W3C Recommendation

A Direct Mapping of Relational Data to RDF

W3C Recommendation 27 September 2012

This version:
http://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/
Latest version:
http://www.w3.org/TR/rdb-direct-mapping/
Previous version:
http://www.w3.org/TR/2012/PR-rdb-direct-mapping-20120814/
Editors:
Marcelo Arenas, Pontificia Universidad Católica de Chile <marenas@ing.puc.cl>
Alexandre Bertails, W3C <bertails@w3.org>
Eric Prud'hommeaux, W3C <eric@w3.org>
Juan Sequeda, University of Texas at Austin <jsequeda@cs.utexas.edu>

Please refer to the **errata** for this document, which may include some normative corrections.

See also **translations**.

Abstract

The need to share data with collaborators motivates custodians and users of relational databases (RDB) to expose relational data on the Web of Data. This document defines a **direct mapping** from relational data to RDF. This definition provides extension points for refinements within and outside of this document.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document was published by the RDB2RDF Working Group. Comments on this document should be sent to public-rdb2rdf-comments@w3.org, a mailing list with a public archive. The following related documents have been made available:

- A wiki page with additional information for users and implementers of R2RML,
- a color-coded diff of all changes since the previous draft,
- the implementation report used by the director to transition to W3C Recommendation.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

Appendices

---

1 Introduction

Relational databases proliferate both because of their efficiency and their precise definitions, allowing for tools like SQL [SQLFN] to manipulate and examine the contents predictably and efficiently. Resource Description Framework (RDF) [RDF-concepts] is a data format based on a web-scalable architecture for identification and interpretation of terms. This document defines a mapping from relational representation to an RDF representation.

Strategies for mapping relational data to RDF abound. The **direct mapping** defines a simple transformation, providing a basis for defining and comparing more intricate transformations. It can also be used to materialize RDF graphs or define virtual graphs, which can be queried by SPARQL or traversed by an RDF graph API. This document includes an informal and a formal description of the transformation.

This specification has a companion, the R2RML mapping language [R2RML], that allows the creation of customized mapping from relational data to RDF. R2RML defines a relaxed variant of the Direct Mapping intended as a default mapping for further customization.

2 Direct Mapping Description (Informative)

The direct mapping defines an RDF Graph [RDF-concepts] representation of the data in a relational database. The direct mapping takes as input a relational database (data and schema), and generates an RDF graph that is called the **direct graph**. The algorithms in this document compose a graph of relative IRIs which must be resolved against a **base** IRI [RFC3987] to form an RDF graph.

Foreign keys in relational databases establish a reference from any row in a table to exactly one row in a (potentially different) table. The direct graph conveys these references, as well as each value in the row.

2.1 Direct Mapping Example

The concepts in direct mapping can be introduced with an example RDF graph produced by a relational database. Following is SQL (DDL) to create a simple example with two tables with single-column primary keys and one foreign key reference between them:

```
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)
```

HTML tables will be used in this document to convey SQL tables. The primary key of these tables will be marked with the `PK` class to convey an SQL primary key such as `ID` in `CREATE TABLE "Addresses" ("ID" INT, ... PRIMARY KEY("ID"))`. Foreign keys will be illustrated with a notation like "`→ Address(ID)`" to convey an SQL foreign key such as `CREATE TABLE "People" (... "addr" INT, FOREIGN KEY("addr") REFERENCES "Addresses"("ID"))`.

People

PK		*→ Address(ID)*
ID	**fname**	**addr**
7	Bob	18
8	Sue	**NULL**

Addresses

PK		
ID	**city**	**state**
18	Cambridge	MA

Given a base IRI `http://foo.example/DB/`, the direct mapping of this database produces a direct graph:

```
@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" .
```

In this expression, each row, e.g. `(7, "Bob", 18)`, 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. The values are RDF literals formed from the lexical form of the column value. 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. Note that these reference row identifiers must coincide with the subject used for the triples generated from the referenced row. The direct mapping does not generate triples for NULL values. Note that it is not known how to relate the behavior of the obtained RDF graph with the standard SQL semantics of the NULL values of the source RDB.

2.2 Foreign keys referencing candidate keys

More complex schemas include composite keys. In this example, the columns **deptName** and **deptCity** in the **People** table reference **name** and **city** in the **Department** table:

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

CREATE TABLE "Department" (
        "ID" INT,
        "name" CHAR(10),
        "city" CHAR(10),
        "manager" INT,
        PRIMARY KEY("ID"),
        UNIQUE ("name", "city")
)

CREATE TABLE "People" (
        "ID" INT,
        "fname" CHAR(10),
        "addr" INT,
        "deptName" CHAR(10),
        "deptCity" CHAR(10),
        PRIMARY KEY("ID"),
        FOREIGN KEY("addr") REFERENCES "Addresses"("ID"),
        FOREIGN KEY("deptName", "deptCity") REFERENCES "Department"("name", "city")
)

ALTER TABLE "Department" ADD FOREIGN KEY("manager") REFERENCES "People"("ID")
```

Following is an instance of this schema:

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

Per the **People** table's compound foreign key to Department:

- The row in **People** with `deptName="accounting"` and `deptCity="Cambridge"` references a row in **Department** with a primary key of `ID=23`.
- The predicate for this key is formed from "`deptName`" and "`deptCity`", reflecting the order of the column names in the foreign key.
- The object of the above predicate is formed from the base IRI, the table name "`Department`" and the primary key value "`ID=23`".

Note: The order of a primary key constraint's columns is determined by the DDL statement used to create it. In SQL implementations that support the information schema, this order can be accessed through the `INFORMATION_SCHEMA.KEY_COLUMN_USAGE.ORDINAL_POSITION` column.

In this example, the direct mapping generates the following triples:

```
@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> .
```

The green triples above are generated by considering the new elements in the augmented database. Note:

- 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).

2.3 Multi-column primary keys

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" .
```

2.4 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:

```
CREATE TABLE "Tweets" (
        "tweeter" INT,
        "when" TIMESTAMP,
        "text" CHAR(140),
        FOREIGN KEY("tweeter") REFERENCES "People"("ID")
)
```

The following is an instance of table **Tweets**:

Tweets

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

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:

```
@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." .
```

2.5 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.

Here is DDL for a schema with references to a `Projects` table which has no primary key:

```
CREATE TABLE "Projects" (
        "lead" INT,
        FOREIGN KEY ("lead") REFERENCES "People"("ID"),
        "name" VARCHAR(50),
        UNIQUE ("lead", "name"),
```

```
        "deptName" VARCHAR(50),
        "deptCity" VARCHAR(50),
        UNIQUE ("name", "deptName", "deptCity"),
        FOREIGN KEY ("deptName", "deptCity") REFERENCES "Department"("name", "city")
)

CREATE TABLE "TaskAssignments" (
        "worker" INT,
        FOREIGN KEY ("worker") REFERENCES "People"("ID"),
        "project" VARCHAR(50),
        PRIMARY KEY ("worker", "project"),
        "deptName" VARCHAR(50),
        "deptCity" VARCHAR(50),
        FOREIGN KEY ("worker") REFERENCES "People"("ID"),
        FOREIGN KEY ("project", "deptName", "deptCity") REFERENCES "Projects"("name", "deptName", "deptCity"),
        FOREIGN KEY ("deptName", "deptCity") REFERENCES "Department"("name", "city")
)
```

The following is an instance of the preceding schema:

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

In this case, the direct mapping generates the following triples from the preceding tables:

```
@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 .
```

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.

3 Direct Graph Definition

The Direct Graph is a formula for creating an RDF graph from the rows of each table and view in a database schema. A base IRI defines a web space for the IRIs in this graph; for the purposes of this specification, all IRIs are generated by appending to a base. Terms enclosed in <> are defined in the SQL specification [SQLFN].

An SQL table has a set of uniquely-named columns and has a set of foreign keys, each mapping a <column name list> to a <unique column list> (a list of columns in some table).

SQL table and column identifiers compose RDF IRIs in the direct graph. These identifiers are separated by the punctuation characters '#', ';', '/' and '='. All SQL identifiers are escaped following R2RML's escaping rules.

> Definition **percent-encode**:
>
> - Replace the string with the IRI-safe form per section 7.3 of [R2RML].

There is either a blank node or IRI assigned to each each row in a table:

Definition **row node**:

- If the table has a primary key, the row node is a relative IRI obtained by concatenating:
  - the percent-encoded form of the table name,
  - the SOLIDUS character '/',
  - for each column in the primary key, in order:
    - the percent-encoded form of the column name,
    - a EQUALS SIGN character '=',
    - the percent-encoded lexical form of the canonical RDF literal representation of the column value as defined in R2RML section 10.2 Natural Mapping of SQL Values [R2RML],
    - if it is not the last column in the primary key, a SEMICOLON character ';'
- If the table has no primary key, the row node is a fresh blank node that is unique to this row.

A table forms a table IRI:

Definition **table IRI**: the relative IRI consisting of the percent-encoded form of the table name.

A column in a table forms a literal property IRI:

Definition **literal property IRI**: the concatenation of:

- the percent-encoded form of the table name,
- the hash character '#',
- the percent-encoded form of the column name.

A foreign key in a table forms a reference property IRI:

Definition **reference property IRI**: the concatenation of:

- the percent-encoded form of the table name,
- the string '#ref-',
- for each column in the foreign key, in order:
  - the percent-encoded form of the column name,
  - if it is not the last column in the foreign key, a SEMICOLON character ';'

Any input database with a given schema has a **direct graph** defined as:

Definition **direct graph**: the union of the table graphs for each table in a database schema.

Definition **table graph**: the union of the row graphs for each row in a table.

Definition **row graph**: an RDF graph consisting of the following triples:

- the row type triple.
- a reference triple for each <column name list> in a table's foreign keys where none of the column values is NULL.
- a literal triple for each column in a table where the column value is non-NULL.

Definition **row type triple**: an RDF triple with:

- subject: the row node for the row.
- predicate: the RDF IRI `rdf:type`.
- object: the table IRI for the table name.

Definition **literal triple**: an RDF triple with:

- subject: the row node for the row.
- predicate: the literal property IRI for the column.
- object: the R2RML natural RDF literal representation of the column value as defined in R2RML section 10.2 Natural Mapping of SQL Values.

Definition **reference triple**: an RDF triple with:

- subject: the row node for the row.
- predicate: the reference property IRI for the columns.
- object: the row node for the referenced row.

4 References

R2RML
R2RML: RDB to RDF Mapping Language , Souripriya Das, Seema Sundara, Richard Cyganiak, Editors. World Wide Web Consortium, 27 September 2012. This version is http://www.w3.org/TR/2012/REC-r2rml-20120927/. The latest version is http://www.w3.org/TR/r2rml/. This document is work in progress.

SPARQL
SPARQL Query Language for RDF, Eric Prud'hommeaux and Andy Seaborne 2008. (See http://www.w3.org/TR/rdf-sparql-query/.)

SQLFW
: SQL. ISO/IEC 9075-1:2008 SQL – Part 1: Framework (SQL/Framework) International Organization for Standardization, 27 January 2009.

SQLFN
: ISO/IEC 9075-2:2008 SQL – Part 2: Foundation (SQL/Foundation) International Organization for Standardization, 27 January 2009.

RDF-concepts
: Resource Description Framework (RDF): Concepts and Abstract Syntax, G. Klyne, J. J. Carroll, Editors, W3C Recommendation, 10 February 2004 (See http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.)

ReuseableIDs
: Reusable Identifiers in the RDB2RDF mapping language, Michael Hausenblas and Themis Palpanas, 2009. (See http://esw.w3.org/topic/Rdb2RdfXG/ReusableIdentifier.)

URI
: RFC3986 - Uniform Resource Identifier (URI): Generic Syntax (See http://tools.ietf.org/html/rfc3986.)

RFC3987
: RFC3987 - Internationalized Resource Identifier (IRIs) (See http://tools.ietf.org/html/rfc3987.)

SQL2SW
: Translating SQL Applications to the Semantic Web. Syed Hamid Tirmizi, Juan Sequeda and Daniel Miranker. 2008 (See http://www.springerlink.com/content/mv58805364k31734/.)

DMSurvey
: Survey of directly mapping SQL databases to the Semantic Web. Juan Sequeda, Syed Hamid Tirmizi, Oscar Corcho, Daniel P. Miranker. 2011 (See http://journals.cambridge.org/abstract_S0269888911000208.)

A Direct Mapping Algebra (Informative)

A.1 Notations

The RDB and RDF data models make use of the commonly defined Abstract Data Types Set, List and MultiSet, used here as type constructors. For example, `Set(A)` denotes the type for the sets of elements of type `A`. We assume that they come with their common operations, such as the function `size : Set → Int`.

The definitions follow a type-as-specification approach; thus the models are based on dependent types. For example, `{ s:Set(A) | size(s) ≤ 1 }` is a type denoting the sets for elements of type A, such that those sets have at most one element.

The denotational RDF semantics makes use of the set-builder notation for building the RDF sets.

The buttons below can be used to show or hide the available syntaxes.

Set notation (s) Show English Syntax (e)

A.2 Relational Data Model

A.2.1 RDB Abstract Data Type

[1] `Database` ::= `Set(Table)`
[2] `Table` ::= `(TableName, Set((ColumnName, Datatype)), Set(CandidateKey), Set(PrimaryKey) | size() ≤ 1, Set(ForeignKey), Body)`
[3] `Body` ::= `MultiSet(Row)`
[4] `Row` ::= `Set((ColumnName, CellValue))`
[5] `CellValue` ::= `Value | NULL`
[6] `ForeignKey` ::= `(List(ColumnName), Table, CandidateKey)`
[7] `PrimaryKey` ::= `CandidateKey`
[8] `CandidateKey` ::= `List(ColumnName)`
[9] `Datatype` ::= `Int | Float | Date | …`
[10] `TableName` ::= `String`
[11] `ColumnName` ::= `String`

A.2.2 RDB accessor functions

[12] `tablename` : `Table → TableName`
[13] `header` : `Table → Set((ColumnName, Datatype))`
[14] `candidateKeys` : `Table → List(CandidateKey)`
[15] `primaryKey` : `Table → { s:Set(CandidateKey) | size(s) ≤ 1 }`
[16] `foreignKeys` : `Table → Set(ForeignKey)`
[17] `unary` : `ForeignKey → Boolean`
[18] `lexicals` : `Table → Set({ c:ColumnName | ! unary(c) })`
[19] `body` : `Table → Body`
[20] `datatype` : `{ h:Set((ColumnName, Datatype)) } → { c:ColumnName | ∃ d, (c,d) ∈ h } → { d:Datatype | (c,d) ∈ h }`
[21] `table` : `{ r:Row } → { t:Table | r ∈ t }`
[22] `value` : `{ r:Row } → { a:ColumnName | a ∈ r } → CellValue`
[23] `dereference` : `{ r:Row } → { fk:ForeignKey | fk ∈ foreignKeys(table(r)) }`

$$\begin{array}{l} \rightarrow \{\ \mathtt{targetRow{:}Row} \mid \textbf{let}\ (\mathtt{columnNames, targetTable, ck}) = \mathtt{fk}\ \textbf{in} \\ \quad \mathtt{targetRow} \in \mathtt{body(targetTable)} \\ \quad \textbf{and}\ \forall\ c_i^{fk} \in \mathtt{columnNames},\ \forall\ c_j^{ck} \in \mathtt{ck}, \\ \quad\quad \forall\ (c_k^{r}, v_k^{r}) \in r,\ \forall\ (c_l^{target}, v_l^{target}) \in \mathtt{targetRow}, \\ \quad\quad i = j \rightarrow c_i^{fk} = c_k^{r} \rightarrow c_j^{ck} = c_l^{target} \rightarrow v_k^{r} = v_l^{target}\ \} \end{array}$$

A.3 RDF Data Model

Per RDF Concepts and Abstract Syntax, an RDF graph is a set of triples of a subject, predicate and object. The subject may be an IRI or a blank node, the predicate must be an IRI and the object may be an IRI, blank node, or an RDF literal.

This section recapitulates for convience the formal definition of RDF.

```
[24] Graph        ::= Set(Triple)
[25] Triple       ::= (Subject, Predicate, Object)
[26] Subject      ::= IRI | BlankNode
[27] Predicate    ::= IRI
[28] Object       ::= IRI | BlankNode | Literal
[29] BlankNode    ::= RDF blank node
[30] Literal      ::= PlainLiteral | TypedLiteral
[31] PlainLiteral ::= lexicalForm | (lexicalForm, langageTag)
[32] TypedLiteral ::= (lexicalForm, IRI)
[33] IRI          ::= RDF URI-reference as subsequently restricted by SPARQL
[34] lexicalForm  ::= a Unicode String
```

A.4 Denotational semantics

In this model, Databases are inhabitants of RDB and they are denoted by mathematical objects living in the RDF domain. This **denotational semantics** is what we call the **Direct Mapping**.

The url-encoding function renders strings in a form suitable to insert into IRIs. Data values are expressed in the XML Schema canonical form before url-encoding.

```
[35] ue            : String → String
     □ , □canon    : (Row, Column) → String
[36] □r, c□canon   = let v = value(r, c) in
                     let d = header(table(r)) in
                     canonical RDF literal(v, d)
```

Most of the functions defining the Direct Mapping are higher-order functions parameterized by a function φ(r) which maps any row to a unique IRI or Blank Node.

```
[37] φ             : ∀ db:Database, ∀ r:Row, r ∈ db
                     if primaryKey(table(r)) ≠ ∅ then
                        ue(tablename(table(r))) + '/' + ue(c_0) + '=' + ue(canon(r, c_0)) + ';'
                        + ⋯ + ';' + ue(c_{n-1}) + '=' + ue(canon(r, c_{n-1}))
                     else
                        a BlankNode unique to r
     □ □tableIRI   : TableName → IRI
[38] □t□tableIRI   = ue(tablename(t))
     □ , □litcol   : (Row, Column) → IRI
[39] □r, c□litcol  = ue(tablename(table(r))) + '#' + ue(c))
     □ , □refcol   : (Row, ForeignKey) → IRI
[40] □r, fk□refcol = let(from*, reftable, to*) = fk in
                       ue(tablename(table(r))) + '/ref-'
                     + ue(from_0) + ';' + ⋯ + ';' + ue(from_{n-1})
```

The Direct Mapping is defined by induction on the structure of RDB. Thus it is defined for any relational database. The entry point for the Direct Mapping is the function □ □^φ_database .

```
     □ □^φ_database      : Database → Graph
[41] □db□^φ_database     = { triple | triple ∈ □t□^φ_table | t ∈ db }
     □ □^φ_table         : Table → Set(Triple)
[42] □t□^φ_table         = { triple | triple ∈ □r□^φ_row | r ∈ body(t) }
     noNULLs             : Row → ForeignKey → Boolean
[43] noNULLs(r, fk)      = let (columnNames, _, _) = fk in
                           ∀ c ∈ columnNames, value(r, c) ≠ NULL
     □ □^φ_row           : Row → Set(Triple)
[44] □r□^φ_row           = let s = φ(r) in
                              { □r□^φ_type }
                           ∪ { □r, c□^φ_lex | value(r, c) ≠ NULL | c ∈ lexicals(r) }
                           ∪ { □r, fk□^φ_ref | noNULLs(r, fk) | fk ∈ foreignKeys(table(r)) }
     □ □type             : (Row) → Triple
[45] □r□type             = let s = φ(r) in
                           let t = table(r) in
                           let o = □t□tableIRI in
                           { (s, rdf:type, o) }

     □ , □lex            : (Row, Column) → Triple
[46] □r, c□lex           = let s = φ(r) in
```

```
[46] □r, c□lex        let s = φ(r) in
                      let p = □table(r), c□litcol in
                      let v = value(r, c) in
                      let d = header(table(r)) in
                      if v is NULL then ∅
                      else let o = natural RDF literal(v, d) in
                           { (s, p, o) }
     □ ,   □ref     : (Row, ForeignKey) → Triple
[47] □r, fk□ref     = let s = φ(r) in
                      let targetSpec = dereference(r, fk) in
                      let p = □table(r), fk□refcol in
                      let o = φ(row(targetSpec)) in
                      (s, p, o)
```

B Direct Mapping as Rules (Informative)

In this section, we formally present the Direct Mapping as rules in Datalog syntax, inspired by the previous approaches in [SQL2SW] [DMSurvey]. The left hand side of each rule is the RDF Triple output. The right hand side of each rule consists of a sequence of predicates from the relational database and built-in predicates. The built-in predicates are divided into four groups. The first group contains some built-in predicates for dealing with repeated rows in a table without a primary key.

- card(r, l, k): Given a table name r without a primary key and the list l of values [v_1, ..., v_n] for a row of table r, it returns in k the multiplicity of l in r (that is, k is the number of times row l appears in r)
- n ≤ m: This is the usual order on positive integer values (given positive integers n and m, it holds if n is smaller than or equal to m)

The second group contains a predicate to deal with null values.

- nonNull(v): Given a value v, it holds if v is not null

The third group of built-in predicates is used to generate IRIs for identifying tables and the columns in a table, and to generate IRIs or blank nodes for identifying each row in a table.

- generateTableIRI(r, i): Given a table name r, it generates the table IRI i of r
- generateLiteralPropertyIRI(r, a, i): Given a table name r and an attribute name a, it generates the literal property IRI i for a
- generateReferencePropertyIRI(r, l, i): Given a table name r and a non-empty list of columns l, it generates the reference property IRI i for l
- generateRowIRI(r, l_1, l_2, i): Given a table name r, a non-empty list l_1 of columns and a non-empty list l_2 of values (for the columns in l_1), it generates the row node (or Row IRI) i for the given row
- generateRowBlankNode(r, l, n, i): Given a table name r without a primary key, a list l of values for a row of table r and a positive integer n, it generates the row node i for the n-th occurrence of row l in r (which is a Blank Node in this case). It is assumed that n is smaller than or equal to the multiplicity of l in r (that is, if card(r, l, k) holds, then 1 ≤ n ≤ k)

Finally, the fourth group of built-in predicates is used to generate typed literals.

- generateTypedLiteral(u, a, r, v): Given a value u, an attribute name a and a table name r, it generates an R2RML natural RDF literal v representation of the column value u, given the type of a in r and as defined in R2RML section 10.2 Natural Mapping of SQL Values.

Throughout the section, boxes containing Direct Mapping rules and examples will appear. These boxes are color-coded. Yellow boxes contain Direct Mapping rules:

```
This box contains a Direct Mapping rule
```

Green boxes contain examples of applying the previous Direct Mapping rule:

```
This box contains examples of applying a Direct Mapping rule
```

Consider again the example from Section Direct Mapping Example. It should be noticed that in the rules presented in this section, a formula of the form Addresses(X, Y, Z) indicates that the variables X, Y and Z are used to store the values of a row in the three columns of the table Addresses (according to the order specified in the schema of the table, that is, X, Y and Z store the values of ID, city and state, respectively). In particular, uppercase letters like X, Y, Z, S, P and O are used to denote variables. Moreover, double quotes are used in the rules to refer to the string with the name of a table or a column. For example, a formula of the form generateRowIRI("Addresses", ["ID"], [X], S) is used to generate the row node (or Row IRI) for the row of table "Addresses" whose value in the primary key "ID" is the value stored in the variable X. The value of this Row IRI is stored in the variable S.

B.1 Generating Row Type Triples

B.1.1 Table has a primary key

Assume that r is a table with columns a_1, ..., a_m and such that [a_{p_1}, ..., a_{p_n}] is the primary key of r, where 1 ≤ n ≤ m and 1 ≤ p_1 < ... < p_n ≤ m. Then the following is the direct mapping rule to generate row type triples from r:

```
Triple(S, "rdf:type", O) ← r(X1, ..., Xm), generateRowIRI("r", ["ap1", ..., "apn"], [Xp1, ..., Xpn], S), generateTableIRI("
```

For example, table **Addresses** in the Direct Mapping Example has columns **ID**, **city** and **state**, and it has column **ID** as its primary key. Then the following is the direct mapping rule to generate row type triples from **Addresses**:

```
Triple(S, "rdf:type", O) ← Addresses(X1, X2, X3), generateRowIRI("Addresses", ["ID"], [X1], S), generateTableIRI("Addresse
```

As a second example, consider table **Department** from the example in Section Foreign keys referencing candidate keys, which has columns **ID**, **name**, **city** and **manager**, and assume that (**name**, **city**) is the multi-column primary key of this table (instead of **ID**). Then the following is the direct mapping rule to generate row type triples from **Department**:

```
Triple(S, "rdf:type", O) ← Department(X1, X2, X3, X4), generateRowIRI("Department", ["name","city"], [X2, X3], S),
                           generateTableIRI("Department", O)
```

B.1.2 Table does not have a primary key

Assume that r is a table with columns a_1, ..., a_m and such that r does not have a primary key. Then the following is the direct mapping rule to generate row type triples from r:

```
Triple(S, "rdf:type", O) ← r(X1, ..., Xm), card("r", [X1, ..., Xm], U), V ≤ U, generateRowBlankNode("r", [X1, ..., Xm], V,
                           generateTableIRI("r", O)
```

For example, table **Tweets** from Section Empty (non-existent) primary keys has columns **tweeter**, **when** and **text**, and it does not have a primary key. Then the following is the direct mapping rule to generate row type triples from **Tweets**:

```
Triple(S, "rdf:type", O) ← Tweets(X1, X2, X3), card("Tweets", [X1, X2, X3], U), V ≤ U, generateRowBlankNode("Tweets", [X1,
                           generateTableIRI("Tweets", O)
```

B.2 Generating Literal Triples

B.2.1 Table has a primary key

Assume that r is a table with columns a_1, ..., a_m and such that [a_{p_1}, ..., a_{p_n}] is the primary key of r, where $1 \le n \le m$ and $1 \le p_1 < ... < p_n \le m$. Then for every a_j ($1 \le j \le m$), the direct mapping includes the following rule for r and a_j to generate literal triples:

```
Triple(S, P, V) ← r(X1, ..., Xm), nonNull(Xj), generateRowIRI("r", ["ap1", ..., "apn"], [Xp1, ..., Xpn], S),
                  generateLiteralPropertyIRI("r", "aj", P), generateTypedLiteral(Xj, "aj", "r", V)
```

For example, table **Addresses** in the Direct Mapping Example has columns **ID**, **city** and **state**, and it has column **ID** as its primary key. Then the following are the direct mapping rules to generate literal triples from **Addresses**:

```
Triple(S, P, V) ← Addresses(X1, X2, X3), nonNull(X1), generateRowIRI("Addresses", ["ID"], [X1], S),
                  generateLiteralPropertyIRI("Addresses", "ID", P), generateTypedLiteral(X1, "ID", "Addresses", V)
Triple(S, P, V) ← Addresses(X1, X2, X3), nonNull(X2), generateRowIRI("Addresses", ["ID"], [X1], S),
                  generateLiteralPropertyIRI("Addresses", "city", P), generateTypedLiteral(X2, "city", "Addresses", V)
Triple(S, P, V) ← Addresses(X1, X2, X3), nonNull(X3), generateRowIRI("Addresses", ["ID"], [X1], S),
                  generateLiteralPropertyIRI("Addresses", "state", P), generateTypedLiteral(X3, "state", "Addresses", V)
```

As a second example, consider again table **Department** from the example in Section Foreign keys referencing candidate keys, which has columns **ID**, **name**, **city** and **manager**, and assume that (**name**, **city**) is the multi-column primary key of this table (instead of **ID**). Then the following are the direct mapping rules to generate literal triples from **Department**:

```
Triple(S, P, V) ← Department(X1, X2, X3, X4), nonNull(X1), generateRowIRI("Department", ["name", "city"], [X2, X3], S),
                  generateLiteralPropertyIRI("Department", "ID", P), generateTypedLiteral(X1, "ID", "Department", V)
Triple(S, P, V) ← Department(X1, X2, X3, X4), nonNull(X2), generateRowIRI("Department", ["name", "city"], [X2, X3], S),
                  generateLiteralPropertyIRI("Department", "name", P), generateTypedLiteral(X2, "name", "Department", V)
Triple(S, P, V) ← Department(X1, X2, X3, X4), nonNull(X3), generateRowIRI("Department", ["name", "city"], [X2, X3], S),
                  generateLiteralPropertyIRI("Department", "city", P), generateTypedLiteral(X3, "city", "Department", V)
Triple(S, P, V) ← Department(X1, X2, X3, X4), nonNull(X4), generateRowIRI("Department", ["name", "city"], [X2, X3], S),
                  generateLiteralPropertyIRI("Department", "manager", P), generateTypedLiteral(X4, "manager", "Departmen
```

B.2.2 Table does not have a primary key

Assume that r is a table with columns a_1, ..., a_m and such that r does not have a primary key. Then for every a_j ($1 \le j \le m$), the direct mapping includes the following rule for r and a_j to generate literal triples:

```
Triple(S, P, V) ← r(X1, ..., Xm), nonNull(Xj), card("r", [X1, ..., Xm], U), V ≤ U, generateRowBlankNode("r", [X1, ..., Xm]
                  generateLiteralPropertyIRI("r", "aj", P), generateTypedLiteral(Xj, "aj", "r", V)
```

For example, table **Tweets** from Section Empty (non-existent) primary keys has columns **tweeter**, **when** and **text**, and it does not have a primary key. Then the following is the direct mapping rule to generate row type triples from **Tweets**:

primary key. Then the following are the direct mapping rules to generate [literal triples](#) from **Tweets**:

```
Triple(S, P, V) ← Tweets(X1, X2, X3), nonNull(X1), card("Tweets", [X1, X2, X3], U), V ≤ U, generateRowBlankNode("Tweets",
                  generateLiteralPropertyIRI("Tweets", "tweeter", P), generateTypedLiteral(X1, "tweeter", "Tweets", V)
Triple(S, P, V) ← Tweets(X1, X2, X3), nonNull(X2), card("Tweets", [X1, X2, X3], U), V ≤ U, generateRowBlankNode("Tweets",
                  generateLiteralPropertyIRI("Tweets", "when", P), generateTypedLiteral(X2, "when", "Tweets", V)
Triple(S, P, V) ← Tweets(X1, X2, X3), nonNull(X3), card("Tweets", [X1, X2, X3], U), V ≤ U, generateRowBlankNode("Tweets",
                  generateLiteralPropertyIRI("Tweets", "text", P), generateTypedLiteral(X3, "text", "Tweets", V)
```

B.3 Generating Reference Triples

For each foreign key from a table r_1 to a table r_2, one of the following four cases is applied.

B.3.1 Table r_1 has a primary key and table r_2 has a primary key

Assume that:

- r_1 is a table with columns $a_1, ..., a_i$ and such that $[a_{p_1}, ..., a_{p_j}]$ is the primary key of r_1, where $1 \le j \le i$ and $1 \le p_1 < ... < p_j \le i$
- r_2 is a table with columns $c_1, ..., c_k$ and such that $[c_{q_1}, ..., c_{q_m}]$ is the primary key of r_2, where $1 \le m \le k$ and $1 \le q_1 < ... < q_m \le k$
- the foreign key indicates that the columns $a_{s_1}, ..., a_{s_n}$ of r_1 reference the columns $c_{t_1}, ..., c_{t_n}$ of r_2, where (1) $1 \le s_1, ..., s_n \le i$, (2) $1 \le t_1, ..., t_n \le k$, and (3) $n \ge 1$

Then the direct mapping includes the following rule for r_1 and r_2 to generate [Reference Triples:](#)

```
Triple(S, P, O) ← r1(X1, ..., Xi), generateRowIRI("r1", ["a_p1", ..., "a_pj"], [X_p1, ..., X_pj], S),
                  r2(Y1, ..., Yk), generateRowIRI("r2", ["c_q1", ..., "c_qm"], [Y_q1, ..., Y_qm], O),
                  nonNull(X_s1), ..., nonNull(X_sn), X_s1 = Y_t1, ..., X_sn = Y_tn, generateReferencePropertyIRI("r1", ["a_s1",
```

For example, table **Addresses** in the [Direct Mapping Example](#) has columns **ID**, **city** and **state**, where column **ID** is the primary key. Table **People** in this example has columns **ID**, **fname** and **addr**, where column **ID** is the primary key, and it has a foreign key in the column **addr** that references the column **ID** in the table **Addresses**. In this case, the following is the direct mapping rule to generate [Reference Triples](#):

```
Triple(S, P, O) ← People(X1, X2, X3), generateRowIRI("People", ["ID"], [X1], S),
                  Addresses(Y1, Y2, Y3), generateRowIRI("Addresses", ["ID"], [Y1], O),
                  nonNull(X3), X3 = Y1, generateReferencePropertyIRI("People", ["addr"], P)
```

B.3.2 Table r_1 has a primary key and table r_2 does not have a primary key

Assume that:

- r_1 is a table with columns $a_1, ..., a_i$ and such that $[a_{p_1}, ..., a_{p_j}]$ is the primary key of r_1, where $1 \le j \le i$ and and $1 \le p_1 < ... < p_j \le i$
- r_2 is a table with columns $c_1, ..., c_k$, and it does not have a primary key
- the foreign key indicates that the columns $a_{s_1}, ..., a_{s_n}$ of r_1 reference the columns $c_{t_1}, ..., c_{t_n}$ of r_2, where (1) $1 \le s_1, ..., s_n \le i$, (2) $1 \le t_1, ..., t_n \le k$, and (3) $n \ge 1$

Then the direct mapping includes the following rule for r_1 and r_2 to generate [Reference Triples:](#)

```
Triple(S, P, O) ← r1(X1, ..., Xi), generateRowIRI("r1", ["a_p1", ..., "a_pj"], [X_p1, ..., X_pj], S),
                  r2(Y1, ..., Yk), card("r2", [Y1, ..., Yk], U), V ≤ U, generateRowBlankNode("r2", [Y1, ..., Yk], V, O),
               nonNull(X_s1), ..., nonNull(X_sn), X_s1 = Y_t1, ..., X_sn = Y_tn, generateReferencePropertyIRI("r1", ["a_s1", ...,
```

For example, assume that table **Addresses** in the [Direct Mapping Example](#) has columns **ID**, **city** and **state**, and that column **ID** is a candidate key (instead of a primary key), so that table **Addresses** does not have a primary key. Moreover, assume that table **People** in this example has columns **ID**, **fname** and **addr**, it has column **ID** as its primary key, and it has a foreign key in the column **addr** to the candidate key **ID** in the table **Addresses**. In this case, the following is the direct mapping rule to generate [Reference Triples](#):

```
Triple(S, P, O) ← People(X1, X2, X3), generateRowIRI("People", ["ID"], [X1], S),
                  Addresses(Y1, Y2, Y3), card("Addresses", [Y1, Y2, Y3], U), V ≤ U, generateRowBlankNode("Addresses", [Y1
                  nonNull(X3), X3 = Y1, generateReferencePropertyIRI("People", ["addr"], P)
```

B.3.3 Table r_1 does not have primary key and table r_2 has a primary key

Assume that:

- r_1 is a table with columns $a_1, ..., a_i$, and it does not have a primary key

- r_2 is a table with columns $c_1, ..., c_k$ and such that $[c_{q_1}, ..., c_{q_m}]$ is the primary key of r_2, where $1 \le m \le k$ and $1 \le q_1 < ... < q_m \le k$
- the foreign key indicates that the columns $a_{s_1}, ..., a_{s_n}$ of r_1 reference the columns $c_{t_1}, ..., c_{t_n}$ of r_2, where (1) $1 \le s_1, ..., s_n \le i$, (2) $1 \le t_1, ..., t_n \le k$, and (3) $n \ge 1$

Then the direct mapping includes the following rule for r_1 and r_2 to generate [Reference Triples:](#)

$$
\begin{aligned}
\mathrm{Triple}(S, P, O) \leftarrow\ & r_1(X_1, ..., X_i), \mathrm{card}(\text{"}r_1\text{"}, [X_1, ..., X_i], U), V \le U, \mathrm{generateRowBlankNode}(\text{"}r_1\text{"}, [X_1, ..., X_i], V, S),\\
& r_2(Y_1, ..., Y_k), \mathrm{generateRowIRI}(\text{"}r_2\text{"}, [\text{"}c_{q_1}\text{"}, ..., \text{"}c_{q_m}\text{"}], [Y_{q_1}, ..., Y_{q_m}], O),\\
& \mathrm{nonNull}(X_{s_1}), ..., \mathrm{nonNull}(X_{s_n}), X_{s_1} = Y_{t_1}, ..., X_{s_n} = Y_{t_n}, \mathrm{generateReferencePropertyIRI}(\text{"}r_1\text{"}, [\text{"}a_{s_1}\text{"},
\end{aligned}
$$

For example, table **People** in the [Direct Mapping Example](#) has columns **ID**, **fname** and **addr**, and it has column **ID** as its primary key, while table **Tweets** from Section [Empty (non-existent) primary keys](#) has columns **tweeter**, **when** and **text**, it does not have a primary key, and it has a foreign key in column **tweeter** that references column **ID** in table **People**. In this case, the following is the direct mapping rule to generate [Reference Triples](#):

$$
\begin{aligned}
\mathrm{Triple}(S, P, O) \leftarrow\ & \mathrm{Tweets}(X_1, X_2, X_3), \mathrm{card}(\text{"Tweets"}, [X_1, X_2, X_3], U), V \le U, \mathrm{generateRowBlankNode}(\text{"Tweets"}, [X_1, X_2, X_3],\\
& \mathrm{People}(Y_1, Y_2, Y_3), \mathrm{generateRowIRI}(\text{"People"}, [\text{"ID"}], [Y_1], O),\\
& \mathrm{nonNull}(X_1), X_1 = Y_1, \mathrm{generateReferencePropertyIRI}(\text{"Tweets"}, [\text{"tweeter"}], P)
\end{aligned}
$$

B.3.4 Table r_1 does not have primary key and table r_2 does not have a primary key

Assume that:

- r_1 is a table with columns $a_1, ..., a_i$, and it does not have a primary key
- r_2 is a table with columns $c_1, ..., c_k$, and it does not have a primary key
- the foreign key indicates that the columns $a_{s_1}, ..., a_{s_n}$ of r_1 reference the columns $c_{t_1}, ..., c_{t_n}$ of r_2, where (1) $1 \le s_1, ..., s_n \le i$, (2) $1 \le t_1, ..., t_n \le k$, and (3) $n \ge 1$

Then the direct mapping includes the following rule for r_1 and r_2 to generate [Reference Triples:](#)

$$
\begin{aligned}
\mathrm{Triple}(S, P, O) \leftarrow\ & r_1(X_1, ..., X_i), \mathrm{card}(\text{"}r_1\text{"}, [X_1, ..., X_i], U_1), V_1 \le U_1, \mathrm{generateRowBlankNode}(\text{"}r_1\text{"}, [X_1, ..., X_i], V_1, S\\
& r_2(Y_1, ..., Y_k), \mathrm{card}(\text{"}r_2\text{"}, [Y_1, ..., Y_k], U_2), V_2 \le U_2, \mathrm{generateRowBlankNode}(\text{"}r_2\text{"}, [Y_1, ..., Y_k], V_2,\\
& \mathrm{nonNull}(X_{s_1}), ..., \mathrm{nonNull}(X_{s_n}), X_{s_1} = Y_{t_1}, ..., X_{s_n} = Y_{t_n}, \mathrm{generateReferencePropertyIRI}(\text{"}r_1\text{"}, [\text{"}a_{s_1}\text{"},
\end{aligned}
$$

For example, assume that table **People** in the [Direct Mapping Example](#) has columns **ID**, **fname** and **addr**, and that column **ID** is a candidate key (instead of a primary key), so that **People** does not have a primary key. Moreover, assume that table **Tweets** from Section [Empty (non-existent) primary keys](#) has columns **tweeter**, **when** and **text**, it does not have a primary key, and it has a foreign in column **tweeter** that references candidate key **ID** in table **People**. In this case, the following is the direct mapping rule to generate [Reference Triples](#):

$$
\begin{aligned}
\mathrm{Triple}(S, P, O) \leftarrow\ & \mathrm{Tweets}(X_1, X_2, X_3), \mathrm{card}(\text{"Tweets"}, [X_1, X_2, X_3], U_1), V_1 \le U_1, \mathrm{generateRowBlankNode}(\text{"Tweets"}, [X_1, X_2, X\\
& \mathrm{People}(Y_1, Y_2, Y_3), \mathrm{card}(\text{"People"}, [Y_1, Y_2, Y_3], U_2), V_2 \le U_2, \mathrm{generateRowBlankNode}(\text{"People"}, [Y_1, Y_2,\\
& \mathrm{nonNull}(X_1), X_1 = Y_1, \mathrm{generateReferencePropertyIRI}(\text{"Tweets"}, [\text{"tweeter"}], P)
\end{aligned}
$$