Migration of the Relational Data Base (RDB) to the Object Relational Data Base (ORDB)

Alae El Alami, Mohamed Bahaj

Abstract—This paper proposes an approach for translating an existing relational database (RDB) schema into ORDB. The transition is done with methods that can extract various functions from a RDB which is based on aggregations, associations between the various tables, and the reflexive relationships. These methods can extract even the inheritance knowing that no process of reverse engineering can know that it is an Inheritance; therefore, our approach exceeded all of the previous studies made for the transition from RDB to ORDB. In summation, the creation of the New Data Model (NDM) that stocks the RDB in a form of a structured table, and from the NDM we create our navigational model in order to simplify the implementation object from which we develop our different types. Through these types we precede to the last step, the creation of tables.

The step mentioned above does not require any human interference. All this is done automatically, and a prototype has already been created which proves the effectiveness of this approach.

Keywords—Relational databases, Object-relational databases, Semantic enrichment.

I. INTRODUCTION

MANY problems have emerged with the RDB [9]. We recall from them the reconstruction of complex objects split across relational tables is costly because it causes many joins. For this a solution has appeared, it is the ORDB [7]. Who has addressed most of these problems, we recall the use of reference facilitates the use of very large multimedia data by allowing them to be easily shared and costs less. Yet, the question that arises is how to achieve a migration from a RDB to an ORDB.

Several approaches have addressed the topic of migration which shows the transformation of the aggregation and associations from the conceptual model to the object relational model [1]. Based on the notion of collections of the Unified Modeling Language [8]. Other approaches are based on the creation of an ORDB from the UML [4]. In those authors have proposed the use of cardinalities to preserve and store the aggregations and compositions [10].

An approach takes all of the relational database and stores it in a structured table [2], [5], [6] that contains several parameters, tables, attributes, classifications, class types (abstract / concrete), the names of the relationships, class that interacts with the relationship and cardinalities in order to realize the schema translation.

Yet all approaches require the interference of the human factor in order to achieve migration, either on all the work, or on one or several parts. Therefore, in this paper we propose a method that requires no human interference.

Our approach is based on capturing metadata from a RDB that treats and stores it as a structured table that holds the information necessary for the migration.

This paper deals with the steps of the migration that are composed of 3 parts: the first is the implementation of the structured table to the NDM in Section II; the second part is the conversion from the NDM to the navigational model [3] in Section III; and the third part deals with the transformation of the Navigational Model (NavM) to the ORDB.

II. SEMANTIC ENRICHMENT OF RELATIONAL DATABASE: NEW DATA MODEL

A. Definition and Identification of the New Data Model

The NDM is a type of table describing the different classes extracted from a RDB with the data necessary for the realization of an 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) \mid 2nd degree (the tables that contain FK without PK).

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

Contributor=class list.

A=attribute:= $\{a \mid a := (an, t, tag, l, n, d)\}$ (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)

*Observation: treating cardinalities cannot help us since the transformation of the Conceptual Data Model (CDM) to the Logical Data Model (LDM) in the RDB has been treated for the migration of attributes.

1. Classification (cls)

For the classification, classes are composed of four parts:

• Aggregation is when the class interacts with a single class (the class itself may be the (first degree | 2nd degree)), not included in the classification and inheritance as the class has a FK;

Alae El Alami is a Phd in the Faculty of Science and Technology / Department of Mathematics and Computer Sciences, University Hassan I Settat, Morocco (e-mail: elalamialae@gmail.com).

Mohamed Bahaj is Professor the Faculty of Science and Technology / Department of Mathematics and Computer Sciences, University Hassan I Settat, Morocco (e-mail: mohamedbahaj@gmail.com).

- Associations: a class of 2nd degree which interacts with two or more classes; Special case: for reflexive relationships including their cardinalities in the CDM were « 1-n» «0-n » becomes an association with two FK with the name of the attribute of FK not found in any other table in the RDB, which includes a special treatment of the entities to know the referencing.
- Inheritance: When a class inherits from another class (no reverse approach can identify the inheritance).

For this we have developed a technique that is based on creating a table that will contain a maximum of possible probabilities of inheritance in accordance with the naming standards. Our table will consist of two columns: the first containing the mother class; and the second, the subclasses.

In order to know the inheritance we compare the names of the first-degree classes of the NDM with the first column of our created table. If a correspondence is found, we compare all the discovered classes that interact with our class within the NDM with the subclasses from the corresponding line of the table that contain possible inheritances. If an inheritance is found in this step, we precede to the last step of verification. We verify whether the PK value of the subclasses is equivalent to the PK value of the mother class in order to avoid a breach of the rules for naming tables.

So if there is a match we will extract an Inheritance, if not we will continue our treatment of classification knowing that the treatment of inheritance is the first step of the classification.

Here is an example that shows the most famous case of use of heritage shown in Table I.

THE CASE OF INHERITANCE THE MOST COMMON				
Inherited By	Inherits			
Person	Student			
	Teacher			
Animal	Cat			
	Dog			
	Horse			
Document	Book			
	Newspaper			
Account	Savings			
	Current			

TABLE I

The simple classes: the classes of the first degree, which are not within the classification of aggregation, inheritance and association.

2. Contributor

It defines the list of classes that the class starts interacting with themselves. The purpose of this part is to know whether it is an aggregation, an association, a single class or a class that inherits from the class starting, also for the creation of reference during the transition to the ORDB.

B. Generation of the NDM from a RDB

The translation of the RDB to the NDM is the first step of the migration into the ORDB

Consider that the RDB includes in Fig. 1. Example of NDM show in Table II generated from a RDB.

kids				proj				
<u>kno</u>	kname	sexe	pno	<u>prno</u>	pname	description		
34	badr	m	d543	1	Payment Management	integration	of a module in ar	n erp open so
23	sarah	f	d543	2	tramway casa	realization (of management o	omplette Tra
21	jeff	m	g234					
works_on				employ				
<u>pno</u>	prno			pno	salary	grade		
<u>d543</u>	1			<u>d543</u>	9000	engineer		
<u>f552</u>	2			<u>g234</u>	12000	director		
e234	1			f552	7000	commercial		
							-	
trainee				dept		_		
pno	level	type		<u>dno</u>	dname			
e234	master	hiring		1	computer			
			-	2	commercial			
				3	after-sales service			
						-		
person								
pno	pname	bdate	adress			<u>dno</u>	pnosup	
d543	alae	15/03/1987	residence ibn sina appt 3			1	g234	
e234	fouad	03/01/1987	rayhan imm 4 appt 5			2	d543	
g234	azar	24/04/1984	lotissemnt 34 rue des far a	ppt 6		1	null	
f552	jean	28/05/1975	rue la favette residence br	no imm maii	d appt 9	3	d543	

Fig. 1 The tables representing the relational database (the pks are underlined in bold ex, the fks are underlined ex)

International Journal of Information, Control and Computer Sciences ISSN: 2517-9942 Vol:8, No:1, 2014

Cn	Degre	Classification	Attribute				Contributor		
	•	-	An	Туре	tag	1	N	D	
Person	1 st	inherBy	Pno	Varchar	РК		Ν		Kids
									Works on
									Trainee
									Employ
			Pname	Varchar			N		Employ
			Bdate	Date			N		
			Adress	Varchar		255	N		
			Dno	Int	FK		Ν		Dept
			PnoSup	Varchar	FK		Y		Person
Trainee	2^{nd}	Inherts	Pno	Varchar	FK		Ν		Person
			Level	Varchar			Ν		
			Type	Varchar			Ν		
Employ	2^{nd}	Inherts	Pno	Varchar	FK		Ν		Person
			Salary	Int			Y		
			Grade	Varchar			Ν		
Works_on	2^{nd}	Association	Prno	Int	FK		Ν		Proj
			Pno	Varchar	FK		Ν		Person
Dept	1 st	Simple	Dno	Int	PK		Ν		Person
			Dname	Varchar			Ν		
Proj	1 st	Simple	Prno	Int	PK		Ν		Work_on
			Prname	Varchar			Ν		
			Description	Varchar		255	Y		
Kids	1 st	Aggregation	Kno	Int	PK		Ν		
			Kname	Varchar			Ν		
			Sex	Char			Ν		
			Pno	Varchar	FK		Ν		Person

TABLE II	
RESULT OF THE GENERATION OF A N	DN

Observation: VARCHAR is synonymous with VARCHAR2 but this usage may change in future versions (provided for backward compatibility only for Oracle datatypes [11]).

III. NAVM (NAVIGATIONAL MODEL)

After obtaining the NDM, we create the navigational model.

A. Definition and Objectives

- A model that plots the object implementation of a database while drawing up the navigation path between relations with the principle of referencing.
- Facilitates the transition towards the object by a set of rules for transposition.
- Promotes the Visualization of complex structures and possible navigation paths.
 - Why navigational?

The model introduces the logical links of the type REF (REF implementation is undetermined in the conceptual level).

The references (ref or REF) facilitate the navigation between objects.



The classes will be divided into two parts, the external classes and internal classes:

- + Internal classes are the classes classified as aggregation in the NDM.
- + External classes are the other classifications in the NDM.

Example



Fig. 2 The composition of the external and internal classes

Example of the NavM extracted from the UML:



Fig. 3 From the conceptual model to the navigational model

B. Transformation Rules



Fig. 4 Modeling inheritance in navigational

Inheritance: It follows the same principle of the class diagram in the UML, either for the parent class or subclass.

Association: The navigation link is simple and necessarily valued keeping attributes if there is an association with the class attributes.

The navigation links starts from the association class to the class that interacts with it (universal solution).



Fig. 5 How to model the association link

Aggregation: It becomes an internal class type object referenced by an attribute of assembly with a multiple link that can be valueless.

Simple class: For the simple class we must see the classification of the class that interacts with it in the NDM.

- + If the simple class interacts with an association, we will not need to trace the path of navigation because it is already done.
- + If a simple class interacts with another simple class, we have two navigation links:
- The first link starts of the class that contains a foreign key, which is a primary key in the other class, and is simple and cannot be valueless.
- -The second link starts from the other class, which is multiple and can be valueless.

C. The NDM Transformed into the NAVM

This stage of the migration is a part of reverse engineering to show the transition by reference and the elimination of joints, and plays a pivot role between the conceptual and implementation object.



Fig. 6 Transformation of the relational model to the navigational model

IV. TRANSLATION OF THE NDM TO AN ORDB

A. Approach for the Translation of the NDM into an ORDB

1. The Creation of Types

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

2. 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 are included in the first-degree class that interacts with itself. All tables are created with the necessary constraints.

B. 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,...)

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.

CREATE [OR REPLACE] TYPE nameRDB1_Type AS OBJECT

(column1 type1, column2 type2,...) NOT FINAL

Syntax

International Journal of Information, Control and Computer Sciences ISSN: 2517-9942 Vol:8, No:1, 2014

 nameRDB_type (column) type1, column2 type2,nameRDB1,t (ATT TYPE (and type (C), see Char(1), prover char(20)) (ATT TYPE (and type (C), see Char(1), prover char(20)) (ATT TYPE (and type (C), see Char(1), prover char(20)) (Constraint the type corresponding with the constraints captured in the NDM (PK constraint, reference constraint, not null constraint). Syntax: (CRATE TYPE (and type as 0.90ECT (do in, drame varchar(20)) (ConstraintChLine) (constraintChLine) (constraintChLine) (constraintChLine) (constraintChLine) (constraintChLine) (constraintChLine) (constraintREFORLine) (constraintStell) (constraintChLine) (constraintStell) (constraintStell)	CREATE [OR REPLACE] TYPE nameRDB2_Type under	TABLE IV
 (column1 ype], column2 ype_nameRDB1_1 mareRDB1_ype_n.) FINAI. 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. agtive of an the NDM (PK constraint, reference constraint, not constraint). CREATE TABLE [schema.]nameTable OF [schema.] (column [DEFAULT expression] [constraintOfLine [constraint(DnLine] [constraintOfLine [constraint(DnLine] [constraintOfLine [constraint(DnLine] [constraintOfLine] constraint(DnLine] [constraintOfLine] constraint(D	nameRDB_type	CREATE TYPE kids_type AS OBJECT
<pre>nameRDB1_type) FINAL Creating tables stars 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 RDB. (arl then we add the keyword OF and the type corresponding with the constraints captured in the RDB. (arl then we add the keyword OF and the type corresponding with the constraints captured in the RDB. (arl then we add the keyword OF and the type corresponding with the constraints captured in the RDB. (arl then we add the keyword OF and the type corresponding with the constraints captured in the RDB. (arl the we add the keyword OF and the type corresponding with the constraints captured in the RDB. (arl the RCB). and the we add the keyword OF and the type corresponding with the constraints constraintREFOOLine [constraint we relationships between different classes, associations, aggregations and as well as the inheritance. classes and respect the navigation links for best listings. This solution texceeds the exaising works as it generates an oRDB to an ORDB With a simple add practical methed to power of the object-relational as our solution to charts mether see how the navigation links is generates an ORDB without the interference of the human factor. This approach also provach. A forthcoming article Will present a prototype that spreading the database. Since and shows the ostation to train the purpoint, preper on types (mover charl(20), relations the ubsequent stage of the migration that affects the passage of the data from a RDB. We can trace the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article Will present a proto</pre>	(column1 type1, column2 type2,nameRDB1_t	(kno int, kname varchar(20),sex char(1),pno varchar(20))
Creating tables starts from typed classes. The table keeps the same name that appears in the RDE, and then we add the keyword OF and the type corresponding with the constraint, nor ull constraint). Syntax: CREATE TABLE [schema.]nameTable OF [schema.] nameType [constraintOnLine [constraintOnLine] [constraintOnLine] constraintOnLine] [constraintOnLine] constraintREFOITIne] [constraintOnLine] constraintREFOITIne] [constraintOnLine] [constraintOnLine] constraintREFOITIne] [constraintOnLine] [constraintonLine] [constraint	nameRDB1_type,)	/
Charle Three Berg Upper & Defacing the same and the mere add then we add the there are add then we add then we add then we add the add then we add the add the add the add then we add then we add the add the add the add the add then we add then we add the add	FINAL Creating tables starts from typed classes. The table keeps	
<pre>dot state name name name name name name name nam</pre>	the same name that appears in the RDB and then we add the	(dpo.int.dpame.varchar(20))
 (a) yind of the NDM (PK constraint, reference constraint, not null constraint). (Syntax: (REATE TABLE [schema]nameTable OF [schema] [constraintOnLine] [constraintOfLine [constraintOnLine] [constraintOfLine [constraintOfLine] [constraintOfLine] [constraintOfLine [constraintOfLine] [constraintOfTime [constraintOfLine] [constraintOfTime] constraintREFOfTime } [.constraintOfTime] constraintOfTime] constraintOfTime] constraintOfTime] constraintOfTime] constraintOfTime [constraintOfTime] constraintOfTime] c	keyword OF and the type corresponding with the constraints	
CREATE TYPE pro[_ype AS OBJECT mameType [(column [DEFAULT expression] [constraintOLIne [constraintREFOrFline] [constraintOLINE] [constraintOLINE [constraintREFORFline] [constraintOLINE] [constraint] [const	captured in the NDM (PK constraint, reference constraint, not	,
<pre>(promotint_prname_varchar(20),description varchar(25)) (columm [DEFAULT expression] [constraintOnLine [constraintOnLine] [constraintOfTime [constraintREFOfTime] [columm])] ;</pre>	null constraint).	CREATE TYPE proj_type AS OBJECT
 Syntax: CREATE TABLE [schema.]nameTable OF [schema.] nameType [(column [DEFAULT expression] [constraintOfLine] constraintREFOffine] [,column])]; ConstraintOfLine [constraintREFOffine] [,column])]; V.CONCLUSION The aforementioned work shows the steps of migrating from a RDB to an ORDB with a simple and practical method to capture the relationships between different classes, associations, aggregations and as well as the inheritance. Currently no approach has proposed such a solution to extract the inheritance from a RDB. We can trace the navigation inheritance is based on a normalized database to exploit the power of the object-relational as our solution of inheritance is based on a normalized database. If the promalized database is on tabke as a solution of inheritance is based on a normalized database. If the promalized database is on tabke as a solution of inheritance is based on a normalized database. If the promalized database is on used, we will have a simple class with an aggregation in place of the mother and sub class. This method excisting works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database obtained in cases where the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE proj Of prol_type[constraint pk_grop Of prol_type] constraint tefe_work_on_pron teferences proson. constraint tefe_work_on_proj efferences proj); CREATE TABLE proj Of prol_type[constraint tefe_work_on_proj efferences dept]; CREATE TABLE proj Of prol_type [constraint tefe_mork_on_proj efferences dept]; 		(prno int, prname varchar(20),description varchar(255))
CREATE TABLE [schema.]nameTable OF [schema.] nameType [column [DEFAULT expression] [constraintOnLine [constraintREFOnLine] [constraintOfTine [constraintREFOnTine] [constraintOfTine [constraintREFOTTine] [constraintOfTine [constraintOfTine] [constraintOfTine [constraintOfTine] [constraintOfTine] [constraintOfTine [constraintOfTine] [constraintO	• Syntax:	/
 nameType <pre>[(column [DEFAULT expression] [constraintOnLine] constraintOnLine] [constraintOnLine] constraintREFOfTine] [constraintOnTime [constraintREFOfTime] [constraintOnTime [constraintPA_constrainted [constrainte] [constraintIne [constraintSet [constraint refer_work_on_type [constraint pk_perion_type]]) [FiNAL [constraint pk_perio [constraint pk_perio [constraint pk_pro] [constraint pk_pro] [constraint pk_pro] [constraint pk_pro] [constraint pk_perio [constraint pk_pro] [constrain</pre>	CREATE TABLE [schema.]nameTable OF [schema.]	CREATE TYPE person, type AS OBJECT
<pre>[column [DEFAULT expression] [constraintREFOnLine]. [constraintORLine[constraintORLine] [constraintREFOnLine] [.constraintORLine[constraintREFOfTine] [.constraintORLine[constraintREFOfTine] [.constraintORLine] [.constraint PL_eron OF prol_type(constraint PL_eron OF prol_type (constraint PL_eron OF prol_type (constraint PL_eron of prol_ty</pre>	патеТуре	(pno varchar(10).pname varchar(20).
 [constraintOnLine [constraintREFOnfline] <pre>[constraintREFOnLine] <pr< td=""><td>[(column [DEFAULT expression]</td><td>bdate date,address varchar(255),</td></pr<></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre>	[(column [DEFAULT expression]	bdate date,address varchar(255),
[constraintREPOnDLINE] [constra	[constraintOnLine [constraintOnLine]	dno int , pnosup varchar(20),
[column])] [column])] ; V.CONCLUSION The aforementioned work shows the steps of migrating from a RDB to an ORDB with a simple and practical method to capture the relationships between different classes, associations, aggregations and as well as the inheritance. Currently no approach has proposed such a solution to extract the inheritance from a RDB. We can trace the navigational model to better see how the navigation in is made between classes and respect the navigation links for best listings. This method is done with a normalized database to exploit the power of the object-relational as our solution of inheritance is based on a normalized database. If the normalized database is not used, we will have a simple class with an aggregation in place of the mother and sub class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE proj OF proj_type(constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_get_proj references person; CREATE TABLE person OF person references person; CREATE TABLE person OF person; type UNDER person;	constraintREFONLINE]	kids_t set(kids_type),
<pre>[Pro</pre>	{ column 1)}	ref_dept ref (dept_type)scope dept,
 V.CONCLUSION The aforementioned work shows the steps of migrating from a RDB to an ORDB with a simple and practical method to capture the relationships between different classes, associations, aggregations and as well as the inheritance. Currently no approach has proposed such a solution to extract the inheritance from a RDB. We can trace the navigation links for best listings. This method is done with a normalized database to exploit the power of the object-relational as our solution of inheritance is based on a normalized database. If the power of the object-relational as our solution of inheritance is based on a normalized database. If the power of the object-relational as our solution of inheritance is based on a normalized database. If the power of the object-relational as our solution of inheritance is based on a normalized database. If the power of the object relation as solutes. This method is done will have a simple class with an aggregation in place of the mother and sub class. This solution exceeds the existing works as it generates an of the database obtained in cases where the user wants to manually update the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines for the data from a RDB to an ODB. CREATE TABLE dept OF proj_type(constraint pk_perjo primary key(grno)); CREATE TABLE person of person_type (constraint refer_work_on_rype(constraint refer_work_on_rype(constraint refer_work_on_rype(constraint refer_work_on_rype(constraint refer_person ref.gept references person; CREATE TABLE employe OF employe_type UNDER person; 	· ·	ref_pno_pnosup set(ref(person_type)),
V.CONCLUSIONThe aforementioned work shows the steps of migrating from a RDB to an ORDB with a simple and practical method to capture the relationships between different classes, associations, aggregations and as well as the inheritance. Currently no approach has proposed such a solution to extract the inheritance from a RDB. We can trace the navigation is made between classes and respect the navigation links for best listings. This method is done with a normalized database to exploit the power of the object-relational as our solution of inheritance is based on a normalized database. If the normalized database is not used, we will have a simple class with an aggregation in place of the mother and sub class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach las allows the possibility to make changes in the physical schema of the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach.CREATE TABLE dept OF dept_type(constraint pk_dept primary key(dno));CREATE TABLE proj OF proj_type(constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_person of person_type(constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_person ref_person ref_person;(REATE TABLE person of person_type(constraint refer_work_on_proj ref_proj references person, constraint refer_person ref_person ref_person;(REATE TABLE person of person_type(constraint refer_work_on_pro	,	
The aforementioned work shows the steps of migrating from a RDB to an ORDB with a simple and practical method to capture the relationships between different classes, associations, aggregations and as well as the inheritance. Currently no approach has proposed such a solution to extract the inheritance from a RDB. We can trace the navigation inks for best listings. This method is done with a normalized database to exploit the power of the object-relational as our solution of inheritance is based on a normalized database. If the normalized database is not used, we will have a simple class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE person OF pros_type(constraint tefe_work_on_proj ref_proson ref_person, type(constraint tefe_work_on_proj ref_proson ref_person; constraint tefe_work_on_proj ref_proson; constraint tefe_person ref_person; centerson; cref_dept references dept); CREATE TABLE employe OF employe_type UNDER person;	V.CONCLUSION	/
CREATE TYPE traines_type UNDER person_type (CREATE TYPE traines_type UNDER person_type (provarchar(10), level varchar(20), type varchar(20)) (FINAL (CREATE TYPE traines_type UNDER person_type (provarchar(10), level varchar(20), (provarchar(10), level varchar(20)) (FINAL (CREATE TYPE traines_type UNDER person_type (provarchar(10), level varchar(20)) (FINAL (CREATE TYPE traines_type UNDER person_type (provarchar(10), level varchar(20)) (FINAL (CREATE TYPE traines_type UNDER person_type (provarchar(20), (provarchar(10), level varchar(20)) (FINAL (CREATE TYPE traines_type UNDER person_type (provarchar(20), (provarcha	The aforementioned work shows the steps of migrating	
 (provarchar(10), level varchar(20), type varchar(20)) (Provarchar(10), type varchar(20), type varchar(20)) (Provarchar(10), type	from a RDB to an ORDB with a simple and practical method	CREATE TYPE trainee_type UNDER person_type
<pre>In the term of the definition of the term of the term of the term of the object-relational as our solution of inheritance is based on a normalized database to exploit the power of the object-relational as our solution of inheritance is based on a normalized database. If the normalized database is not used, we will have a simple class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB.</pre>	to capture the relationships between different classes.	(pno varchar(10), level varchar(20),type varchar(20))
Currently no approach has proposed such a solution to extract the inheritance from a RDB. We can trace the navigational model to better see how the navigation is made between classes and respect the navigation links for best listings. This method is done with a normalized database to exploit the power of the object-relational as our solution of inheritance is based on a normalized database. If the normalized database is not used, we will have a simple class with an aggregation in place of the mother and sub class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE proj OF proj_type(constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_pros or freque terences dept); CREATE TABLE person OF proj_type UNDER person; CREATE TABLE employe OF employe_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	associations, aggregations and as well as the inheritance.	FINAL /
the inheritance from a RDB. We can trace the navigational model to better see how the navigation is made between classes and respect the navigation links for best listings. This method is done with a normalized database to exploit the power of the object-relational as our solution of inheritance is based on a normalized database. If the normalized database is not used, we will have a simple class with an aggregation in place of the mother and sub class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE person OF work_on_type(constraint refer_work_on_proj ref_proj r	Currently no approach has proposed such a solution to extract	7
<pre>model to better see how the navigation is made between classes and respect the navigation links for best listings. This method is done with a normalized database to exploit the power of the object-relational as our solution of inheritance is based on a normalized database. If the normalized database is not used, we will have a simple class with an aggregation in place of the mother and sub class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE work_on_DF porj_type(constraint refer_work_on_peroj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_person ref_person ref_person; CREATE TABLE person OF person_type(constraint refer_person ref_person; CREATE TABLE employe OF employe_type UNDER person; CREATE TABLE employe OF employe_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;</pre>	the inheritance from a RDB. We can trace the navigational	CREATE TYPE employe_type UNDER person_type
classes and respect the navigation links for best listings. This method is done with a normalized database to exploit the power of the object-relational as our solution of inheritance is based on a normalized database. If the normalized database is not used, we will have a simple class with an aggregation in place of the mother and sub class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE proj OF proj_type(constraint pk_proj primary key(prno)); CREATE TABLE work_on_person ref_person references person, constraint refer_work_on_person ref_person references person; CREATE TABLE proson OF person_type(constraint refer_work_on_person ref_person; CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	model to better see how the navigation is made between	(pno varchar(10),salary int,grade varchar(20))
This method is done with a normalized database to exploit the power of the object-relational as our solution of inheritance is based on a normalized database. If the normalized database is not used, we will have a simple class with an aggregation in place of the mother and sub class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database obtained in cases where the user wants to manually update the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE proj OF proj_type(constraint refer_work_on_person ref_person references person, constraint refer_work_on_person ref_person references person, constraint refer_work_on_proj ref_proj references person; CREATE TABLE proj OF proj_type(constraint pk_person primary key(pno), constraint refer_person ref_dept references dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	classes and respect the navigation links for best listings.	FINAL
the power of the object-relational as our solution of inheritance is based on a normalized database. If the normalized database is not used, we will have a simple class with an aggregation in place of the mother and sub class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database obtained in cases where the user wants to manually update the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE proj OF proj_type(constraint refer_work_on_type(constraint refer_work_on_person ref_person references person, constraint refer_work_on_proj ref_proj references person, constraint refer_person primary key(pno), constraint refer_person ref_dept references dept); CREATE TABLE employe OF employe_type UNDER person; CREATE TABLE employe OF employe_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	This method is done with a normalized database to exploit	
inheritance is based on a normalized database. If the normalized database is not used, we will have a simple class with an aggregation in place of the mother and sub class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database obtained in cases where the user wants to manually update the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE work_on OF work_on_type(constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_mork_on_proj ref_proj references person, constraint refer_person ref_dept references dept); CREATE TABLE employe OF employe_type UNDER person; CREATE TABLE employe OF employe_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	the power of the object-relational as our solution of	CREATE TYPE work on type AS OBJECT
normalized database is not used, we will have a simple class with an aggregation in place of the mother and sub class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database obtained in cases where the user wants to manually update the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. 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 dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	inheritance is based on a normalized database. If the	(prno int, pno varchar(20),
<pre>with an aggregation in place of the monter and sub class. This solution exceeds the existing works as it generates an ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database obtained in cases where the user wants to manually update the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB.</pre> CREATE TABLE proj OF proj_type(constraint refer_work_on_person ref_person references person, constraint refer_work_on_person ref_person references person, constraint refer_person of person_type(constraint refer_person of person_type(constraint refer_person of person_type(constraint refer_person ref_dept references dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	normalized database is not used, we will have a simple class	ref_proj set (ref(proj_type)),
 ORDB without the interference of the human factor. This approach also allows the possibility to make changes in the physical schema of the database obtained in cases where the user wants to manually update the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE work_on OF work_on_type(constraint refer_work_on_person ref_person references person, constraint refer_work_on_proj references proj); CREATE TABLE person OF person_type(constraint pk_person primary key(pno), constraint refer_person ref_dept references dept); CREATE TABLE trainee OF trainee_type UNDER person; 	This solution exceeds the existing works as it generates an	ref_person set(ref(person_type)))
CREATE TABLE dept OF dept_type(constraint pk_dept primary key(dno));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 pk_person primary key(pno), constraint refer_person ref_dept references dept);CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	ORDB without the interference of the human factor. This	/
 approach more the position of the position of the database obtained in cases where the user wants to manually update the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE work_on_OF work_on_type(constraint refer_work_on_person references person, constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_person of person_type(constraint pk_person primary key(pno), constraint refer_person ref_dept references dept); CREATE TABLE trainee OF trainee_type UNDER person; 	approach also allows the possibility to make changes in the	CREATE TABLE dent OF dent type(
 user wants to manually update the database. Since the work is done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE work_on_OF work_on_type(constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint refer_person OF person_type(constraint pk_person primary key(pno), constraint refer_person ref_dept references dept); CREATE TABLE trainee OF trainee_type UNDER person; 	physical schema of the database obtained in cases where the	constraint pk_dept primary key(dno));
 done in console mode, a prototype was created to prove the effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE work_on OF work_on_type(constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint pk_person OF person_type(constraint pk_person primary key(pno), constraint refer_person ref_dept references dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person; 	user wants to manually update the database. Since the work is	
effectiveness of this approach. A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. 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 refer_person ref_dept references dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	done in console mode, a prototype was created to prove the	
A forthcoming article will present a prototype that examines the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE work_on_OF work_on_type(constraint refer_work_on_proj ref_proj references person, constraint refer_work_on_proj ref_proj references person, constraint pk_person OF person_type(constraint pk_person OF person_type(constraint pk_person primary key(pno), constraint refer_person ref_dept references dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	effectiveness of this approach.	CREATE TABLE proj OF proj_type(
the subsequent stage of the migration that affects the passage of the data from a RDB to an ODB. CREATE TABLE work_on_OF work_on_type(constraint refer_work_on_proj ref_proj 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 dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	A forthcoming article will present a prototype that examines	constraint pk_proj primary key(prnoj);
of the data from a RDB to an ODB. 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 dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	the subsequent stage of the migration that affects the passage	CREATE TABLE work on OF work on type(
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 dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;	of the data from a RDB to an ODB.	constraint refer_work_on_person ref_person references person,
CREATE TABLE person OF person_type(constraint pk_person primary key(pno), constraint refer_person ref_dept references dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER 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 dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;		
constraint pk_person primary key(pno), constraint refer_person ref_dept references dept); CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;		CREATE TABLE person OF person_type(
CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;		constraint pk_person primary key(pno),
CREATE TABLE trainee OF trainee_type UNDER person; CREATE TABLE employe OF employe_type UNDER person;		
CREATE TABLE employe OF employe_type UNDER person;		CREATE TABLE trainee OF trainee_type UNDER person;
		CREATE TABLE employe OF employe_type UNDER person;

International Journal of Information, Control and Computer Sciences ISSN: 2517-9942 Vol:8, No:1, 2014

References

- G. O. Young, "Synthetic Structure of Industrial Plastics (Book Style with Paper Title and Editor)," in Plastics, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.
- [2] Abdelsalam Maatuk, Akhtar Ali, Nick Rossiter: Semantic Enrichment: The First Phase of Relational Database Migration. In CIS2E '08, 6pp, Bridgeport, USA, 2008.
- [3] Nora Koch, Hubert Baumeister, Rolf Hennicker and Luis Mandel.: Extending UML to Model Navigation and Presentation in Web Applications. In Proc. of the Workshop Modeling Web Applications in the UML, UML'00, 2000.
- [4] Ming Wang.: Using UML for Object-Relational Database Systems Development: A Framework. Issues in Information Systems, VOL 9, No. 2, 2008.
- [5] Maatuk, A., Ali, M. A. and Rossiter, N.: An Integrated Approach to Relational Database Migration. In IC-ICT '08, pp. 16, Bannu, Pakistan, 2008.
- [6] Abdelsalam Maatuk, M. Akhtar Ali, Nick Rossiter.: Converting Relational Databases into Object-relational Databases. in JOT, vol. 9, no. 2, pages 145-161, 2010.
- [7] Stonebraker, Michael, Moore and Dorothy. Object-Relational DBMSs: The Next Great Wave (Morgan Kaufmann Series in Data Management Systems) ISBN: 1558603972.
- [8] K. Barclay and J. Savage-Object-Oriented Design with UML and Java,2004, ISBN 0 7506 6098 8.
- [9] S. Sumathi, S. Esakkirajan —Fundamentals of Relational Database Management Systems, 2007, ISBN 978-3-540-48397-7.
- [10] J.W. Rahayu, E. Pardede and D. Taniar, On Using Collection for Aggregation and Association Relationships in XML Object-Relational Storage. ACM Symposium on Applied Computing, Nicosia, Cyprus, 2004.
- [11] http://www.oracle.com/technetwork/database/enterpriseedition/documentation/index.html.