Mapping of Relational Databases to RDF

Imran Falak Sher

Research Literature

- All the material in the slides are taken directly from the following sources:
- Arenas, M., Bertails, A., Prud'hommeaux, E., & Sequeda, J. (2012). *A Direct Mapping of Relational Data to RDF W3C Recommendation 27 September 2012 .*
- Das, S., Sundara, S., & Cyganiak, R. (2012). *R2RML: RDB to RDF Mapping Language W3C Recommendation.*
- Sequeda, J. F., Arenas, M., & Miranker, D. P. (2012). *On Directly Mapping Relational Databases to RDF and OWL.*

Overview of the Papers

- Semantic web requires mapping of relational database to RDF.
- The paper attempts to provide a solution, which is based on the methods of W3C for directly mapping RDB to RDF and OWL.
- Direct mapping produces an OWL ontology using RD schema and its integrity constraints.
- Conditions of information and query preservation are maintained even for RD containing null values.
- The paper also attempts to prove that monotone mapping is mutually exclusive to semantic preservation.

Problem Under Discussion

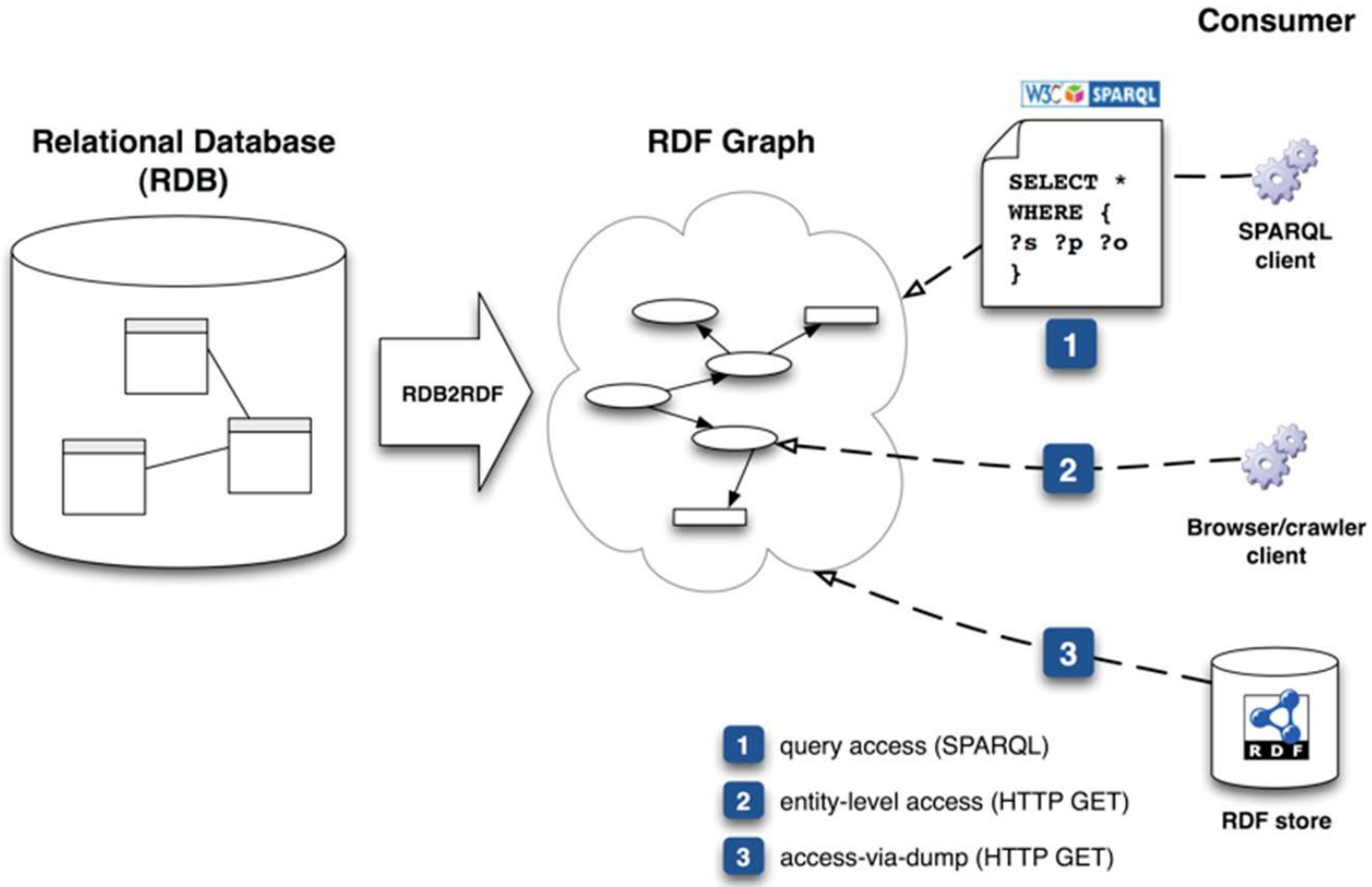
We want data in RDF but it is presently in relational database

Why we want to convert data from RDB to RDF?

- Semantic web requires the data to be on the web as RDF.
- Presently, internet accessible database contains data, which is 500 times more than static web.
- Around 70% of websites are backed by relational database.
- The success of semantic web is dependent on the automatic translation of RDB to RDF and this is done through direct mapping.

What is Data Mapping?

- It is a simple automatic transformation of RDB to RDF.
- The materialized RDF can be queried by SPQARQL.
- The other option is the usage of R2RML mapping language , which allows the creation of customized mapping from relational data to RDF.
- Generated RDF is called the direct graph.
- Direct graphs convey the references established by FKs in relational database as well as the value in the rows.


Consumer
Relational Database
(RDB)
RDF Graph
W3C SPARQL
SELECT *
WHERE {
?s ?p ?o
}
SPARQL
client
RDB2RDF
1
2
Browser/crawler
client
3
RDF
RDF store
1 query access (SPARQL)
2 entity-level access (HTTP GET)
3 access-via-dump (HTTP GET)

Direct Mapping Example

- Following SQL (DDL) is used to create a two tables with single-column PKs and one FKs referenced between them.

People		
PK		Address (ID)
ID	Fname	addr
7	Bob	18
8	Sue	NULL

Addresses		
PK		
ID	City	State
18	Cambridge	MA

```
CREATE TABLE "Addresses" (
        "ID" INT, PRIMARY KEY("ID"),
        "city" CHAR(10),
        "state" CHAR(2)
)

CREATE TABLE "People" (
        "ID" INT, PRIMARY KEY("ID"),
        "fname" CHAR(10),
        "addr" INT,
        FOREIGN KEY("addr")
REFERENCES "Addresses"("ID")
)

INSERT INTO "Addresses" ("ID", "city",
"state") VALUES (18, 'Cambridge', 'MA')
INSERT INTO "People" ("ID", "fname",
"addr") VALUES (7, 'Bob', 18)
INSERT INTO "People" ("ID", "fname",
"addr") VALUES (8, 'Sue', NULL)
```

- The row = 7, 18, Bob produces a set of triples with a common subject.
- The subject is an IRI formed from the concatenation of the base IRI, table name (People), primary key column name (ID) and primary key value (7).
- The predicate for each column is an IRI formed from the concatenation of the base IRI, table name and the column name.
- Each foreign key produces a triple with a predicate composed from the foreign key column names, the referenced table, and the referenced column names. The object of these triples is the row identifier (<Addresses/ID=18>) for the referenced triple.
- It should be noted that the direct mapping does not generate triples for NULL values.

```
@base <http://foo.example/DB/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<People/ID=7> rdf:type <People> .
<People/ID=7> <People#ID> 7 .
<People/ID=7> <People#fname> "Bob" .
<People/ID=7> <People#addr> 18 .
<People/ID=7> <People#ref-addr> <Addresses/ID=18>
.
<People/ID=8> rdf:type <People> .
<People/ID=8> <People#ID> 8 .
<People/ID=8> <People#fname> "Sue" .

<Addresses/ID=18> rdf:type <Addresses> .
<Addresses/ID=18> <Addresses#ID> 18 .
<Addresses/ID=18> <Addresses#city> "Cambridge" .
<Addresses/ID=18> <Addresses#state> "MA" .
```

Example of FKs referencing CKs

- Row=> deptName=accounting and deptCity=Cambridge reference a row in department table with PK of ID=23 (example of People table's compound foreign key to Department).
- The predicate for this key is based on "deptName" and "deptCity", reflecting the order of the column names in the foreign key.
- Object of the predicate is based on the base IRI, the department table name and PK value= ID=23.

People				
PK		→ Addresses (ID)	→ Department(name, city)	
ID	fname	addr	deptName	deptCity
7	Bob	18	accounting	Cambridge
8	Sue	*NULL*	*NULL*	*NULL*

Addresses		
PK		
ID	city	state
18	Cambridge	MA

Department			
PK	Unique Key		→ People(ID)
ID	name	city	manager
23	accounting	Cambridge	8

- The green triples above are generated by considering the new elements in the augmented database.
- The Reference Triple <People/ID=7> <People#ref-deptName;deptCity> <Department/ID=23> is generated by considering a foreign key referencing a candidate key (different from the primary key).

```
@base <http://foo.example/DB/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<People/ID=7> rdf:type <People> .
<People/ID=7> <People#ID> 7 .
<People/ID=7> <People#fname> "Bob" .
<People/ID=7> <People#addr> 18 .
<People/ID=7> <People#ref-addr> <Addresses/ID=18> .
<People/ID=7> <People#deptName> "accounting" .
<People/ID=7> <People#deptCity> "Cambridge" .
<People/ID=7> <People#ref-deptName;deptCity> <Department/ID=23> .
<People/ID=8> rdf:type <People> .
<People/ID=8> <People#ID> 8 .
<People/ID=8> <People#fname> "Sue" .

<Addresses/ID=18> rdf:type <Addresses> .
<Addresses/ID=18> <Addresses#ID> 18 .
<Addresses/ID=18> <Addresses#city> "Cambridge" .
<Addresses/ID=18> <Addresses#state> "MA" .

<Department/ID=23> rdf:type <Department> .
<Department/ID=23> <Department#ID> 23 .
<Department/ID=23> <Department#name> "accounting" .
<Department/ID=23> <Department#city> "Cambridge" .
<Department/ID=23> <Department#manager> 8 .
<Department/ID=23> <Department#ref-manager> <People#ID=8> .
```

Example of Multi-column PKs

- Primary keys may also be composite. If, in the above example, the primary key for *Department* were (*name*, *city*) instead of *ID*, the identifier for the only row in this table would be <Department/name=accounting;city=Cambridge>. The triples involving <Department/ID=23> would be replaced with the following triples:

<People/ID=7> <People#ref-deptName;deptCity>
<Department/name=accounting;city=Cambridge> .
<Department/name=accounting;city=Cambridge> rdf:type <Department> .
<Department/name=accounting;city=Cambridge> <Department#ID> 23 .
<Department/name=accounting;city=Cambridge> <Department#name> "accounting" .
<Department/name=accounting;city=Cambridge> <Department#city> "Cambridge" .

Example of Empty (non-existent) primary keys

- If there is no primary key, each row implies a set of triples with a shared subject, but that subject is a blank node. A *Tweets* table can be added to the above example to keep track of employees' tweets in Twitter:
- The following is an instance of table *Tweets*:
- Given that table *Tweets* does not have a primary key, each row in this table is identified by a Blank Node. In fact, when translating the above table the direct mapping generates the following triples:

Tweets		
→ People(ID)		
tweeter	when	text
7	2010-08-30T01:33	I really like lolcats.
7	2010-08-30T09:01	I take it back.

```
@base <http://foo.example/DB/>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

_:a rdf:type <Tweets> .
_:a <Tweets#tweeter> "7" .
_:a <Tweets#ref-tweeter> <People/ID=7> .
_:a <Tweets#when> "2010-08-30T01:33"^^xsd:dateTime .
_:a <Tweets#text> "I really like lolcats." .

_:b rdf:type <Tweets> .
_:b <Tweets#tweeter> "7" .
_:b <Tweets#ref-tweeter> <People/ID=7> .
_:b <Tweets#when> "2010-08-30T09:01"^^xsd:dateTime .
_:b <Tweets#text> "I take it back." .
```

Example of Referencing tables with empty primary keys

- Rows in tables with no primary key may still be referenced by foreign keys.
- (Relational database theory tells us that these rows must be unique as foreign keys reference candidate keys and candidate keys are unique across all the rows in a table.)
- References to rows in tables with no primary key are expressed as RDF triples with blank nodes for objects, where that blank node is the same node used for the subject in the referenced row.
- The absence of a primary key forces the generation of blank nodes, but does not change the structure of the direct graph or names of the predicates in that graph.

Projects			
Unique key			
		Unique key	
→ People(ID)		→ Department(name, city)	
lead	name	deptName	deptCity
8	pencil survey	accounting	Cambridge
8	eraser survey	accounting	Cambridge

TaskAssignments			
PK			
	→ Projects(name, deptName, deptCity)		
→ People(ID)		→ Departments(name, city)	
worker	project	deptName	deptCity
7	pencil survey	accounting	Cambridge

```
@base <http://foo.example/DB/>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

_:c rdf:type <Projects> .
_:c <Projects#lead> <People/ID=8> .
_:c <Projects#name> "pencil survey" .
_:c <Projects#deptName> "accounting" .
_:c <Projects#deptCity> "Cambridge" .
_:c <Projects#ref-deptName;deptCity> <Department/ID=23> .

_:d rdf:type <Projects> .
_:d <Projects#lead> <People/ID=8> .
_:d <Projects#name> "eraser survey" .
_:d <Projects#deptName> "accounting" .
_:d <Projects#deptCity> "Cambridge" .
_:d <Projects#ref-deptName;deptCity> <Department/ID=23> .

<TaskAssignments/worker=7.project=pencil%20survey> rdf:type
<TaskAssignments> .
<TaskAssignments/worker=7.project=pencil%20survey>
<TaskAssignments#worker> 7 .
<TaskAssignments/worker=7.project=pencil%20survey> <TaskAssignments#ref-
worker> <People/ID=7> .
<TaskAssignments/worker=7.project=pencil%20survey>
<TaskAssignments#project> "pencil survey" .
<TaskAssignments/worker=7.project=pencil%20survey>
<TaskAssignments#deptName> "accounting" .
<TaskAssignments/worker=7.project=pencil%20survey>
<TaskAssignments#deptCity> "Cambridge" .
<TaskAssignments/worker=7.project=pencil%20survey> <TaskAssignments#ref-
deptName;deptCity> <Department/ID=23> .
<TaskAssignments/worker=7.project=pencil%20survey> <TaskAssignments#ref-
project;deptName;deptCity> _:c .
```

Direct Mapping Definition and Properties

- The input of a direct mapping M is a relational schema R, a set Σ of PKs and FKs over R and an instance I of R.
- Output=G is the set of all RDF graphs and RC is the set of all triples of the form (R, Σ, I).
- Therefore, a direct mapping M is a total function from RC to G.
- Two fundamental properties: Information preservation and query preservation.
- Two desirable properties: monotonicity and semantic preservation.

Information Preservation

- Direct mapping is information preserving if it does not lose any information about the relational instance being translated, that is, if there exists a way to recover the original database instance from the RDF graph resulting from the translation process.
- If T is the set of all possible relational instances
- A direct mapping M is information preserving if there is a computable mapping $N : G \rightarrow T$ such that for every relational schema R, set Σ of PKs and FKs over R, and instance I of R satisfying Σ : $N(M(R, \Sigma, I)) = I$.
- Recall that a mapping $N : G \rightarrow T$ is computable if there exists an algorithm that, given $G \in G$, computes N(G).

Query Preservation

- A direct mapping is query preserving if every query over a relational database can be translated into an equivalent query over the RDF graph resulting from the mapping.
- That is, query preservation ensures that every relational query can be evaluated using the mapped RDF data.
- To formally define query preservation, we focus on relational queries that can be expressed in relational algebra and RDF queries that can be expressed in SPARQL.

Example

- Example 1 Assume that a relational schema contains a relation name STUDENT and attributes ID, NAME and AGE. Moreover, assume that t is a tuple in this relation such that t.ID = 1, t.NAME = John and t.AGE = NULL. Then, tr(t) = μ, where the domain of μ is {?ID, ?NAME}, μ(?ID) = 1 and μ(?NAME) = John.

- A direct mapping M is query preserving if for every relational schema R, set Σ of PKs and FKs over R and relational algebra query Q over R, there exists a SPARQL query Q⋆ such that for every instance I of R satisfying Σ : tr([Q]I) =[Q⋆] M(R, Σ,I).

Limitations

- If a direct mapping M is information preserving, this does not guarantee that every relational algebra query Q can be rewritten into an equivalent SPARQL query over the translated data, as M could transform source relational databases in such a way that a more expressive query language is needed to express Q over the generated RDF graphs.
- On the other side, a mapping M can be query preserving and not information preserving if the information about the schema of the relational database being translated is not stored.

Monotonicity

- If we insert new data to the database, then the elements of the mapping that are already computed are unaltered.
- A direct mapping M is monotone if for every relational schema R, set Σ of PKs and FKs over R, and instances I_1, I_2 of R such that $I_1 \subseteq I_2$: $M(R, \Sigma, I_1) \subseteq M(R, \Sigma, I_2)$.

Semantic Preservation

- A direct mapping is semantics preserving if the satisfaction of a set of integrity constraints are encoded in the mapping result.
- Given a relational schema R, a set Σ of PKs and FKs over R and an instance I of R, a semantics preserving mapping should generate from I a consistent RDF graph if $I \models \Sigma$, and it should generate an inconsistent RDF graph otherwise.

Solution Presented by the Paper

- The paper introduce a direct mapping DM, that integrates and extends the functionalities of the direct mapping concept.
- DM is defined as a set of Datalog rules1, which are divided in two parts: translate relational schemas and translate relational instances.

Example

- Consider a relational database for a university. The schema of this database consists of tables STUDENT(SID,NAME), COURSE(CID,TITLE,CODE), DEPT(DID,NAME) and ENROLLED(SID,CID). Moreover, we have the following constraints about the schema of the university: SID is the primary key of STUDENT, CID is the primary key of COURSE, DID is the primary key of DEPT, (SID,CID) is the primary key of ENROLLED, CODE is a foreign key in COURSE that references attribute DID in DEPT, SID is a foreign key in ENROLLED that references attribute SID in STUDENT, and CID is a foreign key in ENROLLED that references attribute CID in COURSE.

Storing Relational Databases

- Given that the direct mapping DM is specified by a set of Datalog rules, its input (R, Σ, I) has to be encoded as a set of relations.
- The paper define the predicates that are used to store the triples of the form (R, Σ, I). More precisely, the following predicates are used to store a relational schema R and a set Σ of PKs and FKs over R.
- REL(r): Indicates that r is a relation name in R; e.g.REL("STUDENT") indicates that STUDENT is a relation name.
- ATTR(a, r): Indicates that a is an attribute in the relation r in R; e.g. ATTR("NAME", "STUDENT") holds.
- $PK_n(a_1, \ldots, a_n, r)$: Indicates that $r[a_1, \ldots, a_n]$ is a primary key in §; e.g. PK1("SID", "STUDENT") holds.
- $FK_n(a_1, \ldots, a_n, r, b_1, \ldots, b_n, s)$: Indicates that $r[a_1, \ldots, a_n] \subseteq_{FK} s[b_1, \ldots, b_n]$ is a foreign key in Σ; e.g. FK_1("CODE", "COURSE", "DID", "DEPT") holds.

Storing an Ontology

- In order to translate a relational database into an RDF graph with OWL vocabulary, the paper first extract an ontology from the relational schema and the set of PKs and FKs given as input.
- In particular, the paper classify each relation name in the schema as a class or a binary relation (which is used to represent a many-to-many relationship between entities in an ER/UML diagram)
- The paper represents foreign keys as object properties and attributes of relations as data type properties.

Identification of binary relations

- Important component of storing an ontology.
- We define auxiliary predicates that identify binary relations to facilitate identifying classes, object properties and data type properties. Informally, a relation R is a binary relation between two relations S and T if (1) both S and T are different from R, (2) R has exactly two attributes A and B, which form a primary key of R, (3) A is the attribute of a foreign key in R that points to S, (4) B is the attribute of a foreign key in R that points to T, (5) A is not the attribute of two distinct foreign keys in R, (6) B is not the attribute of two distinct foreign keys in R, (7) A and B are not the attributes of a composite foreign key in R, and (8) relation R does not have incoming foreign keys.

Identifying classes

- In our context, a class is any relation that is not a binary relation. That is, predicate CLASS is defined by the following Datalog rules:
- CLASS(X) ← REL(X), ¬ISBINREL(X)
- ISBINREL(X) ← BINREL(X,A,B, S,C, T,D)
- In our example, CLASS("DEPT"), CLASS("STUDENT") and CLASS("COURSE") hold.

- For every $n \geq 1$, the following rule is used for identifying object properties that are generated from foreign keys:
- $OP2n(X1, \ldots, Xn, Y1, \ldots, Yn, S, T) \leftarrow FKn(X1, \ldots, Xn, S, Y1, \ldots, Yn, T), \neg ISBINREL(S)$
- This rule states that a foreign key represents an object property from the entity containing the foreign key (domain) to the referenced entity (range). It should be noticed that this rule excludes the case of binary relations, as there is a special rule for this type of relations (see rule (1)). In our example, OP2("CODE", "DID", "COURSE", "DEPT") holds as CODE is a foreign key in the table COURSE that references attribute DID in the table DEPT.

Identifying Data Type Properties

- Every attribute in a non-binary relation is mapped to a data type property: DTP(A,R) ← ATTR(A,R), ¬ISBINREL(R)
- For instance, we have that DTP("NAME", "STUDENT") holds in our example, while DTP("SID", "ENROLLED") does not hold as ENROLLED is a binary relation.

Translating a Relational Schema into OWL

- We introduce a family of rules that produce IRIs for classes, binary relations, object properties and data type properties identified by the mapping (which are stored in the predicates CLASS, BINREL, OPn and DTP, respectively).
- DM uses the following Datalog rules to produce IRIs for classes and data type properties:
- CLASSIRI(R,X) ← CLASS(R), $CONCAT_2$(base,R,X)
  DTP_IRI(A,R,X) ← DTP(A,R), $CONCAT_4$(base,R, "#",A,X)

Translating a Relation Schema

- A rule is used to collect all the classes:
- Second, the family of rules is used to collect all the object properties.
- Third, the following rule is used to collect the domains of the object properties ($n \geq 1$):
- Fourth, the following rule is used to collect the ranges of the object properties ($n \geq 1$):
- Fifth, the following rule is used to collect all the data type properties:
- Finally, the following rule is used to collect the domains of the data type properties:

Translating a database instance into RDF

- We now define the rules that map a relational database instance into RDF.
- More specifically, we first introduce a series of rules for generating IRIs, and then we present the Datalog rules that generate RDF.

Generating IRI for Tuples

- We introduce a family of predicates that produce IRIs for the tuples being translated, where we assume a given a base IRI base for the relational database (for example, "http://example.edu/db/"). First, DM uses the following Datalog rule to produce IRIs for the tuples of the relations having a primary key:
- ROWIRIn(V1, V2, . . . , Vn,A1,A2, . . . ,An, T,R,X) ←
- PKn(A1,A2, . . . ,An,R), VALUE(V1,A1, T,R),
- VALUE(V2,A2, T,R), . . . , VALUE(Vn,An, T,R),
- CONCAT4n+2(base,R, "#",A1, "=", V1, ",",
- A2, "=", V2, ",", . . . , ",",An, "=", Vn,X)
- Thus, given that the facts PK1("SID", "STUDENT") and VALUE("1", "SID", "id1", "STUDENT") hold in our example, the IRI http://example.edu/db/STUDENT#SID=1 is the identifier for the tuple in table STUDENT with value 1 in the primary key.

Translating Relational Instances

- The direct mapping DM generates three types of triples when translating a relational instance: Table triples, reference triples and literal triples. For table triples, DM produces for each tuple t in a relation R, a triple indicating that t is of type r.
- For reference triples, DM generates triples that store the references generated by binary relations and foreign keys.
- Finally, DM produces for every tuple t in a relation R and for every attribute A of R, a triple storing the value of t in A, which is called a literal triple.

Information Preservation DM

- DM does not lose any piece of information in the relational instance being translated.
- Theorem 1: The direct mapping DM is information preserving.
- The proof of this theorem is straightforward, and it involves providing a computable mapping N : G → I that satisfies the condition in Definition 2, that is, a computable mapping N that can reconstruct the initial relational instance from the generated RDF graph.

R2RML

RDB to RDF Mapping Language

Introduction

- It is a language for expressing customized mappings from relational databases to RDF datasets.
- Such mappings provide the ability to view existing relational data in the RDF data model, expressed in a structure and target vocabulary of the mapping author's choice.
- Every R2RML mapping is tailored to a specific database schema and target vocabulary.
- The input to an R2RML mapping is a relational database that conforms to that schema.
- The output is an RDF dataset [SPARQL]as defined in SPARQL, that uses predicates and types from the target vocabulary.
- The mapping is conceptual; R2RML processors are free to materialize the output data, or to offer virtual access through an interface that queries the underlying database, or to offer any other means of providing access to the output RDF dataset.
- R2RML mappings are themselves expressed as RDF graphs and written down in Turtle syntax [TURTLE].

Functioning

- An R2RML mapping refers to logical tables to retrieve data from the input database.
- A logical table can be one of the following: A base table a view, or a valid SQL query (called an “R2RML view” because it emulates a SQL view without modifying the database).
- Each logical table is mapped to RDF using a triples map.
- The triples map is a rule that maps each row in the logical table to a number of RDF triples.

- The rule has two main parts:
- A subject map that generates the subject of all RDF triples that will be generated from a logical table row. The subjects often are IRIs that are generated from the primary key column(s) of the table.
- Multiple predicate-object maps that in turn consist of predicate maps and object maps (or referencing object maps).
- Triples are produced by combining the subject map with a predicate map and object map, and applying these three to each logical table row. For example, the complete rule for generating a set of triples might be:
- **Subjects**: A template http://data.example.com/employee/{empno} is used to generate subject IRIs from the empno column.
- **Predicates**: The constant vocabulary IRI ex:name is used.
- **Objects**: The value of the ename column is used to produce an RDF literal.

Example

EMP			
EMPNO INTEGER PRIMARY KEY	ENAME VARCHAR(100)	JOB VARCHAR(20)	DEPTNO INTEGER REFERENCES DEPT (DEPTNO)
7369	SMITH	CLERK	10

DEPT		
DEPTNO INTEGER PRIMARY KEY	DNAME VARCHAR(30)	LOC VARCHAR(100)
10	APPSERVER	NEW YORK

<http://data.example.com/employee/7369> rdf:type ex:Employee.
<http://data.example.com/employee/7369> ex:name "SMITH".
<http://data.example.com/employee/7369> ex:department <http://data.example.com/department/10>.

<http://data.example.com/department/10> rdf:type ex:Department.
<http://data.example.com/department/10> ex:name "APPSERVER".
<http://data.example.com/department/10> ex:location "NEW YORK".
<http://data.example.com/department/10> ex:staff 1.

- Construction of custom IRI identifiers for departments and employees;
- Use of a custom target vocabulary (ex:Employee, ex:location etc.);
- The ex:staff property has the total number of staff of a department; this value is not stored directly in the database but has to be computed.
- The ex:department property relates an employee to their department, using the identifiers of both entities;

Example: Mapping a Simple Table

- The following partial R2RML mapping document will produce the desired triples from the EMP table (except the ex:department triple, which will be added later).

```
@prefix rr:
<http://www.w3.org/ns/r2rml#>.
@prefix ex:
<http://example.com/ns#>.

<#TriplesMap1>
    rr:logicalTable [ rr:tableName "EMP"
];
    rr:subjectMap [
        rr:template
"http://data.example.com/employee/{
EMPNO}";
        rr:class ex:Employee;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:name;
        rr:objectMap [ rr:column "ENAME"
];
    ].
```

```
<http://data.example.com/employee/7369> rdf:type ex:Employee.
<http://data.example.com/employee/7369> ex:name "SMITH".
```

Example: Computing a Property with an R2RML View

- The definition of a triples map that generates the desired DEPT triples based on this R2RML view follows.

```
<#TriplesMap2>
   rr:logicalTable <#DeptTableView>;
   rr:subjectMap [
      rr:template "http://data.example.com/department/{DEPTNO}";
      rr:class ex:Department;
   ];
   rr:predicateObjectMap [
      rr:predicate ex:name;
      rr:objectMap [ rr:column "DNAME" ];
   ];
   rr:predicateObjectMap [
      rr:predicate ex:location;
      rr:objectMap [ rr:column "LOC" ];
   ];
   rr:predicateObjectMap [
      rr:predicate ex:staff;
      rr:objectMap [ rr:column "STAFF" ];
   ].
```

```
<http://data.example.com/department/10> rdf:type ex:Department.
<http://data.example.com/department/10> ex:name "APPSERVER".
<http://data.example.com/department/10> ex:location "NEW YORK".
<http://data.example.com/department/10> ex:staff 1.
```

Example: Linking Two Tables

- To complete the mapping document, the ex:department triples need to be generated. Their subjects come from the first triples map (<#TriplesMap1>), the objects come from the second triples map (<#TriplesMap2>).
- This can be achieved by adding another rr:predicateObjectMap to <#TriplesMap1>.
- This one uses the other triples map, <#TriplesMap2>, as a parent triples map:
- This performs a join between the EMP table and the R2RML view, on the DEPTNO columns. The objects will be generated from the subject map of the parent triples map, yielding the desired triple:

```
<#TriplesMap1>
    rr:predicateObjectMap [
        rr:predicate ex:department;
        rr:objectMap [
            rr:parentTriplesMap <#TriplesMap2>;
            rr:joinCondition [
                rr:child "DEPTNO";
                rr:parent "DEPTNO";
            ];
        ];
    ].
```

```
<http://data.example.com/employee/7369> ex:department
<http://data.example.com/department/10>.
```

Example: Many-to-Many Tables

- The following example will assume that a many-to-many relationship exists between the extended versions of EMP table and the DEPT table shown below. This many-to-many relationship is captured by the content of the EMP2DEPT table. The database consisting of the EMP, DEPT, and EMP2DEPT tables are shown below:

EMP		
EMPNO INTEGER PRIMARY KEY	ENAME VARCHAR(100)	JOB VARCHAR(20)
7369	SMITH	CLERK
7369	SMITH	NIGHTGUARD
7400	JONES	ENGINEER

DEPT		
DEPTNO INTEGER PRIMARY KEY	DNAME VARCHAR(30)	LOC VARCHAR(100)
10	APPSERVER	NEW YORK
20	RESEARCH	BOSTON

EMP2DEPT PRIMARY KEY (EMPNO, DEPTNO)	
EMPNO INTEGER REFERENCES EMP (EMPNO)	DEPTNO INTEGER REFERENCES DEPT (DEPTNO)
7369	10
7369	20
7400	10

R2RML Mapping

```
<#TriplesMap3>
   rr:logicalTable [ rr:tableName "EMP2DEPT" ];
   rr:subjectMap [ rr:template
"http://data.example.com/employee={EMPNO}/department={DEPTNO}" ];
   rr:predicateObjectMap [
      rr:predicate ex:employee;
      rr:objectMap [ rr:template "http://data.example.com/employee/{EMPNO}"
];
   ];
   rr:predicateObjectMap [
      rr:predicate ex:department;
      rr:objectMap [ rr:template
"http://data.example.com/department/{DEPTNO}" ];
   ].
```

Output Data

```
<http://data.example.com/employee/7369>
   ex:department <http://data.example.com/department/10> ;
   ex:department <http://data.example.com/department/20> .

<http://data.example.com/employee/7400>
   ex:department <http://data.example.com/department/10>.
```

Example: Translating database type codes to IRIs

- Sometimes, database columns contain codes that need to be translated into IRIs, but a direct syntactic translation using string templates is not possible. For example, consider a JOB column in the EMP table with the following possible values, and IRIs corresponding to those database values in the RDF output:

Value	Corresponding RDF IRI
CLERK	http://data.example.com/roles/general-office
NIGHTGUARD	http://data.example.com/roles/security
ENGINEER	http://data.example.com/roles/engineering

- The IRIs are not found in the original database and therefore the mapping from database codes to IRIs has to be specified in the R2RML mapping. Such translations can be achieved using an "R2RML view". The view is defined based on a SQL query that computes the IRI based on the database value. SQL's CASE statement is convenient for this purpose. (Alternatively, one could define this view directly in the database.)

```
<#TriplesMap1>
    rr:logicalTable [ rr:sqlQuery """

        SELECT EMP.*, (CASE JOB
            WHEN 'CLERK' THEN 'general-office'
            WHEN 'NIGHTGUARD' THEN 'security'
            WHEN 'ENGINEER' THEN 'engineering'
        END) ROLE FROM EMP

        """ ];
    rr:subjectMap [
        rr:template "http://data.example.com/employee/{EMPNO}";
    ];
    rr:predicateObjectMap [
        rr:predicate ex:role;
        rr:objectMap [ rr:template "http://data.example.com/roles/{ROLE}" ];
    ].



    <http://data.example.com/employee/7369> ex:role
    <http://data.example.com/roles/general-office>.
```

R2RML Processor

- An *R2RML processor* is a system that, given an R2RML mapping and an input database, provides access to the output dataset.
- There are no constraints on the method of access to the output dataset provided by a conforming R2RML processor. An R2RML processor *MAY* materialize the output dataset into a file, or offer virtual access through an interface that queries the input database, or offer any other means of providing access to the output dataset.
- An R2RML processor also has access to an execution environment consisting of:
- A *SQL connection* to the input database,
- a *base IRI* used in resolving relative IRIs produced by the R2RML mapping.

Direct Mapping VS R2RML

- DM is automatic translation of relational database to RDF.
- R2RML is the customized mapping of RDB to RDF.
- R2RML provides more options in comparison to the DM.

Questions