Relational to RDF Data Migration by Query Co-Evaluation - Extended Abstract

Ryan Wisnesky[1] and Daniel Filonik[2]

[1] Conexus AI, ryan@wisnesky.net
[2] Conexus AI, daniel@filonik.com

Abstract. In this paper we define a new algorithm to convert an input relational database to an output set of RDF triples. The algorithm can be used to e.g. load CSV data into a financial OWL ontology such as FIBO. The algorithm takes as input a set of relational conjunctive (select-from-where) queries, one for each input table; the source of each query is the three column (subject, predicate, object) RDF schema and the target of each query is the corresponding input table's relational schema. The algorithm's output is the only set of RDF triples for which a unique "round-trip" of the input data under the relational queries exists. The output may contain blank nodes, is unique up to unique isomorphism, and can be obtained using elementary formal methods (equational theorem proving and term model construction specifically). In the full version of this paper, we also describe how (generalized) homomorphisms between graphs can be used to write such relational conjunctive (select-from-where) queries, which, due to the lack of structure in the three-column RDF schema, tend to be large in practice. We demonstrate examples of both the algorithm and (in the full version of the paper) the mapping language on the FIBO financial ontology.

1 Introduction

Data migration, also known as data exchange [2] and closely associated with data integration [4], is a perennial topic in computer science, with a large body of existing results and techniques that can be immediately put to use in practical applications. The dominant approach to both data migration and integration (including [2] and [4]) uses techniques from formal logic and model theory [1] to specify declarative "schema mapping" formulae relating source and target relational databases. This approach is epitomized by ETL[3] tools such as Informatica, Altova MapForce, IBM DataStage, Ab Initio, and more, which translate the schema mappings into executable code (perhaps for a big-data engine such as Spark). In this approach, graph data models such as RDF are handled by encoding graphs as their edge relations and nested data models such as XML are handled either by shredding [10] or a higher-order logic [5]. Similarly, in this paper we are primarily concerned with RDF, but our technique applies any data model that can be expressed algebraically [9]. Note that because the three-column RDF schema is a relational schema, the results in this paper apply to RDF-to-RDF translation as well, although we do not explore RDF-to-RDF translation in this paper.

Although the burden of writing (or discovering) schema mappings is generally considered to be "low" in the literature, presumably due to their declarative character, we have found in data migration practice that schema mappings are not a substitute for queries when it come to communicating with programmers. We believe the reason for this is because query languages such as SQL are simply more widely used (and better understood) than mapping formalisms or ETL tools. As such in this paper we propose a new method to migrate relational data to an RDF database that avoids schema mapping altogether, instead relying on a set of programmer-provided relational queries which define how to project RDF data from the relational data; these queries are then evaluated (run; executed) "in reverse" using a new algorithm obtained by instantiating results from a branch of mathematics known as category theory [3]. Although the migrated RDF data will not always be perfectly recoverable by the provided queries, the

[3] See section 4 for a glossary of acronymns.

migrated data will always be "round-trippable" by the provided queries, allowing us to quantify the information change (gain or loss) involved in the relational to RDF translation. In the full version of this paper, available at `https://arxiv.org/abs/2403.01630`, we then generalize the relational to RDF migration algorithm to take into account a lightweight form of RDF schema information described in terms of (extended) graph homomorphisms, which allows us to restructure RDF databases along schema mappings in a lightweight way, as well as construct the relational queries our algorithm requires in a less verbose way.

In this paper we assume familiarity with RDF[8] and relational database theory [1]. In some sections, we will make use of terminology from category theory [3] to describe how a result is obtained, but a knowledge of category theory is not required to understand our results. See [9] for the relevant mathematics.

2 Relational to RDF Migration

Our algorithm migrates data from a relational database to an RDF database by evaluating a set of relational conjunctive (select-from-where) queries "in reverse", and then quantifying round-trip information change. In the next sub-sections we give an operational description of our algorithm and apply it to a simple example.

2.1 Input data preparation

For simplicity we begin by describing the algorithm when the source relational database schema S consists of exactly one table, which we for example take to be Person, with columns named name and age. In the full version of this paper, we describe how our method scales to multiple input tables. In this example, we assume that the input relation for Person contains two rows, which we will refer to as a and b. Our algorithm begins by taking the input table Person, adding a row column to it, and create new row IDs (so-called "surrogate keys") for each row; the row IDs themselves will be used later in our algorithm, and the actual values of the IDs do not matter/can be arbitrarily chosen. That is, in this example starting from the table on the left we start by creating the table on the right:

name	age
Alice	20
Bob	30

row	name	age
a	Alice	20
b	Bob	30

2.2 Query Specification

Because our input relational database consists of one table with the source schema Person(name, age), we must provide a single relational conjunctive (select/from/where) query that specifies how to populate Person from RDF triples. Assuming that the RDF database is as a three column table represented with target schema Rdf(subject,predicate,object), we may project out name and age columns as follows (where foaf is the "friend of a friend" RDF schema (`http://www.foaf-project.org/`):

```
CREATE VIEW Person AS
SELECT r1.object AS name, r2.object AS age
FROM Rdf AS r0, Rdf AS r1, Rdf AS r2 WHERE
r0.subject = r1.subject    AND r1.predicate = "foaf:name" AND
r0.subject = r2.subject    AND r2.predicate = "foaf:age"  AND
r0.predicate = "rdfs:type"  AND r0.object    = "foaf:person"
```

At this point, it is worth noting that the choice of query above is not unique. In fact, many choices of query are possible, and as we will see, the innate information change (gain or loss or neither) of each choice can be quantified by round-tripping.

2.3 Query Co-evaluation

At a mathematical level, the source relational schema S (containing a single table, Person in this example) and target relational schema T (containing a single table, Rdf in this example) form algebraic structures called *categories* [3] and the conjunctive relational query (queries in general) from T to S shown above in this example is a *pro-functor* [9], which can not only be evaluated to transform T-databases into S-databases according to the usual relational (e.g. SQL) semantics, but can also be "co-evaluated" to turn S-databases into T-databases in the "most data quality-preserving way possible". An exact definition of query co-evaluation in the most general case is described in [9] and implemented in the open-source CQL tool (`http://categoricaldata.net`); in this section, we describe its operation on RDF translation directly in elementary terms.

1. For every row p in Person and every Rdf AS v statement in the `FROM` clause:

   ```
   Rdf AS r0,  Rdf AS r1,  Rdf AS r2
   ```

   we consider the pair (p, v) to uniquely identify an output RDF row. For example, we have six output RDF tuples, which we refer to as:

   $$(a, r_0)\ ,\ (a, r_1)\ ,\ (a, r_2)\ ,\ (b, r_0)\ ,\ (b, r_1)\ ,\ (b, r_2)$$

   To determine the RDF subjects, predicates, and objects of these six output rows we will examine the `WHERE`, `FROM`, and `SELECT` clauses of the input queries in turn.
2. We first examine the `WHERE` clause:

   ```
   r0.subject = r1.subject AND r1.predicate = "foaf:name" AND
   r0.subject = r2.subject AND r2.predicate = "foaf:age"  AND
   r0.predicate = "rdfs:type" AND r0.object = "foaf:person"
   ```

   and apply it the output rows enumerated above, yielding equations:

   $$(a, r_0).\mathsf{subject} = (a, r_1).\mathsf{subject} \quad (a, r_1).\mathsf{predicate} = \mathsf{foaf:name} \quad (a, r_0).\mathsf{subject} = (a, r_2).\mathsf{subject}$$
   $$(a, r_2).\mathsf{predicate} = \mathsf{foaf:age} \quad (a, r_0).\mathsf{predicate} = \mathsf{rdfs:type} \quad (a, r_0).\mathsf{object} = \mathsf{foaf:person}$$
   $$(b, r_0).\mathsf{subject} = (b, r_1).\mathsf{subject} \quad (b, r_1).\mathsf{predicate} = \mathsf{foaf:name} \quad (b, r_0).\mathsf{subject} = (b, r_2).\mathsf{subject}$$
   $$(b, r_2).\mathsf{predicate} = \mathsf{foaf:age} \quad (b, r_0).\mathsf{predicate} = \mathsf{rdfs:type} \quad (b, r_0).\mathsf{object} = \mathsf{foaf:person}$$

   The above equations contain redundancy, but we have refrained from simplifying them to make it more apparent how they were computed.
3. To the equations above, we next add equations about the attributes of Person from the input data:

   $$a.\mathsf{name} = \mathsf{Alice} \quad b.\mathsf{name} = \mathsf{Bob} \quad a.\mathsf{age} = 20 \quad b.\mathsf{age} = 30$$

   and from `SELECT` clause of the query:

   ```
   r1.object AS name, r2.object AS age
   ```

   we add equations:

   $$(a, r_1).\mathsf{object} = a.\mathsf{name} \quad (b, r_1).\mathsf{object} = b.\mathsf{name} \quad (a, r_2).\mathsf{object} = a.\mathsf{age} \quad (b, r_2).\mathsf{object} = b.\mathsf{age}$$

   Note that e.g. a appears in both tuples (e.g. (a, r_1)) and alone (e.g. $a.\mathsf{name}$).
4. Finally, we determine the subjects, predicates, and objects for each output RDF row by examining the equations from the steps above and creating the (uniquely determined) minimal number of new fresh values / RDF blank nodes (indicated by ?s) required to complete the table below. We have, for example, that:

Output Row	subject	predicate	object
(a, r_0)	?1	rdfs:type	foaf:person
(b, r_0)	?2	rdfs:type	foaf:person
(a, r_1)	?1	foaf:name	Alice
(b, r_1)	?2	foaf:name	Bob
(a, r_2)	?1	foaf:age	20
(b, r_2)	?2	foaf:age	30

The above table can be constructed by repeatedly re-writing the equations from the above steps into a normal form, or constructing a decision procedure for the equations and then creating an initial term model, or using congruence closure algorithms, or many techniques besides. The above RDF tuples contain two *blank nodes* (terms of string type not equivalent to any string)[8], ?1 and ?2, representing Alice and Bob, respectively; the blank node ?1 might be represented as (a, r_0).subject, for example. The above RDF tuples can be exported directly as e.g. XML or Turtle (.ttl), or closed under RDFS or OWL axioms as desired.

2.4 Computational Complexity

Let r be the size of the input table and q the size of the query. The time complexity of the algorithm in this section is $O(rq\mathsf{log}(rq))$, because of the need to build a term model of a ground (variable-free) equational theory of size rq, such as for example with congruence closure algorithms. In contrast, the time complexity of query evaluation is $O(r^q)$. This analysis can be further refined to distinguish between the sizes of different parts of the query (number of joins, number of where conjuncts, etc). Note that closing the output of our algorithm under the full OWL axioms can make computational complexity semi-computable (due to the need to decide arbitrary first-order formulas).

2.5 Categorical Remarks

In this section we briefly describe the categorical theory of algebraic schemas and databases, following [9]. We first fix a theory, Ty, called the *type side* of our formalism. The sorts of Ty are called *types* and the functions of Ty are the functions that can appear in schemas and instances and queries and formulae. The intended meaning of this theory, $[\![Ty]\!]$, is a category with products. In this paper, our typeside has one type called Dom (domain), and the only functions are constant symbols corresponding to strings.

A *schema* S on type side Ty is a theory extending Ty with new sorts (called *entities*), new unary functions from entities to types (called *attributes*), new unary functions from entities to entities (called *foreign keys*), and new equations (called *data integrity constraints*) of the form $\forall v : s.\ t = t'$, where s is an entity and t, t' are terms of the same type, each containing a single free variable v. The intended meaning of this theory, $[\![S]\!]$, is a category extending $[\![Ty]\!]$. The theory associated to RDF adds an entity called Rdf, and three function symbols $\mathsf{Rdf} \to \mathsf{Dom}$ called $\mathsf{subject}$, $\mathsf{predicate}$, and object. The category associated to a relational schema with N tables adds N entities, one for each table; each column c of a table t adds an associated function symbol $c \to \mathsf{Dom}$.

An *instance* I on schema S is a theory extending S with new 0-ary function (constant) symbols called *generators* and non-quantified equations. The intended meaning of an instance I, written $[\![I]\!]$, is the *term model* (i.e., *initial algebra*) for I which contains, for each sort s, a *carrier set* consisting of the closed terms of sort s modulo provability in I. That is, the intended meaning of an S-instance I is a functor $[\![S]\!] \to \mathsf{Set}$, namely, the initial such functor in the category of all functors consistent with I.

Let I and J be instances on the same schema S. A data mapping $h : I \Rightarrow J$ is defined as a "derived signature morphism" [7] from I to J that is the identity on S. That is, $h : I \Rightarrow J$ assigns to each generator $g : s$ in I a closed term $h(g) : s$ in J in a way that respects equality: if $I \vdash t = t'$, then $J \vdash h(t) = h(t')$. A morphism of instances thus denotes a homomorphism (natural transformation) of algebras $[\![I]\!] \to [\![J]\!]$.

Let S and T be schemas on the same type side Ty. A schema mapping $F : S \to T$ is defined as a "derived signature morphism" [7] from S to T that is the identity on Ty. That is, $F : S \to T$ assigns to each entity $e \in S$ an entity $F(e) \in T$, and to each attribute / foreign key $f : s \to s'$ a term $F(f)$, of type $F(s')$ and with one free variable of type $F(s)$, in a way that respects equality: if $S \vdash t = t'$, then $T \vdash F(t) = F(t')$. The intended meaning of F is a functor $[\![S]\!] \to [\![T]\!]$ that is the identity on Ty.

The database instances and morphisms on a schema S constitute a category with all colimits, denoted $S{-}\mathsf{inst}$, and a schema mapping $F : S \to T$ induces a functor $\Sigma_F : S{-}\mathsf{inst} \to T{-}\mathsf{inst}$

defined by substitution. The functor Σ_F has a right adjoint, $\Delta_F : T{-}\mathsf{inst} \to S{-}\mathsf{inst}$, which corresponds to composition when we are thinking semantically: $[\![\Delta_F(I)]\!] = [\![F]\!]; [\![I]\!]$. Semantically, $[\![\Sigma_F(I)]\!]$ computes the "left Kan-extension" of $[\![I]\!]$ along $[\![F]\!]$ [9]. Δ_F has a right adjoint, $\Pi_F : S{-}\mathsf{Inst} \to T{-}\mathsf{Inst}$, which computes the right Kan-extension of $[\![I]\!]$ along $[\![F]\!]$ [9].

A pro-functor from category C to category D (written $C \Rightarrow D$) is defined as a functor $C^{op} \times D \to \mathsf{Set}$, or equivalently, by currying, $C^{op} \to \mathsf{Set}^D$. A *query* Q from schema S to schema T (on the same typeside) is defined as a presentation [6] of a pro-functor $[\![S]\!] \Rightarrow [\![T]\!]$ that maps each type to its representable instance. Q induces two adjoint data transformation operations: $\mathsf{coeval}_Q : T{-}\mathsf{Inst} \to S{-}\mathsf{Inst}$, corresponding to composition: $[\![\mathsf{coeval}_Q(I)]\!] = [\![I]\!]; [\![Q]\!]$ (where we implicitly convert between instances $X \to \mathsf{Set}$ and queries $1 \Rightarrow X$, where 1 is the schema with a single entity); and right adjoint $\mathsf{eval}_Q : S{-}\mathsf{Inst} \to T{-}\mathsf{Inst}$, with semantics corresponding to relational/SQL evaluation of Q.

In this paper we are given a select/from/where query q_s (on the RDF schema) for each table s in the input schema S, which corresponds to a query $S \Rightarrow \mathsf{Rdf}$ as follows. Each q_s is an Rdf-instance, and S has no generating morphisms between entities, so we have a functor $[\![S^{op}]\!] \to \mathsf{Set}^{[\![\mathsf{Rdf}]\!]}$; by currying, this corresponds to a functor $[\![S^{\mathsf{op}}]\!] \times [\![\mathsf{Rdf}]\!] \to \mathsf{Set}$.

Although coeval_Q has a direct (not necessarily 'simple') categorical description (pro-functor composition), and a direct (not necessarily 'simple') relational description (view unfolding), and it is even shown in [9] that for every Q, there are schema mappings F and G such that $\mathsf{coeval}_Q \cong \Sigma_F \circ \Delta_G$ (and dually for eval and Π), co-evaluation lacks a simple algorithmic description, suitable for a wide audience, with supporting tooling for special cases of real-world practical interest, such as loading CSV data into an RDF ontology, which motivated the results in this paper. It is our hope that anyone familiar with the basics of SQL can use the algorithm in this paper.

2.6 Round-tripping

The *round-trip* of our input data is computed by applying the user queries to the above triples, resulting in two rows on this example:

Round-trip	name	age
$r \mapsto (a, r_0), r_1 \mapsto (a, r_1), r_2 \mapsto (a, r_2)$	Alice	20
$r \mapsto (b, r_0), r_1 \mapsto (b, r_1), r_2 \mapsto (b, r_2)$	Bob	30

Let us refer to the input relational database as I; then it is a theorem [9] that there is a unique function assigning the row IDs of the above table to the row IDs of the original source data I, which in this case we may write as the table:

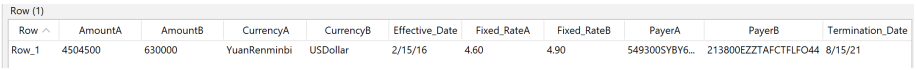
Row	Round-trip
a	$r \mapsto (a, r_0), r_1 \mapsto (a, r_1), r_2 \mapsto (a, r_2)$
b	$r \mapsto (b, r_0), r_1 \mapsto (b, r_1), r_2 \mapsto (b, r_2)$

The above function is *injective* because no target row is mapped to by more than one source ID, and *surjective*, because every target row is mapped to by at least one source ID. We thus say that the *unit* [9] of the relational to RDF data transformation defined in this section is *bijective* on the Person table, and we conclude there is no information loss or gain. When round-tripping we are examining the unit and/or co-unit of the co-evaluation evaluation adjunction.

2.7 FIBO Example

We conclude this section by describing a more elaborate example, of loading data into the FIBO ontology. That is, the set of RDF triples defined in this section can be written down in .xml or .ttl form and loaded into a tool such as Protege that has been pre-populated with the FIBO ontology. (We do note use any of the OWL axioms for FIBO; we are treating it entirely as RDF

in this paper.) We begin with a single row of input payer/payee data on a single table (shown in the CQL tool):



Row (1)

Row	AmountA	AmountB	CurrencyA	CurrencyB	Effective_Date	Fixed_RateA	Fixed_RateB	PayerA	PayerB	Termination_Date
Row_1	4504500	630000	YuanRenminbi	USDollar	2/15/16	4.60	4.90	549300SYBY6...	213800EZZTAFCTFLFO44	8/15/21

The relational conjunctive (select-from-where) query we use to construct such a tuple from FIBO is verbose, but formulaic; in the full version of this paper, we see how such queries can be abbreviated.

```
SELECT r5.object AS PayerA, t5.object AS PayerB,
       u3.object AS Effective_Date, v3.object AS Termination_Date,
       w3.object AS CurrencyA, s3.object AS AmountA,
       x3.object AS Fixed_RateA, c3.object AS CurrencyB,
       a3.object AS AmountB, b3.object AS Fixed_RateB
FROM R AS r1, R AS r2, R AS r3, R AS r4, R AS r5,
     R AS t1, R AS t2, R AS t3, R AS t4, R AS t5,
     R AS u2, R AS u3, R AS v2, R AS v3, R AS w2, R AS w3, R AS x2, R AS x3,
     R AS a2, R AS a3, R AS b2, R AS b3, R AS c2, R AS c3, R AS s2, R AS s3
WHERE r1.predicate = "...hasLeg" AND r1.object = r2.subject AND
r2.predicate = "...hasPayingParty" AND r2.object = r3.subject AND
r3.predicate = "...hasIdentity" AND r3.object = r4.subject AND
r4.predicate = "...isIdentifiedBy" AND r4.object = r5.subject AND
r5.predicate = "...hasTag" AND t1.subject = r1.subject AND
t1.predicate = "...hasLeg" AND t1.object = t2.subject AND
t2.predicate = "...hasPayingParty" AND t2.object = t3.subject AND
t3.predicate = "...hasIdentity" AND t3.object = t4.subject AND
t4.predicate = "...isIdentifiedBy" AND t4.object = t5.subject AND
t5.predicate = "...hasTag" AND
u2.subject = r2.subject AND u2.predicate = "...hasEffectiveDate"  AND
u2.object = u3.subject  AND u3.predicate = "...hasDateValue" AND
v2.subject = r2.subject AND v2.predicate = "...hasTerminationDate" AND
v2.object = v3.subject  AND v3.predicate = "...hasDateValue" AND
w2.subject = r2.subject AND w2.predicate = "...hasNotationalAmount" AND
w2.object = w3.subject  AND w3.predicate = "...hasAmount" AND
x2.subject = r2.subject AND x2.predicate = "...hasInterestRate" AND
x2.object = x3.subject  AND x3.predicate = "...hasRateValue" AND
a2.subject = t2.subject AND a2.predicate = "...hasNotationalAmount" AND
a2.object = a3.subject  AND a3.predicate = "...hasAmount" AND
b2.subject = t2.subject AND b2.predicate = "...hasInterestRate" AND
b2.object = b3.subject  AND b3.predicate = "...hasRateValue" AND
c2.subject = t2.subject AND c2.predicate = "...hasCurrency" AND
c2.object = c3.subject  AND c3.predicate = "...hasAmount" AND
s2.subject = r2.subject AND s2.predicate = "...hasCurrency" AND
s2.object = s3.subject  AND s3.predicate = "...hasAmount"
```

When this query is co-evaluated, it creates 24 RDF triples with 15 blank RDF nodes related as shown below in the CQL tool:

R (24)

Row	subject	predicate	object
0	?0	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasCurrency	USDollar
8	?0	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasPayingParty	?6
20	?0	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasNotationalAmount	?13
22	?0	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasInterestRate	?14
1	?1	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasCurrency	YuanRenminbi
4	?1	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasPayingParty	?3
12	?1	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasEffectiveDate	?9
14	?1	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasTerminationDate	?10
16	?1	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasNotationalAmount	?11
18	?1	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasInterestRate	?12
2	?2	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasLeg	?1
3	?2	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasLeg	?0
5	?3	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasIdentity	?4
6	?4	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/isIdentifiedBy	?5
7	?5	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasTag	549300SYBY6EJPNEBV76
9	?6	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasIdentity	?7
10	?7	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/isIdentifiedBy	?8
11	?8	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasTag	213800EZZTAFCTFLFO44
13	?9	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasDateValue	2/15/16
15	?10	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasDateValue	8/15/21
17	?11	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasAmount	4504500
19	?12	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasRateValue	4.60
21	?13	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasAmount	630000
23	?14	https://spec.edmcouncil.org/fibo/ontology/DER/DerivativesContracts/Swaps/hasRateValue	4.90

Every symbol, for example `USDollar`, that appears in the input row appears in the output. Note also that the blank nodes appear in multiple tuples, for example, subject ?0 has object ?13, which in turn is the subject of row 21.

To further analyze the relational to FIBO RDF migration, we round-trip the data to find, perhaps unexpectedly, that there are two round-tripped rows:

Row (2)

Row	AmountA	AmountB	CurrencyA	CurrencyB	Effective_Date	Fixed_RateA	Fixed_RateB	PayerA	PayerB	Termination_Date
0	4504500	4504500	YuanRenminbi	YuanRenminbi	2/15/16	4.60	4.60	549300SYBY6EJP...	549300SYBY6EJP...	8/15/21
1	4504500	630000	YuanRenminbi	USDollar	2/15/16	4.60	4.90	549300SYBY6EJP...	213800EZZTAFCT...	8/15/21

The round-trip assigns the single input row to the row with both US and Chinese currencies, but what are we to make of the additional row with two Chinese currencies, and why isn't there a row with two US currencies? These are questions we can answer mathematically by examining the relational to RDF query. In particular, although our query binds variables r_1 and t_1 for two "legs", and thus considers four possible leg-leg pairings, it only binds a single variable for an RDF triple with predicate hasEffectiveDate, thereby restricting to just to the pairs with r_1 as the left component (i.e., (t_1, r_1) is not considered because the query is asymmetric, at the level of syntax). Further analysis reveals that there is no way to avoid considering all pairs of legs when we round-trip because we do not have access to the CrossCurrencyInterestRateSwaps that describe which legs belong to the same swap. However, even if the two legs did originate from the same swap, because in FIBO each leg has only one owner, an input row with a different buyer and seller must create two separate legs. In other words, in FIBO, each leg has one owner, and in our input data, we have a two-legged swap; therefore, we should expect our initial row to be round-tripped into two rows. So our conclusion is that the round-trip reveals the loading process is working exactly as intended.

3 Conclusion and Future Work

We have described a new algorithm, based on elementary formal methods, to translate relational databases to RDF triples by "co-evaluating" a set of relational conjunctive (select-from-where) queries that specify how to "round-trip" the relational data in "the best way possible", and we have shown how to quantify the information change (loss or gain) of this algorithm by examining the induced round-trips on the input data. Because the queries required as input to this procedure can be verbose, in the full version of this paper we define a schema mapping language to abbreviate such queries and provide a graph-based representation that can be used to construct such queries through an entirely graphical user interface. Finally, we show how both the algorithm and (in the full paper) the schema mapping language can be applied to load cross-currency interest-rate swaps into the FIBO RDF/OWL ontology.

4 Glossary of Acronyms

- CSV: *comma separated values*. A file format using the ".csv" name suffix, it consists of a series of newline-delimited rows, each row of which consists of comma-delimited columns. The first line indicates the names of the columns.
- ETL: *extract transform load*. A methodology for expressing data transformation as a sequence that first extracts data from a multi-table source system, transforms it according to formalized rules in an external engine, and then loads it into a target multi-table system.
- SQL: *structured query language*. A language for expressing relational queries, invented in the late 1970s and now the dominant language for data manipulation, known for its SELECT-FROM-WHERE syntax that resembles English.
- RDF: *resource description framework*. A language for expressing (subject, predicate, object) triples over a controlled url-based vocabulary invented by the W3C (the standards body for the internet) in the 2000s, it can be stored in several different file formats, including XML and TTL.

- RDFS: *RDF schema*. A controlled vocabulary for expressing meta data (typing, etc) about RDF triples.
- OWL: *Web Ontology Language*. Extends RDFS to include various logics, allowing to state data integrity rules, for example that a predicate such as *after* has an inverse called *before*. Software inference engines derive all the facts that follow from a set of RDF triples according to an OWL ontology.
- FIBO: *Financial Industry Business Ontology*. An OWL ontology that describes the business operations of the financial industry. For example, it defines terms such as "credit" and "debit" to be opposites.
- TTL: *Turtle*. A language for writing RDF used in the examples above, it is much easier to read than XML.
- XML: *Extensible Markup Language*. Techincally readable by humans but intended primarily for computers, XML provides a way to encode many kinds of information using nested trees.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Arenas, M., Barcel, P., Libkin, L., Murlak, F.: Foundations of Data Exchange. Cambridge University Press, USA (2014)
3. Awodey, S.: Category Theory. Oxford University Press, Inc., USA, 2nd edn. (2010)
4. Doan, A., Halevy, A., Ives, Z.: Principles of Data Integration. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. (2012)
5. Fuxman, A., Hernandez, M.A., Ho, H., Miller, R.J., Papotti, P., Popa, L.: Nested mappings: Schema mapping reloaded. In: Proceedings of the 32nd International Conference on Very Large Data Bases. p. 67–78. VLDB '06, VLDB Endowment (2006)
6. Goren-Roig, G., Meyers, J., Minichiello, E.: Presenting profunctors. Electronic Proceedings in Theoretical Computer Science **429**, 88–114 (09 2025)
7. Mossakowski, T., Krumnack, U., Maibaum, T.: What is a derived signature morphism? In: Codescu, M., Diaconescu, R., Ţuţu, I. (eds.) Recent Trends in Algebraic Development Techniques. pp. 90–109. Springer (2015)
8. Sahoo, S., Halb, W., Hellmann, S., Idehen, K., Jr, T., Auer, S., Sequeda, J., Ezzat, A.: A survey of current approaches for mapping of relational databases to rdf. W3C (01 2009)
9. Schultz, P., Wisnesky, R.: Algebraic data integration. J. Funct. Program. **27**, e24 (2017)
10. Van den Bussche, J.: Simulation of the nested relational algebra by the flat relational algebra, with an application to the complexity of evaluating powerset algebra expressions. Theoretical Computer Science **254**(1), 363–377 (2001)