The Migration of a Conceptual Object Model COM (Conceptual Data Model CDM, Unified Modeling Language UML class diagram ...) to the Object Relational Database ORDB

Alae EL ALAMI [1], Mohamed BAHAJ [2]

1 PhD student, Department of Mathematics and Computer Science, University Hassan I, Settat, Morocco

2 Department of Mathematics and Computer Science, University Hassan I, Faculty of Sciences & Technologies Settat, Morocco

Abstract: This article proposes an approach to the migration of a conceptual object model, whatever its nature, to the physical schema of an object relational database, based on a meta-model that enriched the conceptual model with data and constraints necessary to establish the migration, then translate the meta-model to its physical object Relational schema in an automatic way without the interference of human factors, while also showing the difference between some mechanism of the object principle in different ORDBMS.

Keywords: conceptual data model, unified modeling language, object relational database

1. Introduction

Modeling has always been considered an essential practice in the life cycle of software, it is considered the most essential and critical step in the development and the creation of an application, to be made it is necessary to begin by transforming the conceptual model into logical model, then transforming the logical model into physical model.

So why choose to migrate to the ORDB and no other type of database, we have the development of Information and communication technologies ICT, these new technologies are looking for new techniques including the programming languages which are object-oriented, and to avoid losing the benefits that we provides the relational we are moving towards the ORDB which includes both object principles and at the same time benefit from the simplicity and the ease of the relational model.

Several approaches discuss the migration of UML to Extensible Markup Language XML files, Relational Database RDB, ORDB, Web Ontology Language OWL structures. Approaches based on the transformation of the class diagram of UML to object Relational mapping from the layer, divide the transformation in 3 steps, the first step is to define the UML classes and Relationship, the 2nd stage is to establish the object relational layer and 3rd stage carries the persistence layer of the object relational [1] [2].

An approach shows the transformation of class diagrams of UML to an XML document, applied to 5 steps, the first step is the representation of the diagram under mathematical formulas, the 2nd step is to encode chart and validate from a validation algorithm, 3rd step is to import the diagram to the XML schema definition XSD, the 4th step is to define the structure of the class diagrams, and finally the validation and storage [3].

Some methods also offers migrations that are based on an analysis and standardization, the first step in the migration is to establish an analysis of the UML diagrams, a second step is to normalize the class diagram in its logical data structure and behavior data and ultimately make the transition to the ORDB schema [4].

Several application modeling tools that allow among other to make UML diagram, and also can cover all stages of the life cycle of

(DOI: dx.doi.org/14.9831/1444-8939.2014/2-4/MAGNT.40)

software up to a step of generation of an automatic code in a specific language, which is part of the reverse engineering , but no such tools offers the possibility of creating a physical schema of an object relational database, this article is intended to fill this gap by providing a comprehensive approach to migration of a conceptual object model to a physical object relational model [5] [6].

One approach is based on the presentation of ORDB from a graphical model which is based on the mapping of a class diagram UML to its target ORDB schema from some of its elements [15]. Another approach shows the effectiveness of the use of models for the implementation of migration, having as starting point the class diagram of UML, these models become the main features of the software life cycle and its processes development, leading to a final product quality ensure[16].

Our approach will have as starting point a class diagram [7], a CDM Conceptual Data Model [8] or any object modeling, and extract the different patterns used in software engineering for represented classes and relationships between them, describe the responsibilities, behavior and the type of a set of objects, that by transforming the model into a meta-model [9]; the only step in the migration which will be made in a semi-automatic way; the second step is the transformation of the meta-model physical schema which is itself composed of two parts, types creation and tables creation, which will be done in an automatic way without the interference of the human factor.

2. The meta-model

The meta-model is the particular representation of a conceptual data model that characterizes the object modeling, which is considered the model of the model that will be flattened and indexed by classification through semantic enrichment [10].

The first stage of the migration it is the realization of the meta-model.

The meta-model is called New Data Model NDM which is a kind of table that describes the different classes derived from an object conceptual model with the necessary data for the implementation of a migration to ORDB.

The NDM is defined as a collection of classes NDM: = {C | C: = (cn, degree, cls, a, contributor)}

Cn =the name of the class.

Degree = first degree (the tables that contain PK) | 2nd degree (the tables that contain FK without PK).

Cls=aggregation, composition, association (association reflexive), inheritance, simple class (the class that does not belong to the other classifications).

Contributor =Constraint of primary passage of keys and realization of the foreign keys, where from the FKS shall have to respect the repository integrity to specify by the values of the primary keys at the origin of the functional independence (transition from a conceptual model towards the logical model).

A=attribute:={a | a := (an, t, tag, l, n, d)} (An :name of the attribute, T:type of the attribute, Tag: primary key(PK) | foreign key(FK),L: length of the attribute, N:if the attribute takes the parameter null, D:the default value of the attribute).

The realization of NDM starts by specifying the class name and its classification, then the list of attributes and specify the tag if it is a primary key.

According to the classification of each class a degree is automatically assigned, the degree helps to specify referential integrity, the tag designated as foreign key FK is assigned in an automated manner using the contributor, the degree and the classification.

Take the example of a class diagram that identifies the class structure of a system that includes all the objects and principles of the various relationships such as inheritance, aggregation that are most prevalent in the specifications of UML. This article is based on this example, other figures and tables will be drawn from this example.

(DOI: dx.doi.org/14.9831/1444-8939.2014/2-4/MAGNT.40)

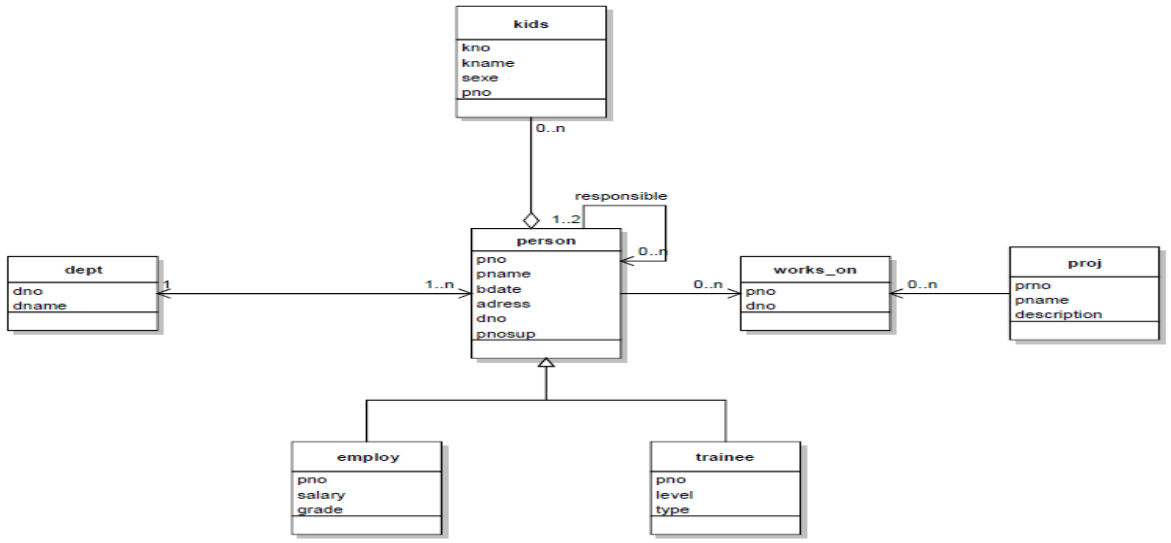


Fig. 1 . Example of a class diagram

Each class is referenced in other classes through a contribution table that links them, eliminating cardinality specifying that objects of a class are connected to other objects of another class, based on the same operating principle of the MMU (memory management unit) the management of operating systems.

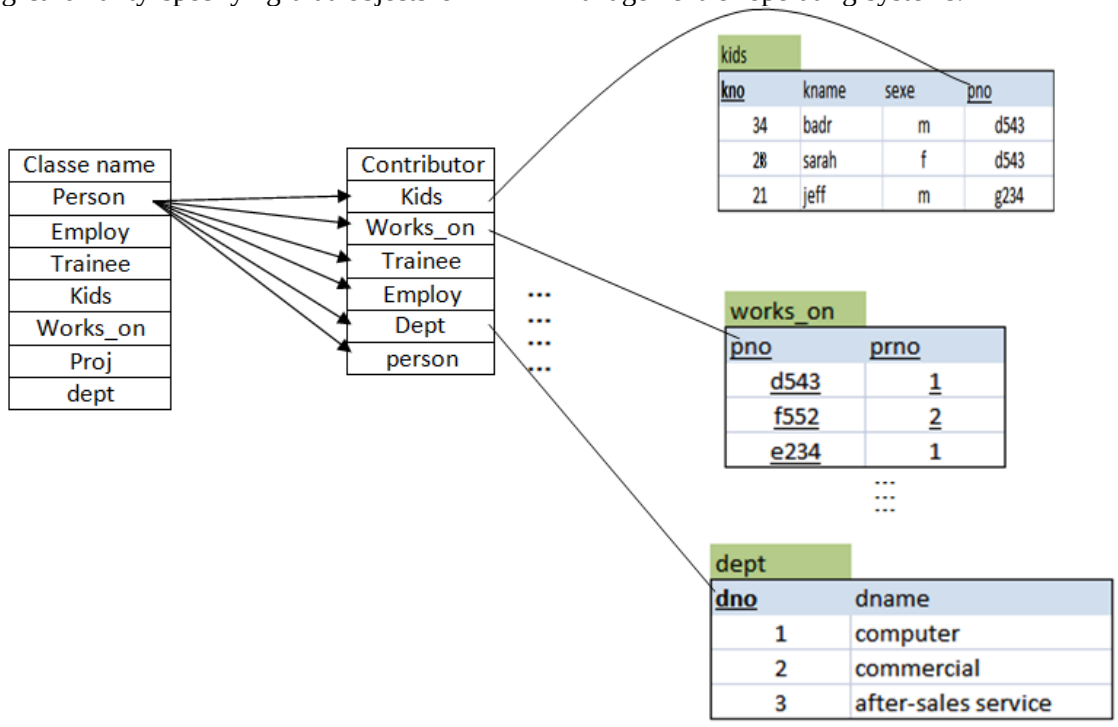


Fig. 2 . Example of functional dependency with a table of contribution

Example of NDM show in Table I generated from a class diagram.

(DOI: dx.doi.org/14.9831/1444-8939.2014/2-4/MAGNT.40)

Table 1. Result of the Generation of a NDM

Cn	Degre	Classification	Attribut						Contributor
			An	Type	ta	L	N	D	
Person	1st	inherBy	Pno	Varchar	PK		N		Kids
									Works_on
									Trainee
									Employ
			Pname	Varch			N		
			Bdate	Date			N		
			Adress	Varch		25	N		
			Dno	Int	F		N		Dept
			PnoSup	Varch	F		Y		Person
Trainee	2nd	Inherts	Pno	Varch	F		N		Person
			Level	Varch			N		
			Type	Varch			N		
Employ	2nd	Inherts	Pno	Varch	F		N		Person
			Salary	Int			Y		
			Grade	Varch			N		
Works_on	2nd	Association	Prno	Int	F		N		Proj
			Pno	Varch	F		N		Person
Dept	1st	Simple	Dno	Int	P		N		Person
			Dname	Varch			N		
Proj	1st	Simple	Prno	Int	P		N		Work_on
			Prname	Varch			N		
			Descripti	Varch		25	Y		
Kids	1st	Aggregation	Kno	Int	P		N		
			Kname	Varch			N		
			Sex	Char			N		
			Pno	Varch	F		N		Person

3. The transformation of the meta-model to ORDB

The second stage of the migration of a UML class diagram, it is the translation of the meta-model in ORDB, who is going to consist in two parts, the first part for the creation of the types and the second for the creation of tables.

(DOI: dx.doi.org/14.9831/1444-8939.2014/2-4/MAGNT.40)

The Creation of Types

+ Creation of the types defined in the NDM as aggregation.

An aggregation relationship establishes a separation between two classes, which is distinct from each other as a component or attached. Aggregation is a special type of association in

which the objects are combined to create a single more complex object.

+ Creation of the types defined in the NDM as composition.

A composition relationship is also a special type of association that represents an exhaustive relation. In other kind of, it is an aggregation which the lifetime of its complex object depends on the lifetime of the entire discriminator.

The main difference between an aggregation and a composition it is their life cycle. In an aggregation even if we destroy the class which there connects or a delete request is made, the history is kept. When it is a composition it depends on the class that connects there, if it is destroyed or a delete request is made, the compositions will be destroyed as it generates a loss of information.

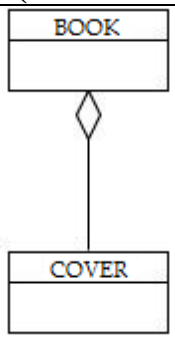
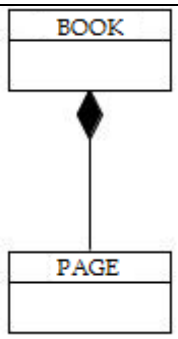
UML aggregation is schematized with an empty diamond, and composition with a solid diamond, noting that in most DBMS it will differentiate between composition and aggregation by the use of certain technology, for example the use of triggers. In the case of Oracle since version 8i is possible to make a differentiation between them, given its rich syntax, using nested table and collections of type Varray [11] [12] [13] [14].

Example of creating an aggregation and composition from different syntax

Table 2 . Creation OF AGGREGATION and composition

	Aggregation	Composition
Since Oracle 8i and later release	create type coverage as object id_coverage number, color varchar(10); create type book as object id_book number, title varchar(20), cover coverage; create table t_book of book (primary key (id_book)) nested table t_cover store as t_coverage; create table cover of coverage (primary key (id_cover));	create type pages as object id_page number, color varchar(10); CREATE OR REPLACE TYPE List_of_pages AS TABLE OF pages create type book as object id_book number, title varchar(20), the_pages list_of_pages; create table t_book of book (primary key (id_book)) nested table t_cover store as t_coverage;
SQL 99	create type book as object id_book number, title varchar(20), cover coverage; CREATE TYPE coverage AS (id_coverage INTEGER, color VARCHAR(10), **Has_cover REF(coverage)** **ARRAY[4]);** CREATE TABLE T_book OF book	

(DOI: dx.doi.org/14.9831/1444-8939.2014/2-4/MAGNT.40)

	(PRIMARY KEY (id_book));	
Schema		

+ Creation of the types defined in the NDM as association.

To create these types we keep the same name listed in the RDB and we add **_type** (concatenation).

+ Creation of composite types, those classes entering in collaboration with the aggregation and composition taking into account their classification, and other types whose classification in the NDM is simple.

+ The creation of the types defined in the NDM as an inheritance starting with the parent class and ending with the subclasses.

Creating Tables

The creation of tables is made by the typed classes and is classified in the NDM as inheritance (parent, subclasses), association, and simple class. The aggregations and compositions are included in the first-degree class that interacts with itself. All tables are created with the necessary constraints.

4 .Method of Creating and Naming Rule

To create the types we keep the same name that appears in the RDB and we add _type (concatenation).

- Syntax

CREATE [OR REPLACE] TYPE nameRDB_Type AS OBJECT
(column1 type1, column2 type2,...)

To create types that contain other types that represent aggregations, the type name that represents the aggregation remains the same and we add **_t** .

- Syntax

CREATE [OR REPLACE] TYPE nameRDB1_Type AS OBJECT
(column1 type1, column2 type2,...)
/
CREATE [OR REPLACE] TYPE nameRDB2_Type AS OBJECT
(column1 type1, column2 type2,nameRDB1_t set(nameRDB1_type),...)

For the creation of types with references, we add a **ref_** next to the name of the RDB with the keyword **REF** and the referenced type.

Observation: for reflexive relationships near many recordings [1-n] we concatenate the PK with the FK, and the side of a single record [1-1] we concatenate the FK with the PK.

- Syntax

CREATE [OR REPLACE] TYPE nameRDB1_Type AS OBJECT
(column1 type1, column2 type2,...)
/
CREATE [OR REPLACE] TYPE nameRDB2_Type AS OBJECT
(column1 type1, column2 type2,nameRDB1_t nameRDB1_type,...)
/
CREATE [OR REPLACE] TYPE nameRDB3_Type AS OBJECT
(column1 type1, column2 type2,ref_nameRDB2 REF nameRDB2_type,...)

(DOI: dx.doi.org/14.9831/1444-8939.2014/2-4/MAGNT.40)

For the creation of types that represent the inheritance, we add **Under** for the sub class and the keyword **not final** if the type has subtypes, and **final** if the type has no subtypes.

- Syntax

CREATE [OR REPLACE] TYPE nameRDB1_Type AS OBJECT
(column1 type1, column2 type2,...)
NOT FINAL
/
CREATE [OR REPLACE] TYPE nameRDB2_Type under nameRDB_type
(column1 type1, column2 type2,nameRDB1_t nameRDB1_type,...)
FINAL

Creating tables starts from typed classes. The table keeps the same name that appears in the RDB, and then we add the keyword **OF** and the type corresponding with the constraints captured in the NDM (PK constraint, reference constraint, not null constraint ...).

- Syntax:

CREATE TABLE [schema.]nameTable OF [schema.] nameType
[(column [DEFAULT expression]
[constraintOnLine [constraintOnLine]...
| constraintREFOnLine]
| { constraintOffline | constraintREFOffline }
[,column...])]
;

Flowchart for the creation of type aggregation

+ contri The number of contributor which is situated in the NDM

+tabAtt the table which contains the list of the attributes extracted from the RDB (Relational Database)

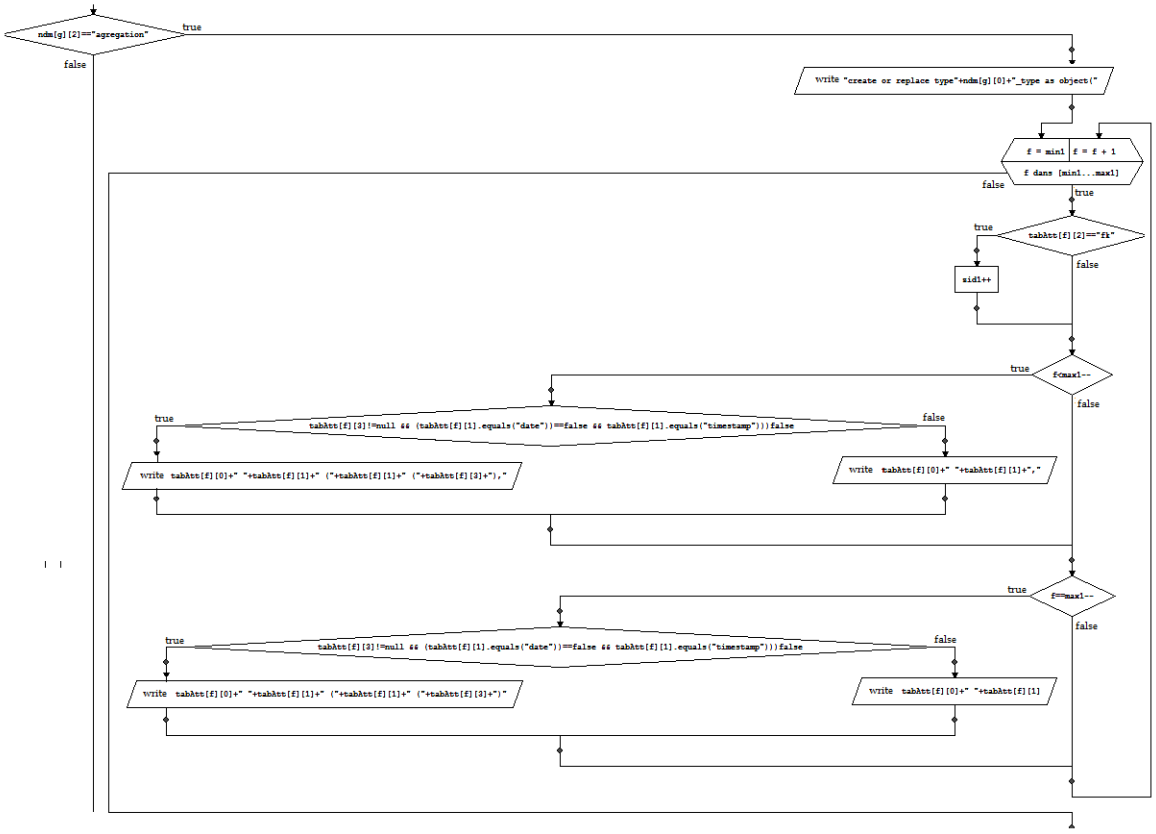


Fig. 3. Figure representing the creation of aggregation

Table.3. Final result of the migration

(DOI: dx.doi.org/14.9831/1444-8939.2014/2-4/MAGNT.40)

```
CREATE TYPE kids_type AS OBJECT
(kno int, kname varchar(20),sex char(1),pno varchar(20))
/
CREATE TYPE dept_type AS OBJECT
(dno int, dname varchar(20))
/
CREATE TYPE proj_type AS OBJECT
(prno int, prname varchar(20),description varchar(255))
/
CREATE TYPE person_type AS OBJECT
(pno varchar(10),pname varchar(20),
bdate date,address varchar(255),
dno int , pnosup varchar(20),
kids_t set(kids_type),
ref_dept ref (dept_type)scope dept,
ref_pno_pnosup set(ref(person_type)),
ref_pnosup_pno ref(person_type) scope person)
NOT FINAL
/
CREATE TYPE trainee_type UNDER person_type
(pno varchar(10), level varchar(20),type varchar(20))
FINAL
/
CREATE TYPE employe_type UNDER person_type
(pno varchar(10),salary int,grade varchar(20))
FINAL
/
CREATE TYPE work_on_type AS OBJECT
(prno int, pno varchar(20),
ref_proj set (ref(proj_type)),
ref_person set(ref(person_type)))
/
CREATE TABLE dept OF dept_type(
constraint pk_dept primary key(dno));

CREATE TABLE proj OF proj_type(
constraint pk_proj primary key(prno));

CREATE TABLE work_on OF work_on_type(
constraint refer_work_on_person ref_person references person,
constraint refer_work_on_proj ref_proj references proj);

CREATE TABLE person OF person_type(
constraint pk_person primary key(pno),
constraint refer_person ref_dept references
```

(DOI: dx.doi.org/14.9831/1444-8939.2014/2-4/MAGNT.40)

dept);

CREATE TABLE trainee OF trainee_type UNDER person;

CREATE TABLE employe OF employe_type UNDER person;

6. Conclusion

Nowadays there are several software for creating and editing the various diagrams of UML, that offers an almost complete set of tools for modeling and analysis for the specialist in the field in the design and development of software (rational rose, Objecteering power AMC ...) offers the possibility of reverse engineering which is to transform the conceptual model to code fragments with its inner workings and methods, but none of these software offers the possibility to achieve the transformation of the conceptual model to the physical object relational model.

In this paper we present an approach to the migration of a conceptual object model to its object-relational schema through semantic enrichment that includes all the specifications of object-relational model to arrive at the final object relational physical schema, the same approach is valid for the passage of a conceptual object model to the object-oriented physical schema.

References

1. María Fernanda Golobisky1 and Aldo Vecchietti1, Fundamentals for the Automation of Object-Relational Database Design. IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 2, May 2011 ISSN (Online): 1694-0814
2. María Fernanda Golobisky and Aldo Vecchietti . Mapping UML Class Diagrams into Object-Relational Schemas. Proc. of Argentine Symposium on Software Engineering (2005) 65-79 65
3. Mohamed BAHAJ,Noreddine GHERABI. Robust Representation for Conversion UML Class into XML Document using DOM. *International Journal of Computer Applications (0975 – 8887) Volume 33– No.9, November 2011*
4. Ming Wang, Using uml for object-relational database systems development: a framework. California State University, ming.wang@calstatela.edu
5. Terry Quatrani. Visual Modeling with Rational Rose 2000 and UML. Publisher: Addison Wesley Second Edition October 19, 1999 ISBN: 0-201-69961-3, 288 pages
6. Junit in action ,vincent massol with ted husted.mannig Greenwich (74° w.long.). isbn 1-9301 10-99-5
7. UML 2 ET MDE 1 .Franck Barbier 1 .Ingénierie des modèles avec études de cas. © Dunod, Paris, 2005 .ISBN 2 10 049526 7
8. jean luc baptiste . Merise guide pratique modélisation des données et des traitements, langage SQL . Editions ENI (11 mai 2009). ISBN-10: 2746048450
9. Alae EL ALAMI, Mohamed BAHAJ. Migration of the Relational Data Base (RDB) to the Object Relational Data Base (ORDB). World Academy of Science, Engineering and Technology International Journal of Computer, Information Science and Engineering Vol:8 No:1, 2014
10. Mohamed Bahaj & Alae Elalami. The Migration Of Data From A Relational Database (Rdb) To An Object Relational (Ordb) Database. Journal of Theoretical and Applied Information Technology .20th December 2013. Vol. 58 No.2 ISSN: 1992-8645 ,E-ISSN: 1817-3195
11. Oracle Database 12c Documentation http://www.oracle.com/technetwork/database/enterpriseedition/documentation/database-093888.html .
12. The API specification for version 6 of the Java™ Platform, Standard Edition: http://docs.oracle.com/javase/6/docs/api/

(DOI: dx.doi.org/14.9831/1444-8939.2014/2-4/MAGNT.40)

13. Allocating enough memory and solving OutOfMemoryErrors:http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2Ftasks%2Frunning_eclipse.htm
14. Stonebraker, Michael, Moore and Dorothy, "Object-Relational DBMSs: *The Next Great Wave "* (Morgan Kaufmann Series in Data Management Systems) ISBN: 1558603972.
15. T.R. Castro and S.N.A. Souza ; Graphic Logical Design for ORDB ; IEEE Latin America Transactions, Volume:11 Issue:4 ,juin 2013.
16. J. Cabot, R. Clarisó, D. Riera ;On the verification of UML/OCL class diagrams using constraint programming ; Journal of Systems and Software, Volume 93, July 2014, Pages 1-23

(DOI: dx.doi.org/14.9831/1444-8939.2014/2-4/MAGNT.40)