# R2RML: RDB to RDF Mapping Language

Werner Nutt

## Acknowledgment

These slides are based on a slide set by Mariano Rodriguez

## Reading Material/Sources

- R2RML specification by W3C
  http://www.w3.org/TR/r2rml/

- R2RML specification byW3C
  http://www.w3.org/2001/sw/rdb2rdf/test-cases/

## Standards and Tools

**Mapping languages**

- Standards by RDB2RDF working group (W3C)
  - Direct Mapping
  - R2RML
- Proprietary

**Tools**

Free: D2R, Virtuoso, Morph, r2rml4net, db2triples, ultrawrap, Quest

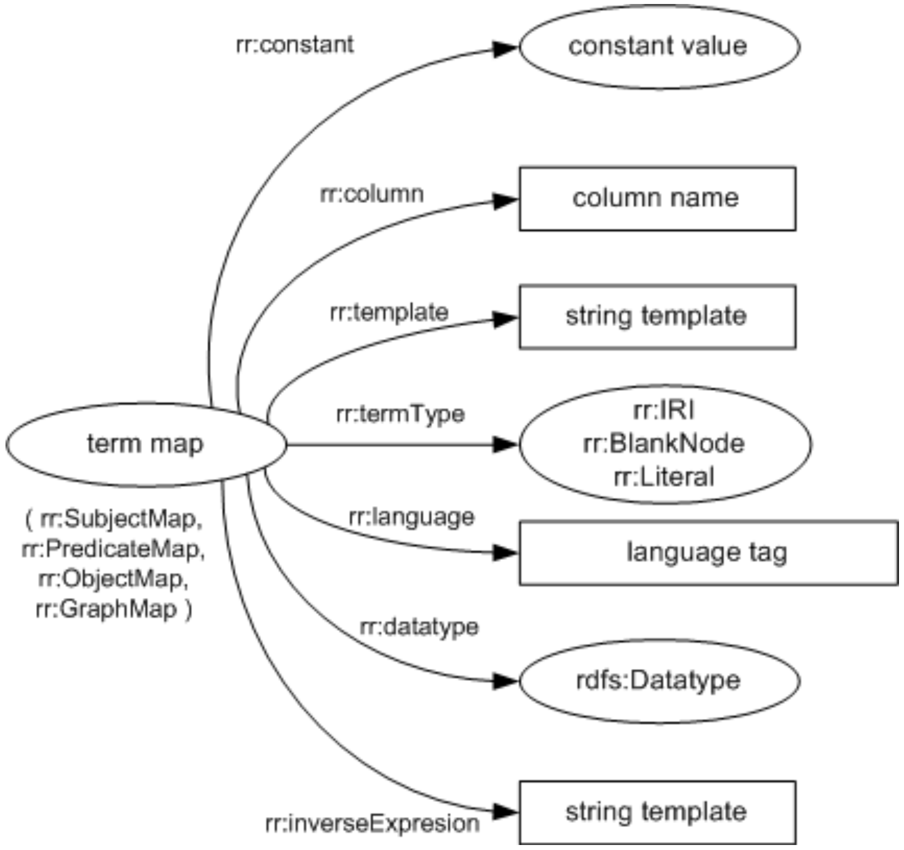
- Commercial: Virtuoso, ultrawrap, Oracle SW

- Detailed Specification
- Books Example
- Tools

- Detailed Specification
- Books Example
- Tools

# Creating RDF Terms with Term Maps

- An **RDF term** is either an IRI, or a blank node, or a literal.
- A **term map** is a function that generates an RDF term from a logical table row. The result of that function is known as the term map's generated RDF term.
- Term maps are used to generate the **subjects, predicates** and **objects** of the RDF triples that are generated by a triples map.
- There are several kinds of term maps, depending on where in the mapping they occur: subject maps, predicate maps, object maps and graph maps.



## Constant RDF terms (rr:constant)

- A **constant-valued term map** is a term map that **ignores the logical table row and always generates the same RDF term**. A constant-valued term map is represented by a resource that has exactly one rr:constant property.
- The constant value of a constant-valued term map is the RDF term that is the value of its rr:constant property.
- If the constant-valued term map is a **subject map, predicate map or graph map**, then its constant value **must be an IRI**.
- **If** the constant-valued term map is an **object map**, then its constant value **must be an IRI or literal**.
- The referenced columns of a constant-valued term map is the empty set.

## Constant RDF term shortcurs

- Constant-valued term maps can be expressed more concisely using the constant shortcut properties rr:subject, rr:predicate, rr:object and rr:graph. Occurrences of these properties must be treated exactly as if the following triples were present in the mapping graph instead:

| Triple involving constant shortcut property: | Replacement triples: |
|---|---|
| ?x rr:subject ?y. | ?x rr:subjectMap [ rr:constant ?y ]. |
| ?x rr:predicate ?y. | ?x rr:predicateMap [ rr:constant ?y ]. |
| ?x rr:object ?y. | ?x rr:objectMap [ rr:constant ?y ]. |
| ?x rr:graph ?y. | ?x rr:graphMap [ rr:constant ?y ]. |

## Example

- The following example shows a predicate-object map that uses a constant-valued term map both for its predicate and for its object.

```
[] rr:predicateMap [ rr:constant rdf:type ];
   rr:objectMap [ rr:constant ex:Employee ].
```

- If added to a triples map, this predicate-object map would add the following triple to all resources ?x generated by the triples map:

```
?x rdf:type ex:Employee.
```

- The following example uses constant shortcut properties and is equivalent to the example above:

```
[] rr:predicate rdf:type;
   rr:object ex:Employee.
```

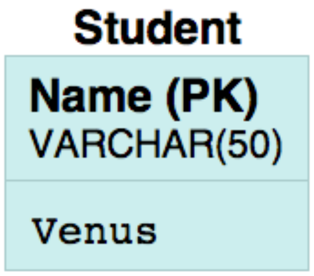
# Example with constants

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.com/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@base <http://example.com/base/> .

<TriplesMap1>
    a rr:TriplesMap;

    rr:logicalTable [ rr:tableName "\"Student\"" ];
    rr:subjectMap [ rr:constant ex:BadStudent ] ;

    rr:predicateObjectMap
    [
      rr:predicateMap [ rr:constant ex:description ];
      rr:objectMap    [ rr:constant "Bad Student"; ]
    ]
    .
```



Student

| Name (PK) VARCHAR(50) |
|---|
| Venus |

## Terms from a Column

- A **column-valued term map** is a term map that is represented by a resource that has exactly one rr:column property.
- The value of the rr:column property must be a valid column name. The column value of the term map is the data value of that column in a given logical table row.
- The referenced columns of a column-valued term map is the singleton set containing the value of the term map's rr:column property.
- The following example defines an object map that generates literals from the DNAME column of some logical table.

```
[] rr:objectMap [ rr:column "DNAME" ].
```

## From template

- **A template-valued term map** is a term map that is represented by a resource that has exactly one rr:template property. The value of the rr:template property must be a valid string template.
- A **string template** is a format string that can be used to build strings from multiple components. It can reference column names by enclosing them in curly braces (“{” and “}”). The following syntax rules apply to valid string templates:
  - Pairs of unescaped curly braces must enclose valid column names.
  - Curly braces that do not enclose column names must be escaped by a backslash character (“\”). This also applies to curly braces within column names.
  - Backslash characters (“\”) must be escaped by preceding them with another backslash character, yielding “\\”. This also applies to backslashes within column names.
  - There should be **at least one pair** of unescaped curly braces.
  - If a template contains multiple pairs of unescaped curly braces, then any pair should be separated from the next one by a **safe separator**. This is any character or string that does not occur anywhere in any of the data values of either referenced column; or in the IRI-safe versions of the data values, if the term type is rr:IRI (see note below).

```
<TriplesMap2> a rr:TriplesMap;
    rr:logicalTable [ rr:tableName  "\"Sport\"" ];
    rr:subjectMap [ rr:template "http://example.com/resource/sport_{\"ID\"}_{\"ID\"}""; ];
    rr:predicateObjectMap
   [
            rr:predicate rdfs:label ;  rr:objectMap [ rr:column "\"Name\""; ];
   ];
            .
```

**Student**

| ID (PK) INTEGER | Sport (FK) INTEGER | Name VARCHAR(50) |
|---|---|---|
| 10 | 100 | Venus Williams |
| 20 | NULL | Demi Moore |

```
<TriplesMap2> a rr:TriplesMap;
    rr:logicalTable [ rr:tableName  "\"Sport\"" ];
    rr:subjectMap [ rr:template "http://example.com/resource/sport_{\"ID\"}”{\"ID\"}"; ];
    rr:predicateObjectMap
   [
            rr:predicate rdfs:label ;  rr:objectMap [ rr:column "\"Name\""; ];
   ];
            .
```

**Sport**

| ID (PK) INTEGER | Name VARCHAR(50) |
|---|---|
| 100 | Tennis |

# From template

- The template value of the term map for a given logical table row is determined as follows:
  - Let result be the template string
  - For each pair of unescaped curly braces in result:
    - Let value be the data value of the column whose name is enclosed in the curly braces
    - If value is NULL, then return NULL
    - Let value be the natural **RDF lexical form** corresponding to value
    - If the term type is rr:IRI, then replace the pair of curly braces with an **IRI-safe** version of **value**; otherwise, replace the pair of curly braces with value
  - Return result
- The IRI-safe version of a string is obtained by applying the following transformation to any character that is not in the iunreserved production in [RFC3987]:
  - Convert the character to a sequence of one or more octets using UTF-8 [RFC3629]
  - Percent-encode each octet [RFC3986]

## Notes

- R2RML always performs percent-encoding when IRIs are generated from string templates. If IRIs need to be generated without percent-encoding, then rr:column should be used instead of rr:template, with an R2RML view that performs the string concatenation.
- In the case of string templates that generate IRIs, any single character that is legal in an IRI, but percent-encoded in the IRI-safe version of a data value, is a safe separator. This includes in particular the eleven sub-delim characters defined in [RFC3987]: !$&'()*+,;=

## IRIs, Literal, Blank Nodes (rr:termType)

- The term type of a column-valued term map or template-valued term map determines the kind of generated RDF term (IRIs, blank nodes or literals).
- If the term map has an optional rr:termType property, then its term type is the value of that property. The value must be an IRI and must be one of the following options:
  - If the term map is a subject map: **rr:IRI or rr:BlankNode**
  - If the term map is a predicate map: **rr:IRI**
  - If the term map is an object map: **rr:IRI, rr:BlankNode, or rr:Literal**
  - If the term map is a graph map: **rr:IRI**

## IRIs, Literal, Blank Nodes (rr:termType)

- If the term map does not have a rr:termType property, then its term type is:
  - **rr:Literal**, if it is an object map and at least one of the following conditions is true:
    - It is a column-based term map.
    - It has a rr:language property (and thus a specified language tag).
    - It has a rr:datatype property (and thus a specified datatype).
  - **rr:IRI**, otherwise.

## Example:

```
<TriplesMap1>
    a rr:TriplesMap;

      rr:logicalTable [ rr:sqlQuery """
       SELECT ('Student' || "Student") AS StudentId,
                  "Student"
         FROM "Student_Sport"
       """;
                    ] ;

    rr:subjectMap [ rr:column "StudentId"; rr:termType rr:Literal;
                   rr:class ex:Student ];

    rr:predicateObjectMap
    [
      rr:predicate                      foaf:name ;
      rr:objectMap                      [ rr:column "\"Student\"" ]
    ]
    .
```

## IRIs, Literal, Blank Nodes (rr:termType)

**Term maps with term type rr:IRI cause data errors** if the value is not a valid IRI (see generated RDF term for details). Data values from the input database may require percent-encoding before they can be used in IRIs**. Template-valued term maps** are a convenient way of **percent-encoding** data values.

**Constant-valued term maps are not considered as having a term type**, and specifying rr:termType on these term maps has no effect. The type of the generated RDF term is determined directly by the value of rr:constant: If it is an IRI, then an IRI will be generated; if it is a literal, a literal will be generated.

## Exercise: model as values

| Patient | CancerType | Stage |
|---|---|---|
| Mary | Lung | 1 |

## Exercise: model as objects

| Patient | CancerType | Stage |
|---|---|---|
| Mary | | |

## Modeling as values vs. objects

- Values:
  - – Values allow for natural comparison and queries
  - – DB style, easy to grasp by beginners
  - – Complexity of the data is low
- Objects
  - – Expressivity high
  - – Data can be closer to the original conceptual model
  - – Possibility to apply hierarchies and reasoning over objects

## Language Tags (rr:language)

- **A term map with a term type of rr:Literal may have a specified language tag.** It is represented by the rr:language property on a term map. If present, its value must be a valid language tag.
- A specified language tag causes generated literals to be language-tagged plain literals

# Example

```
<TriplesMap1>
    a rr:TriplesMap;

            rr:logicalTable [  rr:sqlQuery """
                    SELECT "Code", "Name", "Lan"
                    FROM "Country"
                                                    WHERE "Lan" = 'EN';
                    """ ] ;

    rr:subjectMap [ rr:template "http://example.com/{\"Code\"}" ];

    rr:predicateObjectMap
    [
      rr:predicate                rdfs:label;
      rr:objectMap                [ rr:column "\"Name\""; rr:language "english" ]
    ]
    .
```

## Typed Literals (rr:datatype)

- A **datatypeable term map** is a term map with a term type of rr:Literal that does not have a specified language tag.
- Datatypeable term maps may generate typed literals. The datatype of these literals can be automatically determined based on the SQL datatype of the underlying logical table column (producing a natural RDF literal), or it can be explicitly overridden using rr:datatype (producing a datatype-override RDF literal).
- A datatypeable term map may have a rr:datatype property. Its value must be an IRI. This IRI is the specified datatype of the term map.
- A term map must not have more than one rr:datatype value.
- A term map that is not a datatypeable term map must not have an rr:datatype property.

## Typed Literals (rr:datatype)

- The **implicit SQL datatype** of a datatypeable term map is CHARACTER VARYING if the term map is a template-valued term map; otherwise, it is the SQL datatype of the respective column in the logical table row.
- See generated RDF term for further details on generating literals from term maps.

**One cannot explicitly state that a plain literal without language tag should be generated.** They are the default for string columns. To generate one from a non-string column, a template-valued term map with a template such as "{MY_COLUMN}" and a term type of rr:Literal can be used.

## Example

```
<TriplesMap1>
   a rr:TriplesMap;
   rr:logicalTable [ rr:sqlQuery """
      Select ('Department' || "deptno") AS deptId
           , "deptno"
           , "dname"
           , "loc"
        from "DEPT"
      """ ];

   rr:subjectMap [ rr:column "deptId"; rr:termType rr:BlankNode;
                  rr:inverseExpression "{\"deptno\"} =
substr({deptId},length('Department')+1)"];

   rr:predicateObjectMap
   [
     rr:predicate            dept:deptno ;
     rr:objectMap    [ rr:column "\"deptno\""; rr:datatype xsd:positiveInteger ]
   ];
```

## Inverse Expressions (rr:inverseExpression)

- An **inverse expression** is a string template associated with a column-valued term map or template-value term map. It is represented by the value of the rr:inverseExpression property. This property is optional and there must not be more than one for a term map.
- Inverse expressions are useful for optimizing term maps that reference derived columns in R2RML views. An inverse expression **specifies an expression that allows “reversing” of a generated RDF term and the construction of a SQL query that efficiently retrieves the logical table row from which the term was generated**. In particular, it allows the use of indexes on the underlying relational tables.
- Every pair of unescaped curly braces in the inverse expression is a column reference in an inverse expression. The string between the braces must be a valid column name.

## Example

```
<TriplesMap1>
    a rr:TriplesMap;
    rr:logicalTable [ rr:sqlQuery """
       SELECT ('Department' || "deptno") AS "deptId"
            , "deptno”   , "dname"
            , "loc"
       FROM "DEPT"
       """ ];

    rr:subjectMap [ rr:column "\"deptId\""; rr:termType rr:BlankNode;
                    rr:inverseExpression "{\"deptno\"} =
                                            substr({\"deptId\"},length('Department')+1)"];

    rr:predicateObjectMap
    [
      rr:predicate    dept:location ;
      rr:objectMap   [ rr:column "\"loc\"" ]
    ];
```

## Example

- The following table states distances in kilometers, generate RDF that presents this data in miles, add an inverse map that allows to reverse (1km = 0,62 mi).

| city1 | city2 | distance |
|---|---|---|
| bolzano | munich | 278,4 |

## Foreign Key Relationships among Logical Tables

- A referencing object map **allows using the subjects of another triples map as the objects generated by a predicate-object map**. Since both triples maps may be based on different logical tables, this may require a join between the logical tables. This is not restricted to 1:1 joins.
- A referencing object map is represented by a resource that:
  - has exactly **one rr:parentTriplesMap** property, whose value must be a triples map, known as the referencing object map's parent triples map.
  - may have **one or more rr:joinCondition** properties, whose values must be join conditions.

## Foreign Key Relationships among Logical Tables

- A **join condition** is represented by a resource that has exactly one value for each of the following two properties:
  - rr:child, whose value is known as the join condition's child column and must be a column name that exists in the logical table of the triples map that contains the referencing object map
  - rr:parent, whose value is known as the join condition's parent column and must be a column name that exists in the logical table of the referencing object map's parent triples map.
- **The child query** of a referencing object map is the effective SQL query of the logical table of the term map containing the referencing object map.
- The **parent query** of a referencing object map is the effective SQL query of the logical table of its parent triples map.

# Foreign Key Relationships among Logical Tables

- **The joint SQL query** of a referencing object map is:
  - If the referencing object map has no join condition:

    ```
    SELECT * FROM ({child-query}) AS tmp
    ```

  - If the referencing object map has at least one join condition:

    ```
    SELECT * FROM ({child-query}) AS child,
        ({parent-query}) AS parent
    WHERE child.{child-column1}=parent.{parent-column1}
    AND child.{child-column2}=parent.{parent-column2}
    AND ...
    ```

  where {child-query} is the referencing object map's child query, {parent-query} is its parent query, {child-column1} and {parent-column1} are the child column and parent column of its first join condition, and so on. The order of the join conditions is chosen arbitrarily.

  - **The joint SQL query is used when generating RDF triples** from referencing object maps.

## Assigning Triples to Named Graphs

- **Each triple generated from an R2RML mapping is placed into one or more graphs** of the output dataset. Possible target graphs are the unnamed default graph, and the IRI-named named graphs.
- **Any subject map or predicate-object map may have one or more associated graph maps**. They are specified in one of two ways:
  - using the rr:graphMap property, whose value must be a graph map,
  - using the constant shortcut property rr:graph.
- Graph maps are themselves term maps. When RDF triples are generated, the set of target graphs is determined by taking into account any graph maps associated with the subject map or predicate-object map.
- If a graph map generates the special IRI rr:defaultGraph, then the target graph is the default graph of the output dataset.

- In the following subject map example, all generated RDF triples will be stored in the named graph ex:DepartmentGraph.

```
[] rr:subjectMap [
    rr:template "http://data.example.com/department/{DEPTNO}";
    rr:graphMap [ rr:constant ex:DepartmentGraph ];
].
```

- This is equivalent to the following example, which uses a constant shortcut property:

```
[] rr:subjectMap [
    rr:template "http://data.example.com/department/{DEPTNO}";
    rr:graph ex:DepartmentGraph;
].
```

- In the following example, RDF triples are placed into named graphs according to the job title of employees:

```
[] rr:subjectMap [
    rr:template "http://data.example.com/employee/{EMPNO}";
    rr:graphMap [ rr:template "http://data.example.com/jobgraph/{JOB}" ];
].
```

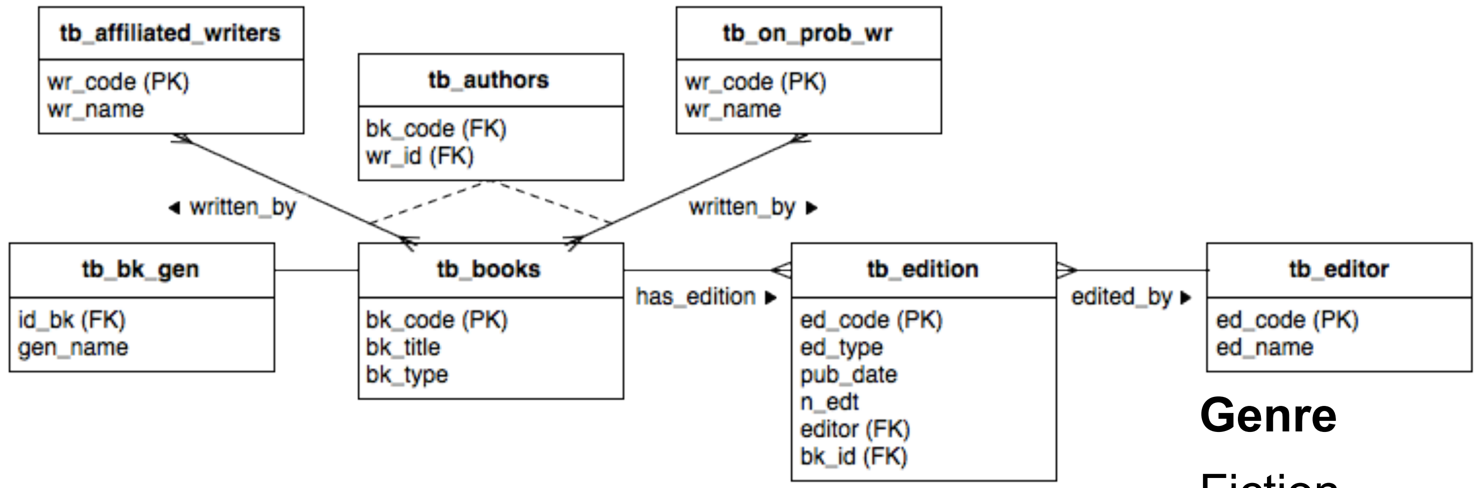
- This is equivalent to the following example, which uses a constant shortcut property:

```
<http://data.example.com/jobgraph/CLERK>
```

- Detailed Specification
- Books Example
- Tools

# Books DB

- Propose a Class/Property vocabulary
- Propose a URI scheme for each entity type
- Define an R2RML mapping and an OWL ontology such that a) the number of mappings is minimal, b) data is correctly typed c) language and type are specified when necessary



**Genre**

Fiction

Horror

Mystery

Fantasy

Romance

Biography

…

**Book types**

A = Audio Book

E = E-Book

P = Paper back

**Edition type**

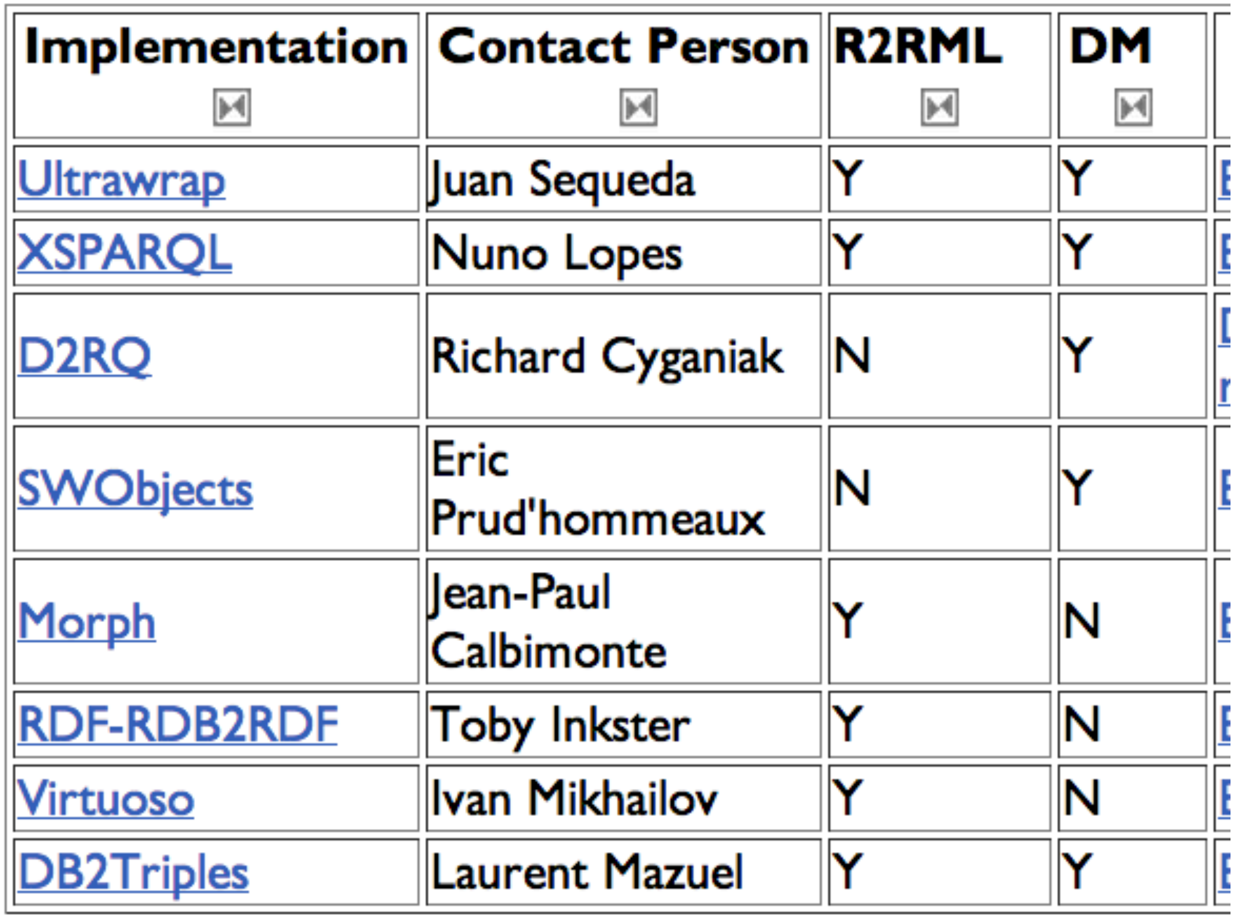
S = Special/Limited

## SPARQL Target queries

- The ontology/mappings should allow for easy (preferably avoiding UNION) querying for the following data:
  - RE-001: Query for all the author names (22 tuples)
  - RE-002: Query for all the book titles (24 tuples).
  - RE-003: Query for all the editor names (11 tuples).
  - RE-004: Query for all the titles that all audio books.
  - RE-005: Query for all the names for all authors that are an emerging writer.
  - RE-006: Query for all the dates of publication and edition numbers of books that are special editions.
  - RE-007: Query for all the book titles written by a specific author.
  - RE-008: Query for the complete information about a book, including its author, genre, publication date and edition number.

- Detailed Specification
- Books Example
- Tools

## Implementations



| Implementation | Contact Person | R2RML | DM |
|---|---|---|---|
| Ultrawrap | Juan Sequeda | Y | Y |
| XSPARQL | Nuno Lopes | Y | Y |
| D2RQ | Richard Cyganiak | N | Y |
| SWObjects | Eric Prud'hommeaux | N | Y |
| Morph | Jean-Paul Calbimonte | Y | N |
| RDF-RDB2RDF | Toby Inkster | Y | N |
| Virtuoso | Ivan Mikhailov | Y | N |
| DB2Triples | Laurent Mazuel | Y | Y |