JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2009

Vol. 9, No. 2, March–April 2010

Converting Relational Databases into Object-relational Databases

Abdelsalam Maatuk
M. Akhtar Ali
Nick Rossiter
School of Computing, Engineering & Information Sciences,
Northumbria University, Newcastle upon Tyne, UK

This paper proposes an approach for migrating existing Relational DataBases (RDBs) into Object-Relational DataBases (ORDBs). The approach is superior to existing proposals as it can generate not only the target schema but also the data instances. The solution takes an existing RDB as input, enriches its metadata representation with required semantics, and generates an enhanced canonical data model, which captures essential characteristics of the target ORDB, and is suitable for migration. A prototype has been developed, which migrates successfully RDBs into ORDBs (Oracle 11_g) based on the canonical model. The experimental results were very encouraging, demonstrating that the proposed approach is feasible, efficient and correct.

1 INTRODUCTION

Relational DataBases (RDBs) have been applied in a number of areas and accepted as a solution for storing and retrieving data due to their maturity. Most traditional database applications are based on traditional Database Management Systems (DBMSs), i.e., Relational DBMSs (RDBMSs) as they have been quite successful in handling simple but large amount of data. The drawbacks of such RDBMSs in supporting complex data structures, user-defined data types and data persistence required by Object-Oriented (OO) and e-commence applications have led to the development of object-based database systems. Object-Oriented DataBases (OODBs) [6] and Object-Relational DataBases (ORDBs) [18], which support various diverse OO concepts, have been proposed in order to fulfil the demands of newer and more complex applications. Consequently, new DBMSs have started to emerge in the market, providing more functionality and flexibility. Since the majority of data are currently stored in RDBMSs, it is expected that conversion of RDBs into the database technology that have emerged recently assumes special significance.

ORDBs are showing potential, because they have a relational base and append object features. The main goal of their design was to incorporate both robust transaction and performance management features of RDBs, and flexibility, scalability and support for rich data types, which are features of OO models. Some of these features are defined in the SQL3 [4] and SQL4 [16] standards, e.g., user-defied types and inheritance. Together with handling simple data types, ORDBs can handle multimedia data types. Developers can work with the tabular structure and DDL of RDBs with a better support for complex data and object management. Moreover, the development of ORDBs was triggered by the growth of object programming languages to avoid the mismatch between these languages and DBMSs. For these reasons, the migration of RDB into its extension ORDB is necessary.

Most existing methods for converting RDBs into ORDBs focus on schema transformation, but in the context of database design. A large body of research has concentrated on transforming well-known conceptual models such as an Extended Entity Relationship (EER) and UML into ORDB schemas [8, 13, 17, 19, 9, 15, 3, 7, 14]. Another group of methods and tools exist to enable non-traditional applications to share data with object schema [21], and to view and publish RDB data in XML [2, 5]. However, none of the existing studies can be considered as a method for migrating an RDB into an ORDB on both levels: schema and data conversion. Typed data, the most important part of databases, are to be converted and utilized within the new environment.

In this paper, we propose a method for migrating RDBs into ORDBs. The method comprises three basic steps. In the first step, the method produces a Canonical Data Model (CDM), which is enriched with RDB integrity constraints and data semantics that may not have been explicitly expressed in its metadata. The CDM so obtained is translated into an ORDB schema in the second step. Data conversion is the third step, in which RDB data are converted into their equivalents in the ORDB environment. The solution is superior to existing approaches as it can automatically generate the ORDB schema as well as data instances. We use SQL4 ORDB so that the approach is independent of a particular product. A prototype has been implemented to demonstrate the migration process and provide proof of concept. As Oracle 11_g supports most of the data types defined by SQL4, we have used it for implementation. An experimental study has been conducted to evaluate the prototype by checking the results it provides regarding the correctness and completeness of the solution and its concepts.

This paper is structured as follows. An overview of the related work is presented in Section 2. Section 3 describes how an existing RDB metadata is enriched in the form of CDM. The translation of CDM into ORDB schema is presented in Section 4. Section 5 explains the conversion of RDB data into an ORDB. Section 6 evaluates the method and reviews its results, and Section 7 concludes the paper.

2 RELATED WORK

Transforming conceptual models (e.g., EER, UML class diagrams) into ORDB have been studied extensively over the past ten years [18, 8, 13, 17, 19, 9, 15, 1, 3, 7, 14]. A common finding from these studies is that the logical structure of an ORDB schema is achieved by creating object-types from UML diagrams. Tables are created based on the pre-defined object-types. An association relationship is mapped using **ref** or a collection of **ref**s depending on the multiplicity of the association. Multi-valued attributes are defined using arrays/nested tables. Inheritance is defined using FKs or **ref** types in Oracle 8_i and the **under** clause in Oracle 9_i/SQL3 [9].

A method of mapping and preserving collection semantics into an ORDB has recently been proposed [15]. The method transforms UML conceptual aggregation and association relationships into ORDB using **row** and **multiset** provided by SQL4 [16]. More recent work has focused on mapping UML aggregation/composition relationships into ORDBs [9, 3]. Urban et al. described essential rules for converting UML class diagrams into ORDB schemas, using triggers to preserve inverse relationships between objects for bi-directional relationships [19]. Marcos et al. proposed new UML stereotype extensions for an ORDB design, focusing on aggregation and composition relationships [8, 9]. Urban and Dietrich presented an approach using UML diagrams as a foundation for analysis, transforming them into RDB/OODB/ORDB schemas [20]. Grant et al. have compared and evaluated most of the above and others similar proposals. Their analysis might aid in the standardisation of these techniques and the development of a tool that could support in ORDBs design [7]. Although most ORDB concepts are present in these proposals, their focus has been on the design of ORDBs rather than on migration. However, if a migration process uses a conceptual model as an intermediate stage, then these proposals could be useful in schema translation.

3 SEMANTIC ENRICHMENT OF RELATIONAL DATABASE

The semantic enrichment of an RBD involves the extraction of its data semantics, to be enriched and converted into a much enhanced CDM. For this task, we have applied our approach [11] for semantically enriching RDBs. The process starts by extracting the basic metadata information about an existing RDB, including relation names and attribute properties (i.e., attribute names, data types, length, default values, and whether the attribute is nullable), and Primary Keys (PKs), Foreign Keys (FKs) and Unique Keys (UKs). We assume that data dependencies are represented by PKs and FKs as for each FK value there is an existing, matched PK value, which can be considered as a value reference. To get the best results, it is preferable that the process is applied to a schema in 3rd Normal Form (3NF). A relation that is not in 3NF may have redundant data, update anomalies problem or no clear semantics of whether it represents one real-world entity or relationship type. These problems may affect the real world meaning materialized in object-relational models. The next step is to identify the CDM constructs based on a classification of relations, attributes and relationships, which may be performed through data access. Lastly, the CDM structure is generated.

Definition of CDM: The CDM is defined as a set of classes: $CDM := \{C \mid C := \langle cn, cls, abs, A_{cdm}, Rel, UK \rangle\}$, where each class C has a name cn, is given a classification cls, and whether or not it is abstract abs. Each C has a set of attributes A_{cdm}, a set of relationships Rel, and a set of unique keys UK.

Classification (cls): Classification divides classes into the three categories:

1. Main classes (classes forming base types in the target database)
   - Regular Strong Class (RST): a class whose PK is not composed of any FKs.
   - Secondary Strong Class (SST): an inherited RST class.
   - Sub-class (SUB): a class that inherits another super-class, but is not inherited by other sub-classes.
   - Secondary Sub-class (SSC): a sub-class that is inherited by other sub-classes.
   - Secondary Relationship Class (SRC): a referenced RRC class, an M:N relationship class with attributes, or n-ary relationships where n>2.
   - Regular Component Class (RCC): a weak class that participates in a relationship with other classes rather than its parent class.
2. Component classes (classes representing multi-valued/composite attributes)
   - Multi-valued Attribute Class (MAC): a class that represents a multi-valued attribute.
   - Composite Attribute Class (CAC): a class that represents a composite attribute.
3. Relationship class (a class describing an M:N relationship between two classes)
   - Regular Relationship Class (RRC): an M:N relationship class without attributes.

Abstraction (abs): A super-class is abstract (i.e., $abs :=$ `true`) when all of its objects are members of its sub-type objects. Instances of an abstract type cannot appear in the database extension, but are subsumed into instances of its sub-types.

Attributes (A_{cdm}): A class C has a set of attributes A_{cdm}. $A_{cdm} := \{a \mid a := \langle a_n, t, tag, l, n, d\rangle\}$, where each attribute a has a name a_n, data type t and a tag, which classifies a as a non-key 'NK', 'PK', 'FK' or both PK and FK 'PF' attribute. Each a can have a length l and may have a default value d whereas n indicates whether or not a is nullable ('y'|'n').

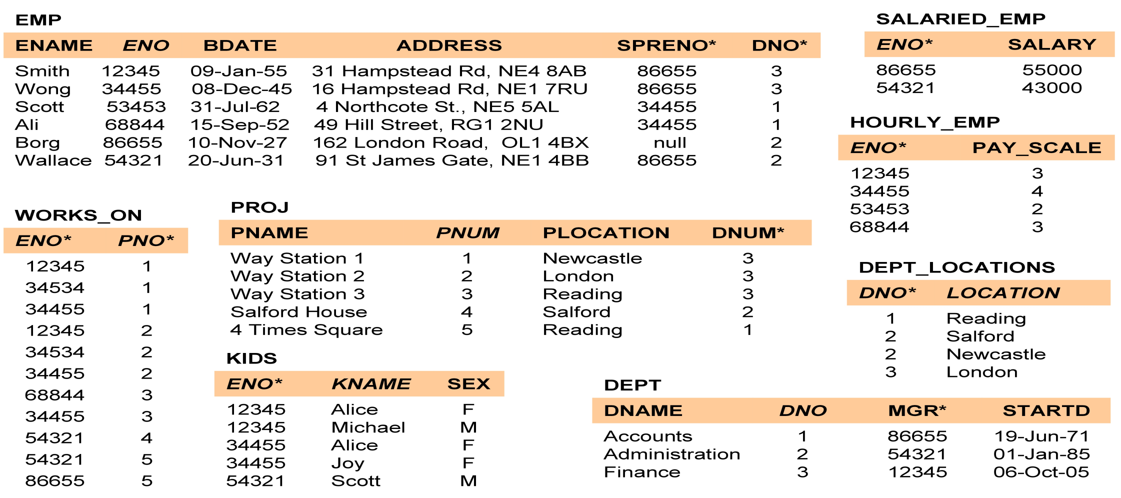
Relationships (Rel): A class C has a set of relationships Rel. Each relationship $rel \in Rel$ between C and class C' is defined in C to represent an association, aggregation or inheritance. $Rel := \{rel \mid rel := \langle RelType, dirC, dirAs, c, invAs\rangle\}$, where $RelType$ is a relationship type, $dirC$ is the name of C', and $dirAs$ denotes a set containing the attribute names representing the relationship from the C' side. The $invAs$ denotes a set of inverse attribute names representing the inverse relationship from the C side, and c is the cardinality constraint of rel from the C side. $RelType$ can have the following values: '*associated with*' for association, '*aggregates*' for aggregation, and '*inherits*' or '*inherited by*' for inheritance. Relationships have two cases: 1:1 and 1:M, and c is defined by $min..max$ notation to indicate the minimum and maximum occurrences of objects of C' within objects of C. Based on c, the object(s) of C' can be single-valued where $c := 0..1$ (optional) or $c := 1..1$ (required), or set-valued where $c := 0..*$ (optional) or $c := 1..*$ (required).

Unique keys (UK): A class C may have a set of UK(s) that are preserved in UK: $UK := \{\delta \mid \delta := \{\langle ua, s\rangle\}\}$, where δ represents one key, ua is an attribute name, and s is a sequence number.

Generation of CDM from RDB: Using key matching, relations and their attributes are classified, relationships among relations are identified and their cardinalities are determined. All these are translated into equivalents in the CDM. The semantically enriched CDM forms the starting point for the remaining steps of the migration process that leads to the generation of the target schema and then the conversion of relational data into target data. Each relation R is classified based on the comparison of its PK with the PKs of other relations, and mapped into one of the nine CDM classes above. After class C is classified, it is important, if $C.cls$:= ("SST" | "SSC"), to check whether C is concrete or abstract. C is a concrete class (i.e., abs := `false`) when all (or some) of its corresponding RDB table rows are not members of other sub-tables, and abstract otherwise. Attributes of R are identified and mapped along with other properties into attributes of C. The keys of R are used to generate the relationships Rel of C. Using this information, the relationships among relations are identified, their cardinalities determined, and they are then mapped into Rel as association, inheritance or aggregation. Using the corresponding data, every relationship that R participates in is identified and mapped into an equivalent relationship rel and added to Rel.

Example 1: Consider the RDB shown in Figure 1. PKs are in italics and FKs are marked by "*". Table 1 shows (partly) the resulting CDM. Each RDB relation is mapped into a class in CDM. For instance, the relation `Emp` is mapped into the CDM class `Emp`, which is an abstract SST class, and has the attributes: *ename*, *eno*, *bdate*, *address*, *spreno* and *dno*. Other properties of the attributes (e.g., types, tags) are shown. The class is '*associated with*' the classes: `Dept` (twice), `Works_on` and with itself (twice). Moreover, it '*aggregates*' the `Kids` class and is '*inherited by*' the

Salaried_emp and Hourly_emp classes. Cardinality c and UKs are also given for each class.



EMP

ENAME	*ENO*	BDATE	ADDRESS	SPRENO*	DNO*
Smith	12345	09-Jan-55	31 Hampstead Rd, NE4 8AB	86655	3
Wong	34455	08-Dec-45	16 Hampstead Rd, NE1 7RU	86655	3
Scott	53453	31-Jul-62	4 Northcote St., NE5 5AL	34455	1
Ali	68844	15-Sep-52	49 Hill Street, RG1 2NU	34455	1
Borg	86655	10-Nov-27	162 London Road, OL1 4BX	null	2
Wallace	54321	20-Jun-31	91 St James Gate, NE1 4BB	86655	2

SALARIED_EMP

*ENO**	SALARY
86655	55000
54321	43000

HOURLY_EMP

*ENO**	PAY_SCALE
12345	3
34455	4
53453	2
68844	3

WORKS_ON

*ENO**	*PNO**
12345	1
34534	1
34455	1
12345	2
34534	2
34455	2
68844	3
34455	3
54321	4
54321	5
86655	5

PROJ

PNAME	*PNUM*	PLOCATION	DNUM*
Way Station 1	1	Newcastle	3
Way Station 2	2	London	3
Way Station 3	3	Reading	3
Salford House	4	Salford	2
4 Times Square	5	Reading	1

DEPT_LOCATIONS

*DNO**	*LOCATION*
1	Reading
2	Salford
2	Newcastle
3	London

KIDS

*ENO**	*KNAME*	SEX
12345	Alice	F
12345	Michael	M
34455	Alice	F
34455	Joy	F
54321	Scott	M

DEPT

DNAME	*DNO*	MGR*	STARTD
Accounts	1	86655	19-Jun-71
Administration	2	54321	01-Jan-85
Finance	3	12345	06-Oct-05

Figure 1: Sample input RBD

cn	cls	abs	A_{cdm}						Rel					UK	
			a_n	t	tag	l	n	d	$RelType$	$dirC$	$dirAs$	c	$invAs$	ua	s
Emp	SST	true	eno	int	PK	25	n		*asso*	Dept	dno	1..1	dno		
			ename	char		40	n		*asso*	Dept	mgr	0..1	eno		
			bdate	date			y		*asso*	Emp	eno	1..1	spreno		
			address	char		40	y		*asso*	Emp	spreno	0..*	eno		
			spreno	int	FK	25	y		*asso*	Works_on	eno	1..*	eno		
			dno	int	FK		n		*aggr*	Kids	eno	0..*	eno		
									inherBy	Salaried_emp	eno	1..1	eno		
									inherBy	Hourly_emp	eno	1..1	eno		
Salaried_emp	SUB	false	eno	int	PF	25	n		*inherts*	Emp	eno	1..1	eno		
			salary	int			y								
Dept	RST	false	dno	int	PK		n		*asso*	Emp	eno	1..1	mgr	mgr	1
			dname	char		40	n		*asso*	Emp	dno	1..*	dno		
			mgr	int	FK	25	n		*asso*	Proj	dnum	1..*	dno		
			startd	date			y		*aggr*	Dept_locations	dno	1..*	dno		
Works_on	RRC	false	eno	int	PF	25	n		*asso*	Emp	eno	1..1	eno		
			pno	int	PF		n		*asso*	Proj	pnum	1..1	pno		

asso: *associated with*　　*aggr*: *aggregates*　　*inherBy*: *inherited by*

Table 1: Results of CDM generation

4 TRANSLATING CDM INTO OBJECT-RELATIONAL SCHEMA

This section presents the translation of CDM into object-relational schema. We first define the SQL4 ORDB target schema, and then explain the rules for translating the CDM constructs into their equivalents in the target schema.

SQL4 ORDB schema

The SQL4 ORDB schema is defined as a set of user-defined types (UDTs), and a set of typed tables created based on these UDTs for storing data. Each UDT consists of a set of attributes defined as literal or reference types. Literal types are defined as a primitive, collection of primitive, **row**, or collection of **row** types. Moreover, an attribute can be defined as a reference (**ref**) or a collection of references (**ref**s), pointing to a specific UDT. Composite attributes are defined using **row** types. An association relationship is expressed among UDTs using **ref**s. The collection (i.e., **set**) of primitive and **row** types are used to define simple and composite multi-valued attributes, respectively, whereas the M side of associations are defined as a collection of **ref**s. UDTs and typed tables can be defined into hierarchies, realising inheritance relationships, in which types/tables can be defined as sub-types/sub-tables **under** their super-types/super-tables.

Definition of ORDB schema: The SQL4 ORDB schema is denoted as 3-tuple: ORschema := $\langle UT, TT, UK_{or}\rangle$, where UT is a set of UDTs, TT is a set of typed tables, and UK_{or} is a set of unique keys. The sets UT and TT are defined as follows:

- $UT := \{udType \mid udType := \langle ut_n, s_{ut}, A_{ut}\rangle\}$, where ut_n is the name of a user-defined type $udType$, s_{ut} is the super-type name of $udType$, and A_{ut} is a set of $udType$'s attributes of literal or ref-based data type:

  $A_{ut} := \{a_{ut} \mid a_{ut} := \langle a_n, t, m, n, d\rangle\}$, where a_n is the name of an attribute a_{ut}, t is its data type, which can be primitive (e.g., integer), user-defined constructed (e.g., **row** type) or ref-based (e.g., **ref**($udType$)); m denotes whether a_{ut} is single-valued or collection-valued, d is a default value in the case of primitive attributes, and n denotes whether or not a_{ut} accepts nulls.

- $TT := \{tTable \mid tTable := \langle tt_n, ut_n, s_{tt}, pk, uoid\rangle\}$, where tt_n is the name of a typed table $tTable$, ut_n is the name of $udType$ based upon which $tTable$ is defined, s_{tt} is the name of its super-table, pk is the primary key of $tTable$, and $uoid$ is the user-defined identifer of the objects of $tTable$.

Algorithm for Schema Translation

When the CDM has been obtained, the schema translation starts by applying an appropriate set of rules to map the CDM constructs into equivalents in the target schema. Each rule maps a specific construct, e.g., **row** type and attribute.

Creating User-Defined Types

To create typed tables for storing data it is necessary to define the underlying object types as UDTs. Each main class $C \in$ CDM is translated into a UDT udt (of type $udType$). The name ut_n of udt takes the same name as that of C, i.e., $C.cn$, suffixed by a string '`_t`', e.g., `Emp_t`. All UDTs, except sub-class types, are defined with a self-referential attribute, using the '**ref using varchar**(25)' string, as part of their definition. The tables defined based on those UDTs must then specify that the identifier of each object *uoid* is user-generated.

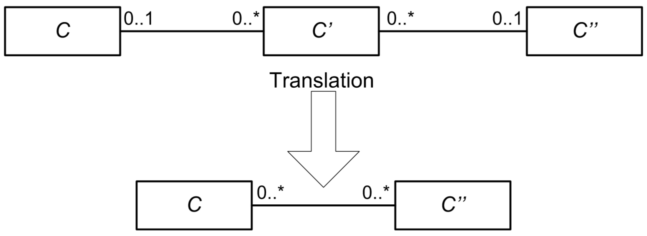
Translating atomic attributes: Each non-FK attribute $a \in C.A_{cdm}$, i.e., $a.tag \neq$ ('FK' | 'PF') is translated into a primitive attribute a_{ut} and added to the attribute set A_{ut} of *udt*. Each $a_{ut} \in A_{ut}$ retains the same properties from $a \in C.A_{cdm}$, i.e., a_n, t and d. The multiplicity m of a_{ut} is single-valued.

Translating relationships: CDM relationships are translated and defined among *udt*s as association, aggregation and inheritance. Each relationship $rel \in C.Rel$ is mapped, based on *rel.RelType*, into a relationship attribute a_{ut} and added into A_{ut}, or mapped into an inheritance relationship. The name a_n of the attribute a_{ut} that represents the association/aggregation relationship is generated by concatenating *rel.dirC* with attribute names in *rel.dirAs*, and *C.cn* with attribute names in *rel.invAs*, e.g., `dept_mgr` and `emp_eno`. These strings are then changed by the user to appropriate relationship names as shown in Figure 2. Object-valued relationships are modeled among rows of UDTs using **ref**s. Collections are used to model multiple values in a single column of a table as a set of **ref**s/literals identifying that a column in one table contains another table. Collection types include **array** and **set** (or **multiset**) in SQL4. As a result of being better for larger, unordered and frequently changed collections, we use the **set** construct for translating the M side of relationships. The multiplicity m of a_{ut} is single-valued when $rel.c$:= (0..1 | 1..1), and a collection when $rel.c$:= (0..m | 1..m). CDM inheritance, defined by *rel.RelType* := "*inherits*", is translated into the target ORDB schema using **under** clause in each sub-type/sub-table definition.

- **Association:** Each relationship $rel \in C.Rel$ where *rel.RelType* := "*associated with*" is translated, in the *udt* corresponding to C, into an attribute a_{ut} where its type $a_{ut}.t$ is a **ref** or a collection of **ref**s (depending on $rel.c$), referencing the type udt' mapped from the corresponding CDM class C'. A **ref** attribute is constrained to be scoped (i.e., using **scope** clause) to a specific table, so that the **ref** values stored in that attribute points at objects of the specified table. Each association is defined bidirectionally between its two types. However, unlike the OODB systems that support ODMG 3.0, neither SQL4 nor any ORDB products allow user to rely on the system to automatically enforce inverse relationships. The inverse direction of the relationship is defined in udt' side. Depending on the cardinalities c of *rel* and the classification of associated class $C'.cls$, *rel* is translated into 1:1, 1:M or M:N relationships as follows:

  – *rel* is mapped into a single-valued attribute a_{ut} of **ref** type in *udt*, pointing to a pre-defined udt' when $rel.c$:= (0..1 | 1..1).

- rel is mapped into a collection-valued attribute a_{ut} that contains a collection of **ref**s, pointing to a pre-defined udt' if $rel.c := (0..m \mid 1..m)$, i.e., **set**(**ref**($udt'$)).
- rel is mapped into an M:N relationship if $C'.cls :=$ "RRC". As C' participates in only two M:1 association relationships with C and another CDM class C'', rel is mapped into a collection-valued attribute a_{ut} inside udt that contains a collection of **ref**s, pointing to a pre-defined udt'', which corresponds C''. Similarly, a collection of **ref**s that references udt is defined inside udt'' when mapping from the C'' side.



- **Aggregation:** Each relationship $rel \in C.Rel$ where $rel.RelType :=$ "*aggregates*" is translated, in udt mapped from C, into an attribute a_{ut} that is typed as a literal type, representing the CDM class C' that participates in a relationship with C. Depending on c of rel and $C'.cls$, rel is translated into multi-valued attributes or **row** types.
  - If $C'.cls :=$ "MAC", rel is mapped into a collection of single-valued attribute a_{ut}. The collection is equivalent to the non-FK attribute in $C'.A_{cdm}$.
  - If $C'.cls :=$ "CAC", rel is mapped into an attribute a_{ut} typed as a **row** type when $rel.c := (0..1 \mid 1..1)$, or as a collection of **row**s when $rel.c := (0..m \mid 1..m)$. The attributes of the **row** type are mapped from the non-FK attributes in $C'.A_{cdm}$.
- **Inheritance:** Each relationship $rel \in C.Rel$ where $rel.RelType :=$ "*inherits*" is mapped as a single inheritance, where udt, translated from C, inherits all of the properties of its super-type utd' using its name, i.e., $s_ut := utd'.ut_n$, where utd' corresponds to the super-class C'. Additional properties of udt are defined in the usual way. Creating a sub-type under its super-type should be considered while creating the super-type, by specifying the **not final** phrase at the end of the super-type definition, which is **final** by default. Specifying **not final** for a super-type in the **create type** statement means that other types can inherit it.

Creating Typed Tables

The creation of typed tables is based on the UDT specifications, which represent object instances for each row in a table. A typed table T_{or} (of type $tTable$) is defined for each declared *udt* and labeled with the same name as the corresponding CDM class C, from which its *udt* has been translated, i.e., without '`_t`'. Because tables contain objects that can be referenced by other objects, a *uoid* column is specified as user-generated OID to facilitate the cyclic referencing among pre-created objects during data loading. When inserting a tuple in T_{or}, the *uoid* can be generated from PK values of the corresponding RBD table. The PK of each table pk is mapped from the attributes in $C.A_{cdm}$, where $tag :=$ 'PK'. However, sub-tables inherit PKs and *uoid*s from their super-tables. UKs for each table are extracted from the CDM equivalents and placed in UK_{or}. Sub-tables are created **under** their super-tables via $s_{tt} := C'.cn$, where $C'.cn$ is the name of corresponding super-class C'. Given two UDTs, *udt* and its sub-type *udt*′, the table T_{or} of type *udt* is created and then the table T'_{or} of type *udt*′ is created "**under**" T_{or}, with which each object in T'_{or} is implicitly represented in T_{or}.

Example 2: Consider the CDM shown in Table 1, Figure 2 shows the output SQL4 ORDB schema, which contains UDTs and typed tables. For example, the type `Emp_t` has been created from the CDM class `Emp` and then used to create the `Emp` table. Non-FK attributes, e.g., *ename* and *eno* are mapped normally from the CDM, whereas other attributes define relationships with other types such as `dept` that references the pre-defined `Dept_t`. This attribute is translated from the 1:1 association (i.e., ⟨"*associated with*", `Dept`, $\{dno\}$, 1..1, $\{dno\}$⟩) between the `Emp` and `Dept` CDM classes, which is defined in the `Emp` class and given in Table 1. In the inverse direction, a collection that contains **ref**s of `Emp_t` is defined in the `Dept_t` type to show that employees are employed by each department. The **set**s are used to store a collection of values on the M side of relationships. The `Kids` class is mapped as a composite multi-valued attribute inside `Emp_t` using **set** and **row**, whereas the `Dept_locations` class is mapped in `Dept_t` as a simple multi-valued attribute using **set**. Typed tables are created to store the actual data. Inheritance relationships among UDTs/tables are defined using the **under** phrase. `Hourly_emp_t` and `Salaried_emp_t` sub-types are mapped from the corresponding CDM classes and defined under the `Emp_t` super-type. The corresponding tables then become sub-tables of the `Emp` super-table, inheriting its properties.

5 CONVERTING RELATIONAL DATA INTO ORDB

This section describes the rules for converting RDB data into files as ORDB format. These files are then used to populate the ORDB schema generated earlier. Having all files generated, they can be loaded into the ORDB system using a bulk-loading facility. Applying the rules, the process is performed in two passes to aid establishing relationships consistently. In the first pass, objects are defined to initialise typed tables with literal data, whereas the second pass defines relationships among pre-created objects.

```
create type Emp_t as (
   ename varchar(20), eno number, bdate date, address varchar(30),
   manages ref(Dept_t) scope Dept,
   supervises set(ref(Emp_t)),
   hasKids set(row(kname varchar(30), sex char(1))),
   projects set(ref(Proj_t)),
   dept ref(Dept_t) scope Dept,
   supervisor ref(Emp_t) scope Emp) not final
   ref using varchar(25);
create table Emp of Emp_t
   constraint Emp_pk primary key(eno), ref is uoid user generated;

create type Hourly_emp_t under Emp_t (pay_scale number) final;
create table Hourly_emp of Hourly_emp_t under Emp;
create type Salaried_emp_t under Emp_t (salary number) final;
create table Salaried_emp of Salaried_emp_t under Emp;

create type Dept_t as (
   dname varchar(20), dno number, startd date,
   locations set(varchar(25)),
   employees set(ref(Emp_t)),
   controls set(ref(Proj_t)),
   manager ref(Emp_t) scope Emp)
   ref using varchar(25);
create table Dept of Dept_t
   constraint Dept_pk primary key(dno), ref is uoid user generated;

create type Proj_t as (
   pname varchar(20), pnum number, plocation varchar(20),
   employees set(ref(Emp_t)),
   controledBy ref(Dept_t) scope Dept)
   ref using varchar(25);
create table Proj of Proj_t
   constraint Proj_pk primary key(pnum), ref is uoid user generated;
```

Figure 2: Sample output SQL4 ORDB schema

Data Conversion Functions

We assume the following functions to be used during the process of data conversion.

- $PKa(C)$, $FKa(C)$ and $NFKa(C)$ are functions which return respectively the PK, FK and non-FK attribute names of a CDM class C.

- $CL(C)$ is a function which returns the $classLeaves$ list, containing all sub-class names of a super-class C ordered from bottom to top.

- $getCond(C, classLeaves)$ is a function which returns an SQL **where** condition $cond$, which is added to the queries used in RDB data retrieval to exclude from a super-class RDB table T (corresponding to C where $C.cls$:= ("SST" | "SSC")), all rows that are members in its sub-class tables. Sub-class names are stored in $classLeaves$ list. Through $cond$, the non-inherited RDB tuples in T are extracted and converted into the target database. The $cond$ is 'Nil' by default.

- $CH(C)$ is a function which returns the $classHierarchy$ list, containing all super-class names of a sub-class C ordered from top to bottom. During the generation of data of a typed sub-table T_{or} corresponding to C, data from each RDB table corresponding to each of its top level super-classes are also retrieved to be converted into the target format and added into each of the T_{or} objects being defined or updated.

Initialising Typed Tables

Data from each RDB table T corresponding to a main and concrete class C in the CDM (i.e., $C.abs$:= `false` and $C.cls \neq$ (“MAC” | “CAC” | “RRC”) are extracted and converted in order to populate the corresponding typed table T_{or}. However, data from T, where $C.cls \neq$ (“MAC” | “CAC” | “RRC”) are converted as part of the establishment of aggregation and association relationships. Converted data are then loaded into a file named with $C.cn$. In the first pass, each target object Obj of T_{or} is generated by defining its user-defined object identifier *uoid* and its structure $objStruct$, consisting of literal-based data. When *uoid* and $objStruct$ are constructed, a DDL statement (using the **insert** operator) is written to the file, defining Obj. Object-valued relationships of Obj are initialised in the second pass.

Literal-based atomic attributes: An SQL query that satisfies a particular condition *cond* is designed in order to retrieve PK data (using $PKa(C)$) and non-FK attributes (using $NFKa(C)$) data from T and stores the results in `SetResult` table. Then, from each tuple $t \in$ `SetResult`, an *uoid* is generated for each Obj by concatenating $C.cn$ with the data values of the PK of C in t, i.e., $t(PKa(C))$; thus the value of *uoid* is guaranteed to be unique for each object, e.g., `‘salaried_emp54321’`. SQL4 allows self-referential attributes that can be user-defined as an identifier and specified as part of the type definition of the referenced table. When the typed table is created, *uoid* is specified as an additional column which stores the value of *uoid* for each object in the table. The *uoid* can then be used in establishing relationships. The data of the non-FK attributes of C in t, i.e., $t(NFKa(C))$ are converted to become the new ORDB atomic data of Obj and are assigned to $objStruct$.

Literal-based collections: For each CDM aggregation relationship $rel \in C.Rel$ between C and a component class C', where $rel.RelType$:= “*aggregates*” and $C'.cls$:= (“MAC” | “CAC”), the object Obj being initialised is appended with literal-based collection data. Tuples of non-FK attributes in RDB table T', corresponding to C' are retrieved (using $NFKa(C')$), where the set of relationship attribute(s) $dirAs$ of rel in T' is equal to the PK value(s) of each tuple t retrieved from the parent table T, i.e., $t(PKa(C))$. The retrieved data are then restructured into ORDB format as a data collection *dataColl*. The attribute values in *dataColl* are generated from the non-FK tuples of C' (using $NFKa(C')$) as normal scalar attributes. The multi-valued attributes data are generated when $C'.cls$:= “MAC”, whereas **row** type data are generated when $C'.cls$:= “CAC”. The *dataColl* is returned as a string which represents a collection, i.e., ‘**set**(’+*dataColl*+‘)’ when $rel.c$:= (0..m | 1..m),

[illegible]

or as a single-valued attribute/**row** otherwise. The $dataColl$ is then assigned to a corresponding relationship attribute and appended to $objStruct$ of Obj being defined.

Inheritance among objects: If class C is a sub-class, where $C.cls :=$ "SUB" (or $C.cls :=$ "SSC" if $C.abs :=$ `false`), then the corresponding ORDB sub-table T_{or} is initialised with its own data from the corresponding RDB table T, in the normal way for defining objects. In addition, data from the super-table(s) related to data from T are converted and initialised through T_{or}; thus realising the inheritance relationship. Each object in T_{or} is populated by its literal and object-valued data and all of its top-level super-classes which have their names stored in the $classHierarchy$ list.

Establishing Relationships

After literal data have been generated, the second pass in the conversion process is to assign $uoid$s of pre-created objects to their relationship attributes. For each CDM relationship rel defined in class C, where $rel \in C.Rel$ and $rel.RelType :=$ "*associated with*", data are retrieved from the RDB tables T corresponding to C and T' corresponding to C' related to C. This is to initialize the relationship attribute defined in T_{or}, which is equivalent to rel defined in C. This is performed by a projection on selected attributes (i.e., $PKa(C)$ and $rel.invAs$), from T and storing the result in `ResultSet` table. The $uoid$ of each object being updated is extracted from the PK data of C of each tuple t stored in `ResultSet`, i.e., $t(PKa(C))$. The identifiers t_uoids of target objects, related to the object being updated, are extracted and stored in a list, called $t\_uoidList$. The t_uoids in $t\_uoidList$ are constructed from one of the following:

1. From the tuples of a set of relationship attribute(s) $rel.invAs$ retrieved from T corresponds to C when the relationship values are available in T, i.e., $rel.invAs \subseteq FKa(C)$. This means that T contains the FK attributes data that represent the relationship,

2. From the tuples extracted by a projection on the PK (extracted using $PKa(C')$) of the RDB table T' corresponds to a class C' related to C, where the set of relationship attribute(s) $rel.dirAs$ in C equals the PK values retrieved from T, i.e., $t(PKa(C))$, or

3. From the tuples of a set of relationship attribute(s) $rel''.invAs$ retrieved from T', corresponding to C', when $C'.cls :=$ "RRC". The rel'' is a CDM relationship that C' participates in with another class C'', where rel' is the other relationship that C' participates in with C and $rel'.dirC := C.cn$.

Example 3: Consider the CDM shown in Table 1 and RDB data in Figure 1, Figure 3 shows the target ORDB object converted from the tuple of the employee "Wallace". The tuple identified by ID 54321 is extracted from the sub-class `Salaried_emp` RDB table. Figure 3(a) shows a sample of ORDB SQL4 statement generated for initialising the ORDB object, whereas Figure 3(b) shows the statements for updating the defined object with its relationships. Data and relationships inherited from the super-class `Emp`'s object are shown.

(a)	`insert into Salaried_emp values (Salaried_emp_t('salaried_emp54321', 'Wallace', 54321, '1931-06-20', '91 St James Gate NE1 4BB', null, null, set(row('Scott', 'M')), null, null, null, 43000);`
(b)	`update Salaried_emp set manages = 'dept2', projects = set('proj4','proj5'), dept = 'dept2', supervisor = 'salaried_emp86655' where uoid = 'salaried_emp54321';`

Figure 3: Output ORDB SQL4 object definition

6 EXPERIMENTAL STUDY

To demonstrate the effectiveness and validity of our method, a prototype has been developed, realizing its algorithms. The algorithms were implemented using Java 1.5 and Oracle 11_g. We setup experiments to evaluate our approach by examining the differences between source RDB and the ORBD generated by the prototype. The method has been evaluated according to the query results provided by the database system, i.e., Oracle 11_g. A set of queries has been designed to observe any differences in the query results between the source RDB and the target ORDB. The experiments were run on a PC with Pentium IV 3.2 GHz CPU and 1024 MB RAM operated under Windows XP Professional. This section presents two sets of queries applied on the RDB shown in Figure 1 and the equivalent ORDB generated by the prototype. Table 2 shows the description, the RDB and ORDB versions, and the result of each query.

Description	Relational query	Object-relational query	Result
Find the name of department 3	`select dname from Dept where dno = 3;`	`select dname from Dept where dno = 3;`	Finance
Find salaried employees in department 2 who make 50000 or more per year	`select e.ename from Emp e, Salaried_emp s where e.dno = 2 and e.eno = s.eno and s.salary >= 50000;`	`select s.ename from Salaried_emp s where s.dept.dno = 2 and s.salary >= 50000;`	Borg
Find all employees working in the Accounts department	`select e.eno, e.ename from Emp e, Dept d where e.dno = d.dno and d.dname = 'Accounts';`	`select s.column_value.eno, s.column_value.ename from Dept d, table(d.employees) s where d.dname = 'Accounts';`	34534 Scott 68844 Ali
Find all employees who have kids named Alice and Michael	`select e.ename from Emp e, Kids d1, Kids d2 where e.eno = d1.eno and e.eno = d2.eno and d1.kname = 'Alice' and d2.kname = 'Michael';`	`select h.ename from Hourly_emp h, table(h.hasKids) d1, table(h.hasKids) d2 where d1.kname = 'Alice' and d2.kname = 'Michael';`	Smith
Display a list of project names that involve an employee called Smith	`select pname from Proj p, Works_on w, Emp e where e.eno = w.eno and w.pno = p.pnum and e.ename = 'Smith';`	`select pname from Proj p, table(p.employees) e where e.column_value.ename= 'Smith';`	Way Station 1 Way Station 2

Table 2: Results of the queries

When the results have been evaluated, the approach described here is shown to be feasible, efficient and correct as the queries return identical results. The target ORDB is generated without loss or redundancy of data. Moreover, many semantics have been converted from RDB into the ORDB, e.g., association, aggregation and inheritance. Update operations (i.e., insert, delete and update) are applied on the databases, which show that integrity constraints in the RDB are preserved in the target ORDB. However, referential integrity on **ref**s that are in nested tables in ORDB is not guaranteed because Oracle does not have a mechanism to do so. This integrity could be preserved, e.g., using triggers once the migration process is completed. In addition, the correctness and applicability of the CDM and the migration algorithms are also tested by checking and comparing the target schemas resulting from the prototype and those generated by existing manual-based mapping techniques. Further details on this can be found in [10, 12].

7 CONCLUSION

This paper contributes a solution to the problem of migrating RDBs into ORDBs. The solution is superior to existing work as it generates the ORDB, including the schema and data, and it exploits the range of powerful features provided by SQL4. A prototype has been developed to realize the solution, and evaluated by comparing query results from the input and output databases. We have designed experiments that involve running queries on an existing RDB and the target ORDB generated by the prototype. We have analysed the query results obtained from both databases and found that both sets of results were identical. Therefore, we conclude that the source and target databases are equivalent. Moreover, the results obtained demonstrate that the solution, conceptually and practically, is feasible, efficient and correct.

REFERENCES

[1] Arora, G., Belden, E. and Iyer, C.: Oracle Database Application Developer's Guide - Object-Relational Features, 10g Release 2 (10.2), Part Number B14260-01. Oracle Corporation, 2005.

[2] Carey, M., Florescu, D., Ives, Z., Lu, Y., Shanmugasundaram, J., Shekita, E. and Subramanian, S.: XPERANTO: Publishing Object-Relational Data as XML. WebDB, pp. 105–110, 2000.

[3] Eessaar, E.: Whole-Part Relationships in the Object-Relational Databases. In WSEAS, Athens, Greece, pp. 1263-1268, 2006.

[4] Eisenberg, A. and Melton, J.: SQL:1999, Formerly Known as SQL3. SIGMOD Record, vol. 28(1), pp. 131–138, 1999.

[5] Funderburk, J., Kiernan, G., Shanmugasundaram, J., Shekita, E. and Wei, C.: XTABLES: Bridging Relational Technology and XML. IBM Systems Journal, vol. 41(4), pp. 616–641, 2002.

[6] Kim, W.: Introduction to Object-Oriented Databases. MIT Press, Cambridge, MA, USA, 1991.

[7] Grant, E. S., Chennamaneni, R. and Reza, H.: Towards Analyzing UML Class Diagram Models to Object-Relational Database Systems Transformations. In Databases and Applications, pp. 129–134, 2006.

[8] Marcos, E., Vela, B., and Cavero, J. M.: Extending UML for Object-Relational Database Design. In 4th Int. Conf. on the Unified Modeling Language, vol. 2185, pp. 225–239, 2001.

[9] Marcos, E., Vela, B. and Cavero, J. M.: A Methodological Approach for Object-Relational Database Design using UML. Soft. and Syst. Modeling, vol. 2, pp. 59–75, 2003.

[10] Maatuk, A., Ali, M. A. and Rossiter, N.: An Integrated Approach to Relational Database Migration. In IC-ICT '08, pp. 1–6, Bannu, Pakistan, 2008.

[11] Maatuk, A., Ali, M. A. and Rossiter, N.: Semantic Enrichment: The First Phase of Relational Database Migration. In CIS^{2}E '08, 6pp, Bridgeport, USA, 2008.

[12] Maatuk, A., Ali, M. A. and Rossiter, N.: Migrating Relational Databases into Object-based/XML Databases: An Evaluation. Tech. Report, School of Computing, Engineering and Information Sciences, Northumbria University, UK, 2008.

[13] Mok, W. Y. and Paper, D. P.: On Transformations from UML Models to Object-Relational Databases. In HICSS, 2001.

[14] Mok, W. Y.: Designing Nesting Structures of User-defined Types in Object-Relational Databases. Info. Soft. Tech., vol. 49(9-10), USA, 2007.

[15] Pardede, E., Rahayu, J. W. and Taniar D.: Mapping Methods and Query for Aggregation and Association in Object-Relational Database using Collection. ITCC, vol. 1, pp. 539-, 2004.

[16] Pardede, E., Rahayu, J. W. and Taniar, D.: New SQL Standard for Object-Relational Database Applications. In SIIT, pp. 191–203, 2003.

[17] Soutou, C.: Modeling Relationships in Object-Relational Databases. Data Knowl. Eng., vol. 36(1), pp. 79–107, 2001.

[18] Stonebraker, J. M., Brown, P. and Moore, D.: Object-Relational DBMSs: The Next Great Wave and Object-Relational DBMSs: Tracking the Next Great Wave, Morgan Publishers, 1999.

[19] Urban, S. D., Dietrich, S. W. and Tapia, P.: Succeeding with Object Databases: Mapping UML Diagrams to Object-Relational Schemas in Oracle 8. John Wiley and Sons, Ltd, pp. 29–51, 2001.

[20] Urban, S. D. and Dietrich, S. W.: Using UML Class Diagrams for A comparative Analysis of Relational, Object-Oriented, and Object-Relational Database Mappings. In SIGCSE '03, ACM Press, New York, NY, USA, pp. 21–25, 2003.

[21] Takahashi, T. and Keller, A. M.: Implementation of Object View Query on a Relational Database. In Data and Knowl. Syst. for Manuf. and Eng., 1994.

ABOUT THE AUTHORS

Dr. Abdelsalam Maatuk is a lecturer at Faculty of Science, Department of Computer, Omar Al-Mukhtar University, Libya. He received his Ph.D. in 2009 from Northumbria University, UK. His research interest include schema integration, database migration and reverse engineering. He can be reached at ammaatuk@yahoo.com.

Dr. M. Akhtar Ali is a senior lecturer at School of Computing, Engineering and Information Sciences, Northumbria University, Newcastle, UK. In 2003 he received his Ph.D. from Manchester University. He is interested in self-tuning of databases for high performance. He can be reached at akhtar.ali@northumbria.ac.uk.

Dr. Nick Rossiter is a reader at School of Computing, Engineering and Information Sciences, Northumbria University, Newcastle, UK. He is interested in interoperability of information systems. He can be reached at nick.rossiter@northumbria.ac.uk.