# Loose joins: dropping referential integrity when constructing RDF from heterogeneous data sources

Els de Vleeschauwer[1], Sitt Min Oo[1], Ben De Meester[1] and Pieter Colpaert[1]

[1]IDLab, Dept. Electronics & Information Systems, Ghent University – imec, Belgium

**Abstract**

Existing knowledge graph construction mapping languages focus on mapping over heterogeneous data sources. Given these languages' legacy of mapping over relational databases, current mapping language join constructs conflate checking referential integrity with linking concepts across data sources. This leads to a significant amount of operations (resulting in performance bottlenecks), and loss of additional context of potential linking triples that are not generated due to a lack of referential integrity. We propose a loose join as an additional join type. In this short research paper, we describe the concept, its applicability, how it could be integrated in existing and future mapping languages, and a proof-of-concept implementation. Our evaluation based on GTFS-Madrid-Bench confirms the assumption that removing these integrity checks leads to much faster generation times, but we also find that using loose joins results in exactly the same graph output, i.e., these alternative semantics do not influence generation results for cases where referential integrity is assumed within the source system. Adding the loose join type keeps the best of all worlds: generation time is shortened where possible and you have more context in the resulting RDF graph, while the mapping file still supplies join construct metadata. For future work, we will further investigate the balance between referential integrity enforced by the data source and heterogeneous data sources on the open Web, and expand our implementation to more complex join conditions. This will allow us to investigate similar performance gains with other benchmarks and other mapping engines.

**Keywords**

RML, join, optimization

## 1. Introduction

Knowledge Graph construction over (semi-)structured heterogeneous data has been largely simplified by using established mapping languages and well-maintained mapping engines. The increasing maturity of this field is exemplified by the activities of W3C's Knowledge Graph Construction Community Group, and recent research detailing optimization strategies implemented in mapping engines.

Existing mapping languages to map over heterogeneous data sources historically extended W3C's recommended Relational to RDF Mapping Language (R2RML) [1]: the

Knowledge Graph Construction Workshop '23

els.devleeschauwer@ugent.be (E. de Vleeschauwer); x.sittminoo@ugent.be (Sitt Min Oo); ben.demeester@ugent.be (B. De Meester); pieter.colpaert@ugent.be (P. Colpaert)
https://ben.de-meester.org/#me (B. De Meester); https://pietercolpaert.be/#me (P. Colpaert)
0000-0002-8630-3947 (E. de Vleeschauwer); 0000-0001-9157-7507 (Sitt Min Oo); 0000-0003-0248-0987 (B. De Meester); 0000-0001-6917-2167 (P. Colpaert)

© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

RDF Mapping Language (RML) being the most direct example [2], but we can safely assume that other languages such as SPARQL-Generate [3] and ShExML [4] have taken inspiration from R2RML/RML given the timeline and references in the respective papers.

As R2RML aimed at mapping data from relational databases, a join in R2RML is represented as an operation that checks referential integrity, i.e., checking that a table's foreign key is equal to a primary key in another table. However, the assumption that we should check referential integrity by default when expressing relationship between data sources is kept in languages such as RML (i.e., requiring that each data source's identifier to another concept exists as an identifier in another data source). Hence, the concept of checking referential integrity (which is the responsibility of certain types of data sources) and linking concepts (which is the responsibility of mapping languages) got conflated.

This conflation has as effect that joins specified in mapping languages can only be interpreted as strict joins: operations checking referential integrity before a triple is made. This leads to a significant amount of operations, and typically results in performance bottlenecks and joining optimization strategies such as: use predicate-join index tables [5], partition triples maps with joins into disjoint sets to parallelize execution [6, 7], and eliminate self-joins [7].

Moreover, this leads to loss of additional context: potential triples that are not generated due to a lack of referential integrity across heterogeneous data sources.

In this short research paper we argue that the default link between referential integrity checks and the expression of relationships between data sources is too restrictive for the task at hand, namely, the generation of RDF graphs using heterogeneous data sources. Performing a referential integrity check is not always needed to integrate data from different sources, and may lead to loss of context.

We propose, next to the traditional (strict) join, an additional join type: a loose join, expressing the relationship between sources without imposing any referential integrity.

We provide a proof-of-concept implementation that replaces the join constructs which do not rely on external information by loose joins, and can be used a first step of the knowledge graph generation process by any RML engine.

We evaluate this proof-of-concept implementation and show the positive effects on the generation time when replacing strict joins by loose joins when this is possible. Much to our surprise, this change in semantics resulted in exactly the same output when executing GTFS-Madrid-Bench, showing the potential impact of this solution in real-world use cases.

After presenting the effects of conflating referential integrity checks with linking concepts during mapping (section 2), we describe the loose join semantics and implementation (section 3), evaluate (section 4), and conclude (section 5).

## 2. Problems with referential integrity conflation

Performing a referential integrity check is not always needed to integrate data from different sources, and may even lead to loss of context.

As running example (fig. 1) we use and extend the RML specification case illustrating

```csv
id,city,bus,latitude,longitude
6523,Brussels,25,50.901389,4.484444
2487,London,72,51.51334,-0.08901
6524,Charleroi,50.41136,4.44448
```

(a) Airport.csv

```json
{"venue": [
  {
    "latitude":"50.901389",
    "longitude":"4.484444",
    "location": {
      "continent":"EU","country":"BE",
      "city":"Brussels"}},
  {
    "latitude":"51.260197",
    "longitude":"4.402771",
    "location": {
      "continent":"EU","country":"BE",
      "city":"Antwerp"}},
  {
    "latitude":"51.51334",
    "longitude":"-0.08901",
    "location":{
      "continent":"EU","country":"GB",
      "city":"London"}}
]}
```

(b) Venue.json

```rdf
<#TriplesMap1> a rr:TriplesMap ;
  rml:logicalSource [
    rml:source "Airport.csv" ;
    rml:referenceFormulation ql:CSV ;
  ] ;
  rr:subjectMap [
    rr:template "http://trans.example.com/airport/{id}" ;
  ];
  rr:predicateObjectMap [
    rr:predicate ex:located ;
    rr:objectMap [
      rr:parentTriplesMap
        <#TriplesMap2> ;
      rr:joinCondition [
        rr:child "city" ;
        rr:parent "location.city" ;
      ];
    ];
  ].
<#TriplesMap2> a rr:TriplesMap;
  rml:logicalSource [
    rml:source "Venue.json"  ;
    rml:referenceFormulation ql:JSONPath  ;
    rml:iterator "$.venue[*]" ;
  ] ;
  rr:subjectMap [
    rr:template "http://venue.example.com/{location.city}" ;
  ] .
```

(c) mapping.rml.ttl

**Figure 1:** Running example

```rdf
<http://trans.example.com/airport/2487> ex:located <http://venue.example.com/London> .
<http://trans.example.com/airport/6523> ex:located <http://venue.example.com/Brussels> .
```

**Figure 2:** Output of running example

"Relationships among Logical Sources"[1]. The subject of a triples map (the parent map) is used as the object of a predicate-object map (the child map). If the logical sources of the parent map and the child map are not the identical, a join condition needs to be added, resulting in a joint query according to the R2RML specification[2].

Due to this joint query there will be only two triples (fig. 2) resulting from the source files and mapping files in the running example (fig. 1). Although the location to Airport 6524 is available in the `Airport.csv`, the triple expressing the location of this airport (fig. 3) will not be generated, because Charleroi does not appear in `Venue.json`.

[1]https://rml.io/specs/rml/#logical-join
[2]https://www.w3.org/TR/r2rml/#dfn-joint-sql-query

```
<http://trans.example.com/airport/6524> ex:located <http://venue.example.com/Charleroi> .
```

**Figure 3:** Missing triple, not available in output of running example

This example shows how we can lose knowledge when imposing referential integrity on incomplete datasets. This missing triple contains valuable information and can be used to solve queries which do not require additional venue information. The identifier `<http://venue.example.com/Charleroi>` can also serve as a link to other Knowledge Graphs available on the Web. As long as one relational database is used as a source for a mapping file, a good alignment and minimal loss of knowledge can be expected because the source system imposes referential integrity [8]. However, often the source data will not be perfectly aligned, especially when this data comes from unrelated source systems, increasing the amount of triples being lost during the join operation. On top of the risk of losing knowledge, checking the referential integrity in these cases comes with a large performance hit, and is one of the reasons why the referential integrity constraint is typically dropped in data warehouses [8].

## 3. Loose Joins

To avoid unwanted loss of knowledge and performance hit whilst keeping the semantic power and consistency of the mapping file, we propose to drop the notion of referential integrity by default in languages that map from heterogeneous data to RDF, and to introduce an a additional join type, next to the traditional (strict) join: a loose join. After describing the concept (section 3.1) and its limitation (section 3.2), we propose a language construct (section 3.3) and a proof-of-concept implementation (section 3.4).

### 3.1. Concept

A loose join expresses the relationship between sources without imposing any relational integrity: the object of the child map is generated following the structure of the subject of the parent map without verifying or using data from the parent source.

A loose join (fig. 4b) is equivalent to a carefully crafted URI (fig. 4a). The use of crafted URIs results however in following disadvantages: (i) risk for inconsistency: the alignment of the URIs becomes a manual process and is therefore error-prone, especially when dealing with bigger mapping files; and (ii) loss of semantics: the relationship between the triple maps can no longer be deducted from the mapping file (relevant metadata for, e.g., deducting the SHACL shape from an RML mapping file [9]). By specifying a loose join type, we can mitigate these disadvantages.

### 3.2. Limitations

Not every join can be re-interpreted as a loose join. We identify two cases for which the use of loose joins is not possible: (i) data from the parent source is needed to build the

```
<#TriplesMap1> a rr:TriplesMap ;
  ...
  rr:predicateObjectMap [
    rr:predicate ex:located ;
    rr:objectMap [
      rr:template
"http://venue.example.com/{city}";
      rr:termType rr:IRI ;
      ] ;
    ] ;
  ] .
```

(a) crafted URI

```
<#TriplesMap1> a rr:TriplesMap ;
  ...
  rr:predicateObjectMap [
    rr:predicate ex:located ;
    rr:objectMap [
      rr:parentTriplesMap
        <#TriplesMap2> ;
      rr:joinCondition [
        rr:child "city" ;
        rr:parent "location.city" ;
        rml:strictJoin false ;
      ];
    ];
  ].
  ...
```

(b) loose join

**Figure 4:** adaptations of mapping.rml.ttl

child object URI, or (ii) the parent source is a selective dataset.

Referring back to our running example, case (i) applies if the template of a venue would include the country code "`http://venue.example.com/{location.country}/{location.city}`". Case (ii) applies if not all cities in the `Airport.csv` are venues. If a city would be a venue only when it appears in the list of venues being `Venue.json`, the integrity check is needed to avoid the generation of wrong triples. In this case the triple of fig. 3 would be wrong.

### 3.3. Language Construct

To correctly handle cases (i) and (ii), strict joins remain needed. We propose to add an additional property to the join condition, specifying whether the join type is loose or strict(fig. 4b). To guarantee backwards compatibility, the join type could be strict by default in existing mapping languages, but we argue that it should be loose by default in new mapping languages, because this join type fits the Open World Assumption supported by RDF and encourages to use of strict joins only when really needed, as in cases (i) and (ii).

### 3.4. Proof-of-concept Implementation

Loose joins can be automatically converted to crafted URIs, which can be handled by default by any existing RML mapping engines. This allows introducing the alternative join type without adding any overhead to the existing RML mapping engines. We published a proof-of-concept implementation converting loose joins to crafted URIs at https://github.com/RMLio/rml-loose-generator.

At this moment our implementation interprets every join that does not need data from the parent source as a loose join (i.e., it does not take case (ii) as described in section 3.2 into account), and is limited to joins with only one join condition, without use of functions,

**Table 1**
Comparison of the knowledge graph generation time in seconds when using mapping files with strict joins versus mapping files with loose joins, average of three runs, time out after 1 hour.

| | | GTFS scale 1 | GTFS scale 5 | GTFS scale 50 | GTFS scale 100 |
|---|---|---|---|---|---|
| RMLMapper | strict joins | - | - | - | - |
| | loose joins | **6** | **18** | - | - |
| RMLStreamer | strict joins | - | - | - | - |
| | loose joins | 25 | 33 | **136** | **229** |
| Morph-KGC | strict joins | 13 | 20 | 163 | 746 |
| | loose joins | 12 | 22 | 155 | 653 |

and where the subject of the parent is built with a template. Outside of these restrictions, the original semantics of strict joins are preserved.

## 4. Evaluation

We tested the effect of loose joining on the widely used GTFS-Madrid-Bench [10]. We used our proof-of-concept implementation to re-interpret all relevant joins from the GTFS-Madrid-Bench mappings as loose joins, before sending the adapted mapping file to RMLMapper, RMLStreamer and Morph-KGC[3]. Although unanticipated, our implementation correctly detected every join construct within GTFS-Madrid-Bench as loose joins.

We tested the effect of loose vs strict joins on these mapping engines for scales 1, 5, 50, and 100 (table 1), using a device with following specifications: 2 x Hexacore Intel E5645 (2.4GHz) CPU, 24GB RAM, 1x 250GB harddisk. RMLMapper and RMLStreamer cannot generate any output for the GTFS-Madrid-Bench within one hour when using strict join. Using loose joins however, these mapping engine were able to generate correct output, with timings similar to using a state-of-the-art mapping engine like Morph-KGC. We note that RMLMapper can not handle GTFS scales 50 and 100, as it handles everything in memory.

When comparing the resulting knowledge graphs from GTFS-Madrid-Bench we found no difference at all between the version generated with strict joins and the one generated with loose joins. In hindsight, this makes sense: GTFS-Madrid-Bench's testing data is also used to fill a relational database, where referential integrity is needed. We conclude that data joined in the GTFS-Madrid-Bench has perfect referential integrity and that none of the joins in the GTFS-Madrid-Bench mapping file impose any limitation to chose the loose join type instead of the strict one.

When applying our implementation on the running example, with incomplete source files, we get a more complete output (fig. 2 and fig. 3).

With a very limited overhead, the loose joining algorithm can accomplish an instant

[3]https://github.com/RMLio/rmlmapper-java, https://doi.org/10.5281/zenodo.3887065, and https://doi.org/10.5281/zenodo.5543552, respectively.

improvement for any RML or R2RML mapping engine on both knowledge graph generation time. In cases where referential integrity cannot be assumed, e.g., when joining over data sources that are not under the same governance, loose joining can improve completeness of the resulting knowledge graph.

## 5. Conclusion

R2RML's assumption of referential integrity has put bias on heterogeneous mapping languages. Meanwhile, in cases where referential integrity is needed, the data sources typically validate this integrity by design (e.g., in the case of relational databases). As such, referential integrity checks can typically be resolved by moving the join from the mapping to the data source system. So either you integrate data sources that require referential integrity and these kinds of joins can be handled by the data sources themselves, or you integrate heterogeneous data sources that cannot require referential integrity and these kinds of joins can be handled loosely.

By adding an additional join type to the mapping languages you keep the best of all worlds: the knowledge graph generation performance is no longer unnecessarily impacted and you have more context in the resulting RDF graph (i.e., relations where the object contains no further information), while the mapping file still remains consistent and supplies the valuable metadata which can be deducted from join constructs.

For future work, we will research how to further optimize joins in cases where referential integrity can either be assumed due to the data source, or can be ignored given the open Web. First, our current implementation only takes equality operators into account, however, we can expand this to other types of join conditions (e.g., comparing the lowercased values between sources) by transforming the child reference. Second, we will verify if we can achieve similar performance gains with other benchmarks and other mapping engines.

Not all joins can be re-interpreted as loose joins (section 3.2). However, due to the completely correct generation within the GTFS-Madrid-Bench, we believe that many use cases can benefit from this additional join type, embracing the Open World Assumption followed by performance gain without any side-effects.

## Acknowledgments

The described research activities were supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10) and the imec ICON project AI4Foodlogistics (Agentschap Innoveren en Ondernemen project nr. HBC.2020.3097).

## References

[1] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, Working Group Recommendation, World Wide Web Consortium (W3C), 2012. URL: http://www.w3.org/TR/r2rml/.

[2] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: Proceedings of the 7th Workshop on Linked Data on the Web, volume 1184, 2014.

[3] M. Lefrançois, A. Zimmermann, N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, in: The Semantic Web 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 – June 1, 2017, Proceedings, Portoroz, Slovenia, 2017, pp. 35–50. doi:10.1007/978-3-319-58068-5_3.

[4] H. García-González, I. Boneva, S. Staworko, J. E. Labra-Gayo, J. M. C. Lovelle, ShExML: improving the usability of heterogeneous data mapping languages for first-time users, PeerJ Computer Science 6 (2020) e318.

[5] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana, M.-E. Vidal, SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs, in: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, 2020. doi:10.1145/3340531.3412881.

[6] E. Iglesias, S. Jozashoori, M.-E. Vidal, Scaling Up Knowledge Graph Creation to Large and Heterogeneous Data Sources, arXiv:2201.09694 [cs] (2022). ArXiv: 2201.09694.

[7] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M. S. Pérez, O. Corcho, Morph-KGC: Scalable knowledge graph materialization with mapping partitions, Semantic Web (2022) 1–20. doi:10.3233/sw-223135.

[8] W. McKnight, Data quality, in: Information Management, Elsevier, 2014, pp. 32–43. doi:10.1016/B978-0-12-408056-0.00004-7.

[9] T. Delva, B. D. Smedt, S. M. Oo, D. V. Assche, S. Lieber, A. Dimou, RML2shacl: RDF generation taking shape, in: Proceedings of the 11th on Knowledge Capture Conference, 2021, pp. 153–160. doi:10.1145/3460210.3493562.

[10] D. Chaves-Fraga, F. Priyatna, A. Cimmino, J. Toledo, E. Ruckhaus, O. Corcho, Gtfs-madrid-bench: A benchmark for virtual knowledge graph access in the transport domain, Journal of Web Semantics 65 (2020) 100596. doi:https://doi.org/10.1016/j.websem.2020.100596.