МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2010
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2010
Proceedings of the Thirty Ninth Spring Conference of
the Union of Bulgarian Mathematicians
Albena, April 6–10, 2010

FORMAL SPECIFICATION OF RELATIONAL MODEL OF DATA IN Z-NOTATION*

Vladimir Dimitrov

The aim of this paper is to specify formally the Relational Model of Data. This specification could be extended to cover Object-Relational Model of Data and Data Streams.

1. Introduction. In this paper, a Relational Model of Data (RMD) is specified following its original presentation given in [1].

Z-notation is used as formal notation. It is now an international standard [2].

The original presentation of RDM is not formalized, nor detailed or consistent. It contains many open topics, which have been solved later in different ways. 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.

2. Schemas. The main types of this specification are:

$$(RNAMES, CNAMES, VALUES)$$

where *RNAMES* is the set of all possible relation names, *CNAMES* is the set of all possible column names, and *VALUES* is the set of all possible tuples component values.

The set of domains is specified as a set of finite non-empty sets of values:

$$DOMAINS == \square_1\ VALUES$$

The relation schema is non-empty finite set of values:

$$SCHEMA == \mathsf{seq}_1\ DOMAINS$$

Note that, domains are non-empty, finite set of values and schemas are non-empty, finite sequences of domains. In such a way, the problems that arise from endless domains and schemas are overcome (such ones exist in relational calculus). In the reality, computers and applications are finite.

Database schema is defined as a partial function from the relation names into schemas:

*2000 Mathematics Subject Classification: 68U35.
Key words: Relational Model of Data, Z-Notation, Formal Specification.
This work was sponsored by the Ministry of Education and Science of Bulgaria, National Foundation "Science and Research", contract BY-TH-202/2006: "Multimodal biometric analysis: methods and algorithms".

$$
\begin{array}{|l}
\hline
\textit{DBSchema} \\
db : \textit{RNAMES} \pfun \textit{SCHEMA} \\
\hline
\end{array}
$$

Database schema describes the state of all relational schemas.

The initial state of database schema is a function with an empty domain:

$$
\begin{array}{|l}
\hline
\textit{DBSchemaInit} \\
\textit{DBSchema} \\
\hline
\mathrm{dom}\ db = \emptyset \\
\hline
\end{array}
$$

With other words, there is no relation names bounded with schemas.

The database schema state is changed when new relation schemas are added; old ones are removed or updated:

$$
\begin{array}{|l}
\hline
\textit{DBSAdd} \\
\Delta \textit{DBSchema} \\
n? : \textit{RNAMES} \\
s? : \textit{SCHEMA} \\
\hline
n? \notin \mathrm{dom}\ db \land \\
db' = db \cup \{n? \mapsto s?\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{DBSRemove} \\
\Delta \textit{DBSchema} \\
n? : \textit{RNAMES} \\
\hline
n? \in \mathrm{dom}\ db \land \\
db' = \{n?\} \ndres db \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{DBSUpdate} \\
\Delta \textit{DBSchema} \\
n? : \textit{RNAMES} \\
s? : \textit{SCHEMA} \\
\hline
n? \in \mathrm{dom}\ db \land \\
db' = db \oplus \{n? \mapsto s?\} \\
\hline
\end{array}
$$

This is a very broad approach. In [1] such operations are not discussed. Note, when a relation is created in an RMD implementation (i.e. its schema is added and an empty relation instance is created), it is done in just the same way as it is specified here. The same is when the relation is removed, but not when a relation schema is updated. Relation schema is updated when domains are added, removed or changed. These operations impact the current relation instance. Relation schema update is a very heavy operation and could be conflicting with some other RMD operations, which are not introduced here. Relational schema update, as it is specified above, is at a very high level and does not create any problems.

3. Relation instance. The instance of a relation is a set of tuples, that is why, the set of all possible tuples has to be specified:

$TUPLES == \mathrm{seq}_1\ VALUES$

The tuple is a non-empty sequence of values.

The relation instance is a finite set of values:

RELATION

$DBSchema$
$n: RNAMES$
$r: \mathbb{F}\ TUPLES$

$n \in \mathrm{dom}\ db \land$
$(\forall t: TUPLES \mid t \in r \Rightarrow \#t = \#(db(n)) \land (\forall i: 1..\#t \mid t(i) \in db(n)(i)))$

At any moment, including initial one, the relation could be empty:

RELATIONInit

$RELATION$
$n?: RNAMES$

$n? \in \mathrm{dom}\ db \land$
$n = n? \land r = \emptyset$

Tuples could be added or removed to/from relation instance, but their components have to be members of the corresponding domains of the relation. This condition has to be satisfied by every relation instance, how it is specified in the above Z-schema *RELATION*. When tuples are added or removed from a relation instance, they have to be checked in advance "Is the tuple possible tuple to that relation?", and this is done by the supporting Z-schema *CHECK* that is used later in the corresponding Z-schemas that add and remove tuples:

CHECK

$db: RNAMES \to SCHEMA$
$n: RNAMES$
$t: TUPLES$

$\#t = \#(db(n)) \land$
$(\forall i: 1..\#t \mid t(i) \in db(n)(i))$

Reaction on incompatible tuples is not defined, because it is not a part of RMD, but it is a part of its implementation.

Here, only operations, that add and remove tuples are specified, because update of key attributes is not permitted, and the concept of key is still not introduced. Tuple updates are implemented as removed (old one) and then add a tuple (updated one). More precisely, a tuple to be removed it is enough to have on input only its key attributes values, but the whole tuple could be used instead. With introduction of concept of key, this could be corrected.

And now these operations are introduced as follows:

RInsert

$\Delta RELATION$

$t?: TUPLES$

$CHECK(t? \,/\, t) \land t? \notin r \land$

$r' = r \cup \{t?\}$

RDelete

$\Delta RELATION$

$t?: TUPLES$

$CHECK(t? \,/\, t) \land t? \in r \land$

$r' = r \setminus \{t?\}$

4. Database Instance. Database instance is a partial function from relation names into relation instances:

DBInstance

$instance: RNAMES \pfun RELATION$

$\forall n: RNAMES \bullet n \in \mathrm{dom}\ instance \Rightarrow n = (instance(n)).n$

Here, in the invariant is specified that every relation instance has to have a schema. In other words, there is no relation instance without schema defined in advance.

Initially, database instance is empty:

DBInstanceInit

$DBInstance$

$instance = \emptyset$

There are two operations on the database instance: add and removed relation instance:

DBAdd

$\Delta DBInstance$

$n?: RNAMES$

$r?: RELATION$

$n? \notin \mathrm{dom}\ instance \land$

$n? = r?.n \land$

$instance' = instance \cup \{n? \mapsto r?\}$

DBRemove

$\Delta DBInstance$

$n?: RNAMES$

$n? \in \mathrm{dom}\ instance \land$

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

These operations are, in reality, supporting ones, because an empty relation instance is added together with the relation schema, and relation schema is removed together with relation schema removal. This is specified with Z-schemas:

DBSCreate

$\Delta DBInstance$

$DBAdd$

$n?: RNAMES$

$n? \notin \mathrm{dom}\ instance \land$

$(\exists r: RELATION \lhd n? = r.n \land r.r = \varnothing \land instance' = instance \cup \{n? \mapsto r?\})$

DBSDrop

$DBSRemove$

$DBRemove$

Relation instance is updated at lower level when tuples are added and removed. Here, this operation is presented at higher level with the supporting operation DBUpdate that update the relation instance as a whole:

DBUpdate

$\Delta DBInstance$

$n?: RNAMES$

$r?: RELATION$

$n? \in \mathrm{dom}\ instance \land$

$n? = r?.n \land$

$instance' = instance \oplus \{n? \mapsto r?\}$

5. Column names. Column names are introduced in [1], but they are supporting concept, because they play only an informative role. A column name has not to be unique in the relation schema. It is not a problem, because the columns (domains) order is fixed.

Column names are a finite, non-empty sequence of names:

$COLUMNS == \mathrm{seq}_1\ CNAMES$

Named relation schema is nothing else than relation schema with its column names:

$$
\begin{array}{|l}
\hline
NSCHEMA \\
\hline
s : SCHEMA \\
ns : COLUMNS \\
\hline
\#\, ns = \#\, s \\
\hline
\end{array}
$$

Named relation schema has no particular value, but in above Z-schemas *NSCHEMA* could be used instead of *SCHEMA* resulting a RMD with named schemas. RMD and relational algebra do not use column names as defined in [1].

4. Conclusion. Further detailing of this specification is an addition of operations for adding, removing and updating of tuple using the concept of key. For completeness, above presented operations could be updated with named schemas.

REFERENCES

[1] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *CACM*, **13** (June, 1974), No 4, 377–387.

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

[3] D. D. Baluta. A Formal Specification in Z of the Relational Data Model, Version 2 of E. F. Codd, Concordia University, Montreal, Quebec, Canada, March 1995.

Vladimir Todorov Dimitrov
Faculty of Mathematics and Informatics
University of Sofia
5, James Bourchier Blvd
1164 Sofia, Bulgaria
`e-mail: cht@fmi.uni-sofia.bg`

ФОРМАЛНА СПЕЦИФИКАЦИЯ НА РЕЛАЦИОННИЯ МОДЕЛ НА ДАННИ В Z-НОТАЦИЯ

Владимир Димитров

Целта на настоящия доклад е формалната спецификация на релационния модел на данни. Тази спецификация след това може да бъде разширена към Обектно-релационния модел на данни и към Потоците от данни.