Specifying Complex Correspondences Between Relational Schemas in a Data Integration Environment

Valéria Pequeno[1] and Helena Galhardas[2]
[1]*INESC-ID, Taguspark, Lisbon, Portugal*
[2]*DEI, Instituto Superior Técnico and INESC-ID, Taguspark, Lisbon, Portugal*

Keywords: Schema Matching, Correspondence Assertions, Data Integration, Relational Model.

Abstract: When dealing with the data integration problem, the designer usually encounters incompatible data models characterized by differences in structure and semantics, even in the context of the same organization. In this work, we propose a declarative and formal approach to specify 1-to-1, 1-m, and m-to-n correspondences between relational schema components. Differently from usual correspondences, our Correspondence Assertions (CAs) have semantics and can deal with joins, outer-joins, and data-metadata relationships. Finally, we demonstrate how we can generate mapping expressions in the form of SQL queries from CAs.

1 INTRODUCTION

A Data Integration (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 a global schema (also named mediation schema). The global 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 (Kimball et al., 2008) is a typical example of the first approach. As examples of the second approach, we can cite federated information systems (Popfinger, 2006) and mediator systems (Langegger et al., 2008). In the present work, both scenarios can be used, but in this paper we will focus on the materialized integration approach.

One of the hardest problems to solve in DI is to define mappings between the global 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. Each correspondence specifies the elements that refer to the same real world entity (Doan et al., 2012). These correspondences can be described using a Local-as-view (LAV), a Global-as-view (GAV), or a Global and Local-as-view (GLAV) language. In summary, in a LAV approach, each data source is described as a view over the global schema. In a GAV approach, the global 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 (Cruz et al., 2009; Seligman et al., 2010; Bellahsene et al., 2011). 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*. Goods surveys can

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

Pequeno V. and Galhardas H..
Specifying Complex Correspondences Between Relational Schemas in a Data Integration Environment.
DOI: 10.5220/0004870200180029
In *Proceedings of the 16th International Conference on Enterprise Information Systems* (ICEIS-2014), pages 18-29
ISBN: 978-989-758-027-7

be found in (Rahm and Bernstein, 2001; Shvaiko and Euzenat, 2005).

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**). (Doan, 2002; Massmann et al., 2011; Mork et al., 2008) are examples of approaches that propose formalisms to specify complex matches.

Some researchers go beyond dealing with complex matches and add semantics to the correspondences, in order to improve the overall matching quality. In the Section 2, we explain more about complex matches and present a motivation example for the current work. The remainder of the paper is structured as follows. In Section 3, we present the necessary background in Correspondence Assertions (CAs), the formalism used in this work to specify correspondences between elements of schemas. In Section 4, we propose new CAs to deal with join operators and metadata. Section 5 shows how to generate mapping expressions from CAs. Section **??** describes the related work. Finally, Section 7 concludes and describes future work.

2 MOTIVATING EXAMPLE

Consider a motivating example with the source schemas $\mathbf{S}_1$ and $\mathbf{S}_2$ in Figure 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. The 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, **format** is the type of the tape (dvd, blu-ray, etc.), **name** can be a producer or a director name, and **role** is the role of a professional of the show business: producer or director. FK1 and FK2 are foreign keys. We use, as in (Vidal et al., 2013), 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 $\mathbf{S}_2$. The attributes in $\mathbf{S}_2$ have the following meaning: **id** is the movie identifier, **year** is the year of a movie, **title** is the title of a movie, **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.

* *Schema* $\mathbf{S}_1$
MOVIE(**id**, **film**, **year**, **summary**)
TAPE(**number**, **format**, **id**)
FK1(TAPE, ⟨**id**⟩, MOVIE, ⟨**id**⟩)
MOVIEMAKERS(**id**, **name**, **role**)
FK2(MOVIEMAKERS, ⟨**id**⟩, MOVIE, ⟨**id**⟩)

* *Schema* $\mathbf{S}_2$
FILM(**id**,**title**, **year**, **rate**)
SHOWTIME(**id**, **location**, **time**, **city**)
FK3(SHOWTIME, ⟨**id**⟩, FILM, ⟨**id**⟩)

* *Schema* **M**
MOVIE(**title**, **genre**, **year**, **description**)
FILMMAKERS(**movie**, **producer**, **director**)
FK4(FILMMAKERS, ⟨**movie**⟩, MOVIE, ⟨**title**⟩)
SCHEDULE(**movie**, **cinema**, **startTime**)
FK5(SCHEDULE, ⟨**movie**⟩, MOVIE, ⟨**title**⟩)
REMAKES(**title**, **nvYear**, **ovYear**)
FK6(REMAKES, ⟨**title**⟩, MOVIE, ⟨**title**⟩)
RATING(**rate**, **quantity**)

Figure 1: Example of source schemas and a global schema.

The global schema **M**, also shown in Figure 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 **M**.MOVIE stores movies shown currently at a cinema[2]. The relation **M**.FILMMAKERS keeps information about professionals of show businesses. The relation **M**.SCHEDULE contains information about the schedule of movies shown in Lisbon. The relation **M**.REMAKES keeps the years of movies for which there is at least one remake. The relation **M**.RATING stores the classification of movies with regard to suitability audience. Some non-self-explanatory at-

[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.

tributes in **M** have the following meaning: **description** is the summary of a movie, **genre** is the category 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, **rate** is the classification of the movie with regard to the audience (M/4, M/6, etc. in the Portuguese classification), 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**. In a first example, we can state that **M**.SCHEDULE corresponds to $\mathbf{S}_2$.SHOWTIME, because both relations store information regarding the same real world concept. 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.

As a second example, consider the attributes $\mathbf{S}_2$.SHOWTIME.**id** and **M**.SCHEDULE.**movie**, which uniquely identify a movie in the corresponding relations. The domain of the former is an integer (the identifier of the movie **id**), while the domain of the latter is a string (the title of the movie **movie**). Since **M**.SCHEDULE corresponds to $\mathbf{S}_2$.SHOWTIME, we need to relate **M**.SCHEDULE.**movie** to an element of $\mathbf{S}_2$.SHOWTIME that identifies a movie. However, $\mathbf{S}_2$.SHOWTIME does not store the title of movies. Moreover, $\mathbf{S}_2$.SHOWTIME.**id**, which is used to identify a movie in $\mathbf{S}_2$.SHOWTIME, has a type and a domain different from the type and domain of **M**.SCHEDULE.**movie**. Thus, $\mathbf{S}_2$.SHOWTIME.**id** and **M**.SCHEDULE.**movie** cannot be matched. Knowing that $\mathbf{S}_2$.SHOWTIME.**id** is a foreign key that refers to $\mathbf{S}_2$.FILM and that $\mathbf{S}_2$.FILM.**title** stores the title of a movie, the correct matching would then be: **M**.SCHEDULE.**movie** *corresponds to* $\mathbf{S}_2$.FILM.**title**, *when* $\mathbf{S}_2$.SHOWTIME.**id** = $\mathbf{S}_2$.FILM.**id**.

The works reported in (Pequeno and Pires, 2009; Bohannon et al., 2006; Pequeno, 2011; Vidal and Lóscio, 1999) and (Bellahsene et al., 2011)(chap. 3) propose schema matching approaches that can specify correspondences to deal with situations as required in the first example, while (Pequeno and Pires, 2009; Pequeno, 2011; Vidal and Lóscio, 1999) propose correspondences to deal with the matching required in the second example. The reader can see other proposals to add semantics to schema matching in (Massmann et al., 2011; Dhamankar et al., 2004; Magnani et al., 2005). However, the following situations have not been fully covered yet:

1. *Correspondences Between Relations Involving a Join with Inequality Conditions:* 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 (Yan et al., 2001), 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 **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 (Pequeno and Pires, 2009; Pequeno and Aparício, 2005; Pequeno, 2011). 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].

3.1 Basic Concept and Notation

We assume that the reader is familiar with the relational concepts. We denote a relation schema as $R(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$, and a foreign key as FK(R:*L*, S:*K*) (see Section 2). We say that FK relates R and S.

[3]In this work, the information sources and global schemas are defined using the RDM as proposed by Codd in (Codd, 1970) that only allows first normal form relations.

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 $R(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$ and a tuple variable **t** over R, we use $\mathbf{t}[\mathcal{A}_i]$ to denote the projection of **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 $\rho = \mathrm{FK}_1 \bullet \mathrm{FK}_2 \bullet \cdots \bullet \mathrm{FK}_{n-1}$ a path from R to T iff there is a list $\mathrm{R}_1, \ldots, \mathrm{R}_n$ of relation schemas in **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 (Pequeno and Pires, 2009; Pequeno, 2011), 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: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2] \cup \mathbf{S}_3[\mathrm{R}_3] \cup \cdots \cup \mathbf{S}_n[\mathrm{R}_n]$.
5. ψ: $\mathbf{S}_1[\mathrm{R}_1] \leftarrow \mathbf{S}_2[\mathrm{R}_2] \cap \mathbf{S}_3[\mathrm{R}_3] \cap \cdots \cap \mathbf{S}_n[\mathrm{R}_n]$.

We say that ψ *matches* R_1 *and* R_j, $2 \leq j \leq n$. □

RCAs express the different 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 one-to-one correspondence between their instances. For instance, ψ_1, shown in Figure 2, is an example of a RCA.

ψ_1: **M**[SCHEDULE] $\leftarrow$ **S**$_2$[SHOWTIME(**city** = "Lisbon")]
ψ_2: **M**[SCHEDULE]•**movie** $\leftarrow$ **S**$_2$[SHOWTIME]• FK$_3$/**title**

Figure 2: Examples of 1:1 correspondence assertions.

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

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

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. $\boldsymbol{S}_2[R_2] \bullet \mathcal{A}$*;*
2. $\boldsymbol{S}_2[R_2] \bullet \rho/\mathcal{B}$. □

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*:* $\boldsymbol{S}_1[R_1] \bullet \mathcal{A}_1 \leftarrow \mathcal{E}_1$.
2. ψ*:* $\boldsymbol{S}_1[R_1] \bullet \mathcal{A}_1 \leftarrow \varphi(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m)$.
3. ψ*:* $\boldsymbol{S}_1[R_1] \bullet \mathcal{A}_1 \leftarrow (\mathcal{E}_1, \boldsymbol{p}_1); \ldots; (\mathcal{E}_m; \boldsymbol{p}_m); \boldsymbol{v}$.
4. ψ*:* $\boldsymbol{S}_1[R_1](\mathcal{A}_1, \ldots, \mathcal{A}_n) \leftarrow (\mathcal{E}_1, \ldots, \mathcal{E}_n)$.

Where φ *is a function over attributes of* R_2*,* $\boldsymbol{p}_j$ *(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 Figure 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*:* $\boldsymbol{S}_1[R_1] \Leftarrow groupby(\boldsymbol{S}_2[R_2](\mathcal{A}'_1, \mathcal{A}'_2, \ldots, \mathcal{A}'_m))$.
2. ψ*:* $\boldsymbol{S}_1[R_1] \Leftarrow groupby(\boldsymbol{S}_2[R_2\sigma](\mathcal{A}'_1, \mathcal{A}'_2, \ldots, \mathcal{A}'_m))$.

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 Figure 3 is a simple example of a SCA.

ψ_3: **M**[RATING] $\Leftarrow$ groupby(**S**$_2$[FILM] (**rate**))
ψ_4: **M**[RATING] • **quantity** $\Leftarrow$ count(**S**$_2$[FILM] • **rate**)

Figure 3: Examples of m:n correspondence assertions.

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*:* $\boldsymbol{S}_1[R_1] \bullet \mathcal{A} \Leftarrow \mathcal{E}_1$.
2. ψ*:* $\boldsymbol{S}_1[R_1] \bullet \mathcal{A} \Leftarrow \varphi(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m)$.
3. ψ*:* $\boldsymbol{S}_1[R_1] \bullet \mathcal{A} \Leftarrow (\mathcal{E}_1, \boldsymbol{p}_1); \ldots; (\mathcal{E}_m; \boldsymbol{p}_m); \boldsymbol{v}$.
4. ψ*:* $\boldsymbol{S}_1[R_1] \bullet \mathcal{A} \Leftarrow \gamma(\mathcal{E}_1)$.
5. ψ*:* $\boldsymbol{S}_1[R_1] \bullet \mathcal{A} \Leftarrow \gamma(\varphi(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m))$.
6. ψ*:* $\boldsymbol{S}_1[R_1] \bullet \mathcal{A} \Leftarrow \gamma(\mathcal{E}_1, \boldsymbol{p})$.
7. ψ*:* $\boldsymbol{S}_1[R_1] \bullet \mathcal{A} \Leftarrow \gamma(\varphi(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_m), \boldsymbol{p})$.

Where φ *is a function over attributes of* R_2*,* $\boldsymbol{p}_j$ *(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 **S**$_2$.FILM and **M**.RATING. ψ_4, represented in Figure 3, specifies that **M**.RATING.**quantity** corresponds to the counting of all distinct values of **S**$_2$.FILM.**rate**.

Definition 6. *Let* $\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots, \boldsymbol{S}_n$ *and* $\boldsymbol{T}$ *be relational schemas;* R_1 *be a relation schema of* $\boldsymbol{T}$*, and* R_2 *a relation schema of some* $\boldsymbol{S}_i$*,* $1 \leq i \leq n$*. Let also* $\mathcal{E}_j$ *(for* $1 \leq j \leq m$*) be expressions with one of the following forms: i)* $\boldsymbol{S}_i[R_2] \bullet \mathcal{A}_2$*; or ii)* $\boldsymbol{S}_i[R_2] \bullet \rho/\mathcal{A}_k$*, with* $\mathcal{A}_2$ *being an attribute of* R_2*,* ρ *a path from* R_2 *to* R_k*, and* $\mathcal{A}_k$ *an attribute of* R_k*. A schema matching between schemas* $\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots, \boldsymbol{S}_n$ *and the schema* $\boldsymbol{T}$ *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': $\mathbf{S}_1[R_1](\mathcal{A}_1,\ldots,\mathcal{A}_n) \leftarrow (\mathcal{E}_1,\ldots,\mathcal{E}_n)$ that matches* R_1 *and* R_2. □

4 SPECIFYING NEW CAs

In Section 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 as those presented in Definition 1.*
2. ψ: $\mathbf{S}_1[R_1] \leftarrow \mathbf{S}_2[R_2] \bowtie \mathbf{S}_3[R_3]\theta$.
3. ψ: $\mathbf{S}_1[R_1] \leftarrow \mathbf{S}_2[R_2] \sqsupset\!\bowtie \mathbf{S}_3[R_3]\theta$.
4. ψ: $\mathbf{S}_1[R_1] \leftarrow \mathbf{S}_2[R_2] \bowtie\!\sqsubset \mathbf{S}_3[R_3]\theta$.
5. ψ: $\mathbf{S}_1[R_1] \leftarrow \mathbf{S}_2[R_2] \sqsupset\!\bowtie\!\sqsubset \mathbf{S}_3[R_3]\theta$.

□

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

1. *Expressions as those presented in Definition 4.*
2. ψ: $\mathbf{S}_1[R_1] \Leftarrow \mathbf{S}_2[R_2] \bowtie \mathbf{S}_3[R_3]\theta$.
3. ψ: $\mathbf{S}_1[R_1] \Leftarrow \mathbf{S}_2[R_2] \sqsupset\!\bowtie \mathbf{S}_3[R_3]\theta$.
4. ψ: $\mathbf{S}_1[R_1] \Leftarrow \mathbf{S}_2[R_2] \bowtie\!\sqsubset \mathbf{S}_3[R_3]\theta$.
5. ψ: $\mathbf{S}_1[R_1] \Leftarrow \mathbf{S}_2[R_2] \sqsupset\!\bowtie\!\sqsubset \mathbf{S}_3[R_3]\theta$.
6. ψ: $\mathbf{S}_1[R_1] \Leftarrow \mathit{metadata}(\mathbf{S}_2[R_2](\mathcal{A}_1,\mathcal{A}_2,\ldots,\mathcal{A}_n))$.

□

Consider the three examples about join, outer-join, and data-metadata correspondences described in Section 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 Figure 4. 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$[MOVIE]($\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$[MOVIE]($\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] $\Leftarrow$ $\Leftarrow$ *metadata* ($\mathbf{S}_1$[MOVIEMAKERS](**id**))

Figure 4: 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 Figure 4. 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 Figure 4. 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 global 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 global schema using a high-level data model. In our case, we use the RDM.
2. Define the CAs that formally specify the relationships between the global 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 global schema.

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

Now, we should define CAs between **M** and $\mathbf{S}_1$, and CAs between **M** and $\mathbf{S}_2$. In our work, the CAs are specified using a GAV approach rather than a LAV one. Our choice was due to two facts: 1) the GAV approach forces an exact association between the global schema and the data sources (i.e., for each relation and attribute of the global schema it should exists a corresponding matching involving at least a relation and an attribute of the source schemas); and 2) 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 **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 GCAs between the attributes of R^T and R^S (if there is a SCA between R^T and R^S).
   (c) Determine which RCAs and SCAs can be combined to form a single CA.

In the current work, CAs were manually specified. However, we can use traditional schema matching tools, such as COMA (Massmann et al., 2011) or OII Harmony (Seligman et al., 2010; Mork et al., 2008), 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 **M** and the source schemas $\mathbf{S}_1$ and $\mathbf{S}_2$ can be found in Figures 4 and 5. The final

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

Figure 5: Examples of ACAs and GCAs.

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 (Lakshmanan et al., 1996).

Let $\mathcal{M}$ be a set of CAs that defines a matching between the source schemas $\mathbf{S}_1$, $\mathbf{S}_2$ and the global schema **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 $\mathcal{M}$.

Algorithm 1: Generate the SQL query to load the relation schemas of a global schema **G**.

for all relation schema R^T in **G do**
  N = R^T; **S** =[ ]; **J** =[ ]; **LA** =[ ]; **O** =[ ]; **RJ** =[ ]; **JAux** =[ ]; **Aux** =[ ];
  Let ψ_R be a RCA or SCA of R^T
  if ψ_R is a RCA **then** G_SQL_ACA(R^T)
  else
    G_SQL_GCA(R^T)
  end if
  switch ψ_R **do**
    case ψ_R: **G**[R^T] ← **S**[R]
      add [**S**.R as R] to **S**
      if J = [ ] **then**
        Use template T1 with parameters (**N**, *Att*(R^T),**LA**, **S**)
      else
        Use template T3 with parameters (**N**, *Att*(R^T), **LA**, **S**, **J**)
      end if
    case ψ_R: **G**[R^T] ← $\mathbf{S}_1$[R_1] ⊐⋈ $\mathbf{S}_2$[R_2]θ
      add [$\mathbf{S}_1$.R_1 as R_1] to **S**
      add [$\mathbf{S}_2$.R_2 as R_2] to **RJ**
      if J = [ ] **then**
        Use template T5 with pars. (**N**, *Att*(R^T),**LA**, **S**, **RJ**, θ)
      else
        Use template T6 with pars. (**N**, *Att*(R^T), **LA**, **S**, **RJ**, θ, **J**)
      end if
    case ψ_R: **G**[R^T] ⇐ *metadata*(**S**[R](*A*))
      add [**S**.R as R] to **S**
      add [R.*A* as ID] to **AL**
      if J = [ ] **then**
        Use template T13 with parameters:
        (**N**, *Att*(R^T),**LA**, **S**, **JAux**, **Aux**)
      else
        Use template T4 with parameters:
        (**N**, *Att*(R^T), **LA**, **S**, **JAux**, **Aux**,**J**)
      end if
  end switch
end for

In Algorithm 1, we assume that **N** is a variable to keep a relation name; **S**, **J**, **LA**, **O**, and **RJ** are lists to keep, respectively, the relation schemas that will be included in the FROM clause, the *join conditions* that will be included in the WHERE clause, the at-

tributes that will be included in the SELECT clause, the *join attributes* that will be included in the ON clause, and the relation schema that will be included in (inner, outer, left, or right) JOIN clause. **JAux**, and **Aux** are lists used only 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. *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 algorithm 1 generates a set of SQL queries, one for each relation schema R^T in the global schema. First it spans all ACAs and GCAs that relates attributes of R^T, and puts the correct value in lists **LA**, **S**, **J**, in accordance to the type of the CA. This is performed by procedures *G_SQL_ACA()* and *G_SQL_GCA()* shown, respectively, in Algorithms 2 and 3 . After, the algorithm spans the RCAs, or SCAs, of R^T, in order to create the SQL query to load R^T, using templates in Table 1. 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.

Algorithm 2: G_SQL_ACA().

Input: R^T

for all attribute $\mathcal{A}^T$ in R^T **do**
  Let ψ_A be an ACA of $\mathcal{A}^T$
  if ψ_A: $\mathbf{G}[R^T] \bullet \mathcal{A}^T \leftarrow \mathbf{S}[R] \bullet \mathcal{A}_1$ **then**
    add [R.$\mathcal{A}_1$ as $\mathcal{A}^T$] to **LA**
  end if
  if ψ_A: $\mathbf{G}[R^T] \bullet \mathcal{A}^T \leftarrow (\mathbf{S}[R] \bullet \mathcal{A}_1, \mathbf{p}_1); \ldots; (\mathbf{S}[R] \bullet \mathcal{A}_m; \mathbf{p}_m); \mathbf{v}$ **then**
    add [case when $\mathbf{p}_1$ then R.$\mathcal{A}_1$ when $\mathbf{p}_2$ then … else **v** end '$\mathcal{A}^T$'] to **LA**
  end if
end for

In Algorithms 2 and 3, we assumed that φ() is a pre-defined SQL function or a user-defined function on SQL. The symbol *v* indicates a value (integer, string, float, etc.), and $\mathbf{p}_1, \ldots, \mathbf{p}_m$ are boolean conditions.

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 global schema is firstly populated, or in situations involving complex operations.

Algorithm 3: G_SQL_GCA( ).

Input: R^T

for all attribute $\mathcal{A}^T$ in R^T **do**
  Let ψ_A be an GCA of $\mathcal{A}^T$ and ψ_R be a SCA of $\mathcal{A}^T$
  if ψ_A: $\mathbf{G}[R^T] \bullet \mathcal{A}^T \leftarrow \mathbf{S}[R] \bullet \mathcal{A}_1$ **then**
    add [R.$\mathcal{A}_1$ as $\mathcal{A}^T$] to **LA**
  end if
  if ψ_A: $\mathbf{G}[R^T] \bullet \mathcal{A}^T \leftarrow \mathbf{S}[R] \bullet \rho/\mathcal{B}_k$ **then**
    for all FK in ρ **do**
      Let R_1 and R_2 be relation schemas related by FK
      Let $[a_1, \ldots, a_n]$ be the list of key attributes of R_1
      Let $[b_1, \ldots, b_n]$ be the list of key attributes of R_2
      add [**S**.R_1 as R_1, **S**.R_2 as R_2] to **S**
      add [$R_1.a_1 = R_2.b_1$, …, $R_1.a_n = R_2.b_n$] to **J**
      add [$R_2.\mathcal{B}$ as $\mathcal{A}^T$] to **LA**
    end for
  end if
  if ψ_A:$\mathbf{G}[R^T] \bullet \mathcal{A}^T \leftarrow (\mathbf{S}[R] \bullet \mathcal{A}_1, \mathbf{p}_1)$ and ψ_R is of metadata **then**
    add [R.$\mathcal{A}_1$ as $\mathcal{A}^T$] to **LA**
    add [T_R] to **Aux**
    add [$\mathbf{p}_1$] to **JAux**
  end if
end for

Figure 6 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 ACAs $\psi_8, \psi_9, \psi_{10}$, 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.

Q_{MOVIE}:
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
04. *on* FILM.**title** = MOVIE.**film** *and* FILM.**year** = MOVIE.**year**;

Figure 6: Query definition to populate **M**.MOVIE from $\mathbf{S}_2$.FILM and $\mathbf{S}_1$.MOVIE.

Figure 7 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

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: $\mathbf{S}_1[R_1] \leftarrow \mathbf{S}_2[R_2]$
T3	*insert into* **N** (**ATT**[1], **ATT**[2], ..., **ATT**[n]) *select* **LA**[1], **LA**[2], ..., **LA**[n] *from* **S**[1], **S**[2], ..., **S**[m] *where* **J**[1] *and* **J**[2] *and* ... *and* **J**[t]	ψ: $\mathbf{S}_1[R_1] \leftarrow \mathbf{S}_2[R_2]$
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: $\mathbf{S}_1[R_1] \leftarrow \mathbf{S}_2[R_2] \supset\!\bowtie \mathbf{S}_3[R_3]\theta$
T6	*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 *where* **J**[1] *and* **J**[2] *and* ... *and* **J**[t]	ψ: $\mathbf{S}_1[R_1] \leftarrow \mathbf{S}_2[R_2] \supset\!\bowtie \mathbf{S}_3[R_3]\theta$ ψ: $\mathbf{S}_1[R_1] \leftarrow \mathbf{S}_2[R_2] - \mathbf{S}_3[R_3]$
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: $\mathbf{S}_1[R_1] \Leftarrow metadata(\mathbf{S}_2[R_2](\mathcal{A}_1)$
T14	*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], **S**[2], ..., **S**[m] *where* **J**[1] *and* ... *and* **J**[t] *and* **JAux**[1]) *as* **Aux**[1]) *outer join* (*select* **LA**[1], **LA**[2], ..., **LA**[w] *from* **S**[1], **S**[2], ..., **S**[m] *where* **J**[1] *and* ... *and* **J**[t] *and* **JAux**[2]) *as* **Aux**[2]) *on* (**Aux**[1].ID = **Aux**[2].ID)	ψ: $\mathbf{S}_1[R_1] \Leftarrow metadata(\mathbf{S}_2[R_2](\mathcal{A}_1)$

$Q_{\text{FILMMAKERS}}$:
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* $\mathbf{S}_1$.MOVIEMAKERS *as* MOVIEMAKERS, $\mathbf{S}_1$.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* $\mathbf{S}_1$.MOVIEMAKERS *as* MOVIEMAKERS, $\mathbf{S}_1$.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);

Figure 7: Query definition to populate **M**.FILMMAKERS from $\mathbf{S}_1$.MOVIEMAKERS.

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}.

6 RELATED WORK

Schema matching is an important step of the data integration process (Filho et al., 2010). 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 (Doan et al., 2012). Cupid (Madhavan et al., 2001), AgreementMaker (Cruz et al., 2009), and OII Harmony (Seligman et al., 2010; Mork et al., 2008) are some examples of tools for schema matching. Cupid (Madhavan et al., 2001) is a generic schema matching that only consider schema information to generate the correspondences. AgreementMaker (Cruz et al., 2009) can match schemas and ontologies using schema information as well as

instance-level data to generate the correspondences. OII Harmony (Seligman et al., 2010; Mork et al., 2008) supports XML schemas, Entity-Relationship schemas, and relational schemas. It combines multiple matchers algorithms, each of which identifies correspondences using a different strategy, in order to generate correspondences with high quality matching between attributes of the compared schemas. (Bellahsene et al., 2011)(chap. 1) provides a brief comparison of some matching tools, while (Bellahsene et al., 2011)(chap. 9) presents, in a systematic way, some metrics to evaluate matching and mapping tools.

Correspondences such as those defined/generated in (Cruz et al., 2009; Seligman et al., 2010; Mork et al., 2008; Madhavan et al., 2001) do not provide the 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 (Doan, 2002; Massmann et al., 2011; Bohannon et al., 2006; Vidal and Lóscio, 1999; Dhamankar et al., 2004; Magnani et al., 2005; Giunchiglia et al., 2005) and (Bellahsene et al., 2011)(chap. 3). These approaches allow to define one-to-one or many-to-one attribute correspondences (i.e., association between attributes of two schemas). (Doan, 2002; Dhamankar et al., 2004) allow to semi-automatically match one attribute to various attributes (e.g., **totalPrice** matches to **unitPrice***quantity), incorporating machine learning, statistics, and heuristics to evaluate candidate matches. COMA and COMA++ (Massmann et al., 2011) are generic prototypes 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. (Dhamankar et al., 2004) describes the IMAP system, which semi-automatically discovers basic and complex matches. Each match in (Dhamankar et al., 2004) discovers specific types of complex matches, using different kinds of information such as domain knowledge, and domain integrity constraints to improve matching accuracy. (Bellahsene et al., 2011)(chap. 3) and (Bohannon et al., 2006) 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).

(Giunchiglia et al., 2005) and (Magnani et al., 2005) set correspondences with semantic relationships, such as equivalence, containment, subsumption, disjointness, and unknown (a special relationship returned by the matching algorithm when none of the others relationships hold). More closely to our approach is the work in (Vidal and Lóscio, 1999) and (Vidal et al., 2013). In (Vidal and Lóscio, 1999), 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 filtering (there named selection). In (Vidal et al., 2013), the authors allow to specify correspondences between relational schemas and RDF schemas, but they are basically 1-to-1 correspondences referring to project-selection-equijoin queries. The unique 1:m correspondence that they deal with relates one attribute to various attributes through concatenation.

(Pequeno and Pires, 2009; Pequeno, 2011) 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 (Vidal and Lóscio, 1999) 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[4]. A distinguished feature of the approach proposed in (Pequeno and Pires, 2009; Pequeno, 2011) 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 (Lakshmanan et al., 1996) and FIRA/FISQL (Wyss and Robertson, 2005) are the most notable works on this subject. SchemaSQL (Lakshmanan et al., 1996) 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 (Wyss and Robertson, 2005) is a sucessor of SchemaSQL. It is a query language that expresses data-metadata transformations using metavariables that range over relation names and column names. In addition, FISQL is equivalent to the query algebra FIRA. FIRA, in addition to the usual relational operators, defines simple operators for data-metadata querying such as: "$\uparrow$" to promote metadata, "$\downarrow$" to demote metadata, and "$\rightarrow$" to dereference data. Furthermore, FIRA/FISQL are capable of producing fully dynamic output schemas,

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

where the exact name of the relation schema or the attribute name is only known at run-time. Both SchemaSQL and FIRA/FISQL were proposed to provide interoperability in relational multi-database systems. Our new SCA of metadata was based on the *promote metadata* operator of FIRA.

In other proposals, such as those in (Haas et al., 2005; Bonifati et al., 2008), correspondences are basic matching (although sometimes they allow to specify selection conditions and combination of attributes) and the aim is automatically generate mapping expressions from the correspondences. Both Clio in (Haas et al., 2005) and Spicy in (Bonifati et al., 2008) define mapping expressions as logical formulas (tuple-generated dependences - tgds, and equality-generated dependences - egds). A drawback of approaches that create tgds automatically, as in (Haas et al., 2005) and (Bonifati et al., 2008), is that tgds do not deal with, for example, groupings, aggregations, many-to-many attribute correspondences, and data-metadata transformations. In addition, the quality of the mapping generated depends on the quality of the correspondences.
Clip (Raffio et al., 2008) (an extension of Clio) and ++Spicy (Marnette et al., 2011) (an extension of spicy) proposed extensions to tgds, named nested tgds, in order to be used in hierarchical data (i.e., XML instances). Their proposal enables to generate mappings between relational and XML schemas and allow to express grouping, aggregation and many-to-many attribute correspondences. Yet, they do not deal with data-metadata relationships, nor specify the condition that can be used for combining relations as well as join variants (e.g., right outer join and left outer join). Mad mapping (Papotti and Torlone, 2009) is an extension of Clio that deals with data-metadata relationships. Its approach is similar to the SchemaSQL approach (Lakshmanan et al., 1996), and, in some aspects to the FISQL. Since the correspondences used as input do not provide the necessary information for deriving the mappings that encode the semantic desired by the user, the solution proposed in (Raffio et al., 2008), (Marnette et al., 2011) and (Papotti and Torlone, 2009) can generate mapping expressions which populate a target relation with duplicate information. In our proposal, we specify correspondences that provide information that is sufficient for determining and deriving the desired schema mappings. In addition, differently from most approaches of schema matching and schema mapping, CAs allow to specify, in a same correspondence, the relationships between several source schemas and one target schema. This contributes to reduce the number of duplicate information in the target schema.

7 CONCLUSION

This paper focused on present Correspondence Assertions (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.

Our CAs were described using a GAV approach, rather than a LAV one. GAV approach has the advantage of making the query answering easier, since there is an exact association between the global schema and the data sources. However, GAV-based systems do not facilitate the addition of a new data source to the system. When a new data source is added or changed, the designer must modify the corresponding matchings/mappings. The semantically rich and formal representation of our Correspondence Assertions (CAs) enable to generate mapping expressions that are easy to reuse and maintain when some change occurs in a schema definition.

We are currently working on the development of a mechanism to discover new CAs from previous ones using inferences and heuristics. This mechanism intends to help the designer in the definition of new CAs, given insights and suggestions in an interactive way.

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 and the grant SFRH/BPD/76024/2011. We are especially grateful to Vania Vidal (UFC, Brazil) and João Moura Pires (UNL, Portugal) for valuable discussion and comments.

REFERENCES

Bellahsene, Z., Bonifati, A., and Rahm, E., editors (2011). *Schema Matching and Mapping*. Data-Centric Systems and Applications. Springer.

Bohannon, P., Elnahrawy, E., Fan, W., and Flaster, M. (2006). Putting context into schema matching. In *VLDB*, pages 307–318.

Bonifati, A., Mecca, G., Pappalardo, A., Raunich, S., and Summa, G. (2008). Schema mapping verification: the spicy way. In *EDBT'08, 11th Intl. Conf. on Extending*

Database Technology: Advances in Database Technology, pages 85–96. ACM.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

Cruz, I. F., Antonelli, F. P., and Stroe, C. (2009). Agreementmaker: Efficient matching for large real-world schemas and ontologies. *Proc. VLDB Endow.*, 2(2):1586–1589.

Dhamankar, R., Lee, Y., Doan, A., Halevy, A. Y., and Domingos, P. (2004). IMAP: Discovering complex mappings between database schemas. In *ACM SIGMOD*, pages 383–394.

Doan, A. (2002). *Learning to Map between Structured Representations of Data*. PhD thesis, University of Washington.

Doan, A., Halevy, A., and Ives, Z. (2012). *Principles of Data Integration*. Morgan Kaufmann.

Filho, F., Lóscio, B., and Macêdo, J. A. (2010). Geração incremental de correspondências e mapeamentos entre ontologias. In *X Workshop de Teses e Dissertações em Banco de Dados*.

Giunchiglia, F., Shvaiko, P., Yatskevich, M., Giunchiglia, F., Shvaiko, P., and Yatskevich, M. (2005). Semantic schema matching. In *On the Move to Meaningful Internet Systems: CoopIS, DOA, and ODBASE*, pages 347–365.

Haas, L. M., Hernández, M. A., Ho, H., Popa, L., and Roth, M. (2005). Clio grows up: from research prototype to industrial tool. In *ACM SIGMOD*, pages 805–810.

Kimball, R., Ross, M., Thornthwaite, W., Mundy, J., and Becker, B. (2008). *The Data Warehouse Lifecycle Tookit*. Wiley Publishing, 2nd edition.

Lakshmanan, L. V. S., Sadri, F., and Subramanian, I. N. (1996). SchemaSQL - a language for interoperability in relational multi-database systems. In *VLDB*, pages 239–250. Morgan Kaufmann Publishers Inc.

Langegger, A., Wöß, W., and Blöchl, M. (2008). *A Semantic Web Middleware for Virtual Data Integration on the Web*, volume The Semantic Web: Research and Applications 5021 of *Lecture Notes in Computer Science*, pages 493–507. Springer.

Madhavan, J., Bernstein, P. A., and Rahm, E. (2001). Generic schema matching with cupid. In *VLDB*, pages 49–58. Morgan Kaufmann Publishers Inc.

Magnani, M., Rizopoulos, N., Mc.Brien, P., and Montesi, D. (2005). Schema integration based on uncertain semantic mappings. In *ER 2005, Conceptual Modeling*, volume 3716 of *Lecture Notes in Computer Science*, pages 31–46. Springer Berlin / Heidelberg.

Marnette, B., Mecca, G., Papotti, P., Raunich, S., and Santoro, D. (2011). ++spicy: an opensource tool for second-generation schema mapping and data exchange. *Proc. VLDB Endow.*, 4(12):1438–1441.

Massmann, S., Raunich, S., Aumueller, D., Arnold, P., and Rahm, E. (2011). Evolution of the COMA match system. In *The 6th Intl. Workshop on Ontology Matching*.

Mork, P., Seligman, L., Rosenthal, A., Korb, J., and Wolf, C. (2008). The Harmony integration workbench. *J. Data Semantics*, 11:65–93.

Papotti, P. and Torlone, R. (2009). Schema exchange: Generic mappings for transforming data and metadata. *Data Knowl. Eng.*, 68(7):665–682.

Pequeno, V. M. (2011). *Using Perspective Schema and a Reference Model to Design the ETL Process*. PhD thesis, Universidade Nova de Lisboa.

Pequeno, V. M. and Aparício, J. N. (2005). Using correspondence assertions to specify the semantics of views in an object-relational data warehouse. In *ICEIS'05, 7th Enterprise Information Systems*, pages 219–225.

Pequeno, V. M. and Pires, J. C. M. (2009). Using perspective schemata to model the ETL process. In *ICMIS'09, Intl. Conf. on Management Information Systems*, pages 332–339. World Academy of Science, Engineering and Technology.

Popfinger, C. (2006). *Enhanced Active Databases for Federated Information Systems*. PhD thesis, Heinrich Heine University Düsseldorf.

Raffio, A., Braga, D., Ceri, S., Papotti, P., and Hernandez, M. (2008). Clip: a visual language for explicit schema mappings. In *ICDE'08, Intl. Conf. on Data Engineering*, pages 30–39. IEEE.

Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350.

Seligman, L., Mork, P., Halevy, A., Smith, K., Carey, M. J., Chen, K., Wolf, C., Madhavan, J., Kannan, A., and Burdick, D. (2010). OpenII: an open source information integration toolkit. In *ACM SIGMOD*, pages 1057–1060.

Shvaiko, P. and Euzenat, J. (2005). A survey of schema-based matching approaches. *Journal on Data Semantics IV*, 3730:146–171.

Vidal, V. M. P., Casanova, M. A., and Cardoso, D. S. (2013). Incremental maintenance of RDF views of relational data. In *ODBASE'13, 12th Intl. Conf. on Ontologies, DataBases, and Applications of Semantics*.

Vidal, V. M. P. and Lóscio, B. F. (1999). Updating multiple databases through mediators. In *ICEIS'99, Intl. Conf. on Enterprise Information Systems*, pages 163–170.

Wyss, C. M. and Robertson, E. L. (2005). Relational languages for metadata integration. *ACM Trans. Database Syst.*, 30:624–660.

Yan, L. L., Miller, R. J., Haas, L. M., and Fagin, R. (2001). Data-driven understanding and refinement of schema mappings. In *ACM SIGMOD*, pages 485–496. ACM.