International Journal of Web Information Sy

# Publication and Maintenance of RDB2RDF Views Externally Materialized in Enterprise Knowledge Graphs

**SCHOLARONE™**
Manuscripts

# Publication and Maintenance of RDB2RDF Views Externally Materialized in Enterprise Knowledge Graphs

First Author[1][0000−1111−2222−3333], Second Author[2,3][1111−2222−3333−4444], and Third Author[3][2222−−3333−4444−5555]

[1] XXX University, department , country
[2] address city, country xxx@xxx
[3] xxx Institute, xxx University, city, country
{xxx,yyy}@xxx

**Abstract.** ***Purpose***. Enterprise knowledge graphs (EKG) in RDF consolidate and semantically integrate heterogeneous data sources into a comprehensive dataspace. However, to make an external relational data source accessible through an EKG, an RDF view of the underlying relational database, called an RDB2RDF view, must be created. The RDB2RDF view should be materialized in situations where live access to the data source is not possible, or the data source imposes restrictions on the type of query forms and number of results. In this case, a mechanism for maintaining the materialized view data up-to-date is also required. This article then addresses the problem of the efficient maintenance of externally materialized RDB2RDF views.

***Approach***. This article proposes a formal framework for the incremental maintenance of externally materialized RDB2RDF views, in which the server computes and publishes changesets, indicating the difference between two states of the view. The EKG system can then download the changesets and synchronize the externally materialized view. The changesets are computed based solely on the update and the source database state, and require no access of the content of the view.

***Findings***. The central result of the article shows that changesets computed according to the formal framework correctly maintain the externally materialized RDB2RDF view. The experiments indicate that the proposed strategy supports live synchronization of large RDB2RDF views and that the time taken to compute the changesets with the proposed approach was almost three orders of magnitude smaller than partial rematerialization, and three orders of magnitude smaller than full rematerialization.

***Originality***. The main idea that differentiates the proposed approach from previous work on incremental view maintenance is to explore the object-preserving property of typical RDB2RDF views, so that the solution can deal with views with duplicates. The algorithms for the incremental maintenance of relational views with duplicates published in the literature require querying the the materialized view data to precisely compute the changesets. By contrast, the approach proposed in this article requires no access to view data. This is important when the view is maintained externally, because accessing a remote data source may be too slow.

**Keywords**. RDF View Maintenance, RDF View, Enterprise Knowledge Graph, Linked Data, Relational Database

**Article Type**. Research Paper

## 1   Introduction

*Enterprise knowledge graphs* (EKG) semantically integrate heterogeneous data sources into a comprehensive dataspace (Pan et al., 2017). An EKG provides a unified data layer that is semantically connected to the data sources thereby providing applications with integrated access to the data sources. In this way, an EKG can support unplanned ad hoc queries and data exploration without requiring a time-consuming data pre-processing step.

A key element of an EKG is the *domain ontology*, which specifies the common vocabulary for integrating data exported by the data sources. The domain ontology acts as a semantic layer that combines and enriches the data stored in data sources. It represents how data are organised and their intended meaning. Therefore, users can query the ontology transparently, without having to deal with the data source schemes. This article concentrates on domain ontologies defined in RDF.

2      F. Author et al.

A data source may export an *RDF view*, which is defined by a set of mapping rules that maps concepts of the data source to concept of the RDF domain ontology. This article focuses on RDF views of relational databases, called *RDB2RDF views*.

A view may be virtual or materialized. In the *virtual approach*, view data are retrieved directly from the data source at query time. This is achieved by unfolding the view mappings, thus translating user queries into queries over the data sources. The advantage of virtual views is that data are always up-to-date with respect to the data sources. On the other hand, it may not be feasible to implement a virtual view, if *live access*, that is, runtime access, to the data source is in some way restricted. This approach, called VKG (virtual Knowledge graph), has been implemented in several systems (Kalaycı et al., 2020; Calvanese et al., 2020; Ding et al., 2021) and adopted in a wide range of use cases (Xiao et al., 2019).

In the *materialized approach*, view data are materialized and stored. An *externally materialized view* is a materialized view which is stored in a system different from that of the data source. Materialized views tend to achieve better query performance than virtual views. Also, they are the only alternative when live access to the data source is restricted. However, a materialized view requires some mechanisms to maintain its data when the underlying data source is updated. The main contribution of this article is an efficient maintenance algorithm for externally materialized RDB2RDF views.

Basically, there are two strategies for materialized view maintenance. *Rematerialization* recomputes view data at pre-established times, whereas *incremental maintenance* periodically modifies part of the view data to reflect updates to the data source. Previous research have shown that incremental maintenance generally outperforms full view rematerialization (Abiteboul et al., 1998; Ali et al., 2000; Ceri and Widom, 1991). Incremental maintenance also enables *live synchronization* of the view data with respect to the data source, that is, it enables maintaining view data up-to-date with only a small delay. This is an important property when the data source is frequently updated.

A strategy for materialized view maintenance is for a data source to compute and publish *changesets*. A *database changeset* indicates the difference between two states of the database, and a *view changeset* indicates the difference between two states of a view. From this point on, the term *changeset* will be used as a shorthand for *view changeset*. A materialized view maintenance algorithm can then download changesets and use them to update the materialized view data. For instance, DBpedia (DBp, Last accessed in Feb/2022) and LinkedGeoData (LG, Last accessed in Feb/2022) publish their changesets in a public folder. The computation of changesets can be challenging for externally materialized views when the database server has no access to the view data. Indeed, in the case of views with duplicates (Griffin and Libkin, 1995), the view maintenance algorithms published in the literature require the use of the materialized view data to compute the changeset.

This article proposes a novel strategy (see Figure 1) for the incremental maintenance of externally materialized RDB2RDF views. The strategy adopts triggers to compute and publish changesets of the relational database, and features a synchronization tool that downloads the changesets and synchronizes the externally materialized RDB2RDF view.
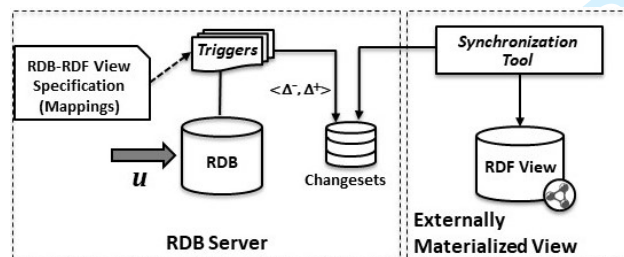


**Fig. 1.** Incremental Maintenance of Externally Materialized RDB2RDF View.

Four design goals guided the development of the proposed strategy:

1. Simplicity — minimizing the complexity associated with the creation of the infrastructure responsible for the construction of the changesets;
2. Efficiency of operation — identifying the minimal data that permits the construction of changesets that correctly maintain the view in the face of updates on the source database;

Publication and Maintenance of RDB2RDF Views Externally Materialized in Enterprise Knowledge Graphs     3

3. No access of the content of the view — computing the changeset based solely on the update and the source database state;

4. Self-maintenance of RDB2RDF views — computing the new view state based solely on the changeset and the view state.

The proposed strategy is based on three key ideas. First, the authors noted that RDB2RDF views typically have the so-called *object-preserving property*. That is, they preserve the base entities (objects) of the source database, rather than creating new entities (Motschnig-Pitrik, 2000). Therefore, an instance of the RDF view corresponds to a *pivot tuple* in the database, and both represent the same real-world object. The main ideas that differentiate the proposed approach from previous work on relational view maintenance is to explore the object-preserving property of typical RDB2RDF views, which allows the identification of the specific pivot tuples that are relevant to a data source update. Only the portion of the view associated with the relevant tuples should be re-materialized. One may then characterize the approach as that of "tracking the relevant tuples in the pivot relations for a given update" rather than "tracking the updated triples in the view for a given update", as adopted by view maintenance algorithms published in the literature.

Second, the authors introduce a formalism to specify object preserving-view mappings. The formalism is based on first-order logic, and has been widely adopted in Ontology-Based Data Access (OBDA) (Sequeda et al., 2014), data exchange (Murlak et al., 2014) and data integration (Lenzerini, 2002) systems. The formalism makes it easier to understand the semantics of the mapping, and provides sufficient information to support: (1) The identification of the tuples that are relevant to a data source update; (2) The automatic construction of the procedures that actually compute the changesets; (3) A rigorous proof of the correctness of the approach.

Third, the authors propose a formal framework for computing the correct changeset for an object preserving view. In the proposed framework, the content of an RDB2RD view is stored in an RDF dataset that contains a set of named graphs, used to describe the context in which the triples were produced. The main reason for separating triples into distinct named graphs is that duplicated triples, produced by tuples in different relations, will be in different named graph. A changeset is computed based solely on the source update and the source state before and after the update and, hence, no access to the materialized view is required. This is important when the view is externally maintained (Volz et al., 2005), because accessing a remote data source may be too slow. Indeed, the experiments indicate that the time taken to compute the changesets with the proposed framework was almost three orders of magnitude smaller than partial rematerialization, and three orders of magnitude smaller than full rematerialization.

The remainder of this article is organized as follows. Section 2 discusses related work. Section 3 presents the formalism used for specification of object-preserving RDB2RDF views. Section 4 introduces the case study that is used throughout the article. Section 5 formalizes the materialization of the data graph for an RDB2RDF view. Section 6 presents a formal framework for computing the correct changeset for an RDB2RDF view. Section 7 describes *LinkedBrainz Live*, a tool that implements the proposed strategy to propagate updates over the **MusicBrainz** database to **LinkedMusicBrainz** view. Section 8 presents the conclusions.

## 2 Related Work

This section separates related work into three groups. First, it reviews proposals that address the incremental view maintenance problem. Then, it overviews current techniques developed to manage knowledge graph evolution. Finally, it is covers Virtual Knowledge Graph (VKG) approaches.

*Incremental Maintenance Problem*: The Incremental View Maintenance problem has been extensively studied in the literature for relational views (Ceri and Widom, 1991; Griffin and Libkin, 1995), object-oriented views (Ali et al., 2000, 2003), semi-structured views (Liefke and Davidson, 2000; Zhao et al., 2017), and XML Views (Anonymous, 2008; Jin and Liao, 2010; Fegaras, 2011).

Most of the work in relational view maintenance proposes an algorithm that computes the changes for the materialized view when the base relations are updated. The work in (Ceri and Widom, 1991) is closest related to this. It shows that the use of triggers is effective for incremental maintenance because most of the work is done at view definition time. However, the method the authors propose does not support efficient maintenance of views with duplicates.

4      F. Author et al.

(Griffin and Libkin, 1995) study the problem of efficient maintenance of materialized views with duplicates. However, the proposed algorithms are not suitable for externally materialized views, because they require querying the content of the materialized view to precisely compute the changesets. For example, the view $W$ is defined using a query $Q$ that is defined through a bag algebra expression. Their proposal is to update $W$ according to changes that the transactions performed on the source data (called by those authors of *base tables*) of $W$. To do this, they identified change propagation rules to derive the incremental expressions that calculate the set of tuples to be added/deleted to/from $W$ from a given expression $Q$ and from insertions and deletions that a single transaction wants to perform. These rules are only suitable for the centralized database, because it requires that the source data and the views are in the same database. In addition, bag algebra expressions could not be used in the context of this article, which adopts the standard set semantics.

(Konstantinou et al., 2015) investigate the problem of incremental generation and storage of an RDF graph that is the result of exporting relational database contents. Their strategy, which is called here as *partial re-materialization*, requires annotating each triple with the mapping definition that generated it. In this case, when one of the source tuples changes (i.e., a table appears to be modified), the triples map definition will be executed for all tuples generated using the affected table and, thus, all triples generated using the affected tables are rematerialized. By contrast, in the approach proposed in this article, it is possible to identify which tuples in the affected tables are possibly affected by the update, and only those tuples are rematerialized.

To summarize, the approach proposed in this article differs from previous work on incremental view maintenance in that it explores the object-preserving property of typical RDB2RDF views, so that the solution can deal with views with duplicates. Furthermore, the proposed approach requires no access to view data, contrasting with the algorithms for the incremental maintenance of relational views with duplicates published in the literature, which require querying the materialized view data to compute the changesets.

*KG evolution*: Various approaches have been proposed to deal with the dynamic evolution of KG in different subjects, such as: (i) detect changes during their evolution (Tasnim et al., 2019; Arispe Riveros et al., 2020), (ii) represent change information (using vocabularies) (Singh, 2019), (iii) propagate changes to replicas or federated systems (Endris et al., 2015; Faisal et al., 2016), and (iv) detect change between LOD datasets (Papavasileiou et al., 2013; Roussakis et al., 2015; Zeginis et al., 2011).

Approaches such as described in (Tasnim et al., 2019; Arispe Riveros et al., 2020) focus on dealing with the problem of multiple versions of the same knowledge graph, keeping a summary out of different versions of the knowledge graph. (Singh, 2019) defines a set of terms for describing changes to resource descriptions. These proposals do not address the incremental maintenance data; thus, their focus is different from the approach proposed in this article.

(Endris et al., 2015) introduce an approach for interest-based RDF update propagation that consistently maintain a full or partial replication of large Linked Open Data (LOD) datasets. (Faisal et al., 2016) present an approach for dealing with co-evolution, which refers to the mutual propagation of the changes between a replica and its origin dataset. Both approaches rely on the assumption that either the source dataset provides a tool to compute a changeset in real-time or a third-party tool can be used for this purpose. Therefore, the contribution of this article is complementary and relevant to satisfy their assumption.

The works described in (Papavasileiou et al., 2013; Roussakis et al., 2015; Zeginis et al., 2011) address the problem of change detection between versions of LOD datasets. In (Zeginis et al., 2011), a low-level change detection approach is used to report simple insertion/deletion operations. In (Papavasileiou et al., 2013; Roussakis et al., 2015), a high-level change detection approach is used to provide deltas that are more readable to humans. Despite their contributions to understanding and analyzing the dynamics of Web datasets, these techniques cannot be applied to compute changesets for RDB2RDF views.

*Virtual Knowledge Graph (VKG) approaches*: Ontop (Xiao et al., 2020) is a canonical example of the VKG approach. Ontop does not materialize an RDF view of the relational database but maintains a virtual RDF view. During runtime, Ontop translates queries over the knowledge graph to SQL queries over the database. A survey of virtual graph systems can be found in (Xiao et al., 2019). As such, Ontop cannot be directly compared with the approach proposed in this article, which has a different motivation.

GraphDB (Ontotext, 2022) offers tools to migrate and materialize RDF data from relational databases, as well as to define virtual RDF views. GraphDB integrates with Ontop and extends it with multiple GraphDB specific features.

Contrasting with the VKG strategies, the approach proposed in this article was designed to cope with contexts where the RDF views must be externally materialized (hence the title of the article). Indeed, the Linked Open Data cloud has large RDF datasets that have been completely or partially replicated and integrated externally in other knowledge graphs. This calls for view maintenance strategies. This article argues, and the experiments reported confirm, that the proposed approach performs better than rematerialization and that it meets the requirement of "live synchronization", as already pointed out in the introduction.

## 3   Object Preserving RDB2RDF Views

### 3.1   *Basic Concepts and Notation*

As usual, a *relation scheme* is denoted as $R[A_1, \ldots, A_n]$. The *relational constraints* considered in this article consist of *mandatory* (or *not null*) *attributes, keys, primary keys* and *foreign keys*. In particular, $F(R{:}L, S{:}K)$ denotes a *foreign key*, named $F$, that *relate* $R$ and $S$, where $L$ and $K$ are lists of attributes from $R$ and $S$, respectively, with the same length.

A *relational schema* is a pair $\boldsymbol{S} = (\boldsymbol{R}, \Omega)$, where $\boldsymbol{R}$ is a set of relation schemes and $\Omega$ is a set of relational constraints such that: (i) $\Omega$ has a unique primary key for each relation scheme in $\boldsymbol{R}$; (ii) $\Omega$ has a mandatory attribute constraint for each attribute which is part of a key or primary key; (iii) if $\Omega$ has a foreign key of the form $F(R{:}L, S{:}K)$, then $\Omega$ also has a constraint indicating that $K$ is the primary key of $S$. The *vocabulary* of $\boldsymbol{S}$ is the set of relation names, attribute names, and foreign key names used in $\boldsymbol{S}$. Given a relation scheme $R[A_1, \ldots, A_n]$ and a *tuple variable* $t$ over $R$, $t.A_k$ denotes the *projection* of $t$ over $A_k$. *Selections* over relation schemes are defined as usual.

Let $\boldsymbol{S} = (\boldsymbol{R}, \Omega)$ be a relational schema and $R$ and $T$ be relation schemes of $\boldsymbol{S}$. A list $\phi = [F_1, \ldots, F_{n-1}]$ of foreign key names of $\boldsymbol{S}$ is a *path from $R$ to $T$* iff there is a list $R_1, \ldots, R_n$ of relation schemes of $\boldsymbol{S}$ such that $R_1 = R$, $R_n = T$ and $F_i$ *relates $R_i$ and $R_{i+1}$*. In this case, the tuples of $R$ *reference tuples of $T$ through* $\phi$. A state $\sigma$ of a relational schema $\boldsymbol{S}$ assigns to each relation scheme $R$ of $\boldsymbol{S}$ a relation $R(\sigma)$, in the usual way.

An *ontology vocabulary*, or simply a *vocabulary*, is a set of *class names, object property names* and *datatype property names*. An *ontology* is a pair $\boldsymbol{O} = (V, \Sigma)$ such that $V$ is a vocabulary and $\Sigma$ is a finite set of formulae in $V$, the *constraints* of $\boldsymbol{O}$. The constraints include the definition of the *domain* and *range* of a property, as well as *cardinality constraints*, defined in the usual way.

### 3.2   *Specification of Object Preserving RDB2RDF View*

This section presents the formalism used for the specification of object-preserving RDB2RDF views. By restricting to this class of views, it is possible to precisely identify the specific tuples that are relevant to a data source update w.r.t. an RDB2RDF view.

Let $\boldsymbol{O} = (V, \Sigma)$ be a target ontology, that is, the organization's ontology, and let $\boldsymbol{S} = (\boldsymbol{R}, \Omega)$ be a relational schema, with vocabulary $U$. Let $\boldsymbol{X}$ be a set of *scalar variables* and $\boldsymbol{T}$ be a set of tuple variables, disjoint from each-other and from $V$ and $U$.

The formal definition of an RDB2RDF view is similar to that given in (Sequeda et al., 2014; Poggi et al., 2008). An *RDB2RDF view* is a triple $\mathcal{W} = (V, \boldsymbol{S}, M)$, where:

1. $V$ is the vocabulary of the *target ontology*;
2. $\boldsymbol{S}$ is the *source relational schema*; and
3. $M$ is a set of *mappings* between $V$ and $\boldsymbol{S}$, defined by transformation rules.

Intuitively, a view satisfies the object-preserving property iff it preserves the base entities (objects) of the source database, rather than creating new entities from the existing ones (Motschnig-Pitrik, 2000). More precisely, a view $\mathcal{W} = (V, \boldsymbol{S}, M)$ satisfies the *object-preserving property* iff:

- the instances of the classes in $V$ correspond to tuples in selected relations of $\boldsymbol{S}$, which are called the *pivot relations* of $\mathcal{W}$;

6       F. Author et al.

– the values of datatype properties in $V$ of these instances are given by (functions of) attributes in the corresponding tuples, or in related tuples;
– the object properties in $V$ correspond to relationships between tuples in the pivot relations of the source database.

A *transformation rule* of $\mathcal{W}$ is an expression of the form $C(x) \leftarrow Q(x)$ or of the form $P(x,y) \leftarrow Q(x,y)$, where $C$ and $P$ are class and property names in $V$, and $Q(x)$ and $Q(x,y)$ are queries over $\boldsymbol{S}$ whose target clauses contain one and two variables, respectively.

The formalism based on DATALOG (Abiteboul et al., 1995) for the specification of the queries $Q(x)$ and $Q(x,y)$ was adopted, which appear on the RHS of the transformation rules. This formalism is much simpler than general query languages, such as SQL, and RDB2RDF mapping languages, such as R2RML (Hert et al., 2011), but it is expressive enough to specify object preserving views, the class of views that is the focus on in this article.

**Table I.** Examples of built-in predicates

Queries $Q(x)$ and $Q(x, y)$ are expressed as a list of literals. A literal can be: (1) a *range expression* of the form $R(r)$, where $R$ is a relation name in $U$ and $r$ is a tuple variable in $\boldsymbol{T}$; (2) a *built-in predicate* or *function*, such as those in Table I. The inclusion of built-in predicates and functions allows the formalism to capture specific notions of concrete domains, such as "string concatenation" and "less than", required for the specification of complex mappings and restrictions.

**Table II.** Transformation Rules

In this article, three specific types of transformation rules were adopted, which are defined in Table II: *Class Transformation Rule* (CTR), *Datatype Property Transformation Rule* (DTR), and *Object Property Transformation Rule* (OTR).

Intuitively, a CTR $\psi$ maps tuples of $R$ into instances of class $C$ in $V$. The predicate $B[r, x]$ establishes a *semantic equivalence relation* between a tuple $r$ in $R$, called the *pivot tuple*, and an instance $x$ of $C$ (i.e., intuitively, $r$ and $x$ represent the same real-world entity).

Since the present proposal is only interested in object preserving views, the predicate $B[r,x]$ should define a partial one-to-one function: (a) each pivot tuple $r$ in $R$ should correspond to at most one instance $x$ of $C$; (b) different pivot tuples $r_1$, $r_2$ should correspond to different instances $x_1$, $x_2$ of $C$. Then, $r$ and $x$ are said to be *semantically equivalent*, denoted $r \equiv x$, w.r.t the CTR $\psi$.

Intuitively, a DTR $\psi$ defines the values of the datatype property $P$ for the instances of class $C$. These values may correspond to attributes of the pivot tuple $r$, or attributes of tuples related to $r$, as specified by $H$ (see Table II). Here, there is no restriction on the predicate $H$, which may associate several values $y$ to the same tuple $r$.

To interpret an OTR $\psi$, remember that the right-hand side of the CTRs $\psi_D$ and $\psi_G$ define instances of classes $D$ and $G$, respectively. So, the OTR $\psi$ maps relations between tuples of $R_D$ and $R_G$ to instances of the object property $P$, as specified by $H$. Again, there is no restriction on the predicate $H$.

The approach proposed in this article efficiently computes changesets by exploring the object-preserving property, which allows the precise identification of the tuples in the pivot relations whose corresponding instances may have been affected by an update. The advantages of this approach are: (1) it simplifies the specification of the view, as the formalism is specifically designed to describe the correspondences that define an object-preserving view; (2) the restrictions on the structure of the queries help ensure the consistency of the specification of the view; (3) the formal expressions that define the queries can be explored to automatically construct the procedures that compute the changesets that maintain the RDB2RDF view; (4) it facilitates the task of providing rigorous proofs for the correctness of the proposed approach.

The problem of generating transformation rules is addressed in (Anonymous, 2013, 2014) and is outside of the scope of this work. However, Table III summarizes a set of Transformation Rule (TR) patterns that lead to the definition of relational to RDF mappings that guarantee that the RDF views satisfies the object preserving property by construction. Table I shows the definitions of the concrete predicates used by the TR Patterns in Table III. The TR Patterns support most types of data restructuring that are commonly found when transforming relational data to RDF, and they suffice to capture all R2RML mapping patterns proposed in the literature (Sequeda et al., 2012; Das et al., 2012). In (Anonymous, 2014), the authors proposed an approach to automatically generate R2RML mappings, based on a set of TRs Patterns. The approach uses relational views as a middle layer, which facilitates the R2RML generation process and improves the maintainability and consistency of the mapping.

**Table III.** Transformation Rule Patterns

## 4 Case Study: *MusicBrainz_RDF*

*MusicBrainz* (Mus, Last accessed in Feb/2022) is an open music encyclopedia that collects music metadata. The **MusicBrainz** relational database is built on the PostgreSQL relational database engine and contains all MusicBrainz music metadata. This data includes information about artists, release groups, releases, recordings, works, and labels, as well as the many relationships between them. Figure 2(a) depicts a fragment of the **MusicBrainz** relational database schema (for more information about the original scheme, see (MBz, Last accessed in Feb/2022)). Each relation has a distinct primary key, whose name ends with "id", except for *gid*, which is a Universally Unique IDentifier (UUID) for use in permanent links and external applications. The relations *Artist, Medium, Release, Recording* and *Track*, in Figure 2(a) represent the main concepts. The relation *ArtistCredit* represents an N:M relationship between *Artist* and *Credit*. The labels of the arcs, such as *fk1*, are the names of the foreign keys.

The case study uses an RDB2RDF view, called **MusicBrainz_RDF**, which is defined over the relational schema in Figure 2(a). Figure 2(b) depicts a fragment of the ontology used for publishing the **MusicBrainz_RDF** view. It reuses terms from three well-known vocabularies, *FOAF* (Friend of a Friend), *MO* (Music Ontology) and *DC* (Dublin Core).

Table IV shows a set of transformation rules that partially specify the mapping between the relational schema in Figure 2(a) and the ontology in Figure 2(b), obtained with the help of the tool described in (Anonymous, 2014).

For the examples in the following sections, consider the database state shown in Figure 3. The transformations rules $\psi_1$ and $\psi_2$, in Table IV, are examples of the CTR Pattern. The predicate *hasURI* is defined in Table I. The CTR $\psi_1$ indicates that, for each tuple $r$ in *Artist*, one should:

1. Compute the URI $s$ such that $hasURI(mbz:, r.gid, s)$=true.
2. Produce triple ($s$ rdf:type mo:MusicArtist). Therefore, $r$ and $s$ are semantically equivalent.

**Table IV.** Examples of Transformation Rules

The CTR $\psi_2$ indicates that, for each tuple $r$ in *Artist*, where $r$.type = 1, one should:

1. Compute the URI $s$ such that $hasURI(mbz:,r.gid, s)$=true.
2. Produce triple ($s$ rdf:type mo:SoloMusicArtist). Therefore, $r$ and $s$ are semantically equivalent.

Note that attribute *gid* is a key for relation *Artist*. Therefore, different tuples in *Artist* generate different URIs, i.e., different instances. Also, note that $\psi_1$ and $\psi_2$ use the same predicate *hasURI*. Therefore, if a tuple $r$ is mapped to triples ($x$ rdf:type mo:MusicArtist) and ($y$ rdf:type mo:SoloMusicArtist), then $x=y$.

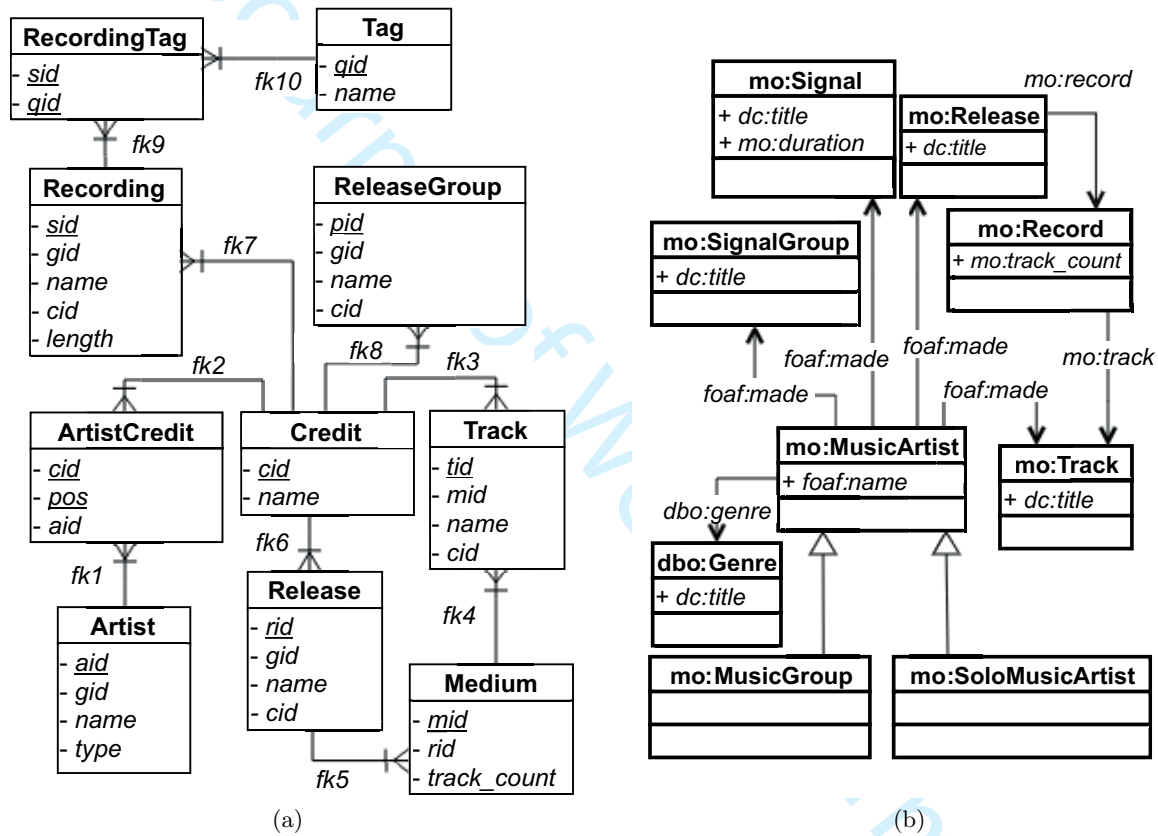**Fig. 2.** (a) Fragment of *MusicBrainz* Schema and (b) Fragment of *MusicBrainz_RDF* View Ontology.

**Artist**

| aid | gid | name | type |
|-----|-----|------|------|
| a1 | ga1 | Kungs | 1 |
| a2 | ga2 | Cookin' on 3 B. | 2 |
| a3 | ga3 | Kylie Auldist | 1 |

**ArtistCredit**

| cid | pos | aid |
|-----|-----|-----|
| c1 | 1 | a1 |
| c1 | 2 | a2 |
| c2 | 1 | a2 |
| c2 | 2 | a3 |
| c3 | 1 | a1 |

**Credit**

| cid | name |
|-----|------|
| c1 | Kungs vs. Cookin' on 3 B. |
| c2 | Cookin' on 3 B. feat. Kylie Auldist |
| c3 | Kungs |

**Track**

| tid | mid | name | cid |
|-----|-----|------|-----|
| t1 | m1 | This Girl | c2 |
| t2 | m1 | Don't You Know | c3 |

**Release**

| rid | gid | name | cid |
|-----|-----|------|-----|
| r1 | gr1 | Layers | c3 |

**RecordingTag**

| sid | qid |
|-----|-----|
| s1 | q1 |
| s1 | q2 |
| s2 | q2 |

**Tag**

| qid | name |
|-----|------|
| q1 | pop |
| q2 | dance |

**Medium**

| mid | rid | track_count |
|-----|-----|-------------|
| m1 | r1 | 12 |

**Recording**

| sid | gid | name | cid | length |
|-----|-----|------|-----|--------|
| s1 | gs1 | This Girl | c1 | 3:15 |
| s2 | gs2 | Don't You Know | c3 | 3:04 |

**ReleaseGroup**

| pid | gid | name | cid |
|-----|-----|------|-----|
| p1 | gp1 | Layers | c3 |

**Fig. 3.** State Example for Relations in Figure 2(a).

Considering the database state in Figure 3, CTRs $\psi_1$ and $\psi_2$ produce the following triples:

($mbz{:}r.ga1$   $rdf{:}type$   $mo{:}MusicArtist$);

($mbz{:}r.ga1$   $rdf{:}type$   $mo{:}SoloMusicArtist$);

($mbz{:}r.ga2$   $rdf{:}type$   $mo{:}MusicArtist$);

($mbz{:}r.ga3$   $rdf{:}type$   $mo{:}MusicArtist$); and

($mbz{:}r.ga3$   $rdf{:}type$   $mo{:}SoloMusicArtist$).

The transformation rule $\psi_{10}$, in Table IV, is an example of the DTR Pattern. The predicates $nonNull(v)$ and $RDFLiteral(u, A, R, v)$ are defined in Table I. Rule $\psi_{10}$ matches the value of attribute *name* of relation *Artist* with the value of datatype property *foaf:name*, whose domain is *mo:MusicArtist*. It indicates that, for each tuple $r$ of $R$, one should:

1. Compute the URI $s$ for the instance of *mo:MusicArtist* that $r$ represents, using the CTR $\psi_1$. Therefore, $s$ and $r$ are semantically equivalent.
2. For each value $v$, where $v$ is the literal representation of $r.name$ and $r.name$ is not NULL, produce triple ($s$   *foaf:name*   $v$).

Considering the database state in Figure 3, $\psi_{10}$ produces the following triples:

($mbz{:}ga1$   *foaf:name*   "Kungs");

($mbz{:}ga2$   *foaf:name*   "Cookin's on 3 B."); and

($mbz{:}ga3$   *foaf:name*   "Kylie Auldist").

The transformations rules $\psi_7$ and $\psi_8$, in Table IV, are examples of the OTR Pattern. The predicate $F(r, s)$, where $F$ is a foreign key of the form $F(R{:}L, S{:}K)$, is defined in Table I. For example, rule $\psi_7$ matches a relationship between a tuple $r$ of *Artist* and a tuple $r_3$ in *Track* to instances of the object property *foaf:made*, whose domain is *mo:MusicArtist* and range is *mo:Track*. It indicates that, for each tuple $r$ of $R$, one should:

1. Compute the URI $s$ for the instance of *mo:MusicArtist* that $r$ represents, using the CTR $\psi_1$. Therefore, $s$ and $r$ are semantically equivalent.
2. For each tuple $r_3$ of *Track* such that $r$ is related to $r_3$ through path $[fk_1, fk_2, fk_3]$, compute the URI $g$ for the instance of *mo:Track* that $r_3$ represents (using CTR $\psi_5$). Therefore, $r_3$ and $g$ are semantically equivalent.
3. Produce triple ($s$   *foaf:made*   $g$)

Considering the database state in Figure 3, $\psi_7$ produces the following triples:

10      F. Author et al.

($mbz{:}ga1$    $foaf{:}made$    $mbz{:}t2$);
($mbz{:}ga2$    $foaf{:}made$    $mbz{:}t1$); and
($mbz{:}ga3$    $foaf{:}made$    $mbz{:}t1$).

## 5  Materialization of the Data Graph for an RDB2RDF view

The materialization of the data graph for an RDB2RDF view requires translating source data into the RDB2RDF view vocabulary as specified by the mappings. An important technical issue that arises in this process is the possibility of *duplicated triples*, i.e., triples which are generated more than once due to different assignments to the variables in the body of one or more transformation rules. Indeed, the main difficulty for the incremental maintenance of views with duplicates is for delete and update operations. Recall that, by definition, the same triple cannot be present twice in an RDF triple store. Thus, if a tuple is removed, it is not possible to determine whether the corresponding triples should be deleted from the view, because triples may still be produced by another tuple in the database.

For a proper handling of the issue of duplicates, it becomes necessary to distinguish between two types of duplication:

1. Duplicated triples generated from different pivot relations (see Table II for definition of pivot relation);
2. Duplicated triples generated from the same pivot relation.

For the case of duplicated triples generated from different pivot relations, a solution based on named graphs is adopted. In the proposed framework, the content of an RDF2RDB view is stored in an RDF dataset that contains a collection of named graphs. As in (Carroll et al., 2005), a *named graph* is defined as a pair comprising a URI and an RDF graph. A named graph can be considered as a set of quadruples (or "quads") having the subject, predicate, and object of the triples as the first three components, and the graph URI as the fourth element. Each quadruple is interpreted similarly to a triple in RDF, except that the predicate denotes a ternary relation, instead of a binary relation. This way of representing quadruples, called *quad-statements*, was incorporated in the specification of *N-Quad* (nQu, 2014).

The main reason for separating triples into distinct (named) graphs is that duplicated triples, produced by tuples in different relations, will be in different named graphs (context). This is an important property for supporting duplicated triples generated by different tuples (Theorem 1 in Section 6.4).

For the case of duplicated triples generated from the same pivot relation, the proposed solution requires detecting the pivot tuples that are possibly affected by an update, and then to re-materializing all triples produced by the affected tuples. In particular, the proposal may be characterized as that of "tracking the relevant tuples in the pivot relations for a given update" rather than "tracking the updated triples in the view for a given update".

The following definitions formalize the materialization of the view $\mathcal{W} = (V, \boldsymbol{S}, M)$ as the result of applying the Transformation Rules (TRs) in $M$ against a state $\sigma$ of database $\boldsymbol{S}$.

For the examples in this section, consider the RDB2RDF view **MusicBrainz_RDF** defined in Section 4, and the database state shown in Figure 3. Also, consider that mbz:ga, mbz:gm, mbz:gr, and mbz:gt are the named graph URIs for the pivot relations *Artist, Medium, Recording,* and *Track*, respectively.

In the rest of the article, $R(\sigma)$ denotes the relation associated with the relation scheme $R$ in the database state $\sigma$.

**Definition 1.** *Let $\Psi$ be a TR in M and $r_1, \ldots, r_n$ be tuple variables appearing in $\Psi$ associated with relations $R_1, \ldots, R_n$, where $n \geq 2$ and $r_i \neq r_j$ for $i \neq j$. Also, let $\sigma$ be a database state and let $p_1, \ldots, p_n$ be tuples in $R_1(\sigma), \ldots, R_n(\sigma)$.*

1. *$\Psi[r_1/p_1, \ldots, r_n/p_n]$ denotes the TR obtained from $\Psi$ by substituting the tuples $p_1, \ldots, p_n$ for the tuple variables $r_1, \ldots, r_n$, respectively.*
2. *$\Psi[r_1/p_1, \ldots, r_n/p_n](\sigma)$ denotes the set of triples which are produced when $\Psi[r_1/p_1, \ldots, r_n/p_n]$ is applied to $\sigma$.*

Publication and Maintenance of RDB2RDF Views Externally Materialized in Enterprise Knowledge Graphs     11

**Definition 2 (RDF state of a tuple).** *Let $\sigma$ be a database state and let $R$ be a pivot relation in M. The RDF state of tuple p in $R(\sigma)$, denoted $M[p](\sigma)$, is defined as:*
$M[p](\sigma) = \{(s, q, o, g) \mid (s, q, o)$ is a triple in $\Psi[r/p](\sigma)$ where $\Psi$ is a TR in M with pivot relation R, and g is the named graph URI for pivot relation $R\}$.

Note that, if $p$ is a tuple in $R(\sigma)$ such that $R$ is not a pivot relation in any TR in $M$, then $M[p](\sigma) = \emptyset$.

*Example 1.* Consider the transformation rules for the **MusicBrainz_RDF** view defined in Table IV. Also, consider the relation *Artist* in Figure 2(a), which is the pivot relation of the TRs $\Psi_1$, $\Psi_2$, $\Psi_3$, $\Psi_7$, $\Psi_{10}$, and $\Psi_{24}$. Thus, the RDF state of a tuple $p$ in relation *Artist* is computed by applying the TRs $\Psi_1[r/p]$, $\Psi_2[r/p]$, $\Psi_3[r/p]$, $\Psi_7[r/p]$, $\Psi_{10}[r/p]$, and $\Psi_{24}[r/p]$. Considering the database state in Figure 3, the RDF state of tuple *a1* in *Artist* contains the following quads:

– (mbz:ga1 *rdf:type mo:MusicArtist* mbz:ga), by $\Psi_1[r/a1]$
– (mbz:ga1 *rdf:type mo:SoloMusicArtist* mbz:ga), by $\Psi_2[r/a1]$
– (mbz:ga1 *foaf:made* mbz:t2 mbz:ga), by $\Psi_7[r/a1]$
– (mbz:ga1 *foaf:name* "Kungs" mbz:ga), by $\Psi_{10}[r/a1]$
– (mbz:ga1 *dbo:genre* mbz:q1 mbz:ga);
  (mbz:ga1 *dbo:genre* mbz:q2 mbz:ga);
  (mbz:ga1 *dbo:genre* mbz:q2 mbz:ga), by $\Psi_{24}[r/a1]$

Note that, $\Psi_3[r/a1]$ produces no triple, while $\Psi_{24}[r/a1]$ produces duplicated triples. The triple (mbz:ga1 *dbo:genre* mbz:gq2) is generated twice by assigning different tuple variables in $\Psi_{24}$.

**Definition 3 (RDF state of a relation).** *Let $\sigma$ be a database state of **S** and $R$ be a relation in **S**. The RDF state of R at state $\sigma$, denoted $M[R](\sigma)$, is defined as:*

$$M[R](\sigma) = \bigcup_{p \text{ is a tuple in } R(\sigma)} M[p](\sigma)$$

Intuitively, the RDF state of $R$ at state $\sigma$ is computed by the materialization of the RDF state of all tuples in $R$. Note that, if $R$ is not a pivot relation in any TR in $M$, then $M[R](\sigma) = \emptyset$.

**Definition 4 (Materialization of $\mathcal{W}$).** *The materialization or state of $\mathcal{W}$ at state $\sigma$, denoted $M(\sigma)$, is defined as:*

$$M(\sigma) = \bigcup_{\substack{p \text{ is a tuple in } R(\sigma) \text{ and} \\ R \text{ is a Pivot Relation of } \mathcal{W}}} M[p](\sigma)$$

Intuitively, the RDF state of $\mathcal{W}$ at state $\sigma$ is computed by the materialization of RDF states of all pivot relation of $\mathcal{W}$.

# 6   Formal Framework for Computing Correct Changesets for RDB2RDF Views

This Section presents the proposed framework for computing a correct changeset for the materialized RDB2RDF view $\mathcal{W}= (V, \textbf{S}, M)$, when an update u occurs in the source relational database $\textbf{S}$.

## 6.1   *Overview*

An update $u$ on a relation $R$ is defined as two sets, $D$ and $I$, of tuples of $R$. The update $u$ indicates that the tuples in $D$ must be deleted and the tuples in $I$ must be inserted into $R$. More precisely:

**Definition 5 (updates, insertions, and deletions).** *An update on a relation R is a pair $u=(D,I)$ such that D and I are, possibly empty, sets of tuples of R. If $I = \emptyset$, the authors say that the update is a deletion, and, if $D=\emptyset$, they say that the update is an insertion.*

The semantics of an update $u = (D, I)$ is straightforward and is defined as follows:

**Definition 6 (semantics of an update).** *Let $u = (D, I)$ be an update on $R$ and $\sigma_0$ be a database state. Then, the execution of $u$ on $\sigma_0$ results in the database state $\sigma_1$ which is equal to $\sigma_0$ except that $R(\sigma_1) = R(\sigma_0) - D \cup I$. The authors also say that $\sigma_1$ is the result of executing $u$ on $\sigma_0$ and that $\sigma_0$ is the database state before $u$ and $\sigma_1$ is the database state after $u$.*

Note that updates are deterministic, in the sense that, given an initial state, an update always results in the same state.

The diagram in Figure 4 describes the problem of computing a correct changeset for a materialized RDB2RDF view $\mathcal{W}$, when an update $u$ occurs in the source relational database $\boldsymbol{S}$. In the diagram of Figure 4, assume that:

$$M(\sigma_0) \xrightarrow{\langle \Delta^-(u), \Delta^+(u) \rangle} (M(\sigma_0) - \Delta^-(u)) \cup \Delta^+(u)$$
$$\begin{array}{ccc} M \uparrow & \uparrow & \begin{array}{c} =? \\ M(\sigma_1) \\ M \uparrow \end{array} \\ \sigma_0 & \xrightarrow{\quad \boldsymbol{u} \quad} & \sigma_1 \end{array}$$
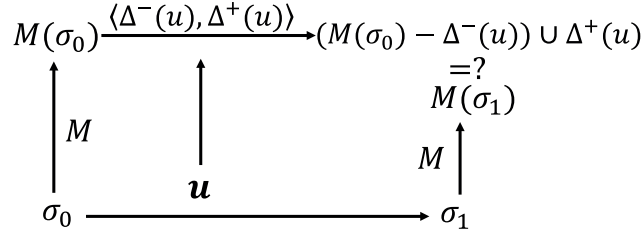
**Fig. 4.** The problem of computing correct changeset for RDB2RDF View.

1. $M$ is the set of mappings that materialize view $\mathcal{W}$;
2. $\sigma_0$ and $\sigma_1$ are the states of $\boldsymbol{S}$ respectively before and after the update $u$; and
3. $M(\sigma_0)$ and $M(\sigma_1)$ are the materializations of $\mathcal{W}$ respectively at $\sigma_0$ and $\sigma_1$.

A *correct changeset* for $\mathcal{W}$ w.r.t. $u$, $\sigma_0$ and $\sigma_1$ is a pair $\langle \Delta^-(u), \Delta^+(u) \rangle$, where $\Delta^-(u)$ is a set of triples removed from $\mathcal{W}$ and $\Delta^+(u)$ is a set of triples added to $\mathcal{W}$, that satisfies the following restriction (see Figure 4):

$$M(\sigma_1) = (M(\sigma_0) - \Delta^-(u)) \cup \Delta^+(u) \tag{1}$$

Putting this in words, the changeset $\langle \Delta^-(u), \Delta^+(u) \rangle$ is correctly computed iff the new view state $(M(\sigma_0) - \Delta^-(u)) \cup \Delta^+(u)$, computed with the help of the changeset, and the new view state $M(\sigma_1)$, obtained by the rematerialization of the view, using the view mappings, are identical.

The computation of the changesets depends on the update $u$, the initial and final database states, $\sigma_0$ and $\sigma_1$, and the view mappings $M$. However, the notation for changesets indicates only the update $u$, to avoid a cumbersome notation, since the database states may be considered as the context for $u$, and the view mappings as fixed for the database in question.

The proposed approach to compute a correct changeset for an update $u$ follows three main steps:

1. **Identification of *Relevant Relations*.** Identify the relations in $\boldsymbol{S}$ that are relevant to update $u$. A relation is relevant to $u$ if its RDF state is possibly affected by $u$.
2. **Identification of *Relevant Tuples*.** Identify the tuples, in the relevant relations, that are relevant to the update. A tuple is relevant to an update $u$ if its RDF state is possibly affected by the update.
3. **Computation of Changesets.** Compute the changeset $\langle \Delta^-(u), \Delta^+(u) \rangle$. $\Delta^-(u)$ contains the old RDF states of the relevant tuples, which are removed from $\mathcal{W}$, and $\Delta^+(u)$ contains the new RDF states of the relevant tuples, which are inserted into $\mathcal{W}$. Therefore, only the RDF state of relevant tuples, identified in Step 2. are rematerialized.

### 6.2   *Identifying Relevant Relations*

Definition 7 formally specifies, based on the TRs in $M$, the necessary and sufficient conditions for a relation to be considered relevant to an update.

**Definition 7.** *Let R be a relation in* **S**.

(i) *Let $\Psi$ be a TR in M. R is relevant to $\Psi$ iff R appears in the body of $\Psi$ or R is referenced by a foreign key in the body of $\Psi$.*

(ii) *R is relevant to view $\mathcal{W}$ iff R is relevant to some TR in M.*

(iii) *Let u be an update on relation R. A relation $R^*$ in* **S** *is relevant to u iff $R^*$ is the pivot relation of a TR $\Psi$ in M and R is relevant to $\Psi$.*

*Example 2.* Consider the transformation rules for the **MusicBrainz_RDF** view defined in Table IV. The relation *Track* is relevant to TRs $\Psi_5$, $\Psi_7$ and $\Psi_8$ (Definition 7(i)). The relations *Track, Artist* and *Medium* are the pivot relations of $\Psi_5$, $\Psi_7$ and $\Psi_8$, respectively. Therefore, the relations *Track, Artist* and *Medium* are relevant to updates on the relation *Track* (Definition 7(iii)).

In the rest of this section, $R^*$ denotes a pivot relation and $r^*$ denotes a pivot tuple variable. Proofs for all lemmas and theorems have been omitted here due to space limitations. For the interested reader, they can be found at https://doi.org/10.5281/zenodo.5850244.

**Lemma 1.** *Let u be an update on a relation R and let $\sigma_0$ and $\sigma_1$ be the database states before and after u, respectively.*

(i) *If R is not relevant to $\mathcal{W}$, then $M(\sigma_0) = M(\sigma_1)$. Thus, an update on a relation that is not relevant to the view does not affect the state of the view.*

(ii) *Let $R^*$ be a relation in* **S**. *If $R^*$ is not relevant to u, then $M[R^*](\sigma_0) = M[R^*](\sigma_1)$. Thus, an update u does not affect the RDF state of the relations which are not relevant to u.*

Based on Lemma 1(i), the authors focused their attention only on the relations that are relevant to $\mathcal{W}$ (Definition 7 (ii)). On the other hand, from Lemma 1(ii), an update on a relevant relation may affect only the RDF states of the relations that are relevant to the update. Consider, for example, an update on relation *Track*. This update may affect the RDF state of relations *Track, Artist* and *Medium*, which are relevant to updates on *Track* (see Example 2). The RDF states of other relevant relations are not affected by updates on *Track*.

### 6.3 Identifying Relevant Tuples

Definitions 8 and 9 formally define sufficient conditions to identify, based on the database state and the view mappings, which tuples, in a relevant relation, are relevant to an update. The key idea of the proposed approach is to rematerialize only the RDF state of the tuples that are relevant to the update, i.e., the tuples whose RDF state might possibly be affected by the update.

**Definition 8.** *Let:*

– *$u=(D, I)$ be an update on R.*

– *$\sigma_0$ and $\sigma_1$ be the database states before and after u, respectively.*

– *$\Psi$ be a TR in M, where R is relevant to $\Psi$, $R^*$ is the pivot relation of $\Psi$, r and $r^*$ are the tuple variables for R and $R^*$ in $\Psi$, respectively.*

(i) *Let $r_{old}$ be a tuple in D.*
   *$\mathcal{P}[R,\Psi](r_{old}) = \{p \mid p$ is a tuple in $R^*(\sigma_0)$ and $(\Psi[r^*/p, r/t])$ $(\sigma_0) \neq \emptyset) \}$.*

(ii) *Let $r_{new}$ be a tuple in I.*
   *$\mathcal{P}[R,\Psi](r_{new}) = \{p \mid p$ is a tuple in $R^*(\sigma_1)$ and $(\Psi[r^*/p,r/t](\sigma_1) \neq \emptyset) \}$.*

(iii) *A tuple p in $R^*(\sigma_0) \cup R^*(\sigma_1)$ is relevant to u w.r.t. $\Psi$ iff p is in $\mathcal{P}[R,\Psi](t)$ for some t in $D \cup I$.*

In Definition 8, given a tuple t in $D \cup I$, $\mathcal{P}[R,\Psi](t)$ returns all tuples in $R^*(\sigma_0) \cup R^*(\sigma_1)$ that are related to t. Therefore, the RDF state of a tuple p in $\mathcal{P}[R,\Psi](t)$ may be affected by the update u. This makes p relevant to u.

**Definition 9.** *Let $u=(D, I)$ be an update on R and $\sigma_0$ and $\sigma_1$ be the database states before and after u, respectively. A tuple p in $\sigma_0 \cup \sigma_1$ is relevant to u iff:*

14      F. Author et al.

*(i) p is relevant to u w.r.t a TR $\Psi$ in M, or*
*(ii) p occurs in $D \cup I$ and R is a pivot relation of a TR in M.*

**Lemma 2.** *Let $u=(D, I)$ be an update on R. Let $R^*$ be a relation relevant to u and let p be a tuple in $R^*(\sigma_0)$, but not in D. If p is not relevant to u w.r.t any TR $\Psi$ in M, then $M[p](\sigma_0) = M[p](\sigma_1)$.*

*Example 3.* Consider the transformation rules for the **MusicBrainz_RDF** view defined in Table IV, and the database state in Figure 3. Also, consider $u$ an update which deletes tuple $r_{old}$ and inserts tuple $r_{new}$ in table *Track*, where:

 – $r_{old} = \langle$t1, m1, "This Girl", c2$\rangle$
 – $r_{new}=\langle$t1, m1, "This Girl (feat. Cookin' On 3 B.)", c1$\rangle$

From TRs $\Psi_7$ , $\Psi_8$, you have that the relations *Track, Artist* and *Medium* are relevant to updates on table *Track* (See Example 2).
From Definition 8 and TR $\Psi_7$, you have that:

 – $\mathcal{P}[Track, \Psi_7](r_{new})= \{a1, a2\}$
 – $\mathcal{P}[Track, \Psi_7](r_{old})= \{a2, a3\}$

Therefore, tuples *a1* and *a2* in relation *Artist* are related to $r_{new}$ w.r.t. $\Psi_7$, and tuples *a2* and *a3* in relation *Artist* are related to $r_{old}$ w.r.t. $\Psi_7$. Thus, tuples *a1, a2* and *a3* are relevant to update $u$ w.r.t. $\Psi_7$.
From Definition 8 and TR $\Psi_8$, you have that:

 – $\mathcal{P}[Track, \Psi_8](r_{new})= \{m1\}$
 – $\mathcal{P}[Track, \Psi_8](r_{old})= \{m1\}$

Therefore, tuple *m1* in relation *Medium* is relevant to update $u$ w.r.t $\Psi_8$. From Definition 9(i), tuples *a1, a2, a3* and *m1* are relevant to $u$. Since table *Track* is a pivot relation, from Definition 9(ii), $r_{new}$ and $r_{old}$ are also relevant to $u$.

### 6.4   *Computing Changesets*

Algorithm 1 shows a high-level description of the algorithm for computing changeset for updates on a relation $R$. In the algorithm, Step 3 is processed in two phases. Phase 1 uses the database state before the update, while Phase 2 uses the database state after the update. The algorithms for insertions and deletions are similarly defined and are omitted here.

In the proposed strategy, database triggers are responsible for computing and publishing the correct changeset for the RDB2RDF view to stay synchronized with the relational database. The proposed strategy first identifies the relations in the source database that are relevant for the RDB2RDF view, that is, the relations whose updates might possibly affect the state of the RDB2RDF view.

For each update operation $u$ on a relevant relation (see Definition 7) two triggers are defined:

 – **BEFORE Trigger**: fired immediately before the update to compute the set $\Delta^-(u)$, using the view mapping and the database state BEFORE the update.

 – **AFTER Trigger**: fired immediately after the update to compute the set $\Delta^+(u)$, using the view mapping and the database state AFTER the update.

Figure 5 shows the templates of the triggers for the update operations on a relation $R$. Note that procedures $COMPUTE\_\Delta^-[R]$ and $COMPUTE\_\Delta^+[R]$ can be generated at view definition time, based on the TRs of the view, as discussed in a companion article.

In the following, the precise definition of the algorithm's key concepts is presented, which allowed the authors to provide rigorous arguments for the correctness of the algorithm. Based on Lemma 2, only the RDF state of the tuples that are relevant to the update should be rematerialized. This result motivates the following definition for the changesets that maintain the state of the view in the presence of an update $u$ on a tuple $p$.

Publication and Maintenance of RDB2RDF Views Externally Materialized in Enterprise Knowledge Graphs     15

---

**ALGORITHM 1:** Algorithm for computing changeset for updates on a relation $R$

---

**Input:**

  $u = (D, I)$ − an update on $R$;

  $\sigma_0$ and $\sigma_1$ − the states of the database respectively before and after the update $u$;

**Output:**

  $\Delta^-$ and $\Delta^+$

**2**

**3**  **if** $R$ *is relevant to the view* $\mathcal{W}$ *(Definition 7; (ii))* **then**

**4**

**5**    **Phase 1:** Before the update do:

**6**      1.1 Compute $\mathcal{P}_0$, the set of tuples in $\sigma_0$ that are relevant to $u$ (Definition 9)

**7**      1.2 Compute $\Delta^- := \bigcup\limits_{p \in (\mathcal{P}_0)} M[p]$;

**8**      `//` $\Delta^-$ `contains the union of the RDF states of tuples in` $\mathcal{P}_0$ `(Definition 10)`

**9**

**10**    **Phase 2:** After the update do:

**11**      2.1 Compute $\mathcal{P}_1$, the set of tuples in $\sigma_1$ that are relevant to $u$ (Definition 9)

**12**      2.2 Compute $\Delta^+ := \bigcup\limits_{p \in (\mathcal{P}_1)} M[p]$;

**13**      `//` $\Delta^+$ `contains the union of the RDF states of tuples in` $\mathcal{P}_1$ `(Definition 10)`

**14**  **return** $(\Delta^+, \Delta^-)$;

---

**BEFORE**{update, insert, delete} ON $R$
**BEGIN**
  $\Delta^- := \mathbf{\textit{COMPUTE\_}}\Delta^-[R](D, I)$,where
      $D$ is the set of deleted tuples and
      $I$ is the set of inserted tuples;
  ADD $\Delta^-$ to changeset file of $\mathcal{W}$
**END**

(a)

**AFTER**{update, insert, delete} ON $R$
**BEGIN**
  $\Delta^+ := \mathbf{\textit{COMPUTE\_}}\Delta^+[R](D, I)$,where
      $D$ is the set of deleted tuples and
      $I$ is the set of inserted tuples;
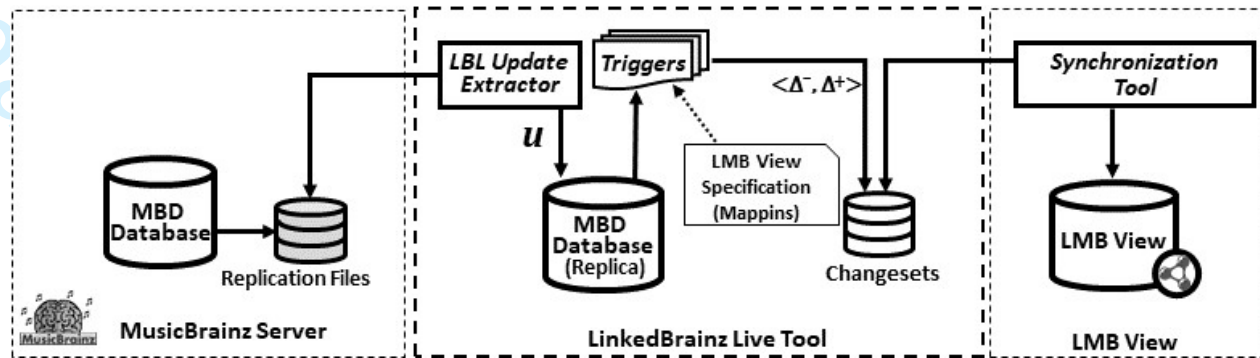  ADD $\Delta^+$ to changeset file of $\mathcal{W}$
**END**

(b)

**Fig. 5.** Triggers templates for updates on $R$.

16　　F. Author et al.

**Artist**

| aid | gid | name | type |
|-----|-----|------|------|
| a1 | ga1 | Kungs | 1 |
| a2 | ga2 | Cookin' on 3 B. | 2 |
| a3 | ga3 | Kylie Auldist | 1 |

**ArtistCredit**

| cid | pos | aid |
|-----|-----|-----|
| c1 | 1 | a1 |
| c1 | 2 | a2 |
| c2 | 1 | a2 |
| c2 | 2 | a3 |
| c3 | 1 | a1 |

**Credit**

| cid | name |
|-----|------|
| c1 | Kungs vs. Cookin' on 3 B. |
| c2 | Cookin' on 3 B. feat. Kylie Auldist |
| c3 | Kungs |

**Track**

| tid | mid | name | cid |
|-----|-----|------|-----|
| t1 | m1 | This Girl(feat. Cookin' On 3 B.) | c1 |
| t2 | m1 | Don't You Know | c3 |

**Release**

| rid | gid | name | cid |
|-----|-----|------|-----|
| r1 | gr1 | Layers | c3 |

**Recording**

| sid | gid | name | cid | length |
|-----|-----|------|-----|--------|
| s1 | gs1 | This Girl | c1 | 3:15 |
| s2 | gs2 | Don't You Know | c3 | 3:04 |

**RecordingTag**

| sid | qid |
|-----|-----|
| s1 | q1 |
| s1 | q2 |
| s2 | q2 |

**Tag**

| qid | name |
|-----|------|
| q1 | pop |
| q2 | dance |

**Medium**

| mid | rid | track_count |
|-----|-----|-------------|
| m1 | r1 | 12 |

**ReleaseGroup**

| pid | gid | name | cid |
|-----|-----|------|-----|
| p1 | gp1 | Layers | c3 |

**Fig. 6.** Database State after the update $u$.

**Definition 10 (Changeset for update $u$).** *Let $u=(D,I)$ be an update on $R$. Let $\mathcal{P}_0$ be the set of tuples in $\sigma_0$ that are relevant to $u$ (Definition 9). Let $\mathcal{P}_1$ be the set of tuples in $\sigma_1$ that are relevant to $u$ (Definition 9). Then,*

$$- \;\; \Delta^-(u) = \bigcup_{p \,\in\, \mathcal{P}_0} M[p](\sigma_0)$$
$$- \;\; \Delta^+(u) = \bigcup_{p \,\in\, \mathcal{P}_1} M[p](\sigma_1)$$

As already pointed out in the introduction, changesets depend on the update $u$, the initial and final database states, $\sigma_0$ and $\sigma_1$, and the view mappings $M$. However, the notation for changesets indicate only the update $u$, to avoid a cumbersome expression, since one can consider the database states as the context for $u$, and the view mappings as fixed for the database and the view in question.

In Definition 10, the set $\Delta^-(u)$ contains the old RDF state of the tuples in $\mathcal{P}_0$, and the set $\Delta^+(u)$ contains the new RDF state of the tuples in $\mathcal{P}_1$. In the following, Theorem 1 shows that the new state of the view is correctly computed using $\Delta^-(u)$ and $\Delta^+(u)$. So, $\langle \Delta^-(u), \Delta^+(u) \rangle$ is a correct changeset for $\mathcal{W}$ w.r.t. update $u$.

A third auxiliary lemma is needed to prove the central result of the article, which is shown as follows.

**Lemma 3.** *Let $\sigma$ be database state and, for $i=1,2$, let $\boldsymbol{T}_i$ be sets of tuples in pivot relations of an RDB2RDF view. Define $Q_i = \{x \; / \; x \text{ is a quad in } M[p](\sigma), \text{ where } p \text{ is a tuple in } \boldsymbol{T}_i\}$. If $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$ are disjoint, then $Q_1$ and $Q_2$ are also disjoint.*

**Theorem 1.** *Let:*

- *$u=(D,I)$ be an update on $R$*
- *$\sigma_0$ and $\sigma_1$ be the database states before and after $u$*
- *$\mathcal{P}_0$, $\mathcal{P}_1$, $\Delta^-(u)$ and $\Delta^+(u)$ be as in Definition 10*

*Then, $M(\sigma_1) = (M(\sigma_0) - \Delta^-(u)) \cup \Delta^+(u)$*

*Example 4.* To illustrate this strategy, consider the update $u$ as in Example 2, and $\mathcal{P}_0$ and $\mathcal{P}_1$ is in Definition 10. Figure 6 shows the new state of database $\boldsymbol{S}$ after the update $u$. From Example 3, you have:

$$\mathcal{P}_0 = \{a1, a2, a3, m1, r_{\text{old}}\} \text{ and } \mathcal{P}_1 = \{a1, a2, a3, m1, r_{\text{new}}\} \tag{2}$$

Figure 7 shows the set $\Delta^-(u)$, which contains the old RDF states of the tuples in $\mathcal{P}_0$. Figure 8 shows the set $\Delta^+(u)$, which contains the new RDF states of the tuples in $\mathcal{P}_1$.

Publication and Maintenance of RDB2RDF Views Externally Materialized in Enterprise Knowledge Graphs          17

```
{(mbz:t1    rdf:type     mo:track      mbz:gt);
 (mbz:t1    dc:title     "This Girl"    mbz:gt);
 (mbz:ga2   rdf:type     mo:MusicArtist    mbz:ga);
 (mbz:ga2   foaf:name    "Cookin's on 3 B."    mbz:ga);
 (mbz:ga2   foaf:made    mbz:t1    mbz:ga);
 (mbz:ga2   dbo:genre    mbz:q1    mbz:ga);
 (mbz:ga2   dbo:genre    mbz:q2    mbz:ga);
 (mbz:ga2   rdf:type     mo:MusicGroup    mbz:ga);
 (mbz:ga3   rdf:type     mo:MusicArtist    mbz:ga);
 (mbz:ga3   foaf:name    "Kylie Auldist"    mbz:ga);
 (mbz:ga3   foaf:made    mbz:t1    mbz:ga);
 (mbz:ga3   rdf:type     mo:SoloMusicArtist    mbz:ga);
 (mbz:m1    rdf:type     mo:Record    mbz:gm);
 (mbz:m1    mo:track_count  12    mbz:gm);
 (mbz:m1    mo:track     mbz:t1    mbz:gm);
 (mbz:m1    mo:track     mbz:t2    mbz:gm);
 (mbz:ga1   rdf:type     mo:MusicArtist    mbz:ga);
 (mbz:ga1   foaf:name    "Kungs"    mbz:ga);
 (mbz:ga1   foaf:made    mbz:t2    mbz:ga) ;
 (mbz:ga1   dbo:genre    mbz:q1    mbz:ga);
 (mbz:ga1   dbo:genre    mbz:q2    mbz:ga);
 (mbz:ga1   rdf:type     mo:SoloMusicArtist    mbz:ga) }
```

**Fig. 7.** $\Delta^-(u)$ for update $u$ in example 4.

```
(mbz:t1    rdf:type   mo:track   mbz:gt);
(mbz:t1    dc:title "ThisGirl(feat. Cookin'On 3B.)" mbz:gt);
{(mbz:ga2   rdf:type     mo:MusicArtist    mbz:ga);
 (mbz:ga2   foaf:name    "Cookin's on 3 B."    mbz:ga);
 (mbz:ga2   foaf:made    mbz:t1    mbz:ga);
 (mbz:ga2   dbo:genre    mbz:q1    mbz:ga);
 (mbz:ga2   dbo:genre    mbz:q2    mbz:ga);
 (mbz:ga2   rdf:type     mo:MusicGroup    mbz:ga);
 (mbz:ga3   rdf:type     mo:MusicArtist    mbz:ga);
 (mbz:ga3   foaf:name    "Kylie Auldist"    mbz:ga);
 (mbz:ga3   rdf:type     mo:SoloMusicArtist    mbz:ga);
 (mbz:m1    rdf:type     mo:Record    mbz:gm);
 (mbz:m1    mo:track_count  12    mbz:gm);
 (mbz:m1    mo:track   mbz:t1    mbz:gm);
 (mbz:m1    mo:track   mbz:t2    mbz:gm);
 (mbz:ga1   rdf:type     mo:MusicArtist    mbz:ga);
 (mbz:ga1   foaf:name    "Kungs"    mbz:ga);
 (mbz:ga1   foaf:made    mbz:t1    mbz:ga);
 (mbz:ga1   foaf:made    mbz:t2    mbz:ga) ;
 (mbz:ga1   dbo:genre    mbz:q1    mbz:ga);
 (mbz:ga1   dbo:genre    mbz:q2    mbz:ga);
 (mbz:ga1   rdf:type   mo:SoloMusicArtist    mbz:ga) }
```

**Fig. 8.** $\Delta^+(u)$ for update $u$ in example 4.

18      F. Author et al.



**Fig. 9.** *LinkedBrainz* Live tool.

## 7   Implementation and Experiments

In order to test the proposed strategy, the *LinkedBrainz Live* tool (*LBL* tool) was implemented.The *LBL tool* propagates updates over the **MusicBrainz** relational database (**MBD** database) to an external materialized RDF view, called **LinkedMusicBrainz** view (**LMB** view).

Figure 9 shows the main components of the *LBL* tool, which are described in what follows.

- The **MBD database** is a local replica of the **MusicBrainz** database. The **MBD** database scheme contains 411 relations, and it includes information about artists, release groups, releases, recordings, works, and labels. The decompressed database dump has about 13 GB and was stored in PostgreSQL version 9.4.
- The *Music Ontology* vocabulary (MO, Last accessed in Feb/2022) is used for publishing the **LMB** view. The mapping for translating **MBD** data into the *Music Ontology* vocabulary is shown in https://doi.org/10.5281/zenodo.6465759.
- The **Triggers** are responsible for computing and publishing the changesets for updates on relevant relation of the **MBD** database. The **MBD** database scheme has 43 relations that are relevant to the **LMB** view. The triggers required for the relevant relations are presented in https://doi.org/10.5281/zenodo.6465759.
- The **LBL update extractor** extracts updates from the replication file provided by MusicBrainz, which contains a sequential list of the update instructions processed by the **MusicBrainz** database. When there is a new replication file, the updates should be extracted and then executed against the local replica of the **MusicBrainz** database.
- The **Synchronization tool** enables the **LMB** view to stay synchronized with the **MBD** database. It simply downloads the changeset files sequentially, creates the appropriate INSERT/DELETE statement and executes it against the **LMB** view triplestore.

The experiments measure the time spent to compute the changeset for 109 replication files published by MusicBrainz. A total number of 556,872 updates were processed, 332,172 of which were relevant to the **LMB** view. Table V shows a summary of the processed replication files. On average, a replication file contained 5,066 updates, 3,202 of which were relevant to **LMB** view, and the average time to compute the changeset was 3,574 seconds. The biggest replication file contained 10,455 updates, 7,034 of which were relevant to LMB view, and the time to compute the changeset was 6,449 seconds. The smallest replication file had 1,230 updates, 708 of which were relevant to **LMB** view, and the time to compute the changeset was 445 seconds.

The experiments were divided into three parts. The first part measures the overhead that the triggers cause in the performance of the data source updates. The second part compares the proposed incremental maintenance strategy for RDB2RDF views against the full and partial re-materialization of RDB2RDF views. Finally, the third part compares the proposed strategy against the mechanism for incremental maintenance of relational view supported by the Oracle Database.

**Part I - Overhead caused by Computing Changesets using Triggers**

This experiment measures the average time spent by the triggers to compute the changesets ($\Delta^-$ and $\Delta^+$) for updates on relevant relations of the **LMB** view. Table VI summarizes the average time required to

Publication and Maintenance of RDB2RDF Views Externally Materialized in Enterprise Knowledge Graphs      19

**Table V.** Summary of replication files(109 replication files)

compute the changesets for updates on the relevant relations *Artist*, *Medium*, *Recording*, *Track*, and *Credit*, which are among the most updated relations in the **MusicBrainz** database.

For each relevant relation (RR), Table VI shows: the average number of updates by replication files; the average number of relevant tuples by update on RR; and the average time (in milliseconds) to compute $\Delta^-$ and $\Delta^+$ per update on RR. The time was computed separately for steps 1 and 2 of the procedures **COMPUTE_$\Delta^-[R]$** and **COMPUTE_$\Delta^+[R]$**. The experiments demonstrated that the runtime for computing the changeset is negligible because the number of tuples that are relevant to an update is relatively small. For example, in the worst case in Table VI, the average time to compute $\Delta^-$ and $\Delta^+$ was less than 1.2 seconds in the relation *Credit*. These results indicate that the proposed strategy can support live synchronization for large RDB2RDF views.

**Table VI.** Changeset Computation Performance for Updates on some Relevant Relations

**Part II - Evaluation of relevant tuples re-materialization against full and partial re-materialization approaches**

As part of the evaluation, the proposed incremental strategy was compared with full and partial rematerializations, for updates on relevant relations of the **LMB** view. Figure 10 shows the comparison for updates on the relevant relations *Artist*, *Medium*, *Recording* and *Track*. Figure 10 shows:

- (1) The average time spent to compute $\Delta^-$ and $\Delta^+$ (see Table VI) considering only the tuples that are relevant to the update.
- (2) The time spent for materialization of the relations that are relevant to the update (*Partial Rematerialization*). The time is computed by the sum of the time for materializing each relevant relation. Table VII shows the size (number of tuples), the time (in milliseconds) spent to rematerialize some of the relevant relations.
- (3) The time spent to rematerialize the view (*Full Rematerialization*). The time is computed by the sum of the time for materializing all pivot relations.

**Table VII.** Materialization time for some of the Relevant Relations

The average time to rematerialize the **LMB** view was 206 minutes (12,360,871 milliseconds). Note that, for updates on relations in Figure 10 (*Track, Artist, Recording* and *Medium*), the difference between full and partial strategies is not very significant. That is because, the updates on those relations are relevant to other relations, and the time spent to materialize them is almost 73% of the time spent to materialize the **LMB** view.

As Figure 10 clearly shows, the time to compute the changeset with the proposed approach is almost three orders of magnitude smaller than partial rematerialization, and three orders of magnitude smaller than full rematerialization strategy. Thus, one may conclude that, in a situation where the RDB2RDF view should be frequently updated, the incremental strategy far outperforms full rematerialization, and also partial rematerialization. The results also show that full rematerialization and partial rematerialization are not a solution for live synchronization of large RDB2RDF views.

**Part III - Evaluation of our strategy against incremental maintenance of relational views**

This experiment compares the proposed incremental maintenance strategy with the mechanism for incremental maintenance of relational view supported by the Oracle Database. It was not possible to use the PostgreSQL database because it has no support for incremental view maintenance.

The mechanism for incremental maintenance (incremental refresh) of relational view implemented by Oracle does not support views with duplicate (Griffin and Libkin, 1995). To create a materialized view in
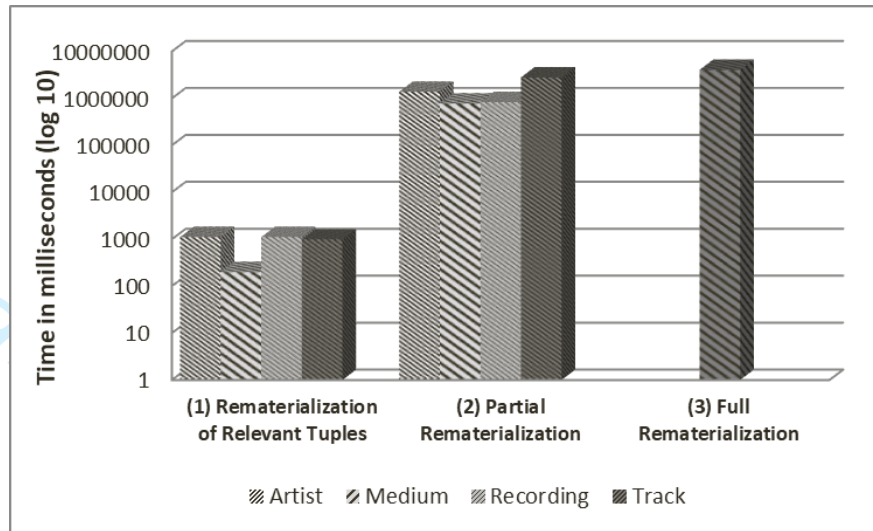
20      F. Author et al.



**Fig. 10.** Comparison of the three rematerialization approaches.

Oracle, and use the incremental refresh mechanism, the select clause should include the key (rowid) for all base relations. It also requires the creation of logs table to keep track of updates on the base relation. The changesets are computed using the states of the materialized view, log tables and base relations.
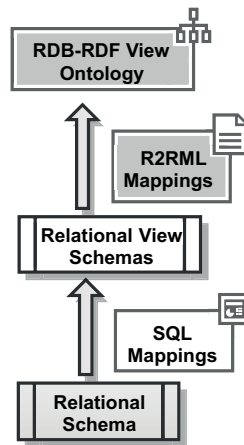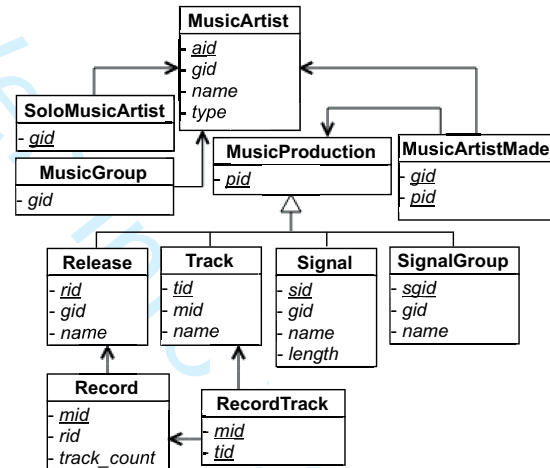


**Fig. 11.** Three-Level Schema.



**Fig. 12.** Relational View Schema.

For the experiments, a set of relational views were defined in such a way that the mappings from relational view schemas to RDF view ontology were *direct mappings* (Group, 2012). The proposal was to break the definition of the RDB to RDF mappings in two stage, as depicted in Figure 11. The *SQL mappings*, from relation schema to relational view schema, absorb the complexity of the mappings, so that the mappings from the relational views to RDB RDF views are direct mappings. Authors in (Anonymous, 2014) presents a strategy to automatically generate the relational view schema and direct R2RML mappings based on a set of transformation rules. Figure 12 depicted the schema for some relational view used in our experiments. The SQL definition for the relational views in Figure 12 are shown in https://doi.org/10.5281/zenodo.6465759.

Notice that an update on a base relation may affect several relational views. Table VIII shows the list of relational views that are relevant to relations *Artist*, *Medium*, *Recording* and *Track*. Updates on these tables

should be propagated to their relevant views. Therefore, the refresh time for an update on a base relation is the sum of the refresh time for all the relational views that are relevant to that base relation.

**Table VIII.** List of Relevant Views to Relevant Relations in *Artist, Medium, Recording* and *Track*.

The experiments adopt the same updates on *Artist, Medium, Recording* and *Track* relations that were used in the first part of the experiment. The updates are applied to the Oracle database, and then the total time to refresh all relevant views is computed.

Table IX shows the results for 5 (five) on each relation. The updates were selected considering the runtime for computing the changeset obtained in the first part of the experiment. Updates 1 and 2 are the updates with the smallest runtime, update 3 has an average runtime, and updates 4 and 5 have the highest runtime. Figure 13 shows the comparison of the results for each relation.

**Table IX.** Time for Changeset Computation and Incremental Refresh.



(a) Updates on relation *Artist*    (b) Updates on relation *Medium*    (c) Updates on relation *Recoding*
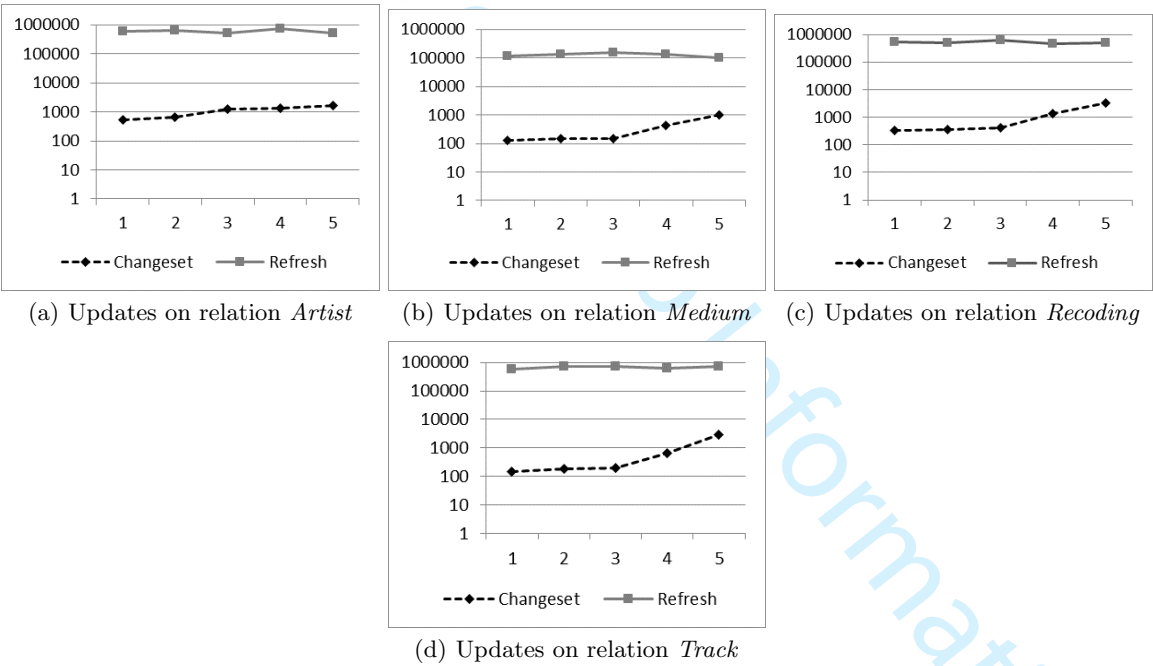


(d) Updates on relation *Track*

**Fig. 13.** Changeset Computation x Incremental Refresh.

Figure 14 shows the comparison considering the average time for both approaches. Notice that the average time to compute the changeset is almost two orders of magnitude smaller than the average refresh time. The experiments also demonstrated that the refresh time increases when the views are very large. Thus, the incremental refresh mechanism is not a good solution for live synchronization of large relational views.

Another limitation of the Oracle mechanism is that it requires access to the view for incremental refresh, which can be very slow when the view is maintained externally, because accessing a remote data source may be too slow. One may conclude that the proposed strategy is much simpler, more efficient, and less restrictive, since the views can have duplicated tuples and are maintained externally.
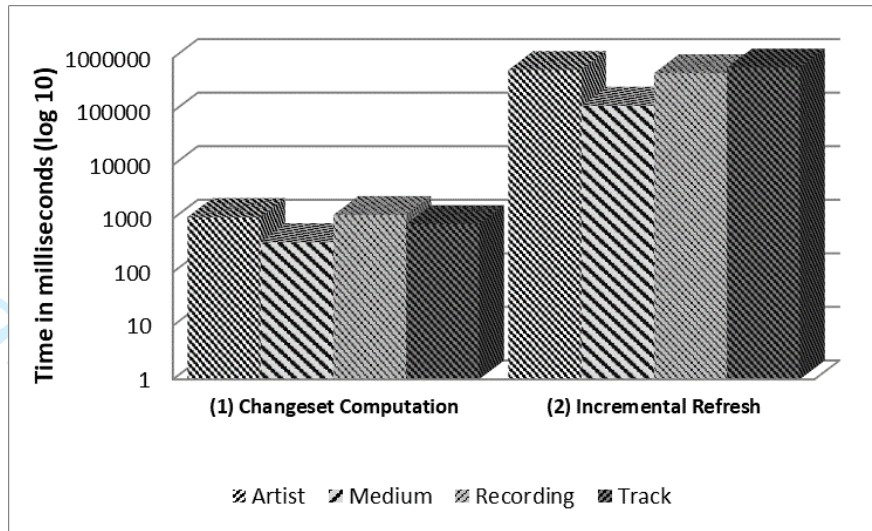
22    F. Author et al.



**Fig. 14.** Comparison of Changeset Computation for RDB2RDF view and Incremental Refresh.

## 8    Conclusions and Final Remarks

This article presented a formal framework for the construction and incremental maintenance of RDB2RDF views, which are externally materialized in an Enterprise Knowledge Graph. In the proposed framework the server computes and publishes changesets, which indicate the difference between two states of the view. The EKG system can then download the changesets and synchronize the externally materialized view. The changesets are computed based solely on the update and the source database state, and no access to the content of the view is required.

In the formal framework, changesets are computed in three steps: identification of relevant relations, identification of relevant tuples, and computation of changesets. The formal framework was based on three key ideas. First, it assumed that the RDB2RDF views are object-preserving, that is, the views preserve the base entities of the source database, rather than creating new entities from the existing ones (Motschnig-Pitrik, 2000). This assumption makes it possible to precisely identify the specific tuples that are relevant to a data source update w.r.t. an RDB2RDF view. Second, the formal framework included a rule language to specify object preserving-views mappings. Third, the proposed framework assumes that the content of an RDB2RD view is stored in an RDF dataset that contains a set of named graphs, used to describe the context in which the triples were produced. The central result of the article, Theorem 1, showed that changesets computed according to the formal framework correctly maintain the RDB2RDF views. The main idea that differentiates the proposed approach from previous work on incremental view maintenance is to explore the object-preserving property of typical RDB2RDF views, so that the solution can also be able to deal with views with duplicates.

In order to test the proposed strategy, the *LinkedBrainz Live* tool (*LBL* tool) was implemented. Very briefly, the *LBL* tool propagates updates over the **MusicBrainz** database (**MBD** database) to the **Linked-MusicBrainz** view (**LMB** view). The **LMB** view is intended to help **MusicBrainz** publish its database as Linked Data. Based on the tool, experiments were conducted in two parts. The first part measured the overhead that the triggers cause in the performance of the data source updates. The second part compared our incremental maintenance strategy against the full and partial re-materialization of RDB2RDF views. The experiments indicated that the proposed strategy supports live synchronization of large RDB2RDF views and that the time taken to compute the changesets with the proposed approach was almost three orders of magnitude smaller than partial and full rematerialization.

# Bibliography

Abiteboul, S., Hull, R. and Vianu, V., eds (1995), *Foundations of Databases: The Logical Level*, 1st edn, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Abiteboul, S., McHugh, J., Rys, M., Vassalos, V. and Wiener, J. L. (1998), Incremental maintenance for materialized views over semistructured data, *in* 'Proceedings of the 24th International Conference on Very Large Data Bases', VLDB '98, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 38–49.

Ali, M. A., Fernandes, A. A. A. and Paton, N. W. (2000), Incremental maintenance of materialized OQl views, *in* 'Proceedings of the 3rd ACM International Workshop on Data Warehousing and OLAP', DOLAP '00, ACM, New York, NY, USA, pp. 41–48.

Ali, M. A., Fernandes, A. A. A. and Paton, N. W. (2003), 'Movie: An incremental maintenance system for materialized object views', *Data Knowl. Eng.* **47**(2), 131–166.

Anonymous (2008), Details omitted for double-blind reviewing, *in* 'Proceedings of the Tenth International Conference on Enterprise Information Systems', Vol. Volume DISI, Barcelona, Spain.

Anonymous (2013), Details omitted for double-blind reviewing, *in* 'Proc. The 12th International Conference on Ontologies, DataBases, and Applications of Semantics', Springer, Graz, Austria.

Anonymous (2014), Details omitted for double-blind reviewing, *in* 'Proceedings of the 29th Annual ACM Symposium on Applied Computing', SAC '14, ACM, New York, NY, USA.

Arispe Riveros, M., Tasnim, M., Graux, D., Orlandi, F. and Collarana, D. (2020), Verbalizing the evolution of knowledge graphs with formal concept analysis, *in* 'Advances in Semantics and Linked Data: Joint Workshop Proceedings from ISWC 2020'.

Calvanese, D., Gal, A., Lanti, D., Montali, M., Mosca, A. and Shraga, R. (2020), 'Mapping patterns for virtual knowledge graphs'.
**URL:** *https://arxiv.org/abs/2012.01917*

Carroll, J. J., Bizer, C., Hayes, P. and Stickler, P. (2005), Named graphs, provenance and trust, *in* 'Proceedings of the 14th International Conference on World Wide Web', WWW '05, ACM, New York, NY, USA, pp. 613–622.

Ceri, S. and Widom, J. (1991), Deriving production rules for incremental view maintenance, *in* 'Proceedings of the 17th International Conference on Very Large Data Bases', VLDB '91, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 577–589.

Das, S., Sundara, S. and Cyganiak, R. (2012), 'R2RML: RDB to RDF mapping language', W3C working draft. W3C working draft. Retrieved from https://www.w3.org/TR/r2rml/.

DBp (Last accessed in Feb/2022), 'Dbpedia. http://wiki.dbpedia.org'.

Ding, L., Xiao, G., Pano, A., Stadler, C. and Calvanese, D. (2021), 'Towards the next generation of the linkedgeodata project using virtual knowledge graphs', *Journal of Web Semantics* **71**, 100662.
**URL:** *https://www.sciencedirect.com/science/article/pii/S1570826821000378*

Endris, K. M., Faisal, S., Orlandi, F., Auer, S. and Scerri, S. (2015), Interest-based RDF update propagation, *in* 'Proceedings of the 14th International Conference on The Semantic Web - ISWC 2015 - Volume 9366', Springer-Verlag New York, Inc., New York, NY, USA, pp. 513–529.

Faisal, S., Endris, K. M., Shekarpour, S., Auer, S. and Vidal, M.-E. (2016), *Co-evolution of RDF Datasets*, Springer International Publishing.

Fegaras, L. (2011), Incremental maintenance of materialized xml views, *in* 'International Conference on Database and Expert Systems Applications', Springer-Verlag, Berlin, Heidelberg, pp. 17–32.

Griffin, T. and Libkin, L. (1995), 'Incremental maintenance of views with duplicates', *SIGMOD Rec.* **24**(2), 328–339.

Group, R. W. (2012), 'A direct mapping of relational data to RDF'. W3C Recommendation. Retrieved from https://www.w3.org/TR/rdb-direct-mapping/.

Hert, M., Reif, G. and Gall, H. C. (2011), A comparison of RDB-to-RDF mapping languages, *in* 'Proceedings of the 7th International Conference on Semantic Systems', I-Semantics '11, ACM, New York, NY, USA, pp. 25–32.

Jin, X. and Liao, H. (2010), An algorithm for incremental maintenance of materialized xpath view, *in* 'Proceedings of the 11th International Conference on Web-age Information Management', WAIM'10, Springer-Verlag, Berlin, Heidelberg, pp. 513–524.

24      F. Author et al.

Kalaycı, E. G., Grangel González, I., Lösch, F., Xiao, G., ul Mehdi, A., Kharlamov, E. and Calvanese, D. (2020), Semantic integration of bosch manufacturing data using virtual knowledge graphs, *in* J. Z. Pan, V. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne and L. Kagal, eds, 'The Semantic Web – ISWC 2020', Springer International Publishing, Cham, pp. 464–481.

Konstantinou, N., Spanos, D.-E., Kouis, D. and Mitrou, N. (2015), 'An approach for the incremental export of relational databases into RDF graphs', *International Journal on Artificial Intelligence Tools* **24**(2), 1540013.

Lenzerini, M. (2002), Data integration: a theoretical perspective, *in* 'Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems', PODS '02, ACM, New York, NY, USA, pp. 233–246.

LG (Last accessed in Feb/2022), 'Linkedgeodata. http://linkedgeodata.org/'.

Liefke, H. and Davidson, S. B. (2000), View maintenance for hierarchical semistructured data, *in* 'Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery', DaWaK 2000, Springer-Verlag, London, UK, UK, pp. 114–125.

MBz (Last accessed in Feb/2022), 'Musicbrainz database scheme. https://wiki. .musicbrainz.org/musicbrainz_database/schema'.

MO (Last accessed in Feb/2022), 'Music ontology. http://musicontology.com/'.

Motschnig-Pitrik, R. (2000), The viewpoint abstraction in object-oriented modeling and the UML, *in* 'Proceedings of the 19th International Conference on Conceptual Modeling', ER'00, Springer-Verlag, Berlin, Heidelberg, pp. 543–557.

Murlak, F., Libkin, L., Barceló, P. and Arenas, M. (2014), *Foundations of Data Exchange*, Cambridge University Press, New York, NY, USA.

Mus (Last accessed in Feb/2022), 'Musicbrainz. http://musicbrainz.org/doc/about'.

nQu (2014), 'RDF 1.1 N-Quads, a line-based syntax for RDF datasets'. W3C Recommendation. Retrieved from https://www.w3.org/TR/2014/REC-n-quads-20140225/.

Ontotext (2022), *GraphDB Free Documentation Release 9.11.0*.

Pan, J., Vetere, G., Gomez-Perez, J.M. and Wu, H., eds (2017), *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, Springer International Publishing.

Papavasileiou, V., Flouris, G., Fundulaki, I., Kotzinos, D. and Christophides, V. (2013), 'High-level change detection in RDF(S) KBs', *ACM Trans. Database Syst.* **38**(1), 1:1–1:42.

Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M. and Rosati, R. (2008), Linking data to ontologies, *in* S. Spaccapietra, ed., 'Journal on Data Semantics X', Springer-Verlag, Berlin, Heidelberg, pp. 133–173.
    **URL:** *http://dl.acm.org/citation.cfm?id=1793934.1793939*

Roussakis, Y., Chrysakis, I., Stefanidis, K., Flouris, G. and Stavrakas, Y. (2015), A flexible framework for understanding the dynamics of evolving RDF datasets, *in* 'The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, October 11-15, 2015, Proceedings, Part I', Springer, Cham, Bethlehem, PA, USA, pp. 495–512.

Sequeda, J. F., Arenas, M. and Miranker, D. P. (2014), OBDA: query rewriting or materialization? in practice, both!, *in* 'The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, October 19-23, 2014. Proceedings, Part I', Springer, Cham, Riva del Garda, Italy, pp. 535–551.

Sequeda, J., Priyatna, F. and Villazón-Terrazas, B. (2012), Relational database to RDF mapping patterns, *in* 'Proceedings of the 3rd International Conference on Ontology Patterns - Volume 929', WOP'12, CEUR-WS.org, Aachen, Germany, Germany, pp. 97–108.

Singh, A. K. (2019), 'Regions in a linked dataset for change detection', *ArXiv* **abs/1905.07663**.

Tasnim, M., Collarana, D., Graux, D., Orlandi, F. and Vidal, M.-E. (2019), Summarizing entity temporal evolution in knowledge graphs, *in* 'Companion Proceedings of The 2019 World Wide Web Conference', WWW '19, Association for Computing Machinery, New York, NY, USA, p. 961–965.
    **URL:** *https://doi.org/10.1145/3308560.3316521*

Volz, R., Staab, S. and Motik, B. (2005), 'Incrementally maintaining materializations of ontologies stored in logic databases', *J. Data Semantics* **2**, 1–34.

Xiao, G., Ding, L., Cogrel, B. and Calvanese, D. (2019), 'Virtual Knowledge Graphs: An Overview of Systems and Use Cases', *Data Intelligence* **1**(3), 201–223.

Xiao, G., Lanti, D., Kontchakov, R., Komla-Ebri, S., Güzel-Kalaycı, E., Ding, L., Corman, J., Cogrel, B., Calvanese, D. and Botoeva, E. (2020), The virtual knowledge graph system ontop, *in* J. Z. Pan, V. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne and L. Kagal, eds, 'The Semantic Web – ISWC 2020', Springer International Publishing, Cham, pp. 259–277.

Publication and Maintenance of RDB2RDF Views Externally Materialized in Enterprise Knowledge Graphs        25

Zeginis, D., Tzitzikas, Y. and Christophides, V. (2011), 'On computing deltas of RDF/S knowledge bases', *ACM Trans. Web* **5**(3), 14:1–14:36.

Zhao, W., Rusu, F., Dong, B., Wu, K. and Nugent, P. (2017), Incremental view maintenance over array data, *in* 'Proceedings of the 2017 ACM International Conference on Management of Data', SIGMOD '17, ACM, New York, NY, USA, pp. 139–154.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60

| Built-in predicate | Intuitive definition |
|---|---|
| $nonNull(v)$ | $nonNull(v)$ holds iff value $v$ is not null |
| $RDFLiteral(u, A, R, v)$ | Given a value $u$, an attribute $A$ of $R$, a relation name $R$, and a literal $v$, $RDFLiteral(u, A, R, v)$ holds iff $v$ is the literal representation of $u$, given the type of $A$ in $R$ |
| $F(r, s)$ where $F$ is a foreign key of the form $F(R{:}L, S{:}K)$ | Given a tuple $r$ of $R$ and a tuple $s$ of $S$, $F(r, s)$ holds iff $r$ is related to $s$ by a foreign key $F$ |
| $hasURI(P, A, s)$ where $P$ is the namespace prefix and $A$ is a list of attributes of $R$ | Given a tuple $r$ of $R$, $hasURI(P, A, s)$ holds iff $s$ is the URI obtained by concatenating the namespace prefix $P$ and the attribute values $a_1, \ldots, a_n$ where $A$ is the list (in Prolog notation) $[a_1, \ldots, a_n]$. To further simply matters, we admit denoting a list with a single element, "$[a]$", simply as "$a$". |

Table I: Examples of built-in predicates

2

| TR | Transformation Rules |
|---|---|
| CTR | $\psi$: $C(\mathrm{x}) \leftarrow R(r)$, $\mathrm{B}[r, \mathrm{x}]$, where <br> - $\psi$ is the name of the CTR. <br> - $C$ is a class in $V$ and $x$ is a scalar variable whose value is a URI. <br> - $R$ is relation in $\boldsymbol{S}$ and $r$ is tuple variable; $R$ is called the pivot relation and $r$ the pivot tuple variable of the rule. <br> - $B[r, x]$ is a list of literals. |
| DTR | $\psi$: $P(\mathrm{x}, \mathrm{y}) \leftarrow R(r)$, $\mathrm{B}[r, \mathrm{x}]$, $\mathrm{H}[r, \mathrm{y}]$, where <br> - $\psi$ is the name of the DTR. <br> - $P$ is a datatype property in $V$ with domain $D$. <br> - "$\mathrm{Rn}(r), B[r, x]$" is the right-hand side for the CTR that matches class $D$ with pivot relation $R$. <br> - $H[r, y]$ is a list of literals which define a predicate $H$ that relates a tuple $r$ and data values in $y$. |
| OTR | $\psi$: $P(\mathrm{x}, \mathrm{y}) \leftarrow R_D(r_1), B_D[r_1, x], H[r_1, r_2], R_G(r_2), B_G[r_2, y]$, where <br> - $\psi$ is the name of the OTR. <br> - $P$ is an object property in $V$ with domain $D$ and range $G$. <br> - "$R_D(r_1), B_D[r_1, x]$" is the RHS of the CTR $\psi_D$ that matches class $D$ with pivot relation $R_D$. <br> - "$R_G(r_2), B_G[r_2, y]$" is the RHS of the CTR $\psi_G$ that matches class $G$ with pivot relation $R_G$. <br> - $H[r_1, r_2]$ is a list of literals which define a predicate $H$ that relates tuples in $R_D$ with tuples in $R_G$. |

Table II: Transformation Rules

3

| TR | Transformation Rule Pattern |
|---|---|
| CTR | $\psi{:}C(s) \leftarrow R(r),\ hasURI(P,\ A,\ s),\ \delta(r)$, where<br>- $R$ is a relation name in $\boldsymbol{S}$ and $r$ a is tuple variable associated with $R$,<br>- $A$ is a list of attributes of a primary key of $R$,<br>- $\delta$ is an optional selection over $R$, and<br>- $P$ is a namespace prefix |
| OTR | $\psi{:}P(s,\ o) \leftarrow R_D(r),\ B_D[r,\ s], F_1(r,\ r_1),\ \ldots,\ F_n(r_{n-1},\ r_n),\ R_G(r_n),\ B_G[r_n,\ o]$,<br>where:<br>- $P$ is an object property of $V$,<br>- $R$ is a relation name in $\boldsymbol{S}$ and $r$ is a tuple variable associated with $R$,<br>- $[F_1,\ \ldots,\ F_n]$ is a path from $R$ to relation $R_n$ where $F_1$ relates $R$ and $R_1$, and<br>$F_i$ relates $R_{i-1}$ and $R_i$, and $r_i$ is a tuple variable associated with $R_i$, $1< i \leq n$ |
| DTR | $\psi{:}P(s,\ v) \leftarrow R(r),\ B[r,\ s],\ F_1(r,\ r_1), \ldots,\ F_n(r_{n-1},\ r_n),\ nonNull(r_n.A_1),\ \ldots,$<br>$nonNull(r_n.A_k),\ RDFLiteral(r_n.A_1,\ "A_1",\ "R_n",\ v_1),\ \ldots,$<br>$RDFLiteral(r_n.A_k,\ "A_k",\ "R_n",\ v_k),\ T([v_1,\ \ldots,\ v_k],\ v)$, where<br>- $P$ is a datatype property of $V$,<br>- $R$ is a relation name in $\boldsymbol{S}$ and $r$ a is tuple variable associated with $R$,<br>- $[F_1,\ \ldots,\ F_n]$ is a path from $R$ to relation $R_n$ where $F_1$ relates $R$ and $R_1$, and<br>$F_i$ relates $R_{i-1}$ and $R_i$, and $r_i$ is a tuple variable associated with $R_i$, $1< i \leq n$<br>- $A_1,\ \ldots,\ A_k$ are the attributes of $R_n$. If there is no path, then $A_1,\ \ldots,\ A_k$ are attributes of $R$<br>- $T$ is an optional function that transforms values of attributes $A_1,\ \ldots,\ A_k$ to values of property $P$ |

Table III: Transformation Rule Patterns

4

| TR | Transformation Rules |
|---|---|
| $\Psi_1$ | $mo{:}MusicArtist(s) \leftarrow Artist(r),\ hasURI(mbz{:},r.gid,\ s)$ |
| $\Psi_2$ | $mo{:}SoloMusicArtist(s) \leftarrow Artist(r),\ hasURI(mbz{:},r.gid\ ,\ s),\ (r.type = 1)$ |
| $\Psi_3$ | $mo{:}MusicGroup(s) \leftarrow Artist(r),\ hasURI(mbz{:},\ r.gid,\ s),\ (r.type = 2)$ |
| $\Psi_4$ | $mo{:}Record(s) \leftarrow Medium(r),\ hasURI(mbz{:},\ r.mid,\ s)$ |
| $\Psi_5$ | $mo{:}Track(s) \leftarrow Track(r),\ hasURI(mbz{:},\ r.tid,\ s)$ |
| $\Psi_7$ | $foaf{:}made(s,\ g) \leftarrow Artist(r),\ hasURI(mbz{:},\ r.\ gid,\ s),\ fk1(r,\ r1),\ fk2(r1,\ r2),$ $fk3(r2,\ r3),\ Track(r3),\ hasURI(mbz{:},\ r3.tid,\ g)$ |
| $\Psi_8$ | $mo{:}track\ (s,\ g) \leftarrow Medium(r),\ hasURI(mbz{:},\ r.mid,\ s),\ fk4(r,\ f),\ Track(f),$ $hasURI(mbz{:},\ f.tid,\ g),\ hasURI(mbz{:},\ f.tid,\ g)$ |
| $\Psi_{10}$ | $foaf{:}name(s,\ v) \leftarrow Artist(r),\ hasURI(mbz{:},\ r.gid,\ s),\ nonNull(r.name),$ $RDFLiteral(r.name,\ "name",\ "Artist",\ v)$ |
| $\Psi_{11}$ | $mo{:}track\_count(s,\ v) \leftarrow Medium(r), hasURI(mbz{:},\ r.\ mid,\ s), nonNull(r.track\_count),$ $RDFLiteral(r.track\_count,\ "track\_count",\ "Medium",\ v)$ |
| $\Psi_{16}$ | $dbo{:}Genre(s) \leftarrow Tag(r),\ hasURI(mbz{:},\ r.gid,\ s)$ |
| $\Psi_{20}$ | $dc{:}title(s,\ v) \leftarrow Tag(r),\ hasURI(mbz{:},\ r.gid,\ s),\ nonNull(r.name),$ $RDFLiteral(r.name,\ "title",\ "Tag",\ v)$ |
| $\Psi_{24}$ | $dbo{:}genre(s,\ g) \leftarrow Artist(r),\ hasURI(mbz{:},\ r.\ gid,\ s),\ fk1(r,\ r1),\ fk2(r1,\ r2),$ $fk7(r2,\ r3),\ fk9(r3,\ r4),\ fk10(r4,\ r5), Tag(r5),$ $hasURI(mbz{:},\ r5.qid,\ g)$ |

Table IV: Examples of Transformation Rules

5

| Replication File (size) | N$^o$ updates | N$^o$ Relevant Updates | Time to Compute Changeset(ms) |
|---|---|---|---|
| **Average** | 5,066 | 3,022 | 3,574,000 |
| **Biggest** | 10,455 | 7,034 | 6,449,000 |
| **Smallest** | 1,230 | 708 | 445,000 |

Table V: Summary of replication files(109 replication files)

6

| Relevant Relation | Avg Nᵒ of Relevant Updates by Rep. Files | Avg Nᵒ of Tuples by Updates | $COMPUTE\_\Delta^-[R]$ (ms) | | $COMPUTE\_\Delta^+[R]$ (ms) | |
|---|---|---|---|---|---|---|
| | | | Step 1 | Step 2 | Step 1 | Step 2 |
| *Artist* | 41.49 | 5.87 | 439 | 510 | 6 | 62 |
| *Medium* | 324.56 | 18.52 | 134 | 33 | 4 | 12 |
| *Recording* | 644.50 | 4.99 | 156 | 691 | 5 | 194 |
| *Track* | 585.28 | 4.87 | 145 | 601 | 6 | 211 |
| *Credit* | 785.06 | 10.49 | 151 | 813 | 4 | 224 |

Table VI: Changeset Computation Performance for Updates on some Relevant Relations

| Relevant Relation | N$^o$ of Tuple (k) | Materialization Time (ms) |
|---|---|---|
| Artist | 1,962 | 2,697,630 |
| Medium | 3,587 | 480,252 |
| Recording | 26,759 | 961,251 |
| Track | 37,041 | 4,980,582 |
| Credit | 2,278 | * not a pivot relation |

Table VII: Materialization time for some of the Relevant Relations

8

| Relevant Relation (RR) | Relevant views | N$^o$ of Relevant views |
|---|---|---|
| *Artist* | *MusicArtist, SoloMusicArtist, MusicGroup, MusicArtisMade, MusicArtistGender, MusicArtistBaseNear, MusicArtistIsPrimaryTopicOf, MusicArtistAccount, MusicArtistMemberOf, MusicArtistSameAs, MusicArtistSeeAlso, MusicArtistComment, MusicArtisHasTag, MusicArtistComposer* | 14 |
| *Medium* | *Record, RecordTrack, RecordMediaType, ReleaseRecord* | 4 |
| *Recording* | *Signal, SignalComment, MusicArtisMade, TrackPublicationOf* | 4 |
| *Track* | *Track, TrackDuration, TrackPublicationOf, MusicArtisMade* | 4 |

Table VIII: List of Relevant Views to Relevant Relations in *Artist, Medium, Recording* and *Track*.

9

| Update | Artist | | Medium | | Recording | | Track | |
|---|---|---|---|---|---|---|---|---|
| | (C) | (R) | (C) | (R) | (C) | (R) | (C) | (R) |
| 1 | 520 | 604228 | 130 | 117946 | 341 | 533080 | 150 | 579260 |
| 2 | 661 | 618189 | 151 | 132401 | 371 | 498770 | 180 | 713914 |
| 3 | 1211 | 510184 | 151 | 154775 | 430 | 609917 | 201 | 705353 |
| 4 | 1391 | 725992 | 420 | 136459 | 1401 | 480010 | 640 | 619580 |
| 5 | 1600 | 517710 | 971 | 101685 | 3320 | 486462 | 2800 | 704016 |

Table IX: Time for Changeset Computation and Incremental Refresh.
*Legend:* (C) Changeset (R) Refresh

10

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
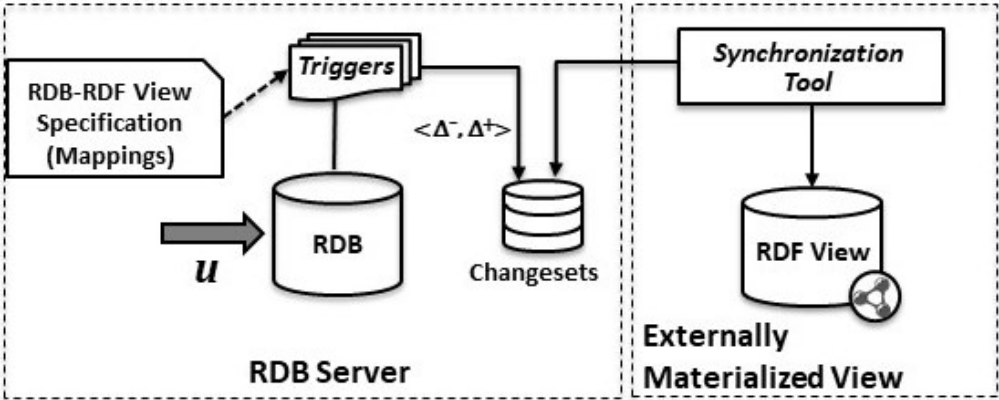


Fig. 1. Incremental Maintenance of Externally Materialized RDB2RDF View.

144x58mm (96 x 96 DPI)

**RecordingTag**
- *sid*
- *qid*

fk10

**Tag**
- *qid*
- *name*

fk9

**Recording**
- *sid*
- *gid*
- *name*
- *cid*
- *length*

**ReleaseGroup**
- *pid*
- *gid*
- *name*
- *cid*

fk7

fk2

fk8

fk3

**ArtistCredit**
- *cid*
- *pos*
- *aid*

**Credit**
- *cid*
- *name*

**Track**
- *tid*
- *mid*
- *name*
- *cid*

fk1

fk6

fk4

**Artist**
- *aid*
- *gid*
- *name*
- *type*

**Release**
- *rid*
- *gid*
- *name*
- *cid*

**Medium**
- *mid*
- *rid*
- *track_count*

fk5

2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

## Artist

| aid | gid | name | type |
|-----|-----|------|------|
| a1 | ga1 | Kungs | 1 |
| a2 | ga2 | Cookin' on 3 B. | 2 |
| a3 | ga3 | Kylie Auldist | 1 |

## ArtistCredit

| cid | pos | aid |
|-----|-----|-----|
| c1 | 1 | a1 |
| c1 | 2 | a2 |
| c2 | 1 | a2 |
| c2 | 2 | a3 |
| c3 | 1 | a1 |

## Credit

| cid | name |
|-----|------|
| c1 | Kungs vs. Cookin' on 3 B. |
| c2 | Cookin' on 3 B. feat. Kylie Auldist |
| c3 | Kungs |

## Release

| rid | gid | name | cid |
|-----|-----|------|-----|
| r1 | gr1 | Layers | c3 |

## Track

| tid | mid | name | cid |
|-----|-----|------|-----|
| t1 | m1 | This Girl | c2 |
| t2 | m1 | Don't You Know | c3 |

## RecordingTag

| sid | qid |
|-----|-----|
| s1 | q1 |
| s1 | q2 |
| s2 | q2 |

## Tag

| qid | name |
|-----|------|
| q1 | pop |
| q2 | dance |

## Medium

| mid | rid | track_count |
|-----|-----|-------------|
| m1 | r1 | 12 |

## Recording

| sid | gid | name | cid | length |
|-----|-----|------|-----|--------|
| s1 | gs1 | This Girl | c1 | 3:15 |
| s2 | gs2 | Don't You Know | c3 | 3:04 |

## ReleaseGroup

| pid | gid | name | cid |
|-----|-----|------|-----|
| p1 | gp1 | Layers | c3 |

Publication and Maintenance of RDB2RDF Views Externally Materialized in Enterprise Knowledge Graphs     15

---

**ALGORITHM 1:** Algorithm for computing changeset for updates on a relation $R$

---

**Input:**

    $u = (D, I)$ — an update on $R$;

    $\sigma_0$ and $\sigma_1$ — the states of the database respectively before and after the update $u$;

**Output:**

    $\Delta^-$ and $\Delta^+$

**2**

**3**   **if** *$R$ is relevant to the view $\mathcal{W}$ (Definition 7; (ii))* **then**

**4**

**5**       **Phase 1:** Before the update do:

**6**           1.1 Compute $\mathcal{P}_0$, the set of tuples in $\sigma_0$ that are relevant to $u$ (Definition 9)

**7**           1.2 Compute $\Delta^- := \bigcup\limits_{p \in (\mathcal{P}_0)} M[p]$;

**8**           `// `$\Delta^-$` contains the union of the RDF states of tuples in `$\mathcal{P}_0$` (Definition 10)`

**9**

**10**       **Phase 2:** After the update do:

**11**           2.1 Compute $\mathcal{P}_1$, the set of tuples in $\sigma_1$ that are relevant to $u$ (Definition 9)

**12**           2.2 Compute $\Delta^+ := \bigcup\limits_{p \in (\mathcal{P}_1)} M[p]$;

**13**           `// `$\Delta^+$` contains the union of the RDF states of tuples in `$\mathcal{P}_1$` (Definition 10)`

**14**   **return** $(\Delta^+, \Delta^-)$;

---

| |
|---|
| **BEFORE**{update, insert, delete} ON $R$ <br> **BEGIN** <br>    $\Delta^- := \textbf{\textit{COMPUTE\_}}\Delta^-[R](D, I)$, where <br>       $D$ is the set of deleted tuples and <br>       $I$ is the set of inserted tuples; <br>    ADD $\Delta^-$ to changeset file of $\mathcal{W}$ <br> **END** <br><br> (a) |

| |
|---|
| **AFTER**{update, insert, delete} ON $R$ <br> **BEGIN** <br>    $\Delta^+ := \textbf{\textit{COMPUTE\_}}\Delta^+[R](D, I)$, where <br>       $D$ is the set of deleted tuples and <br>       $I$ is the set of inserted tuples; <br>    ADD $\Delta^+$ to changeset file of $\mathcal{W}$ <br> **END** <br><br> (b) |

**Fig. 5.** Triggers templates for updates on $R$.

## Artist

| aid | gid | name | type |
|-----|-----|------|------|
| a1 | ga1 | Kungs | 1 |
| a2 | ga2 | Cookin' on 3 B. | 2 |
| a3 | ga3 | Kylie Auldist | 1 |

## ArtistCredit

| cid | pos | aid |
|-----|-----|-----|
| c1 | 1 | a1 |
| c1 | 2 | a2 |
| c2 | 1 | a2 |
| c2 | 2 | a3 |
| c3 | 1 | a1 |

## Credit

| cid | name |
|-----|------|
| c1 | Kungs vs. Cookin' on 3 B. |
| c2 | Cookin' on 3 B. feat. Kylie Auldist |
| c3 | Kungs |

## Track

| tid | mid | name | cid |
|-----|-----|------|-----|
| t1 | m1 | This Girl(feat. Cookin' On 3 B.) | c1 |
| t2 | m1 | Don't You Know | c3 |

## Release

| rid | gid | name | cid |
|-----|-----|------|-----|
| r1 | gr1 | Layers | c3 |

## Recording

| sid | gid | name | cid | length |
|-----|-----|------|-----|--------|
| s1 | gs1 | This Girl | c1 | 3:15 |
| s2 | gs2 | Don't You Know | c3 | 3:04 |

## RecordingTag

| sid | qid |
|-----|-----|
| s1 | q1 |
| s1 | q2 |
| s2 | q2 |

## Tag

| qid | name |
|-----|------|
| q1 | pop |
| q2 | dance |

## Medium

| mid | rid | track_count |
|-----|-----|-------------|
| m1 | r1 | 12 |

## ReleaseGroup

| pid | gid | name | cid |
|-----|-----|------|-----|
| p1 | gp1 | Layers | c3 |

Publication and Maintenance of RDB2RDF Views Externally Materialized in Enterprise Knowledge Graphs        17

```
{(mbz:t1    rdf:type     mo:track     mbz:gt);
 (mbz:t1    dc:title     "This Girl"    mbz:gt);
 (mbz:ga2   rdf:type     mo:MusicArtist    mbz:ga);
 (mbz:ga2   foaf:name    "Cookin's on 3 B."    mbz:ga);
 (mbz:ga2   foaf:made    mbz:t1    mbz:ga);
 (mbz:ga2   dbo:genre    mbz:q1    mbz:ga);
 (mbz:ga2   dbo:genre    mbz:q2    mbz:ga);
 (mbz:ga2   rdf:type     mo:MusicGroup    mbz:ga);
 (mbz:ga3   rdf:type     mo:MusicArtist    mbz:ga);
 (mbz:ga3   foaf:name    "Kylie Auldist"    mbz:ga);
 (mbz:ga3   foaf:made    mbz:t1    mbz:ga);
 (mbz:ga3   rdf:type     mo:SoloMusicArtist    mbz:ga);
 (mbz:m1    rdf:type     mo:Record    mbz:gm);
 (mbz:m1    mo:track_count  12    mbz:gm);
 (mbz:m1    mo:track     mbz:t1    mbz:gm);
 (mbz:m1    mo:track     mbz:t2    mbz:gm);
 (mbz:ga1   rdf:type     mo:MusicArtist    mbz:ga);
 (mbz:ga1   foaf:name    "Kungs"    mbz:ga);
 (mbz:ga1   foaf:made    mbz:t2    mbz:ga) ;
 (mbz:ga1   dbo:genre    mbz:q1    mbz:ga);
 (mbz:ga1   dbo:genre    mbz:q2    mbz:ga);
 (mbz:ga1   rdf:type     mo:SoloMusicArtist    mbz:ga) }
```

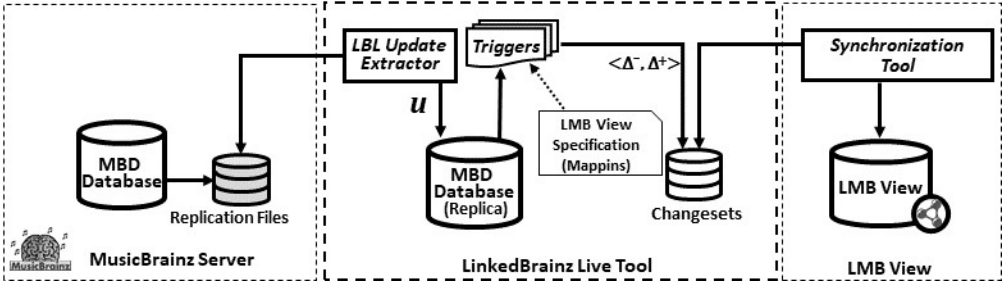**Fig. 7.** $\Delta^-(u)$ for update $u$ in example 4.

Publication and Maintenance of RDB2RDF Views Externally Materialized in Enterprise Knowledge Graphs     19

```
(mbz:t1    rdf:type   mo:track     mbz:gt);
(mbz:t1    dc:title   "ThisGirl(feat. Cookin'On 3B.)"  mbz:gt);
{(mbz:ga2   rdf:type    mo:MusicArtist    mbz:ga);
(mbz:ga2   foaf:name   "Cookin's on 3 B."    mbz:ga);
(mbz:ga2   foaf:made   mbz:t1    mbz:ga);
(mbz:ga2   dbo:genre   mbz:q1    mbz:ga);
(mbz:ga2   dbo:genre   mbz:q2    mbz:ga);
(mbz:ga2   rdf:type   mo:MusicGroup    mbz:ga);
(mbz:ga3   rdf:type   mo:MusicArtist    mbz:ga);
(mbz:ga3   foaf:name   "Kylie Auldist"    mbz:ga);
(mbz:ga3   rdf:type   mo:SoloMusicArtist    mbz:ga);
(mbz:m1    rdf:type   mo:Record    mbz:gm);
(mbz:m1    mo:track_count  12   mbz:gm);
(mbz:m1    mo:track   mbz:t1    mbz:gm);
(mbz:m1    mo:track   mbz:t2    mbz:gm);
(mbz:ga1   rdf:type   mo:MusicArtist    mbz:ga);
(mbz:ga1   foaf:name   "Kungs"    mbz:ga);
(mbz:ga1   foaf:made   mbz:t1    mbz:ga);
(mbz:ga1   foaf:made   mbz:t2    mbz:ga) ;
(mbz:ga1   dbo:genre   mbz:q1    mbz:ga);
(mbz:ga1   dbo:genre   mbz:q2    mbz:ga);
(mbz:ga1   rdf:type   mo:SoloMusicArtist    mbz:ga) }
```

**Fig. 8.** $\Delta^+(u)$ for update $u$ in example 4.

Fig. 9. LinkedBrainz Live tool.

206x57mm (96 x 96 DPI)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
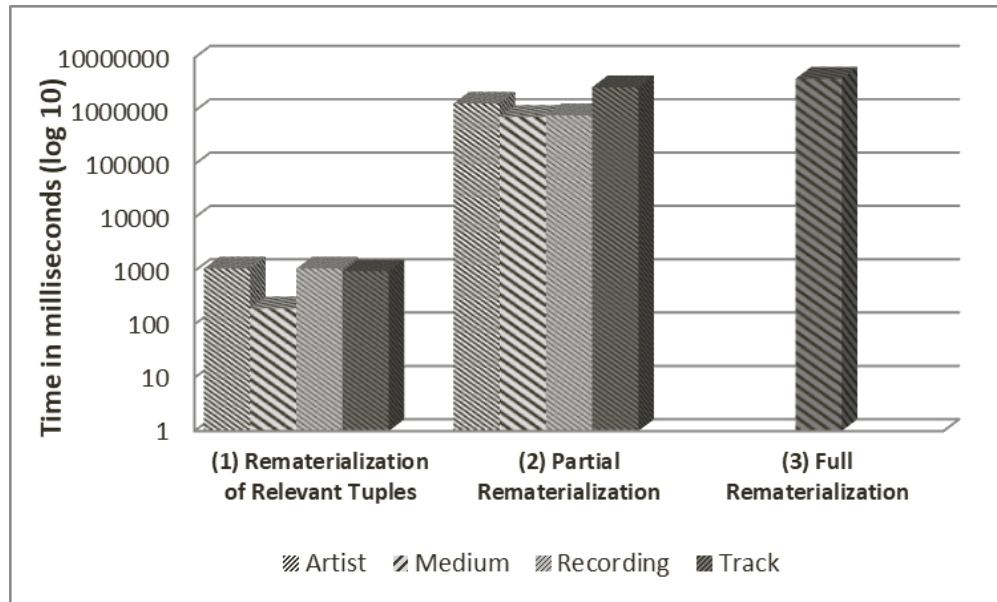44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60



Fig. 10. Comparison of the three rematerialization approaches.
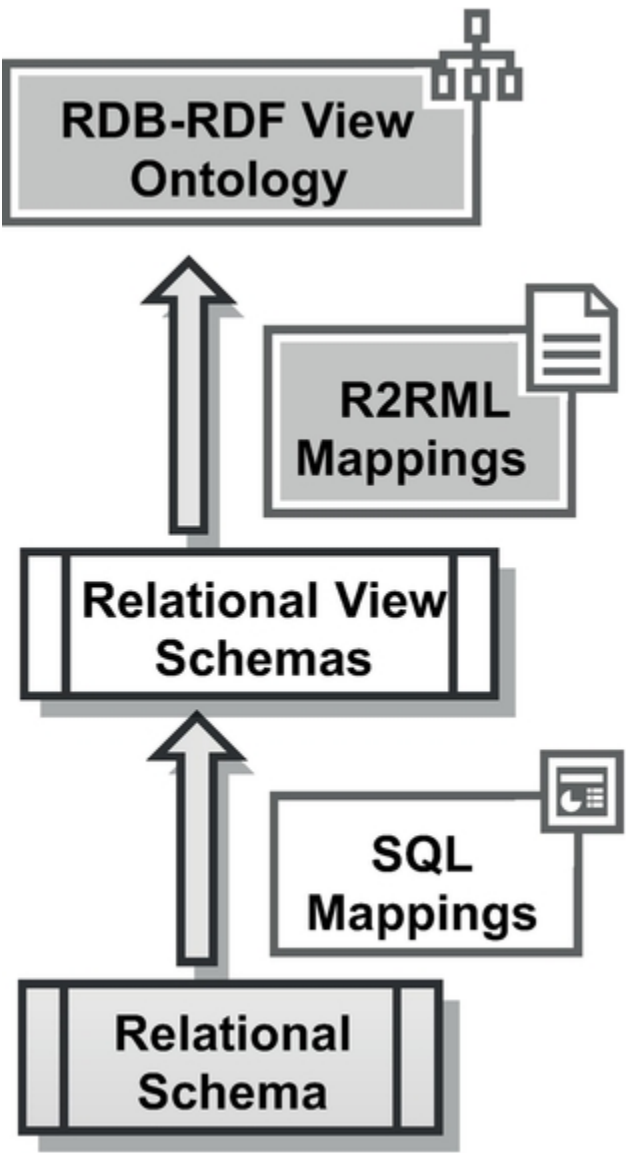
502x302mm (38 x 38 DPI)
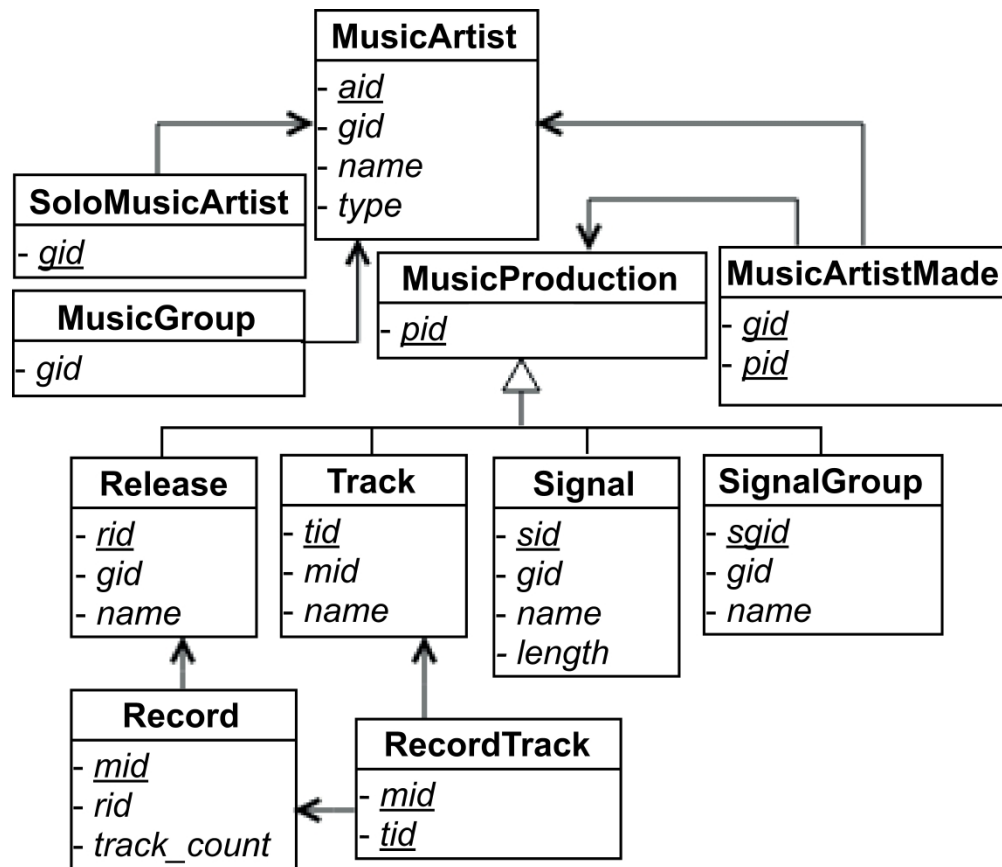
Fig. 11. Three-Level Schema.

52x97mm (150 x 150 DPI)

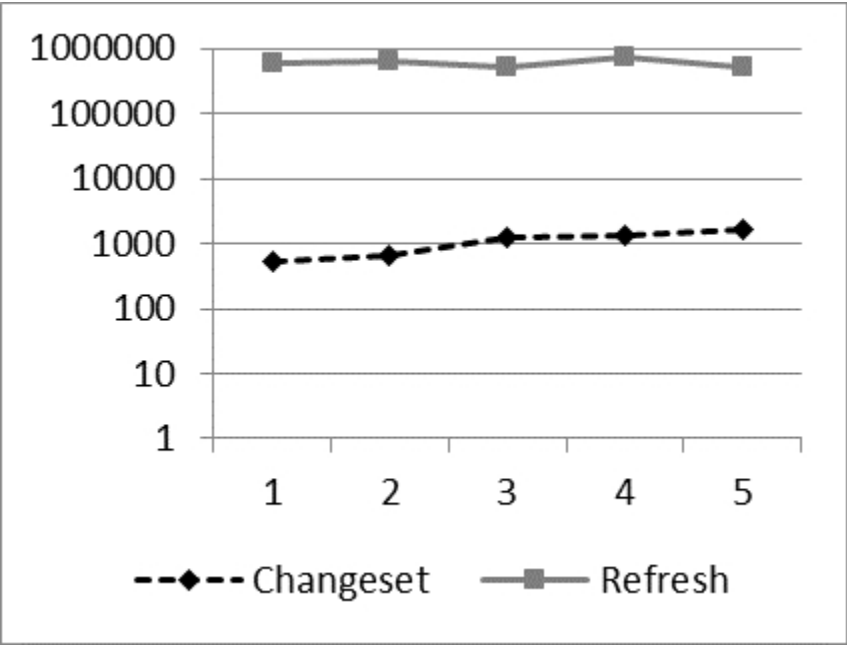Fig. 12. Relational View Schema.

537x462mm (600 x 600 DPI)

Fig. 13. (a) Updates on relation Artist.
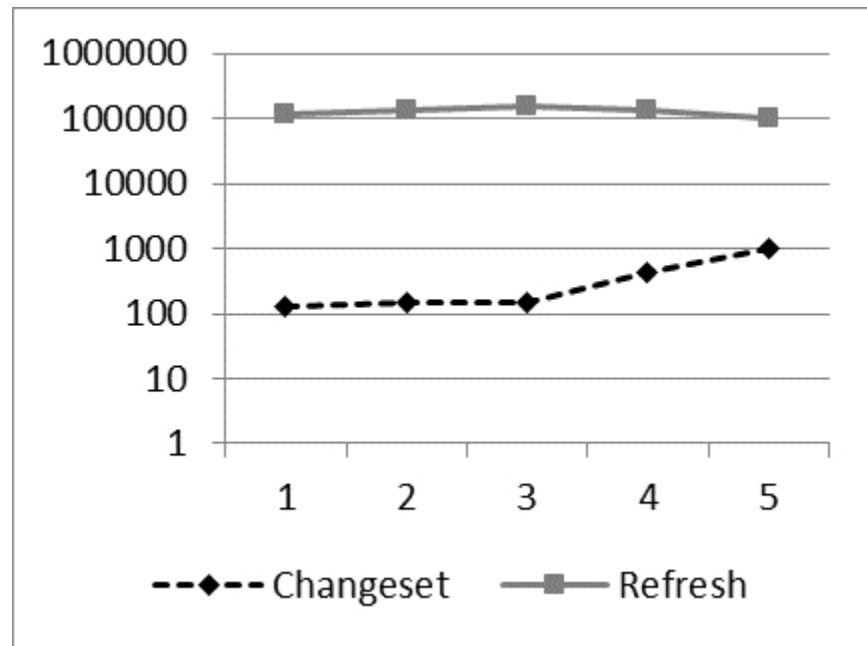
182x138mm (59 x 59 DPI)

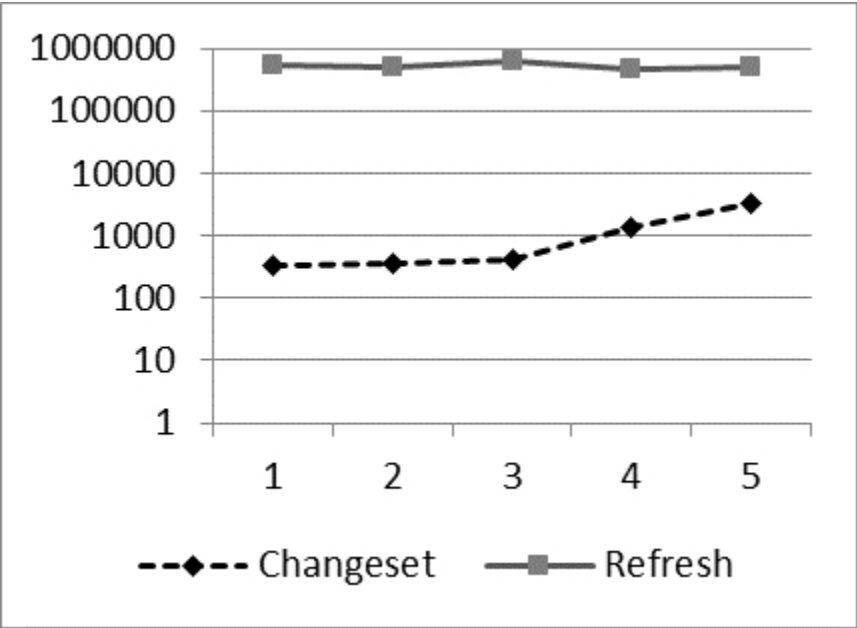Fig. 13. (b) Updates on relation Medium.

185x138mm (59 x 59 DPI)

Fig. 13. (c) Updates on relation Recoding.
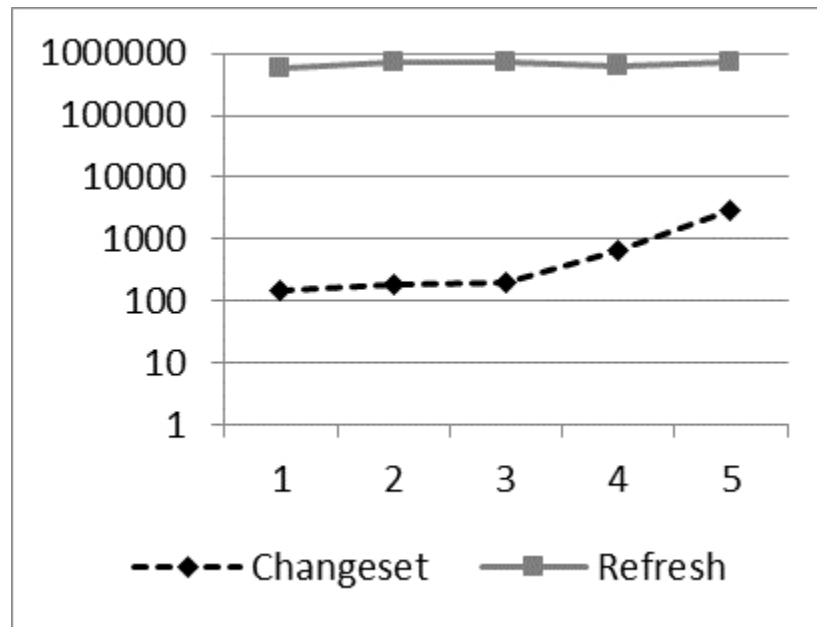
184x134mm (59 x 59 DPI)
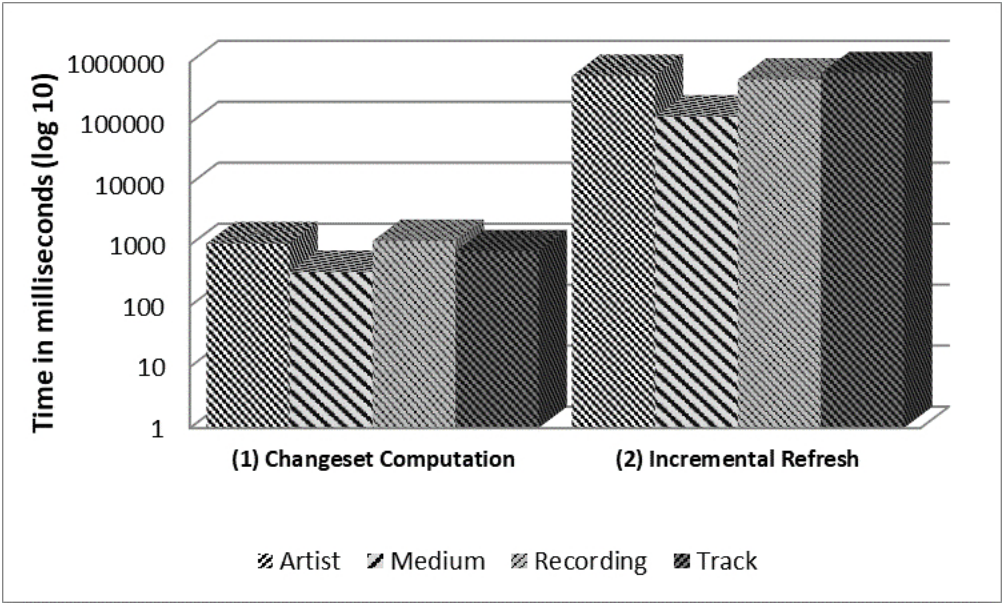
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60



Fig. 13. (d) Updates on relation Track.

176x133mm (59 x 59 DPI)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60



Fig. 14. Comparison of Changeset Computation for RDB2RDF view and Incremental Refresh.

322x193mm (59 x 59 DPI)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40

$$M(\sigma_0) \xrightarrow{\langle \Delta^-(u), \Delta^+(u) \rangle} (M(\sigma_0) - \Delta^-(u)) \cup \Delta^+(u)$$

$$\overset{=?}{M(\sigma_1)}$$

$$M \uparrow \qquad \uparrow u \qquad \qquad M \uparrow$$

$$\sigma_0 \longrightarrow \sigma_1$$

41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60