation FKR = (FKRelation) listIt.next();
             if (FKR.referencedAttr.TableName.equals
             (this.Measure.TableName)){
                Path P = new Path(A);
                P.addEnd(FKR.referencedAttr);
                P.addRelation(FKR);
                FinalPaths.addPath(P);
            }
        }
        return FinalPaths;
    }
}
```

A.2 Package Multidimensional Model

A.2.1 Class AttributeMulti

```
/*
 * AttributeMulti.java
 * Created on November 23, 2006, 5:56 PM
 */
/*********************************************************************
 * Package
*********************************************************************/
package Multidimensional_Model;
/*********************************************************************
 * Import Class
*********************************************************************/
import Relational_Model.Attribute;

/**
 * @author Ana M Giral
 */
public class AttributeMulti {


/*****************************************************************
     * FIELD DECLARATION
```

```
  ****************************************************************/
    public String Name;
    private boolean isGrain;
    private Attribute AttributeRef;
    private Semantic SemanticInfo;

/*****************************************************************
     * PUBLIC METHODS OF CLASS Attribute
*****************************************************************/
    /** public AttributeMulti(Attribute A, boolean isG)
     *      Creates a new instance of AttributeMulti and make a mark if
     *      it is a grain attribute
     *  Input parameters :
     *      Attribute A : Attribute relational that is referenced by
     *                    the new AttributeMulti
     *      boolean isG: if true the new AttributeMulti is a grain one
     */
    public AttributeMulti(Attribute A, boolean isG) {

        AttributeRef = A;
        isGrain = isG;
        Name = A.TableName+"_"+A.Name;
    }
/*****************************************************************/
    /** public void addSemanticInfo(AttributeMulti AM, int TheType)
     *      Add semantic information to an attribute
     *  Input parameters :
     *      AttributeMulti AM : AttributeMulti for the semantic
     *           information
     *      int TheType : semantic type of the attribute
     */
    public void addSemanticInfo(AttributeMulti AM, int TheType){
        SemanticInfo = new Semantic(TheType, AM);
    }
/*****************************************************************/
    /** public Attribute getReferencedAttribute()
     *      return the relational attribute that the class is
     *     referencing
     *  Output parameters :
     *      Attribute : relational attribute referenced
     */
    public Attribute getReferencedAttribute(){
        return this.AttributeRef;
    }
/*****************************************************************/
    /** public Semantic getSemanticInfo()
     *      return the semantic information of the attribute
     *  Output parameters :
     *      Semantic : semantic information
     */
    public Semantic getSemanticInfo(){
```

```
        return SemanticInfo;
    }
/*****************************************************************/
    /** public String showSemanticInfo()
     *      return a string with the semantic information of the
     *     attribute
     *  Output parameters :
     *      String : string with semantic information
     */
    public String showSemanticInfo(){
       String S ="";
       if (SemanticInfo.isHierarchy())
           S = S + "Hierarchy Attribute. Less level : " +
               SemanticInfo.getAttribute().Name;
       else
           S = S + "Descriptive Attribute of : " +
                SemanticInfo.getAttribute().Name;
       return S;
    }
/*****************************************************************/
    /** public boolean isGrain()
     *      return true if the multidimensional attribute is a grain
     *     attribute
     *  Output parameters :
     *      boolean : true
     */
    public boolean isGrain() {
        return this.isGrain;
    }
/*****************************************************************/
    /** public boolean isHierarchy()
     *      return true if the multidimensional attribute is a a
     *      hierarchy attribute
     *  Output parameters :
     *      boolean : true in is hierarchy attribute
     */
    public boolean isHierarchy(){
        return this.getSemanticInfo().isHierarchy();
    }
}
```

A.2.2 Class Semantic

```
/*
 * Semantic.java
 * Created on November 23, 2006, 5:56 PM
 */
/*****************************************************************
```

```
 * Package
*********************************************************************/
package Multidimensional_Model;
/*********************************************************************
 * Import Class
*********************************************************************/
import multidimensionalbuilder.Type;
/**
 * @author Ana M Giral
 */
public final class Semantic {

/*******************************************************************
     * FIELD DECLARATION
  ******************************************************************/
    private int TheType;
    private AttributeMulti AM;

    /** Creates a new instance of Semantic */
    public Semantic(int T, AttributeMulti A) {
        TheType = T;
        AM= A;
    }
/*******************************************************************/
    /**    public int getTheType(){
     *      return the semantic type
     *  Output parameters :
     *      int : Type of the restriction
     */
    public int getTheType(){
        return TheType;
    }
/*******************************************************************/
    /**    public AttributeMulti getAttribute(){
     *      return the refered Attribute
     */
    public AttributeMulti getAttribute(){
        return AM;
    }
/*******************************************************************/
    /**    public AttributeMulti getAttribute(){
     *      return true is the type is HIERARCHY
     */
    boolean isHierarchy() {
        Type AType = new Type();
        if (TheType == AType.Hierarchy )
            return true;
        else
            return false;
    }
}
```

A.2.3 Class Dimension

```
/*
 * Dimension.java
 * Created on November 23, 2006, 5:55 PM
 */
/*********************************************************************
 * Package
*********************************************************************/
package Multidimensional_Model;
/*********************************************************************
 * Import Class
*********************************************************************/
import Relational_Model.*;
import java.util.*;
import javax.swing.JTree;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreeSelectionModel;
/**
 * This class is used to describe a multidimensional Dimension with all
 * the inherits operations
 * @author Ana M Giral
 */
public class Dimension {


/*********************************************************************
     * FIELD DECLARATION
 *********************************************************************/
    public String Name;
    public AttributeMulti GrainAttribute;
    private LinkedList Attributes;
    private Path pathToMeasure;
    private Path pathHierachy;

    private String CREATE_TABLE;
    private JTree tree;
    private String TransformationSchema;

/*********************************************************************
     * PUBLIC METHODS OF CLASS Attribute
 *********************************************************************/

/*********************************************************************
     * public Dimension(String TheName, Attribute A, Path ptM , Path
ptH) {
     *      Creates a new instance of Dimension
```

```
     *  Input parameters :
     *  String TheName : Name of the dimension
     *  Attribute A : Grain attribute of the dimension
     *  Path ptM : Path from the grain attribute to the measure
     *  Path ptH : Hierarchy path
     */
    public Dimension(String TheName, Attribute A, Path ptM ,Path ptH) {
        Name = TheName;
        Attributes = new LinkedList();
        pathToMeasure = ptM;
        GrainAttribute = new AttributeMulti(A, true);
        Attributes.add(GrainAttribute);
        CREATE_TABLE = "";
        TransformationSchema="";
        pathHierachy= ptH ;
    }
/*****************************************************************
     * public Dimension( Path ptM , Path ptH) {
     *      Creates a new instance of Dimension
     *  Input parameters :
     *  Path ptM : Path from the grain attribute to the measure
     *  Path ptH : Hierarchy path
     */
    public Dimension(Path ptM , Path ptH) {
        Name = ptM.getStart().TableName+"_"+ptM.getStart().Name;
        Attributes = new LinkedList();
        pathToMeasure = ptM;
        GrainAttribute = new AttributeMulti(ptM.getStart(), true);
        Attributes.add(GrainAttribute);
        pathHierachy= ptH ;
    }
/*****************************************************************/
    /** public void AddAttribute(Attribute A)
     *      Create and add a new attribute to the multidimensional
     *     model
     *  Input Parameters :
     *      Attribute -> Attribute from the Relational model that is
     *      the corresponcende with the new
     */
    public AttributeMulti AddAttribute(Attribute A){
        AttributeMulti AM = new AttributeMulti(A, false);
        Attributes.add(AM);
        return AM;
    }
/*****************************************************************/
    /** public AttributeMulti getAttribute(Attribute Attr)
     *      get an Attribute from the Dimension
     *  Input Parameters:
     *      Attribute Attr-> Attribute of the attribute
     *      multidimensional to be returned
     */
```

```
    public AttributeMulti getAttribute(Attribute Attr){

        AttributeMulti A;
        ListIterator ListIt = Attributes.listIterator();
        while (ListIt.hasNext()){
            A = (AttributeMulti)ListIt.next();
            if ( A.getReferencedAttribute() == Attr) return A;
        }
        return null;
    }
/*****************************************************************/
    /** public AttributeMulti getGrainAttribute(String Attr)
     *      get the Grain  Attribute of the Dimension
     *  Input Parameters:
     *      String Attr-> Name of the attribute to be returned
     */
    public AttributeMulti getGrainAttribute(){
        return this.GrainAttribute;
    }
/*****************************************************************/
    /** public LinkedList getAttributes()
     *      return a list with all the attributes of a dimemension
     *  OutputParameters:
     *      LinkedList -> List of AttributesMulti
     */
    public LinkedList getAttributes(){
        return this.Attributes;
    }
/*****************************************************************/
    /** public int numberDescriptiveAttribute(AttibuteMulti AM)
     *      return a list with all the names of theattributes of a
     *      dimemension
     *  OutputParameters:
     *      LinkedList -> List of String
     */
    public int numberDescriptiveAttribute(AttributeMulti AM){

        /***/
        int i = 0;
        ListIterator listIT = this.Attributes.listIterator();
        while (listIT.hasNext()){
            Semantic sem =
                 ((AttributeMulti)listIT.next()).getSemanticInfo();
            if ( sem.getTheType() ==
                 multidimensionalbuilder.Type.Descriptive &&
                 sem.getAttribute() == AM)
                i++;
        }
        return i;
    }
/*****************************************************************/
```

```
    /** public LinkedList getDescriptiveAttribute
     *      return a list with all the names of the attributes of a
     *      dimension
     *  OutputParameters:
     *      LinkedList -> List of String
     */
     public LinkedList getDescriptiveAttribute(AttributeMulti AM){

        LinkedList LI = new LinkedList();
        ListIterator listIT = this.Attributes.listIterator();
        while (listIT.hasNext()){
            AttributeMulti AMd = (AttributeMulti)listIT.next();
            Semantic sem = AMd.getSemanticInfo();
            if ((!AMd.isHierarchy())&& sem.getAttribute().equals( AM ))
                LI.add(AMd);
        }
        return LI;
    }
/*********************************************************************/
    /** public AttributeMulti getHierarchyAttribute(AttributeMulti AM)
     *      return the next level attribute in a hierarchy from a
     *      multidimensional attribute
     *  Input parameters:
     *      AttributeMuilti : attribute with less level
     *  Output parameters:
     *      AttributeMulti: next hiercharchy attribute
     */
    public AttributeMulti getHierarchyAttribute(AttributeMulti AM){

        ListIterator listIT = this.Attributes.listIterator();
        while (listIT.hasNext()){
            AttributeMulti AMh = (AttributeMulti)listIT.next();
            Semantic sem = AMh.getSemanticInfo();
            if ( AMh.isHierarchy() && (sem.getAttribute() == AM ) &&
                 !(AMh.equals(AM)))
                return AMh ;
        }
        return null;
    }
/*********************************************************************/
    /** public Path getPath()
     *      return the path from the grain attribute to the measure
     */
    public Path getPath() {
        return this.pathToMeasure;
    }
/*********************************************************************/
    /** public Path getPath()
     *      return the star schema of the dimension
     */
    public String getSchema() {
```

```
        return this.CREATE_TABLE;
    }
/*****************************************************************/
    /** void orderAttributes()
     *      order the attributes of the dimension into a JTree
     */
    public void orderAttributes(){
        //make node
        DefaultMutableTreeNode top = new
                DefaultMutableTreeNode("Dimension " + this.Name);
        // make tree
        DefaultMutableTreeNode hierarchy = null;
        DefaultMutableTreeNode aux = null;
        DefaultMutableTreeNode descriptive = null;

        LinkedList newAttributes = new LinkedList();
        AttributeMulti AM = this.GrainAttribute;
        while (!(AM == null)){
            if (AM == this.GrainAttribute ){
                hierarchy = new DefaultMutableTreeNode(AM.Name);
                aux = hierarchy;
                top.add(hierarchy);
            }
            else{
                aux = hierarchy;
                hierarchy = new DefaultMutableTreeNode(AM.Name);
                aux.add(hierarchy);
            }

            newAttributes.add(AM);

            ListIterator DescriptiveAttr =
                 this.getDescriptiveAttribute(AM).listIterator();
            while (DescriptiveAttr.hasNext()){
                AttributeMulti AMd = (AttributeMulti)
                          DescriptiveAttr.next();
                newAttributes.add(AMd);
                descriptive = new DefaultMutableTreeNode(AMd.Name);
                hierarchy.add(descriptive);
            }
            AM = getHierarchyAttribute(AM);
        }
        this.Attributes = newAttributes;
        tree = new JTree(top);
         tree.getSelectionModel().setSelectionMode(
                 TreeSelectionModel.SINGLE_TREE_SELECTION);
    }
/*****************************************************************/
    /** public JTree getJTree()
     *      return a JTree with the name of the multidimensional
     *      attributes of the dimension
```

```
     *  Output parameters:
     *      JTree : JTree with the names of the attributes
     */
    public JTree getJTree(){
        return this.tree;
    }
/*****************************************************************/
    /** public void createSchema
     *      creates the Star schema for the dimension
     */
     public void createStarSchema() {

        // get the hierarchy attributes in order less important most
         //important
        ListIterator LI = this.Attributes.listIterator();
        LinkedList ListATT = new LinkedList();

        CREATE_TABLE = "CREATE TABLE \""+this.Name+"\" (\n\t\"";
        CREATE_TABLE +=  this.GrainAttribute.Name+"\" "+
                this.GrainAttribute.getReferencedAttribute().getType();
         String PRIMARY_KEY = "\n\tCONSTRAINT PK_"+this.Name + "
                 PRIMARY KEY(\"" + this.GrainAttribute.Name+"\")";
        while (LI.hasNext()){
            AttributeMulti AM = (AttributeMulti)LI.next();
            if (!AM.isGrain()){
                CREATE_TABLE += ", \n\t\"" +
                        AM.Name + "\" "+
                        AM.getReferencedAttribute().getType()+" ";
            }
        }
        CREATE_TABLE += ", "+ PRIMARY_KEY+" );";
    }
/*****************************************************************/
    /** String createTransformationSchema()
     *      creates the transformation schema from the relational
     *      database to the multidimensional one for the dimension
     *  Output parameter: transformation schema for the dimension
     */
    public String createTransformationSchema() {

        // get the hierarchy attributes in order less important most
         //important
        ListIterator LI = this.Attributes.listIterator();
        LinkedList ListATT = new LinkedList();

        // Select first the grain attribute
        String SELECT = "SELECT DISTINCT " +
         this.GrainAttribute.getReferencedAttribute().TableName +".\""
         + this.GrainAttribute.getReferencedAttribute().Name + "\"";
        // select the rest of the attributes
        while (LI.hasNext()){
```

```
            AttributeMulti AM = (AttributeMulti)LI.next();
            if (!AM.isGrain()){
                 SELECT += ", \n\t" +
                 AM.getReferencedAttribute().TableName + ".\""+
                 AM.getReferencedAttribute().Name+"\" ";
            }
        }
        // Make the clause FROM AND WHERE
        String FROM ="\nFROM ";
        String WHERE = "\nWHERE " ;
        LI = this.pathHierachy.getIteratorRelation();
        String coma =" ";
        if ( pathHierachy.sizePath() == 0 ){
             FROM += "\"" +
             this.GrainAttribute.getReferencedAttribute().TableName +
             "\" " +
             this.GrainAttribute.getReferencedAttribute().TableName ;
             coma =" ,";
             WHERE ="";
        }
        else{
            while (LI.hasNext()){
                FKRelation FKR = (FKRelation)LI.next();
                if (!FROM.contains("\"" + FKR.referencedAttr.TableName
                 +"\"")){
                    FROM += coma + "\"" + FKR.referencedAttr.TableName+
                          "\" " + FKR.referencedAttr.TableName ;
                    coma =" ,";
                }
                if (!FROM.contains("\"" + FKR.Attr.TableName +"\"")){
                    FROM += coma + "\"" + FKR.Attr.TableName+ "\" " +
                          FKR.Attr.TableName ;
                    coma =" ,";
                }
                WHERE += FKR.Attr.TableName + ".\"" + FKR.Attr.Name +
                          "\"=" +
                          FKR.referencedAttr.TableName + ".\"" +
                         FKR.referencedAttr.Name + "\"";
                if (LI.hasNext()){
                    WHERE += " AND \n";
                }
            }
        }
        TransformationSchema = SELECT + FROM+ WHERE;
        return TransformationSchema;
    }
/*****************************************************************/
    /** String public String getTransformationSchema()
     *      returns the transformation schema from the relational
     *      database to the multidimensional one for the dimension
     **  Output parameter: transformation schema for the dimension
```

```
     */
    public String getTransformationSchema() {
        return this.TransformationSchema;
    }
}
```

A.2.4 Class Fact_Table

```
/*
 * Fact_Table.java
 * Created on November 23, 2006, 5:55 PM
 */
/*********************************************************************
 * Package
*********************************************************************/
package Multidimensional_Model;
/*********************************************************************
 * Import Class
*********************************************************************/
import Relational_Model.Attribute;
import Relational_Model.FKRelation;
import Relational_Model.Path;
import Relational_Model.Paths;
import multidimensionalbuilder.AggregationType;
import java.util.*;


/**
 *  Class Fact Table : Describe the fact table of a multidimensional
 *  model and the operation inherits.
 *
 * @author Ana M Giral
 */
public class Fact_Table {


/*********************************************************************
     * FIELD DECLARATION
*********************************************************************/
    LinkedList measures;
    Paths ThePaths; // list of path between the grain attributes and
                    //the measure
    String CREATE_TABLE;
    String TransformationSchema;
    /** Creates a new instance of Fact_Table */
    public Fact_Table() {
        measures = new LinkedList();
        this.ThePaths = new Paths();
    }
```

```
/*****************************************************************/
    /** public void addMeasure(Attribute A, AggregationType AT)
     *      Introduce a new measure in the FactTable
     *  Input Parameters :
     *      Attribute A : Attribute from the relational model
     *      AggregationType AT: Type of aggregation of the attrbityte
     *      in the model
     */
    public void addMeasure(Attribute A, String AT, Paths TP){
        this.ThePaths = TP;
        measures.add(new measure(A, AT));
    }
/*****************************************************************/
    /** public void addMeasure(AttributeMulti A, AggregationType AT)
     *      Introduce a new measure in the FactTable
     *  Input Parameters :
     *      AttributeMulti A : Attribute already created from the
     *           multidimensional model
     *      AggregationType AT: Type of aggregation of the attrbityte
     *           in the model
     */
    public void addMeasure(AttributeMulti A, String AT, Paths TP){
        this.ThePaths = TP;
        measures.add(new measure(A, AT));
    }
/*****************************************************************/
    /** public LinkedList getMeasuresName()
     *      return the name of all the measures
     *  Output Parameters :
     *      LinkedList : list with the names of the measure
     */
    public LinkedList getMeasuresName() {
        ListIterator listIt = this.measures.listIterator();
        LinkedList L = new LinkedList();
        while (listIt.hasNext()){
                L.add(((measure)listIt.next()).AM().Name) ;
        }
        return L;
    }
/*****************************************************************/
    /** public LinkedList getMeasures()
     *      return the measures
     *  Output Parameters :
     *      LinkedList : list with the measure
     */
    public LinkedList getMeasures() {
        return this.measures;
    }
/*****************************************************************/
    /** public void createSchema
```

```
     *      creates the Star schema for the factTable
     *  Input parameters :
     *      LinkedList Dimension : List of the dimensions of the MD
     *           model
     */
    void createStarSchema(LinkedList Dimensions) {

        CREATE_TABLE = "CREATE TABLE \"FACT_TABLE\" ( " ;
        String PRIMARY_KEY = "\nCONSTRAINT PK_FACTTABLE PRIMARY KEY(";
        String FOREIGN_KEY ="";
        ListIterator ListIt =Dimensions.listIterator();
        while (ListIt.hasNext()){
            Dimension D = (Dimension)ListIt.next();
            CREATE_TABLE += "\n\t\""+D.GrainAttribute.Name+"\"
         "+D.GrainAttribute.getReferencedAttribute().getType()+", ";
            PRIMARY_KEY += "\""+D.GrainAttribute.Name+"\"";
            FOREIGN_KEY += "\n\tCONSTRAINT FK_FT"+D.Name+ " FOREIGN KEY
                 (\""+ D.GrainAttribute.Name +"\") REFERENCES \""
                 + D.Name + "\" (\"" + D.GrainAttribute.Name +"\")";
            if (ListIt.hasNext()){
                PRIMARY_KEY +=", ";
                FOREIGN_KEY +=", ";
            }
        }
        ListIt = this.measures.listIterator();
        while ( ListIt.hasNext() ){
            AttributeMulti AM = ((measure)ListIt.next()).AM() ;
            CREATE_TABLE += "\""+AM.Name+"\"  "+
                 AM.getReferencedAttribute().getType()+" ";
            if ( ListIt.hasNext() )
                 CREATE_TABLE += ", ";
        }
        CREATE_TABLE += ", "+ PRIMARY_KEY+"),"+FOREIGN_KEY +");";
    }
/*****************************************************************/
    /** public Path getPath()
     *      return the star schema of fact table
     */
    public String getSchema(LinkedList Dimensions) {
        return this.CREATE_TABLE;
    }

/*****************************************************************/
    /** public String createTransformationSchema()
     *      creates the transformation schema from the relational
     *      database to the multidimensional one for the fact table
     *  Ouput parameter:
     *      String transformation schema for the fact table
     */
    public String createTransformationSchema( ){
```

```
String SELECT = "SELECT ";
String WHERE = "\nWHERE ";
String FROM = "\nFROM ";
String GROUPBY = "\nGROUP BY ";
ListIterator listIt = this.ThePaths.getIterator();
Vector setFKRelation = new Vector();
String coma= " ";
String coma1= " ";
int i = 1;

String measureTable = this.getMeasuresTableName();

while ( listIt.hasNext() ) {
    Path P = (Path)listIt.next();
    String Name  = P.getStart().TableName + i +".\"" +
             P.getStart().Name+"\" , ";
    SELECT += Name;
    GROUPBY += Name;
    ListIterator listItP = P.getIteratorRelation();
    while ( listItP.hasNext() ){

       String FKRReferencedTableName;
       String FKRReferencedAlias;
       String FKRTableName;
       String FKRAlias;

        // insert the table in from clause
        FKRelation FKR = (FKRelation)listItP.next();

        if (FKR.referencedAttr.TableName == measureTable ){
            FKRReferencedTableName =
                      FKR.referencedAttr.TableName;
            FKRReferencedAlias =
                      FKR.referencedAttr.TableName;
        }
        else{
            FKRReferencedTableName =
                      FKR.referencedAttr.TableName;
            FKRReferencedAlias =
                      FKR.referencedAttr.TableName + i;
        }

        if (FKR.Attr.TableName == measureTable ){
            FKRTableName = FKR.Attr.TableName;
            FKRAlias = FKR.Attr.TableName;
        }
        else{
            FKRTableName = FKR.Attr.TableName;
            FKRAlias = FKR.Attr.TableName + i;
        }
        if (!FROM.contains( FKRReferencedAlias )){
```

```
                        FROM += coma1 + "\"" + FKRReferencedTableName+
                                  "\" " + FKRReferencedAlias  ;
                        coma1 =" ,";
                    }

                    if (!FROM.contains( FKRAlias )){
                        FROM += coma1 + "\"" + FKRTableName+ "\" " +
                                  FKRAlias  ;
                        coma1 =" ,";
                    }
                    String Relation = coma + FKRAlias + ".\"" +
                          FKR.Attr.Name + "\"=" +
                           FKRReferencedAlias + ".\"" +
                          FKR.referencedAttr.Name+"\"";
                   if (! WHERE.contains(Relation)) {
                           WHERE += Relation;
                    }
                     coma = " AND " ;
                }
                  i++;
            }
            listIt = this.measures.listIterator();
            while (listIt.hasNext()){
                measure M = (measure)listIt.next();
                String Name = M.AM.getReferencedAttribute().TableName +
                          ".\"" + M.AM.getReferencedAttribute().Name +
                          "\" ";
                SELECT += M.Aggregator() + "(" + Name + ")";
                GROUPBY += Name;
                if (listIt.hasNext()){
                          SELECT+=" , ";
                          GROUPBY+=" , ";
                 }
            }
            TransformationSchema  = SELECT + FROM + WHERE + GROUPBY;
            return TransformationSchema ;
        }
/*****************************************************************/
    /** public String getTransformationSchema()
     *      returns the transformation schema from the relational
     *      database to the multidimensional one for the fact table
     **  Ouput parameter:
     *      String : transformation schema for the fact table
     */
    public String getTrasformationSchema() {
        return this.TransformationSchema;
    }
/*****************************************************************/
    /**  private String getMeasuresTableName()
     *      returns the name of the table of the first measure
     **  Ouput parameter:
```

```
     *      String : name of the table
     */
    private String getMeasuresTableName() {
        LinkedList m = this.getMeasures();
        if (m.size() > 0){
            return
((measure)m.getFirst()).AM().getReferencedAttribute().TableName;
        }
        else return "";
    }


/*********************************************************************
     * CLASS MEASURE
*********************************************************************/
    public class measure{

/*********************************************************************
        * FIELD DECLARATION
*********************************************************************/
        private AttributeMulti AM;
        private String AgrType;

        public measure(Attribute A, String AT ){
            AM = new AttributeMulti(A, false);
            AgrType = AT;
        }
        public measure(AttributeMulti A, String AT ){
            AM = A;
            AgrType = AT;
        }
        public AttributeMulti AM() {return AM;}
        public String Aggregator(){return AgrType;}
        public void modifyAgrType(String i){ AgrType = i;}
    }
}
```

A.2.5 Class Multidimensional_Model

```
/*
 * Multidimensional_Model.java
 * Created on November 23, 2006, 5:55 PM
 */
/*********************************************************************
 * Package
*********************************************************************/
package Multidimensional_Model;
/*********************************************************************
 * Import Class
```

```
*********************************************************************/
import databaseConnection.Metadata;
import Relational_Model.*;
import multidimensionalbuilder.AggregationType;
import java.util.*;

/**
 * @author Ana M Giral
 */

public class Multidimensional_Model {


/*******************************************************************
     * FIELD DECLARATION
*******************************************************************/
    LinkedList Dimensions;
    Fact_Table FactTable;
    Metadata MultidimMetadata;


/*******************************************************************
     * PUBLIC METHODS of the class Multidimensional_Schema
*******************************************************************/
    /**  public Multidimensional_Model()
     *     Creates a new instance of Multidimensional_Model
     */
    public Multidimensional_Model() {
        //Fact table
        FactTable = new Fact_Table();
        //Dimensions
        Dimensions = new LinkedList();
    }
  /*****************************************************************/
    /** public Dimension createDimension(Path P, Path PH)
     *      Creates a new Dimension
     *  Input parameters :
     *  Path ptM : Path from the grain attribute to the measure
     *  Path ptH : Hierarchy path
     */
    public Dimension createDimension(Path P, Path PH) {
        Dimension D = new Dimension(P, PH);
        Dimensions.add(D);
        return D;
    }
 /*****************************************************************/
    /**public void InsertAttribute(Dimension D , Attribute A)
     *      Create and insert a new attribute in the model
     *  Input Attributes :
     *      Dimension D -> Dimension where the attribute has to be
     *           introduced
```

```
     *      Attribute A -> Attribute from the relational model
     */
    public AttributeMulti InsertAttribute(Dimension D , Attribute A){
        return D.AddAttribute(A);
    }

/*****************************************************************/
    /** public void InsertMeasure(Attribute A, AggregationType AT)
     *      Insert a new measure in the fact Table of the
     *     multidimensional model
     *  Input Parameters :
     *      Attribute A : Attribute from the relational model
     *      AggregationType AT: Type of aggregation of the attrbityte
     *           in the model
     */
    public void InsertMeasure(Attribute A, String AT){

        ListIterator listIt = this.getDimensions().listIterator();
        Paths thePaths = new Paths();
        while (listIt.hasNext()){
            thePaths.addPath(((Dimension)listIt.next()).getPath());
        }
        FactTable.addMeasure( A , AT , thePaths);
    }
/*****************************************************************/
    /** public Dimension getDimension(String DimName)
     *     return a Dimesion with the name DimName
     * Input Parameters :
     *     String DimSrc : Name of the Dimension to be returned
     */
    public Dimension getDimension(String DimName){

        ListIterator ListIt = this.Dimensions.listIterator();
        while (ListIt.hasNext()){
            Dimension D = (Dimension)ListIt.next();
            if ( D.Name.equals(DimName ))
                return D;
        }
        return null;
    }
/*****************************************************************/
    /** public LinkedList getDimensions()
     *      Return the dimensions of the multidimesional model
     *  Output parameters :
     *      LinkedList : list of object with type Dimension
     */
    public LinkedList getDimensions(){
        return this.Dimensions;
    }
/*****************************************************************/
    /** public Object[] getDimensionsName()
```

```
     *      Return a array with the dimension names of the
     *     multidimesional model
     *  Output parameters :
     *     Object[] : array of object (Strings)
     */
    public Object[] getDimensionsName(){

        Object L[] = new Object[this.Dimensions.size()];
        ListIterator ListIt = this.Dimensions.listIterator();
        int i = 0;

        while (ListIt.hasNext()){
            Dimension D = (Dimension)ListIt.next();
            L[i] = D.Name;
            i++;
        }
        return L;
    }
/*****************************************************************/
    /** public LinkedList getDimensionsNameList()
     *      Return a list with the dimension names
     *  Output parameters :
     *     LinkedList : list of String with the names of the dimensions
     */
    public LinkedList getDimensionsNameList(){

        LinkedList dims = new LinkedList();
        ListIterator ListIt = this.Dimensions.listIterator();
        while (ListIt.hasNext()){
            dims.add(((Dimension)ListIt.next()).Name);
        }
        return dims;
    }
/*****************************************************************/
    /** public Dimension getDimensionbyGrain(Attribute GrainAttr)
     *      Return the dimension which has a concre grain attribute
     *  Input Attributes :
     *      Attribute GrainAttr : Attribute wich is the referenced
     *                            attribute of the grain attribute
     */
    public Dimension getDimensionbyGrain(Attribute GrainAttr){

        ListIterator ListIt = this.Dimensions.listIterator();
        while (ListIt.hasNext()){
            Dimension D = (Dimension)ListIt.next();
            if (D.GrainAttribute.getReferencedAttribute()==GrainAttr ){
                return D;
            }
        }
        return null;
    }
```

```
/*****************************************************************/
    /** public Fact_Table getFactTable()
     *      return the metadata of the multidiemsnional database
     *  Output Parameters :
     *      Metadata: metada of multidimensional datavbase
     */
    public Fact_Table getFactTable() {
        return this.FactTable;
    }
/*****************************************************************/
    /** public LinkedList getMeasures()
     *      return the measures
     *  Output Parameters :
     *      LinkedList : list of type measure
     */
    public LinkedList getMeasures() {
        return this.FactTable.getMeasures();
    }
/*****************************************************************/
    /** public Object[] getDimensionsName()
     *      Return a list with the name of the measure
     *  Output parameters :
     *     LinkedList : list of String with the names of the measures
     */
    public LinkedList getMeasuresName(){
        return this.FactTable.getMeasuresName();
    }
/*****************************************************************/
    /** public void createStarSchema
     *      creates the Star schema for multidimensional model
     */
    public void createStarSchema(){
        // For each dimension create the the schema
        ListIterator listIt = this.Dimensions.listIterator();
        while (listIt.hasNext()){
            Dimension D = (Dimension)listIt.next();
            D.createStarSchema();
        }
        // create the star schema fot the fact Table
        this.FactTable.createStarSchema(this.Dimensions);
    }
/*****************************************************************/
    /** public String getStarSchema()
     *      return a Start schema that has been already created
     *  Output parameters
     *      String : Strig with the schema
     */
    public String getStarSchema() {
        String S = "CREATE DATABASE DW\n";
        // Schemas from Dimensions
        ListIterator listIt = this.Dimensions.listIterator();
```

```
        while (listIt.hasNext()){
            S+= ((Dimension)listIt.next()).getSchema() + "\n\n";
        }
        // Schemas from fact Table
        S +=  this.FactTable.getSchema(this.Dimensions);
        return S;
    }
/*****************************************************************/
    /**  public void createDatabase(Metadata database)
     *       creates the physical Tables of the multidimensional
     *        database
     *  Input parameters:
     *       Metadata database: multidimensional database already
     *                          created
     */
    public void createDatabase(Metadata database) {

        MultidimMetadata = database;
        // Creation of the tables for the DImensions
        ListIterator listIt = this.Dimensions.listIterator();
         while (listIt.hasNext()){
                 database.createTable(
                         ((Dimension)listIt.next()).getSchema());
        }
        // create the fact Table
        database.createTable(
                 this.FactTable.getSchema(this.Dimensions));

    }
/*****************************************************************/
    /**  public String createTransformationSchema()
     *       creates and return the transformation schema for the
     *       multidiemsional model
     */
    public String createTransformationSchema(){

        //For each dimension create transformation schema
        String TS = "";
        ListIterator listIt = this.Dimensions.listIterator();
        while (listIt.hasNext()){
            Dimension D = (Dimension)listIt.next();
            TS += D.createTransformationSchema() +"\n\n";
        }
        // create the transformation schema for fact Table
        TS += this.FactTable.createTransformationSchema();
        return TS;
    }
/*****************************************************************/
    /** public public Metadata getMetadata()
     *       return the metadata of the multidiemsnional database
     *  Output Parameters :
```

```
     *      Metadata: metada of multidimensional datavbase
     */
    public Metadata getMetadata() {
        return this.MultidimMetadata;
    }
}
```

A.3 Package OLAP Model

A.3.1 Class OLAP_Schema

```
/*
 * OLAP_Schema.java
 * Created on February 16, 2007, 4:48 PM
 */
/*****************************************************************
 * Package
*****************************************************************/
package OLAP_Model;
/*****************************************************************
 * Import Class
*****************************************************************/
import Multidimensional_Model.AttributeMulti;
import Multidimensional_Model.Fact_Table.measure;
import java.io.FileOutputStream;
import org.jdom.*;
import Multidimensional_Model.Dimension;
import org.jdom.output.XMLOutputter;
import org.jdom.output.Format;
import java.util.*;

/**
 * @author Ana M Giral
 */
public class OLAP_Schema {


/*****************************************************************
     * FIELD DECLARATION
*****************************************************************/
    private Element Schema;
    private Element Cube;

/*****************************************************************
     * PUBLIC METHODS OF CLASS RELATIONAL_MODEL
*****************************************************************/
    /**  public OLAP_Schema()
     * Creates a new instance of OLAP_Schema
```

```
     */
    public OLAP_Schema() {
        Schema = new Element("Schema");
        Cube = new Element("Cube");
        Schema.addContent(Cube);
    }
 /*****************************************************************/
    /**  public void createCube(String CubeName)
     *      Creates a new hypercube
     *   Input parameters :
     *      String CubeName : Name of the cube ;
     */
    public void createCube(String CubeName){
        Schema.setAttribute("name", CubeName);
        Cube.setAttribute("name", CubeName);
        //Element Table
        Element Table = new Element("Table");
        Table.setAttribute("name", "FACT_TABLE");
        //Add the table to the Hierarchy
        Cube.addContent(Table);
    }
/*****************************************************************/
    /**  public void createDimension(Dimension D)
     *      makes the mapping of a dimension into the OLAP schema
     */
    public void createDimension(Dimension D){
        //Element dimension
        Element DimXML = new Element("Dimension");
        DimXML.setAttribute("name", D.Name);
        DimXML.setAttribute("foreignKey", D.GrainAttribute.Name);
        DimXML.addContent(createHierarchy(D));
        Cube.addContent(DimXML);
    }
/*****************************************************************
     *   public void addMeasure(AttributeMulti AM, String Aggregator)
     *      Mapping a measure attribute into a OLAP measure
     *  Input parameters:
     *      AttributeMulti AM : Attribute to be mapped
     *      String Aggregator : type of aggregation
     */
    public void addMeasure(AttributeMulti AM, String Aggregator){
        //Element Measure
        Element theMeasure = new Element("Measure");
        theMeasure.setAttribute("name", AM.Name);
        theMeasure.setAttribute("column", AM.Name);
        theMeasure.setAttribute("aggregator", Aggregator);
        Cube.addContent(theMeasure);
    }
/*****************************************************************/
    /**   public void createFileSchema(LinkedList Dimensions, String
     *     DBName, String DBURL, String DBPort, String DBUser, String
```

```
     *     DBPassword)
     *     Creates a XML file with the OLAP SCHEma, a query file, and
     *     store its in the disc
     *  Input parameters:
     *      LinkedList Dimensions : List of string with the dimension
     *           names
     *      String DBName: Name of the new multidimenisonal database
     *      String StrDBURL : URL to connect to the database manager
     *      String StrDBPort : Port of the database server
     *      String StrDBUser: Name of the user
     *      String DBPassword : User's password
     */
    public void createFileSchema(LinkedList Dimensions, String DBName,
                String DBURL, String DBPort, String DBUser, String
                DBPassword){

        XMLOutputter outputter = new
                XMLOutputter(Format.getPrettyFormat());
        try{
            outputter.output(new Document(Schema),new
                FileOutputStream("/home/s061032/Desktop/apache-tomcat-
5.5.20/webapps/mondrian/WEB-INF/queries/"+DBName+".xml"));
        } catch (Exception e){
            e.getMessage();
        }
        MDX_Query QueryFile = new MDX_Query();
        QueryFile.createQueryFile(Dimensions, DBName,DBURL, DBName,
                DBPort, DBUser, DBPassword);
    }
/*********************************************************************/
    /**   private Element createHierarchy(Dimension D)
     *      Mapping of a Dimension into OLAP Hierachy
     *  Input parameters:
     *      Dimension D : dimesnion to be mapped
     *  Output parameters :
     *      Element : Element mapped
     */
    private Element createHierarchy(Dimension D){

        //Element Hierarchy
        Element Hierarchy = new Element("Hierarchy");
        Hierarchy.setAttribute("hasAll", "true");
        Hierarchy.setAttribute("primaryKey", D.GrainAttribute.Name);
        Hierarchy.setAttribute("allMemberName", "All "+
                D.GrainAttribute.Name);
        //Element Table
        Element Table = new Element("Table");
        Table.setAttribute("name", D.Name);
        //Add the table to the Hierarchy
        Hierarchy.addContent(Table);
        // Add the diferent levels to the table
```

```
        ListIterator Attributes = D.getAttributes().listIterator();
        Element Level = null;
        while (Attributes.hasNext()){
            AttributeMulti AM = (AttributeMulti) Attributes.next();
            if (AM.isHierarchy()){
                Level = createLevel(D, AM);
                Hierarchy.addContent(1,Level);
            }else {
                Level.addContent(createProperty(AM));
            }
        }
        return Hierarchy;
    }
/*****************************************************************/
    /**   private Element createLevel(Dimension D, AttributeMulti AM)
     *      Mapping a hierarchy attribute into a OLAP level
     *  Input parameters:
     *      Dimension D : dimension to be mapped of the attribute
     *      AttributeMulti AM : Attribute to be mapped
     *  Output parameters :
     *      Element : Element mapped
     */
    private Element createLevel(Dimension D, AttributeMulti AM){

        //Element Level
        Element Level = new Element("Level");
        Level.setAttribute("name", AM.Name);
        Level.setAttribute("column",AM.Name);
        if (AM.equals(D.GrainAttribute)){
            Level.setAttribute("uniqueMembers", "true");
        } else{
            Level.setAttribute("uniqueMembers", "false");         }
        return Level;
    }
/*****************************************************************
     *   private Element createProperty(AttributeMulti AM)
     *      Mapping a descriptive attribute into a OLAP property
     *  Input parameters:
     *      Dimension D : dimension to be mapped of the attribute
     *      AttributeMulti AM : Attribute to be mapped
     *  Output parameters :
     *      Element : Element mapped
     */
    private Element createProperty(AttributeMulti AM){

        //Element Property
        Element Property = new Element("Property");
        Property.setAttribute("name", AM.Name);
        Property.setAttribute("column", AM.Name);
        return Property;
    }
```

```
}
```

A.3.2 Class MDX_Query

```
/*
 * MDX_Query.java
 * Created on February 19, 2007, 8:23 PM
 */
/*********************************************************************
 * Package
*********************************************************************/
package OLAP_Model;
/*********************************************************************
 * Import Class
*********************************************************************/
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;
/**
 * @author Ana M Giral
 */
public class MDX_Query {

/*********************************************************************
     * PUBLIC METHODS OF CLASS RELATIONAL_MODEL
*********************************************************************/
    /** Creates a new instance of MDX_Query */
    public MDX_Query() {}
/*********************************************************************/
    /** public void  createQueryFile(LinkedList Dimensions , String
     *      fileName, String DBURL, String DBName,String DBPort, String
     *      DBUser, String DBPassword)
     *      creates the MDXquery file and store it in the tomcat server
     *      LinkedList Dimensions : List of string with the dimension
     *                    names
     *      String fileName : name of the file to be created
     *      String DBName: Name of the new multidimenisonal database
     *      String StrDBURL : URL to connect to the database manager
     *      String StrDBPort : Port of the database server
     *      String StrDBUser: Name of the user
     *      String DBPassword : User's password
     */
    public void  createQueryFile(LinkedList Dimensions , String
        fileName, String DBURL, String DBName,String DBPort, String
        DBUser, String DBPassword){

        String S = "<%@ page session=\"true\" contentType=\"text/html;
```

```
 charset=ISO-8859-1\" %>";
S += "\n<%@ taglib uri=\"http://www.tonbeller.com/jpivot\"
          prefix=\"jp\" %>";
S += "\n<%@ taglib prefix=\"c\"
          uri=\"http://java.sun.com/jstl/core\" %>";
S += "\n<%-- uses a dataSource --%>";
S += "\n<jp:mondrianQuery id=\"query01\"
          jdbcDriver=\"org.postgresql.Driver\"
 jdbcUrl=\"jdbc:postgresql://localhost:" + DBPort+ "/" + DBName
          + "\" jdbcUser=\"" + DBUse r+ "\" jdbcPassword=\"" +
          DBPassword + "\" catalogUri=\"/WEB-INF/queries/" +
          filename + ".xml\">";
S += "\n\nselect ";
int count = Dimensions.size();
ListIterator ItDimensions = Dimensions.listIterator();
if (count%2 == 0 ){
    S += "\n{(";
    for (int i = 0; i < count/2 ; i ++){
        String DimName = (String) ItDimensions.next();
        S +="["+DimName+"].[All "+DimName+"]";
        if (i < count/2 - 1){
            S+=" , ";
        }
    }
    S += ")} ON COLUMNS ";
    if (ItDimensions.hasNext()){
        S += ",\n{(";
        for (int i = 0; i < count/2 ; i ++){
            String DimName = (String) ItDimensions.next();
            S +="["+DimName+"].[All "+DimName+"]";
            if (i < count/2 - 1){
            S+=" , ";
            }
        }
        S+=")} ON ROWS ";
    }
    S += "\nFROM "+ DBName;
}else{
    S += "\n{(";
    for (int i = 0; i < count/2+1 ; i ++){
        String DimName = (String) ItDimensions.next();
        S +="["+DimName+"].[All "+DimName+"]";
        if (i < count/2 ){
            S+=" , ";
        }
    }
    S += ")} ON COLUMNS ";

    if (ItDimensions.hasNext()){
        S += ",\n{(";
        for (int i = 0; i < count/2 ; i ++){
```

```
                    String DimName = (String) ItDimensions.next();
                    S +="["+DimName+"].[All "+DimName+"]";
                    if (i < count/2 - 1){
                         S+=" , ";
                }
               }
               S+=")} ON ROWS ";
            }
            S += "\nFROM "+ DBName;
        }
        S +="\n</jp:mondrianQuery>";
        S +="\n<c:set var=\"title01\" scope=\"session\">Test Query uses
                Mondrian OLAP</c:set>";

        String filePath = "/home/s061032/Desktop/apache-tomcat-
                5.5.20/webapps/mondrian/WEB-INF/queries/" + filename +
                ".jsp";
        File file = new File(filePath);

        if (file.exists()){
            file.delete();
        }

        try{
            BufferedWriter bw = new BufferedWriter(new
                                        FileWriter(filePath));
            bw.write(S);
            bw.close();
        } catch (IOException ioe){
            System.err.println("Err writing file" + filePath);
            ioe.printStackTrace();
        }
    }
}
```

A.4 Package Multidimensional_Construction

A.4.1 Class Relational_Schemas

```
/*
 * Relational_Schemas.java
 * Created on November 16, 2006, 12:45 PM
 */
/**************************************************************************
 * Package
**************************************************************************/
```

```
package Multidimensional_Construction;

/*********************************************************************
 * Import class
*********************************************************************/
import Relational_Model.*;
import databaseConnection.Metadata;
import multidimensionalbuilder.userInteraction;
import java.util.*;
import javax.swing.*;

/**
 * @author Ana M Giral
 */
public class Relational_Schemas {

/*****************************************************************
     * FIELD DECLARATION
*****************************************************************/
    private Relational_Model RelationalModel;
    private Metadata MetadataDB;

/*****************************************************************
     * PUBLIC METHODS OF CLASS RELATIONAL_MODEL
*****************************************************************/
    /** public Relational_Schemas(String theIp, String thePort, String
     *   theNamespace, String theUser, String thePass)
     * Creates a new instance of Relational_Schemas
     *      String theIp : IP or URL to connect to the database manager
     *      String StrDBPort : Port of the database server
     *      String theNamespace : Namessapce of the relational database
     *      String StrDBUser: Name of the user
     *      String DBPassword : User's password
     */
    public Relational_Schemas(String theIp, String thePort, String
                theNamespace, String theUser, String thePass) {

        MetadataDB = new Metadata(theIp, thePort, theNamespace,
                        theUser, thePass);
        RelationalModel = new Relational_Model();
    }
/*****************************************************************/
    /** public void ExtractRelationalModel()
     *      Extraction of the relational model from a physical database
     */
    public void ExtractRelationalModel() {

        // Instancied variables
        String TheTable;
        LinkedList TheAttr;
        ListIterator ListItAtt;
```

```
        // Tables Extraction
        LinkedList TheTables = MetadataDB.Tables();
        ListIterator ListIt = TheTables.listIterator();
        //For each table insert the table in the model
        while (ListIt.hasNext()) {

            TheTable = (String)ListIt.next();
            RelationalModel.InsertTable( TheTable );
            TheAttr = MetadataDB.Attributtes(TheTable);
            ListItAtt = TheAttr.listIterator();
            while (ListItAtt.hasNext()) {
                 // insert the attributes of the table in the model
                String NameAttribute = (String)ListItAtt.next();
                String Restr =
                          MetadataDB.AttributeRestriction(TheTable,
                          NameAttribute);
                RelationalModel.InsertAttTable(TheTable, NameAttribute,
                          Restr);
            }
        }
        // Foreign key Extraction
        LinkedList TheFKRelations = MetadataDB.ForeignKeyRelation();
        ListIt = TheFKRelations.listIterator();
        // For each Foreign key relation, insert it in the model
        String FKRel[] = new String[4];
        while (ListIt.hasNext()) {
            FKRel =(String[])ListIt.next();
            RelationalModel.InsertFKRelation( FKRel[0], FKRel[1],
                 FKRel[2], FKRel[3] );
        }
    }
/*****************************************************************/
    /** public LinkedList selectTables(LinkedList L)
     *      Select a set of tables from the model to know the list of
     *       attributes
     *  Input Parameters :
     *      LinkedList L -> List of String with the Names of the tables
     *  Output Parameters :
     *      LinkedList -> List of Attributes of each table
     */
    public LinkedList selectTables(LinkedList L){

        // List for the information
        LinkedList LI = new LinkedList();
        ListIterator ListIt = L.listIterator();
        Attribute A;
        //Add to the list all the attributes of each table
        while (ListIt.hasNext()){
            String S = (String)ListIt.next();
            LI.addAll(RelationalModel.SelectTable(S));
        }
```

```
      return LI;
    }
/*****************************************************************/
    /** public LinkedList selectTables(String TableName)
     *      Select a table from the model to know the list of
     *       attributes
     *  Input Parameters :
     *     String TableName : Name of the table
     *  Output Parameters :
     *      LinkedList -> List of Attributes of a table
     */
    public LinkedList selectTables(String TableName){
         return RelationalModel.SelectTable(TableName);
    }
/*****************************************************************/
    /** public LinkedList showTables()
     *      Return all the tables of the model
     *  Output Parameters :
     *      LinkedList -> List of Strings with the Name of all the
     *           tables
     */
    public LinkedList showTables(){
        return RelationalModel.getTables();
    }
/*****************************************************************/
    /** public void selectAttributes(LinkedList Attributes)
     *      Select the attributes necessaries to build the model
     *  Input parameters :
     *      LinkedList Attributes -> List of Attributes that user want
     * to select
     */
    public void selectAttributes(LinkedList Attributes){
        Attribute A;
        ListIterator ListIt = Attributes.listIterator();
        while (ListIt.hasNext()){
            A = (Attribute)ListIt.next();
            // Extract the information of the data TYPE
            A.setType(MetadataDB.AttributteType(A.TableName, A.Name));
            // Select the atribute
            RelationalModel.SelectAttribute(A);
        }
    }
/*****************************************************************/
    /** public void selectGrainAttributes(LinkedList Attributes)
     *      Select the grain attributes
     *  Input parameters :
     *      LinkedList Attributes -> List of Grain Attributes that user
     *       want to select
     */
    public void selectGrainAttributes(LinkedList Attributes){
        ListIterator ListIt = Attributes.listIterator();
```

```
        while (ListIt.hasNext()){
            // Select the atribute
          RelationalModel.SelectGrainAttribute(
                (Attribute)ListIt.next());
        }
    }
/*****************************************************************/
    /** public void SelectAttribute(Attribute A)
     * Select an attribute to be in the multidimenional modell
     *  Input parameters :
     *      Attribute A: Attribute to be selected
     */
    void selectAttributes(Attribute A) {
        this.RelationalModel.SelectAttribute(A);
        A.setType(MetadataDB.AttributteType(A.TableName, A.Name));
    }
/*****************************************************************/
    /** public void SelectMeasure(String TableName, String AttrName)
     *      select an attribute as a measure attribute
     *  Input parameters :
     *      String TableName: Name of the Table that contains the
     *           attribute
     *      String AttrName : Name of the attribute
     */
    public void SelectMeasure(String TableName, String AttrName){

        this.RelationalModel.SelectMeasure(TableName, AttrName);
        this.RelationalModel.getMeasure().setType(
                MetadataDB.AttributteType(TableName, AttrName));
    }
/*****************************************************************/
    /**  public LinkedList getSelectedAttributes()
     *      Return a list with all the selected Attributes
     *  Output Parameters :
     *      LinkedList : List of object with type Attribute
     */
    public LinkedList getSelectedAttributes() {
        return this.RelationalModel.getSelectedAttributes();
    }
/*****************************************************************/
    /** public Attribute getPKAttribute(String TableName)
     *      Return the first attribute with a primary key retriction of
     *       a table
     * Input parameter:
     *      String Name : Name of the table
     *  Output Parameter:
     *      Attribute : Attribute that is primary key
     */
    public Attribute getPKAttribute(String TableName) {
       return this.RelationalModel.getPKAttribute(TableName);
    }
```

```
/*****************************************************************/
    /** public Attribute getUNIQUEAttribute(String TableName)
     *      Return the first attribute with a UNIQUE retriction of a
     *     table
     * Input parameter:
     *      String Name : Name of the table
     *  Output Parameter:
     *      Attribute : Attribute that has a UNIQUE restriction
     */
    Attribute getUNIQUEAttribute(String TableName) {
        return this.RelationalModel.getUNIQUEAttribute(TableName);
    }
/*****************************************************************/
    /** boolean isSelected(Attribute PKA)
     *      Return true if an attribute of the realtional model is
     *      selected
     * Input parameter:
     *     Attribute PKA : attribute of the relational model
     *  Output Parameter:
     *      boolean : tru if the input attribute is selected
     */
    boolean isSelected(Attribute PKA) {
        return this.getSelectedAttributes().contains(PKA);
    }
/*****************************************************************/
    /**  public Attribute getMeasure()
     *      return the measure attribute
     *  Output parameters :
     *      Attribute : measure attribute
     */
    public Attribute getMeasure() {
        return this.RelationalModel.getMeasure();
    }
    /***************************************/
    /** public Relational_Model getRelationalModel()
     *      Return the relational model
     */
    public Relational_Model getRelationalModel(){
        return this.RelationalModel;
    }
    /***************************************/
    /** public Paths selectGrainTable(String TableName, JPanel
     *   JpaneltoWrite)
     *      return a path list between the grain attributes selected
     *      from a table and the measure
     *  Input parameters :
     *      String TableName : Name of the table to choose the grain
     *                         attributes
     *      JPanel JpaneltoWrite : panel to interatuate with the user
     */
    public Paths selectGrainTable(String TableName, JPanel
```

```
        JpaneltoWrite){

        Paths FinalPaths = new Paths();
        // Table that contains the possible grain attribute
        Table T = this.RelationalModel.searchTable(TableName);
        // Instance the Jpanel for the user interaction
        userInteraction UID = new userInteraction(JpaneltoWrite);
        // Choose grain attribute or grain Attributes from the table
        LinkedList  Attributes =
                 UID.userGrainAttributeSelection(T.getAttributes());
        ListIterator ListIt = Attributes.listIterator();
        while (ListIt.hasNext()){
            Attribute A = (Attribute)ListIt.next();
            //make the path between the attribute and the measure
            FinalPaths.addPaths(ChoosePathsFromMeasure(A,
                 JpaneltoWrite));
        }
        return FinalPaths;
    }
    /*****************************************/
    /** public Paths ChoosePathsFromMeasure(Attribute A, JPanel
                 JpaneltoWrite)
     *       return a set of Paths from a grain Attribute to a measure
     *      attribute
     *       that the user has select from the diferent posibilities.
     *  Input parameters :
     *      Attribute A : Grain attribute
     *      JPanel JpaneltoWrite : panel to interatuate with the user
     *  Output parameters :
     *      Paths : list of path
     */
    public Paths ChoosePathsFromMeasure(Attribute A, JPanel
         JpaneltoWrite){

         // Instance the Jpanel for the user interaction
        userInteraction UID = new userInteraction(JpaneltoWrite);
        // instance the returned value
        Paths FinalPaths = new Paths();
        Paths ThePaths = this.RelationalModel.makePathsFromMeasure(A);
        //user select the pahth
        if ( ThePaths.size() > 1 ){
            FinalPaths.addPaths(UID.UserPathSelection(ThePaths));
        }
        else{
            if ( ThePaths.size() == 1 )
                 FinalPaths.addPath(ThePaths.getPath(0));
            else
                System.out.println("There are not path between the
                                    grain attribute and the measure");
        }
        return FinalPaths;
```

```
}
/*****************************************/
/** public Paths lookForHierarchies(Attribute A, JPanel *
 *  JpaneltoWrite)
 *       return a set of Hierarchy Paths from a grain Attribute
 *       that the user has select from the  diferent posibilities.
 *  Input parameters :
 *      Attribute A : Grain attribute
 *      JPanel JpaneltoWrite : panel to interatuate with the user
 *  Output parameters :
 *      Paths : list of path of hierarchies
 */
public Paths lookForHierarchies(Attribute A, JPanel JpaneltoWrite){

    //instance of the returned value
    Paths FinalPaths = new Paths();
    // Instance the JPanel for the user interaction
    userInteraction UID = new userInteraction(JpaneltoWrite);
    // if the attribute is a Pimary key or foreign key then is
    // posible to find a hierarchy
    if (A.isPK() || A.isUNIQUE() || A.isFK() ){
        // choose the hierarchy
        Paths Hierarchies =
                    this.RelationalModel.makeHierarchies(A);
       // the user has to select the hierarchy
        if ( Hierarchies.size() > 0 )
             FinalPaths.addPaths(
                    UID.chooseHierarchies(Hierarchies));
        else
            System.out.println("There are no hierarchy attributes
                               for this attribute");
    }
    else {
        Path P = new Path(A);
        P.addEnd(A);
        FinalPaths.addPath(P);
    }

    return FinalPaths;
}
/*****************************************/
/** LinkedList selectHierachyAttributes(Path P, JPanel
 *  JpaneltoWrite)
 *       return a set of selected attributes by user from a
 *      hierarchy
 *  Input parameters :
 *      Path P : hierarchy path
 *      JPanel JpaneltoWrite : panel to interatuate with the user
 *  Output parameters :
 *      LinkedList : list of type Attribute
 */
```

```
    public LinkedList selectHierachyAttributes(Path P, JPanel
         JpaneltoWrite){

        LinkedList PathTables = new LinkedList();
        ListIterator ItRelationPath = P.getIteratorRelation();

        if (P.sizePath() == 0){
            PathTables.add(P.getStart().TableName);
        }else{
            while ( ItRelationPath.hasNext() ){
                FKRelation FKR = (FKRelation)ItRelationPath.next();
                if ( ! PathTables.contains(FKR.Attr.TableName))
                    PathTables.add(FKR.Attr.TableName);
                if ( ! PathTables.contains(
                          FKR.referencedAttr.TableName))
                    PathTables.add(FKR.referencedAttr.TableName);
            }
        }
        // Select al the tables of the path
        LinkedList TheAttributes = this.selectTables(PathTables);
        //user selection
        userInteraction UID = new userInteraction(JpaneltoWrite);
        LinkedList LI = UID.selectHierarchyAttributes(TheAttributes);
        this.selectAttributes(LI);
        return  LI;
    }
/*****************************************************************/
    /** public public Metadata getMetadata()
     *      return the metadata of the relational database
     *  Output Parameters :
     *      Metadata: meta-data of the relational database
     */
    public Metadata getMetadata(){
        return this.MetadataDB;
    }
}
```

A.4.2 Class Multidiemsnional_Schemas

```
/*
 * Multidimensional_Schemas.java
 * Created on November 23, 2006, 5:50 PM
 */
/*********************************************************************
 * Package
*********************************************************************/
package Multidimensional_Construction;
/*********************************************************************
```

```
 * Import Class
**********************************************************************/
import Multidimensional_Construction.*;
import OLAP_Model.OLAP_Schema;
import Multidimensional_Model.*;
import Multidimensional_Model.Dimension;
import Multidimensional_Model.Fact_Table.*;
import Relational_Model.*;
import multidimensionalbuilder.*;
import databaseConnection.Metadata;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.ResultSet;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/**
 * @author Ana M Giral
 */

public class Multidimensional_Schemas {

/*****************************************************************
     * Instance variables
  *****************************************************************/
        // Interface with the multidimensional model
    private Multidimensional_Model Multidimensional;
        // Interface with the relational model
    private Relational_Schemas Relational;
/*****************************************************************
     * PUBLIC METHODS of the class Multidimensional_Schema
*****************************************************************/
    /** public Multidimensional_Schemas(Relational_Schemas RM)
     *      Constructor Multidimensional_Schemas
     *  Input Parameter:
     *      Relational_Schemas RM : relational schema to extract the
     *      information to make the multidimensional model
     */
    public Multidimensional_Schemas(Relational_Schemas RM) {
        Relational = RM;
        Multidimensional = new Multidimensional_Model();
    }

/*****************************************************************/
    /** public void making_dimensions(Paths ThePaths, JPanel
     *   JPanelUserInteraction)
```

```
     *      Make a multidimensional model from a Relatinal model
     *      If there are some ambiguous situation user is asked
     *  Input Parameter:
     *      Paths ThePaths : all the paths selected by the user between
     *                       a grain attribute a dimension
     *      JpanelUserInteraction : JPanel in order to comunicate with
     *                              the user
     */
    public void make_dimensions(Paths ThePaths, JPanel
                JPanelUserInteraction){
        while (ThePaths.hasPath())
            make_dimension(ThePaths.nextPath(), JPanelUserInteraction);
    }
/*****************************************************************/
    /** public void make_FactTable(Attribute attribute, String
     *   Aggregator)
     *      make the fact table with a measure attribute
     *  Input Parameter:
     *      Attribute attribute: measure relational attribute
     *      String Aggregator: type of aggregation for the measure
     *           attribute
     */
    public void make_FactTable(Attribute attribute, String Aggregator){
        this.Multidimensional.InsertMeasure(attribute, Aggregator);
    }
/*****************************************************************/
    /** public LinkedList dimension()
     *      Return a list with the dimensions of the model
     *  Output Parameter:
     *      LinkedList : LikedList with objects with type dimension
     */
    public LinkedList dimensions(){
        return this.Multidimensional.getDimensions();
    }
/*****************************************************************/
    /** public LinkedList getDimensionsName()
     *      Return a list with the names of dimensions of the model
     *  Output Parameter:
     *      LinkedList : LikedList with objects with type String
     */
    public Object[] getDimensionsName(){
        return this.Multidimensional.getDimensionsName();
    }
/*****************************************************************/
    /**  public LinkedList Measures()
     *      Return a list with the names of measure
     *  Output Parameter:
     *      LinkedList : LikedList with objects with type String
     */
    public LinkedList Measures(){
        return this.Multidimensional.getMeasuresName();
```

```
    }
/*****************************************************************/
    /** public void createStarSchema
     *      creates the Star schema for multidimensional model and
     *      store it in a file with the name "CREATE_DATABASE.sql"
     */
    public void createStarSchema( ){

        this.Multidimensional.createStarSchema();
        String fileName = "CREATE_DATABASE.sql";
        File file = new File(fileName);

        if (file.exists()){
            file.delete();
        }
        try{
            BufferedWriter bw = new BufferedWriter(new
                                        FileWriter(fileName));
            bw.write(this.Multidimensional.getStarSchema());
            bw.close();
        } catch (IOException ioe){
            System.err.println("Err writing file");
            ioe.printStackTrace();
        }
    }
/*****************************************************************/
    /** public void createMultiDimensionalDatabase( String StrMultiDB ,
     * String StrDBURL, String StrDBPort, String StrDBUser, String
     * StrDBPwd )
     *      creates the physical database in the server specify
     *      String StrMultiDB:Name of the new multidimenisonal database
     *      String StrDBURL : URL to connect to the database
     *      String StrDBPort : Port of the server
     *      String StrDBUser: Name of the user
     *      String StrDBPwd : User's password
     */
    public void createMultiDimensionalDatabase( String StrMultiDB ,
             String StrDBURL, String StrDBPort, String StrDBUser,
             String StrDBPwd ) throws IOException{

        this.Relational.getMetadata().createDatabase(StrMultiDB);
        this.Multidimensional.createDatabase(new Metadata(StrDBURL,
                 StrDBPort, StrMultiDB , StrDBUser, StrDBPwd, true));
    }
/*****************************************************************/
    /** public void  createTransformationSchema()
     *      creates the transformation schema between the the
     *      realtional database and the multidimensional one and save
     *      it in the file "TRANSFORMATION_SCHEMA"
     */
    public void  createTransformationSchema(){
```

```
        this.Multidimensional.createStarSchema();
        String fileName = "TRANSFORMATION_SCHEMA";
        File file = new File(fileName);
        if (file.exists()){
            file.delete();
        }
        try{
            BufferedWriter bw = new BufferedWriter(new
                                            FileWriter(fileName));
            bw.write(
                 this.Multidimensional.createTransformationSchema());
            bw.close();
        } catch (IOException ioe){
            System.err.println("Err writing file");
            ioe.printStackTrace();
        }
    }
/*****************************************************************/
    /** public void  loadMultidimensionalDatabase()
     *      make the first data load of the multidimensional database
     *      from the relational one
     */
    public void loadMultidimensionalDatabase(){

        ListIterator Dims =
                Multidimensional.getDimensions().listIterator();
        // load data into de dimensions
        while (Dims.hasNext()){
            Dimension D = (Dimension)Dims.next();
            String S = D.getTransformationSchema();
            // make the query to the relational database
            ResultSet R = Relational.getMetadata().getResultQuery(S);
            // insert the Result of the query in the multidimensional
            //database
            Multidimensional.getMetadata().makeUpdate(D.Name, R);
        }
        // Load data into the fact Table
        String S =
         Multidimensional.getFactTable().getTrasformationSchema();
        ResultSet R = Relational.getMetadata().getResultQuery(S);
        Multidimensional.getMetadata().makeUpdate("FACT_TABLE", R);
    }
/*****************************************************************/
    /** public void createOLAPSchema(String DBName, String DBURL,
     *                  String DBPort, String DBPassword, String DBUser)
     *      creates the OLAP schema
     *      String DBName: Name of the new multidimenisonal database
     *      String StrDBURL : URL to connect to the database manager
     *      String StrDBPort : Port of the database server
     *      String StrDBUser: Name of the user
```

```
     *      String DBPassword : User's password
     */
    public void createOLAPSchema(String DBName, String DBURL, String
        DBPort, String DBPassword, String DBUser){

        OLAP_Schema theSchema = new OLAP_Schema();
        ListIterator Dimensions =
                this.Multidimensional.getDimensions().listIterator();
        theSchema.createCube(DBName);
        // create the dimensions
        while (Dimensions.hasNext()){
            theSchema.createDimension((Dimension)Dimensions.next());
        }
        // create tthe measures of the fact table
        ListIterator measures =
                this.Multidimensional.getMeasures().listIterator();
        while (measures.hasNext()){
            measure Measure =  ((measure)measures.next());
            theSchema.addMeasure(Measure.AM(), Measure.Aggregator());
            theSchema.createFileSchema(
                this.Multidimensional.getDimensionsNameList(), DBName
                , DBURL, DBPort, DBUser, DBPassword);
        }
    }
/*********************************************************************
     * PRIVATE METHODS
*********************************************************************/

/*********************************************************************/
    /** private void make_dimension(Path thePath, JPanel
     *  JPanelUserInteraction)
     *      Make a dimension from a hierarchy path
     *  Input Parameter:
     *      Path ThePath : Hierachy paths to create a ned dimension
     *      JpanelUserInteraction : JPanel in order to comunicate with
     * the user
     */
    private void make_dimension(Path thePath, JPanel
        JPanelUserInteraction){

        Attribute A = thePath.getStart();
        Paths theHierarchies = Relational.lookForHierarchies(A,
                                            JPanelUserInteraction);
        ListIterator listItH = theHierarchies.getIterator();

        while (listItH.hasNext()){
            //creation of a single Dimension without attributes
            Path PathHierarchy = (Path)listItH.next();
            Dimension D = this.Multidimensional.createDimension(
                                            thePath, PathHierarchy);
            //add the hierarchy to the dimension
```

```
            if (PathHierarchy.sizePath() > 0 ||
                  D.getGrainAttribute().getReferencedAttribute().isPK()
                 ||
          D.getGrainAttribute().getReferencedAttribute().isUNIQUE() ) {
                 //Selection of the attributes from the relational model
                 LinkedList SelectedAttr =
                  this.Relational.selectHierachyAttributes(
                           PathHierarchy, JPanelUserInteraction);
                 ListIterator ListIt = SelectedAttr.listIterator();
                 // create the multidimensional attributes with the
                  //attributes selected
                 while (ListIt.hasNext()){
                     Attribute AS = (Attribute)ListIt.next();
                     AttributeMulti AM;
                     if (AS ==
                           D.GrainAttribute.getReferencedAttribute())
                         AM = D.getGrainAttribute();
                     else
                         AM = Multidimensional.InsertAttribute(D, AS);
                     this.addSemanticInfo(AM, D, PathHierarchy );
                 }
             } else{
                 //Only one attribute in the DImension
                 this.Relational.selectAttributes(
                           D.GrainAttribute.getReferencedAttribute());
                 D.getGrainAttribute().addSemanticInfo(
                  D.getGrainAttribute(), (new Type()).Hierarchy());
             }
             // Orther the attributes into a JTree
             D.orderAttributes();
         }
     }
/*********************************************************************/
     /** private void addSemanticInfo(AttributeMulti AS, Dimension D,
Path PathHierarchy)
      *      Add the semantic information to an attribute
      *  Input Parameter:
      *     AttributeMulti AS : Attribute to add the dimension
      *     Dimension D: Dimension of the attribute
      *     Path PathHierarchy : Hierarchy path that belong the
attribute
      */
     private void addSemanticInfo(AttributeMulti AS, Dimension D, Path
          PathHierarchy) {
         // instancied variables
         Type semantic_type = new Type();
         boolean end = false;
         boolean meYetInPath = false;

         if (AS.getReferencedAttribute().TableName.equals(
                  PathHierarchy.getStart().TableName)) {
```

```
        // Path is is the same table that the selected Attribute

            if (AS.Name.equals(D.getGrainAttribute().Name)){
                AS.addSemanticInfo(AS, semantic_type.Hierarchy);
            }else{
                if ( AS.getReferencedAttribute().isFK() ){
                 //Atribute is involve in a FK Relation
                    AS.addSemanticInfo(D.GrainAttribute,
                                       semantic_type.Hierarchy);
                } else{    // The attribute is not involve in a FK
                          //relation, then is a Descriptive Attribute
                    AS.addSemanticInfo(D.GrainAttribute,
                                       semantic_type.Descriptive);
                }
            }
        } else{  // The attribute is inside the path
            ListIterator ListIt = PathHierarchy.getIteratorRelation();
            while (ListIt.hasNext() && ! end){
                // browse follow the relatins
                FKRelation FKR = (FKRelation)ListIt.next();
                if (AS.getReferencedAttribute().TableName.equals(
                          FKR.referencedAttr.TableName)) {
                    meYetInPath = true;
                }
                if (meYetInPath){
                    if (AS.getReferencedAttribute().isPK()){
                        if (
Relational.getSelectedAttributes().contains( FKR.Attr.Name)){
                            if (!(FKR.referencedAttr ==
                                  AS.getReferencedAttribute())){
                                AttributeMulti AM = D.getAttribute(
                                                          FKR.Attr );
                                AS.addSemanticInfo(AM,
                                          semantic_type.Hierarchy() );
                                end = true;
                            }
                        }
                        if (!(end == true)){
                            if (
                 Relational.getSelectedAttributes().contains(
                 this.Relational.getPKAttribute(FKR.Attr.TableName))){

                                AttributeMulti AM = D.getAttribute(
                                      this.Relational.getPKAttribute(
                                          FKR.Attr.TableName));
                                AS.addSemanticInfo( AM,
                                          semantic_type.Hierarchy());
                                end = true;
                            }
                        }
                    }else{
```

```
                        if (
Relational.getSelectedAttributes().contains( FKR.referencedAttr)){
                         //Is Selected the for
                            AttributeMulti AM = D.getAttribute(
                                        FKR.referencedAttr );
                            if ( AS.getReferencedAttribute().isNONE()){
                                AS.addSemanticInfo( AM,
                                  semantic_type.Descriptive());
                            }else{
                                if(
                            AS.getReferencedAttribute().isUNIQUE() &&
                            AM.getReferencedAttribute().TableName.equals(
                            FKR.referencedAttr.TableName)){
                                      AS.addSemanticInfo( AM,
                                            semantic_type.Descriptive());
                                } else{
                                      AS.addSemanticInfo( AM,
                                             semantic_type.Hierarchy());
                                }
                            }
                            end = true;
                        } else{ // the Primary key of the table not
selected ->the loof for the FK of the other table
                            // The primary key is not selected
                            // UNIQUE
                            Attribute UNIQUEAttr =
                  Relational.getUNIQUEAttribute(FKR.Attr.TableName);
                            if (UNIQUEAttr != null){    // there are a
attribute with the restriction UNIQUE in the table
                                if(Relational.isSelected(UNIQUEAttr)) {
                                    AttributeMulti AM = D.getAttribute(
                                          UNIQUEAttr ) ;
                                    if (
                            AS.getReferencedAttribute().isNONE()){
                                        AS.addSemanticInfo( AM,
                                   semantic_type.Descriptive());
                                    } else{
                                        AS.addSemanticInfo( AM,
                                          semantic_type.Hierarchy());
                                    }
                                    end = true;
                                }
                            }else{
                            // Is selected the foreign key the
                           //primary key of the table is selected
                                if
                            (Relational.getSelectedAttributes().contains(
                            FKR.referencedAttr)){
                                      AttributeMulti AM = D.getAttribute(
                                            FKR.referencedAttr ) ;
                                      if (
```

```
                        AS.getReferencedAttribute().isNONE()){
                                AS.addSemanticInfo( AM,
                                  semantic_type.Descriptive());
                            } else{
                                AS.addSemanticInfo( AM,
                                    semantic_type.Hierarchy());
                            }
                            end = true;
                        }
                    }
                }
            }
        }
    }
/*****************************************************************/
    /**  private LinkedList attributesTable(Attribute A, LinkedList
     *   ListAttr)
     *     Return all the attributes of the list ListAttr that are in
     *     the same table than the Attribute A
     *     Pre condition : A in not in ListAttr
     * Input Parameters :
     *     Attribute A : Attribute
     *     LinkedList ListAttr :List of attributes
     * Output Parameters:
     *     LinkedList : List with object of type attributes
     */
    private LinkedList attributesTable(Attribute A, LinkedList
         ListAttr){

        LinkedList L = new LinkedList();
        ListIterator ListIt = ListAttr.listIterator();
        // for all the attributes in the input list
        while (ListIt.hasNext()){
            Attribute At = (Attribute)ListIt.next();
            // if the attribute At is in the same list -> add At to the
            // output list
            if (At.TableName.equals( A.TableName ))
                L.add(At);
        }
        return L;
    }
}
```

A.5 Package databaseConnection

A.5.1 Class Metadata

```
/*
 * Metadata.java
 * Created on November 16, 2006, 6:16 PM
 */
/*****************************************************************
 * Package
*****************************************************************/
package databaseConnection;
/*****************************************************************
 * Import Class
*****************************************************************/
import java.sql.*;
import java.util.*;
/**
 * @author Ana M Giral
 */
public class Metadata {

/*****************************************************************
     * FIELD DECLARATION
  ****************************************************************/
    private Postgresql database;

/*****************************************************************
     * PUBLIC METHODS OF CLASS RELATIONAL_MODEL
*****************************************************************/

/*****************************************************************
     * public Metadata(String theIp, String thePort, String
     * theNamespace, String theUser, String thePass)
     *      Creates a new instance of Metadata
     * Input parameters:
     *      String theIp : IP or URL to connect to the database manager
     *      String theDBPort : Port of the database server
     *      String theNamespace : Namessapce of the relational database
     *      String theUser: Name of the user
     *      String thePass : User's password

*****************************************************************/
    public Metadata(String theIp, String thePort, String theNamespace,
        String theUser, String thePass) {

        database = new Postgresql( theIp, thePort, theNamespace,
                                   theUser, thePass);
    }
/*****************************************************************/
    /** public Metadata(String theIp, String thePort, String
     *   theNamespace, String theUser, String thePass)
     *      Creates a new instance of Metadata
```

```
     * Input parameters:
     *      String theIp : IP or URL to connect to the database manager
     *      String StrDBPort : Port of the database server
     *      String theNamespace : Namessapce of the relational database
     *      String StrDBUser: Name of the user
     *      String DBPass : User's password
*****************************************************************/
    public Metadata(String theIp, String thePort, String theNamespace,
                String theUser, String thePass, boolean bol) {
        if (bol == true){
            database = new Postgresql( theIp, thePort, theNamespace,
                theUser, thePass, bol);
        }
         else{
        database = new Postgresql( theIp, thePort, theNamespace,
                theUser, thePass);
        }
    }

/*****************************************************************/
    /** public LinkedList Tables()
     *      Return a list with all the tables of the database
     *  Output parameters :
     *      LinkedList : list with strings
     */
    public LinkedList Tables(){

        LinkedList T = new LinkedList();
        // Database Connection
        database.connect();
        // Make the query
        String StringQuery = "select table_name from
         information_schema.tables where table_schema = 'public'";
        ResultSet R = database.query(StringQuery);


        try {
            while (R.next()) {
                T.add(R.getString(1));
            }
        }catch ( java.sql.SQLException e ) {
                System.err.println("Postgres Connection: ResultSet
                 Error.");
                e.printStackTrace();
        }
        // Database desconection
        database.disconnect();
        return T;
      }

/*****************************************************************/
```

```
    /** public LinkedList Attributtes(String TableName)
     *      Return a list with all the attributes of a table of the
     *      model
     *  Output parameters :
     *      LinkedList : list with strings
     */
    public LinkedList Attributtes(String TableName){

        LinkedList L = new LinkedList();
        // Database Connection
        database.connect();
        // Make the query
        String StringQuery = "select column_name from
                 information_schema.columns where table_schema =
                 'public' AND table_name = '" + TableName +"'";
        ResultSet R = database.query(StringQuery);
        try {
            while (R.next()) {
                L.add(R.getString(1));
            }
        }catch ( java.sql.SQLException e ) {
                System.err.println("Postgres Connection: ResultSet
                 Error.");
                e.printStackTrace();
        }
        // Database desconection
        database.disconnect();
        return L;
      }
/*****************************************************************/
    /** public String AttributteType(String tableName, String AttrName)
     *      Return the type of an attribute
     *  Input parameters :
     *      String tableName : Name of the table that contains the
     *           attribute
     *      String AttrName : Name of the attribute
     *  Output parameters :
     *      String : String with the type of the attribute
     */
    public String AttributteType(String tableName, String AttrName){

       // Database Connection
        database.connect();
        // Make the query
        String StringQuery =" select data_type from
                             information_schema.columns " +
                            " where table_schema = 'public' AND
                            table_name = '" + tableName + "' AND
                             column_name = '" + AttrName + "'";
        String StringQueryIfArray = " select data_type from
                          information_schema.element_types " +
```

```
                        " where object_schema = 'public' AND
                        object_name = '" + tableName + "' AND
                        array_type_identifier =  (Select
                        ordinal_position " + " from
                        information_schema.columns where
                        table_schema = 'public' AND table_name = '" +
                        tableName + "' AND column_name = '" +
                        AttrName + "');";

        ResultSet R = database.query(StringQuery);
        // Database desconection
        database.disconnect();
        // transform the type
        try {
            if (R.next()) {
                String data_type =  R.getString(1);
               data_type = convertDataType(data_type, tableName,
                        AttrName);
               return data_type;
            }else
                return "";
        }catch ( java.sql.SQLException e ) {
                System.err.println("Postgres Connection: ResultSet
                                Error.");
                e.printStackTrace();
                return "";
        }
    }
/*********************************************************************/
    /** public String AttributeRestriction(String tableName, String
     *  AttrName)
     *      Return the restriction of an attribute
     *  Input parameters :
     *      String tableName : Name of the table that contains the
     *        attribute
     *      String AttrName : Name of the attribute
     *  Output parameters :
     *      String : String with the restriction of the attribute
     */
    public String AttributeRestriction(String tableName, String
         AttrName){

        // Database Connection
        database.connect();
        // Make the query
        String StringQuery =" select constraint_type from
                            information_schema.table_constraints " +
                            " where table_schema = 'public' AND " +
                            " ( constraint_type = 'PRIMARY KEY' OR
                            constraint_type = 'UNIQUE') AND" +
                            " constraint_name IN ( select
```

```
                            constraint_name " +
                            "from
                            information_schema.constraint_column_usage"
                            + " where table_schema = 'public' AND
                            table_name = '" + tableName + "' AND
                            column_name = '" + AttrName + "')";

        ResultSet R = database.query(StringQuery);
        // Database desconection
        database.disconnect();
        try {
            if (R.next()) {
                return R.getString(1);
            }else
                return "";
        }catch ( java.sql.SQLException e ) {
                System.err.println("Postgres Connection: ResultSet
                 Error.");
                e.printStackTrace();
                return "";
        }
      }
/*****************************************************************/
    /**  public LinkedList ForeignKeyRelation()
     *      Return a list with all the foreign key of the database
     *  Output parameters :
     *      String : list of string[4] ex :(t, a, tref, aref)
     */
    public LinkedList ForeignKeyRelation(){

        // Database Connection
        database.connect();
        String StringQuery1 = "Create view FK as (select
                 constraint_name " +
                        "from information_schema.table_constraints"
                        + "where table_schema = 'public' " +
                        "AND constraint_type = 'FOREIGN KEY' )";

         String StringQuery2 = "select   KCU.table_name,
                         KCU.column_name," +
                        " CCU.table_name, CCU.column_name " +
                        "from
                         information_schema.constraint_column_usage
                         CCU, " + "information_schema.key_column_usage
                         KCU, FK "+ "where  FK.constraint_name =
                         CCU.constraint_name " +" AND
                         FK.constraint_name =KCU.constraint_name";
        String StringQuery3 = "drop view FK";
        database.update(StringQuery1);
        ResultSet R1 = database.query(StringQuery2);
        database.update(StringQuery3);
```

```
        //Make Return List
        LinkedList ListFKRel = new LinkedList();
        try {
            while (R1.next()) {
                String FKRel[] = new  String[4];
                for (int i=1; i<= 4; i++) {
                    FKRel[i-1] = R1.getString(i);
                }
                ListFKRel.add(FKRel);
            }
        }catch ( java.sql.SQLException e ) {
                System.err.println("Postgres Connection: ResultSet
                                    Error.");
                e.printStackTrace();
        }
        // Database Connection
        database.disconnect();
        return ListFKRel;
      }
/*****************************************************************/
    /**  public ResultSet getResultQuery(String S)
     *          return a restul set of a query
     *  Input parameters:
     *      String S : query
     *  Output parameters:
     *      ResultSet : result of the query
     */
    public ResultSet getResultQuery(String S) {
       // Database Connection
        database.connect();
        ResultSet R = database.query(S);
        // Database desconection
        database.disconnect();
        return R;
    }
/*****************************************************************/
    /**  public void createTable(String StrQuery)
     *      create a table in the database
     *  Input parameters :
     *      String StrQuery: lquery of the creation
     */
    public void createTable(String StrQuery){

        this.database.connect();
        this.database.update(StrQuery);
        this.database.disconnect();
    }
/*****************************************************************/
    /**  public void createDatabase(String StrMultiDB)
     *         create a new database into the database manager
     *  Input parameters:
```

```
     *      String StrMultiDB: Name of th enew database
     */
    public void createDatabase(String StrMultiDB) {

        String S = "CREATE DATABASE \"" + StrMultiDB +
                   "\" ENCODING = 'UTF8' " +
                   " TABLESPACE = pg_default ;";
        // Database Connection
        database.connect();
        database.update(S);
        // Database desconection
        database.disconnect();
    }
/*****************************************************************/
    /**  public void makeUpdate(String Name, ResultSet R)
     *         load a result set into a table of the database
     *  Input parameters:
     *      String Name: Name of the table to insert the data
     *      ResultSet R: data to insert into the table
     */
    public void makeUpdate(String Name, ResultSet R) {

        try {
            ResultSetMetaData md = R.getMetaData();
            while (R.next()) {
                String I = "INSERT INTO \"" + Name + "\" VALUES (";
                for (int i = 1;  i <= md.getColumnCount(); i++){
                    int type = md.getColumnType(i);
                    switch (type) {
                        case java.sql.Types.VARCHAR:
                        case java.sql.Types.LONGVARCHAR:
                            I += "'" + R.getString(i) + "'";
                            break;
                        case java.sql.Types.CHAR:
                            I += "'" + R.getString(i) + "'";
                            break;
                        case java.sql.Types.BOOLEAN:
                            I += "'" + R.getString(i) + "'";
                            break;
                        case java.sql.Types.DECIMAL:
                        case java.sql.Types.DOUBLE:
                        case java.sql.Types.FLOAT:
                        case java.sql.Types.INTEGER:
                        case java.sql.Types.NUMERIC:
                        case java.sql.Types.REAL:
                        case java.sql.Types.SMALLINT:
                            I += R.getString(i) ;
                            break;
                        case java.sql.Types.LONGVARBINARY:
                        case java.sql.Types.STRUCT:
                            I+= R.getString(i) ;
```

```
                        break;
                    case java.sql.Types.DATE:
                    case java.sql.Types.TIME:
                    case java.sql.Types.TIMESTAMP:
                        I+= R.getString(i);
                        break;
                    case java.sql.Types.OTHER:
                    default :
                        I+= R.getString(i);
                }
                if (i == md.getColumnCount()) I+= " );";
                else I += " , ";
            }
            // Database Connection
            database.connect();
            this.database.update(I);
            // Database desconection
            database.disconnect();
        }
    }catch ( java.sql.SQLException e ) {
        System.err.println("Postgres Connection: ResultSet
                    Error.");
        e.printStackTrace();
    }
}
/*********************************************************************
    * PRIVATE METHODS OF CLASS Metadata
*********************************************************************/
    /**  private void print_ResultSet(ResultSet R)
     *      Print into the standard output a result set
     */
    private void print_ResultSet(ResultSet R){
        try {
            ResultSetMetaData md = R.getMetaData();
            int count = md.getColumnCount();
            for (int i=1; i<=count; i++) {
                System.out.print(md.getColumnLabel(i));
                System.out.print("  ");
            }
                System.out.println("");
            while (R.next()) {
                for (int i=1; i<= count; i++) {
                    System.out.print(R.getString(i));
                    System.out.print("  ");
                }System.out.println();
            }
        }catch ( java.sql.SQLException e ) {
                System.err.println("Postgres Connection: Conennection
                                Error.");
                e.printStackTrace();
        }
```

```
}
/*****************************************************************/
    /** private String convertDataType(String data_type, String
     * tableName, String AttrName)
     *      extracts the type of an SQL ARRY of postgreSQL
     *  Input parameters:
     *      String data_type : type of the attribute
     *      String tableName : name of the table that contains the
attribute
     *      String AttrName : name of the attribute
     */
    private String convertDataType(String data_type, String tableName,
                String AttrName) {

        if (data_type.equals("ARRAY")){
            // Make the query;
            String StringQuery = " select data_type from
                        information_schema.element_types " +
                        " where object_schema = 'public' AND
                        object_name = '" + tableName + "' AND
                        array_type_identifier =  (Select
                        ordinal_position " + " from
                        information_schema.columns where table_schema
                        = 'public' AND table_name = '" +
                        tableName + "' AND column_name = '" + AttrName
                        + "');";
            // Database Connection
            database.connect();
            ResultSet R = database.query(StringQuery);
            // Database desconection
            database.disconnect();
            try {
                if (R.next()) return (R.getString(1)+"[]");
                else return "";
            }catch ( java.sql.SQLException e ) {
                System.err.println("Postgres Connection: ResultSet
                                Error.");
                e.printStackTrace();
                return "";
            }
        }else{
            return data_type;}
      }
}
```

A.5.2 Class Postgres

```
/*
 * Postgresql.java
```

```
 * Created on November 16, 2006, 3:17 PM
 */
/*********************************************************************
 * Package
*********************************************************************/
package databaseConnection;
/*********************************************************************
 * Import Class
*********************************************************************/
import java.sql.*;
/**
 * @author Ana M Giral
 */
public class Postgresql implements AConnection {


/*****************************************************************
     * FIELD DECLARATION
*****************************************************************/
    /*Connected*/
    private boolean isConnected = false;
    /*Connectin*/
    private Connection connection;
    /*Statement y ResultSet*/
    private Statement statement;
    private ResultSet resultSet;
    /*Paramof connection*/
    private String ip;
    private String port;
    private String namespace;
    private String user;
    private String pass;
/*****************************************************************
     * PUBLIC METHODS OF CLASS RELATIONAL_MODEL
*****************************************************************/
/*****************************************************************
     * public Metadata(String theIp, String thePort, String
        theNamespace, String theUser, String thePass)
     *      Creates a new instance of Postgesql
     * Input parameters:
     *      String theIp : IP or URL to connect to the database manager
     *      String theDBPort : Port of the database server
     *      String theNamespace : Namessapce of the relational database
     *      String theDBUser: Name of the user
     *      String thePass : User's password
*****************************************************************/
    public Postgresql( String theIp, String thePort, String
        theNamespace, String theUser, String thePass ) {
       /*Almacenamos los parametros de conexion*/
       ip = theIp;
       port = thePort;
```

```
        namespace = theNamespace;
        user = theUser;
        pass = thePass;
        isConnected = false;
    }
/*****************************************************************/
    /**  public boolean connect()
     *      makes the connection to the database
     */
    public boolean connect() {

        if ( ! isConnected ) {
            try {
                Class.forName("org.postgresql.Driver");
                connection = DriverManager.getConnection(
                          "jdbc:postgresql://" + ip + ":" + port +"/" +
                          namespace , user, pass );
                isConnected = true;
                return true;
            } catch ( java.lang.ClassNotFoundException e ) {
                System.err.println("Postgres Connection: Driver jdbc
                                    not found.");
                e.printStackTrace();
                return false;
            } catch ( java.sql.SQLException e ) {
                System.err.println("Postgres Connection: Conennection
                                    Error.");
                e.printStackTrace();
                return false;
            }
        }
        else
            return true;
    }
/*****************************************************************/
     /** public boolean disconnect()
      *     makes the Disconnection to the database
      */
    public boolean disconnect() {
        if ( isConnected ) {
            try {
                connection.close();
                isConnected = false;
                return true;
            } catch ( java.sql.SQLException e ) {
                System.err.println("PostgreSQL Connection:
                                    Disconnection Error.");
                e.printStackTrace();
                return false;
            }
        }
```

```
        else
            return true;
    }
/*********************************************************************/
    /** public boolean connected
     * Return true if is conected to the database
     */
    public boolean connected () {
        return isConnected;
    }
/*********************************************************************/
    /** public ResultSet query ( String strQuery )
     *      Make a query in the database
     *  Input parameters :
     *      String Query: SQL query
     *  Output parameters :
     *      ResultSet: Result of the query
     */
    public ResultSet query ( String strQuery ) {
        if ( isConnected ) {
            try {
                statement = connection.createStatement();
                resultSet = statement.executeQuery( strQuery );
                return resultSet;
            } catch( java.sql.SQLException e ) {
                System.err.println("PostgreSQL Connection: Error in the
                                    query execution.");
                e.printStackTrace();
                return null;
            }
        }
        else
            return null;
    }
/*********************************************************************/
    /** public void update ( String strUpdate )
     *      Make aN UPDATE in the database
     *  Input parameters :
     *      String Query: SQL UPDATE
     */
    public void update ( String strUpdate ) {
        if ( isConnected ) {
            try {
                statement = connection.createStatement();
                statement.executeUpdate( strUpdate );
            } catch( java.sql.SQLException e ) {
                System.err.println("Postgres Connection: Error in
                                    update execution.");
                System.err.println(strUpdate);
                e.printStackTrace();
            }
```

```
        }
    }
}
```

A.6 multidimensionalBuilder

A.4.1 Class userInteraction

```
/*
 * userInteraction.java
 * Created on 20 de enero de 2007, 17:54
 */

package multidimensionalbuilder;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import Relational_Model.*;
import Relational_Model.Paths.*;
import Relational_Model.Path.*;

/**
 * @author Ana Maria Giral Castillo
 */
public class userInteraction extends JDialog {

    Attribute SelectedAttribute;
    LinkedList SelectedAttributes;
    JPanel ThePanel;
    /** Creates a new instance of userInteraction */
    public userInteraction(JPanel TheJPanel) {
        this.ThePanel = TheJPanel;
    }


    public LinkedList userGrainAttributeSelection(LinkedList
         Attributes){

        LinkedList FinalAttributes = new LinkedList();
        ListIterator ListIt = Attributes.listIterator();
        Vector AttributesName = new Vector(Attributes.size());
        int i = 0;
        while (ListIt.hasNext()){
```

```
            AttributesName.add(i ,((Attribute)ListIt.next()).Name);
            i++;
        }
        boolean more= true;
        while (more) {
            String response =
                 (String)JOptionPane.showInputDialog(ThePanel,
                "Select the grain attribute of the table ?",
                "Grain attribute Selection",
                 JOptionPane.QUESTION_MESSAGE,
                null, AttributesName.toArray(),
                (String)AttributesName.get(i-1));

            FinalAttributes.add(
                 Attributes.get(AttributesName.indexOf(response)));

           int n = JOptionPane.showConfirmDialog(
                ThePanel, "Select more Gain attribute from this
                 Table?",
                "An Inane Question",
                JOptionPane.YES_NO_OPTION);

            if (n == JOptionPane.NO_OPTION) {
                more = false;
            }
        }
        return FinalAttributes;
    }

    //user select the pahth
    public Paths UserPathSelection(Paths ThePaths){

        Paths FinalPaths = new Paths();
        Vector PathsName = new Vector(ThePaths.size());
        ListIterator ListIt = ThePaths.getIterator();

        while (ListIt.hasNext()){
            Path P = (Path)ListIt.next();
            PathsName.add( P.showPath() );
        }

        boolean more= true;
        while (more) {
            String response =
                 (String)JOptionPane.showInputDialog(ThePanel,
                "Select path from grain to measure table ?",
                "Measure ways", JOptionPane.QUESTION_MESSAGE,
                null, PathsName.toArray(),
                (String)PathsName.get(0));

            FinalPaths.addPath(ThePaths.getPath(
```

```
                PathsName.indexOf((Object)response)));

            int n = JOptionPane.showConfirmDialog(
                ThePanel, "Select more Pathsfor these grain
                 attribute?",
                "An Inane Question",
                JOptionPane.YES_NO_OPTION);

            if (n == JOptionPane.NO_OPTION) {
                more = false;
            }
        }
        return FinalPaths;
    }

    public Paths chooseHierarchies(Paths Hierarchies) {
        Paths FinalPaths = new Paths();
        Vector PathsName = new Vector(Hierarchies.size());
        ListIterator ListIt = Hierarchies.getIterator();

        while (ListIt.hasNext()){
            Path P = (Path)ListIt.next();
            PathsName.add( P.showPath() );
        }
        boolean more= true;
        while (more) {
            String response =
                 (String)JOptionPane.showInputDialog(ThePanel,
                "Select the hierarchy that you want ?",
                "Measure ways", JOptionPane.QUESTION_MESSAGE,
                null, PathsName.toArray(),
                (String)PathsName.get(0));

            FinalPaths.addPath( Hierarchies.getPath(
                          PathsName.indexOf((Object)response)));
            int n = JOptionPane.showConfirmDialog(
                ThePanel, "Select more hierarchies?",
                "An Inane Question",
                JOptionPane.YES_NO_OPTION);

            if (n == JOptionPane.NO_OPTION) {
                more = false;
            }
        }
        return FinalPaths;
    }

     public String chooseAggregator(String Name) {

        String Aggregator[]={"avg", "count", "max", "min", "sum"};
```

```
        return  (String)JOptionPane.showInputDialog(ThePanel,
                "Select the aggregator for the measure " + Name,
                "Measure aggregators", JOptionPane.QUESTION_MESSAGE,
                null, Aggregator,
                (String)Aggregator[4]);
    }

     public String getMultiDBName() {
        return  (String)JOptionPane.showInputDialog(ThePanel,
                "Type the name for the new Multidimensional Database ",
                "Name of Multidimensional Database",
                 JOptionPane.QUESTION_MESSAGE,
                null, null,
                null);
    }

    public LinkedList selectHierarchyAttributes(LinkedList
         TheAttributes){
        hierarchyAttributes HA = new
                 hierarchyAttributes(TheAttributes);
        return this.SelectedAttributes;
    }

    public class hierarchyAttributes extends JDialog{

        JPanel Presentation;
        LinkedList TheAttributes;
        ArrayList LBox;
        /** Creates a new instance of userInteraction */
        public hierarchyAttributes( LinkedList Attrib){

            SelectedAttributes = new LinkedList();
            TheAttributes = Attrib;
            this.setModal(true);
            this.setTitle("Selection of the hierarchy attributes");
            this.setSize(400,300);
            this.selectAttributes();
            this.setVisible(true);
        }

        public void selectAttributes(){
            //Panel Inicialization
            Presentation = new JPanel();
            Presentation.setLayout(new BorderLayout());
            JLabel JLabelTittle = new JLabel("Select the aatributes
                 that will be involve in the hierarchy/n" +
                 " the already selected attributes ...level  ");
            Presentation.add(JLabelTittle, BorderLayout.NORTH);

            JPanel APanel = new JPanel();
```

```
        APanel.setLayout(new GridLayout(TheAttributes.size()+1,2));
        JLabel JLabelSubTitlle1 = new JLabel(" Attribute ");
        JLabel JLabelSubTitlle2 = new JLabel("Table Name ");
        APanel.add(JLabelSubTitlle1);
        APanel.add(JLabelSubTitlle2);
        LBox = new ArrayList();
        JCheckBox JCB;
        CheckBoxListener listenCheck = new CheckBoxListener();

        ListIterator ListIt = TheAttributes.listIterator();
        int i = 0;
        while ( ListIt.hasNext() ){
            Attribute A =  (Attribute)ListIt.next();
                JLabel TableLabel = new JLabel(A.TableName);
                JCB = new JCheckBox(A.Name);
                // the hierarchy attributes will be the grain
                if (A.isPK()){
                    JCB.setSelected(true);
                    SelectedAttributes.add(A);
                }
                else{
                    if ( A.isFK()) {
                        JCB.setEnabled(false);
                    }
                    else
                        JCB.setSelected(false);
                }
                JCB.addItemListener(listenCheck);
                LBox.add(JCB);
                APanel.add(JCB);
                APanel.add( TableLabel );
        }
        JButton JButtonOK = new JButton();
        JButtonOK.setText("FINISH");
        JButtonOK.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
              //getContentPane().
               dispose();// close windows and return value
               ThePanel.repaint();
            }
        });
        JScrollPane TheScrollPane = new JScrollPane(APanel);
        Presentation.add(TheScrollPane, BorderLayout.CENTER);
        Presentation.add(JButtonOK, BorderLayout.SOUTH);
        this.getContentPane().removeAll();
        this.getContentPane().add(Presentation);
    }

    class CheckBoxListener implements ItemListener {
        public void itemStateChanged(ItemEvent e) {
```

```
                Object source = e.getItemSelectable();
                if (e.getStateChange() == ItemEvent.SELECTED)  {
                    SelectedAttributes.add(TheAttributes.get(
                                        (LBox.indexOf(source))));
                }
                else {
                    if (e.getStateChange() == ItemEvent.DESELECTED)  {
                        SelectedAttributes.remove(TheAttributes.get(
                                        (LBox.indexOf(source))));
                    }
                }
            }
        }
    }
}
```

A.4.1 Class Multidimensionalbuilder

```
/*
 * multidimensionalbuilder.java
 * Created on November 15, 2006, 4:16 PM
 */
/*****************************************************************
 * Package
*****************************************************************/
package multidimensionalbuilder;
/*****************************************************************
 * Import Class
*****************************************************************/
import Multidimensional_Construction.*;
import Multidimensional_Construction.Multidimensional_Schemas;
import Relational_Model.Attribute;
import Multidimensional_Model.Dimension;
import Multidimensional_Model.AttributeMulti;
import Relational_Model.Paths;
import Relational_Model.Table;
import java.io.IOException;

import java.util.*;
import org.freixas.jwizard.*;
import javax.swing.*;
import java.awt.*;
import java.net.URL;
import java.awt.*;
import java.awt.event.*;

/**
 *
 * @author Ana M Giral
```

```
 */
public class multidimensionalBuilder extends  JWizardDialog {

    Relational_Schemas Relational;
    Multidimensional_Schemas Multidimensional;

    JPanel JPanelStep1;
    JPanel JPanelStep2;
    JPanel JPanelStep3;
    JPanel JPanelStep4;
    JPanel JPanelStep5;
    JPanel JPanelStep6;
    LinkedList SelectedTables;
    LinkedList SelectedAttributes;
    LinkedList GrainAttributes;
    String MeasureTable;
    String MeasureAttribute;

    String StrDBURL;
    String StrDBPort;
    String StrDBNamespace;
    String StrDBUser;
    String StrDBPwd;


/*********************************************************************
     * MAIN
*********************************************************************/
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        new multidimensionalBuilder();
        System.exit(0);
    }



/*********************************************************************
     * PUBLIC METHODS OF CLASS MULTIDIMENSIONALBUILDER
*********************************************************************/
    /** Creates a new instance of multidimensionalBuilder */
    public multidimensionalBuilder() {
        // We want the dialog modal
        setModal(true);
        // Set the dialog title
        setTitle("Multidimensional Builder");
        // Set the logo image
        URL url = getClass().getResource("banned.gif");
        setWizardIcon(new ImageIcon(url));
```

```
        // Create each step
        addWizardPanel(new Step0());
        addWizardPanel(new Step1());
        addWizardPanel(new Step2());
        addWizardPanel(new Step3());
        addWizardPanel(new Step4());
        addWizardPanel(new Step5());

        // Make the dialog visible
        pack();
        setVisible(true);
  }


//*********************************************************************
    // Protected
//*********************************************************************
*
    /**
     * If the user presses cancel, we want to give him/her the option
     * of continuing with the installation.
     */
    protected void cancel(){
        int response =
                JOptionPane.showConfirmDialog(
                this,
                "Cancel the program execution?",
                "Finish Program",
                JOptionPane.OK_CANCEL_OPTION);

        if (response == JOptionPane.OK_OPTION) {
            super.cancel();
        }
    }
//*********************************************************************
// Inner Classes
//*********************************************************************

//*********************************************************************
// Step0
//*********************************************************************
    // The user has to introduce the data od the database manager and
    // the system extract the information

    private class Step0 extends JWizardPanel{

        JTextField JTextFielDBURL;
        JTextField JTextFielDBPort;
        JTextField JTextFielDBNamespace;
        JTextField JTextFielDBUser;
        JTextField JTextFielDBPwd;
```

```
ArrayList LBox;
//Group the radio buttons.
ButtonGroup ButtonTableGroup;
LinkedList TheTables;

public Step0() {

    setStepTitle("Database information");

    JPanel contentPane = getContentPane();
    contentPane.setLayout(new BorderLayout());

    JLabel label =new JLabel(
            "The manager database information of the relational
         dataabse is needed");

    contentPane.add(label, BorderLayout.NORTH);
    // Fields for recober the information
    JPanel PanelForm = new JPanel();
    PanelForm.setLayout(new GridLayout(5,2));


    JLabel JLabelDBURL = new JLabel("Database Server URL : ");
    JLabel JLabelDBPort = new JLabel("Database Server Port :");
    JLabel JLabelDBNamespace=new JLabel("Database Namespace:");
    JLabel JLabelDBUser = new JLabel("Database User : ");
    JLabel JLabelDBPwd = new JLabel("Password");

    JTextFielDBURL = new JTextField("localhost");
    JTextFielDBPort = new JTextField("5432");
    JTextFielDBNamespace = new JTextField("test");
    JTextFielDBUser = new JTextField("postgres");
    JTextFielDBPwd = new JTextField("postgres");

    PanelForm.add(JLabelDBURL);
    PanelForm.add(JTextFielDBURL);
    PanelForm.add(JLabelDBPort);
    PanelForm.add(JTextFielDBPort);
    PanelForm.add(JLabelDBNamespace);
    PanelForm.add(JTextFielDBNamespace);
    PanelForm.add(JLabelDBUser);
    PanelForm.add(JTextFielDBUser);
    PanelForm.add(JLabelDBPwd);
    PanelForm.add(JTextFielDBPwd);
    contentPane.add(PanelForm, BorderLayout.CENTER);

    // Set the previous (none) and next steps

    setBackStep(-1);
    setNextStep(1);
```

```
        // Logic
        SelectedTables = new LinkedList();
    }

    // We're going to override the next button so we can check if
    // there are some empty JTextField
    protected void next(){

        StrDBURL = JTextFielDBURL.getText();
        StrDBPort = JTextFielDBPort.getText();
        StrDBNamespace = JTextFielDBNamespace.getText();
        StrDBUser = JTextFielDBUser.getText();
        StrDBPwd = JTextFielDBPwd.getText();

        String SMS = "";
        if (StrDBURL.equals(""))
             SMS = SMS + "\n\tDatabase Server URL";
        if (StrDBPort.equals(""))
             SMS = SMS + "\n\tDatabase Server Port";
        if (StrDBNamespace.equals(""))
             SMS = SMS + "\n\tDatabase Namespace";
        if (StrDBUser.equals(""))
             SMS = SMS + "\n\tDatabase User";
        if (StrDBPwd.equals(""))
             SMS = SMS + "\n\tDatabase Password";
        if (!SMS.equals("") ) {
            JOptionPane.showMessageDialog(
               Step0.this,
               "Please enter the data in the next fields :" + SMS,
               "Error",
               JOptionPane.ERROR_MESSAGE);
        }else{  //NO EMPTY FIELDS
            ////////////////////////////// LOGIC
            // Relational model instancion and creation
            Relational = new Relational_Schemas(StrDBURL,
               StrDBPort, StrDBNamespace, StrDBUser,StrDBPwd);
            Multidimensional = new
               Multidimensional_Schemas(Relational);
            Relational.ExtractRelationalModel();
            // Extract Tables
            TheTables = Relational.showTables();

            ////////////////////////////// GUI
            JPanelStep1.setLayout(new
                     GridLayout(TheTables.size(),1));
            ListIterator ItTables = TheTables.listIterator();
            LBox = new ArrayList(TheTables.size());
            int i = 0;
            JRadioButton JRB;
            ButtonTableGroup = new ButtonGroup();;
            RadioListener listenCheck = new RadioListener();
```

```
                JRB = new JRadioButton("Tables");
                while (ItTables.hasNext()){
                    JRB = new JRadioButton((String)ItTables.next());
                    JRB.setActionCommand(JRB.getText());
                    JRB.addActionListener(listenCheck);
                    ButtonTableGroup.add(JRB);
                    JPanelStep1.add(JRB);
                }
                super.next();
            }
            return;
        }
        class RadioListener implements ActionListener {
            public void actionPerformed(ActionEvent e) {
                    MeasureTable = e.getActionCommand();
            }
        }
    }


//*********************************************************************
    // Step1
//*********************************************************************

    // User select the tables
    private class Step1 extends JWizardPanel{

        JPanel contentPane;
        ArrayList LBox;
        LinkedList TheAttributes;

        public Step1() {

            setStepTitle("Measure Attribute Selection");
            contentPane = getContentPane();
            contentPane.setLayout(new BorderLayout());

            //JPanel Inicialization
            JPanelStep1 = new JPanel();
            JLabel label =new JLabel( "Choose the table that conteins
                                  the measuete attribute");
            contentPane.add(label, BorderLayout.NORTH);
            contentPane.add(JPanelStep1, BorderLayout.CENTER);
            setBackStep(0);
            setNextStep(2);

            /// logic
            SelectedAttributes = new LinkedList();
        }

        // OverWrite the next button in order tu add
```

```
        protected void next(){
            // Select the tables
            // build the next screen with attributes for the measure
           //Table

            TheAttributes = Relational.selectTables(MeasureTable);
            ListIterator ListIt = TheAttributes.listIterator();

            JPanelStep2.setLayout(new
                                  GridLayout(TheAttributes.size(),1));
            LBox = new ArrayList();
            JRadioButton JRB;

            ButtonGroup ButtonTableGroup = new ButtonGroup();;
            RadioListener listenCheck = new RadioListener();

            JRB = new JRadioButton("Atttributes");
            while (ListIt.hasNext()){
                JRB = new
                      JRadioButton(((Attribute)ListIt.next()).Name);
                JRB.setActionCommand(JRB.getText());
                JRB.addActionListener(listenCheck);
                ButtonTableGroup.add(JRB);
                JPanelStep2.add(JRB);
            }
            super.next();
        }
        class RadioListener implements ActionListener {
            public void actionPerformed(ActionEvent e) {
                    MeasureAttribute = e.getActionCommand();
            }
        }
    }

//*********************************************************************
    // Step2
//*********************************************************************

    // Select the concrete attribute for the measure table

    private class Step2 extends JWizardPanel{

        JPanel contentPane;
        ArrayList LBox;
        LinkedList TheAttributes;
        LinkedList TheTables;

        public Step2() {

            setStepTitle("Attribute Selection");
```

```
            contentPane = getContentPane();
            contentPane.setLayout(new BorderLayout());

            JLabel label =new JLabel("Choose the measure Attribute");
            JPanelStep2 = new JPanel();
            contentPane.add(label, BorderLayout.NORTH);
            contentPane.add(JPanelStep2, BorderLayout.CENTER);


            setBackStep(1);
            setNextStep(3);

            /// logic
            GrainAttributes = new LinkedList();

        }

        // OverWrite the next button in order tu add funcionality
        protected void next(){
            //Select the measure attribute
            Relational.SelectMeasure(MeasureTable, MeasureAttribute);
             // Extract Tables of the realctional model
            TheTables = Relational.showTables();

            ////////////////////////////////GUI
            // Show all the tables for the user
            JPanelStep3.setLayout(new GridLayout(TheTables.size(),1));
            ListIterator ItTables = TheTables.listIterator();
            LBox = new ArrayList(TheTables.size());

            int i = 0;
            JCheckBox JCB;
            CheckBoxListener listenCheck = new CheckBoxListener();
            while (ItTables.hasNext()){
                JCB = new JCheckBox((String)ItTables.next());
                JCB.setSelected(false);
                JCB.addItemListener(listenCheck);
                LBox.add(JCB);
                JPanelStep3.add(JCB);
            }
            super.next();
        }

    class CheckBoxListener implements ItemListener {
        public void itemStateChanged(ItemEvent e) {

            Object source = e.getItemSelectable();
            if (e.getStateChange() == ItemEvent.SELECTED)  {
                SelectedTables.add( TheTables.get(
                                        (LBox.indexOf(source))));
                ////////////////////LOGIC
```

```
                //select the table and the grain attributes of the
                //table
                Paths PathTableSelected = Relational.selectGrainTable(
                        (String) TheTables.get(
                        (LBox.indexOf(source))), JPanelStep3);
                // make the dimension for this grain attributes
                Multidimensional.make_dimensions(PathTableSelected ,
                                                    JPanelStep3);
            } else {
                if (e.getStateChange() == ItemEvent.DESELECTED)  {
                    SelectedTables.remove(TheTables.get(
                                        (LBox.indexOf(source))));
                }
            }
        }
    }
}
//*********************************************************************
// Step3
//*********************************************************************
    // User grain Attributes

    private class Step3 extends JWizardPanel{

        JPanel contentPane;
        ArrayList LBox;
        LinkedList TheAttributes;
        LinkedList SelectedAttributes;

        public Step3() {

            setStepTitle("What do you want to know");

            contentPane = getContentPane();
            contentPane.setLayout(new BorderLayout());
            JLabel label =new JLabel( "Choose all the Tables with Grain
                        Attributes for the multidimensional model");
            JPanelStep3 = new JPanel();
            contentPane.add(label, BorderLayout.NORTH);
            contentPane.add(JPanelStep3, BorderLayout.CENTER);

            setBackStep(2);
            setNextStep(4);

            /// logic
            SelectedAttributes = new LinkedList();
        }

        protected void next(){

            //Create The Fact Table
```

```
userInteraction UI = new userInteraction(JPanelStep4);
String Aggregator =
     UI.chooseAggregator(Relational.getMeasure().TableName+
     "." + Relational.getMeasure().Name);
Multidimensional.make_FactTable(
                      Relational.getMeasure(), Aggregator);
//show purpose
LinkedList Dimensions = Multidimensional.dimensions();
ListIterator theDimensions = Dimensions.listIterator();
Object[] DimensionsName =
                      Multidimensional.getDimensionsName();
Dimension D = null;
GridLayout GridLay = new GridLayout();
GridLay.setRows(Dimensions.size()+1);
JPanelStep4.setLayout(GridLay);

// FACT TABLE
    LinkedList Measure = Multidimensional.Measures();

    ListIterator Measures = Measure.listIterator();
    JPanel JPanelMeasures  = new JPanel();
    JPanelMeasures.setLayout(
             new GridLayout(Measure.size() + 1, 2));
    JLabel LabelM = new JLabel("FACT TABLE");

    JPanelMeasures.add(LabelM);
    JPanelMeasures.add(new JLabel("MEASURE"));
    JPanelMeasures.add(new JLabel(" "));

    while (Measures.hasNext()){
        String MeasureName = (String)Measures.next();
        JLabel LabelA = new JLabel(MeasureName);
        JPanelMeasures.add(new JLabel(" "));
        JPanelMeasures.add(LabelA);
        JPanelMeasures.add(new JLabel(" "));
    }
    JPanelStep4.add(JPanelMeasures);

//DIMENSION
JPanel JPanelDimensions = new JPanel();
JPanelDimensions.setLayout(
             new GridLayout(1, Dimensions.size()));
while (theDimensions.hasNext()){

    D = (Dimension)theDimensions.next();
    JPanel JPanelDimension = new JPanel();
    JPanelDimension.setLayout(new BorderLayout());
    JPanelDimension.add(new JLabel("DIMENSION : "+ D.Name),
                                    BorderLayout.NORTH);
    JPanelDimension.add(D.getJTree(), BorderLayout.CENTER);
    JScrollPane treeView=new JScrollPane(JPanelDimension );
```

```
                JPanelDimensions.add(treeView);
            }
                JPanelStep4.add(JPanelDimensions);
                super.next();
        }
    }
//*********************************************************************
// Step4
//*********************************************************************
// Making  ways between the grain attribute and the measure

    private class Step4 extends JWizardPanel{

        JPanel contentPane;
        ArrayList LBox;
        LinkedList TheAttributes;
        LinkedList SelectedAttributes;

        public Step4() {

            setStepTitle("Modification of the dimensions ");
            contentPane = getContentPane();
            contentPane.setLayout(new BorderLayout());
            JLabel label =new JLabel(" multidimensional model");
            JPanelStep4 = new JPanel();
            contentPane.add(label, BorderLayout.NORTH);
            contentPane.add(JPanelStep4, BorderLayout.CENTER);

            setBackStep(3);
            setNextStep(5);
        }

        protected void next() {
            try{
                ////////////////////LOGIC
                userInteraction UI = new userInteraction(JPanelStep5);
                String StrDBMultiName = UI.getMultiDBName();
                Multidimensional.createStarSchema();
                Multidimensional.createMultiDimensionalDatabase(
                          StrDBMultiName, StrDBURL, StrDBPort,
                          StrDBUser, StrDBPwd);
                Multidimensional.createTransformationSchema();
                Multidimensional.createOLAPSchema(StrDBMultiName,
                          StrDBURL, StrDBPort, StrDBUser, StrDBPwd);
                Multidimensional.loadMultidimensionalDatabase();
            } catch (IOException ioe){
                ioe.printStackTrace();
            }
            super.next();
        }
    }
```

```
//*********************************************************************
// Step5
//*********************************************************************
    // Making dimensions
    private class Step5 extends JWizardPanel{

        JPanel contentPane;
        ArrayList LBox;
        LinkedList TheAttributes;
        LinkedList SelectedAttributes;

        public Step5() {
            setStepTitle("SchemaCreated");

            contentPane = getContentPane();
            contentPane.setLayout(new BorderLayout());
            JLabel label =new JLabel(" The schema has been created");
            JPanelStep6 = new JPanel();
            contentPane.add(label, BorderLayout.NORTH);
            contentPane.add(JPanelStep6, BorderLayout.CENTER);

            setBackStep(4);
            setNextStep(6);
        }}
}
```

Bibliography

[Kimball, Ross 2002] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: the complete guide to dimensional modelling* . Second edition. Wiley Publishing, Inc 2002.

[W.H. Inmon 2005] W.H. Inmon. *Building the Data Warehouse*, Fourth Edition. Wiley Publishing, Inc 2005.

[Hackathorn, Inmon 1994] Richard D. Hackathorn and W.H. Inmon . *Using the Data Warehouse*. Wiley 1994.

[Erik Thomsen 2002] Erik Thomsen. *OLAP Solutions . building Multidimensional Information Systems.* Second Edition. Wiley Publishing, Inc 2002

[John Wang 2006] John Wang. Encyclopedia *of Data Warehousing and Mining* . Idea Group 2006

[Gerald Grant 2003] Gerald Grant. *ERP & Data Warehousing in Organizations: Issues and Challenges*. Idea Group Publishing 2003.

[Elmasri, Navathe 1998] R. Elmasri and S. B. Navathe. *Fundamentals of Database System*, Second Edition . Addison-Wesley 1998