Improving Performance in Integrated DSS with Object Oriented Modeling

ADELA BÂRA
bara.adela@ie.ase.ro, ba_adic@yahoo.com
VLAD DIACONITA
diaconita.vlad@csie.ase.ro
ION LUNGU
ion.lungu@ie.ase.ro
MANOLE VELICANU
manole.velicanu@ie.ase.ro
Faculty of Economics Cybernetics
Statistics and Informatics, Academy of Economic Studies
Bucharest
ROMANIA

Abstract: - The development cycle of a decision support system involves many resources, time and high cost and above all, the database schema used in the system is built only for some specific tasks. So, a relational database schema or a data warehouse prototype cannot be easily mapped and reused in multiple DSS projects. In this paper we propose an object-oriented (OO) approach and an OO database schema to accomplish the DSS implementation which require a multidimensional modeling at the conceptual level using fundamental concepts of OO (class, attribute, method, object, polymorphism, inheritance, hierarchy) to represent all multidimensional properties of a data warehouse, at both the structural and dynamic levels.

Key-Words: - Decision support systems (DSS), SQL optimization, integration**,** object-oriented modeling, object-oriented databases.

1 Introduction

In many cases the development of a decision support system involves the design and the implementation of a data warehouse. Different factors such as high costs and resources or an improper system design may lead to a failure in over 50% of these cases. One of the main factors of failure is that the data warehouse is built only for specific tasks or requirements and the expansion of the system cannot be achieved or achieved at very high costs.
In a research grant, we are developing a DSS system for a public institution in Romania. We tried to meet the requests from the managers of the institution, the main request being that the system was scalable, easily extendable. For the project lifecycle we applied the framework described in the book [8] (Lungu I, Bara A, 2007). The system has to gather data from an ERP system with modules like: financials, inventory, purchase, order management and production. So the need for a data warehouse is obvious and we had to choose between the two data warehouses solutions: stored data vs. virtual extraction. Because of time and costs and in order to test the DSS functionality we choose the second solution and we used virtual extraction. We designed a prototype [3,4], but because the ERP system was not yet fully implemented a major problem occurred: there will be many more changes in the structure of the organization and the impact of these changes may affect the DSS system. So, we needed to find a solution, and, based on our previous researches on Object Oriented modeling, we have chosen this type of modeling. We'll show in the following sections aspects regarding oriented implementation of a virtual data warehouse and some of the performance results of this implementation.

2 Decision support systems' modeling techniques

The main objective of DSS (Decision Support Systems) is to provide in real time representative information to every level of management and to gather, analyze, and integrate internal and external data into dynamic profiles and intuitive reports. In essence, the system must provide to each level a customized view that extracts information from disparate sources and summarizes it into meaningful indicators.

In order to provide aggregate information and indicators, DSS systems collect, transform and integrate data from various sources through Business Intelligence tools and technologies like: data warehouses, OLAP, data mining, analytic SQL reports.

But also, a major objective of DSS systems is to provide a friendly graphical interface and when this is customized for the individual manager, allows users to access corporate data and complements the managers' personal knowledge and provide quantitative diagnostics to monitor the progress of decisions.

In order to gather data from various sources and ERP systems implemented in an organization for different functional areas or modules such as: financials, inventory, purchase, order management, production we need to analyze and design the business model and strategic requests. This model have to be mapped on a logical model and physical model in the data warehouse and also used for extracting and presenting data through OLAP technology. These models are known as multidimensional models and basically they represent an extension of the relational model, an ER schema or a multidimensional view over facts.

Multidimensional models are classified in two major types:

- models that are an extension of ER model are based on a star schema and consist in the relationship between some dimensions and facts or measures;
- n-dimensional cube based models that use a multidimensional view over an individual situation or data.

Among ER extension models we can mention: Gray's model based on CUBE and ROLLUP operators with GROUP BY clause in SQL language that aggregate data over some attributes; Li and Wang's model or Gyssens and Lakshman's model that are an extension of relational schema [8]. But the most important model is Ralph Kimball's model in which he proposed the star schema as a representation of a n-dimensional cube. This schema contains a central fact table with many rows and measures in relation with the smaller tables called dimensions. Basically the joins between the fact table and the dimensions are similar with the ER joins. From this model later was developed the snow flake schema with joins between dimensions not only between fact and dimensions. Later it was developed a galaxy or a fact constellation schema with many fact tables in relation with many dimension tables.

In the cube based models' area we can mention Agrawal, Gupta and Sarawagi's model with minimal set of relational algebra's operators, but in which data structure is based on one or more n-dimensional cubes. In Agraval's vision these cubes are made of dimensions defined by name and values and cube's elements defined through a function that associates values to a n-dimensional row represented by the cells of the cube.

Also, in this category we can mention Cabibbo and Torlone's model or Blaschka's model [8] that defines an extension of ER technique called ME/R technique. In his vision the model contains dimensional levels, a 1: n fact relationship and a binary relationship called classification relationship between two hierarchical levels.

In Decision Support Systems the multidimensional model that is used, has to be able to overhear the business requests. All we need is a business vision over data structure so the star schema or the n-cube based models have to design and incorporate business aspects or demands not only the facts or the relationship between data.

Functional requirements tend to be a mixture of high- and low-level requirements—virtually a stream of consciousness from managers, customers, and the marketing team captured in serial form and placed into a document. This is just the first, early step along the path of getting a finalized, clear, unambiguous set of behavioral requirements that can realistically help to create a design from.

The executives usually request a synthetic view over facts and indicators and these key performance indicators are built from the entire organizational data or even from external data.

Another request is to provide a friendly graphical interface with advanced capabilities of slicing and dicing through data and easily get a new perspective over data by rotating dimensions and drill down or roll up over hierarchical levels. So we need a multidimensional model in which these operations can be made easily, in real time and that can it overhead the entire business model with relationship between dimensions, facts and hierarchies and it is based on the entire organizational data at operational level, tactical level and strategically level.

Based on these considerations we proposed in [8] an extension of the star or the constellation schema but with aggregate data and hierarchies in fact tables not only in dimension tables. The model is structured over three distinct levels and we can call it a pyramidal model with the following structure:

- *Organizational level* (or the base of the pyramid) – containing dimensions and facts with an organizational scope, at a general level, that shape and are common to the entire activities. Such dimensions can be: <time>, <zone>, <product>, <currency> and facts: production, purchasing etc. Data are at a detailed level with multiple hierarchies over each dimension table;

- *Departmental level* – containing dimensions and facts for the departmental levels of the organization and particular activities in these departments or field of interests, group by data marts or data centers. Such dimensions can be: <account>, <client>, <vendor> and facts: stocks, payments, sales etc. Data are at a detailed and aggregate level with specialized hierarchies over each dimension table;
- *Strategically level* – containing dimensions and facts derived from the base dimensions and facts, with specific elements for the strategic analysis, like <intercompany>, <plan>, <budget> and facts: cash-flow, kpi. Data are at an aggregate, synthetic level with specialized hierarchies over each dimension table.

The main characteristic of the model is that between the dimension tables and the facts from different levels of the architecture can be establish a relationship and also the fact tables can have hierarchies and class attributes that can be used for drill down or roll up.

Advantages of the model:

- Flexibility – new elements or objects like new dimensions or facts can easily be included in the model without affecting the existing architecture or remodeling the system and the loading process for a specific level can be made without refreshing the whole data;
- Real model of business requirements – the three level architecture is based on the real model of business requirements thus this model can be mapped on the each level of the pyramid;
- Performance in the drill-down or roll-up operations – because the dimensions and facts are separated at each level we can easily navigate through hierarchies from a level to another;
- Incremental development – the model can be build in stages and each stage can be validated and used before the next stage;
- EIS, MIS and DSS support – the bottom and middle levels can be used for design and realized a Management Information System (MIS) or a Decision Support System (DSS) because these systems can use the specific dimension and fact tables from these levels and the top level can be used for Executive Information Systems (EIS).

Disadvantages of the model:

- High complexity – because it is containing three different level the business model need to be careful analyzed and designed in order to identified the proper and suitable dimensions and facts and also the hierarchies at each level. An inadequate choice can have a major effect on the performance of the entire system;
- Moderate performance of the interrogation process – in order to perform a complex query the model need to establish many relationships and joins between the fact and dimension tables and this can reduce the performance of interrogation;
- Top-down and bottom-up development – In order to overhear the entire aspects of the business process we need to build the systems in two directions: first top-bottom to model the strategic requirements and second, bottom-up for validating and setting up the hierarchical flux of data.

3 Object oriented approach and extensions

We need a multidimensional model (MD) in which the OLAP functionalities and characteristics can be made easily, in real time and that can overhead the entire business model with relationship between dimensions, facts and hierarchies. This model has to be based on the entire organizational data at operational level, tactical level and also strategically level. Also, the DSS system has to be able to extend and introduce new different capabilities. A possible solution to these requests is the object oriented approach of multidimensional modeling.

The particularities and the characteristics of MD cannot be easily accomplished with the basic elements of OO modeling. Some extensions of these basic elements have to be made. Extending the research of (Luján-Mora S, Trujillo J, 2002) [6] we defined a set of object-oriented extensions that can be used for modeling the components and requirements of a data warehouse. We proposed an extension by means of stereotypes to the Unified Modeling Language (UML) for MD modeling [6,7]. In the following sections we'll show a possible implementation of these extensions in Oracle Database 10g using PL/SQL language.

The multidimensional modeling implies the definition of the properties at two main levels: *static* (structural) and *dynamic* (behavior) levels.

- *Static (structural) level*: we'll consider the components of a data warehouse as objects, so we'll have distinct classes and super-classes for dimensions and facts. It's generally good practice to organize your classes around key abstractions in the problem domain. The domain model is a first-cut class diagram that becomes the foundation of any software architecture. This makes the model more resilient in the face of change. Organizing the architecture around real-world abstractions makes the model more resilient in the face of changing requirements, as the requirements will usually change more frequently than the real world does. As it was proposed and described in [6] (Luján-Mora S, Trujillo J, 2002) and applied and developed in [8] (Lungu I, Bara A, 2007), in order to model the hierarchies of a dimension we need to have two types of classes: *base* and *level* classes to specify the order of a hierarchical level. Between these levels of a dimension there is a classification or an association relationship. Each class must have three types of attributes: OID (Identifying attribute) used for unique identification of an object, OD (Descriptive attribute) for dimension's values and PA (Parent attribute) for establishing the hierarchies. Facts are also considered as classes with the following attributes: OID (Identifying attribute), OD (Descriptive attribute) for descriptive values, FA (Fact attribute) for measures and CA (Formula attribute) for derived measures. The relationship between facts and dimensions is an aggregation relationship.

- *Dynamical (behavior) level*: in the book [8] (Lungu I, Bara A, 2007) we consider the functionality of a data warehouse in the terms of aggregation and drilling for OLAP (online analytical processing) functions such as drill-down/roll-up, slicing, dicing and rotations. These functions were modeled through sequence, interaction and activity UML diagrams.

In theory, every single aspect of the UML is potentially useful, but in practice, there never seems to be enough time to do modeling, analysis, and design. There's always pressure from management to jump to code, to start coding prematurely because progress on software projects tends to get measured by how much code exists.

In our case with the following UML extensions we can model the star schema of a data warehouse.

In the following table we'll present the stereotype defined in UML language [table 1:]:

Table 1. The UML stereotypes used for MD modeling

No	Stereotype	Description
1	<<Dimension>>	Dimension class
2	<<Fact>>	Fact class
3	<<Level>>	Level class
4	<<Base>>	Base class
5	<<OID>>	Identifying attribute for dimensions and facts
6	<<OD>>	Descriptive attribute for dimensions and facts
7	<<Parent ID>>	Parent attribute (only for dimension classes)
8	<<Dimension Attribute>>	Dimension attribute (only for dimension classes)
9	<<Fact Attribute>>	Fact attribute (only for fact classes) for measures
10	<<Formula Attribute>>	Formula attribute (only for fact classes) for derived measures
11	<<DAG>>	Directed Acyclic Graph for hierarchical relationship
12	<<Completeness>>	The completeness of a hierarchy

A Fragment of the implementation in a CASE tool of these extensions are detailed below:

Class:Base
Class:Level
Class:Fact
Class:Dimension
Attribute: OID
Attribute: OD
Attribute: Parent ID
Attribute: Dimension Attribute
Attribute: Fact Attribute
Attribute: Formula Attribute
Association: DAG
Association: Completeness
Logical Package:Dimension Package
[Class:Base]
Item =Class
Stereotype =Base
Metafile=&\stereotypes\color\base.wmf
ToolTip=Creates a Base Class\nBase Class

```
[Class:Level]
Item =Class
Stereotype = Level
Metafile=&\stereotypes\color\level.wmf
ToolTip=Creates a Level Class\nLevel Class
[Class:Fact]
Item =Class
Stereotype =Fact
Metafile=&\stereotypes\color\fact.wmf
ToolTip=Creates a Fact Class\nFact Class
[Class:Dimension]
Item =Class
Stereotype =Dimension
Metafile=&\stereotypes\color\dimension.wmf
ToolTip=Creates a Dimension Class\nDimension Class
[Attribute: OID]
Item=Attribute
Stereotype=OID
ToolTip=Creates a Dimension Attribute ID\nDimension Attribute ID
[Attribute: OD]
Item=Attribute
Stereotype=OD
ToolTip=Creates a Dimension DescriptionID\nDimension Description
[Attribute: Parent ID]
Item=Attribute
Stereotype=Parent ID
ToolTip=Creates a Parent Attribute ID\nParent Attribute ID
[Attribute: Dimension Attribute]
Item=Attribute
Stereotype=Dimension Attribute
ToolTip=Creates a Dimension Attribute\nDimension Attribute
[Attribute: Fact Attribute]
Item=Attribute
Stereotype=Fact Attribute
ToolTip=Creates a Fact Attribute\nFact Attribute
[Attribute: Formula Attribute]
Item=Attribute
Stereotype=Formula Attribute
ToolTip=Creates a Formula Attribute\nFormula Attribute
[Association: DAG]
Item=Association
Stereotype=DAG
ToolTip=Creates a DAG Association\nDAG Association
[Association: Completeness]
Item=Association
Stereotype=Completeness
ToolTip=Creates a Completeness Association\nCompleteness Association
[Logical Package:Dimension Package]
Item=Logical Package
Stereotype=Dimension Package
ToolTip=Creates a Dimension Package\nDimension Package
```

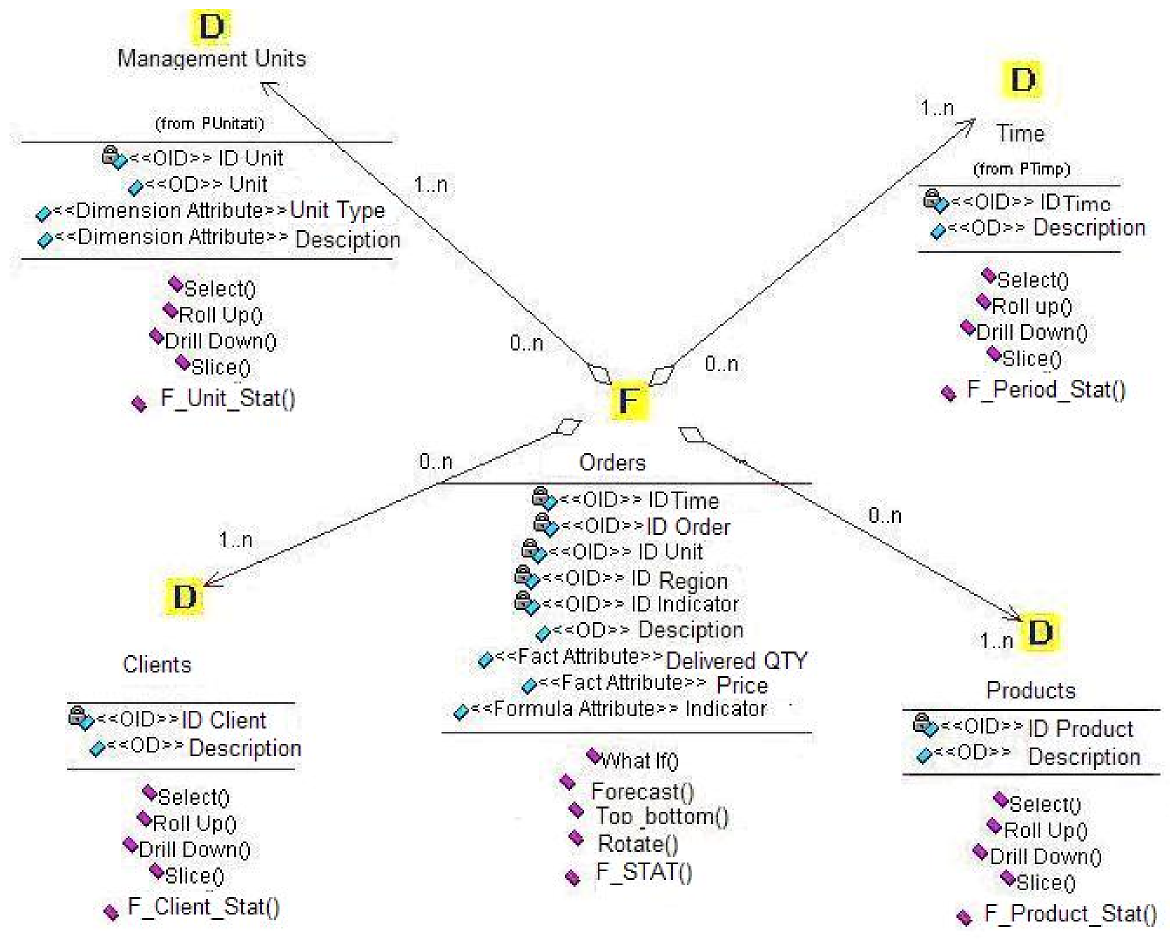

Fig 1: Class diagram for modeling the clients' orders analyses – Star Schema.

In our project we have to model the dimensions and the facts classes for the whole DSS system, but here we present only one star schema that models the clients' orders. Figure 1 presents this star schema.

4 Object oriented implementation

The multidimensional extensions can be implemented in an object-oriented database environment such as Oracle Database 10g which support the object type concepts that can be used to specify the MD classes and MD attributes from the structural model and MD functions from the dynamical model. An object type differs from native SQL data types in that it is user-defined, and it specifies both the underlying persistent data (attributes) and the related behaviours (methods) [10].

The object type is an object layer that can map the MD model over the database level, but data is still stored in columns and tables. Internally, statements about objects are still basically statements about relational tables and columns, and you can continue to work with relational data types and store data in relational tables. But we have the option to take advantage of object-oriented features too. Data persistency is assured by the object tables, where each row of the table corresponds to an instance of a class and the table columns are the class's attributes. Every row object in an object table has an associated logical object identifier. There can be use two types of object identifiers: a unique system-generated identifier of length 16 bytes for each row object assigned by default by Oracle in a hidden column, and primary-key based identifiers specified by the user and in which we have the advantage of enabling a more efficient and easier loading of the object table.

The structure of an object type consists of three components that can be used to implement the MD components, as it is shown below:

- *The class's name* – to uniquely identify the MD class within that schema;
- *The class's attributes* – to model the structure and state of the dimensions' and facts' attributes;
- *The class's methods* – to model the dynamical level which is the MD functions.

The methods are functions or procedures written in PL/SQL or Java and stored in the database, or written in a language such as C and stored externally. Oracle Database 10g supports four types of methods [8,10]:

- *The object type's constructor method* represents an implicitly defined method that is not tied to specific objects. Its name is the name of the object type and its parameters have the names and types of the object type's attributes. The constructor method is a function that returns the new object as its value.
- *Member methods* are functions or procedures that always have an implicit SELF parameter as its first parameter, whose type is the containing object type.
- *Static methods* are functions or procedures that do not have an implicit SELF parameter and can be invoked by qualifying the method with the type name (type_name.method). These static methods are useful for specifying user-defined constructors or cast methods. We'll use these methods to implement the MD functions.
- *Comparison methods* are used for comparing instances of objects.

The relationship between MD classes is implemented with a built-in data type called REF that encapsulates references to row objects of a specified object type.

In Oracle Database 10g a single inheritance model is supported: the subtype can be derived from only one parent type and a type inherits all the attributes and methods of its direct super type but it can add new attributes and methods, and it can override any of the inherited methods [10]. For inheritance there are to options in the object type defining area: NOT FINAL if you want it to have subtypes and FINAL (which is the default option) if no subtypes can be created for the type. This allows for backward compatibility. In the MD case we'll define the LEVEL classes with the NOT FINAL option and the BASE classes with the FINAL option.

For the DIMENSION classes, which are the super dimension classes, we'll use NOT INSTANTIABLE option that implies that there is no constructor (default or user-defined) for the object type. Thus, it is not possible to construct instances of this type. All the LEVEL and BASE classes will be instantiable. So, we'll define a super class – DIMENSION, for each dimension class from the MD model and which is NOT INSTANTIABLE, and many derived classes, LEVEL and BASE, which inherit the super class properties and which are INSTANTIABLE.

The FACT classes will be define as INSTANTIABLE and NOT FINAL because of the extended snowflake schema in which we can have relations between two facts, one as parent and one as derived from the parent.

In our case, as it is shown in the figure 1, we'll use the Oracle object type characteristics to implement the clients' orders star schema. The following sections present these steps:

4.1. Star schema implementation

For exemplification we'll present here only the classes of management unit dimension. We'll use a super type class to define the management unit space dimension, which has the DIMENSION stereotype and it is NOT INSTANTIABLE. We'll call it as UnitSpace_OT. For hierarchical levels of the dimension, as you can observe, there are two major hierarchies (fig. 2):

- *geographical locations (H1)*: *zone->region->country->location->management_unit*
- *organizational and management (H2)*: *division->sector->management_unit.*

So, the final object in both hierarchies is management_unit, which has the BASE stereotype and it will inherit all the attributes from the above objects, which have the LEVEL stereotype and are NOT FINAL. This object type will have two REFs, one for H1 and one for H2 hierarchies.

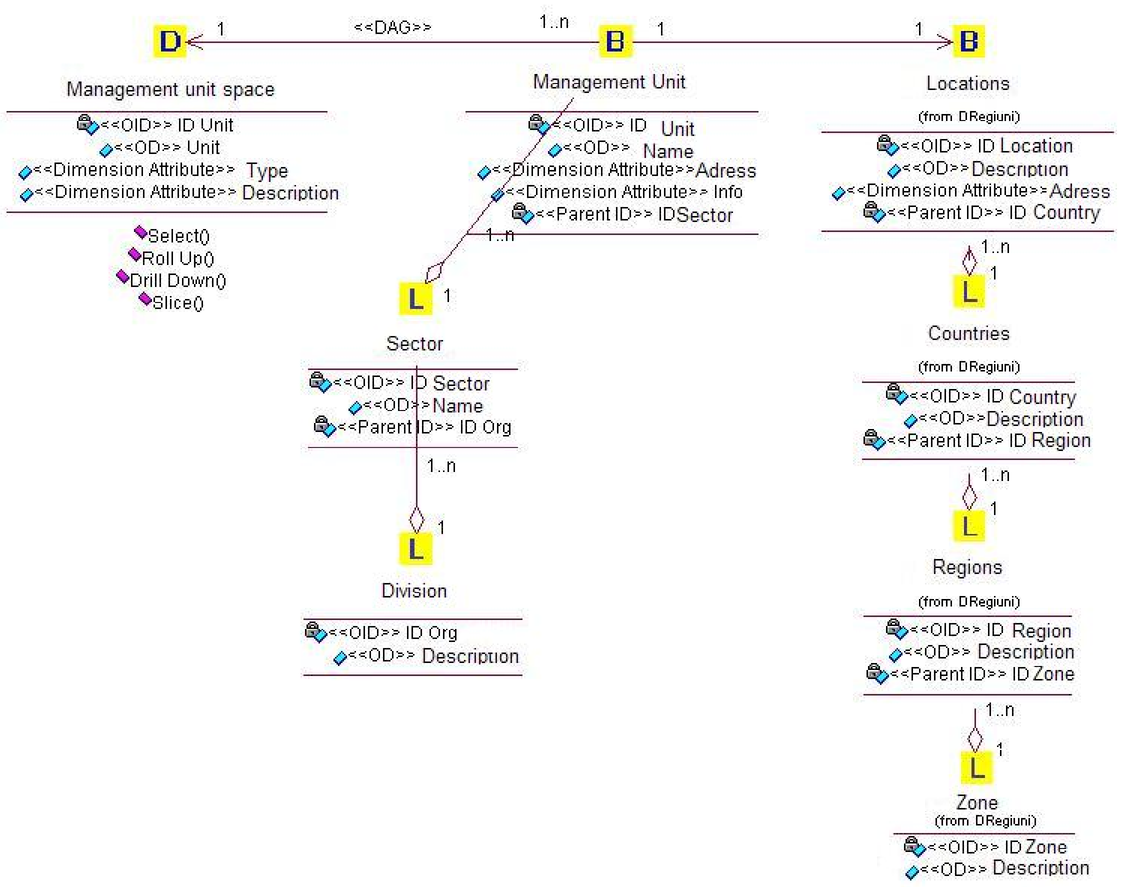

Fig 2: Class diagram for modeling the Management Unit dimension.

The script is shown below:
For the first hierarchy (H1):

create or replace type unitspace_ot as object (unitspace_id number, unitspace_desc varchar2 (50), unitspace_type varchar2 (50)) not instantiable not final;
*create or replace type zone_o under unitspace_ot (/*also add other attributes and methods*/) not final;*
*create or replace type region_o under unitspace_ot (zone ref zone_o /*also add other attributes and methods*/) not final;*
*create type country_o under unitspace_ot (region ref region_o /*also add other attributes and methods*/) not final;*
*create type location_o under unitspace_ot (country ref country_o /*also add other attributes and methods*/) not final;*
The second hierarchy (H2):
*create or replace type division_o under unitspace_ot (/*also add other attributes and methods*/) not final;*
*create type sector_o under unitspace_ot (division ref division_o /*also add other attributes and methods*/) not final;*
*create or replace type unit_o under unitspace_ot (sector ref sector_o, location ref location_o /*also add other attributes and methods*/) final;*

The orders' fact is implemented also as an object type INSTANTIABLE and NOT FINAL:

create or replace type orders_o as object
("type" varchar2(25) , “aproved" varchar2(25),
"status" varchar2(25), "client_id" number,
"loc_id" number, "id_order" number,
"data" date, "id_order_line" number ,
"id_shipping_loc" number, "id_shipping_unit" number,
"id_product" number, "ordered_qty" number,
"delivered_qty" number, "price" number,
"unit_id" number);

The methods of each MD object type are implemented as object type bodies in PL/SQL language that are similar with package bodies. For example the unit_o object type has the following body:

create or replace type body unit_o as
static function f_unit_stat(p_tab varchar2,p_gf varchar2, p_col_gf varchar2, p_col varchar2, p_val number) return number as
/ the function return the aggregate statistics from orders for a specific unit */*
v_tot number;

```
  text varchar2(255);
  begin
   text:= 'select '|| p_gf || ' ('||p_col_gf||') from '||p_tab||'
cd where '||p_col||'= '||p_val;
   execute immediate text into v_tot;
    return v_tot;
  end f_unit_stat;
/*others functions or procedures*/
end;
/
```

We can use this function to get different aggregate values for a specific unit, such as the total amount of quantity per unit or the average value per unit. Data persistency is assured with object tables that will store the instances of that object type, for example:

CREATE TABLE unit_t OF unit_o;

Through an ETL (extract, transform and load) process data is loaded into the object tables from the transactional tables of the ERP organizational system. This process can be implemented also through object types' methods or separately, as PL/SQL packages. Our recommendation is that the ETL process should be implemented separately from the object oriented implementation in order to assure the independency of the MD model.

4.2. Improving performance

Decision Support Systems usually work with large sets of data and require a short response time. If you consider not using analytical tools like OLAP and data warehousing techniques then you have to build your system through SQL queries and retrieve data directly from OLTP systems. In this case, the large amount of data in ERP systems may lead to an increase of responding time for the DSS. That's why you should consider phrasing the queries using the best optimization techniques.

When a SQL statement is executed on an Oracle database, the Oracle query optimizer determines the most efficient execution plan after considering many factors related to the objects referenced and the conditions specified in the query. The optimizer estimates the cost of each potential execution plan based on statistics in the data dictionary for the data distribution and storage characteristics of the tables, indexes, and partitions accessed by the statement and it evaluates the execution cost. This is an estimated value depending on resources used to execute the statement. The optimizer calculates the cost of access paths and joins orders based on the estimated computer resources, which includes I/O, CPU, and memory [1,2]. This evaluation is an important factor in the processing of any SQL statement and can greatly affect execution time.

As we study in our previous research [1,7] the execution cost of a query can be reduced by using techniques like indexes, hints and partitions. The current approach, with object oriented implementation can be used also to reduce the execution cost by avoid the multiple joins between the fact and the dimension tables. For example the static function *f_unit_stat* from the *unit_o* class can be use to retrieve the total order value for each unit:

```
(1) select unit_id, description,
unit_o.f_unit_stat('orders_t', 'SUM',
'delivered_qty*price', 'unit_id', unit_id) total
from unit_t
```

instead of using the join between the unit_t and the orders_t tables:

```
(2) select t.unit_id, t.description,
SUM(delivered_qty*price) total
from unit t, orders o
where t.unit_id=o.unit_id
```

We'll use for testing two types of tables: object tables (unit_t and orders_t) and relational tables (unit and orders). The cardinality of these tables is about 1500 records in orders and about 100 in unit tables.

We analyze the impact of calling the function in the SQL query in different situations, as we present in the following table:

No	Query	Cost
1	*select unit_id, description, unit_o.f_unit_stat('orders_t', 'SUM', 'delivered_qty*price', 'unit_id', unit_id) total from unit_t*	3
2	*select t.unit_id, t.description, SUM(delivered_qty*price) total from unit t, orders o where t.unit_id=o.unit_id*	11
3	Example (1) with an index on unit_id on both object tables	2
4	Example (2) with an index on unit_id on both relational tables	11
5	Example (1) with an index on unit_id on both relational tables with *use_nl* hint	8

Table 2. The execution costs of the queries

We analyze the execution cost of the function’s query, it has 6 units, but the SQL Tuning Advisor makes a recommendation: “consider collecting statistics for this table and indices”. We used DBMS_STATS package to collect statistics from both object and relational tables:

```
begin
dbms_stats.gather_table_stats (user, 'orders_t');
dbms_stats.gather_table_stats (user, ' orders');
end;
/
```

Then we re-run the queries and observe the execution plans; there is no change and the Advisor doesn’t make any recommendation.
By introducing these types of functions we have the following *advantages*:

- The function can be used in many reports and queries with different types of arguments, so the code is re-used and there is no need to build another query for each report;
- The amount of joins is reduced; the functions avoid the joins by searching the values in the fact table;
- Soft parsing is used for the function’s query execution instead of hard parsing in the case of another SQL query.

The main *disadvantage* of the model is that the function needs to open a cursor to execute the query which can lead to an increase of PGA resources if the fact table is too large. But the execution cost is insignificant and does not require a full table scan if an index is used on the corresponding foreign key attribute.

5 Conclusion

Building scalable and flexible Decision Support Systems involve important resources like: time, high-costs and human resources, efforts and it require a flexible modelling for business needs. One of these risks is the system design that stem from poor conceptualization of an enterprise’s true business needs before the systems is deployed and for every change in these requirements the prototype must be also revised. A solution for covering this risk is object oriented modelling of a data warehouse. This technique helps us to improve the designing phase and the development cycle and also we can re-use some parts of the prototype that it was implemented in an organization in order to design and implement another prototype in other organizational environments. The data warehouse can be implemented in an object oriented database environment, such as Oracle Database 10g, with the help of the object types and PL/SQL or Java languages. Object types offer a very efficient mode to map the MD objects to the relational database structure in which data is store. So, object types can store structured business data in its natural form in object tables and then allow applications, such as OLAP applications, to work in a multidimensional way using the object oriented properties and facilities.

References:

[1] Bâra A, Lungu I., Velicanu M, Diaconița V, Botha I - *Improving query performance in virtual data warehouses*, WSEAS TRANSACTIONS ON INFORMATION SCIENCE AND APPLICATIONS, May 2008, ISSN: 1790-0832;

[2] Bâra A, Lungu I., Velicanu M, Diaconița V, Botha I – Extracting data from virtual data warehouses – a practical approach of how to improve query performance, Proceedings of the 7th WSEAS International Conference on ARTIFICIAL INTELLIGENCE, KNOWLEDGE ENGINEERING and DATA BASES (AIKED '08), Cambridge UK, 20-22 feb. 2008, ISBN: 978-960-6766-41-1, ISSN, 1790-5109;

[3] Bologa A.R, Bologa R., Bâra A. - Technology Vs Business Needs In Business Intelligence Projects, Proceedings of the International Conference on e-Business (ICE-B 2008), 26-29 July, Porto, Portugal, 2008, ISBN 978-989-8111-58-6;

[4] Diaconita V., Botha I., Bâra A., Lungu I., Velicanu M. - Two Integration Flavors in Public Institutions, WSEAS TRANSACTIONS ON INFORMATION SCIENCE AND APPLICATIONS, May 2008, ISSN: 1790-0832;

[5] Diaconita V., Botha I., Bâra A., Lungu I., Velicanu M. – Portal oriented integration in public institutions, Proceedings of the 7th WSEAS International Conference on ARTIFICIAL INTELLIGENCE, KNOWLEDGE ENGINEERING and DATA BASES (AIKED '08), Cambridge UK, 20-22 feb. 2008, ISBN: 978-960-6766-41-1, ISSN, 1790-5109;

[6] Luján-Mora S., Trujillo J. - Extending UML for Multidimensional Modelling - Proceedings of the 5th International Conference on the Unified Modelling Language, 2002;

[7] Lungu I., Velicanu M., Bâra A., Diaconita V., Botha I. – Practices for designing and improving data extraction in a virtual data warehouses project, Proceedings of ICCCC 2008, International conference on computers, communications and control, Baile Felix, Oradea, Romania, 15-17 May 2008, pag 369-375, published in International Journal

of Computers, Communications and Control, vol 3, 2008, ISSN 1841-9836;
[8] Lungu I., Bâra A. - *Executive Information Systems*, Academy of Economic Studies Publisher, 2007, ISBN: 978-973-594-690-6;
[9] Moss L., Atre S. - *Business Intelligence Roadmap - the complete project lifecycle for decision-support applications*, Addison-Wesley Publisher, 2004, ISBN: 0-201-78420-3;
[10] ORACLE Corporation – Oracle Database Concepts 10g, www.oracle.com;