Письма в ЭЧАЯ. 2011. Т. 8, № 4(167). С. 655–663

«RELATIONSHIP» SPECIFICATION IN Z-NOTATION

V. Dimitrov
University of Sofia, Sofia, Bulgaria

Initially, relational model of data has been specified by E. F. Codd with the naming conventions to the attributes called «relationship». It is something between relations and tables, i.e., between implementation and user view on data. In this paper, ideas for «relationship» are formally specified in Z-notation. The last one is an ISO standard now. The purpose of this paper is to reinvestigate ideas behind the «relationships» in a more formal way. This approach is useful for further research in extending relation model of data to capture multimedia data and data streams, which are, usually, generated by different kinds of sensors.

Первоначально термин «реляционная модель» данных был использован Е. Ф. Коддом для обозначения свойств, связанных с «соотношением», которое обозначает связь между отношениями и таблицами, т. е. между абстрактным понятием и визуальным представлением данных. В статье «соотношение» рассматривается в формальном Z-обозначении. Сейчас это стандарт ISO. Целью этой статьи является пересмотр идей, стоящих за «соотношением», в рамках более формального подхода. Представленное здесь приближение можно использовать в дальнейших исследованиях при расширении реляционной модели данных для описания мультимедиаданных и потоков данных, которые обычно генерируются различными датчиками.

PACS: 02.10.Ab; 07.05.Kf

INTRODUCTION

The Relational Model of Data (RMD) that is specified here, follows the original presentation given in [1]. Z-notation is used as formal notation. It is now an ISO standard [2]. The original presentation of RDM is not formalized, nor detailed or consistent. It contains many open topics, which have been solved later sometimes in different ways in RMD implementations. In a formal specification such topics could not be evaded. The investigation on the topic shows that there is only one attempt in that direction [3]. It is a Master Thesis that is very huge and impractical to be used as a basis for any extensions.

1. SCHEMAS

The main types in this specification are:

$$[RNAMES, DNAMES, ROLES, VALUES],$$

where RNAMES is the set of all possible relation names, DNAMES is the set of all possible column names, ROLES is the set of all possible roles, and VALUES is the set of all possible tuple component values.

$$NULL : VALUES$$

NULL is a component value with a special purpose, that means «no value is specified».

DOMAINS

$$name : DNAMES$$
$$role : ROLES$$
$$domain : \mathbb{F}_1\ VALUES$$

Every domain (column) has a name, role, and finite set of possible values.

$$SCHEMAS == \mathrm{seq}_1\ DOMAINS$$

Relation schema is an ordered set of domains (sequence).

DBSchema

$$db : RNAMES \pfun SCHEMAS$$

$$(\forall n : \mathrm{dom}\ db \bullet$$
$$(\forall i, j : 1 \mathinner{..} \#(db(n)) \bullet$$
$$i \neq j \land ((db(n))(i)).name = ((db(n))(j)).name \Rightarrow$$
$$((db(n))(i)).role \neq ((db(n))(j)).role))$$

Database schema is a partial function between relation names and schemas. It is important, that two different columns in a relational schema to have the same names, but they have to have different roles. This rule is specified as an invariant in the Z-schema and relations defined in such a way are called «relationships».

DBSchemaInit

$$DBSchema$$

$$\mathrm{dom}\ db = \emptyset$$

Initially, the database schema is empty.

DBSRelationSchemaSize

$$\Xi DBSchema$$
$$n? : RNAMES$$
$$size! : \mathbb{N}_1$$

$$n? \in \mathrm{dom}\ db \land size! = \#(db(n?))$$

Above, Z-schema returns (size!) the relation name (given at input — name?). This Z-schema works for relations that have yet schemas; it does not consider errors when the input name has no schema. The case with errors can be upgraded further, but here this approach is not used, because the paper would be very huge considering error details.

DBSAdd

$$\Delta DBSchema$$
$$n? : RNAMES$$
$$s? : SCHEMAS$$
$$n? \notin \operatorname{dom} db \land$$
$$db' = db \cup \{n? \mapsto s?\}$$

In this Z-schema, the relation name is bounded with a schema (added).

DBSRemove

$$\Delta DBSchema$$
$$n? : RNAMES$$
$$n? \in \operatorname{dom} db \land$$
$$db' = \{n?\} \lhd db$$

A relation is removed by unbinding its name from the schema.

DBSRename

$$\Delta DBSchema$$
$$n? : RNAMES$$
$$new? : RNAMES$$
$$n? \in \operatorname{dom} db \land new? \notin \operatorname{dom} db \land$$
$$db' = \{n?\} \lhd db \cup \{new? \mapsto db(n?)\}$$

A relation can be renamed, the new name has not to be used yet.

DBSRemoveColumn

$$\Delta DBSchema$$
$$n? : RNAMES$$
$$dn? : DNAMES$$
$$r? : ROLES$$
$$n? \in \operatorname{dom} db \land$$
$$(\exists_1 i : 1 \mathinner{..} \#(db(n?)) \bullet$$
$$((db(n?))(i)).name = dn? \land ((db(n?))(i)).role = r? \land$$
$$db' = db \oplus$$
$$\{n? \mapsto ((1 \mathinner{..} (i-1)) \cup ((i+1) \mathinner{..} \#(db(n?)))) \upharpoonright db(n?)\})$$

A relation schema can be changed by removing a column from it. Because column names are not unique, in the relation schema domain role is used. Domain name and its role together uniquely identify the column in the schema. That is relationship approach.

DBSInsertColumn

$$
\begin{array}{l}
\Delta DBSchema \\
n? : RNAMES \\
dn? : DNAMES \\
r? : ROLES \\
d? : DOMAINS \\
\hline
n? \in \mathrm{dom}\ db \land \\
(\exists_1 i : 1 \mathinner{..} \#(db(n?)) \bullet ((db(n?))(i)).name = dn? \land \\
((db(n?))(i)).role = r? \land \\
db' = db \oplus \{n? \mapsto ((1 \mathinner{..} (i-1)) \upharpoonright db(n?)) \frown \\
\langle d? \rangle \frown ((i \mathinner{..} \#(db(n?))) \upharpoonright db(n?))\})
\end{array}
$$

A new column ($d?$) is inserted right before the column identified by its name ($dn?$) and role ($r?$).

DBSAddColumn

$$
\begin{array}{l}
\Delta DBSchema \\
n? : RNAMES \\
dn? : DNAMES \\
r? : ROLES \\
d? : DOMAINS \\
\hline
n? \in \mathrm{dom}\ db \land \\
(\exists_1 i : 1 \mathinner{..} \#(db(n?)) \bullet ((db(n?))(i)).name = dn? \land ((db(n?))(i)).role = r? \land \\
db' = db \oplus \{n? \mapsto ((1 \mathinner{..} i) \upharpoonright db(n?)) \frown \langle d? \rangle \frown ((i \mathinner{..} \#(db(n?))) \upharpoonright db(n?))\})
\end{array}
$$

This operation is like the previous one, the new column is inserted after the given one. In such a way, it is possible to add a column at the end of the schema. With the previous operation new column can be added at the beginning of the schema.

Here, we end with schema operations and begin with tuple operations on the database instance (relation instance).

2. INSTANCES

TUPLES

$$
\begin{array}{l}
DBSchema \\
n : RNAMES \\
t : DOMAINS \nrightarrow VALUES \\
\hline
\mathrm{dom}\ t = \mathrm{ran}\,(db(n)) \land \\
(\forall i : 1 \mathinner{..} \#(db(n)) \bullet t((db(n))(i)) \in ((db(n))(i)).domain)
\end{array}
$$

Every tuple is a function between schema domains and values. The relation name is needed to identify relation schema. Tuple values have to be members of the corresponding domain of the schema.

RELATIONS

$$n : RNAMES$$
$$instance : \mathbb{F}\ TUPLES$$
$$\forall t : instance \bullet t.n = n$$

Relation instance is a finite set of tuples possibly not empty. All the relation instance tuples have the same schema — schema of *n*.

DBInstance

$$DBSchema$$
$$instance : RNAMES \pfun RELATIONS$$
$$\mathrm{dom}\ instance = \mathrm{dom}\ db$$

The database instance is a partial function between relation names and its instances.

DBInstanceInit

$$DBInstance$$
$$\mathrm{dom}\ instance = \emptyset$$

Initially, the database instance is empty.

DBSize

$$\Xi DBInstance$$
$$n? : RNAMES$$
$$size! : \mathbb{N}$$
$$size! = \#((instance(n?)).instance)$$

This Z-schema returns the size of relation *n*? the number of its tuples.

DBRelationSchemaSize

$$\Xi DBInstance$$
$$DBSRelationSchemaSize$$

This Z-schema is an upgrade, it returns relation schema size.

DBCreate

$$\Delta DBInstance$$
$$DBSAdd$$
$$\exists ins : RELATIONS \bullet ins.n = n? \land ins.instance = \emptyset \land$$
$$instance' = instance \cup \{n? \mapsto ins\}$$

This upgrade relation creation is adding an empty instance to the database instance and binding it to the newly created relation.

DBDrop

$\Delta DBInstance$
DBSRemove

$$instance' = \{n?\} \lhd instance$$

This Z-schema removes the binding between relation name and its schema, but drops the relation instance from the database instance.

DBRename

$\Delta DBInstance$
DBSRename

$$instance' = (\{n?\} \lhd instance) \cup \{new? \mapsto instance(n?)\}$$

Here is an upgrade of relation renaming the old instance, has to be bound, with the new name.

DBInsertColumn

$\Delta DBInstance$
DBSInsertColumn

$$\begin{array}{l}
\mathrm{dom}\ instance' = \mathrm{dom}\ instance \land \\
(\forall n : \mathrm{dom}\ instance \bullet n \neq n? \Rightarrow instance'(n) = instance(n)) \land \\
(instance'(n?)).n = n? \land \\
(instance'(n?)).instance = \{nt : TUPLES \mid \\
\forall u : (instance(n?)).instance \bullet nt.n = u.n \land nt.n = n? \land \\
nt.t = u.t \cup \{d? \mapsto NULL\}\}
\end{array}$$

When a new column is inserted in the schema, all tuples have to be modified to contain NULL value in the component corresponding to the new column.

DBAddColumn

$\Delta DBInstance$
DBSAddColumn

$$\begin{array}{l}
\mathrm{dom}\ instance' = \mathrm{dom}\ instance \land \\
(\forall n : \mathrm{dom}\ instance \bullet n \neq n? \Rightarrow instance'(n) = instance(n)) \land \\
(instance'(n?)).n = n? \land \\
(instance'(n?)).instance = \{nt : TUPLES \mid \\
\forall u : (instance(n?)).instance \bullet nt.n = u.n \land nt.n = n? \land \\
nt.t = u.t \cup \{d? \mapsto NULL\}\}
\end{array}$$

This Z-schema is analogous to the previous one.

SuperKeys

$$
\begin{array}{l}
DBInstance \\
n? : RNAMES \\
superKeys! : \mathbb{F}_1\,(\mathbb{F}_1\ DOMAINS) \\
\hline
n? \in \operatorname{dom} db \ \land \\
superKeys! = \{key : \mathbb{F}_1\ DOMAINS \mid \\
\quad (\forall t1, t2 : (instance(n?)).instance \bullet \\
\quad\ (\forall ka : key \bullet ka \in \operatorname{ran}(db(n?)) \ \land \\
\quad\ t1.t(ka) = t2.t(ka) \Leftrightarrow t1 = t2))\}
\end{array}
$$

This Z-schema returns the set of all superkeys. The superkey uniquely identifies all tuples in the relation.

Keys

$$
\begin{array}{l}
DBInstance \\
n? : RNAMES \\
keys! : \mathbb{F}_1\,(\mathbb{F}_1\ DOMAINS) \\
\hline
\exists_1\, superKeys : \mathbb{F}_1\,(\mathbb{F}_1\ DOMAINS) \bullet \\
\quad SuperKeys[superKeys\,/\,superKeys!] \ \land\ keys! \subseteq superKeys \ \land \\
\quad (\forall k : keys! \bullet \neg\,(\exists subset : superKeys \bullet subset \subset k))
\end{array}
$$

Here, keys are defined as superkeys that do not contain own subset, that is, superkey.

*DBInsert*0

$$
\begin{array}{l}
\Delta DBInstance \\
n? : RNAMES \\
t? : TUPLES \\
\hline
t?.n = n? \ \land\ t? \notin (instance(n?)).instance \ \land \\
\operatorname{dom} instance' = \operatorname{dom} instance \ \land \\
(\forall n : \operatorname{dom} instance \bullet n \neq n? \Rightarrow instance'(n) = instance(n)) \ \land \\
(instance'(n?)).n = n? \ \land \\
(instance'(n?)).instance = (instance(n?)).instance \cup \{t?\}
\end{array}
$$

With this Z-schema, new tuple is added to the relation instance, but without any constrain check.

CHECKInsert

$$
\begin{array}{l}
\Xi DBInstance \\
n? : RNAMES \\
t? : TUPLES \\
\hline
(\exists_1 keys : \mathbb{F}_1(\mathbb{F}_1\ DOMAINS) \bullet Keys[keys/keys!] \ \land \\
\quad (\forall key : keys \bullet (\forall ka : key \bullet t?.t(ka) \neq NULL)))
\end{array}
$$

In this Z-schema, preliminary check on the new tuple is done, this tuple has to have no NULL-values in key attributes.

$$DBInsert \mathrel{\widehat{=}} CHECKInsert \fatsemi DBInsert0$$

This is finally the operation insert with a preliminary check on the tuple.

*DBDelete*0

$\Delta DBInstance$
$n? : RNAMES$
$keyValue? : DOMAINS \pfun VALUES$

$\mathrm{dom}\ instance' = \mathrm{dom}\ instance \land$
$(\forall n : \mathrm{dom}\ instance \bullet n \neq n? \Rightarrow instance'(n) = instance(n)) \land$
$(\exists_1 t : (instance(n?)).instance \bullet t.t = keyValue? \land$
$(instance'(n?)).n = n? \land$
$(instance'(n?)).instance = (instance(n?)).instance \setminus \{t\})$

Here, a tuple is simply deleted by key values.

CHECKDelete

$\Xi DBInstance$
$n? : RNAMES$
$keyValue? : DOMAINS \pfun VALUES$

$(\exists_1 keys : \mathbb{F}_1(\mathbb{F}_1\ DOMAINS) \bullet Keys[keys/keys!] \land \mathrm{dom}\ keyValue? \in keys) \land$
$NULL \notin \mathrm{ran}\ keyValue?$

Preliminary check of delete tuple operation requires used key values not to be NULL-ones.

$$DBDelete \mathrel{\widehat{=}} CHECKDelete \fatsemi DBDelete0$$

Finally, this is the operation delete tuple by key values.

*DBUpdate*0

$\Delta DBInstance$
$n? : RNAMES$
$keyValue? : DOMAINS \pfun VALUES$
$attsValue? : DOMAINS \pfun VALUES$

$\mathrm{dom}\ instance' = \mathrm{dom}\ instance \land$
$(\forall n : \mathrm{dom}\ instance \bullet n \neq n? \Rightarrow instance'(n) = instance(n)) \land$
$(\exists_1 t : (instance(n?)).instance \bullet t.t = keyValue? \land$
$(instance'(n?)).n = n? \land$
$(instance'(n?)).instance = (instance(n?)).instance \setminus \{t\} \cup$
$\{u : TUPLES \mid u.n = n? \land u.t = t.t \oplus attsValue?\})$

This operation is tuple update using key values to identify it, but without any checks.

CHECKUpdate

$\Xi DBInstance$
$n? : RNAMES$
$keyValue? : DOMAINS \pfun VALUES$
$attsValue? : DOMAINS \pfun VALUES$

$(\exists_1 keys : \mathbb{F}_1(\mathbb{F}_1\ DOMAINS) \bullet Keys[keys/keys!] \land \operatorname{dom} keyValue? \in keys \land$
$(\forall k : keys \bullet k \cap \operatorname{dom} attsValue? = \emptyset)) \land$
$NULL \notin \operatorname{ran} keyValue?$

This is the preliminary check before update tuple operation. It is not permitted to update key attributes and key values used to identify the tuple have to be no NULL.

$$DBUpdate \mathrel{\widehat=} CHECKUpdate \fatsemi DBUpdate0$$

Finally, here is the update tuple operation with the checks.

CONCLUSION

What more can be done with this specification? Originally, in [1] relational algebra is specified as operations on RMD, so this specification could be extended to support it.

Acknowledgements. This research is supported by the Project 240/2010 — Development of Grid infrastructure for research and education, funded by the Scientific Research Fund of University of Sofia.

REFERENCES

1. *Codd E. F.* A Relational Model of Data for Large Shared Data Banks // CACM. 1974. V. 13, No. 4. P. 377–387.

2. ISO/IEC 13568 : 2002 (E) Information Technology. Z Formal Specification Notation. Syntax, Type System and Semantics. www.iso.org

3. *Baluta D. D.* A Formal Specification in Z of the Relational Data Model, Version 2 of E. F. Codd. Concordia Univ., Montreal, Quebec, Canada, 1995. P. 129.

Received on August 9, 2010.