THE DEVELOPMENT OF AN INTEGRATED TOOL TO MANAGE THE CONTRACTUAL RELATIONSHIPS OF ELECTRICITY STAKEHOLDERS

5th March 2005

Contents

Chapter 1

Project Objectives

1.1 Introduction: New Legislation

On December 19, 1996, the European Parliament adopted the directive 96/92/EC. The directive established common rules for generation, transmission and distribution of electricity. It laid down the rules relating to the organization and functioning of the electricity sector, access to the market, the criteria and procedures applicable to call for tender and the grating of authorizations and the operations of systems[1]. The main purpose of such a directive was the liberation of the electricity market among the states of the European Union. As specified by the procedures of the legislation, every member of the Union has emanated decrees to perform all the necessary changes in order to achieve a free market. The main change brought about by the free market is that every eligible costumer can buy electricity from anyone in any European state. In theory, this should lead to a regime of full competition and lower the price of the electricity. Until a few years ago, each state of the European Union supplied electricity to their respective citizens from a publicly owned company. In that system, it was clear that a switch from a monopolist environment to a regime of full competition could not be made overnight. It was decided to ease the costumers' entry into the free market. The approximate times for such entries were indicated in paragraph 19 of the previously mentioned directive.

On March 16, 1999, the President of the Italian Republic emanated a decree to execute the suggestions of the directive. He also gave the following program for the entry into the market[2]:

- Beginning January 1, 2000, eligible costumers were considered companies with electrical consumption greater than 20GWh per year
- Beginning January 1, 2002, eligible costumers were considered companies with electrical consumption greater than 9GWh per year
- Beginning July 1, 2004, all companies were considered eligible
- Beginning July 1, 2007, household consumers will also be able to access the free market in their homes

1.2 The Needs of Customers

The first realization of the directive, as the legislators had hoped, was the appearance of many suppliers in the market. Each of these companies proposed more than one kind of contract to customers. The competition among suppliers made it necessary to meet the needs of the costumers with different contractual options. These contracts were, and still are, very different from one to another. The format and the contents of the new contracts are still developing and it is difficult for a customer to determine which of the contracts best fit their needs.

In the lasts few years, a new type of energy consumption counter has appeared on the market and this device has substituted the pre-existing counters. These new counters register the electrical consumption, as well as other interesting information every quarter of an hour. It then sends all the data to the power company. A copy of the data is available for the customer and could be used to help find the best contract

for his or her needs. One data point every quarter of an hour means that there are 35,040 values for a single counter and for each characteristic (such as active absorbed power) per year. Considering the fact that there are a dozen contracts available, and that every contract can potentially define a different price of the electricity for every minute of the year, it is easy to understand the difficulties in choosing the most cost-effective contract.

The ACC database was born to help buyers choose the best contract for their needs. It intelligently and efficiently uses all the information provided by the consumption and contracts and analyzes the data to help the buyer make the best choice. Actually, the database is built to manage the consumption of companies. However, with the future global liberation of the market, the database will also be able to manage the consumptions for the household consumers.

1.2.1 The Consumptions Analysis

The analysis of consumptions is very important for a customer for many reasons. If the buyer is a company the importance of the analysis increases sensibly its value. This happens because of the magnitude and the nature of such consumptions. Actually, the analysis tells where and when the consumption is made and this information can be used to find out if and where the company is inefficient. This could happen if there is a broken or not appropriately used appliance, and it would be possible for the managers to discover such problems and resolve them. Another thing that the analysis can identify is if the company is loosing money because a high value of the reactive inductive component of the consumption. There are in fact penalties for such undertakings with a low power factor even if it is not difficult or expensive to limit its value. One other use of the consumptions analysis is the prediction of the consumption that will take place in the future months. This prediction is a matter of vital importance, because many contracts give economic advantages to those companies that can determine in advance how much they are going to consume.

1.2.2 The Contracts Management

Another aspect that the database keeps in account is contracts' management. Contracts' management is important because it helps companies choose the best contract for their needs. "Management", shall be defined as the storage and the use of all the information contained in a contract. This includes contracts even if they are no longer in use or no longer offered by the distribution company. Among the functions that must be performed, there is the need for calculating how much a company has historically spent and how much it is spending right now. It is essential to determine the advantages and the disadvantages provided by another contract and the management of deadlines and payment would be very helpful. The last, and most complicated function that must be performed is the capability to handle all the possible contracts with their various features. This is not easy because some characteristics are discordant and it is very difficult to find a homogeneous way to handle everything.

1.3 The Need of Permissions

The database developed for management of the electrical consumptions will be available on the web. This means that all the data will be saved onto a system and any user could potentially insert or modify information. Information is not of public domain and must be protected from malicious people and unauthorized users. It is also possible that some users may unintentionally damage information in the database. This is mainly due to human error and it is highly probable to occur when there are new operators that are just learning how to use the database. Permissions were introduced to preserve the integrity of the database from such sources of problems. Permissions are mainly used to accomplish the task of restricting the access of certain users to the database. The details of one possible implementation of permissions will be given in Chapter 5.

Chapter 2

Background Information

2.1 Introduction

This chapter will introduce very basic background information on databases and on operating system theory. The mathematical reasoning behind the theory will be limited as much as possible. It must be said that there is nothing new or innovative in the following paragraphs. A reader with a little of experience with databases and operating systems can safely skip reading this chapter.

2.2 Databases: Basic Information

A database is a collection of data, structured and correlated, managed by a DBMS (Database Management System). It represents a certain aspect of the real world, usually called mini-world or reality of interest. A database is essentially a collection of coherent data with a certain intrinsic meaning.

2.2.1 The relational model

Usually, the relational model is used for the representation of the data in a database. This model is based upon the mathematical concept of relation. A relation has the very important characteristic of being independent from the physical structure of the data. This independence is very important, because with this model, it is possible to develop a database without worrying about which DBMS to use. Another important feature is that the logical links are represented with values. It is a "model based upon values"! The relational model has a unique theoretical structure for the organization and the representation of the information. This theoretical structure as many can imagine is the relation. The relation has a formal definition based upon the mathematical theory of sets. The details of this theory are very interesting, but for the sake of the simplicity, a graphical definition of the relation will be given here. A relation can be implemented by a table made with a fixed number of columns (Attributes), with values from a finite set, and a variable number of rows (Tuples). Figure 2.1 shows clearly the details of this implementation.

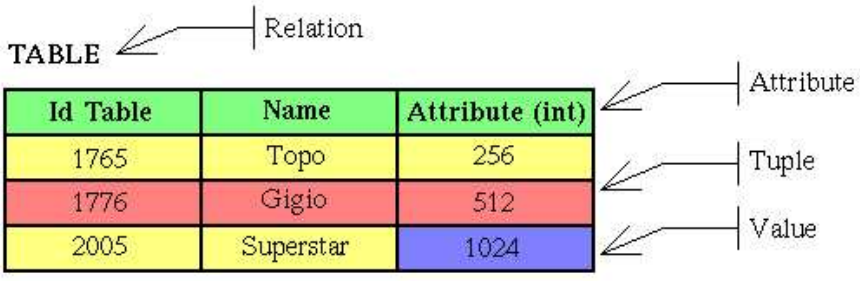

Figure 2.1: Relation seen as a Table

2.2.1.1 Types of Keys

There is another aspect of the theory of relations that must be explained. It is the concept of a key and a primary key. But first, the concept of super-key must be given. A super-key of a relation is a set of attributes that makes it possible to identify, in a unambiguous manner, the tuples of the relation itself. A key is a minimal super-key. This means that if one of the attributes of the key is eliminated, there is no longer the possibility to identify, in a unambiguous manner, the tuples of the relation. An eligible key is a representative of the set of groups of attributes that could be a key. Finally, a primary key is an eligible key that was chosen to identify the tuples of the relation.

2.2.2 Normal Forms

The normal forms are rules for the organization of the data in the relational model that must be verified in order to obtain favourable characteristics and avoid problems in queries and other operations. A relational model is said to be in the "Pippo" Normal Form if it matches all the rules of the "Pippo" Normal Form. These forms have a well defined rank and they must be verified in sequence. This means that it is not possible to have a relational model in the Second Normal Form (2NF) if it is not already in the First Normal Form (1NF). With the exception for the 1NF, it is not necessary for a relational model to satisfy any normal form. It would be better, but it is the users' choice. The only thing that is important is that the user knows what he is doing and the consequences of his choices.

A user can define an arbitrary number of normal forms. The most commonly used are the 1NF, the 2NF and the 3NF. They where introduced by Codd in the 1972 but still used today as the Boice-Codd Normal Form (BCNF) which is an extension of the 3NF.

2.2.2.1 First Normal Form (1NF)

The First Normal Form is considered an integral part of the formal definition of a basic relational model at the current time. The form was historically defined to avoid the use of multivalued attributes, composite attributes or a combination of both them.

The First Normal Form requires that the domain of an attribute includes only *atomic values* (or simple, indivisible values) and that the value of every attribute in a tuple be a *single value* in the domain of the attribute. Therefore, the 1NF does not contain a set of values, a tuple of values, or a combination of both as value for a single tuple. In other words, "relation inside relations" and "relation with attributes of tuples" are not allowed.

2.2.2.2 Second Normal Form (2NF)

To explain what Second and Third normal forms want to achieve, the concept of Functional Dependence (FD) must first be introduced. In a relation, a set of attributes, Y, depends from a set of attributes, X, (in symbols $X \rightarrow Y$) if for every couple of tuple $t_1$and $t_2$where $t_1[X] = t_2[X]$, it is $t_1[Y] = t_2[Y]$. An FD is considered "complete" if the removal of an attribute of X has as result, the loss of validity of the dependence itself. One can easily observe that the previous condition is always verified if X is a primary key.

A relational model, R, is in 2NF if every attribute not-prime A of R functionally depends in a complete manner from the primary key of R. If a model is not in 2NF it can be "normalized in 2NF", splitting it in a certain number of relations 2NF. In every one of these relations the not-prime attributes are associated to the subset of the primary key from which they depend in a complete way.

2.2.2.3 Third Normal Form (3NF)

The Third Normal Form is based upon the concept of Transitive Dependence. A Functional Dependence is said to be transitive if a set of attributes Z exists, and Z is not a candidate key or a subset of a key, for which are valid at the same time $X \rightarrow Z$ and $Z \rightarrow Y$.

A relational model, R is said to be in 3NF if it satisfies the 2NF and there no attributes not-prime of R depends in manner transitive from the primary key.

2.2.3 Entity-Relation model

The Entity-Relation is a model for the conceptual representation of data at a highly abstract level. The Entity-Relation model was proposed by P.P. Chen around the 1976. It is very intuitive, easy to understand, and it is very diffused in its usage for the conceptual planning of databases. One of its particulars is that it furnishes a set of constructs and ties suitable for describing the reality of interest in a natural and easy way. The most important characteristic of this model is that it consents to abstract from the particular, logical and physical organization of the data required by the DBMS. So it can be used without worrying about the implementation details and allows the developer to focus globally on the problem.

2.3 Operating Systems: Background Information

This paragraph will explain only a few subjects of the vast theory of operating systems. An operating system is software that manages the resources of a computer and makes the computer work. It manages the hardware, the memory and furnishes the proper resources to the processes that need them. It is evident that the topic is very vast, but here there is no need for explaining every aspect of an operating system. One of the features of an operating system that is important for database management is the ability to render the access to a certain resource in a mutually exclusive way. This is necessary because a database usually serves several users at the same time. This implies that every user can perform any operation he wants to perform and can use every resource available at any time. If two or more users perform a transaction[1] on the same tuple of a table at the same time (it is highly improbable, but still very frequent) the result of the transaction is not deterministic. This means that one or more user can lose the data, or worse, the database can lose consistency, which is really bad!

In operating systems there are semaphores that block access to some resources until the present task has finished using them. When a task requires resources that another tasks wants to use, the first task sets a semaphore on the resources, in order to have exclusive control of them. That way, no other task cannot access to the resources until the first task has finished using them and thus removed the semaphore. This is one of the approaches used in operating system to avoid deadlocks. In databases the semaphores are implemented by special instruction called LOCK and UNLOCK. Actually it is possible for a user to temporarily lock one or more tables and gain an exclusive access to them in order to perform the desired set of operations without risking the loss of data.

[1] In database jargon a transaction usually means a sequence of information exchange and related work (such as database updating) that is treated as a unit for the purposes of satisfying a request and for ensuring database integrity. For a transaction to be completed and database changes to made permanent, a transaction has to be completed in its entirety.

Chapter 3

A Versatile Approach in Database Realizations

3.1 Introduction

This chapter will present a versatile approach for planning and managing databases. A big effort has been made to obtain a level of flexibility that is unimaginable with a traditional approach to managing databases. The structure of the database is organized as an acyclic oriented graph. This is possible thanks to the introduction of a set of "auxiliary" tables and functions developed for this purpose. The functions mainly work on the auxiliaries table and make the management of the data structure stable and effective.

This approach can easily implement many-to-many relationships using the arcs table instead of auxiliaries tables rendering the structure very flexible. It is thus possible to create such relationship at any time and without any negative side effects or loss of performance.

The creation of the database happens in four steps. Each step is contained into a different script in order to keep them separate. The sequence of the script is as follows:

1. The first script drops the old database, if it exists, and create a new database that can support the PlPgSQL and the PlPythonu procedural languages. It also calls in sequence the other scripts.
2. The second script creates the functions and the triggers used in the database.
3. The third script creates the tables and sets their characteristics.
4. The fourth script initializes the database inserting opportune values in the table of interest.

3.2 Core-Tables

3.2.1 Management Philosophy

Before describing the details of the core-tables, it is necessary to explain which improvements were considered from the beginning, and observe the consequences of such improvements.

The first improvement stems from the consideration that a DBMS manages much better numbers than strings and that it is easier to manage small numbers than large numbers even for a database. For this reason the primary key of every table in the database must be a numeric ID. This ID will be given automatically during the insertion phase from a specific function and it will be the smallest possible. In this manner even if after some deletion there will "holes" in the sequence of ID of a table, these "holes" will be soon taken after a sufficient number of insertions, because the new inserted rows will have ID given in order to restore the natural sequence of the IDs.

The second improvement consists in associating an ID to every table name and giving even this ID in a manner transparent from the user. The information upon the association "Table↔ID" is stored into a special table named entity2id and is not available to the user.

To avoid ambiguity, an entire table will be referred as “ENTITY” and a tuple of a table will be referred as “INSTANCE”. Both Entity-ID and Instance-ID are strictly positive. This is because a negative number has the first bit set to 1, and this would eliminate the advantages of using small numbers. The value zero is avoided because it is reserved for service purposes.

3.2.2 Entity2id Table

In this case the name says everything. The Entity2id Table is in fact the table depositary of the relation that binds the name of a table to its ID. At the moment, the ID is inserted by the database administrator, but it could be given automatically thereby simplifying the job of the database manager. This is not a big deal, because even a big database has no more than two or three dozen tables. Figure 3.1 shows an example of the table entity2id and its contents. The extreme simplicity of the table entity2id must be emphasized. The table has only two fields, but this is normal for core-tables, because the new approach is meant to be the simplest possible.

entity2id	
id	**entity**
1	users
2	groups
3	arcs
4	events

Figure 3.1: Entity2id Table

3.2.3 Free_id Table

This table keeps memory of any free ID for any table in the database. Since the possible IDs are infinite, it is evident that it is not possible to keep memory for them all. Only the disposable IDs less than the bigger ID actually in use (ID*), and the next ID (ID*+1) are saved in the table. These IDs are to be used from the lowest. To identify the table it is used the same ID saved in the entity2id table. The insertion of the IDs in the free_id table is performed by the “use_free_id_for_table” function 3.3.2 when a new row is inserted in a table, and by the “delete_row_from_table” trigger 3.3.5.2 when a row is removed from a table. Figure 3.2 on the facing page shows an example of the content of the free_id table for the table groups. The tuple highlighted is the first that will be used, so it is possible to say that at the next insertion into the groups table, the new instance will have ID=2.

3.2.4 Arcs Table

This is the most important table in this new technology of building databases. This table makes it possible to transform a set of tables in a real graph. Just as the name suggests, in this table are saved the arcs that link the various instances of the database. To tell the truth, four numeric values are memorized in the arcs table that give the starting point and the ending point of any arc connecting two instances. The implemented graph structure is oriented, so the first two values are the *“PSEUDO-CARTESIAN”* coordinates of the tail of the *“arrow”* that represent the arc, while the other two values identify the point of the arrow.

groups	
id	name
1	pacinotti
3	valpolicella
6	topo gigio
7	superstar

free_id	
entity_id	instance_id
2	5
2	4
2	8
2	2

Figure 3.2: Example of the Free_id table for the Groups table

There is one CONSTRAINT in the insertion of an arc. It is not possible to insert an arc with the same starting and ending point. This is the first countermeasure in order to avoid the creation of cycles during the insertion phase.

Another limitation, very useful to be sure, is that the arcs table gives a truthful representation of reality, is that it is impossible to link non-existent instances. This limitation is realized firing a trigger during the insertion of a new row. Precisely in the subsection *"AFTER"* of such trigger (for more details see 3.3.5.3).

The last limitation, the one that avoids the creation of a cycle in the structure, is also realized with the execution of a trigger, precisely in the subsection *"AFTER"* of such trigger (for more details see 3.3.5.3).

Figure 3.3 on the next page shows as the table arcs works.

3.2.5 Last_used_id Table

This table keeps memory of the last ID assigned to the last instance inserted into a table. The use of such a table is necessary, because every insertion is regulated by a trigger function and such function cannot return any value. Since the ID is assigned automatically, this is the only way to inform the user about the value of the ID. One could say that it is possible to obtain the ID with an opportune query in the table, but this is only possible if the data inserted is an eligible key of the table, which is not always true.

The last_used_id table contains a row for each table of the database with a entity_ID in the table entity2id. The row of interest can only be updated by the trigger and there must be a value for the start of each row. The value chosen is zero, so each ID of this table will be set to this value during the creation

arcs			
src_entity_id	src_instance_id	dst_entity_id	dst_instance_id
2	1	2	2
2	1	1	1

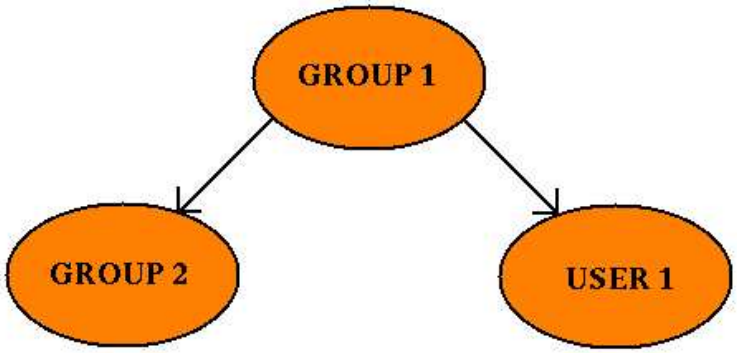

Figure 3.3: Example of the use of the arcs table

of a row, namely during the database start-up process.

3.3 Functions

Each function used to form the *"KERNEL"* of this technology has been written in PlPgSQL, a Procedural Language that uses the basic structures of a high-level programming language (Conditions, Cycles and Variables) and queries written in SQL.

This solution is probably not the most efficient in terms of speed of execution, but it is very convenient in the programming phase. It could eventually be possible to translate functions and procedure into the C programming language. Anyway, for a lack of time this translation is not going to be done in the immediate future.

3.3.1 Get_id_from_table_name Function

This is a function built to avoid the repetition of the query that extracts the ID from the table entity2id given the name of the entity. The function is not strictly necessary, but it is very useful while writing the other functions of the database.

3.3.2 Use_free_id_for_table Function

This is the function that extracts the ID from the table free_id and inserts the next one. The operation could seem very simple, but it is not so trite because of the presence of some particular case. First, if the insertion is the first, the table free_id does not contain any ID and in this case the function returns the value 1 and puts into the table the value 2. If at least one value is already present in the table, then the function returns the smallest one. If the smallest ID is also the biggest ID (there is only that ID), into the table free_id is inserted the value $ID_{NEW} = ID_{OLD} + 1$ and the old valued is eliminated. If there is more than one ID in the table, the smallest is returned and deleted from the table.

3.3.3 Relatives Function

This recursive function returns every ancestor or every descendant of a certain instance of the database and can go to any level of depth wished by the user. The function calls itself recursively until it reaches the the desired level of depth or until it reaches the last level available. It also returns any examined node

and its distance from the starting node. To determine the new nodes to visit the function “get_arcs” is invoked. Contrary to what its name suggests, “get_arcs” returns the nodes or better, the other end of the arc that has starting point (if the descendant are wanted) or ending point (if the ancestors are wanted) the node of interest.

The function returns the current node, if and only if, such node has relatives of the kind searched, otherwise nothing will be returned. This characteristic can be used to check if a node has ancestors or descendants.

The parameters to pass to the function are in order:

1. _entity_id: source type
2. _instance_id: source id
3. _family: types of destination entity to return
   - 0= every family
   - n= family with n as entity_id in the the table entity2id
4. _recurs_lev: desired recursion level
   - 0: only this node;
   - -1: recursion infinite;
   - > 0: n recursions;
   - < -1: Invalid entry
5. _actual_lev: recursion level counter
6. _kind_of_relatives: type of destination arcs to return:
   - 0=parents
   - 1=children

3.3.4 Get_arcs Function

This function returns each node that has arcs entering into or getting out of the node of interest. The direction of the arcs depends on the parameters passed to the function.

The parameters are:

1. _entity_id: source entity id
2. _instance_id: source tuple id
3. _family: types of destination entity to return
   - 0= every family
   - n= family with n as entity_id in the the table entity2id
4. _kind_of_relatives: type of destination arcs to return:
   - 0=parents
   - 1=children

3.3.5 Trigger Functions

Triggers are functions that can be called "fired" when operations such as insertion, deletion, or updating are performed on certain tables of the database. The main purpose of triggers is to render the execution of such operations mutually exclusive. In a well determined moment, only a trigger with a certain name can be fired on a table . Therefore, if two users want to perform an insertion on the same table at the same time, only the one who fires the trigger first will be able to accomplish its purpose. The other will have to wait until the first user is done with his insertion.

In the case of an insertion, it is possible to decide "WHEN" to fire the trigger. If a piece of code such as a control on the data or a modification must be executed before the insertion, there are two possible ways to perform this operation. The first is to create a trigger to fire before the insertion with its own function. The second method is to include the piece of code into an "IF" statement executed if the "WHEN" of the trigger is "BEFORE". If there is the necessity to perform some controls after the insertion, the same operations with the only difference of substituting"BEFORE" with "AFTER" must be performed.

An operation performed with a trigger will be executive only if the called function returns a object of type trigger. This is necessary to be sure that all the desired operations have been performed and to avoid errors. If such a trigger is not returned on the database a ROLLBACK operation is performed and everything returns to the previous state.

An example of SQL code for the creation of some triggers for the table "table" is the following:

```
CREATE TRIGGER table_insert_b BEFORE INSERT ON table
FOR EACH ROW EXECUTE PROCEDURE proc_1();
CREATE TRIGGER table_insert_a AFTER   INSERT ON table
FOR EACH ROW EXECUTE PROCEDURE proc_2();
CREATE TRIGGER table_delete_a AFTER  DELETE ON table
FOR EACH ROW EXECUTE PROCEDURE proc_3();
```

3.3.5.1 Insert_row_into_table Trigger

This function is used to insert rows in almost all the tables of the database. The mechanism of insertion is very simple and it is handled partially *"BEFORE"* and partially *"AFTER"* the insertion itself.

In the *"BEFORE"* subsection the new row is inserted with ID=0. If an error occurs during this operation it means that a constraint has been violated and everything returns to the previous state because the rollback operation is performed automatically by the database.

If no errors occur during the execution of the *"BEFORE"* subsection of the trigger, then the *"AFTER"* subsection is called. In this subsection the function "use_free_id_for_table" is called , and the returned value is assigned as new ID for the newly inserted row. A LOCK operation is then executed on the last_free_id table and the row of interest is updated with the new ID. The last_free_id table remains LOCKED until the end of the transaction in order to assure the user that the ID he will read is the one assigned to his row by the function use_free_id_for_table.

3.3.5.2 Delete_row_from_table Trigger

This function deletes the desired row from the desired table, inserts the ID of the just deleted row into the table free_id, and removes all the arcs entering or exiting from the newly eliminated element. This is necessary to maintain the table arcs consistent with the entire database.

3.3.5.3 Insert_row_into_arcs Trigger

"Before" Before inserting an arc, it is necessary to verify that the new arc is not going to generate a cycle. To be sure about this, it is sufficient verify that the destination instance of the arc (this node is going to be a descendant of source instance of the arc) is not an ancestor of the source instance of the arc. The control is executed asking to the function relative to return any ancestor of the source node of the arc and verifying that, among these, it is not present the destination node of the arc that is going to be inserted.

"After" This piece of code is executed only if the arc has been inserted, namely if the arc does not introduce cycles. Before rendering the insertion definitive, the existence of the source and destination instances must be verified. This task is accomplished by selecting the ID from the tables that should contain the instances of interest. Cursors are used to create and execute the query because they are more flexible than the PlPgSQL commands and they allow the programmer to pass the name of the table of interest as variable. The values into the variables are obtained with a query on the entity2id table. This is necessary because the insert_row_into_arcs function knows the IDs but not the name of the tables whose rows needs to be connected.

Note: In the *"BEFORE"* subsection one must verify that the arc is conceptually admissible, namely that it does not render the graph cyclic. In the *"AFTER"* subsection it is verified that the arc is coherent with reality, namely that the structure represented by the arcs table have a correspondent into the database. The controls are divided between before and after to let the CONSTRAINT of the table do its job and to be sure that it does not perform such controls twice.

Chapter 4

The ACC Database

4.1 Introduction

The ACC is a database built to manage contracts for electrical supply. The database is based on the new technology explained in chapter 3. For this reason it is very flexible and suitable for the management of electrical contracts. This new technology could be used in many other applications especially where a high level of flexibility is needed and where many-to-many relationship are necessary.

This chapter will give a description of the database structure. The details of the management will be only furnished in a superficial way. The important aspects of the management will be given in the next few chapters.

4.2 The Purpose of the ACC Database

With the arrival of the free market in the electricity supply field, a large amount of freedom for many subjects has also arrived. The freedom to sell electricity for the suppliers and to decide the terms of the contracts. The liberty to buy electrical energy from any supplier for the companies, at the present time, and for everyone in the near future. Just as it often happens for any big change, one has a bit of difficulty in adapting himself to a new environment. Usually the stronger are the first to exploit the new situation and to obtain the greater profit, often to the detriment of the weaker. In this particular case, the weak entities are the small or medium-sized companies, that have no tools or enough knowledge to help themselves in the choosing the best contract for their needs. Fortunately, many of these companies have decided to adhere to some consortiums, in order to utilize the strength of the numbers. Unfortunately, not all the components of a consortium behave the same way, and sometimes there are certain individual that damage the whole set of companies because of their bad behavior.

The ACC database wants to be an easy-to-use tool that can help companies choose a good contract for their electricity supply. It wants to help the consortiums choose a contract too, but also be able to provide all the necessary information to analyze whether or not a certain company is damaging the consortium.

The last purpose of the ACC database is to be available on the web so that even a company with only an obsolete machine and a dial-up Internet connection will be able to use all the database's capabilities and all the analysis programs that can work with it.

4.3 Management Problems

The problem for the management of the electrical contracts and consumption are multifarious.

4.3.1 The Need for Permissions

The first problem comes from the choice to make the database available on the web, via a simple tool such us a web browser. There will hopefully be several users for the database,and each of them will have his own data and will want to keep his information totally or partially secret. There will also be

"occasional" users whose purpose could be to evaluate the possibility to use the database and become customers of it. There could also be users that have to be allowed only to read data but without the privileges to write into any table. For all these possibilities and many others that could become important in the future, there is the necessity of managing permission for users and groups of users. These permissions must be powerful and simple to use.

4.3.2 Handling Many Different Contracts and the Need to Save Historical Information

The second problem for the management comes from the number of possible contracts. There are many electricity suppliers, and they propose at least two or three contracts each. It is evident that each contract will have different characteristics from the other contracts. Even the procedure to determine the price for the electrical energy vary from one contract to another. Actually, there are contracts whose prices vary during the day according to the electricity market, and this means that there could be a different price for every hour of the year. On the other hand, there are contracts that define one price for the whole year, and this price depends on the date of stipulation of the contracts themselves. Between these two extremes, there are infinite possibilities and the database must be able to handle every possible condition in a unique way.

Another thing that must not be missing from the database is the possibility for a company to have all the historical data of its consumptions and contracts. This is very useful during the phase of determining if a new contract is better than the old one (maybe is time for renew it) and also to have an idea about the electricity expenditures of the company through the years. And how is it possible to determine if someone is saving money this year compared to last year if there is no data to deal with? Only saving the important data for a sufficient amount of time will guarantee that the users will have all the information they may need.

4.3.3 Supporting Analysis Tools is Essential

Many suppliers define contractually that a buyer must request in advance the amount of electrical energy it is going to use in the next week, month, or even quarter-year. If the buyer asks for a certain amount of energy, but then uses a quantity differing from the declared one, he has to pay more for his electricity. This means that there is the need for tools to help the customers in the analyzing consumptions. There must be a predictor of consumption that is capable of saying, within a certain margin of approximation, the amount of the consumptions that are going to take place. There is also the need to define given set of parameters calculated starting from the consumptions. These parameters have to make it possible to distinguish if a costumer is a good buyer, namely that it is easy to predict its consumption.

It must also be possible to analyze other characteristics of the consumptions, such as power factor and others, in order to define if a buyer is a good costumer under the suppliers' point of view. It is easy to imagine that a customer will receive some *"rewards"* from a provider if he has a good loading curve.

It is obvious that these tools cannot be implemented within a database, but the database must be able to give all the possibly necessary information to make these tools operate at their best.

4.4 Application Tables

4.4.1 description

Table to be inherited by other tables requiring a given set of description attributes.

4.4.2 users

Table containing the users of the database and their passwords.

4.4.3 groups

This table defines the names of groups of users. The groups are useful if it is necessary to assign the same characteristic to more than one user. For example every user member of the guest group will have

restricted access to certain features and tables of the database.

4.4.4 events

Table containing the timestamp fore every event that may occur to any instance of the database. If an event is associated to a databases instance this table answers only to the question “When did/does the event happen?”.

4.4.5 events_type

This table gives more information concerning the events happening to some instances of the database. The purpose of the table is to determine the kind of events that takes place. The separation between “WHEN” and “WHAT” happens is made, because it is possible to have two or more events of the same type occurring in different moments or two different events happening at the same time. This distinction allows the database management to save space and to group the information in a lower number of tuples.

4.4.6 events_priority

There are events that are more crucial than others and must be considered quicker. This table holds the information regarding the priority of certain events. It is possible to assign a default value of priority to such events that do not have a specified priority and save space in the database. Also, in this case, the separation of information is utilized to avoid blank spaces and repetitions in the event table.

4.4.7 intervals

Table containing intervals that will mainly be used for joining the electricity prices to different periods of the year. This table will be also used any other time an interval will be required.

4.4.8 counters

This table holds technical information of every counter considered for the electrical consumption management.

4.4.9 manufacturers

Table holding information of the counters’ makers. This is useful if there is any problem with a counter, because it makes it contacting the manufacturer in a short amount of time possible.

4.4.10 drawing_points

Usually the object of a contract for electricity supply is the physical point where the electrical energy is taken from the buyer. This information is held in this table. The separation between counters and drawing points is due to the different nature of the two things (counters are physical entities while drawing points are abstract entities) and because there could be more than one counter per drawing point.

4.4.11 companies

This table contains information about companies involved in the electricity market. This means that there will be a unique way to manage both providers and buyers. The choice to use only one table is made because the same information is required to describe these two categories of electricity stakeholders.

4.4.12 consumptions

Only the information concerning the type and the value of consumptions are held by this table. Other information, such as when a consumption took place, will be spread around the structure of the graph implemented by the database.

4.4.13 bills

Table necessary to manage the payment of bills and to keep records of past payments for the electricity supply on the buyer's side. Also, in this case, only the information concerning the economical aspect of the bills are saved in this table. The other information will be available in another location of the graph.

4.4.14 consortiums

This table contains useful information regarding consortiums of companies that want to buy electricity together and save money.

4.4.15 contracts

Basic information concerning the management of contracts are saved here. Other information such as the price of the electricity for different moments of the day will be held in different places of the database.

4.4.16 taxation

This table contains information of all the possible taxation defined in an electrical contract. Usually these taxation are due to late payments or a bad profile of electricity supply.

4.4.17 toponyms

Table containing information upon one geographic characteristic of a location. In this table there are only fields for the name and the type of toponyms. The complete information for a location will be synthesized linking different tuples of the toponyms table in order to unambiguously determine the desired location.

4.4.18 timeframes

In the ACC database, timeframes are periods of time with variable length. The length is a multiple of the val_type measure unit and this unit can be hours, days, weeks and so on. This table is useful in the phase of defining the rules for the consumption prices.

4.4.19 prices

This table defines the prices for the electricity for a certain contract in a well defined moment of the day and the year. In the table there is also the field value_type for compatibility with foreign countries and their currencies.

4.4.20 daily_bands

The starting moment and the ending moment for a certain consumption band in a day are saved in this table. The days are usually grouped by their type. A day classification that will be surely used could be working days or weekend days. This division of days will differ from one contract to another because of the different rules that every contract introduces.

4.4.21 permissions

This table contain basic information upon permissions. Permissions are used in the ACC database to restrict the users' possibilities and to guarantee that the behaviour of anyone user will not harm the integrity of the database or violate someone else's privacy.

4.4.22 perm_types

Table containing further information upon permission. Particularly the type of operation that a user is allowed to perform on certain areas of the database.

4.4.23 agents

This is one of the most important tables in the ACC database. The main purpose of this table is to give semanticity to the arcs. In other words, when an arc can have more than one significance, an instance of kind agent is inserted between the instances that have to be connected and the meaning of the link is saved as field of the agent itself. The agents have also a numeric field that can be used to allow some descendant of the link's destination to inherit the characteristic saved into the agent.

Chapter 5

Permissions Management

5.1 Introduction

This chapter is going to explain how to handle the permissions in the ACC database. There are several solutions to accomplish this task, and the one presented here is hopefully the best. The details of this solution will be shown with all the advantages and the flexibility that is involved.

First, one must consider some of the problems that the realization of a database induces into permissions management. We have a graph structure that doesn't allow us to introduce cycles, so the permissions go to a higher level of this structure. In other words, the permissions wouldn't have ancestors, and every database instance which will make use of the permissions will be a descendant of them. This implies the problem to decide if an instance pointed by a permission is the owner or the object of the previously mentioned permission. Another difficulty comes from the fact that we can connect only single instances of the database to each other, and we have no possibility to refer a permission to a entire table. We could, of course, connect the permission to every single element of the table, but this doesn't solve the problem to give someone the ability to create a new element in that table. A further obstacle is how deep in the subgraph of the referred instance the permission should be applied. We could consider the whole subgraph, but what can we do if we want to apply a writing privilege only to a single element of the graph, but not to its maybe numerous descendants? Should we have a negative permission on each of its children? The last problem to take into account is the inheritance, but this issue is strictly related to the previous one and it isn't a necessity to make an example.

All these problems are easily resolved if taken one by one, but the solution that is going to be shown will solve the lot and will try to use the minimal amount of arcs.

First, the ACC database is a very flexible and powerful database. It allows the user to create relationships connecting the various instances at any moment and with only few limitations. There are essentially two limitations:

1. There isn't the possibility to create a cycle
2. The two instances connected by a arc must exist

To fully use all this feature it was decided to make every table the smallest possible, to split all the similar characteristics into other tables, and to connect the main table with the table containing the characteristics. For this reason, the permissions table has to be considered only as a label and the information is held by the other tables and by the arcs connecting these tables. A graphical demonstration on how this method works can be seen in Figure 5.1 on the following page.

With this schematic it is not clear how to define which instance is the owner of the permission and which is the object, but this topic will be discussed later on. The thing that must be pointed out here is that permissions don't have ancestors and that if one needs to add other characteristics, it is very easy to accomplish this task.

5.2 Type of Permissions

Permissions are mainly divided into two categories:

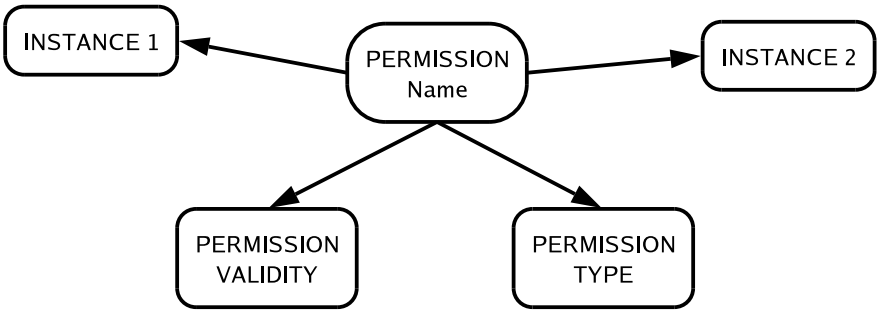

Figure 5.1: Permissions Management

1. *"Positive"* which means that the information indicates the operations that a certain instance is allowed to do.
2. *"Negative"* which means that the information indicates the operations that a certain instance cannot perform.

The number of permissions applicable to this database is finite, so we can consider the permissions as a set with a limited amount of elements. The positive and negative permissions will be a subset of the main set. They will have for a single instance, a null intersection and the union of them will give the whole set. In other words, they will be two complementary subsets of the permissions set. This becomes very useful, because one doesn't need to use both the subsets to obtain the necessary information. It is sufficient to define the positive permissions that an instance possesses and the negative permissions can be deducted from these. It is evident that the previous considerations are valid, if and only if, the permissions are the simplest possible that is, the permissions will concern only elementary operations. This means that there will be permissions of read, write, but not read/write.

One must also consider that certain operations can't be performed unless other operations have been accomplished beforehand. For example, one cannot delete an instance without knowing which information it contains. For the same reason, if someone wants to allow a user to write on a certain instance, that person has to give to the user the faculty to read also, and more importantly, this operation must be performed automatically. An example of some permissions and their logical relationships is shown in Figure 5.2.

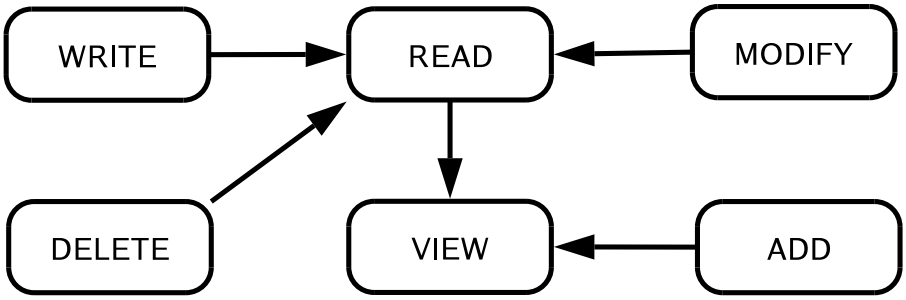

Figure 5.2: Relationships Among Elementary Permissions

These are not all the possible elementary permissions, but won't be restrictive for the explanation of the management method given here. Even with the introduction of new permissions, the methodology to handle the problem will remain the same.

5.3 Temporal Validity

The temporal validity of a permission will be considered infinite as long as there isn't any specification regarding this aspect. Otherwise, the usual convention will be used as the Figure 5.3 on the facing page shows.

Figure 5.3: Permissions' Temporal Validity

5.4 The Table Root

In the ACC database one can connect only single instances of a table to create a graph structure, but it is evident that full functionality for the permissions management is not realized. The root table is the solution chosen to solve the problem of the absence of the possibility to have arcs connecting entire tables of the database. Sometimes there is the necessity to allow some instances to create other instances in other tables. This is the case of a user that wants to create a new group. The new group will be, once it exists, an instance of the groups table. At that point, it will be easy to define some permissions upon it. But before the group exists, how is it possible to allow someone to create it?

One solution could be to have one permission to create an instance for every table of the database. This would increase the number of the elementary permissions and the number of arcs necessary to define few characteristics. If someone creates a new table, it is necessary to add an opportune instance to the permission_types table. Another problem that arises is that it is not possible to connect the permission with an object instance. This will create a subset of permissions with different peculiarity from the others. The permissions in this realization are designed to have a *"subject"*, which is the owner of the permission, and an *"object"*, on which the subject applies the permission. This implementation for the *"ADD"* property would create an absence of uniformity in the method used to manage the permissions and would increase the difficulties rendering the solution less elegant. Another complication that this solution introduces is that there isn't the possibility to allow someone to add an instance to a table and to modify every instance of the previously mentioned table in a single permission. Two or more permission instances will be required to obtain this result with an increase of the number of queries necessary to determine what can be done on the table.

In the solution chosen for this implementation there will be an instance of the table that will represent all the elements. This instance will be the father of every other element in the table. To apply a permission on an entire table, it will be sufficient to have the *"root"* as the object of the permission. Then there will be uniformity in the realization and the number of elements in the permission_types table will remain low. Another advantage of this artifice is that we can allow an instance to perform on the instances in another table even if these elements do not exist yet. Thus, there is not the necessity to apply a new permission every time a new item is created, thereby saving time and lowering the probability of an error for the programmer. The Figure 5.4 gives a crystal clear example of the potential of this implementation. With a unique permission there is the possibility to allow INSTANCE 1 to create instances in TABLE 2 and to modify all the elements contained in that table.

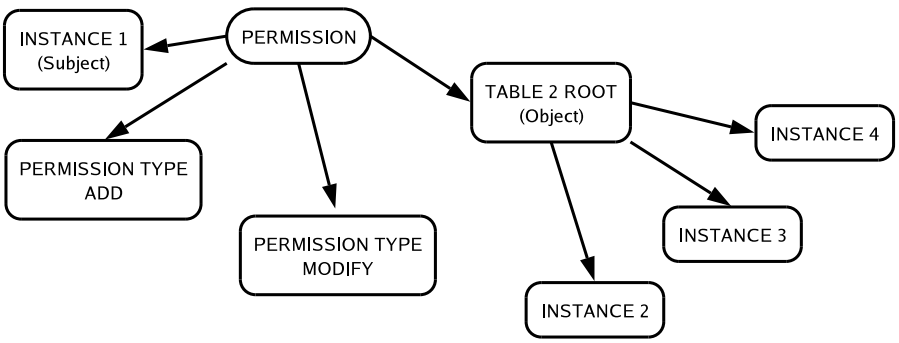

Figure 5.4: Permissions on an Entire Table

5.5 The Agents Table

All the examples discussed in this subsection are based upon the permissions assignment described in the subsections 5.6 on page 33. It is advised to take another look at the examples after reading this subsection if they do not seem very clear.

The problem of the permission owner and object identification is solved in this implementation by the introduction of an agent entity. This new entity has, as side effects, other advantages. Among the agent fields there are the ID which allows the use of arcs with this entity and the ag_type field to define the type of agent. The latter field is not only applied to the permissions problem, but the agents can be very useful in many other situations that occur in database management. Regarding the problem being dealt with here, the agents can be split into two categories:

- Subject which means the owner of the permission
- Object which means the instance upon which the permission is applied

With this definition of agents it is now clear how to define the owner and upon whom the permission is applied. This is indeed shown in Figure 5.5.

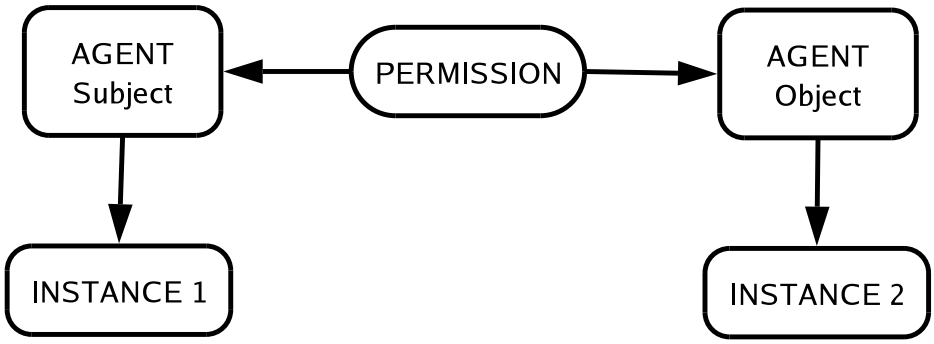

Figure 5.5: Subject and Object of a Permission

The agents utility is not only limited to this first aspect, but also in a numeric field that indicates at which depth the agent is applied. The insertion of this field aims to handle the permission inheritance and the permission application. It also has many benefits such as rendering the permissions themselves extremely compact and flexible. Before showing this methodology of management, the other solutions that were considered are shown here to give a complete view of the problem.

When one wants to define a permission upon an instance, one must first decide who owns the permission and upon whom the permission applies itself. These two matters are not as simple as the may seem because a universal standard for the permission inheritance and permission application must be defined. It is possible to think about three methods to resolve this matter.

The first method is to agree that the permission must be applied only upon the instances that are actually connected to the agents. This is the simplest and safest way to resolve the matter in question, but it is also the least powerful method. If one wants, for example, to apply a permission over an entire subgraph of the object instance, there is the need of an arc connecting the agent and every nodes' descendant. If the instance has ten-thousand descendants, it is necessary to use as many arcs as the number of descendants. The matter can be posed also to the owner's side of the permission. In that case, there is the necessity to make an entire subgraph of the permission's proprietary. To get an idea of the number of arcs necessary, a simple graph with ten instances is presented in Figure 5.6 on the facing page. It does not go unnoticed that the number of arcs raises because the information held by the main graph structure is totally ignored. It is also evident that a waste of information causes inefficiencies and that every inefficiency must be avoided.

The second method makes the whole subgraph of an instance aware of the instance's permissions. It uses as much information contained into the main graph structure as possible. This solutions is extremely useful in the case that one wants to apply a permission upon an whole subgraph, but must be used with a certain care when there is the need of some exceptions on the permission's application. If one wants to apply a certain condition only to the first level of descendants, one will need to create an

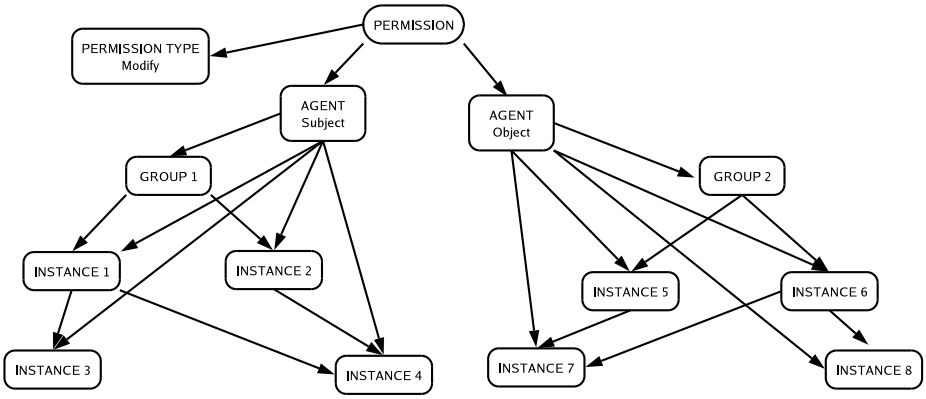

Figure 5.6: 1-Depth Permissions

adequate permission and apply this to the second level of descendants. If one wants to give greater power to one of the children of the permission's owner there will also be the need of a suitable permission for the child and some of the arcs to enforce the main permission to the grandchildren.

The Figure 5.7 gives some examples concerning these special cases.

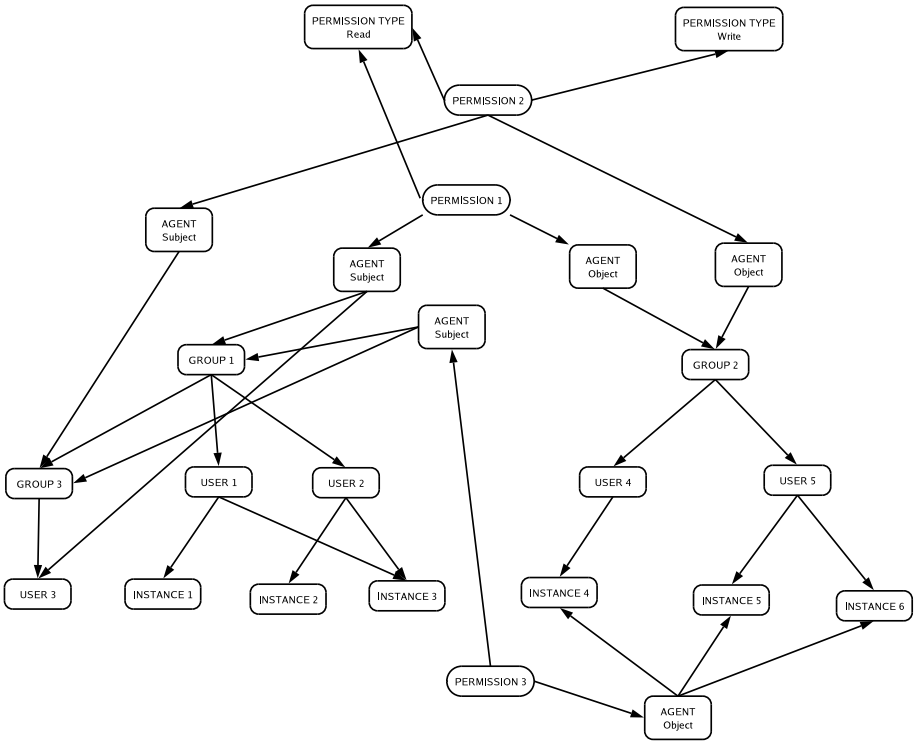

Figure 5.7: Infinite Depth Permissions

PERMISSION 1 is of Read type and is owned by all the GROUP 1's subgraph with the exception of GROUP 3. GROUP 3 has a permission that gives it the possibility to write also. USER 3, is the son of GROUP 3, but does not possess his father's permissions because he inherits his grandfather's permissions. The permissions on the side of the objects are applied to GROUP 2 and to its descendant of first level. All of other descendants GROUP 2 are the object of a null permission owned by all of GROUP 1's subgraph and are invisible and untouchable to all those instances.

The other two management methods are based upon a hybrid use of the previously mentioned methodologies. It is possible to have a unique subject and a entire subgraph as object or a subgraph as owner and a single element as object. The usefulness and the complications of this *modus operandi*

would be the *"weighed"* average of what was seen above. It is a final user's task to evaluate the pros and cons of each and to choose the one that permits the best use of the available resources.

And now let's show the approach devised for the permissions management in the ACC database. This method can be considered a generalization of the previous ones and consequently results in a very effective and versatile way. This technique's greatest novelty is to associate a numeric field to the agent. This field indicates the geometric validity of the permission in the sense of path length. In simpler words, the permission remains valid only for the descendant of the entity connected to the agent until the indicated depth. The convention used for the depth is the same that was used to pass the parameters to the parents function:

- n = -1: infinite depth
- n = 0: only the instance pointed from the agent
- n > 0: all the descendant of the instance pointed from the agent until the n-th level of depth

With this convention, the previous implementation can be obtained by using n=0 or n=-1 in a opportune way . The advantages which this method introduces are basically the same as the above realizations, plus the possibility to use an optimized combination of these realizations in order to avoid all of the drawbacks. In certain situations, for example, it is better to use an infinite inheritance, while in certain others, it is preferable to have a single inheritance. Now it will be possible to apply both approaches on the same instance if there is the need. Another powerful characteristic that the user can appreciate is the possibility to apply a certain permission only for few levels of descendants and to exclude all of the other levels. This peculiarity combines itself very well with the use of the table root. If there is, in actuality, the necessity to be able to modify all the instances of a table, it will be sufficient to have a permission on the table root and to impose n=1. This makes it possible to have the permission upon all the instances and also on elements to be created later. Also the task of allowing someone to visualize only certain levels of the graph structure is extremely easy to accomplish now with the lower expenditure of resources!

Only these few examples can make everyone aware of the potential and of the versatility inherent in this implementation. Figure 5.8 shows how the use of the *"Customizable Depth Permissions"* can simplify also the realization of the schematic in Figure 5.7 on the page before.

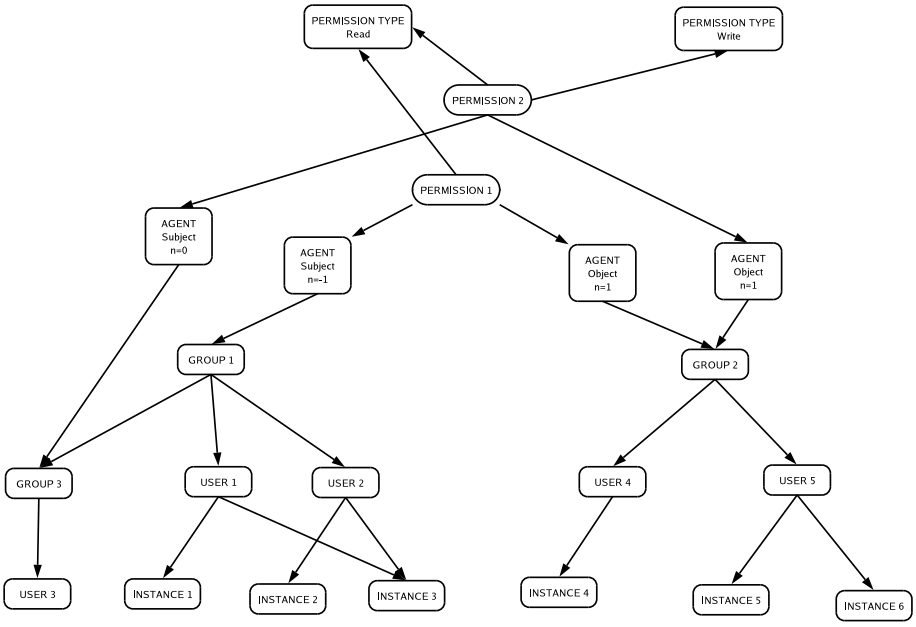

Figure 5.8: Customizable Depth Permissions

It is easy to notice that the number of instances in the table permissions is decreased from three to two. There are also fewer arcs needed. They are passed from thirty-two to twenty-four with a

decrement of about the twenty-five percent. Likewise, the agents are reduced too. There are now only four compared to the six of the previous implementation.

5.6 How to Determine an Instance's Permission

The method to determine which permissions a instance posses is very simple. The closest permission to the considered node is the one that applies. There are anyway some things to consider to be able to apply this method. Mainly there is to verify that the permission is valid and an algorithm to find the closest permission must be introduced.

5.6.1 Valid permissions

The first step to determine an instance's permission, is to select only the permissions among all the ancestors of the starting node (*"A"*), which have influence upon A. To perform this choice, one path at the time must be checked. Once the path has been selected for the analysis there are three instances that have to be highlighted:

- the instance "A" itself
- the agent that is in the path and that will be called *"B"*
- the child of B, called "C" which is the instance for which the permission has been created

To clarify this concept let's take a look at the Figure 5.9 on the following page and suppose one wants to determine the permissions that apply upon INSTANCE 7.

In Figure 5.10 on page 35 the paths from INSTANCE 7 and the various permissions have been extracted from the original graph and a label has been posted on the interesting nodes.

Using these figures it is easier to understand the concept of valid permissions. The word valid implies one of the following things:

- The numeric field in the node B is greater or equal than the distance between C and A
- The numeric field in the node B is equal to -1

Once the valid permissions have been found, one must designate the right agent type. If one is searching for the permission owned by INSTANCE 7 the paths to consider are only those containing a Subject agent.

5.6.2 Determination of Permissions

The most common case of determination is when one wants to find out which permissions the instance *"A"* has upon the instance *"B"*. This decision is made choosing among the valid permissions with the right agent based on values l and f.

- l is the distance, defined in number of arcs, from the permission to A following a path with a Subject agent
- f is the distance, defined in number of arcs, from the permission to B following a path with an Object agent

Among the permissions only those with the minimum value of l will be selected and from this set those with the minimum value of f will be chosen. If there aren't any permissions left, then it is possible to say that the first instance has no allowed operations upon the second one. If there are one or more permissions left, then the allowed operation will be the union of the sets of all these permissions' types[1]. In Figure 5.11 on page 35 is shown a typical graph structure.

[1]This operation is possible because in the structure are considered only the positive permission. If it was considered also the negative permission it wouldn't be so easy to determine the allowed operations.

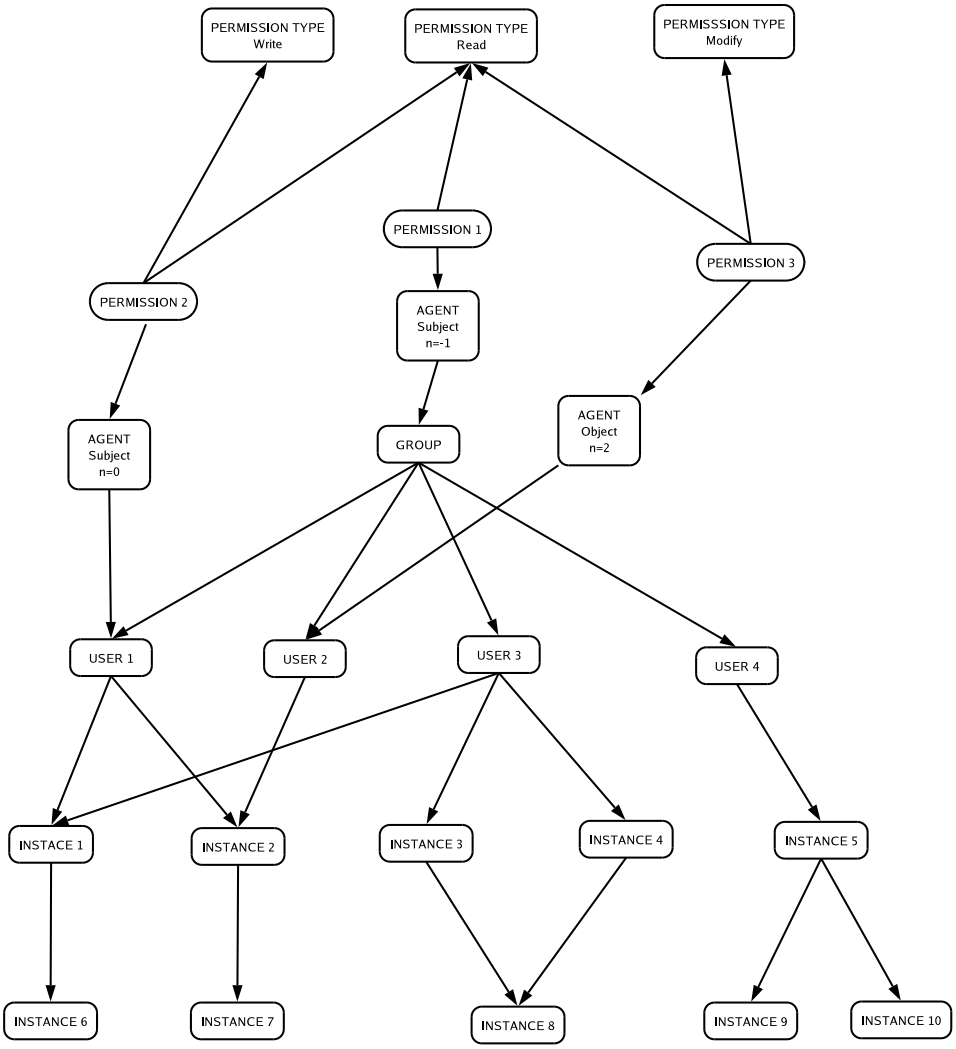

Figure 5.9: Permission Validity

Suppose that it is necessary to calculate the permissions that USER 1 has upon INSTANCE 3. All the permissions result valid. There is a subject agent in the path between every permission and node A and an object agent in the path between every permission and node B. It is possible to say that the study must concern every permission.

Let's draw up the schematics of the situation of the permissions including the values l and f:

PERMISSION	l VALUE	f VALUE	PERMISSION TYPE
PERMISSION 1	3	2	
PERMISSION 2	2	4	Read
PERMISSION 3	2	3	Read, Write
PERMISSION 4	3	5	Read, Write, Modify

After the permissions with minimum value of l has been selected the situation can be charted as follows:

PERMISSION	l VALUE	l VALUE	PERMISSION TYPE
PERMISSION 2	2	4	Read
PERMISSION 3	2	3	Read, Write

Choosing now the permissions with the minimum value of f, the applicable permissions remain:

PERMISSION	l VALUE	l VALUE	PERMISSION TYPE
PERMISSION 3	2	3	Read, Write

At this point it is clear that USER 1 has a permission for Read and Write upon INSTANCE 3.

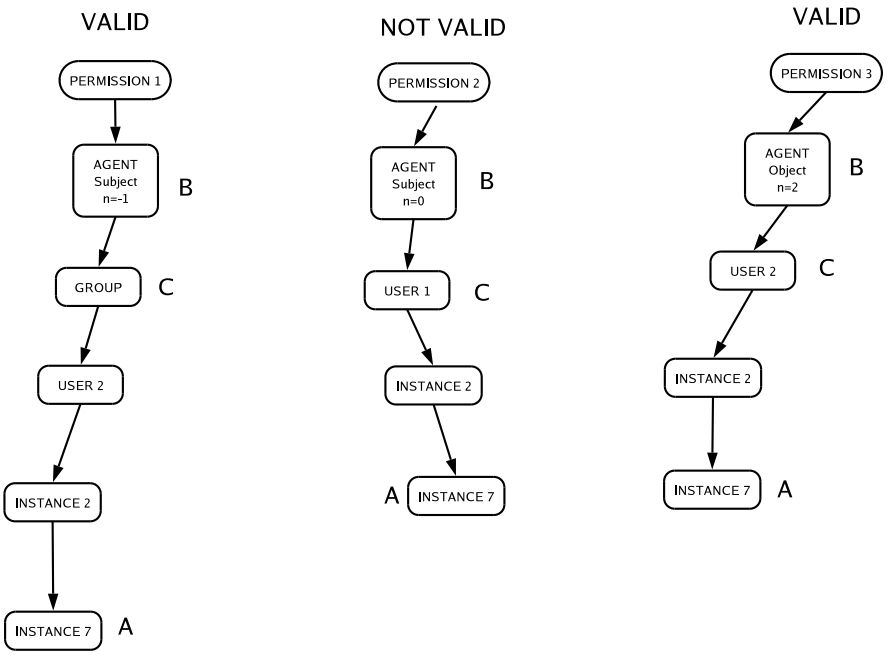

Figure 5.10: Highlighted Permission Paths

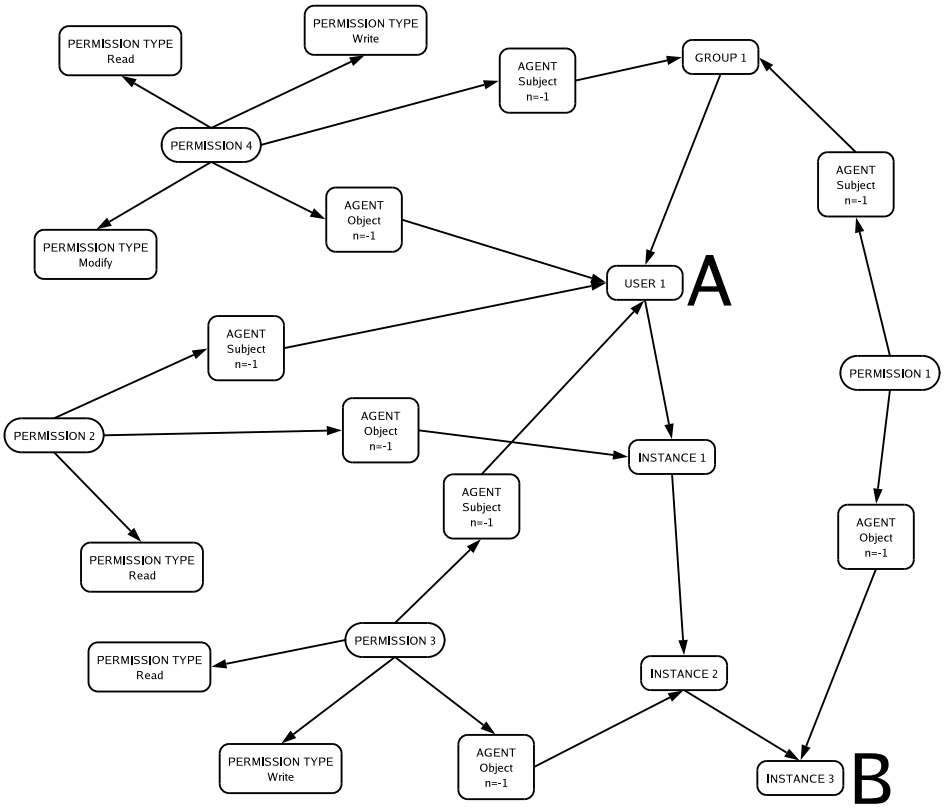

Figure 5.11: Determination of Applicable Permissions

The method just described can seem a bit complicated. After a bit of familiarization, the reader will see that this approach is quite natural and that it is very simple if the graphical structure is kept clearly

in mind. A few good tips to familiarize one's self with this method are as follows:

1. Draw out the graph
2. Use different colors to highlight the paths for l and f
3. Choose the permissions closest to the node A
4. Chose the permissions closest to B within the set defined at point 3.

Chapter 6

Contracts' Management

6.1 Introduction

This chapter will present a possible way to manage contracts using the ACC database. It is possible to consider different aspects of the contracts management. For each aspect it is useful to find out a suitable methodology and to integrate the various techniques to obtain a comprehensive contract management to handle everything. A desirable division of the management is the following:

- Management of the subjects of the contract.
- Management of the objects upon which a contract applies.
- Management of the rules that determine the economical aspect of the contract.

Other functionalities must also be given to this management, in such a way as to create a tool that can help stakeholders in making their decisions. Actually, it is necessary to determine if, for a certain company, one contract is better than another even before the agreement takes place. It is also necessary to give feedback to the company managers and their consultants and make it possible for them to verify that their choice best fit their needs. Another aim of this management is to be flexible enough to be able to handle also consortiums of companies and all the possible changes of behaviour that a company could perform in this context. A company could try to change from a consortium after a year and then decide to return to the old one or to try another option. Otherwise, a company could verify that the contract suggested by a consortium is the best for a subset of its counters and could decide to adhere to the consortium only with a subset of counters and stipulate a different contract for the others. On the other hand, the consortium could decide to stipulate more than a contract with one or more suppliers and to apply each contract to the suitable subset of managed counters. With only these few examples, it is clear that the management is not very easy to perform and that a global contract management for both the administrative and economical aspects require a considerable amount of time and of work before being completely developed.

Because of the scarcity of time and for educational purposes, it was decided to implement the database also using a standard methodology. Only the administrative portion of the management has been developed, and will be described here. Although it is very important, the economical management has been set aside for the moment, but will hopefully be realized as soon as possible.

6.2 Storing Historical Information

The main purpose of a database is to save information and to make the information available when the user needs it. In a versatile structure like the one furnished by the ACC database, everything must be planned–even if it is a simple operation. "From great power comes great responsibility," is the phrase that best fit this situation. Anyway, a user wants to be sure that the historical information concerning past consumptions and contracts is not going to be lost. When one user wants to weigh his options to know if another contract would be better than the current one, the management of the contract has to be split

in "REAL" and "VIRTUAL". As the reader can easily imagine, the REAL management will consider the contracts that actually have been or are signed by the company itself. Meanwhile, the VIRTUAL management will consider all the possible combinations that could be tried to verify the correctness of the choice of a contract. The most important thing to keep in mind is that the REAL management must be in a certain way static, and that it will not be modified often. It is clear that, except for extraordinary events, once a contract has be signed, it will be associated with a certain company until the expiration of the contract itself and that the information upon the just mentioned contract must be available for at least two or three years in order to allow comparisons.

After all these considerations of theoretical character it is time to introduce the sheer contractual management and to explain all the technical and more interesting characteristic of the management itself.

6.3 Administrative Contracts' Management

The first consideration to take in regards to the management of contracts is that usually a contract has two subjects that enter into the agreement and an object that represent the "thing" the contract applies on. In the case of electricity supply the parts that enter into contract are a supplier company that sells the electrical energy and a company or a consortium of companies that buys the electricity for one or more of its drawing points (Once again, the drawing point is the physical point where the electricity is taken by the supplier network). As it is described in Paragraph4.4 in the ACC database, there is no difference of representation between supplier and buyer companies, and this is logically admissible because a supplier is a company too. For this reason and for the fact that the ACC database does not allow the introduction of loops into its structure, even in this case, the contracts go to an higher level of the graph. With this organization of the structure, it is necessary to use agents to determine which company is the supplier and which is the buyer as it is shown in Figure 6.1 on the next page. As it was explained in Chapter 5 , the agents have a numeric field that indicates "how far" the characteristic goes, and in this case the number is always -1. This is due to the fact that the subject buying the electricity is always a company or a consortium. These two entities are always adjacent in the ACC database and a consortium does not have to supply itself so there is no necessity to stop the influence of the agent at a certain level.

Another useful thing that comes from the use of the agents table is the possibility to manage the eventuality that a supplier company buys electricity from another company or even from itself. This could be necessary during the drawing up of a balance sheet or for any other purpose. Anyway, it will be easy to implement this eventuality in the database defining the desired company both supplier and buyer.

6.4 REAL Contracts' Management

The real contract management is very important for many reasons. It is useful for the companies to maintain historical information about the choices that a company has made during the years. It can be used to know in a short amount of time how much the electrical consumptions have counted on the balance of the company in the previous years. It can be used to calculate if a contractual choice would be good in another period of the company's life.

This is only a small amount of the information that can be obtained knowing the contracts that a company has signed during the years, but it is sufficient to understand the importance of managing this aspect.

6.4.1 Contracts' Validity

The first issue that must be considered for the management is that a contract has a limited life. In fact, a contract defines in an unambiguous way the moment in which the contract begins to produce its effects and the moment in which the effects end. These two events define the temporal bounds for the validity of the contract and in the ACC database are handled using a combination of events and event_types instances as shown in Figure 6.2 on the facing page.

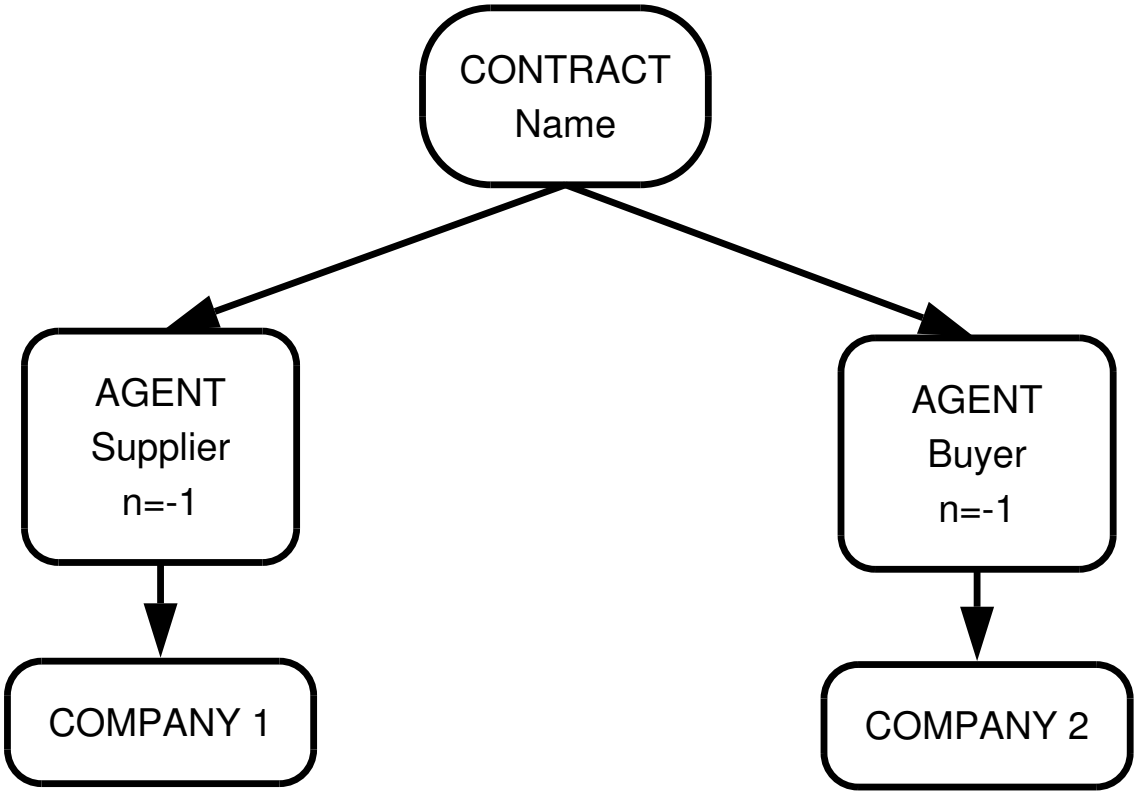

Figure 6.1: Subjects of a Contract

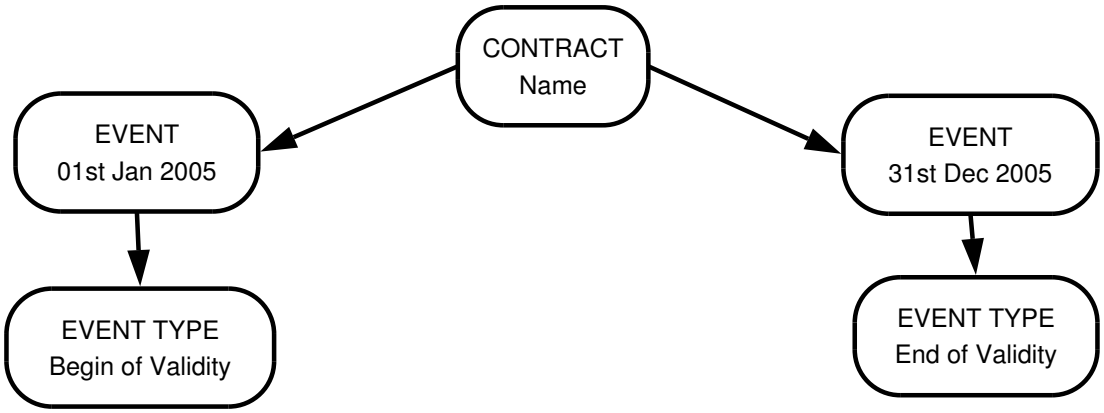

Figure 6.2: Contracts' Temporal Validity

6.4.2 Renewal Methodologies and Management

Another aspect of the contracts that is in strict relation to the time is the renewal of the contract itself. Before illustrating the modalities to manage this kind of events, some considerations must be taken. A contract can mainly have two possibilities for its renewal:

- The renewal is automatic and a company has to give the notice of termination of the contract to end it
- The company has to ask before a certain date for the renewal or will lose the possibility of renewal

The two renewal methodologies can also be handled by the combined use of events and events_type elements, but the user must pay attention to the type of the event because it is the type that determines the meaning of the event. If there are two contracts and both contracts have an event with a timestamp

of 30th of September, but one of type Renewal Expiration and the other with type Termination Notice Expiration, this means that the provider company must be notified to renew the contract in the first case and to terminate the contract in the second case. Figure 6.3 shows the example just mentioned.

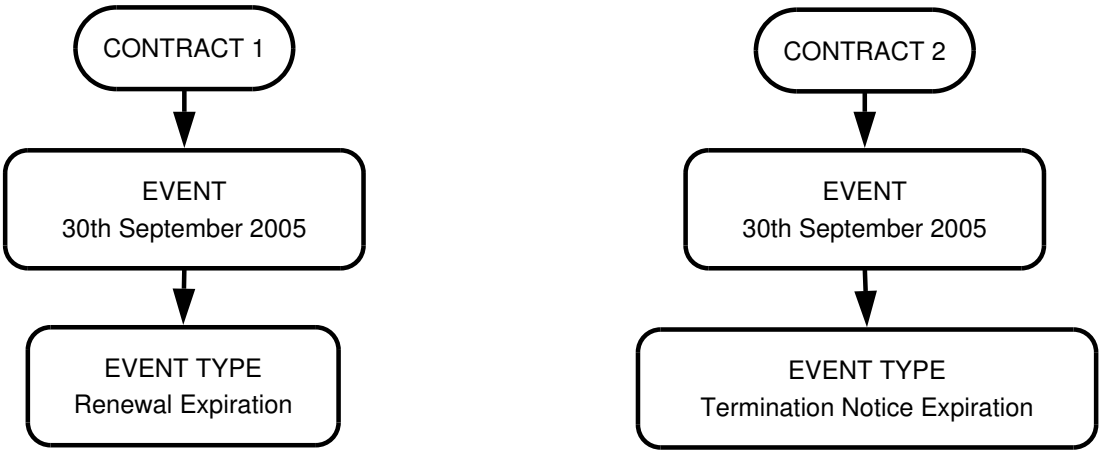

Figure 6.3: Renewal Methodologies

6.4.3 Object of a Contract

The objects of an electricity supply contract are the drawing points. Every drawing point has one, and only one, owner company. This, in computer science jargon, can be defined as a tree structure and can be used in combination with the numeric field of the agents to simplify this aspect of the management. Using an agent of type Object and with numeric field -1 is a simple and effective way to solve the problem even for companies with non-standard behaviour. Figure illustrate a simple possible schema, but more complicated examples will be exposed in 6.4.4.

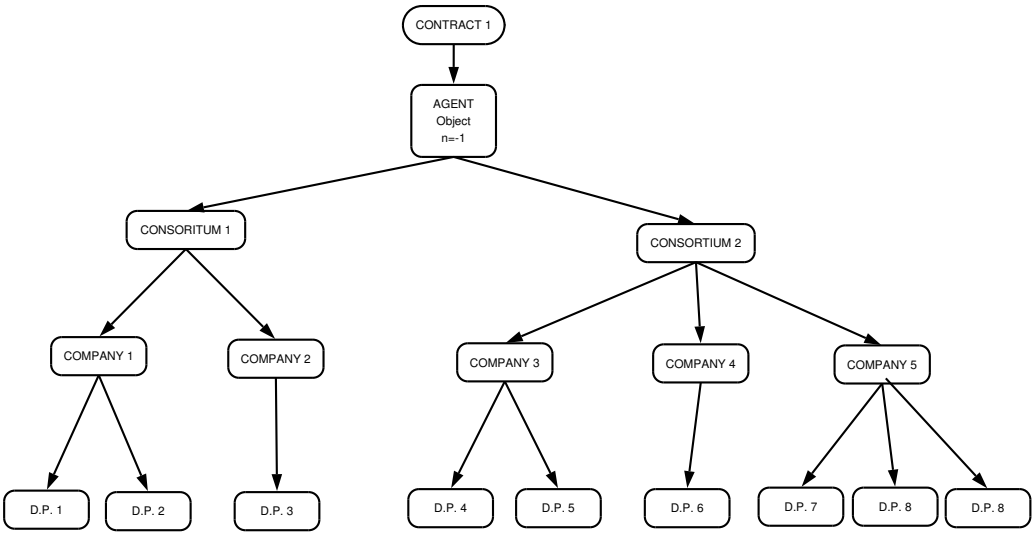

Figure 6.4: Simple schema for Contracts' Object Management

6.4.4 Example on How to Define the Object of a Contract

Now it is time to make an example more complicated in order to explore the potentialities of this method for defining the contracts' objects. Let's suppose a company has a lot of drawing points. With some analysis the company has discovered that the drawing points can be divided into three sets of consumptions

with different loading profiles. Using the ACC database, the managers of the company have decided to adhere to a consortium with the first subset of drawing points, to adhere to another consortium with the second subset of drawing points, and to have an independent contract for the others drawing points. For the implementation of this situation using the ACC database, one must keep in consideration that the agent that acts upon a certain instance of the database is the closest to the instance itself. To reduce the use of arcs one possible solution is to consider a contract (the one with more drawing points) as the "main" contract and the other as exceptions. Considering a contract the "main" one implies that the definition of the object of the contract will be done upon the Consortium and will be inherited by all the descendant of the consortium itself. For the other contract, the object will be directly connected to the contract, and this will define the relationship as shown in Figure 6.5.

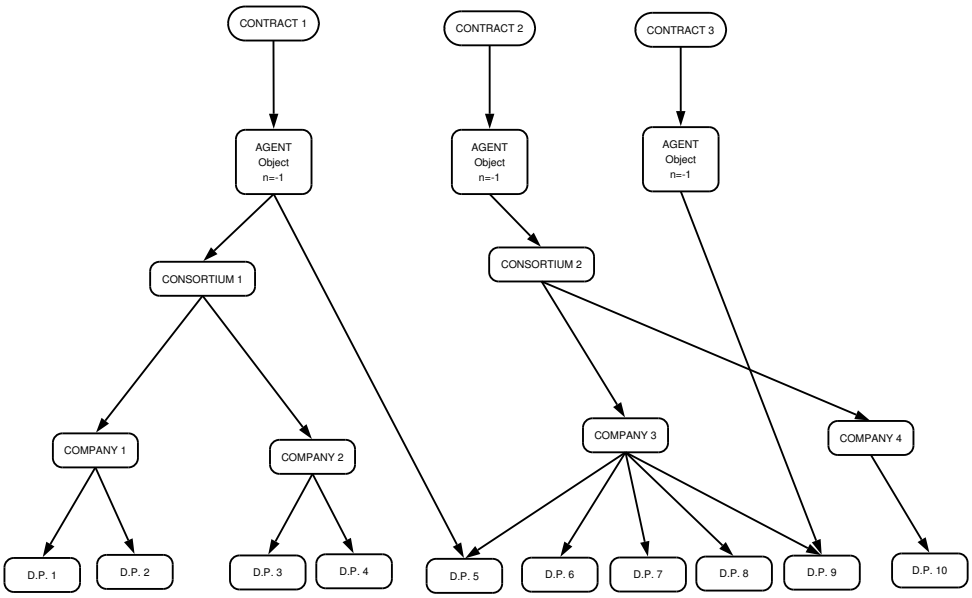

Figure 6.5: Example of Contracts' Objects

6.5 VIRTUAL Contracts' Management

The virtual management of the contracts will handle real contracts with the only restriction that these contracts are not the one in use for the considered company. The management is called virtual because it only deals with contracts hypothetically, without modifying the graph structure and lose the historical information.

A contract proposed for the next year can be handled as virtual, because they have not yet been analyzed and could be potentially all suitable to be signed. Contracts actually in use in other companies can also be managed virtually, with the purpose of verifying that the signed contract still best fits the needs of the company.

To define a contract as virtual, it is sufficient to use an appropriate agent with numeric field -1 and to connect the agent to the desired instance of the graph. If the purpose of the analysis is to find the best contract for an entire consortium the virtual agent will be connected directly to the consortium. If the analysis is meant to be restricted to a company or even some drawing points the connection will be done with these instances.

The main advantage of using the virtual contract is that it is possible to perform any kind of analysis without the danger of loosing the knowledge of the real graph structure. Figure 6.6 on the following page shows a possible configuration that uses virtual contracts. In this case there is the desire to compare CONTRACT 1, which is the one in use by CONSORTIUM 1, with CONTRACT 2 for the consortium, and with CONTRACT 3 only for COMPANY 3. As it is evident, the main structure of the graph is not changed and it is still possible to say that the the real contract is CONTRACT 1.

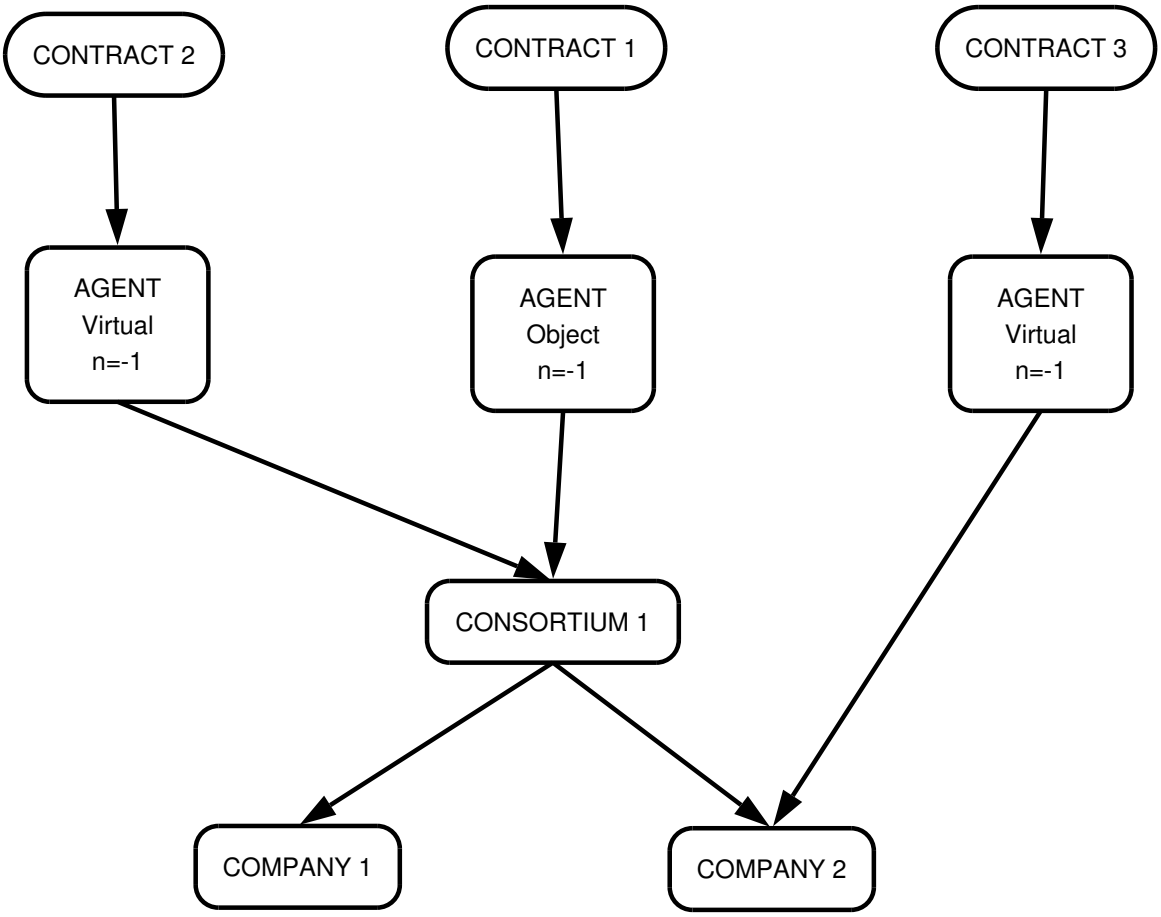

Figure 6.6: Virtual Contracts' Management

Chapter 7

Hybrid Realization

7.1 Introduction

The ACC is a database based upon a "versatile" technology of realization. The main purpose of this realization is to give extreme flexibility to the data structure. This flexibility comes from the renunciation of having a database with relations implemented within the database. The relations in this versatile technology are realized into a higher software level and have the drawback of slowing down the database. This is apparently a big deal, because it is possible to define relations m:n (Many-to-Many) on the fly without the introduction of new weak entities. In this way, it is not necessary to plan in advance the entire database structure and, if there is the need later, it is possible to modify everything without any hassles.

Anyway, there are some problems that haven't been resolved yet and therefore, the ACC database is not quite ready to be used. Under an methodological point of view the first problem is that the arcs, even though very powerful, need semanticity namely to contain more detailed information. However, under actual operating conditions, it is difficult to associate certain characteristics, such as a temporal validity to an arc. This means that it is not very easy to save historical information if at a certain time the database structure must be modified. The agents have been introduced to solve this problem, but they induce an increase in distance between instances of the database. The increase of distance between instances is considered to be a drawback and this could be a problem with some of the procedures. It might be possible to bypass the agents with direct arcs or add agents in any place that is necessary, but this solution could become complex and unsuitable for debugging. Another probable solution is to make the arcs become entities like any other entities in the database. This means that there could be arcs linking an arc with other instances of the database and use these instances to save the information concerning that arc.

Another problem for the ACC database is that by renouncing a database with internally implemented relations could involve certain problems of the user interface. There is a common drawback of these flexible designs: it could be possible that two different versions of the interface software running at the same time on the database could jeopardize its integrity. So, the problem was studied and a solution arose: preventing the user from directly interfacing themselves with the database. This is possible by inserting a software layer (middleware or application server) between the users and the database itself. In this manner, even if the clients have different versions of the interface software, only the application server will talk with the database and this will guarantee the univocity of the interface.

This spans a scope beyond a single thesis project. For this reason, it was decided to realize the database using a hybrid methodology, where the permissions management is realized with the approach using arcs, and some of the other entities are linked with DB foreign keys. This solution was also productive, under an educational point of view, because there was the possibility to learn the whole design processes of constructing a database.

7.2 Realization of the ACC_Hybrid Database

This section will describe and comment on the choice of tools and procedures used in the realization of the ACC_Hybrid database. It must be said that all the tools that are going to be mentioned are free software and that among the motives for this choice, there is the will to develop the database under the Linux behaviour. Another noteworthy point is that, although the ACC database was not completely realized, the conceptual work that was done to organize the data with such a versatile tool has rendered the planning of the hybrid realization of the database quick and easy.

Although the realization of the database was done using the hybrid technology, it is important to say that the first choice was the versatile version of the database. The lack of a native support in C language in the API of the database makes this kind of approach slow, especially for the entities associated to the consumptions. In particular the insertion of the arcs takes a considerable amount of time, because of the necessary controls to prevent the insertion of loops. Considering the high number of consumption entities and arcs necessary for the management, and that a development in the C language of a quick support takes too much time for a thesis project, it was necessary to use another method such as the classical ER to realize the database.

7.2.1 First step: Graphical Design (Top-Down Method)

There are mainly two ways to plan a database: the bottom-up method and the top-down method. The bottom-up method, as the name suggests, begins from the bottom of the database structure namely by the entities. This means that the tables of the database will be built first (creating usually small objects) and the relations between the entities will be planned secondarily. By proceeding in such a way, the resulting structure for the database will be optimized, but the resulting relation will not be accurately defined.

On the other hand the top-down method will begin from planning the Entity-Relation model and will consider the effective realization of the database secondarily. This produces well structured databases where it is easy for anyone to understand the database structure and where the resulting structure is optimized for the previously established aims. In case of later extensions, these earlier decisions could be not easily implemented.

In the case of the ACC_Hybrid database, it was chosen to follow the top-down method and build the database mainly considering the organization of the data. This choice is mainly due to the fact that the greater part of the structural analysis was already performed trying to realize the versatile version of the database. Therefore, even if the details of the ACC_Hybrid database structure were not completely planned there was already a good design of the result.

Realizing a database with the top-down method is still a difficult task. The whole structure of the database must be clear in mind from the beginning, but it is not easy to remember every relation if the database has many entities. And what about the entities themselves? How can one remember their names and the fields they contain? For this reason, graphical tools have been developed in order to help programmers in the realization of databases. These programs have many useful potential capabilities, such as the possibility to translate the graphical planning of the database in the SQL language. Some such programs can connect themselves to the DBMS and make the database run even if the development has not yet been completed. Therefore once it is clear that such a kind of program is needed the only thing that remains to do is to choose which program to use.

For the realization of the ACC_Hybrid database, the DBDesigner was chosen as the graphical tool. The DBDesigner was chosen, because it works under both Linux and Windows environments, and because it can generate SQL code for the PostgreSQL DBMS[1]As with any other graphical tool DBDesigner is very easy and intuitive to use and reduces the time of development.

[1]The choice of PostgreSQL DBMS has many reasons: it works under Linux and it has full adherence to SQL92 and great part of SQL99; it is open source and allows full control on deployment and internal structure; it grants full control on inner engine Application Programming Interface.

For these reasons, some (if not all) aspects of the arcs structure could easily implemented in C language to be integrated in such API.

7.2.1.1 Contracts' Management

The main objective of the database is the management of electrical supply contracts. This happens because all the rules necessary for defining the management are contained in the contracts and with them strictly related. For this reason a lot of tables in the ACC_Hybrid database are connected to the contracts table. This table contains only basic information about the contracts, but has many relations m:n with any other area of the database.

The database can, in fact, be thought of as an entity divided in different graphical areas:

- The Stakeholders Area
- The Drawing Points Area
- The Toponyms Area
- The Economical Area
- The Permission Management Area

This division was realized in order to apply the "Divide and Conquer" paradigm. Namely, divide the problem into a number of different and hopefully easy to solve subproblems, solve each subproblem individually, and then group the solutions to obtain a comprehensive solution to the problem as a whole.

The application of this method united with the use of the DBDesigner tool of development has led to a graphical division of the problem and to the birth of the above mentioned areas. Because contracts are central in the management, each area (except for the permission management one) is connected with the contracts table as the Figure 7.1 on the next page (a screenshot from the DBDesigner program) shows clearly.

7.2.1.2 Stakeholders' Management: Suppliers and Purchasers

Section6.3 explained the reasons for having a division between subjects of a contract and an object of a contract. The separation between these two categories has been adopted also in the ACC_Hybrid database. There is anyway a difference between the ACC and the ACC_Hybrid because the supplier and the companies have two different tables in the ACC_Hybrid database. This matter has the disadvantage that if a company is both a supplier and a purchaser, the information upon this company must be inserted twice in the database even if the fields in the two tables are the same. Using only one table for suppliers and companies could be possible setting a flag on the company in order to indicate if the company is a purchaser and another one in order to indicate if the company is a supplier. However, this does not solve the problem of a supplier that buys electricity from itself because the only effect of this solution is to move the problem towards the contracts table. There is, in fact, the need for a relation m:n between contracts and purchasers/providers and the weak entity holding the information upon this relation would have to possess a flag too. Such a flag is necessary to indicate if the connected company in the companies table is the proposer of the contract. And what happens if the flags in the company table and in the relation table are not in agreement? The solution of having two different tables for suppliers and purchasers, even if sometimes it introduces some repetitions, is the simplest under the planning point of view and much more simple to understand by people which see the database for the first time .

In the ACC_Hybrid database, stakeholder consortiums are considered to be purchaser companies and provider companies. Each of these categories has its own table to save the related information. There are also the weak entities (tables) that defines the relation between these tables. To acquaint the providers with the contract there is the table "providers_have_contracts". This table defines a m:n relation, because even if in a certain moment a contract can be proposed only by a single provider, it is possible for a provider to cease its activity and to pass its own contracts to another provider. For this reason, the relation "providers_have_contracts" has two field indicating when the relation begin its influence and another indicating when this influence ceases to have effect. The consortium has also a relation m:n with the contract, because during the years a contract can be signed by different consortiums and a consortium can decide to change contracts. Also, in this case, there are two timestamp fields in the purchaser_consortiums_definition table in order to maintain historical information upon the choices made by the consortium during its existence.

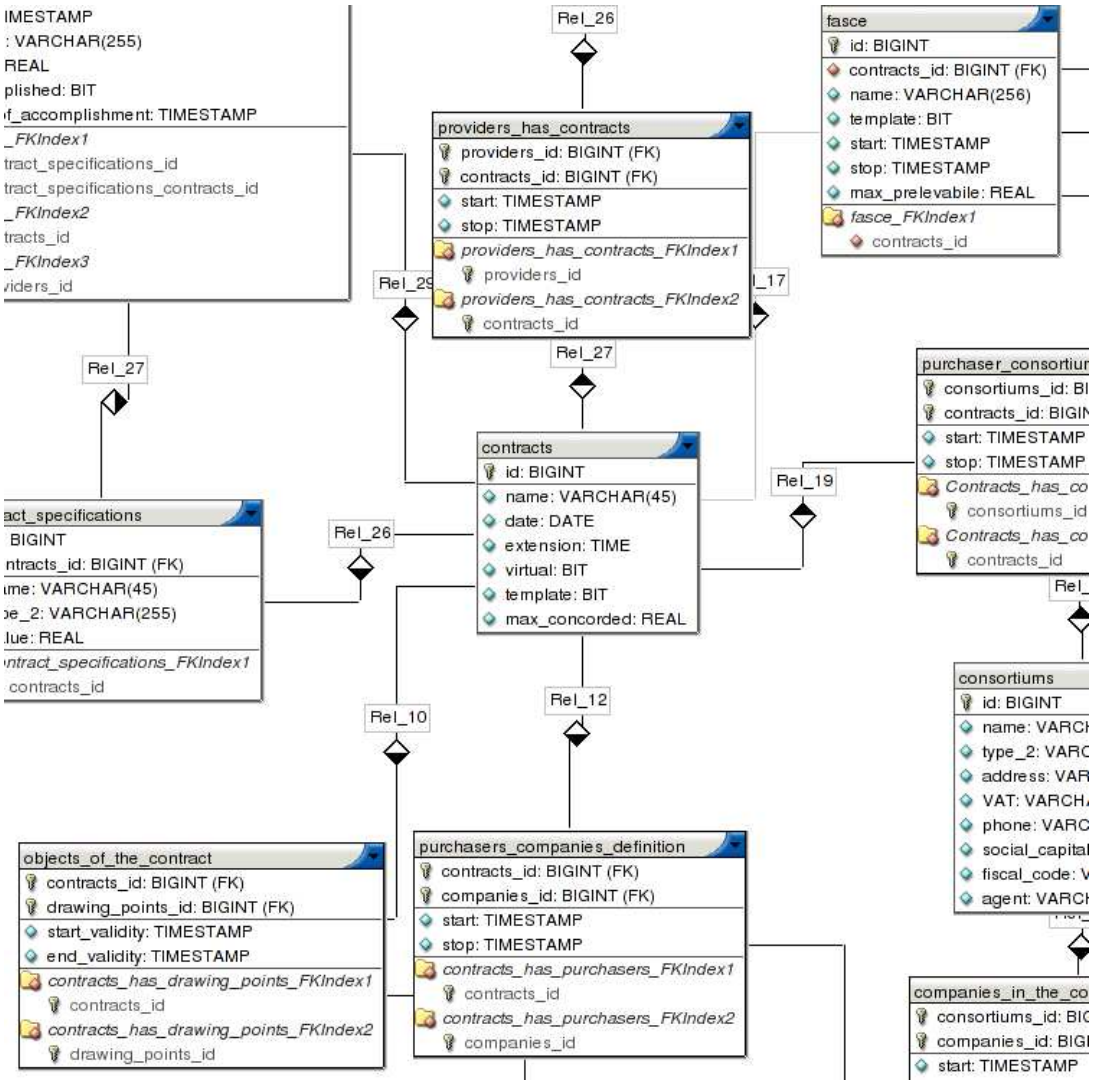

Figure 7.1: Centrality of the Contracts

For the companies table, the situation is a bit more complicated, and there are some considerations to take into account. First, even if there are consortiums available, it is not certain that a company has adhered to anyone and such a company can still sign one or more contracts. It is also possible that a company belongs to a consortium and decides to trust every decision of the consortium without trying to sign a different contract. It is just as possible as the previously mentioned cases that a company decides to apply the contract chosen by the consortium to a subset of its drawing point and to find one or more contracts to satisfy its remaining needs of electrical energy. These considerations have led to the creation of a relation m:n between companies and consortium and another relation m:n between companies and contract. Each of these two weak entities have the two timestamps that limit the temporal extension of the implemented relations.

In Figure 7.2 on the facing page is shown a DBDesigner screenshot with the Stakeholders' Management portion of the database.

7.2.1.3 Drawing Points' Management

Drawing Point is the name chosen in this thesis to indicate the physical point where one purchaser can take electrical power from the distribution network. Generally, a contract for electricity supply regulates the modalities and the prices for the supply on a number of agreed upon drawing points. In other words, the drawing points are the objects on which the contracts act. For this reason, the management of the

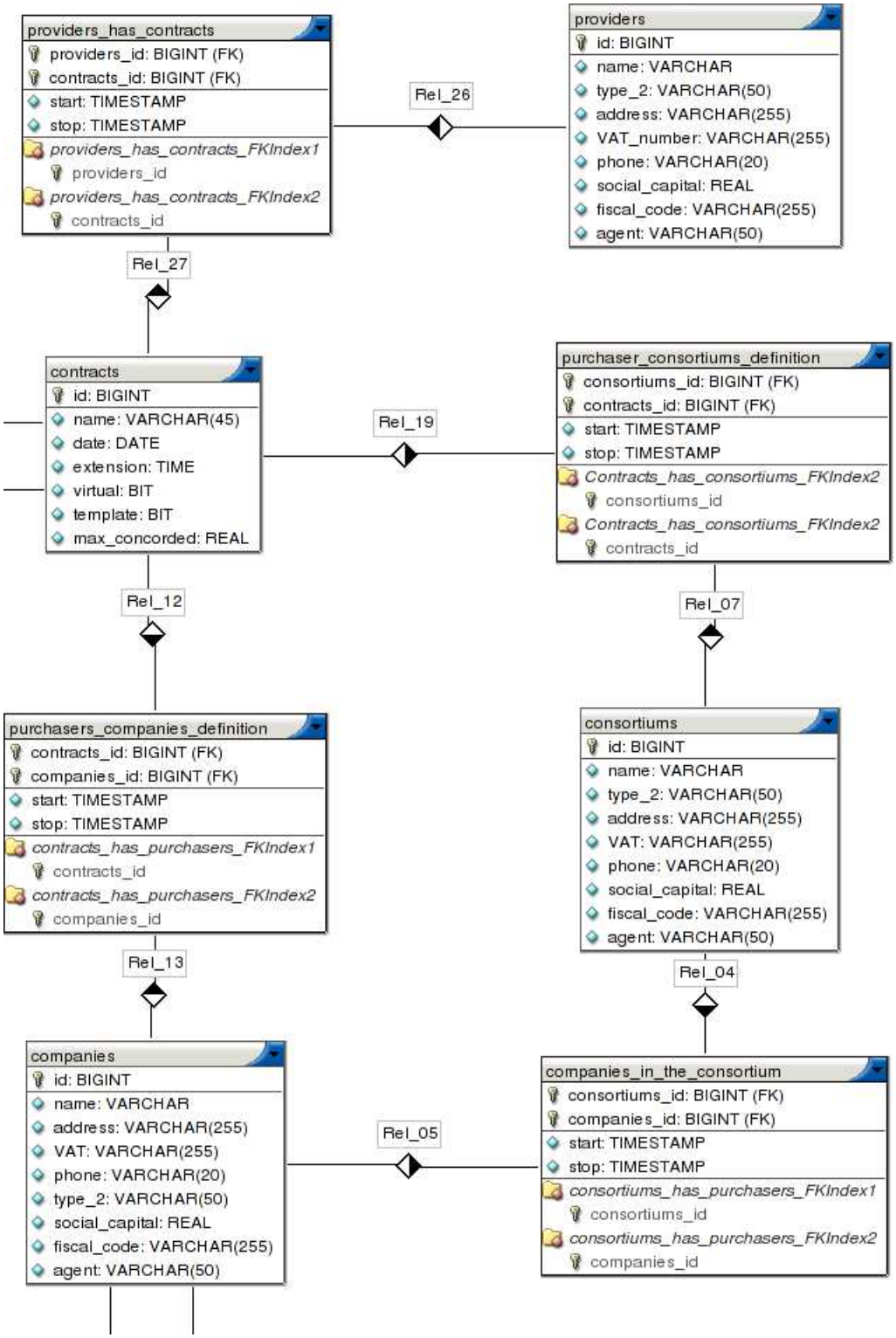

Figure 7.2: Stakeholders' Management

drawing points and the consumption associated to them is very important and must be planned very well.

One of the main characteristics of this portion of the database is that there will surely be a lot of insertions and queries and these queries will usually be simple. The insertions will be necessary to save

the data concerning the consumption of any company managed by the database. It is easy to calculate that in a year there will be at least 70,080 values per company and this is if the company has only a single counter! And what about the queries? It is necessary to extract information from the database: to calculate the expenditure for the electrical consumptions, to decide if one contract is better than another, or to perform a statistical analysis. So it is clear that the queries will be very frequent.

Another consideration that deserves to be taken, is that the consumptions must not be kept directly in relation with the counters. It is beyond any doubt that one consumption is related to the counter that measured it, but for the contractual aspects of the management, it is not necessary to have a direct relation between the two. If there were a relation m:n between counters and drawing points and a relation 1:n (One-to-Many) between counters and consumption, in the case of a counter cancellation, it would be impossible to determine to which drawing point the eliminated counter's consumptions belong.

To avoid this possibility, the solution of having a relation m:n between drawing points and counters, a relation 1:n between drawing points and consumptions, and a relation 1:n between counters and consumptions was embraced. As for any other relation m:n in this database, also the aforementioned one has a limited temporal validity and the reason for this is to preserve historical information even if the relations between entities change. The relation between consumption and drawing_points/counters does not have a temporal validity because any consumption belongs only to one drawing point and to one counter in any case. If for any reason a user wants to have any information upon the counter that measured a certain consumption, it is possible to directly query the counters table to accomplish this task.

As it was stated before, the drawing points represent the objects of the contracts and there must be also a relation m:n of finite duration between these two entities in this case. The logical connection between the management of the purchaser and the drawing points is realized putting into relation the drawing_points table and the companies table. It is easy to understand that a drawing point can belong only to one company and not to a consortium because a consortium represents a group of companies, but it is not a physical entity. With the implementation described until now, it is easy to manage the cases where a company has signed more than one contract and it is not difficult to associate the right contract to any consumption. Figure 7.3 shows the graphical representation of the drawing points' management portion of the database.

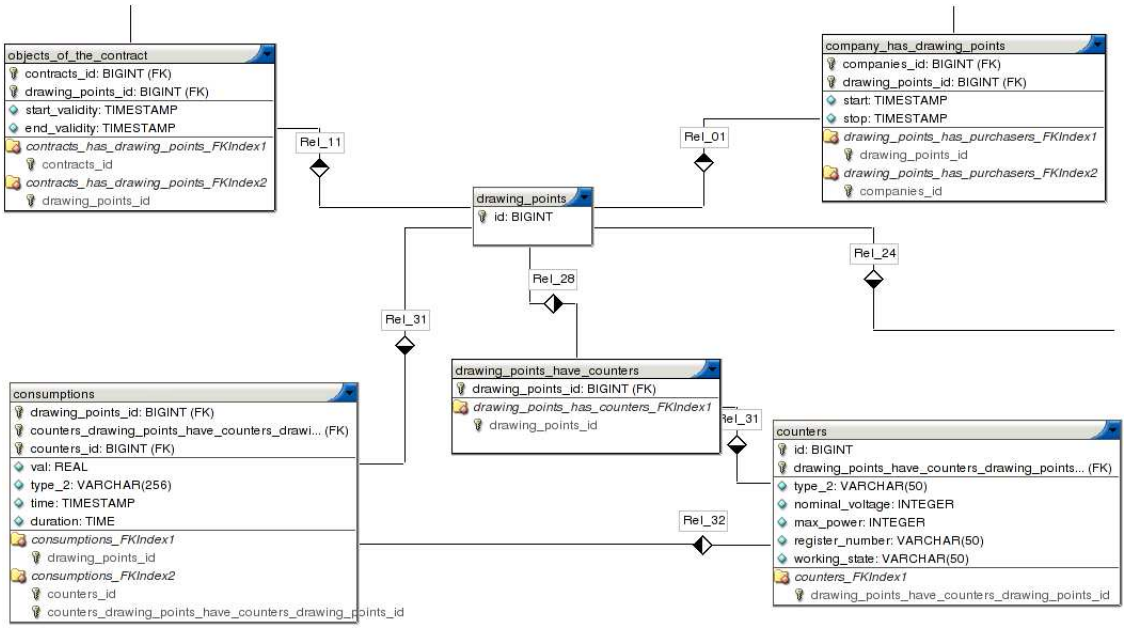

Figure 7.3: Drawing Points' Management

7.2.1.4 Toponyms Management

For the contractual management it is sometimes necessary to know the geographical location of drawing points and companies. It is possible that electrical energy prices for working days could be different from prices on holidays. The problem is determining which days are holidays. This might seem like a very simple task, but there are differences in the definition of a holiday from one country to another. Sometimes there are also differences between two neighbouring counties and being able to know where a company or a drawing point is located is essential to determine which days are holidays.

To solve this problem, an area of the ACC_Hybrid database has been dedicated to the management of the toponyms. There is only one table containing the geographical information and its name is "toponyms" as anyone could easily guess. The toponyms table has a m:n, with a limited temporal validity, relationships with the companies table and another similar relationship with the drawing_points table. Having a relationship between toponyms and a company and another relationship between toponyms and drawing_points, could seem a waste of space. If any drawing point is located in the same place of the company the preceding affirmation is true, but in the case of a drawing point located in a different place it would be impossible to determine the exact toponym for the drawing point and the real amount of the electrical expenditure without having the second relationships table.

If one company has a lot of drawing points located in the same place of the company itself it could seem not necessary to save this information for any of such drawing point. It could seem smarter to obtain the location of the drawing point using the company's one. In that case, the relation between drawing points and toponyms could be used only for those drawing points residing in different location than their owner company. When a user would want to know where a drawing point is located the user would try to verify if there is a specific definition for the previously mentioned drawing point otherwise the user would use the company location. This method has the drawback that it is necessary to execute a higher amount of queries to obtain the location of a drawing point than with a direct definition of the location for any drawing point. furthermore the time for the queries will be greater for the second realization than for the first one, also because it must first be verified that the drawing point has the same location of its company. For these reason the information will remain divided even if more space is required.

It might be interesting to have the geographical information upon other entities of the database such as providers and consortiums. Fortunately, this is not strictly necessary for the contractual management and in this way it is possible to avoid the introduction of two more tables in the database and keep the structure as simple as possible.

In Figure 7.4 on the following page is shown a screenshot taken from the DBDesigner program of the toponyms management area.

7.2.1.5 Prices and *Fasce* Management

One of the most important tasks that the ACC_Hybrid database has to accomplish is the one under an economical point of view. The management of *Fasce* and Prices is in fact essential to calculate how much a company has to pay for its electricity supply. First, a definition of the word *Fasce* must be given. In this thesis, the intervals of consumptions with the same nominal value will be called *fasce*. It must be remarked that the value is defined by the providers. All the consumptions with the same nominal value, will be grouped in the same *fascia* (Italian singular form of *fasce*). It was said that one *fascia* defines a nominal value, because the price for the same *fascia* is not constant with the flow of the time. With the possibility of buying energy from the electricity market, there could be even 24 different prices for the same *fascia* per day. The number of *fasce* differs from one contract to another. Also, the rules necessary to determine if a consumption belongs to the *fascia* "A" instead of to a fascia "B" are defined in the contracts and are extremely varying.

Another issue to consider is that the queries in this area of the database will be very complicated and will require a great computational effort to be done. Fortunately, such queries are not going to be very frequent, because there is no need to know how much a company is spending every ten seconds. Even when one user wants to perform a comparison between two or more contracts, the total amount of the expenditure will be calculated once for each contract and the number of contracts will not be too great.

In order to speed up the extraction of the data, the data will be inserted into the tables with an organization that can facilitate the extraction. A lot of care must be used for the insertion into the table containing the rule necessary to define when a *fascia* must be applied. As it is easy to imagine, the

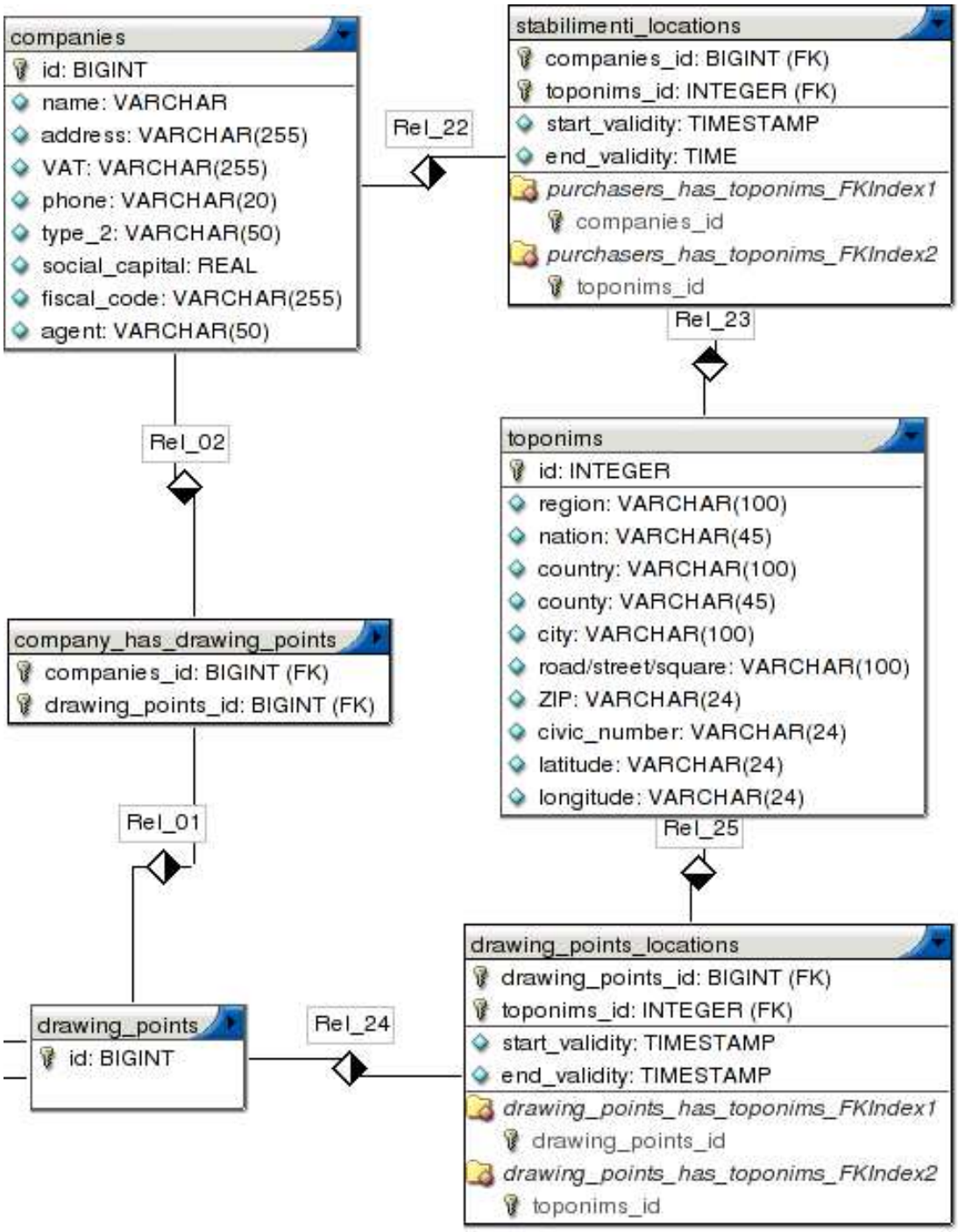

Figure 7.4: Toponyms' Management

insertion of the data in this table will be extremely important and will necessitate a lot of effort from the database users. For the users there is the consolation that once the data is inserted, it will remain in the database practically unchanged for years and that the hard work during the insertion time will allow for a quick extraction and less work in the phase of calculating expenditures.

Let's now describe how this area of the database is organized. To have a better idea, let's take a look at Figure 7.5 on the next page.

The first thing to notice is that there is a 1:n relationship between the contracts table and the *fasce* table. This is necessary, because a contract can define one or more *fasce* each with its name. Even for

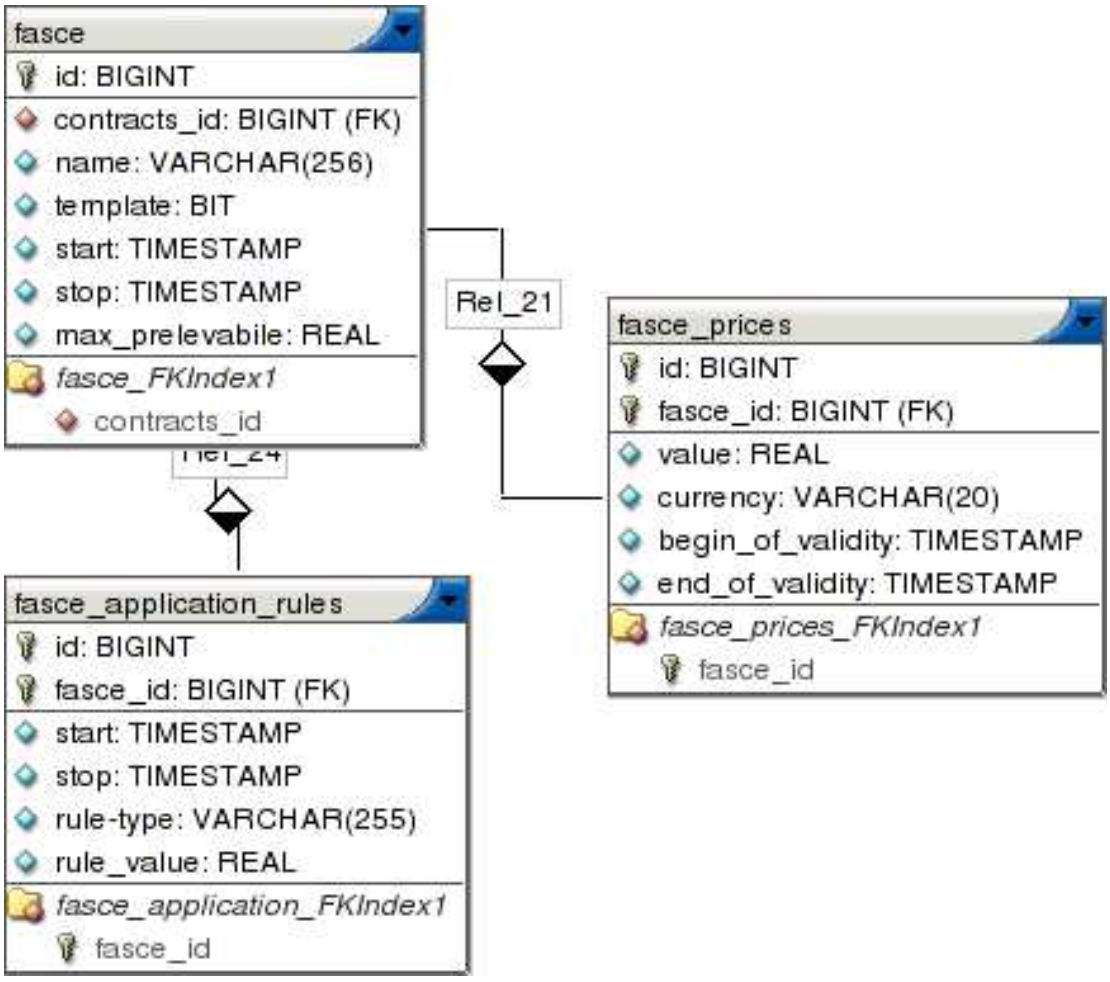

Figure 7.5: Fasce and Prices' Management

the *fasce* there is a limited temporal validity. This happens for example with the *fascia* F4 defined by some contracts. This *fascia* is applicable only during August and does not exist in other periods of the year. Sometimes, a *fascia* also has a maximum amount of energy that can be taken (the field is called max_prelevabile in the *fasce* table). If one buyer needs more that the just defined quantity of energy, this energy will be considered as taken from another, and usually more expensive, *fascia.*

Continuing the analysis, there is a 1:n relation between the *fasce* table and fasce_prices table. This happens because the price for the electricity is not constant during the year and a price can be valid for an hour just as it can be for a year. Usually the price for a certain *fascia* is changed once every three months, but as it was mentioned previously, there may be a different price for every hour of a day. To tell the truth, there are two tendencies on the proposed prices. The first tendency is the one just described and the second one is to have a single price for the whole year. Both can be manage by this area of the database, but it is desirable to have a procedure for the insertion of prices for those contracts with prices defined by the market.

The last, and most important part of this area of the database is the fasce_application_rules table. This table defines the period of time included between the start and the stop timestamps the kind and the value of the rules necessary to decide to which *fascia* a certain consumption belongs. The rules could be based upon the hours of the day or upon some other characteristic of the consumption such as the percentage of the daily consumptions. Only one important restriction must be taken into consideration, namely that the rules have to be mutually exclusive. In other words, one user has to be able to determine which *fascia* a consumption belongs to without any doubt.

7.2.1.6 The "Versatile" Permissions Management

The permission Management in the ACC_Hybrid database is made by applying some of the ideas developed for the versatile version of the database. Some opportune modifications have been introduced in order to integrate the versatile technology with a standard database. This is the area of the database that makes it hybrid. The ACC_Hybrid database without this portion would be a normal database and could work although it would have security problems. The integration between the two philosophies of development was not painless and some compromises were necessary. These compromises have led to a substantial reduction of the permissions' power, but the permissions' management is still good and surely flexible enough for the needs of the database.

The area of the database containing the permissions management is not connected with the rest of the structure, and the ACC_Hybrid database, for this reason, does not satisfy the 2NF constraints. This is not a big loss, because the greatest part of the database is in 2NF and the unconnected portion can use all the functions developed for the ACC database to remain consistent.

Before entering into the details of this management, it is better to take a look at Figure 7.6 on the facing page where the area of interest is shown. The first point that is self-evident is that the only entities of the database with permission on the others are the groups. To tell the truth, it is also possible for the users to determine the permission upon other entities of the database, because one user inherits the permission of the groups to which it belongs. The table entity2id has the same purpose of the homologous one in the ACC database. The table permission_of_groups behave similarly as the arcs table of the ACC database. The difference is that in this case, the desired group will have the permission indicated into the permission_of_groups table upon the instance of the database defined by the couple of values entity_id, instance_id. In this manner, it is possible to set permission recursively upon groups. This recursion would be impossible with a standard database, but it is evident that the permission implemented here are much less powerful than those presented in Chapter5. The function and triggers necessary to guarantee the coherence of this portion of the database are essentially two:

- The get_entity_id_from_table_name function that extract the entity_id from the table entity2id given a table name as input.
- The delete_row_from groups trigger, called when a group is deleted, that eliminates every tuple from the permission_of _groups table with the deleting group as pointed instance.

Both trigger and function have been implemented in PLPgSQL, but they could be written in C language to improve the performance of the database.

7.2.2 Second step: Low Level Optimization with SQL (Bottom-up Method)

Planning a database with a graphical tool like DBDesigner is very useful in the first steps of the development process. With a vision of the database, it is possible to decide in a short amount of time the number and the characteristics of the tables to use. There is also the possibility of creating the relationships between tables and to translate the graphical schema into the SQL language. When the number of tables grows, the advantages of using of a graphical development method cease to have their effects. This happen mainly for two reasons. The first reason is that the database becomes big graphically and it is difficult to have a complete view of the structure. The second reason is that it is difficult to have a clear view of logical structure of the database. Especially when there are many relations between the entities of the database the structure begins to be a bit confused, and it is impossible to improve it using the graphical environment. In theory, it would still be possible with a 3D schema, but this solution is not applicable yet.

The usual, and hopefully best, way to operate when there are too many tables in the database, is to directly use the SQL language and continue the development in such a way. This gives also the advantage that it is possible to optimize the database at the entities' level, maintaining the structure as it was optimized graphically. The ACC_Hybrid database at the point of the development described until now, already has approximately 25 tables and it was planned to create a "Logs" and an "Events" table. It is true that two more tables would be easy to add to the graphical schema, but not every weak entity of the database would have a relation 1:n with the "Events table", while every table of the database would have a relation 1:n with the "Logs table." Even if a 1:n relation is represented by a single line it is evident that the schema would not be so clear with something like 44 more lines running through it.

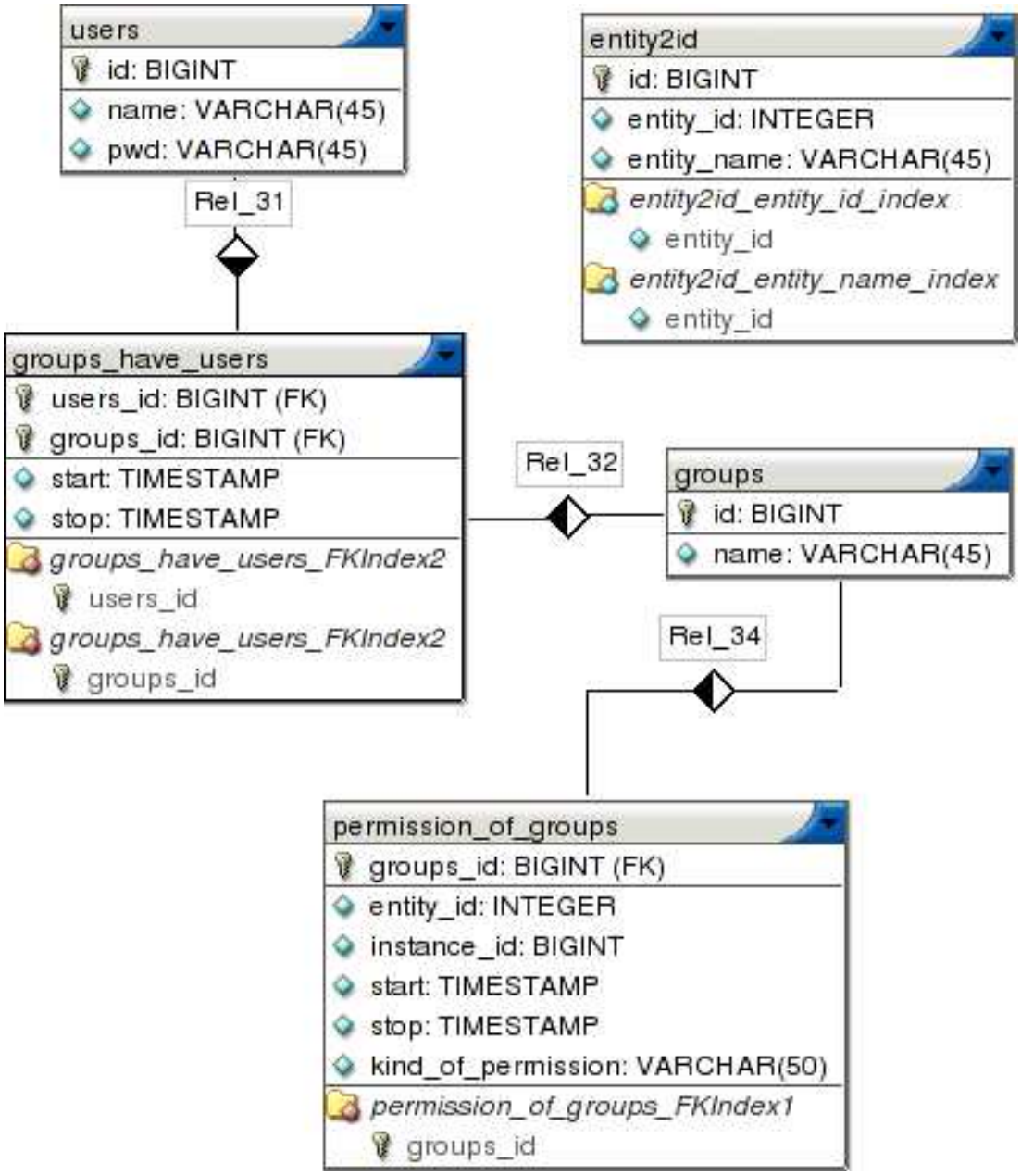

Figure 7.6: Versatile Permissions' Management

Therefore, the graphical development was stopped at this point and the schema was translated into the SQL language in order to make it possible to edit the code and modify the database. The first modification to the SQL code was the creation of the trigger for the deletion from the groups table. The trigger is necessary to maintain the coherence in the versatile portion of the database. The trigger has the task to call the delete_from_groups function. When a group is going to be deleted the function that was just mentioned deletes any permission that has the deleting group as an object, in order to avoid the presence of the permission pointing towards non-existing objects.

The successive features added to the database have been the "Events table" and the "Logs table." The "Events table" was meant to manage everything that could happen and that was not yet handled by the database. An example of event could be the malfunction of a counter. This event will be generated by the counters table and will indicate that from a certain moment all the consumption measured by the broken counter could be wrong. There are also events representing tasks that have to be accomplished and for this reason there must be something to indicate such a possibility. Anyway, it is more pressing

to describe the fields contained in the “Events table” than make the reader guess them. The fields of interest into the events table are:

- times of kind timestamp
- type_2 of kind varchar
- val field of kind real number
- flags field of kind varchar
- time_of_accomplishment of kind timestamp

The times field says when the event was generated. The type_2 field is necessary to determine what kind of event is represented by any tuple. The val field indicates the events that necessitate an opportune value. The flags field contains flags, one of which could indicate if the event is an accomplishable task. A time_of_accomplishment that is not null indicates for those events that support it, that the time was accomplished in that precise moment. The other fields of the table are necessary to define the 1:n relation between every table of interest and the events table. Anyway, such fields will not be described here.

The other table added to the database was the “Logs table.” The purpose of this table is to save information upon events that happen to any table of the database. At this point it is not defined if any event or only the errors will be saved in the “Logs table,” but this is only a marginal issue because only a timestamp and a description are necessary for a log. And these two are the only fields that the “Logs table” has for the description of what happens to a table, while the other fields are necessary only to set a relation 1:n between any other table of the database and the “Logs table.”

An observation that could be made at this point is that it is usually better to avoid the presence of tables with many null fields. It is easy to understand that both the “Events table” and the “Log table” will have only one of the fields necessary for the relationship to be not null while the others will be null. It is true that if any table of the database would have its own “Logs” and “Events table” there would not be the aforementioned problem. There would only be the little difference of having twice as many tables as with the adopted method, and the necessity to look in two dozen tables if it is necessary to know what happened to the database in the last hour.

For this reason, the information in the log and the events will be contained in only two table, with the awareness that even if a bit inefficient. The inefficiency cams from the use of single table for events and log that take place in every table of the database. In order to have a relation 1:n from any table of interest and the events and logs tables it is necessary to have a specific field into the just mentioned tables. It is also highly probable that for any event or log there will be only one or two of such field not null. This is a waste of space because even if a field is empty its space is assigned to him and unusable by the database. Anyway, this solution is good under a programmer’s point of view, because there are only two tables for every log with a simplification in the interfaces and queries construction.

7.3 Middelware and Python Interface

A database is usually used by a lot of users and often these people do not have a clue about what a database is and how it works. For a programmer, it is impossible to pretend that a user will learn the SQL language in order to use a database. It would be a very good thing, even if unnecessary, to have users that know what a *“SELECT”* SQL command does. Otherwise, there must be other instruments for facilitating people’s interaction with the database.

Usually there is one interface running on a web server and the user can connect himself to the server using his web browser and use the interface to interact with the database. The interface can have two or three layers depending on the characteristics of the database. If the database is a standard relational database the interface has usually only two layers: the web server and the database server. In this case the user “talks” with the web server and the web server “talks” directly with the database server translating in a common language the requests of the user. The database server performs all the operations required and then returns the result to the web server that gives such results to the user. If the database uses a fuzzy data modeling, this means that only few operations on the data contained into the

database are performed by the database server, and the other operations–such as insertion checks–must be performed at a higher level. Unfortunately, a web server was not planned to accomplish such tasks and an application server is necessary between the web server and the database server. The task of the application server is to "listen" to the requests that the web server passes to it and translate such requests into a series of operations that the database server can perform on the database. Usually, this conversion is made using a series of functions running on the application server, and developed for such a purpose. When the database server gives back the result of the operations the application server manipulates the results and gives them to the web server.

Apache as the web server, Skunkweb as the application server and PostgreSQL as the database server were chosen for the ACC and the ACC_Hybrid databases.

This section is going to discuss these just mentioned functions. Actually there are not functions, but there is an interface written in python language[2]. The interface is divided in two layers: a inner layer with basic functions common to any other class, and an outer layer with one class for any function found to not be a weak entity of the database.

7.3.1 DBCommon Package

The DBCommon contains the DBEntity class. DBEntity is inherited by any entity class of the interface and defines all the functions that interact with the database. To perform any operation on the database only the function defined into this class will be used.

Here is a list of the most important functions defined in the DBEntity class:

- queryOut: receives as input a SQL query and returns the result of the query as a dictionary
- validateParams: receives a list of parameters as input and replace some of the characters with others if they could compromise the good result of the desired operation
- executeSQL: receives a string containing a SQL command and returns the result of the query
- getList: receives a string defined in the outer calling class and a dictionary and returns a formatted list of rows satisfying the "WHERE" clause passed with the dictionary
- getListID: receives a dictionary as input and returns the ID and the name of the rows satisfying the "WHERE" clause passed with the dictionary
- getListShort: receives a string defined in the outer calling class and a dictionary and returns a formatted list of rows satisfying the "WHERE" clause passed with the dictionary
- insertRow: receives a dictionary as input and inserts a row with the values defined in the dictionary into the desired table
- deleteRows: receives a dictionary as input and delete all the rows satisfying the "WHERE" clause passed with the dictionary
- updateRow: receives a dictionary as input and update with the values contained in the dictionary the row satisfying the "WHERE" clause passed with the dictionary
- updateSingleField: receive a dictionary as input and update the field defined into the dictionary with a value also passed with the dictionary of the row satisfying the "WHERE" clause passed with the dictionary

7.3.2 Entities' Interfaces

Any entity determined to not be a weak entity of the ACC_Hybrid database has its own python class that performs any necessary operations upon the database. The classes defined for this purpose extend the DBEntity class and inherit all the functions defined there. For this reason, only the definition of a few class attributes of kind string must be performed to customize any class and nothing else is necessary.

Here is a list of the class attributes names and characteristics:

[2]The choice of the python language was made, because it is an easy to learn and powerful language. It is also supported by the Skunkweb application server.

- _entity_name: contains the name of the entity as it is defined in the database
- _SQL_getListShort: contains the SQL command for the extraction of rows from the table without formatting the input
- _conn: contains the connection to the database
- _curs: contains a cursor[3] on the database
- _sql_getList: contains the SQL command for the extraction of rows from the the table with the definition of a desired name for any field of the query's output
- _sql_insert: contains the SQL command for the insertion into the database
- _sql_update: contains the SQL command for update a row of the database
- _sql_delete: contains the SQL command for the deletion of one or more rows from the database
- _sql_update_single_field: contains the SQL command for update a single field of a certain row of the database

[3]The cursor is defined upon a database connection and are used to perform operation upon the database. The main characteristic of the cursor is that there could be more than one cursor per connection and that any operation executed with a cursor does not have effect on the database until a commitment is executed on the cursor. With the cursor it is possible to limit the number of connections to the database and work on different parts of the database simultaneously.

Chapter 8

Statistical Analysis

8.1 Introduction

Statistical analysis is one of the most important tasks that a tool for the management of electrical consumptions must perform. The analysis is necessary to determine how well, under the point of view of a supplier, a company is behaving. It is important for a company to have an account in good standing, because this could mean saving money. One of the aspects that a supplier cares about the most is the percentage of reactive power absorbed by a company compared to the total amount of the consumptions. This parameter does not need a statistical analysis, because in theory, the reactive component of the absorbed electrical power must be kept as low as possible and usually, its ratio to active power is constant. To keep such a value as low as possible, it is necessary only to calculate the reactive inductive component of the load–this is a usually a constant value and can be determined in a single point–and decide if there is the need to insert compensating loads after the counters of interest or before some problematic machineries.

Another important aspect that a supplier cares about, and that needs a statistical analysis, is the ability of knowing how much a company is going to consume in the next day, week, or month. This is useful because it is impossible for a supplier to increase the production of electrical power instantaneously. Knowing within 24 hours that there will be an increase of required power could give the supplier enough time to adapt its energy production or to buy the necessary quota from another supplier. Analogously, a supplier could reduce its production or sell some energy to another supplier if there is the capability of knowing that the request for electricity is going to decrease. Predicting the consumptions may seem to the advantage of only the suppliers. For a company there is no need to behave well in order to have predictable consumptions, because there are not any advantages. Therefore, some suppliers have proposed discounts to those companies or consortiums who can commit in advance to how much energy they will need. For this reason, it is necessary to have a tool for predicting the consumption of a certain company, given its past consumptions. For a consortium, it is necessary to define if the consumption of a company within the consortium is predictable or not. For a consortium, it might be equally as useful to sub-divide its companies into sets with different degrees of predictability. In such a way, it would be possible to avoid contracts for certain sub-divisions of the consortium where it is required to make a commitment to a fixed quantity of electrical energy in advance. This is especially useful for the companies that a consortium defines as having a low degree of predictability and allows those companies to save money.

In statistics, tools have been developed for historical economical series analysis, but these tools are arduous under a computational point of view or difficult to use without consulting an expert in statistics. One of the intents of this thesis was to try to develop a lightweight software able to perform all the necessary analysis with a limited need for human assistance and statistical background. It was clear from the beginning that such software would not be easy to plan, but after some attempts, an algorithm was developed and it looks promising.

It must be remarked that such software is at the moment under testing. It is also being studied as a thesis project in collaboration with Domenico Caputo, a statistics student of the University of Padua and also one of the authors of the algorithm.

The next sections will describe the methods kept in consideration, including a promising one, with

particular attention on the aspects that must be developed or tested.

8.2 Linear Regression

The linear regression is a statistical analysis method useful where a variable can be interpreted as a linear combination of independent variables. The simplest general formula of the linear regression in the case of linear dependence of one variable from n independent variables can be written as follows:

$$\underline{y} = \beta_0 + \beta_1 \underline{x_1} + \beta_2 \underline{x_2} + ... + \beta_n \underline{x_n} + \underline{\epsilon} \tag{8.1}$$

where $\underline{y}$ is the vector representing the variable, $\underline{x_1}, \cdots, \underline{x_n}$ are the vectors of the n independent variables, $\beta_0, \cdots, \beta_n$ are the coefficients of the linear combination and the vector $\underline{\epsilon}$ is the estimate for error.

The regression is a procedure that, using the "Minimal Squares Method," calculates the coefficients β_i in order to minimize error. The Equation 8.1 can also be written in the compact form

$$\underline{y} = Xb + \underline{\epsilon} \tag{8.2}$$

where

$$\underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} ; \; X = \begin{pmatrix} 1 & x_{11} & x_{21} & \cdots & x_{p1} \\ 1 & x_{12} & x_{22} & \cdots & x_{p2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{1n} & x_{2n} & \cdots & x_{pn} \end{pmatrix} ; \; b = \begin{pmatrix} a \\ b_1 \\ \vdots \\ b_p \end{pmatrix} ; \; \underline{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix} \tag{8.3}$$

The calculation of the b vector can be done by applying the simple following equation:

$$b = \left(X^T X\right)^{-1} X^T \underline{y} \tag{8.4}$$

It must be remarked that the columns of the X matrix can be composed of functions of the independent variables instead of the variables themselves. It could be possible, for example, to use x^2 if there is the possibility that the relation between the x and y variable is of the parabolic kind.

The such calculated b vector is then used to calculate the estimate $\underline{\hat{y}}$of $\underline{y}$

$$\underline{\hat{y}} = Xb \tag{8.5}$$

Generally, to verify if the adaptation to the real data of the regression is good, the following coefficient is used

$$R^2 = \frac{\sum_{k=1}^{n} (\hat{y_k} - \bar{y})}{\sum_{k=1}^{n} (y_k - \bar{y})} \tag{8.6}$$

This coefficient varies from 0 to 1 and if it is close to 1 this means that the adaptation is truthful.

8.3 Linear Regression with Fourier Transform

Performing a good linear regression is not such a difficult task, but it isn't considered easy either. Usually, the linear regression is calculated using the Equation 8.2 with the coefficient b given by the Equation 8.4. As it was stated in the Section8.2 the columns of the X matrix can be functions of independent variables, and the person who performs the regression chooses the functions. In the case of the active component of the electrical consumptions, the plot of the data shows a periodic progression of the consumptions in relation to time. It must be remarked that for any periodic real function $x(t)$ exists the following development into Fourier's series

$$x(t) = A_0 + \sum_{k=1}^{\infty} A_k \cos\left(2\pi k f_0 t + \theta_k\right) \tag{8.7}$$

Where $A_0, A_2, \cdots$ are the coefficients of the development into Fourier's series.

Comparing the Equation 8.5 with the Equation 8.7 and the Equation 8.3 and considering that the consumptions are measured once every 15 minutes and therefore that every function has a discrete time domain, it was decided to try to perform a linear regression upon the last n consumptions using a matrix X of kind

$$X = \begin{pmatrix} 1 & \cos(2\pi f_0 t_0 + \theta_0) & \cdots & \cos(2\pi f_k t_0 + \theta_k) \\ 1 & \cos(2\pi f_0 t_1 + \theta_0) & \cdots & \cos(2\pi f_k t_1 + \theta_k) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \cos(2\pi f_0 t_n + \theta_0) & \cdots & \cos(2\pi f_k t_n + \theta_k) \end{pmatrix} \tag{8.8}$$

Where the frequencies $f_0, \cdots, f_k$ and the phases $\theta_0, \cdots, \theta_k$were calculated performing a Fourier transform upon the consumptions–choosing the points where the absolute value of the transformation was greater than a defined percentage of the absolute value of the transformation at frequency equal to zero. Because of the almost periodicity of the consumptions the frequencies $f_1, \cdots, f_k$ were multiple of the frequency f_0 and for the properties of the function $\cos(x)$ it resulted that

$$\left(X^T X\right)^{-1} = \begin{pmatrix} \frac{1}{n} & 0 & 0 & \cdots & 0 \\ 0 & \frac{2}{n} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{2}{n} \end{pmatrix} \tag{8.9}$$

This matrix was then multiplied with $X^T \underline{y}$ in order to perform the linear regression and to calculate the b vector. The results of the calculation show that the first element of the b vector was $\frac{1}{n}\sum_{k=1}^{n} y_k$ which is the estimated mean from sample.

Once the b vector had been calculated using the regression, it was possible to estimate the further p values of the consumptions $\underline{\hat{y}}$ applying the Equation 8.5 under the form $\underline{\hat{y}} = X_1 b$ where

$$X_1 = \begin{pmatrix} 1 & \cos(2\pi f_0 t_{n+1} + \theta_0) & \cdots & \cos(2\pi f_k t_{n+1} + \theta_k) \\ 1 & \cos(2\pi f_0 t_{n+2} + \theta_0) & \cdots & \cos(2\pi f_k t_{n+2} + \theta_k) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \cos(2\pi f_0 t_{n+p} + \theta_0) & \cdots & \cos(2\pi f_k t_{n+p} + \theta_k) \end{pmatrix} \tag{8.10}$$

8.3.1 Why This Method Does Not Work in this Context

Although this method of predicting is easy to understand and could seem suitable for the electrical consumptions analysis, there are some considerations that must be taken. First, the linear regression is a method developed for the analysis of a variable inside the domain of the independent variables and is not meant to be used for calculating values outside such domain. This means that the method is very good in approximating the past consumptions, but it is not useful in the predicting phase.

The second and most important reason for rejecting this method is that it is not good for predicting periodic functions. This happens because the function was built as linear combination of cosinusoidal functions that repeats itself once the domain is extended. This repetition implies that the first weeks prediction will be more similar to the first week used to perform the analysis than that to the last and closest one. Under a logical and statistical point of view, it is more probable that the consumptions on the next week have a progression closer to the week just before that than to a week further along the line. Figure 8.1 on the next page represents an example of the predictions for five weeks using the same amount of weeks for the regression. In the first graph the approximation is shown (in green) of the consumptions (in blue) made by the method. As it was said before, the approximation follows the real progression very well. When the method is used for predicting the consumptions (second graph) it is evident that the green curve is a replica of the approximation made in the previous graph (once again green curve) and that this method is not suitable for a prediction. Figure 8.2 on the following page shows the error made by this method in the prediction of the consumption in the first week. As it is evident the magnitude of such error is very high and also the Root Mean Square Error is pretty high. In fact $RMSE = 115$ for this week and it is a symptom of the inefficiency of this prediction method. Another drawback of this procedure is the long computational time. Even if the FFT is a pretty fast algorithm, the calculation of the columns of the X matrix take a lot of time and the process proceeds very slowly. At this point, it is evident that this method has to be rejected and that a better one must be found.

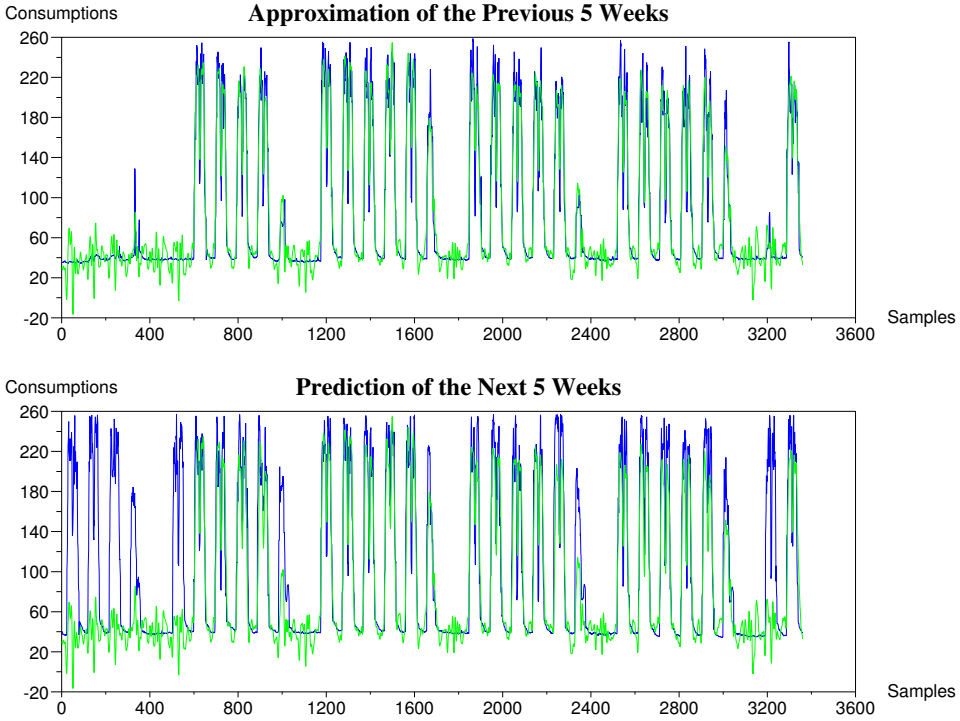

Figure 8.1: Linear regression and Fourier Transform

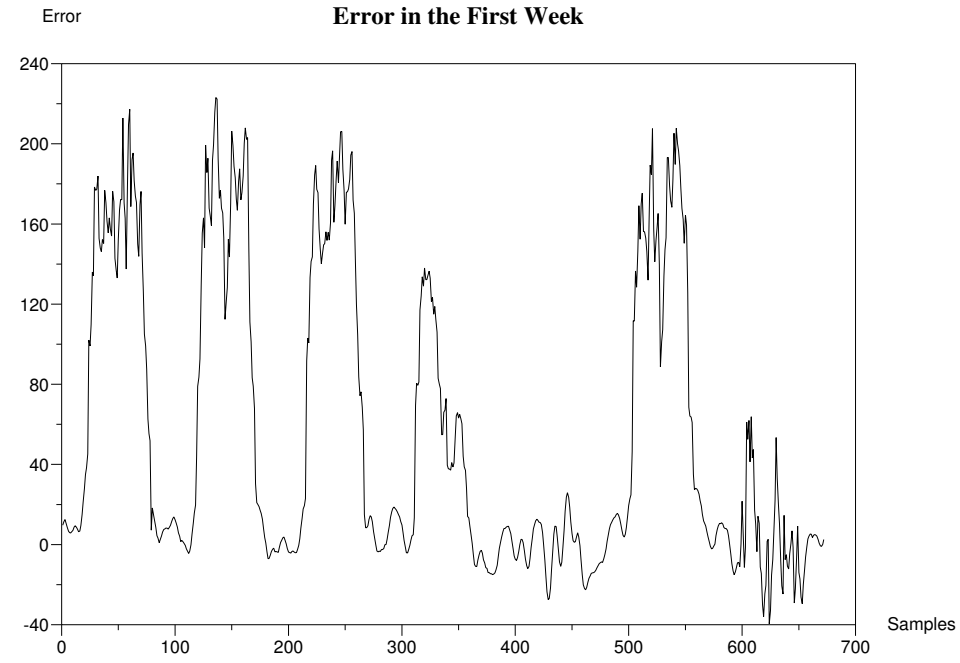

Figure 8.2: Prediction Error with the Regression-FFT Method

8.4 A Different Application of the Linear Regression

To develop this new procedure, a more adequate statistical approach was applied. It was made possible only after an analysis of the data. The application of the previously described method had made the

conclusion that one company's consumptions has a weekly progress and it was decide to use the week as the main periodicity[1] . From the correlation analysis it was soon evident that the companies whose plotting of the consumptions seemed to be more regular, had meanly a high value of correlation between the consumption made in a certain week and those made in the next week. It was therefore thought to write down a week (meant as consumptions) as a linear combination of a certain number of previous weeks

$$W_k = \alpha_1 W_{k-1} + \alpha_2 W_{k-2} + \cdots + \alpha_p W_{k-p} + \underline{\epsilon_k} \tag{8.11}$$

where W_k is the week expressed as linear combination of the $W_{k-1}, \cdots, W_{k-p}$ previous weeks, $\alpha_1, \cdots, \alpha_p$ are the coefficients of the linear combination, and $\underline{\epsilon}$ is the estimate error.

The second supposition that was made is that said

$$W_{k+1} = \beta_1 W_k + \beta_2 W_{k-1} + \cdots + \beta_k W_{k-p+1} + \underline{\epsilon_{k+1}} \tag{8.12}$$

the week after the W_K one it is $\alpha_1 \simeq \beta_1, \alpha_2 \simeq \beta_2, \cdots, \alpha_p \simeq \beta_p$ namely that it is possible to write

$$W_{k+1} = \alpha_1 W_k + \alpha_2 W_{k-1} + \cdots + \alpha_p W_{k-p+1} + \underline{\epsilon_{k+1}} \tag{8.13}$$

At this point, linear regression was used as the method to find out the coefficients $\alpha_1, \cdots, \alpha_p$ considering the week W_{k+1} as the week to predict. Applying the regression to the week W_k and using the weeks $W_{k-1}, \cdots, W_{k-p}$ as columns of the matrix

$$X = \begin{pmatrix} W_{k-1} & W_{k-2} & \cdots & W_{k-p} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \tag{8.14}$$

it was calculate the vector b of the coefficients $\alpha_1, \cdots, \alpha_p$ with the application of the Equation 8.4.

The vector b obtained in such a way was then multiplied by the matrix

$$X_1 = \begin{pmatrix} W_k & W_{k-1} & \cdots & W_{k-p+1} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \tag{8.15}$$

as it was stated in the Equation 8.5 obtaining then the estimated value of the week W_{k+1}.

Using a linear combination of the previous weeks in the estimate of the current week, gives an advantage to the mediation of the statistical fluctuation implicit in the weeks in order to extract the predictable component of the consumptions.

This method seems to be very promising, also under an applicative point of view, because it has low enough execution times, and allows the user to estimate the consumptions of a week using a relatively small number of the previous weeks. Even with 10-12 weeks it is possible to get good prediction and also in the case of 100 weeks the execution times are under 7 sec using a Celeron 1.6 GHz. This is a big improvement, because even with only 5 weeks for the analysis the previous method had execution time of more than 43 seconds on the same machine.

Figure 8.3 on the next page show an example of the result of the application of this method for predicting a single week. The first portion of the graphic shows the approximation (in green) calculated using the linear regression of the consumptions (in blue) that took place the week before the predicted one. The second graphic shows the predicted consumptions (in green) calculated using the algorithm that was just explained compared to the real consumptions (in blue). Figure 8.4 on the following page shows the prediction error for this method of analysis. Also in this case the Root Mean Square Error was calculated and it resulted: $RMSE = 3.39$ that is very better than the previous method's value.

8.4.1 Self-Correlation Function

One thing that the reader could ask for, is the reason why it was decided to use the week as temporal quantum of the statistical analysis. The reason cams from the application of the self correlation function, a statistical method to analyze historical economical series. Given a series $S(k)$ of data representing the samples of the phenomenon of interest the self-correlation function is defined as follows:

$$SCF(k) = Correlation\left(S(1:(n-k)), S(k:n)\right);\ k = 0, \cdots, n \tag{8.16}$$

[1] For companies with a daily progress of the consumptions it could be possible to consider the single day.

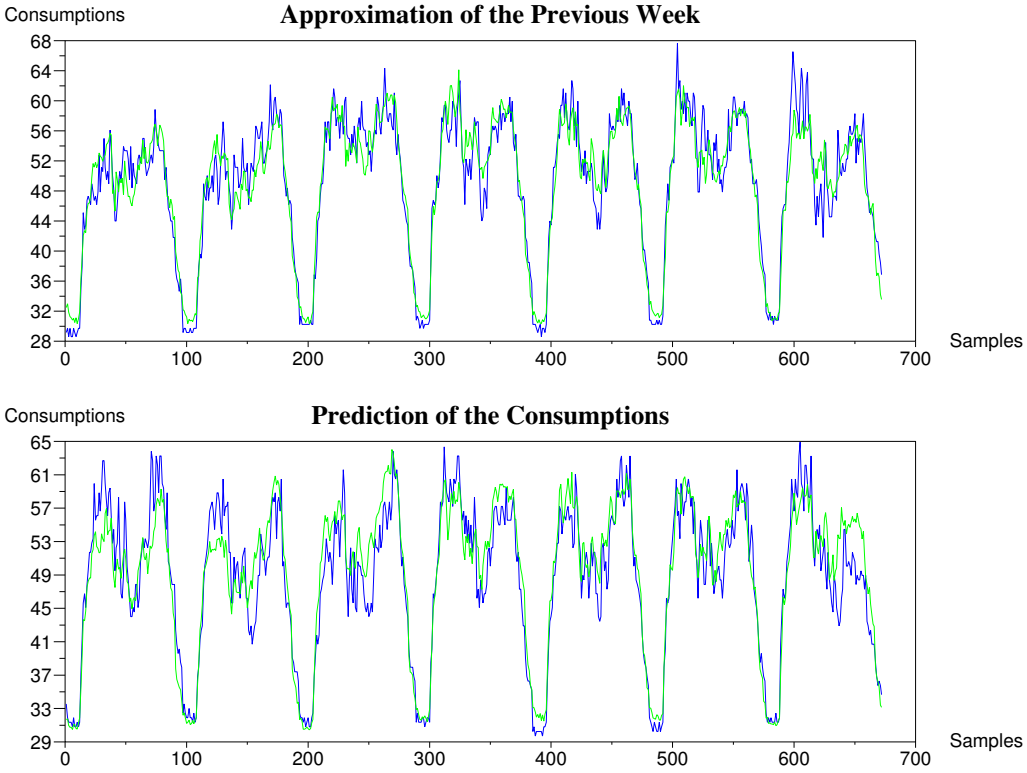

Figure 8.3: Example of Forecasting using the new Method

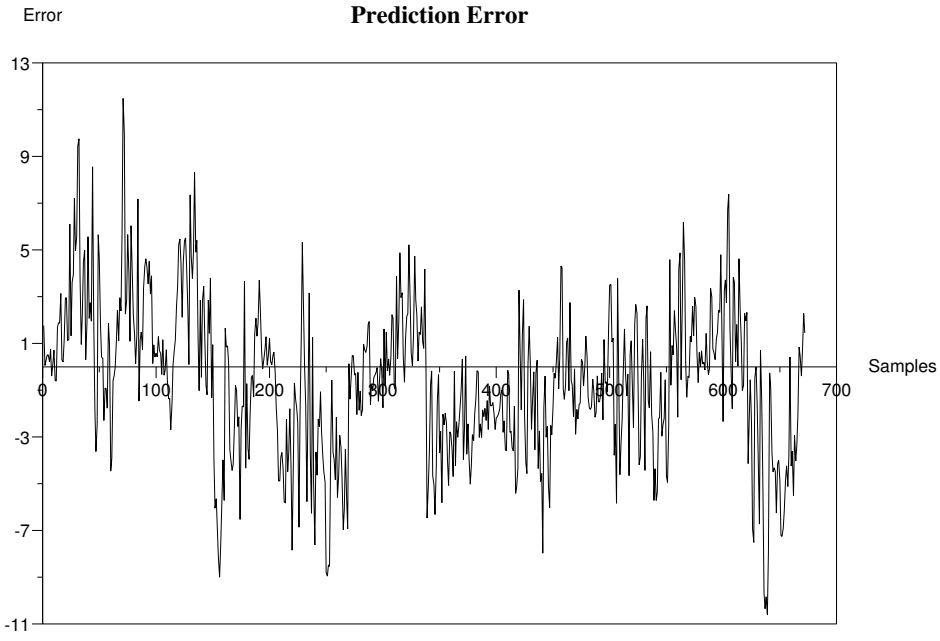

Figure 8.4: Prediction Error for the Regression Method

where $S(a:b);\ b \geq a$ is the sub-series $S(a), S(a+1), \cdots, S(b)$ of S and n is the length of the series S.

The self-correlation function has normally an oscillatory progress. The points where the series has maximums indicate that a translation of such amplitude of the series let the series very similar to the original one. In other word the points of local maximums are the periods after that the series repeats itself.

In the case of the data used for testing the statistical analysis, it was seen that the self correlation function had a very high peak for a periodicity of 672 points. Considering that there is a sample every 15 minutes, this means a periodicity of a week and so a week was chosen as the period for the prediction. If there would have been a periodicity of 960 this would mean a ten day periodicity and the analysis would have had 10 days as temporal quantum. Figure 8.5 shows the function correlation for a typical company .

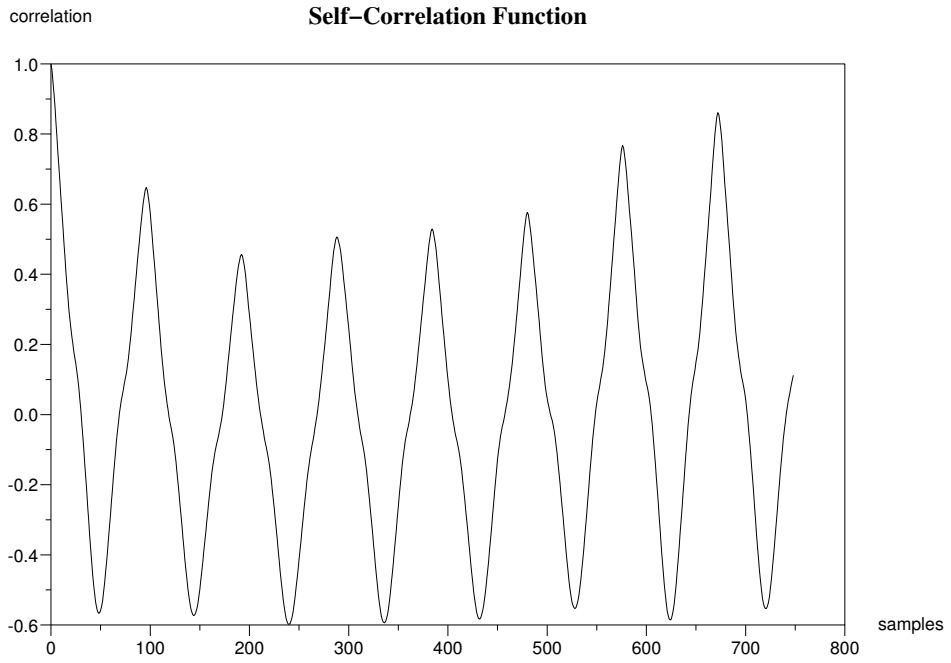

Figure 8.5: Self-Correlation Function

An argument that could be done against this method is that, because of the high correlation between a week and the next one, it could be possible to replicate the current week in order to predict the next one. Given y the current week, y_1 the estimation of the current week made using the linear regression, $y_{predicted}$ the prediction week, and y_{true} the next week, it is possible to say that meanly it is $Correlation\,(y_{predicted}, y_{true}) \geq Correlation\,(y, y_{true})$ and this is sufficient to say that this estimation is not worse than the replication of the current week.

With the use of this methodology, it is natural to use the correlation between a week and the next one as a index of the regularity of a company. In reality the most important factor is the mean of the correlation and its minimal value, but for the moment only the mean will discriminate if a company is good or not, even if some tests are necessary in order to give the right correspondence between correlation and predictability.

Another point that must be made is that the algorithm for calculating the self-correlation function is heavy under a computational point of view. It is desirable to use it only once or twice in a year because if the application of this algorithm gave a different result every week this would indicate that a company's consumptions are not easily predictable.

Chapter 9

Future Directions and Conclusions

9.1 Future Directions

Among the things that have not yet been performed, and that deserve to be done, is the necessity to develop the versatile version of the database. It must be given semanticity to the arcs and the possibility to consider them as any other entity. The permissions and the contracts management must be adapted to this new kind of arcs and properly tested.

For the hybrid version of the database, the calculation of the expenditures must be tested as long as the toponyms management and the permission management. More consumptions must also be inserted in order to verify the correctness of consumption prediction.

Also, the predicting algorithm must be improved. It must be given the possibility of choosing the use a week or any other period of time as the base for the regression. It must also be given the possibility of predicting more than a single period. In the case of periods with a low correlation with the others, or with one or more null values in them, it is desirable to allow the user to eliminate such periods.

For the more complicated cases, the possibility of using other predicting algorithms such as the ARIMA[1] has to be given to the user .

9.2 Conclusions

The aim of this thesis was to develop a database able to handle the electrical consumptions for companies and consortiums and to provide users with tools in order to help the stakeholders in their decisions. The first realization taken into consideration was a versatile realization of the database where the relations between entities were implemented using arcs entities. This realization had the great advantage of being very flexible, but unfortunately, was not very easy to customize. The power of this versatile approach was evident with the permission management where the problem of allowing some entities of the database to perform certain operations upon other entities of the database was solved in a simple and effective way. The introduction of the agents table temporarily solved the problem of the lack of semanticity of the arcs. When it was time to plan the contracts management, it was evident that the use of the agents did not give all the necessary guarantees of stability in the long run and it was decided to temporarily put aside the versatile version of the database and to build the database using a standard approach.

To tell the truth, it was clear that a small portion of the permissions' management realized with the versatile approach could be inserted into the new database, and a hybrid version of the database was built. For this realization, a graphical development tool was used first and then some modifications were made directly using the SQL programming language.

A python interface was then made and the insertion of the consumptions of few companies was performed in order to improve the insertion times and to develop a statistical tool capable of predicting consumptions. After quite a few attempts, a promising procedure was conceived and realized using the Scilab program. Some tests were performed on the consumptions and it was verified that the new

[1] the use of ARIMA algorithm is discouraged because of the long executing time and the necessity of a statistical expert's advice

algorithm works well if there is a high correlation between every week and the next one. Some problems were encountered with correlation locally low and if there are consumptions with a punctual null value. The case of consortiums was not tested, but it is easy to demonstrate that the sum of predictable consumptions is also predictable. In the case of not predictable companies it could be possible for the sum of their consumptions to be predictable and particular care must be taken in this analysis.

Bibliography

[1] Directive 96/92/EC of the European Parliament and of the Council of 19 December 1996.

[2] Legislative Decree of the Italian Republic of 16 March 1999, n. 79.

[3] R. A. Elmasri, S. B. Navathe, *Sistemi di basi di dati - Fondamenti,* Revisione ed adattamento a cura di M. Agosti, Addison Wesley Longman Italia Editoriale, Milano, 2001.

[4] P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone, *Basi di dati: concetti, linguaggi e architetture (seconda edizione),* McGraw-Hill Libri Italia, Milano, 1999.

[5] P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone, *Basi di dati - Modelli e linguaggi di interrogazione,* McGraw-Hill Libri Italia, Milano, 2002.

[6] J. Bowman, S. Emerson, M. Darnovsky, *The Practical SQL Handbook: Using Structured Query Language*, Third Edition, Addison-Wesley, 1996.

[7] C. J. Date, H. Darwen, *A Guide to the SQL Standard: A User's Guide to the Standard Database Language SQL*, Fourth Edition, Addison-Wesley, 1997.

[8] C. J. Date, *An Introduction to Database Systems*, Volume 1, Sixth Edition, Addison-Wesley, 1994.

[9] R. Elmasri, S. Navathe, *Fundamentals of Database Systems*, Third Edition, Addison-Wesley, August 1999.

[10] J. Melton, A. R. Simon, *Understanding the New SQL: A Complete Guide*, Morgan Kaufmann, 1993.

[11] J. D. Ullman, *Principles of Database and Knowledge: Base System*, Volume 1, Computer Science Press, 1988.

[12] S. Simkovics, *Enhancement of the ANSI SQL Implementation of PostgreSQL*, Department of Information Systems, Vienna University of Technology, November 29, 1998.

[13] A. Yu, J. Chen, The POSTGRES Group, *The Postgres95 User Manual*, University of California, Sept. 5, 1995.

[14] *Python Reference Manual*, May 20, 2004, Release 2.3.4.

[15] G. Clemente, F. Filira, M. Moro, *Sistemi Operativi: Architettura e Programmazione Concorrente,* Libreria Progetto, Padova, 2003.

[16] W. Stallings, *Operating Systems, Internal and Design Principles, 4/e,* Prentice Hall, Upper Saddle River, N.J.,2001.

[17] A. S. Tanenbaum, *Modern Operating Systems, 2/e,* Prentice Hall, Englewoods Cliffs, 2001.

[18] EGL Italia S.p.A., *Standard Electricity Supply Contract for the Year 2005*, Genova, 2004.

[19] EniPower Trading S.p.A, *Standard Electricity Supply Contract for the Year 2005*, Milano, 2004.

[20] Energia S.p.A., *Standard Electricity Supply Contract for the Year 2005*, Milano, 2004.

[21] Enel Energia S.p.A., *Standard Electricity Supply Contract for the Year 2005*, Roma, 2004.

[22] Edison Energia S.p.A., *Standard Electricity Supply Contract for the Year 2005*, Milano, 2004.

[23] J.L. Doob, *Stochastic processes,* New York, Wiley, 1953.

[24] G. Cariolaro, G. Pierobon, *Teoria della probabilità e dei processi aleatori,* Bologna, Patron, 1982.

[25] I.I. Gihman, A.V. Skorohod, *The theory of stochastic processes,* Berlin, Springer-Verlag, 1974.

[26] M. Loeve, *Probability theory,* New York, Van Nostrand, 1963.

[27] Yu.V. Prophorov, Yu.A. Rozanov, *Probability theory,* Berlin, Springer-Verlag, 1969.

[28] C.W. Burrill, *Measure, integration, and probability*, New York, McGraw-Hill, 1972.

[29] G. Scorza Dragoni, *Elementi di analisi matematica,* Padova, CEDAM, 1969, Vol. II.

[30] T. Di Fonzo, F. Lisi, *Complementi di Statistica Economica, Analisi delle Serie Storiche Univariate*, Cleup Editrice, Padova, 2000.

[31] D. Piccolo, *Analisi Moderna delle Serie Storiche*, Franco Angeli Editore, Milano Stampa Tipomonza, 1983.

[32] Makridakis, Wheelwright, *Interactive Forecasting Univariate and Multivariate Methods*, Second Edition, Holden Day, California, U.S.A., 1978.