Uppsala Master's Thesis in
Computer Science 312
2007-08-17
ISSN 1100-1836

Archiving Relational Databases using a Semantic Web Representation

Santosh Kumar Reddy Maddula

Information Technology
Computer Science Department
Uppsala University
Box 337
S-751 05 Uppsala
Sweden

Supervisor: Silvia Stefanova
Examiner: Tore Risch

Abstract

A database independent migration tool is developed for long term archival of relational databases. The approach is to represent both schema and data of an existing relational database in an RDF-Schema based representation. RDF-Schema is a semantic web standard for representing any kind of data and meta-data. The system, SARD, can automatically convert data from an existing relational database into an RDF-Schema based flat file representation through its Relational *RDF-archiver* module. Later SARD can recreate the archived data in another relational database by a *Relational RDF*-loader module. Standard relational database interfaces, the query language SQL, and RDf_Schema are used to make the migration DBMS independent. A proof-of-concept implementation shows the viability of the approach for simple relational databases.

Table of Contents

1 Introduction

Relational databases are extensively used for storing information. In many cases long time archival and restoring the data in relational databases is of prime importance. Each DBMS vendor provides tools for archival and restoration of the vendor's relational databases. However, these tools are only standardized for migrating relational data between different versions of the database of the same vendor. This is problematic when migrating data of a relational database from one vendor's database to another vendor. Changing DBMS vendor requires unloading the data into a sequential file and then reloaded the file into a new database. This requires that the sequential unload format is DBMS vendor independent and such a standard for archiving relational databases is not defined. Furthermore, for long term storage of relational databases beyond the lifetime of a DBMS product it is desired to have the archived data stored in a database independent format.

One of the goals with the semantic web initiative is to provide schema descriptions, called *ontologies,* for different kinds of web resources. Metadata and ontologies play major roles for retrieving and combining information from different sources. RDF and RDF-Schema are the W3C standards for describing the information used in the semantic web. RDF allows annotation of web resources (e.g. URLs) with properties and property values. While RDF allows to associate any property with any web resources, the extended language RDF-Schema [1] is used to define schemas of web resources. With RDF-Schema standardized properties, i.e. ontologies, are defined for different application domain. They are similar to relational database schemas.

The purpose of this project is to implement a system, SARD (Semantic Web Archival of Relational Databases), to demonstrate semantic web based archiving and loading of relational databases. SARD can archive the data of a relational database into a standardized sequential file and later load back the archived data into another relational database. The following is developed:

1. An *RDF schema* representation is defined that can represent both schema and contents of a relational database.
2. An *RDF-archiver* module is developed that generates the RDF-schema representation of a given relational database.

3. An *RDF-loader* module reloads an archived relational database to recreate the original relational database.

The implementation of SARD utilizes the SWARD system [13]. SWARD allows the user to view the contents of a relational database as RDF-Schema, given specification of mappings between the database schema and the RDF-Schema ontology. The RDF-archiver uses SWARD for extracting all data stored in the relational database as a query to SWARD's RDF view of the relational database. The query extracts all data according to a SARD ontology. The RDF-loader uses the RDFAmos system [2] to parse an RDF-Schema document and then apply database operations on the parsed RDF statements. The RDF loader calls SQL statements that recreates the relational database schema and populates the new database.

This project is proof of concept only and is developed for only very simple relational databases. The system can archive tables having only the basic data types VARCHAR, INTEGER, DATE, and FLOAT. Other database features such as compound keys, foreign keys, triggers, constraints, advanced datatypes, etc. are not handled.

2 Background

SARD is based on the following technologies: relational databases, RDF, SWARD, and Amos II. The concepts of relational databases are assumed known to the reader and are described in many text books, e.g. [14].

2.1 Semantic Web and RDF

In a relational database the description of tables is provided by the *schema*. Similarly semantic web [3] representations can be used to describe the schema of web resources. The semantic web is a way of representing World Wide Web information, in which the information is expressed not only for interfacing users as with HTML, but also in a form so that that is understandable and used by programmers and programs.

RDF [4] is the basic language for describing web contents used in the semantic web. Any expression in RDF is represented as a collection of RDF *triples* [4], also called RDF *statements*. A set of such triples is called an RDF *graph*. An RDF triple contains three components:

- The *subject* is used to identify the entity described by the statement. For example, if we consider an RDF statement that states that the document in *http://www.example.org/index* has a language whose value is English then the subject of the statement is *http://www.example.org/index*.
- *The predicate* identifies the property of the subject described by the statement. The predicate for the example statement is a property describing languages of subjects. It is identified, e.g., by the URL *http://www.purl.org/Language,* defined in the Dublin Core ontology [5].
- *The object* identifies the value of the property. The object for the above statement is the string (literal) *"English"*.

In all RDF triples the subject must be a *URI* or a *blank node*. A URI reference is a Unicode string with an *optional fragment* identifier at the end. If we consider the URI *http://www.example.org/schemas/vechicles#motorvehicle*, *#motorvehicle* is the optional fragment. A blank node represents an anonymous internal URI. The predicate must be a URI often defined in some ontology. The object can be a URI, a blank node, or a *literal*. Literals

are used to define constant values. There are two kinds of literals: a *plain literal* is a string and a *typed literal* is a string with an associated data type URI. In this work we consider plain literals only. An RDF triple can be illustrated by a node and directed-arc diagram, in which each triple is represented as a node-arc-node link (Figure 2.1.1).

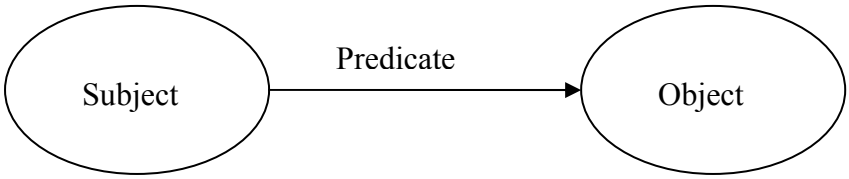

Figure 2.1.1: RDF Triple

For example, the triple in Fig.2.1.2 represent the RDF statement "*http://www.example.org/index* has a property *language* whose value is *English*".

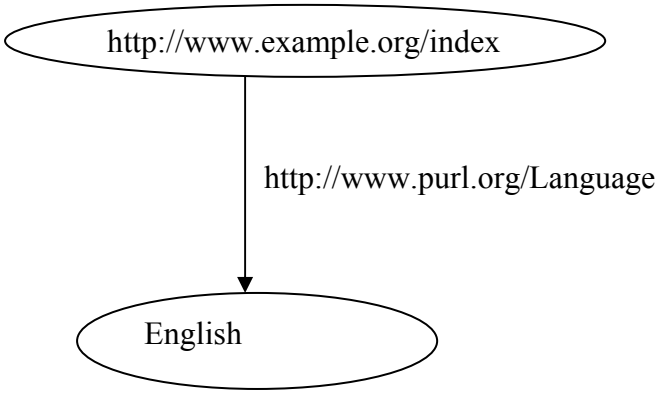

Figure 2.1.2: RDF Statement

The extended language *RDF-Schema* (RDFS) [1] also provides a class system. RDF-Schema is used to define standardized properties for different application domains, analogous to relational database schemas. The classes defined for a specific application is called a *schema* or an *ontology*. Different RDF-Schemas have been developed for different kinds of web

documents, the most well known is Dublin Core [5], Open Directory [6], and RSS [7]. In this project we investigate how RDF-Schema can be used as a sequential text format for representing both schemas and contents of relational databases. This would allow RDF-Schema to be used for long term archiving of relational databases.

A *class* in RDF-schema has similar meaning as a class in object-oriented programming languages such as Java. Using RDF classes we can represent any kind of object such as places, movie types, and so on. The specification of the class *c* of a subject *s* is done by a triple <*s*, *rdf:Class,c*>[1] stating that the class of *s* is the URI *c*. Each class *c* can have an number of allowed *properties* p_i defined by triples <*c, rdfs:Property,* p_i >[2]. Figure 2.1.4 illustrates how classes and properties are defined corresponding to the schema of the relational table AUTHORS relational Table in Figure: 2.1.3(b). The relational tables in Figure 2.1.3 (a) and (b) are used as to illustrate SARD.

SERVICENR	EID	NAME
1	ABC1234H	Issuing a birth certificate
2	ABC1234H	Online payment

Figure: 2.1.3 (a): The RELATEDSERVICES table

SURNAME	NAME	AUTHOR_ID
BARISH	GREG	1
BUDD	TIMOTHY	2

Figure:2.1.3(b): The AUTHORS table

[1] rdf: is an XML Qualified Name (QName) used to represent name space URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#
[2] rdfs: is a QName used to represent name space URI: http://www.w3.org/2000/01/rdf-schema#

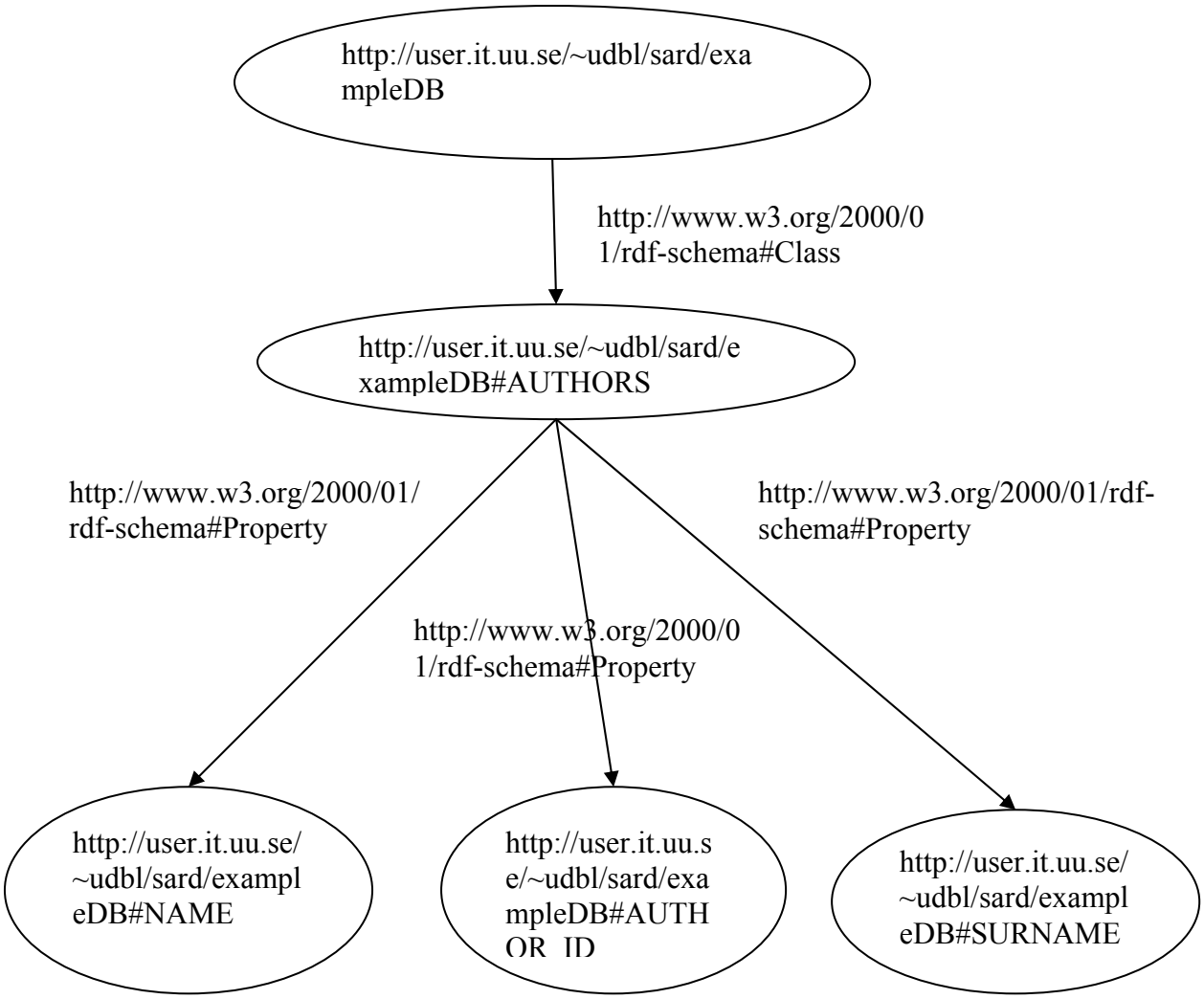

Figure 2.1.4: AUTHORS meta-data

The triple format of the above schema example is shown in Figure: 2.1.5. In RDF-Schema a class is any resource having an *rdf:type* property whose value is the resource *rdf:Class*. So here the class *Authors* is defined by an RDF URI reference *sard:Authors*, where namespace *sard:* is defined as *http://user.it.uu.se/~udbl/sard/exampleDB.*

sard:Authors.	rdf:type	rdfs:Class
sard:Authors.	rdfs:Property	sard:SURNAME .
sard:Authors.	rdfs:Property	sard: AUTHOR_ID

```
sard:Authors.          rdfs:Property          sard:NAME.
```

Figure: 2.1.5: Triples describing AUTHORS table

RDF statements are usually represented in an XML-format. Figure: 2.1.6 shows the RDF/XML format [8] of the above example.

```
<rdf:RDF xml:lang='en'
xml:base='http://user.it.uu.se/~udbl/sard/exampleDB1#'
xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
<rdf:Description ID='AUTHORS'>
<rdf:type resource='http://www.w3.org/2000/01/rdf-schema#Class'/>
</rdf:Description>
<rdf:Description ID='SURNAME'>
<rdf:type resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
<rdfs:domain rdf:resource='#AUTHORS'/>
<rdfs:range rdf:resource='xsd:string'/>
</rdf:Description>
<rdf:Description ID='NAME'>
<rdf:type resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
<rdfs:domain rdf:resource='#AUTHORS'/>
<rdfs:range rdf:resource='xsd:string'/>
</rdf:Description>
<rdf:Description ID='AUTHOR_ID'>
<rdf:type resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
<rdfs:domain rdf:resource='#AUTHORS'/>
<rdfs:range rdf:resource='xsd:integer'/>
</rdf:Description>
</rdf:RDF>
```

Figure: 2.1.6 RDF/XML representation of RDF-Schema

SARD uses RDF-schema to represent both the schema and the contents of a relational database in RDF/XML format.

2.2 Amos II

Amos II [10] is a main-memory DBMS having a functional query language, AMOSQL. SARD and SWARD are built on top of Amos II. Amos II allows wrapping external data sources, e.g. relational databases, so they can be queried with AMOSQL. SARD and SWARD uses a general wrapper of relational databases to archive and reload the contents of relational databases. The relational wrapper streams data though Amos II, rather than loading it into the database in Amos II, which makes the system work for relational databases of any size. SARD is implemented as Amos II interface functions that archive and reload the relational databases.

The basic concepts of the Amos II data model are *objects, types and functions:*

Objects represent all entities that are stored in an Amos II database. Every object is an instance of a *type*. Literal objects are self identifying and instances of built in data types like *Charstring, Integer, Boolean*, or *Real*. Objects that are not treated as literal objects are called as *surrogate* objects that have unique object identifiers.

Types are used to structure and group objects. A type hierarchy relates sub- and super-types.

Functions are used for representing attributes of objects, to make relationship between objects, and to do operations on objects. The signature of a function consists of its name and the types of its arguments and results, e.g.:

header(Charstring namespace, Charstring dbname) -> Charstring

There are four kinds of functions:

1. A ***stored function*** represents object properties explicitly stored in a table inside Amos II. For example, in SARD a stored function *header* represents the header of the archive RDF files given the namespace for SARD (*sard:*) and the name of the relational database to archive:

   create function header(Charstring namespace, Charstring dbname)
   -> Charstring as stored.

2. A ***derived function*** is defined by a query statement similar to relational database views. For example,

   create function table_name_down(JDBC jd) -> Charstring
   as select tname
   from Charstring tname, Charstring catalog, Charstring schema,
   Charstring owner, Charstring relationalname
   where tables(jd) = <tname,catalog,schema,owner,relationalname>
   and c4=tables

   SARD uses this derived function to retrieve the names of the tables in a relational database.
3. A ***foreign function*** is defined in a programming language, e.g. JAVA, C/C++ or Lisp. Foreign functions are used internally by SWARD to access relational databases.
4. A ***procedure*** is a function with side effects. SARD uses stored procedures to implement side effect actions, such writing and reading RDF files.

2.3 SWARD

The SWARD (Semantic Web Abridged Relational Databases) [13] system provides RDF views of data stored in an existing relational database. SWARD allows querying a relational database in terms of some RDF ontology. It extracts data from a relational database as a single relation of triples, called the *Universal Property View, UPV*. The UPV is defined as a union of a *schema view* and a *content view*, where the former represents the relational schema and the latter its contents. SWARD generates the UPV automatically for a user specified relational database and ontology. The user must specify the *property mappings* that map the names for table columns to RDFS properties and *class mappings* that map table names to RDFS classes.

To explain the SWARD system we use the example database that contains two tables named RELATEDSERVICES and AUTHORS in Figure 2.1.3(a) and Figure: 2.1.3(b).

SWARD uses following Amos II procedures to generate the UPV of a database automatically:

- **ExportRDB(Charstring UPV)-> Boolean**

The argument of the above procedures is the chosen name of the UPV. In SARD the UPV name is always the same, “myUPV”. However, in order to export the relational database SWARD requires to specify *property* and *class mapping* tables for mapping column and table names to the used ontology. In SARD these tables are generated automatically by the RDF-archiver by iterating through all the tables in the relational database to archive and construct a URI for each table by appending each table name with the namespace *sard: . sard:* is defined as the URI *http://user.it.uu.se/~udbl/sard/exampleDB*. The class mapping table for the example database is shown in Table 2.2.1.

Table	ClassID
RELATEDSERVICES	sard:/RELATEDSERVICES
AUTHORS	sard:/AUTHORS

Table 2.2.1: Class mapping table

The system iterates through all columns in the all tables of the relational database to unload and generates a URI by concatenating the table name and column name to the namespace :sard. The column mapping table for the example database is illustrated by the following Table 2.2.2.

Table	Column	PropID
RELATEDSERVICES	NAME	sard:/RELATEDSERVICES#NAME
RELATEDSERVICES	SERVICENR	sard:/RELATEDSERVICES#SERVICENR
RELATEDSERVICES	EID	sard: /RELATEDSERVICES#EID
AUTHORS	SURNAME	sard:/AUTHORS#SURNAME
AUTHORS	NAME	sard: /AUTHORS#NAME
AUTHORS	AUTHOR_ID	sard: /AUTHORS#AUTHOR_ID

Table 2.2.2: column mapping table

3. The SARD system

This chapter explains how SARD works with a simple usage example, followed by a description of its architecture.

3.1 Simple Usage Example

Two user interface procedures are defined to archive a relational database and later reload an archived database:

- **SARDUnload(Charstring jdbc_id, /* JDBC URL */**
  Charstring driver, /* JDBC driver */
  Charstring user, /* RDB user name */
  Charstring passw, /* RDB password */
  Charstring dbname) /* Source DB name */
) -> Charstring
- **SARDLoad(Charstring jdbc_id, /* JDBC URL */**
  Charstring driver, /* JDBC driver */
  Charstring user, /* RDB user name */
  Charstring passw, /* RDB password */
  Charstring dbname) /* Source DB name */
) -> Charstring

The arguments used in the above functions are the JDBC connection identifier for the relational database, the JDBC driver name, the relational database user name, the user password, and the name of the relational database we want to archive and load back.

Next we show how our simple example database in Figure 2.1.3 is archived and loaded back using SARD.

Archiving:
First the user sets the variable *:jdbc* to the identifier of the JDBC connection for the relational database (Firebird, *http://www.firebirdsql.org/*).

```
set :jdbc = "jdbc:interbase://localhost/C:/Program/Firebird/bin/reloadDB.gdb";
```

The database is archived under the name *exampleDB* by calling:

SARDUnload(:jdbc, “interbase.interclient.driver”, “SYSDBA”, “masterkey”, ”exampleDB”);

SARDUnload returns **“**Archival successful**”** if we are succeeding to generate the RDF files. Two RDF files are generated: one for the database schema and one for the contents. The generated RDF files are shown in the Figure: 3.3.1 and Figure: 3.3.2.

Loading:

In order to reload an archived database the user first has to create a new empty relational database (not described here) and then run SARD to reload the archived RDF files into the new database.

The user binds a variable *:jdbc* to the JDBC identifier of the empty relational database:

set :jdbc = "jdbc:interbase://localhost/C:/Program/Firebird/bin/reloadDB.gdb";

Then *SARDLoad* is called to create the relational database schema and populate the database:

SARDLoad(:jdbc, “interbase.interclient.driver”, “SYSDBA”, “masterkey”, ”exampleDB”);

The output of the above function is the string **“**Tables are updated**”** if we succeed to reload the database using archived RDF files.

```
/* generated RDF schema is stored in a RDF file “exampleDB_schema.rdf”*/
<rdf:RDF xml:lang='en'
xml:base='http://user.it.uu.se/~udbl/sard/exampleDB#'
xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
<rdf:Description ID='AUTHORS'>
<rdf:type resource='http://www.w3.org/2000/01/rdf-schema#Class'/>
</rdf:Description>
<rdf:Description ID='SURNAME'>
<rdf:type resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
```

```
<rdfs:domain rdf:resource='#AUTHORS'/>
<rdfs:range rdf:resource='xsd:string'/>
</rdf:Description>
<rdf:Description ID='NAME'>
<rdf:type resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
<rdfs:domain rdf:resource='#AUTHORS'/>
<rdfs:range rdf:resource='xsd:string'/>
</rdf:Description>
<rdf:Description ID='AUTHOR_ID'>
<rdf:type resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
<rdfs:domain rdf:resource='#AUTHORS'/>
<rdfs:range rdf:resource='xsd:integer'/>
</rdf:Description>
<rdf:Description ID='RELATEDSERVICES'>
<rdf:type resource='http://www.w3.org/2000/01/rdf-schema#Class'/>
</rdf:Description>
<rdf:Description ID='NAME'>
<rdf:type resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
<rdfs:domain rdf:resource='#RELATEDSERVICES'/>
<rdfs:range rdf:resource='xsd:string'/>
</rdf:Description>
<rdf:Description ID='SERVICENR'>
<rdf:type resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
<rdfs:domain rdf:resource='#RELATEDSERVICES'/>
<rdfs:range rdf:resource='xsd:integer'/>
</rdf:Description>
<rdf:Description ID='EID'>
<rdf:type resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'/>
<rdfs:domain rdf:resource='#RELATEDSERVICES'/>
<rdfs:range rdf:resource='xsd:string'/>
</rdf:Description>
</rdf:RDF>
```

Figure 3.1.1: RDF Schema representation of example database

```
/* generated RDF data is stored in a RDF file “exampleDB_data.rdf”*/
<rdf:RDF xml:lang='en'
  xmlns:dbs='http://user.it.uu.se/~udbl/sard/exampleDB#'
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
<dbs:AUTHORS about='http://user.it.uu.se/~udbl/sard/exampleDB1/AUTHORS#AUTHOR_ID/1'>
<dbs:SURNAME>BARISH</dbs:SURNAME>
<dbs:NAME>GREG</dbs:NAME>
<dbs:AUTHOR_ID>1</dbs:AUTHOR_ID>
</dbs:AUTHORS>
<dbs:AUTHORS about='http://user.it.uu.se/~udbl/sard/exampleDB1/AUTHORS#AUTHOR_ID/2'>
<dbs:SURNAME>BUDD</dbs:SURNAME>
<dbs:NAME>TIMOTHY</dbs:NAME>
<dbs:AUTHOR_ID>2</dbs:AUTHOR_ID>
</dbs:AUTHORS>
<dbs:RELATEDSERVICES
about='http://user.it.uu.se/~udbl/sard/exampleDB/RELATEDSERVICES#SERVICENR/1'>
<dbs:NAME>Issuing a birth certificate</dbs: NAME>
<dbs:SERVICENR>1</dbs:SERVICENR>
<dbs:EID>ABC1234H</dbs:EID>
</dbs:RELATEDSERVICES>
<dbs:RELATEDSERVICES
about='http://user.it.uu.se/~udbl/sard/exampleDB/RELATEDSERVICES#SERVICENR/2'>
<dbs: NAME>Online payment</dbs: NAME>
<dbs:SERVICENR>2</dbs:SERVICENR>
<dbs:EID>ABC1234H</dbs:EID>
</dbs:RELATEDSERVICES>
</rdf:RDF>
```

Figure: 3.1.2: RDF representation of example database contents

.

3.2 Implementation

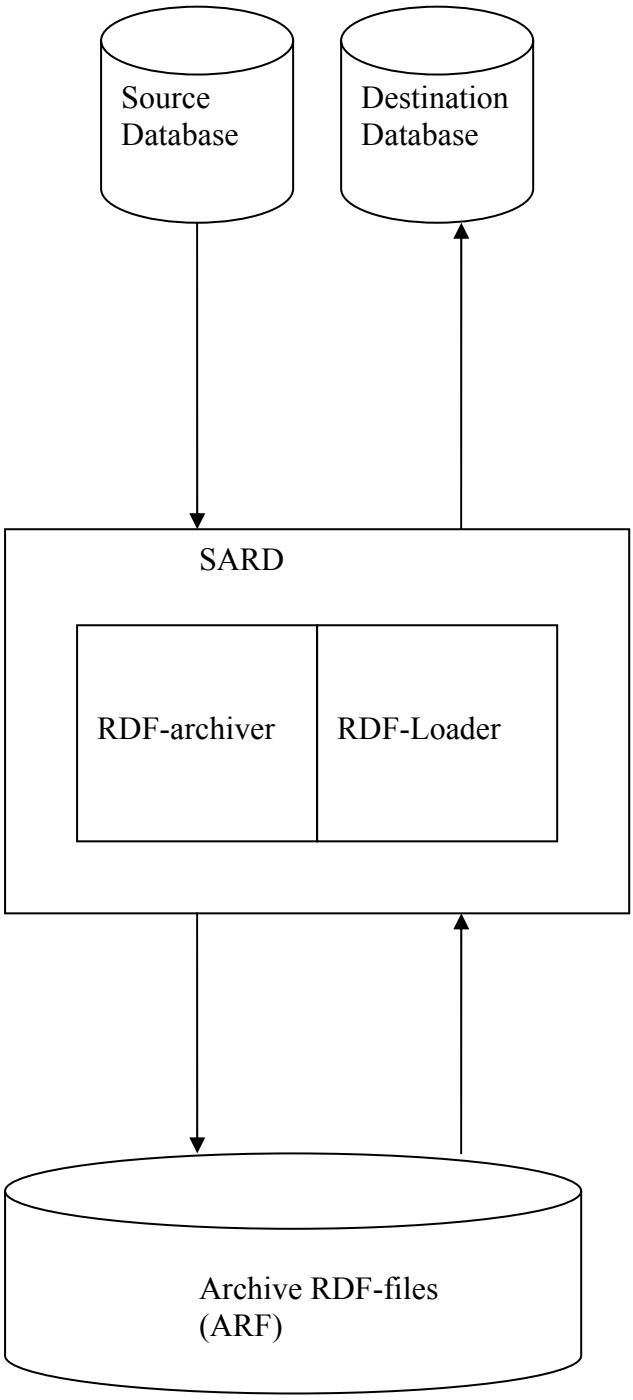

Figure 3.2: SARD System architecture

The modules of the SARD system are.

- The *RDF-archiver* exports the relational database schema and contents as RDF statements stored in two separate *archive RDF files* called the *schema file* and the *contents file*, respectively.
- The *RDF-loader* loads back the archived relational database by reading the archive RDF files. First the schema file is loaded to define the relational schema; then the contents file is loaded to populate the database.

3.2.1 The RDF-archiver

Figure 3.3.1 illustrates the implementation of the RDF-archiver:

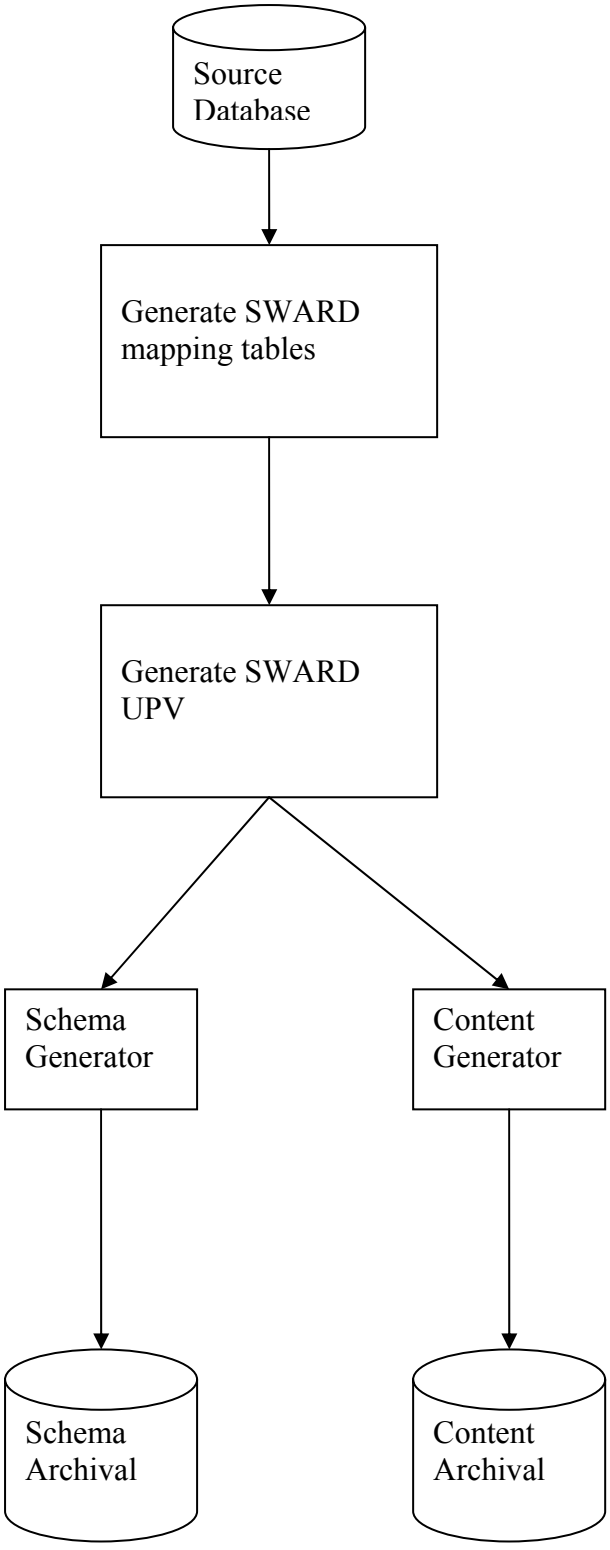

Figure 3.3.1: The RDF-archiver

The schema and content files are generated as follows:

Generating SWARD mapping tables:

A connection is made to the source database and then the *class mapping table* and *property mapping table* are automatically generated by calling the following Amos II functions:

- **table_map(Charstring upv)-> Charstring (For *class mappings*)**
- **column_map(Charstring upv)-> Charstring (For *property mappings)***

These two functions take an UPV name as the argument. The *table_map* function iterates through all the tables in the relational database to construct the URIs of the tables and the columns. Similarly *column_map* function generates the property mapping table by mapping through all columns of all tables in the relational database.

The UPV is automatically generated by SWARD by calling *ExportRDB* when the mapping tables are generated by SARD.

Generating schema and contents files:

For writing the RDF files representing the schema and contents of the relational database SARD uses the following functions:

- **allclassXML(JDBC jd, Charsting dbname, Charstring upv)-> Bag of Charstring (For Schema)**
- **allXMLclass(JDBC jd, Charstring dbname)-> Bag of Charstring (For Contents)**

Both functions take a JDBC connection object *jd* and the name of the relational database *dbname* as arguments. A JDBC connection object is an Amos II object representing a connection to an relational database. The argument *upv* is the UPV name which is always the string "myUPV". The written RDF Schema and contents files are named by concatenating the name of the relational database with the "_schema.rdf" and "_data.rdf", respectively. For example if the name of the database is the "exampleDB" then the file names are:

- "exampleDB_schema.rdf" (For Schema)
- "exampleDB_data.rdf" (For Contents)

When generating the RDF schema file we allow only a small set of data types of the relational database. Table: 3.3.3 lists the data types which can be handled by SARD.

Data type	Mapping Name
Integer	xsd:integer
Float	xsd:float
Varchar	xsd:string
Date	xsd:date

Table 3.3.3: Column data types supported

3.2.2 The RDF-loader

The RDF-loader reads back the archived relational database into the new database from the archived RDF files. It is illustrated by Figure: 3.3.2.

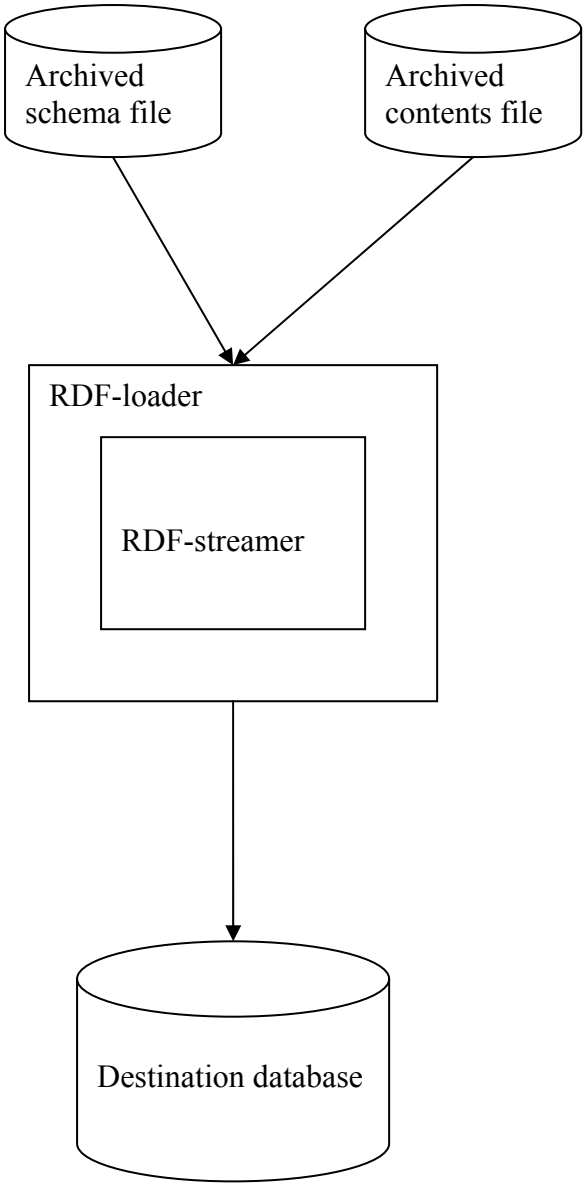

Figure 3.3.2: The RDF-loader

The RDF-loader uses internally the *RDFAmos* wrapper to extract the database information from the archived RDF files in the submodule *RDF-streamer*. RDFAmos uses Jena [15] to parse a RDF/XML file to generate a stream of RDF triple representated in Amos II. In the RDF-loader RDFAmos parses the unloaded schema and contents files. The following RDFAmos function parses an RDF file:

- **parseRDF(Charstring file) -> < Charstring sub, Charstring pred, Charstring obj, Integer int >**

The RDF-loader uses following functions to generate the schema and contents of the relational database from the archived RDF files

- **generate_table_query(JDBC jd, Charstring contentfile, Charstring schemafile, Charstring tname, Charstring namespace) -> Charstring**
- **updatetable(JDBC jd, Charstring contentfile, Charstring schemafile, Charstring tname, Charstring namespace) -> Charstring**

These two functions takes a JDBC connection identifier, the RDF data file name, the RDF schema file name, table name and name space of SARD (*sard:*) as arguments. First function used to generate the required tables from the xml files and then creates the tables in the new database, the second function used to insert the values in the created tables.
SARDunload calls the above functions.

Table 3.3.4 describes internal functions used by SARD.

Function	**Task**
parseRDF(Charstring file) -> <Charstring sub, Charstring pred, Charstring obj, Integer int>	To generate the triples from an RDF/XML file. Implemented by RDFAmos wrapper.
get_colinfo(Charstring schemafile, Charstring namespace, Charstring tname) -> <Charstring type, Charstring column>	To get a column name and its type for a specific table.
get_pkey(Charstring contentfile, Charstring schemafile,Charstring namespace,Charstring tname, Charstring namespace) -> Charstring column	To get the primary key of the table
generate_create_table_query(JDBC jd,Charstring schemafile, Charstring contentfile, Charstring tname, Charstring namespace) -> Charstring	To generate the SQL statement for creating a table
updatetable(JDBC jd,Charstring schemafile , Charstring contentfile,Charstring tname, Charstring namespace) -> Charstring	To insert the values in the table
sqlu(JDBC jd,Charstring query) -> Integer int	To update the SQL query in Amos II.

Table 3.3.4: Internal functions called and their tasks

4. Conclusion and Future work

We have developed a system SARD that automatically archives the data in simple relational databases into an RDF/XML-based relational database representation. SARD can later reload archived data to reconstruct the relational database. Both the relational database sceham and the contents of the relational tables can be archived and reloaded.

The system is a very simple proof-of-concept implementation and needs to be generalized in many ways, e.g.:

- This system currently works only for only Firebird databases and should work for any relational database. One problem here is how to handle non-standard SQL features.
- SARD can handle only non-compound primary keys which should be generalized.
- The supported set of data types of the columns is very limited and should be extended.
- Complete handling of all or most relational database features should be supported.
- The system performance should be improved to provide scalable archival and loading of large relational databases.

References:

[1]. Dan Brickely, R.V.Guha : RDF Vocabulary Description Language 1.0: RDF Schema
http://www.w3.org/TR/rdf-schema/

[2]. P.Gray, L.Kerschberg, P.King, and A.Poulovassillis: Functional Queries to Wrapped Educational Semantic Web Meta-data, in Functional Aproach to Data Management – Modeling, Analyzing and Integrating Heterogeneous Data, Springer, ISBN 3-540-00375-4, 2003.
http://user.it.uu.se/~torer/publ/semfdm.pdf

[3]. W3C: Semantic Web Activity
http://www.w3.org/2001/sw/

[4]. G.Klyne and J:J. carroll: Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Working Draft,
http://www.w3.org/TR/rdf-concepts/, 2004

[5]. Dublin Core Metadata Initiative:
http://dublincore.org/

[6]. Open Directory Project:
http://www.dmoz.org/

[7]. RDF Site Summary (RSS) 1.0:
http://web.resource.org/rss/1.0/

[8]. F.Manola, E.Miller: RDF Primer, W3C Working Draft.
http://www.w3.org/TR/rdf-primer/

[9]. SWARD – Semantic Web Abridged relational Databases.
http://user.it.uu.se/~udbl/sward/

[10]. T. Risch, V.Josifovski,T.Katchaounov: Functional Data Integration in a Distributed Mediator System in P.Gray, L.Kerschberg, P.King, and A.Poulovassillis (:eds): Functional Aproach to Data Management – Modeling, Analyzing and Integrating Heterogeneous Data, Springer, ISBN 3-540-00375-4, 2003.
http://user.it.uu.se/~torer/publ/FunMedPaper.pdf

[11]. D.Elin and T.Risch: Amos II Java Interfaces
http://user.it.uu.se/~torer/publ/javaapi.pdf

[12]. T.Risch: Amos II External Interfaces, UDBL Technical Report
http://user.it.uu.se/~torer/publ/external.pdf

[13] J.Petrini and T.Risch: Scalable Queries to RDF Views of Relational Databases
http://user.it.uu.se/~udbl/sward/SWARD.pdf

[14] Ramez Elmasri and Shamkant B.Navathe: Fundamentals of Database systems, 5th Edition, Addison-Wessely, 2007.

[15] Jena – A Semantic Web Framework for Java http://jena.sourceforge.net/