SCHEMA MAPPING FOR RDBMS

Calin-Adrian Comes, Ioan Ovidiu Spatacean, Daniel Stefan, Beatrice Stefan
Petru Maior University, Department of Financial & Accounting
Nicolae Iorga 1, 540088 Tg-Mures, Romania

Lucian-Dorel Savu
Dimitrie Cantemir University, Department of Financial & Accounting
Bodoni Sandor 3-5, 540055 Tg-Mures, Romania

Vasile Paul Bresfelean, Nicolae Ghisoiu
Babes-Bolyai University, Department of IT Applied to Economics
Teodor Mihali 58-60, 400691, Cluj-Napoca, Romania

Keywords: Schema mapping, Stored procedures, Triggers.

Abstract: Schema mapping is a specification that describes how data structured from one schema $S$ the source schema is to be transformed into data structured under schema $T$, the target schema. Schemata $S$ and $T$ without triggers and/or stored procedures (functions and procedures) are statical. In this article, we propose a Schema Mapping Model specification that describes the conversion of a Schema Model from one Platform-Specific Model to other Platform-Specific Model according to Meta-Object Facility-Query/Verify/Transform in dynamical mode.

1 INTRODUCTION

Applications as database warehousing, global information systems and eletronic commerce need to take the existing schema with particular source $S$ and use it in different form, but they need to start with understanding how will be the target schema $T$. Data exchange are used in many tasks in theoretical studies research and practical in software products. In early stage 1977, in (Shu et al., 1977) with their EXPRESS, data exchange system with main functionality conversion data between hierarchical schemata the data exchange was in the top research topics. In (Fagin et al., 2003) Ronald Fagin et al. underline that the data exchange problem meet the foundation and algorithmic issues; their theoretical work has been motivated by the development of Clio (Miller et al., 2000; Popa et al., 2002), a prototype for data exchange and schema mapping from source schema $S$ to target schema $T$, the precursor of changes in SQL Assist from IBM DB2 family.

2 RELATED WORK

According to (Fagin et al., 2003) we have the source schema $S = \langle S_1, S_2, \ldots, S_n \rangle$, where $S_i$'s are the source relation symbols, the target schema $T = \langle T_1, T_2, \ldots, T_m \rangle$, where $T_i$'s are the target relation symbols and the schema $(S, T) = \langle S_1, S_2, \ldots, S_n, T_1, T_2, \ldots, T_m \rangle$. All instances over the $S$ represent source instances $I$, while instances over $T$ $J$ are target instances. If $i$ is a named source instance in $S$ and $j$ is a named target instance the $\kappa = \langle i, j \rangle$ is the named instance over the schema $(S, T)$. A dependency named source-to-target dependencies over $(S, T)$ of the form

$$(\forall x)(\phi_S(x) \rightarrow \chi_T(x))$$

where $\phi_S(x)$ is an expression (formula), with free variable $x = \langle x_1, x_2, \ldots, x_k \rangle$ of logical formalism over $S$ and $\chi_T(x)$ is an expression (formula) with free variable $x = \langle x_1, x_2, \ldots, x_l \rangle$ of logical formalism over $T$. A dependency named target dependencies over the target schema $T$ (the target dependencies are different from those use for the source-to-target dependencies)
Definition 2.1. A data exchange represent a 4-tuple $DE = (S, T, \Sigma_{st}, \Sigma_{st})$ with a source schema $S$, a target schema $T$, a set $\Sigma_{st}$ of source-to-target dependencies and set $\Sigma_{st}$ of target dependencies.

In (Berri and Vardi, 1984) Berri et al., proved that for practical purposes each source-to-target dependency $\Sigma_{st}$ represents a tuple-generating-dependency (tgd) of the form

$$(\forall x)(\phi_S(x) \rightarrow \phi_T(x,y))$$

where $\phi_S(x)$ represents a conjunction of atomic expression (formulas) over $S$ and $\chi_T(x,y)$ represents a conjunction of atomic expression (formulas) over $T$. In (Fagin et al., 2005b) Fagin et al. identified a particular universal solution for data exchange and schema mappings, and argued that this is the best universal solution.

Definition 2.2. A translator represents a program that reads on input in one language the source language - source code program - and translate it into output in an equivalent program in other language the target language - source code - see Figure 1.

A translator operates in the following phases: lexical analyzer, syntax analyzer, semantic analyzer, target code generator. In early stage 1950’s Naom Chomsky (Chomsky, 1956) proposed the formal definition for context-free grammar, see Figure 2. Context-free are used in the design and description of programming languages, compilers and translators. A context-free grammar is 4-tuple:

$$G = (V, \Sigma, R, S)$$

where $V$ - represents a finite set of non-terminal characters or variables; $\Sigma$ - represents set of terminals, disjoint with $V$; $R$ - represents a finite set of rules; $S$ - represents the start variable, used to represent the or program.

Definition 2.3. Let $\Sigma_1$ and $\Sigma_2$ be two alphabets, named source alphabet respective target alphabet and two languages $L_1 \subset \Sigma_1$, $L_2 \subset \Sigma_2$. A translator from the language $L_1$ to the language $L_2$ is a relation $T$ from $\Sigma_1$ to $\Sigma_2$ when the domain of $T$ is $L_1$ and the image of $T$ is $L_2$.

3 OUR APPROACH

Our algebraical approach to data exchange and schema mapping is to include the stored procedures in schema mappings and to snapshot the dynamical of the schema content in time extending (Fagin et al., 2003; Fagin et al., 2005b; Fagin et al., 2005a; Fagin, 2007; Fagin and Nash, ings), because they parse the statical schema mapping not a dynamical schema mapping. We propose the source schema $S(t) = \langle S_1(t), S_2(t), \ldots, S_n(t) \rangle$, where $S_i(t)'s$ are the source relation symbols, the target schema $T(t) = \langle T_1(t), T_2(t), \ldots, T_m(t) \rangle$, where $T_i(t)'s$ are the target relation symbols and the schema $\langle S(t), T(t) \rangle = \langle S_1(t), S_2(t), \ldots, S_n(t), T_1(t), T_2(t), \ldots, T_m(t) \rangle$. All instances over the $S(t)$ represent source instances $I(t)$, while instances over $T(t)$ $J(t)$ are target instances. If $i(t)$ is a named source instance in $S(t)$ and $f(t)$ is a
Given a set of attributes \( \text{Attr}(t) \) the named instance over the schema \( \langle S(t), T(t) \rangle \). A dependency named source-to-target dependencies over \( \langle S(t), T(t) \rangle \) of the form

\[
(\forall x(t))(\phi_{\text{S}(t)}(x(t)) \rightarrow \chi_{\text{T}(t)}(x(t)))
\]

where \( \phi_{\text{S}(t)}(x(t)) \) is an expression(formula), with free variable \( x(t) = (x_1(t), x_2(t), \ldots, x_k(t)) \) of logical formalism over \( \text{S}(t) \) and \( \chi_{\text{T}(t)}(x(t)) \) is an expression(formula) with free variable \( x(t) = (x_1(t), x_2(t), \ldots, x_k(t)) \) of logical formalism over \( \text{T}(t) \). A dependency named target dependencies over the target schema \( \text{T}(t) \) (the target dependencies are different from those use for the source-to-target dependencies).

**Definition 3.1.** A data exchange represent a 4-tuple \( \text{DE}(t) = \langle \text{S}(t), \text{T}(t), \sum_{\text{at}(t)}, \sum_{\text{st}(t)} \rangle \) with a source schema \( \text{S}(t) \), a target schema \( \text{T}(t) \), a set \( \sum_{\text{at}(t)} \) of source-to-target dependencies and set \( \sum_{\text{st}(t)} \) of target dependencies.

For practical purposes each source-to-target dependency \( \sum_{\text{at}(t)} \) represents a tuple-generating-dependency(tgd) of the form

\[
(\forall x(t))(\phi_{\text{S}(t)}(x(t)) \rightarrow \chi_{\text{T}(t)}(x(t)), y(t)))
\]

where \( \phi_{\text{S}(t)}(x(t)) \) represents a conjunction of atomic expression(formulas) over \( \text{S}(t) \) and \( \chi_{\text{T}(t)}(x(t)), y(t)) \) represents a conjunction of atomic expression(formulas) over \( \text{T}(t) \). A stored procedure named stored-procedure-s over \( \text{S}(t) \), of the form

\[
(\forall x(t))(\alpha_{\text{S}(t)}(x(t)) \rightarrow \chi_{\text{T}(t)}(x(t)))
\]

where \( \alpha_{\text{S}(t)}(x(t)) \) is a stored procedure over \( \text{S}(t) \) and a stored procedure named stored-procedure-t over \( \text{T}(t) \), of the form

\[
(\forall x(t))(\beta_{\text{S}(t)}(x(t)) \rightarrow \chi_{\text{T}(t)}(x(t)))
\]

where \( \beta_{\text{S}(t)}(x(t)) \) is a stored procedure over \( \text{T}(t) \).

**Definition 3.2.** A schema mapping model represent a 6-tuple \( \text{DE}(t) = \langle \text{S}(t), \sum_{\text{at}(t)}, \text{T}(t), \sum_{\text{tp}(t)}, \sum_{\text{st}(t)}, \sum_{\text{td}(t)} \rangle \) with a source schema \( \text{S}(t) \), all stored procedures over \( \text{S}(t) \), a target schema \( \text{T}(t) \), all stored procedures over \( \text{T}(t) \), a set \( \sum_{\text{at}(t)} \) of source-to-target dependencies and set \( \sum_{\text{st}(t)} \) of target dependencies.

Our approach on symbolic modeling of data exchange and schema mapping are:

**Definition 3.3.**

\[
\text{DB}(t) := \bigcup \{ \text{db}(t) | is - database(db(t)) \}
\]

where \( \text{db}(t) \) is a database

Given a set of attributes \( \text{Attr}(t) \) and a set containing sets of attribute values \( D(t) \), we define a column as a function mapping an attribute into the set containing its corresponding values:

\[
\text{ValColumn}(t) : \text{Attr}(t) \rightarrow D(t),
\]

\[
\text{ValColumn}(a(t)) := \{ d(t) | d(t) \in D(t) \}
\]

where \( d \) is a value for attribute \( 'a(t)' \)

**Figure 3: Database diagram for schema S.**

**Definition 3.4.** Given a set of attributes \( \text{Attr}(t) \), \( i = 1, \ldots, n \) the table \( \text{T}(t) \) from database is defined by:

\[
is - Table(T_n(t), \text{Attr}(t), i = 1, \ldots, n; D_i(t), i = 1, \ldots, n) \leftrightarrow \]

\[
T(t) \in \bigcup_{i=1}^{n} (\text{Attr}(t), \text{ValColumn}(\text{Attr}(t)))
\]

where \( \text{Card}(\text{ValColumn}(\text{Attr}(t))) = \text{nrw}(t) = \text{NoRows}(T(t)) \)

the number of lines in table \( \text{T}(t) \), \( i = 1, \ldots, n \), \( n(t) = \text{NoCol}(T(t)) \) the number of columns in the table \( \text{T}(t) \).

In practice is possible to have \( \text{S}=\text{T} \) but \( \text{T}(t) \) that case is named by us data exchange for copy schema mapping because all stored procedures over \( \text{S}(t) \), all stored procedures over \( \text{T}(t) \), have the same semantic but different syntax in SQL and Procedural Languages / SQL flavors on different RDBMS.

We consider the following subdiagram with schema \( S=\langle \text{EMPLOYEE}, \text{DEPARTMENT} \rangle \) with EMPLOYEE (EmployeeID, FName, LName, CompanyID), DEPT (DeptID, EmpN, Location) see the Database Diagram for schema S. In our case \( S=T=\langle \text{EMPLOYEE}, \text{DEPARTMENT} \rangle \). A trigger that increments the number of employees each time a new person is hired, that is, each time a new row is inserted into the table EMPLOYEE has the same **semantic** in \( S \) and \( T \) but different **syntax** in different **Procedural Language** over different SQL flavors.
Table 1: The triggers when a new person is hired.

<table>
<thead>
<tr>
<th>RDBMS</th>
<th>STORED PROCEDURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM DB2</td>
<td>CREATE TRIGGER NEWHIRED AFTER INSERT ON EMPLOYEE FOR EACH ROW MODE DB2SQL UPDATE DEPT SET EmpN = EmpN + 1</td>
</tr>
<tr>
<td>Oracle</td>
<td>CREATE TRIGGER NEWHIRED AFTER INSERT ON EMPLOYEE BEGIN UPDATE DEPT SET EmpN = EmpN + 1 WHERE EmployeeID=:New.EmployeeID END;</td>
</tr>
<tr>
<td>Sybase</td>
<td>CREATE TRIGGER &quot;NEWHIRED&quot; AFTER INSERT OF EmployeeID ON EMPLOYEE REFERENCING OLD AS EO NEW AS EN FOR EACH ROW BEGIN UPDATE DEPT SET DEPT.EmpN = DEPT.EmpN + 1 WHERE EMPLOYEE.EmployeeID=EN END</td>
</tr>
<tr>
<td>MySQL</td>
<td>CREATE TRIGGER NEWHIRED AFTER INSERT ON EMPLOYEE FOR EACH ROW UPDATE DEPT SET EmpN = EmpN + 1</td>
</tr>
<tr>
<td>Postgres</td>
<td>CREATE FUNCTION EmpA() BEGIN UPDATE FIRMA SET EmpN = EmpN + 1; END; LANGUAGE 'plpgsql' VOLATILE CREATE TRIGGER NEWHIRED AFTER INSERT ON EMPLOYEE FOR EACH ROW EXECUTE PROCEDURE EmpA();</td>
</tr>
</tbody>
</table>

4 CONCLUSIONS

In this paper we proposed data exchange metamodel for copy schema mappings that describes the conversion of Schema Model from one Platform-Specific Model to other Platform-Specific Model according to Meta-Object Facility-Query/Verify/Transform in dynamical mode. A prototype application, named ANCUTZA (ANalytiCal User Tool ZAMLokyx)-universal SQL and Procedural Language/SQL translator-for data exchange metamodel is in project phase in idea to support a part of SQL flavors on different RDMBS.

ACKNOWLEDGEMENTS

The authors would like to thank for the support to PhD. Zsuzsanna-Katalin Szabo, Dean at Faculty of Economics, Law, and Administrative Science from Petru Maior University of Tg-Mureș.

REFERENCES


