Using Complex Correspondences for Integrating Relational Data Sources

Valéria Pequeno[1(✉)], Helena Galhardas[2], and Vânia M. Ponte Vidal[3]

[1] INESC-ID, Taguspark, Oeiras, Portugal
vmp@inesc-id.pt
[2] Instituto Superior Técnico, Universidade de Lisboa and INESC-ID, Taguspark, Oeiras, Portugal
helena.galhardas@tecnico.ulisboa.pt
[3] Universidade Federal do Ceará, Fortaleza, Brazil
vvidal@lia.ufc.br

Abstract. Data Integration (DI) is the problem of combining a set of heterogeneous, autonomous data sources and providing the user with a unified view of these data. Integrating data raises several challenges, since the designer usually encounters incompatible data models characterized by differences in structure and semantics. One of the hardest challenges is to define correspondences between schema elements (e.g., attributes) to determine how they relate to each other. Since most business data is currently stored in relational databases, here present a declarative and formal approach to specify 1-to-1, 1-m, and m-to-n correspondences between relational schema components. Differently from usual approaches, our (CAs) have semantics and can deal with outer-joins and data-metadata relationships. Finally, we demonstrate how to use the CAs to generate mapping expressions in the form of SQL queries, and we present some preliminary tests to verify the performance of the generated queries.

Keywords: Schema matching · Correspondence assertions · Data integration · Relational model

1 Introduction

A DI system aims at integrating a variety of data obtained from different data sources, usually autonomous and heterogeneous, and providing a unified view of these data, often using an integrated schema. The integrated schema makes a bridge between the data sources and the applications that access the DI system. Data in a DI system can be physically reconciled in a repository (*materialized data integration approach*), or can remain at data sources and is only consolidated when a query is posed to the DI system (*virtual data integration approach*). A data warehouse system [1] is a typical example of the first approach. As examples of the second approach, we can cite federated information systems [2] and mediator systems [3]. In the present work, both scenarios can be used, but in this paper we will focus on the materialized integration approach.

J. Cordeiro et al. (Eds.): ICEIS 2014, LNBIP 227, pp. 57–74, 2015.
DOI: 10.1007/978-3-319-22348-3_4

One of the hardest problems to solve in DI is to define mappings between the integrated schema (the target) and each data source schema, known as *the schema mapping problem*. It consists of two main tasks: *i) schema matching* to define/generate correspondences (a.k.a. matches) between schema elements (e.g., attributes, relation, XML tags, etc.); and *ii) schema mapping* to find data transformations that, given data instances of a source schema, obtain data instances of the target schema.

The result of schema matching is a set of correspondences that relate elements of a source schema to elements of the target schema, where an element can be a relation name or attribute in the relational model. These correspondences can be described using a Local-as-view (LAV), a Global-as-view (GAV), or a Global and Local-asview (GLAV) language. In summary, in a LAV approach, each data source is described as a view over the integrated schema. In a GAV approach, the integrated schema is expressed as a view over the data sources. Finally, the GLAV combines the expressive power of both GAV and LAV. Once the schema matching is performed, the correspondences are used to generate the schema mappings. For example, a schema mapping can be codified through an SQL query that transforms data from the source into data that can be stored in the target.

Extensive research on schema matching has been carried out in recent years [4,5]. The majority of the works on this subject identifies 1-1 correspondences between elements of two schemas. For example, a 1-1 correspondence can specify that element **title** in one schema matches element **film** in another schema, or that relation GENRE matches relation CATEGORY[1]. This kind of schema matching is known in the literature as *basic matching*. Good surveys can be found in [6,7].

While basic matching is common, it leaves out numerous correspondences of practical interest, in particular when we consider DI systems. Thus, more complex matches are necessary. A complex matching specifies 1:n, m:n, or n:1 correspondences between elements of two schemas. For example, it may specify that **totalPrice** corresponds to **unitPrice** * **quantity**; or that **name** matches *concatenate*(**firstName, lastName**), where *concatenate* is a function that applies to two strings and returns a concatenated string; or even that the average departmental salary **avgWage** corresponds to grouping the salaries (**salary**) of all employees (**emp**) by department (**dept**). Works in [8,9] are examples of approaches that deals with complex matches.

Some researchers go beyond dealing with complex matches and add semantics to the correspondences to improve the overall matching quality. In the Sect. 2, we explain more about complex matches and show a motivation example. The remainder of the paper is structured as follows. In Sect. 3, we present the necessary background in Correspondence Assertions (CAs), the formalism used in this work to specify correspondences between elements of schemas. In Sect. 4, we propose new CAs to deal with join operators and metadata. Section 5 shows how to

[1] We use **bold** to represent attribute names and UPPERCASE to represent relation names.

generate mapping expressions from CAs. Section 6 shows some preliminary tests to evaluate our approach. Section 7 describes the related work. Finally, Sect. 8 concludes and describes future work.

2 Motivating Example

Consider a motivating example with the source schemas $\mathbf{S}_1$ and $\mathbf{S}_2$ in Fig. 1, which contain information about movies. $\mathbf{S}_1$ keeps a catalog of movies with information about different types of media (dvd, blue rays, etc.) in which the movies are available. The names of the relations and attributes are mostly self-explanatory. Some non-self-explanatory attributes in $\mathbf{S}_1$ have the following meaning: **id** is the movie identifier, **year** is the year of a movie, **film** is the title of a movie, **number** is the tape identifier, **name** can be a producer or a director name, and **role** can be *producer* or *director*. FK1 and FK2 are foreign keys. We use the notation FK(R:L, S:K) to denote a foreign key, named FK, where R and S are relation names and L and K are list of attributes from R and S, respectively, with the same length. FK1 is the foreign key of TAPE that refers to MOVIE and FK2 is the foreign key of MOVIEMAKERS that refers to MOVIE. $\mathbf{S}_2$ stores general information about movies and the places (in different cities) where movies are being shown. We assume that $\mathbf{S}_1$ can store older movies than $\mathbf{S}_2$. Some non-self-explanatory attributes in $\mathbf{S}_2$ have the following meaning: **rate** is the classification of the movie with regard the audience, **location** and **city** are, respectively, the cinema and the name of the city where the movie is shown, and **time** is the date when the movie is shown.

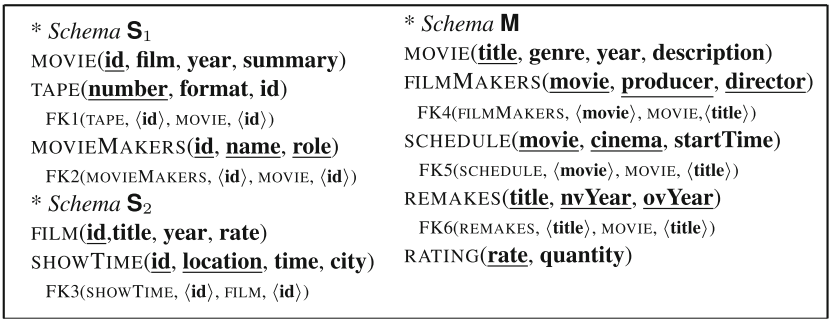

Fig. 1. Example of source schemas and a integrated schema.

The integrated schema **M**, also shown in Fig. 1, provides a unified user view of movies currently shown in cinemas of Lisbon. It is populated by information from schemas $\mathbf{S}_1$ and $\mathbf{S}_2$. The relation MOVIE stores movies shown currently at a cinema. The relation FILMMAKERS keeps information about professionals of show businesses. The relation SCHEDULE contains information about the schedule of movies shown in Lisbon. The relation REMAKES keeps the years of movies for

which there is at least one remake. The relation RATING stores the classification of movies with regard to suitability audience. Some non-self-explanatory attributes in **M** have the following meaning: **description** is the summary of a movie, **nvYear** is the year of the most recent version of a movie, **ovYear** is the year of the older versions of a movie, and **quantity** is the total of movies with the same rating.

Given the schemas $\mathbf{S}_1$, $\mathbf{S}_2$, and **M**, we can consider the correspondences between the source schemas $\mathbf{S}_1$ and $\mathbf{S}_2$, and the target schema **M**. As an example, we can state that **M**.SCHEDULE corresponds to $\mathbf{S}_2$.SHOWTIME, because both relations store information regarding the same real world concept[2]. However, in this correspondence, it is not clear that **M**.SCHEDULE only keeps schedules about movies shown in Lisbon. The additional information: **M**.SCHEDULE *corresponds to* $\mathbf{S}_2$.SHOWTIME *when* $\mathbf{S}_2$.SHOWTIME.**city** = "*Lisbon*", specifies better the matching.

The works reported in [10–12] and [5](chap. 3) propose schema matching approaches that can specify correspondences to deal with situations as required in the example The reader can see more proposals to add semantics to schema matching in [8,10,12,13]. However, the following situations have not been fully covered yet:

1. *Correspondences Between Relations Involving Join Conditions Other than Equality of Attributes.* Consider the relation **M**.REMAKES that keeps a list of remakes with the years of the oldest versions. Knowing that $\mathbf{S}_2$.FILM keeps current movies and $\mathbf{S}_1$.MOVIE may contain older versions of the same movie, we want to indicate which of the current movies are remakes and store this information in **M**.REMAKES. The correspondence between these relations can be specified as: **M**.REMAKES *corresponds to* $\mathbf{S}_2$.FILM *join* $\mathbf{S}_1$.MOVIE *where* $\mathbf{S}_2$.FILM.**title** = $\mathbf{S}_1$.MOVIE.**film** *and* $\mathbf{S}_2$.FILM.**year** > $\mathbf{S}_1$.MOVIE.**year**. Usual schema matching approaches cannot specify this correspondence, because join conditions are not explicitly defined in schema matching. Moreover, join paths are normally automatically discovered in the schema mapping phase [14], and the algorithms used can only find equi-join conditions, so they cannot automatically discover the condition $\mathbf{S}_2$.FILM.**year** > $\mathbf{S}_1$.MOVIE.**year**. Hence, we need a schema matching approach that makes it possible to specify the join between relations and allows general join conditions containing operators different from equality.
2. *Correspondences Between Relations Involving Outer-joins (Full, Left, or Right).* We want to indicate how **M**.MOVIE is related to source schemas $\mathbf{S}_1$ and $\mathbf{S}_2$. **M**.MOVIE and $\mathbf{S}_2$.FILM represent the same concept of the real world (i.e., both relations store current movies shown at some cinema). However, it is not enough to specify that **M**.MOVIE matches $\mathbf{S}_2$.FILM, because there are attributes in $\mathbf{S}_1$.MOVIE (namely, **category** and **summary**) that contain information required in the schema of **M**.MOVIE. Hence, we should specify that

[2] We use a path representation: an attribute **A** of a given relation R in a given database schema **D** is referred to as **D**.R.**A**. For simplicity, we omit the database schema when the context is clear.

M.MOVIE is related to both $\mathbf{S}_1$.MOVIE and $\mathbf{S}_2$.FILM. However, it is not correct we simply match **M**.MOVIE to $\mathbf{S}_1$.MOVIE because $\mathbf{S}_1$.MOVIE can store movies that are not being shown in a cinema anymore and **M**.MOVIE can store recent movies that are not available in dvds yet. In summary, we should specify that: **M**.MOVIE *corresponds to* $\mathbf{S}_2$.FILM *left outer-join* $\mathbf{S}_1$.MOVIE *on* $\mathbf{S}_2$.FILM.**title** = $\mathbf{S}_1$.MOVIE.**film** *and* $\mathbf{S}_2$.FILM.**year** = $\mathbf{S}_1$.MOVIE.**year**. Note that the condition $\mathbf{S}_2$.FILM.**title** = $\mathbf{S}_1$.MOVIE.**film** and $\mathbf{S}_2$.FILM.**year** = $\mathbf{S}_1$.MOVIE.**year** guarantees that we refer to a same movie stored in both $\mathbf{S}_1$.MOVIE and $\mathbf{S}_2$.FILM. Again, we cannot specify this type of correspondence since joins (and their variants) are not explicitly defined in current schema matching approaches.

3. *Correspondences Between Data and Metadata.* Consider the relations $\mathbf{S}_1$.MOVIEMAKERS and **M**.FILMMAKERS. Both keep information about the relationship between a movie, a producer, and a director. We want to indicate that **M**.FILMMAKERS corresponds to $\mathbf{S}_1$.MOVIEMAKERS since they represent the same concept in the real world. In addition, we want to specify the correspondences between the attributes of these relations. Knowing that $\mathbf{S}_1$.MOVIEMAKERS.**name** can be a producer name or a director name, we would like to specify that **M**.FILMMAKERS.**producer** corresponds to $\mathbf{S}_1$.MOVIEMAKERS.**name** when $\mathbf{S}_1$.MOVIEMAKERS.**role** = "producer" and that **M**.FILMMAKERS.**director** corresponds to $\mathbf{S}_1$.MOVIEMAKERS.**name** when $\mathbf{S}_1$.MOVIEMAKERS.**role** = "director". However, we cannot specify these correspondences using traditional schema matching approaches, because these correspondences involve semantics not covered yet by these approaches. Actually, we can only specify that **M**.FILMMAKERS.**producer** matches to $\mathbf{S}_1$.MOVIEMAKERS.**name** and **M**.FILMMAKERS.**director** matches to $\mathbf{S}_1$.MOVIEMAKERS.**name**.

In order to deal with these situations, we propose to use a formalism based on CAs [10,15]. Using CAs, we can declaratively specify basic and complex matchings with semantics. We propose to adapt CAs to be able to express schema matching between relational schemas, as well as to extend this formalism with new types of CAs to deal with joins, outer-joins, and data-metadata relationships. Finally, we demonstrate how mapping expressions in the form of SQL queries can be generated from CAs.

3 Background

In this section, we present the basic terminology used in this paper. We also review the different classes of CAs, and adapt them to the Relational Data Model (RDM).

3.1 Basic Concept and Notation

We assume that the reader is familiar with the relational concepts. We denote a relation schema as R($\mathcal{A}_1$, $\mathcal{A}_2$, ..., $\mathcal{A}_n$), and a foreign key as FK(R:L, S:K). We say that FK relates R and S.

A relational schema is a pair $\mathbf{S}= (\mathcal{R}, \Omega)$, where $\mathcal{R}$ is a set of relation schemas and Ω is a set of relational constraints such that: (i) Ω has a unique primary key for each relation schema in $\mathcal{R}$; (ii) if Ω has a foreign key of the form FK(R:L, S: K), then Ω also has a constraint indicating that K is the primary key of S. Given a relation schema $\mathrm{R}(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$ and a tuple variable $\mathbf{t}$ over R, we use $\mathbf{t}[\mathcal{A}_i]$ to denote the projection of $\mathbf{t}$ over $\mathcal{A}_i$.

Let $\mathbf{S}= (\mathcal{R}, \Omega)$ be a relational schema and R and T be relation names of relation schemas in $\mathcal{R}$. We denote $\varrho = \mathrm{FK}_1 \bullet \mathrm{FK}_2 \bullet \cdots \bullet \mathrm{FK}_{n-1}$ a path from R to T iff there is a list $\mathrm{R}_1, \ldots, \mathrm{R}_n$ of relation schemas in $\mathbf{S}$ such that $\mathrm{R}_1 = \mathrm{R}$, $\mathrm{R}_n = \mathrm{T}$, and FK_i relates R_i and R_{i+1}. We say that tuples of R reference tuples of T through ϱ.

3.2 Correspondence Assertions

We use Correspondence Assertions (CAs) in order to express schema matchings between schema elements. CAs are formal expressions of the general form ψ: $\mathcal{T} \leftarrow \mathcal{S}$, where ψ is the name of the CA, $\mathcal{T}$ is an expression formed by elements of the target schema, and $\mathcal{S}$ is an expression formed by elements of a source schema. The symbol "$\leftarrow$" means "is matched from".

In accordance to [10], there are four types of CAs: Relation Correspondence Assertion (RCA), Attribute Correspondence Assertion (ACA), Summation Correspondence Assertion (SCA), and Grouping Correspondence Assertion (GCA). RCAs and SCAs specify the relationship between relations of distinct schemas, while ACAs and GCAs specify the relationship between attributes of relations of distinct schemas. We now shortly describe each type of CA, adapting them to the RDM. In the remainder of this Section, consider: $\mathbf{S}_i= (\mathcal{R}_i, \Omega_i)$ be relational schemas for $1 \leq i \leq n$, with R_i being relation names of relation schemas in $\mathcal{R}_i$.

Definition 1. *Let σ be a selection over R_2. A Relation Correspondence Assertion RCA is an expression of one of the following forms:*

1. ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2]$
2. ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2\sigma]$
3. ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2] - \mathbf{S}_3[\mathrm{R}_3]$
4. ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \bigcup_{i=1}^{n} \mathbf{S}_i[\mathrm{R}_i]$
5. ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \bigcap_{i=1}^{n} \mathbf{S}_i[\mathrm{R}_i]$ □

In Definition 1, we say that ψ matches R_1 and R_i, for $1 \leq i \leq n$. RCAs express the various kinds of semantic equivalent relationships. Two relations R_1 and R_2 are semantically equivalent if they represent the same real concept and there is a 1-to-1 correspondence between their instances. ψ_1, shown in Fig. 2, is an example of a RCA.

ψ_1 specifies that $\mathbf{M}$.SCHEDULE is semantically equivalent to $\mathbf{S}_2$.SHOWTIME when the condition $\mathbf{S}_2$.SHOWTIME.**city** = "Lisbon" is satisfied. This means that only a subset of tuples of $\mathbf{S}_2$.SHOWTIME, those that satisfy the condition are involved in the match.

Before we define an Attribute Correspondence Assertion (ACA), we need introduce the concept of *attribute expression*, as follows:

ψ_1:**M**[SCHEDULE]$\leftarrow$**S**$_2$[SHOWTIME(**city** = "Lisbon")]	RCA
ψ_2:**M**[SCHEDULE]$\bullet$**movie**$\leftarrow$**S**$_2$[SHOWTIME]$\bullet$FK$_3$/**title**	ACA
ψ_3:**M**[RATING]$\Leftarrow$groupby(**S**$_2$[FILM] (**rate**))	SCA
ψ_4:**M**[RATING] $\bullet$ **quantity**$\Leftarrow$ count(**S**$_2$[FILM] $\bullet$ **rate**)	GCA

Fig. 2. Examples of correspondence assertions.

Definition 2. *Let* R_2 *and* T *be relation names in* $\mathcal{R}_2$*, with* $\mathcal{A}$ *being an attribute of* R_2 *and* $\mathcal{B}$ *an attribute of* T*. Let also* ϱ *be a path from* R_2 *to* T*. An attribute expression* $\mathcal{E}$ *over* R_2 *is an expression with one of the following forms:*

1. $\mathbf{S}_2[\mathrm{R}_2] \bullet \mathcal{A}$
2. $\mathbf{S}_2[\mathrm{R}_2] \bullet \varrho/\mathcal{B}$. □

Definition 3. *Let* $\mathcal{A}_i$ *be attributes of* R_1 *(for* $1 \leq i \leq n$*). Let also* $\mathcal{E}_j$*, for* $1 \leq j \leq m$*, be attribute expressions over* R_2*. An Attribute Correspondence Assertion (ACA) is an expression of one of the following forms:*

1. ψ: $\mathbf{S}_1[\mathrm{R}_1] \bullet \mathcal{A}_1 \leftarrow \mathcal{E}_1$
2. ψ: $\mathbf{S}_1[\mathrm{R}_1] \bullet \mathcal{A}_1 \leftarrow \varphi(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m)$
3. ψ: $\mathbf{S}_1[\mathrm{R}_1] \bullet \mathcal{A}_1 \leftarrow (\mathcal{E}_1, \mathbf{p}_1); \ldots; (\mathcal{E}_m; \mathbf{p}_m); \boldsymbol{v}$
4. ψ: $\mathbf{S}_1[\mathrm{R}_1](\mathcal{A}_1, \ldots, \mathcal{A}_n) \leftarrow (\mathcal{E}_1, \ldots, \mathcal{E}_n)$

Where φ *is a function over attributes of* R_2*,* $\mathbf{p}_j$ *(for* $1 \leq j \leq m$*) are boolean conditions over attributes of* R_2*, and* $\boldsymbol{v}$ *is a value. We say that* ψ *matches* R_1 *and* R_2*.* □

ACAs specify the relationship between the attributes of relations that are matched by a RCA. They allow to define 1:1, 1:n, n:1, or m:n relationships between attributes of relations of different schemas. For example see the ACA ψ_2 presented in Fig. 2. It specifies the correspondence between **M**.SCHEDULE.**movie** and **S**$_2$.FILM.**title** through a path from SHOWTIME to FILM.

Definition 4. *Let* σ *be a selection over* R_2*. Let also* $\mathcal{A}'_i$ *attributes of* R_2 *(for* $1 \leq i \leq m$*). A Summation Correspondence Assertion (SCA) is an expression of one of the following forms:*

1. ψ: $\mathbf{S}_1[\mathrm{R}_1] \Leftarrow \mathit{groupby}(\mathbf{S}_2[\mathrm{R}_2](\mathcal{A}'_1, \mathcal{A}'_2, \ldots, \mathcal{A}'_m))$
2. ψ: $\mathbf{S}_1[\mathrm{R}_1] \Leftarrow \mathit{groupby}(\mathbf{S}_2[\mathrm{R}_2\sigma](\mathcal{A}'_1, \mathcal{A}'_2, \ldots, \mathcal{A}'_m))$
3. ψ: $\mathbf{S}_1[\mathrm{R}_1] \Leftarrow \mathit{normalise}(\mathbf{S}_2[\mathrm{R}_2](\mathcal{A}'_1, \mathcal{A}'_2, \ldots, \mathcal{A}'_m))$
4. ψ: $\mathbf{S}_1[\mathrm{R}_1] \Leftarrow \mathit{normalise}(\mathbf{S}_2[\mathrm{R}_2\sigma](\mathcal{A}'_1, \mathcal{A}'_2, \ldots, \mathcal{A}'_m))$ □

In Definition 4, we say that ψ matches R_1 and R_2. SCAS specify 1:n, n:1, m:n relationships between relations with distinct schemas. Here we use the symbol "$\Leftarrow$" instead of "$\leftarrow$" in order to emphasize that the correspondence is not 1:1 as is usual in the most part of schema matching approaches. SCAS are used to describe the summary of a relation whose tuples are related to the tuples of another relation by gathering them into logical groups. This means that a SCA has only the necessary information to indicate which grouping field is involved in

the relationship and the process used to grouping the tuples. ψ_3 shown in Fig. 2 is a simple example of a SCA.

GCAs specify the relationship 1:1, 1:n, n:1, or m:n between attributes of relations that are matched by a SCA.

Definition 5. *Let $\mathcal{A}$ be an attribute of R_1. Let also $\mathcal{E}_i$, for $1 \leq i \leq m$, be attribute expressions over R_2. A Grouping Correspondence Assertion (GCA) is an expression of one of the following forms:*

1. ψ: $\mathbf{S}_1[\mathrm{R}_1] \bullet \mathcal{A} \Leftarrow \mathcal{E}_1$
2. ψ: $\mathbf{S}_1[\mathrm{R}_1] \bullet \mathcal{A} \Leftarrow \varphi(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m)$
3. ψ:$\mathbf{S}_1[\mathrm{R}_1] \bullet \mathcal{A} \Leftarrow (\mathcal{E}_1, \mathbf{p}_1); \ldots; (\mathcal{E}_m; \mathbf{p}_m); \boldsymbol{v}$
4. ψ: $\mathbf{S}_1[\mathrm{R}_1] \bullet \mathcal{A} \Leftarrow \gamma(\mathcal{E}_1)$.
5. ψ: $\mathbf{S}_1[\mathrm{R}_1] \bullet \mathcal{A} \Leftarrow \gamma(\varphi(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m))$
6. ψ: $\mathbf{S}_1[\mathrm{R}_1] \bullet \mathcal{A} \Leftarrow \gamma(\mathcal{E}_1, \mathbf{p})$.
7. ψ: $\mathbf{S}_1[\mathrm{R}_1] \bullet \mathcal{A} \Leftarrow \gamma(\varphi(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m), \mathbf{p})$

Where φ is a function over attributes of R_2, $\mathbf{p}_j$ (for $1 \leq j \leq m$) are boolean conditions over attributes of R_2, $\boldsymbol{v}$ is a value, and γ is one of the aggregate functions: sum (summation), max (maximum), min (minimum), avg (average), or count. We say that ψ matches R_1 and R_2. □

Consider the relations $\mathbf{S}_2$.FILM and $\mathbf{M}$.RATING. ψ_4, represented in Fig. 2, specifies that $\mathbf{M}$.RATING.**quantity** corresponds to the counting of all distinct values of $\mathbf{S}_2$.FILM.**rate**.

Definition 6. *Let $\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_n$ and $\mathbf{T}$ be relational schemas; R_1 be a relation schema of $\mathbf{T}$, and R_2 a relation schema of some $\mathbf{S}_i$, $1 \leq i \leq n$. Let also $\mathcal{E}_j$ (for $1 \leq j \leq m$) be expressions as defined in Definition 2. A schema matching between schemas $\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_n$ and the schema $\mathbf{T}$ is a set $\mathcal{M}$ of CAs such that:*

1. *if $\mathcal{M}$ has an ACA ψ such that ψ matches R_1 and R_2, then $\mathcal{M}$ has a RCA ψ' that matches R_1 and R_2.*
2. *if $\mathcal{M}$ has a GCA ψ such that ψ matches R_1 and R_2, then $\mathcal{M}$ has a SCA ψ' that matches R_1 and R_2.*
3. *if $\mathcal{M}$ has a RCA ψ such that ψ matches R_1 and R_2, then $\mathcal{M}$ has an ACA ψ': $\mathbf{S}_1[\mathrm{R}_1](\mathcal{A}_1, \ldots, \mathcal{A}_n) \leftarrow (\mathcal{E}_1, \ldots, \mathcal{E}_n)$ that matches R_1 and R_2.* □

4 Specifying New CAs

In Sect. 1, we identified the following types of relationships between schemas elements that are not properly handled in current schema matching approaches: (1) matches involving explicit join conditions; (2) matches involving outer-joins; and (3) matches involving data-metadata. Join (and outer-join) relationships can express one-to-one or many-to-many correspondences between the relations involved. Matches involving data-metadata can express many-to-many correspondences between the relations involved. So, we extend our previous definitions of RCA and SCA in order to better specify these types of matchings. In the following text consider $\mathbf{S}_i$ relational schemas, R_i relation schemes of $\mathbf{S}_i$ (for $1 \leq i \leq 3$), θ a join condition between R_2 and R_3, and $\mathcal{A}_j$ attributes of R_2 (for $1 \leq j \leq n$)

Definition 7. *A Relation Correspondence Assertion (RCA) is an expression of one of the following forms:*

1. *Expressions shown in Definition 1*
2. ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2] \bowtie \mathbf{S}_3[\mathrm{R}_3]\theta$
3. ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2] \sqsupset\!\bowtie \mathbf{S}_3[\mathrm{R}_3]\theta$
4. ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2] \bowtie\!\sqsubset \mathbf{S}_3[\mathrm{R}_3]\theta$
5. ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2] \sqsupset\!\bowtie\!\sqsubset \mathbf{S}_3[\mathrm{R}_3]\theta$

□

Definition 8. *A Summation Correspondence Assertion (SCA) is an expression of one of the following forms:*

1. *Expressions shown in Definition 4*
2. ψ: $\mathbf{S}_1[\mathrm{R}_1] \Leftarrow \mathbf{S}_2[\mathrm{R}_2] \bowtie \mathbf{S}_3[\mathrm{R}_3]\theta$
3. ψ: $\mathbf{S}_1[\mathrm{R}_1] \Leftarrow \mathbf{S}_2[\mathrm{R}_2] \sqsupset\!\bowtie \mathbf{S}_3[\mathrm{R}_3]\theta$
4. ψ: $\mathbf{S}_1[\mathrm{R}_1] \Leftarrow \mathbf{S}_2[\mathrm{R}_2] \bowtie\!\sqsubset \mathbf{S}_3[\mathrm{R}_3]\theta$
5. ψ: $\mathbf{S}_1[\mathrm{R}_1] \Leftarrow \mathbf{S}_2[\mathrm{R}_2] \sqsupset\!\bowtie\!\sqsubset \mathbf{S}_3[\mathrm{R}_3]\theta$
6. ψ:$\mathbf{S}_1[\mathrm{R}_1] \Leftarrow metadata(\mathbf{S}_2[\mathrm{R}_2](\mathcal{A}_1,\ldots,\mathcal{A}_n))$ □

Consider the three examples about join, outer-join, and data-metadata correspondences described in Sect. 1. The correspondence between **M**.REMAKES and both $\mathbf{S}_1$.MOVIES and $\mathbf{S}_2$.FILM can be specified by the SCA ψ_5 shown in Fig. 3. ψ_5 specifies that **M**.REMAKES corresponds to a join between $\mathbf{S}_1$.MOVIES and $\mathbf{S}_2$.FILM where the join condition: $\mathbf{S}_2$.FILM.**title**= $\mathbf{S}_1$.MOVIE.**film** and $\mathbf{S}_2$.FILM.**year** > $\mathbf{S}_1$.MOVIE.**year** is satisfied.

ψ_5:**M**[REMAKES]$\Leftarrow$$\mathbf{S}_2$[FILM]$\bowtie$$\mathbf{S}_1$[MOVIE]$\theta$, where θ= ($\mathbf{S}_2$[FILM].**title**=$\mathbf{S}_1$[MOVIE].**film** and $\mathbf{S}_2$[FILM].**year** > $\mathbf{S}_1$[MOVIE].**year**
ψ_6:**M**[MOVIE]$\leftarrow$$\mathbf{S}_2$[FILM]$\sqsupset\!\bowtie$$\mathbf{S}_1$[MOVIE]$\theta'$, where θ' = ($\mathbf{S}_2$[FILM].**title**=$\mathbf{S}_1$[MOVIE].**film** and $\mathbf{S}_2$[FILM].**year** = $\mathbf{S}_1$[MOVIE].**year**
ψ_7:**M**[FILMMAKERS] $\Leftarrow$ *metadata* ($\mathbf{S}_1$[MOVIEMAKERS] (**id**))

Fig. 3. Examples of CAs involving joins, outer-joins and data-metadata.

The correspondence between **M**.MOVIE and both $\mathbf{S}_2$.FILM and $\mathbf{S}_1$.MOVIE can be specified by the RCA ψ_6, shown in Fig. 3. ψ_6 specifies that **M**.MOVIE corresponds to a left outer-join between $\mathbf{S}_2$.FILM and $\mathbf{S}_1$.MOVIE.

The correspondence between **M**.FILMMAKERS and $\mathbf{S}_1$.MOVIEMAKERS can be specified by the SCA ψ_7, shown in Fig. 3. ψ_7 specifies that **M**.FILMMAKERS corresponds to grouping $\mathbf{S}_1$.MOVIEMAKERS by the attribute **id**, being that a data-metadata translation should be performed (i.e., some data should be converted into metadata).

Once the schema matching is finished, the CAs generated can be used, for example, to generate mapping expressions that convert data sources into data target. We propose that the mapping expressions are automatically generated in the form of SQL queries, which are used to load the relations (the materialized views) of the integrated schema.

5 From CAs to Mapping Expressions

In our proposal, the process to create queries to transform data from a schema to another one consists of three steps:

1. Indicate the source schemas and the integrated schema using a high-level data model. In our case, we use the RDM.
2. Define the CA that formally specify the relationships between the integrated schema and the source schemas.
3. Generate a set of queries based on the CAs generated in step 2, in order to populate the relations of the integrated schema.

In order to illustrate our approach, consider the integrated schema $\mathbf{M}$ and the sources schemas $\mathbf{S}_1$ and $\mathbf{S}_2$ shown in Fig. 1.

Now, we should define CAs between $\mathbf{M}$ and $\mathbf{S}_1$, and CAs between $\mathbf{M}$ and $\mathbf{S}_2$. In our work, the CAs are specified using a GAV approach rather than a LAV one. One of reasons for our choices was due to the GAV approach makes the query answering easier than LAV one, both in materialized and in virtual integration approaches.

The process to generate the CAs consists of the following steps:

1. To each relation R^T of the target $\mathbf{T}$ do:
   (a) Identify the correspondences at a relation level (i.e., if there is a RCA or a SCA matching a target relation R^T and some source relation R^S).
   (b) Identify the correspondences at an attribute level: (1) identify the ACAs between the attributes of R^T and R^S (if there is a RCA between R^T and R^S); (2) identify the GCA between the attributes of R^T and R^S (if there is a SCA between R^T and R^S).
   (c) Determine which RCA and SCA can be combined to form a single CA.

In the current work, CAS were manually specified. However, we can use traditional schema matching tools (e.g.,[8,9]) as a starting point to find basic matchings. Then these basic matchings can be enriched through our formalism (using the CAs).

Some examples of RCAs, ACAs, SCAs, and GCAs between elements of $\mathbf{M}$ and the source schemas $\mathbf{S}_1$ and $\mathbf{S}_2$ can be found in Figs. 3 and 4.

The final step in the process of creating queries to transform data from a schema to another is the generation of the queries. In our proposal, they are defined based on the definition of the schemas and the CAs. Here we use SQL syntax of MySQL, since MySQL is an open source database that allows to combine the information from many databases in a single query. However, our CAs can be used to generate queries in any SQL syntax or even other federating queries languages as SchemaSQL [16].

Let $\mathcal{M}$ be a set of CAs that defines a matching between the source schemas $\mathbf{S}_1$, $\mathbf{S}_2$ and the integrated schema $\mathbf{G}$, that is, $\mathcal{M}$ satisfies the conditions stated in Definition 6. Algorithm 1 shows the procedure to automatically generate the statements of SQL queries from the CAs in $\mathcal{M}$.

ψ_8:**M**[MOVIE]•**title** ← **S**$_2$[FILM]• **title**
ψ_9:**M**[MOVIE]•**year** ← **S**$_2$[FILM]• **year**
ψ_{10}:**M**[MOVIE]•**genre** ← **S**$_1$[MOVIE]• **category**
ψ_{11}:**M**[MOVIE]•**description**←**S**$_1$[MOVIE]• **summary**
ψ_{12}:**M**[FILMMAKERS]•**producer**⇐(**S**$_1$[MOVIEMAKERS]• **name**, **S**$_1$[MOVIEMAKERS]• **role** = = "producer")
ψ_{13}:**M**[FILMMAKERS]•**director**⇐(**S**$_1$[MOVIEMAKERS]• **name**, **S**$_1$[MOVIEMAKERS]• **role** = = "director")
ψ_{14}:**M**[FILMMAKERS]• **movie** ⇐ **S**$_1$ [MOVIEMAKERS] • FK4/**film**

Fig. 4. Examples of ACAs and GCAs.

The Algorithm 1 generates a set of SQL queries, one for each relation schema R^T in the integrated schema. First it spans all ACAs and GCAs that relates attributes of R^T, and puts the correct value in lists **S**, **J**, and **LA**, in accordance to the type of the CA. **S** keeps the relation schemas that will be included in the FROM clause, **J** keeps the *join conditions* that will be included in the WHERE clause, and **LA** keeps the attributes that will be included in the SELECT clause. The procedure *G_SQL_ACA()*, shown in Algorithm 2, spans the ACAs, while the procedure *G_SQL_GCA()*, shown in Algorithm 3, spans the GCAs. After, the algorithm spans the RCAs and SCAs, of R^T, in order to create the SQL query to load R^T, using templates in Table 1. In accordance to type of CA besides **S**, **J**, and **LA**, other variables are needed to keep the *join conditions* that will be included in the ON clause (θ), the relation schema that will be included in (inner, outer, left, or right) JOIN clause (**RJ**), and the grouping attributes that will be included in the GROUP BY clause (**G**). Due to space limitations, Algorithms 1, 2, and 3, as well as the Table 1, do not cover the whole set of CAs as defined in Definitions 3, 5, 7, and 8.

In Algorithms 2, we assumed that $\varphi()$ is a pre-defined SQL function or a user-defined function on SQL. In Algorithms 3, **JAux**, and **Aux** are lists used when it is necessary to create temporary tables in SQL. This occurs when the SQL query is created from a SCA of metadata. **JAux** stores the *joins* that will be included in the WHERE clause of the temporary table, while **Aux** keeps the relation schema that will be the alias of the temporary table.

In Table 1, *Att*() is a function that returns the list of attribute names of a relation schema. We use the short word *outer join* to emulate a UNION of a LEFT JOIN and a RIGHT JOIN, since MySQL does not support directly full outer-joins.

The SQL queries generated by our algorithms can be used to compute the data target once, and to recompute them at pre-stablished times in order to maintain the target data up-to-date (this approach is named *rematerialization*). Generally, a more efficient approach is to periodically modify only part of the target data to reflect updates in data sources (this approach is named *incremental maintenance*). Rematerialization is adequate, for example, when the integrated schema is firstly populated, or in situations involving complex operations.

Algorithm 1. Generate the SQL query to load a integrated schema **G** from the sources.

for all relation schema $\mathbf{N} = \mathrm{R}^T$ in **G do**
  Let ψ_R be a RCA or SCA of R^T
  if ψ_R is a RCA **then** G_SQL_ACA(R^T,ψ_R)
  else G_SQL_GCA(R^T,ψ_R)
  append $[\mathrm{R}^T]$ to **S**
  switch ψ_R **do**
    case ψ_R: $\mathbf{G}[\mathrm{R}^T] \leftarrow \mathbf{S}[\mathrm{R}]$
      if J = [ ] **then** use template T1
      else use template T3
    case ψ_R: $\mathbf{G}[\mathrm{R}^T] \leftarrow \mathbf{S}_1[\mathrm{R}_1] \cup \mathbf{S}_2[\mathrm{R}_2]$
      append $[\mathrm{R}_2]$ to **RJ**
      θ keeps join conditions formed by primary key attributes of R_1 and R_2
      if J = [ ] **then** use template T7
      else use template T8
    case ψ_R: $\mathbf{G}[\mathrm{R}^T] \leftarrow \mathbf{S}_1[\mathrm{R}_1] \supset\!\bowtie \mathbf{S}_2[\mathrm{R}_2]\theta$
      append $[\mathrm{R}_2]$ to **RJ**
      if J = [ ] **then** use template T5
      else use template T6
    case ψ_R: $\mathbf{G}[\mathrm{R}^T] \Leftarrow metadata(\mathbf{S}[\mathrm{R}](\mathcal{A}))$
      append $[\mathcal{A}]$ to **AL**
      if J = [ ] **then** use template T13
      else use template T4
    case ψ_R: $\mathbf{G}[\mathrm{R}^T] \Leftarrow normalise(\mathbf{S}[\mathrm{R}](\mathcal{A}))$
      append $[\mathcal{A}]$ to **G**
      Use template T15
  end switch
end for

Algorithm 2. G_SQL_ACA().

Input: R^T, ψ_R

Let $\mathbf{S}[\mathrm{R}]$ be a source's relation in ψ_R
for all attribute $\mathcal{A}^T$ in R^T **do**
  while $\exists \psi_A / \psi_A$ is an ACA of $\mathcal{A}^T$ relating it to some atribute of $\mathbf{S}[\mathrm{R}]$ **do**
    if ψ_A: $\mathbf{G}[\mathrm{R}^T] \bullet \mathcal{A}^T \leftarrow \mathbf{S}[\mathrm{R}] \bullet \mathcal{A}_1$ **then** append $[\mathcal{A}_1]$ to **LA**
    if ψ_A: $\mathbf{G}[\mathrm{R}^T] \bullet \mathcal{A}^T \leftarrow \varphi(\mathbf{S}[\mathrm{R}] \bullet \mathcal{A}_1, \mathbf{S}[\mathrm{R}] \bullet \mathcal{A}_2, \ldots, \mathbf{S}[\mathrm{R}] \bullet \mathcal{A}_m)$ **then** append $[\varphi(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m)]$ to **LA**
  end while
end for

Figure 5 presents the SQL query to transform data from $\mathbf{S}_1$.MOVIE and $\mathbf{S}_2$.FILM to **M**.MOVIE from the RCA ψ_6 and ACA $\psi_8, \psi_9, \psi_{10}$, and ψ_{11}. The "*select*" clause (in line 2) is derived based on ACAS ψ_8, ψ_9 , ψ_{10} and ψ_{11}. The "*from*" clause (in line 3) implements a join operation as specified by RCA ψ_6. The "*on*" clause (in line 4) is based on the join condition indicated in the end of ψ_6.

01. *insert into* **M**.MOVIE(**title**,**year**,**genre**,**description**)
02. *select* FILM.**title** *as* **title**, FILM.**year** *as* **year**, MOVIE.**category** *as* **genre**,MOVIE.**summary** *as* **description**
03. *from* $\mathbf{S}_2$.FILM as FILM *left join* $\mathbf{S}_1$.MOVIE as MOVIE *on* FILM.**title** = MOVIE.**film** *and* FILM.**year** = MOVIE.**year**;

Fig. 5. Query definition to populate **M**.MOVIE from $\mathbf{S}_2$.FILM and $\mathbf{S}_1$.MOVIE.

Algorithm 3. G_SQL_GCA( ).

Input: R^T, ψ_R

```
Let S[R] be a source's relation in ψ_R
for all attribute A^T in R^T do
    while ∃ψ_A/ψ_A is an GCA of A^T relating it to some atribute of S[R] do
        if ψ_A: G[R^T] • A^T ← S[R] • A_1 then append [A_1] to LA
        if ψ_A: G[R^T] • A^T ← S[R] • ϱ/B_k then
            for all FK in ϱ do
                Let R_1 and R_2 be relation schemas related by FK
                Let [a_1, ..., a_n] and [b_1, ..., b_n] be lists of key attributes of, respectively, R_1 and R_2
                append [R_1, R_2] to S
                append [a_1 = b_1, ..., a_n = b_n] to J
                append [B] to LA
            end for
        if ψ_A:G[R^T] • A^T ← (S[R] • A_1, p_1) and ψ_R is of metadata then
            append [A_1] to LA
            append [Temp_R] to Aux
            append [p_1] to JAux
    end while
end for
```

Table 1. Templates to generate SQL Statements induced by RCAS and ACAS.

T1	*insert into* **N** (**att**[1], **att**[2], . . . , **att**[n]) *select* **LA**[1], **LA**[2], . . . , **LA**[n] *from* **S**[1]	ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2]$
T5	*insert into* **N** (**att**[1], **att**[2], . . . , **att**[n]) *select* **LA**[1], **LA**[2], . . . , **LA**[n] *from* **S**[1], **S**[2], . . . , **S**[m] *left join* **RJ**[1] *on* θ	ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2] \supset\!\bowtie \mathbf{S}_3[\mathrm{R}_3]\theta$
T7	*insert into* **N** (**att**[1], **att**[2], . . . , **att**[n]) *select* **LA**[1], **LA**[2], . . . , **LA**[n] *from* **S**[1] *outer join* **RJ**[1] *on* θ	ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2] \cup \mathbf{S}_3[\mathrm{R}_3]$ ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2(\mathbf{p}_2)] \supset\!\!\subset \mathbf{S}_3[\mathrm{R}_3(\mathbf{p}_3)]\theta$
T13	*insert into* **N** (**att**[1], **att**[2], . . . , **att**[n]) *select* **att**[1], **att**[2], . . . , **A TT**[n] *from* (*select* **LA**[1], **LA**[2], . . . , **LA**[w] *from* **S**[1] *where* **JAux**[1]) *as* **Aux**[1]) *outer join* (*select* **LA**[1], **LA**[2], . . . , **LA**[w] *from* **S**[1] *where* **JAux**[2]) *as* **Aux**[2]) *on* (**Aux**[1].ID = **Aux**[2].ID)	ψ: $\mathbf{S}_1[\mathrm{R}_1] \Leftarrow metadata(\mathbf{S}_2[\mathrm{R}_2](\mathcal{A}_1)$
T15	*insert into* **N** (**att**[1], **att**[2], . . . , **att**[n]) *select* **LA**[1], **LA**[2], . . . , **LA**[n] *from* **S**[1] *group by* **G**[1]	ψ: $\mathbf{S}_1[\mathrm{R}_1] \Leftarrow normalise(\mathbf{S}_2[\mathrm{R}_2](\mathcal{A}_1))$

Figure 6 presents the definition of the query to transform data from $\mathbf{S}_1$.MOVIEMAKERS to **M**.FILMMAKERS. For this query, we have to define a nested select statement to each case-base GCA that relates attributes of $\mathbf{S}_1$.MOVIEMAKERS to attributes of **M**.FILMMAKERS. Each nested select statement must be joined through an outer-join in order to guarantee both: i) that duplicate tuples will be merged properly, and 2) not duplicate tuples will be stored in **M**.FILMMAKERS. Thus, the clauses "*from*" (line 3), "*outer join*" (line 7), and "*on*" (line 12) correctly implement the data-metadata relationship specified by the SCA ψ_7. The "*on*" clause (line 12) is based on the attribute indicated in ψ_7. The first nested select statement (lines 4 to 6) is defined based on the GCAS ψ_{12} and ψ_{14}. The second nested select statement (lines 8 to 11) is similar to the first one, but now it is based on ψ_{13} and ψ_{14}. The "*select*" clause in line 2 is based on the left-hand side of GCAs ψ_{12}, ψ_{13} and ψ_{14}.

```
01. insert into M.FILMMAKERS(movie, producer, director)
02. select movie, producer, director
03. from
04. (select MOVIE.film as movie,MOVIEMAKERS.name as producer, MOVIEMAKERS.id as ID
05.     from S1.MOVIEMAKERS as MOVIEMAKERS, S1.MOVIE as MOVIE
06.     where MOVIEMAKERS.id=MOVIE.id and MOVIEMAKERS.role = 'producer') as T1_MOVIEMAKERS
07. outer join
08. (select MOVIE.film as movie,MOVIEMAKERS.name as director, MOVIEMAKERS.id as ID
09.     from S1.MOVIEMAKERS as MOVIEMAKERS, S1.MOVIE as MOVIE
10.     where MOVIEMAKERS.id=MOVIE.id
11.     and MOVIEMAKERS.role = 'director') as T2_MOVIEMAKERS
12.     on (T1_MOVIEMAKERS.ID = T2_MOVIEMAKERS.ID);
```

Fig. 6. Query definition to populate **M**.FILMMAKERS from **S**$_1$.MOVIEMAKERS.

6 Empirical Evaluation

We have performed some preliminary tests to verify that our approach is tractable for reasonably sized input.

6.1 Study Case Scenario

For our evaluation, we create a case study to simulate a situation close to the real world. We need to integrate information of three different sources: IES, FSP, and CDV to get a more complete information about Brazil's universities. IES contains data about Brazil's universities (name, city, state, etc.). It has a single relation (IES_2011) with 26 attributes and 2366 tuples[3]. FSP contains data about the ranking of the Brazil's universities (ranking, university, grade, etc.). It has two relations, but only one of them (RANKING, with 13 attributes and 191 tuples) was used in the evaluation[4]. CDV contains data about the living cost of some cities of Brazil. It has a single relation (LIVINGCOST) with 5 attributes and 84 tuples[5].

The integrated schema, named G, contains the necessary structure to keep the information required by the designer. It contains 8 relations with a total of 31 attributes and 8 foreign keys, as can be saw in the Fig. 7.

6.2 Method

We measure the performance of the data translation (i.e., the run time of the queries to load the schema G). For our tests, we have used a Macbook Pro/2.3GHz Intel Core (4GB of RAM and 499Gb of HD) running OSx 10.9.5. All databases were locally stored in this machine using the MySQL 5.6.

We first manually defined the CAs, with the aid of a tool implemented by us. For this case study, we defined 3 RCAs, 5 SCAs of normalize, 31 ACAs,

[3] IES data was extract from http://www.dados.gov.br/dataset/instituicoes-de-ensino-superior.

[4] FSP data was extract from http://ruf.folha.uol.com.br/2014/rankingdeuniversidades/.

[5] CDV data was extract from http://wwwcustodevida.com.br/brasil.

FSP_RANKING(**univ_id**, **census_year**, **ranking**, **grade**)
FK1(FSP_RANKING, ⟨**univ_id**⟩, UNIVERSITY, ⟨**id**⟩)
LIVINGCOST(**city_id**, **year**, **livingCost**, **state_id**)
FK3(LIVINGCOST, ⟨**city_id**⟩, CITY, ⟨**city_id**⟩)
FK4(LIVINGCOST, ⟨**state_id**⟩, STATE, ⟨**ibge_id**⟩)

UNIVERSITY_TYPE(**code**, **type**)
STATE(**ibge_id**, **state**, **s_mark**)
PLACE(**code**, **place**)
ORGANIZATION(**code**, **type**)
CITY(**city_id**, **city**, **state_id**)
FK2(CITY, ⟨**state_id**⟩, STATE, ⟨**ibge_id**⟩)

UNIVERSITY(**id**, **name**, **short_name**, **city_id**, **place_id**, **type_id**, **org_id**, **postalCode**, **address**, **telephone**, **contact**, **homepage**)
FK5(UNIVERSITY, ⟨**place_id**⟩, PLACE, ⟨**code**⟩)
FK6(UNIVERSITY, ⟨**org_id**⟩, ORGANIZATION, ⟨**code**⟩)
FK7(UNIVERSITY, ⟨**city_id**⟩, CITY, ⟨**city_id**⟩)
FK8(UNIVERSITY, ⟨**type_id**⟩, UNIVERSITY_TYPE, ⟨**code**⟩)

Fig. 7. The integrated schema G.

and 8 GCAs, being a total of 52 CAs. Using the Algorithm 1, we generate 8 SQL queries: 5 queries of group by, 2 simple *select-from* queries, and 1 more complex query that simulates the outer join operator. Some queries use stored functions defined to looks for the value of a primary key in a target relation based on the attribute value of a source relation.

For data translation test, we measured the time that MySQL took to load each target relation using the queries generated by the Algorithm 1. Due to the run time of SQL queries can change depending on internal and external factors, we ran each query by 50 times and took the average to each 10 executions. All tests were performed locally in a same machine and only the MySQL server and MySQLWorkbench were running at the time. The result of the test can be observed through the chart shown in Fig. 8.

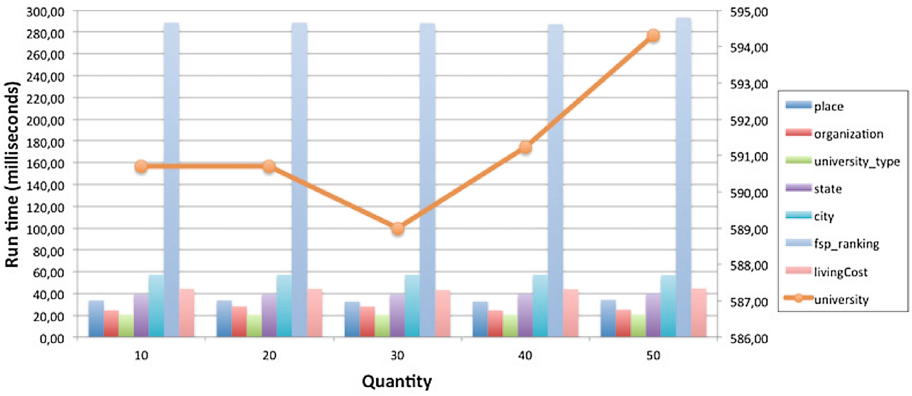

Fig. 8. Run time query by quantity of executions chart.

6.3 Discussion

We noted that the run time to most queries are more or less constant and below that 60 ms. It is not a surprise that the queries with higher execution time were those to load FSP_RANKING (about 200ms) and UNIVERSITY (about 590 ms), because both has more tuples to load than the others. Considering the number of tuples of the UNIVERSITY (more than 2000) and that the query generated is a bit complex (includes left-join, right-join, union all, and 5 stored functions), we believe that 590 ms is a good performance.

7 Related Work

Schema matching is an important step of the data integration process. Typically, 1:1 correspondences between two different schemas are manually defined using a GUI or are (semi-) automatically discovered using matchers (usually through heuristics). Each correspondence, in general, only specifies which elements refer to a same attribute or relation in the real world [17]. AgreementMaker [4], and OII Harmony [9] are some examples of tools for schema matching. AgreementMaker [4] can match schemas and ontologies using schema information as well as instance-level data to generate the correspondences. OII Harmony [9] combines multiple matchers algorithms based on natural-language processing to identify correspondences between schemas.

Correspondences such as those defined/generated in [4,9] do not provide all necessary information for discovering expressions to transform data sources in data target (i.e., the mapping expressions), the next phase in the schema mapping process. Richer models for specifying correspondences between schemas were proposed by [8,11–13] and [5](chap. 3). These approaches allow to define one-to-one or many-to-one attribute correspondences (i.e., association between attributes of two schemas). COMA++ [8] is a generic prototype for schema and ontology matching, schema-based and instance-based, and support a semi-automatic or manual enrichment of simple 1:1 correspondences into more complex mapping expressions including functions to support data transformations. [13] describes the IMAP system, which semi-automatically discovers complex matches, using different kinds of information such as domain knowledge, and domain integrity constraints to improve matching accuracy. [5](chap. 3) and [11] allow to express conditional correspondences (i.e., the value of an attribute A is the same of an attribute B if a given condition is satisfied). More closely to our approach is the work in [12]. In [12], the authors allow to manually specify one-to-one correspondence assertions between elements of Entity Relationship models. Although they cannot specify many-to-many matches, their correspondences have some semantic and allow to specify relationships such as: equivalence, union, intersection, and selection.

[10] specify one-to-one and many-to-many basic, complex, and semantic matches between elements of object-relational schemas. They can specify most part of the correspondences specified in [12] and other more complex. For example, they can deal with aggregate functions, denormalisations, and grouping (i.e., *group by* in SQL). Joins and outer-joins are implicitly defined based on the

integrity constraints or match functions[6]. A distinguished feature of the approach proposed in [10] is that it allows to match, in the same correspondence, relations and attributes of two or more schemas. Yet, the information they provide is not sufficient, since they do not explicitly enable the specification of join paths and its variants, nor to deal with data-metadata relationships.

Data-metadata translations between elements of different relational schemas have been studied extensively. SchemaSQL [16] and FIRA/FISQL [18] are the most notable works on this subject. SchemaSQL [16] is a SQL-like metadata query language that uses view statements to restructure one column of values of a relation into metadata in another one. FISQL [18] is a sucessor of SchemaSQL and it is equivalent to the query algebra FIRA. Both SchemaSQL and FIRA/FISQL were proposed to provide interoperability in relational multidatabase systems. Our SCA of metadata was based on the *promote metadata* operator of FIRA.

8 Conclusions

This paper focused on present CAs that deal with 1:1 and m:n matchings between schemas components, including correspondences involving aggregations, joins, and metadata. We emphasize that, in our approach, the CAs can specify basic and complex correspondences with semantics. Using CAs, we shown how SQL queries can be automatically generated to populate relations (views) of a global schema.

We presented some preliminary tests to evaluate the performance of the queries generated from CAs. We intend to realize more tests to evaluate the performance to different types of queries and other datasets.

We currently are working in as specifying complex correspondences between relational schemas and (RDF). Some initial work was published in [19]. We intent extend the initial proposal with the CAs presented here.

Acknowledgements. This work was partially supported by national funds through FCT - Fundação para a Ciência e a Tecnologia, under the project PEst-OE/EEI/LA0021/2013, DataStorm Research Line of Excellency funding (EXCL/EEI-ESS/0257/2012) and the grant SFRH/BPD/76024/2011. We are especially grateful to Diego Cardoso (UFC, Brazil) for the implementation of the algorithms.

References

1. Kimball, R., Ross, M., Thornthwaite, W., Mundy, J., Becker, B.: The Data Warehouse Lifecycle Tookit, 2nd edn. Wiley, Indianapolis (2008)
2. Popfinger, C.: Enhanced Active Databases for Federated Information Systems. PhD thesis, Heinrich Heine University Düsseldorf (2006)

[6] Match functions are functions that determine if two different instances represent the same concept in the real world.

3. Langegger, A., Wöß, W., Blöchl, M.: A semantic web middleware for virtual data integration on the web. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 493–507. Springer, Heidelberg (2008)
4. Cruz, I.F., Antonelli, F.P., Stroe, C.: Agreementmaker: efficient matching for large real-world schemas and ontologies. Proc. VLDB Endow. **2**(2), 1586–1589 (2009)
5. Bellahsene, Z., Bonifati, A., Rahm, E. (eds.): Schema Matching and Mapping. Data-Centric Systems and Applications. Springer, Heidelberg (2011)
6. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. **10**(4), 334–350 (2001)
7. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. In: Spaccapietra, S. (ed.) Journal on Data Semantics IV. LNCS, vol. 3730, pp. 146–171. Springer, Heidelberg (2005)
8. Massmann, S., Raunich, S., Aumueller, D., Arnold, P., Rahm, E.: Evolution of the COMA match system. In: The 6th Intl. Workshop on Ontology Matching. (2011)
9. Mork, P., Seligman, L., Rosenthal, A., Korb, J., Wolf, C.: The Harmony integration workbench. J. Data Semant. **11**, 65–93 (2008)
10. Pequeno, V.M., Pires, J.C.M.: Using perspective schemata to model the ETL process. In: ICMIS 2009, pp. 332–339. World Academy of Science, Engineering and Technology (2009)
11. Bohannon, P., Elnahrawy, E., Fan, W., Flaster, M.: Putting context into schema matching. In: VLDB, pp. 307–318 (2006)
12. Vidal, V.M.P., Lóscio, B.F.: Updating multiple databases through mediators. In: ICEIS 1999, pp. 163–170 (1999)
13. Dhamankar, R., Lee, Y., Doan, A., Halevy, A.Y., Domingos, P.: IMAP: Discovering complex mappings between database schemas. In: ACM SIGMOD, pp. 383–394 (2004)
14. Yan, L.L., Miller, R.J., Haas, L.M., Fagin, R.: Data-driven understanding and refinement of schema mappings. In: ACM SIGMOD, pp. 485–496. ACM (2001)
15. Pequeno, V.M., Aparício, J.N.: Using correspondence assertions to specify the semantics of views in an object-relational data warehouse. In: ICEIS 2005, pp. 219–225 (2005)
16. Lakshmanan, L., Sadri, F., Subramanian, I.: SchemaSQL - a language for interoperability in relational multi-database systems. In: VLDB, pp. 239–250. Morgan Kaufmann (1996)
17. Doan, A., Halevy, A., Ives, Z.: Principles of Data Integration. Morgan Kaufmann, Waltham (2012)
18. Wyss, C.M., Robertson, E.L.: Relational languages for metadata integration. ACM Trans. Database Syst. **30**, 624–660 (2005)
19. Pequeno, V.M., Vidal, V.M.P., Casanova, M.A., Neto, L.E.T., Galhardas, H.: Specifying complex correspondences between relational schemas and rdf models for generating customized R2RML mappings. In: IDEAS 2014, pp. 96–104. ACM (2014)