On The Formal Specification of Database Schema Constraints

Ivan Luković

University of Novi Sad,

Faculty of Technical Sciences

Novi Sad, Serbia and Montenegro

ivan@uns.ns.ac.yu

Sonja Ristić

Higher Business School

Novi Sad, Serbia and Montenegro

sdristic@uns.ns.ac.yu

Pavle Mogin

Victoria University of Wellington,

School of Mathematical and Computing Sciences

Wellington, New Zeland

pmogin@mcs.vuw.ac.nz

Abstract: A database schema is a formal and abstract definition of data and constraints of the real system. In order to formulate the formal criteria of "well-designed" database schema at schema level of abstraction, the precise formal specification and the classifications of different types of constraints are needed. The formal specification of constraints could lead towards a transaction program and application specifications that are fully independent of a chosen programming and run-time environment, which is highly important in order to preserve investments in a software development. In the paper we suggest one possible classification of the database constraints, in the relational data model, and present common characteristics of the different constraint types. An attention is paid to the importance of precise specifications of the constraints and their interpretation.

Keywords: Integrity Rule, Constraint, Transaction Program Specification

1 Introduction

A database schema is a formal and abstract definition of data and constraints of the real system that should faithfully represent real system and its business rules. A set of constraints is the mandatory component of the structure of a database schema. Database constraints arise from the written or habitual behaviour rules, or business rules that hold in the real system. Database management system (DBMS) is used to implement database constraints, so as to keep the database instance consistent with a current state of the real system. A common formal specification of the different constraint types could be an important formalism leading towards the well-designed database schema.

A transaction program issues queries and updates that are executed by a DBMS. DBMS would reject a transaction that violates any of the constraints embodied in the database (db) schema.

Specifying transaction programs is an important and time-consuming methodological task in the process of the design of an information system (IS). The design specification of a transaction program (program specification) is a formalized description of that program. It is aimed at supporting the implementation of an end user business task that is defined by means of a user request. It is usually assumed that the transaction programs will be executed against a database. Accordingly, the basic structural components of a transaction program specification are: (i) A specification of the human-computer interface; (ii) A data definition; and (iii) A formal description of a data processing procedure. The data definition part of the transaction program specification is a formal and abstract definition of data, constraints, and database update activities that are needed to make a transaction program. Establishing a formal specification of different constraint types in a precise manner is an important task, enabling a design of the program specification which would be applicable in practice.

The aim of the paper is to suggest a general specification of the database constraints and to make a classification of the database constraints in the relational data model.

Apart from the Introduction and Conclusion, the paper has four sections. Section two formally introduces a common formal specification of the database constraints. In the third section a classification of the db constraints is presented. Some constraint types in the relational data model that conform to the introduced common specification are presented in section four.

2 A Common Specification of Database Constraints

A relational database schema is a pair (S, I), where S is a set of relation schemes and I is a set of interrelation constraints.

Each relation scheme from S is a named pair: $N_i(R_i, C_i)$, where N_i is a unique name, R_i is an attribute set, and C_i is a specification of the constraints. A relation scheme will be often referred simply by its name N_i. A set of database constraints is denoted by O , where $O = I \cup \left(\bigcup_{N_j \in S} C_j \right)$ holds.

Different constraint types are defined in the relational data model, such as: domain integrity constraint, uniqueness integrity constraint, tuple integrity constraint, null-value integrity constraint, referential integrity constraint, etc. Let as denote the constraint type with *ConTyp*, where:

$$ConTyp \in \{\text{Dom, Key, Unique, RefInt, UserDef } \ldots\} \quad (1)$$

Each constraint type *ConTyp* has the following characteristics:

- Notification formalism (constraint type definition);
- Interpretation rule (validation rule);
- *Definition scope - a logical structure of* ***attributes*** embodied in the constraint type definition;
- *Validation scope - a logical structure of* ***data*** necessary to validate a constraint of the *ConTyp* type;
- *A set of critical operations* – a set of database operations which could cause the violation of the constraint of the *ConTyp* type; and
- The set of possible *activities* of preserving database consistency. They are applied to preserve data consistency during an attempt of the constraint violating by means of some *critical operation*.

Definition 1. *A definition scope* of a constraint o, denoted by T(o) is a nonempty set of four-tuples:

$$(N_m, \rho_m, At_m, \{(op_m^{i_m}, act_m^{i_m}) \mid i_m \geq 1\}) \quad (2)$$

where:

– N_m is the name of a relation scheme that is spanned by o;

– $\rho_m \in \{referenced, referencing, ...\}$ is the role of N_m in o;

– At_m is a set or sequence of attributes from R_m that are *relevant* for o. o is used to check values of attributes form At_m; and

– $\{(op_m^{i_m}, act_m^{i_m}) \mid i_m \geq 1\}$ is a set of pairs (*critical operation*, *activity*), where for the critical operation $op_m^{i_m}$, $op_m^{i_m} \in \{insert, delete, update\}$ holds. $act_m^{i_m} \in \{NoAction, Cascade, SetDefault, SetNull\}$ is an activity for preserving data consistency during an attempt of the constraint violating by means of the critical operation $op_m^{i_m}$. □

Definition 2. Let o be a constraint. *The specification of the constraint* o is a four-tuple:

$$(ConTyp, T(o), ConCon, ConIntR) \tag{3}$$

where:

- *ConTyp* ∈ {Key, Unique, τ, RefInt,...} is a constraint type;
- *T*(o) is a definition scope of the constraint o;
- *ConCon* is a *logical condition of the constraint*, that database instance must satisfy; and
- *ConIntR* is an interpretation rule for user-defined constraint types, denoted as UserDef. □

Components *ConTyp* and *T*(o) are mandatory, while *ConCon and ConIntR* are optional.

Example 1. Let us consider a database schema (*S*, I).

$S = \{INVOICE(R_1, C_1), PARTNER(R_2, C_2)\}$;
- $R_1 = \{InvId, Invdate, PartId, Total\}$;
- $C_1 = \varnothing$;
- $R_2 = \{PartId, PartName, PartAdrr\}$;
- $C_2 = \varnothing$;
- $I = \{INVOICE[PartId] \subseteq PARTNER[PartId]\}$.

One possible formal specification of the constraint o ∈ I is: (RefInt, *T*(o), Δ, Δ), where:

T(o) = {(*INVOICE*, *referencing*, *PartId*, {(*insert*, *NoAction*), (*modify*, *NoAction*)}), (*PARTNER*, *referenced*, *PartId*, {(*delete*, *Cascade*)}.

A symbol Δ stands for non-specified components. □

3 A Classification of The Constraint Types

The constraint types could be classified based on:

- Definition scope; and
- Validation scope.

According to the definition sope, constraint type may be defined as:

- Single-relation constraint;
- Multi-relation constraint; and
- Out-of-relation constraint.

A *single-relation* constraint is defined over exactly one database relation scheme. Consequently $|T(o)| = 1$and the role of relation schema N_1 from $T(o)$ is not specified, i.e. $\rho_1 = \Delta$.

A *multi-relation* constraint is defined over more than one, not necessarily different database relation schemes. Namely, $T(o)$ contains at least two elements.

A *out-of-relation* constraint is not defined over any database relation scheme. Instead, it is defined over an initially existing or predefined domain (data type).

According to the validation scope, constraint type may be defined as:

- Value constraint;
- Tuple constraint;
- Relational constraint; and
- Inter-relational constraint.

A *value constraint* is validated over just one value from the domain of some attribute.

A *tuple constraint* is validated over exactly one tuple of a database relation instance.

A *relational constraint* is validated over the set of tuples of a database relation instance.

An *inter-relation constraint* is validated over the sets of tuples, extracted from different database relation instances, or from the same relation instance, but having more than one role, according to the definition scope of the constraint $T(o)$.

4 Formal Specifications of Some Constraint Types in The Relational Data Model

The constraint types in the relational data model may be defined as: the domain integrity constraint, the attribute value constraint, the tuple integrity constraint, the extended tuple integrity constraint, the uniqueness integrity constraint, the key constraint, the inclusive dependency constraint, the extended inclusive dependency constraint, the selective inclusive dependency constraint, the selective extended inclusive dependency constraint and the other user defined integrity constraints. Concerning the inclusive dependency constraints, two special kinds of those constraints could be notified: the referential integrity and the inverse referential integrity constraint. For both of them, the extended, the selective and the selective-extended variations exist, too.

In the following sections, the formal specification of some of the mentioned constraint types would be presented, as an illustration of well-specified constraint types.

4.1 The Domain Integrity Constraint

Informally, a domain of an attribute A is a set of such values that can be assigned to A. This set can be specified by defining a *data type*, a *maximal data length* and a *logical condition* that each constant, which is supposed to be a domain value, has to satisfy. There are two classes of domains: *predefined* (*primitive*) and *user defined*. Let Dp denote a set of predefined domains. It is represented by means of implicitly defined, initially existing data types in a DBMS. For example, Dp = {*Character, Integer, Real, Logical, Date,...*}. User defined domain is explicitly specified by means of a previously specified domain, using the inheritance principle.

Definition 3. Let D be a set of all the domains in a system, such that $Dp \subseteq D$ holds. Let D be a domain from D. A *domain integrity constraint* id(D) is a triple:

$$\mathrm{id}(D) = (\mathrm{Typ}(D), \mathrm{Len}(D), \mathrm{Con}(D)), \tag{4}$$

where Typ(D) is a *super domain* (*type constraint*), Len(D) is a *maximal length constraint* and Con(D) is a *logical condition*. If any component in (1) is not specified for a domain $D \in D$, it will be denoted with an "empty" symbol Δ. □

Domain constraint for a predefined domain $D_p \in Dp$ is not explicitly specified. Thus, $\mathrm{Typ}(D_p) = \Delta$, and $\mathrm{Con}(D_p) = \Delta$ holds, for each $D_p \in Dp$. It is supposed that there are some predefined domains in a DBMS for which $\mathrm{Len}(D_p)$ has to be specified (for example *Character*). Otherwise, $\mathrm{Len}(D_p) = \Delta$ holds.

A specification of the type constraint Typ defines a relation of *domain inheritance* $\text{Typ} \subseteq \text{D}^2$ in the following way. $(D_i, D_j) \in \text{Typ}$ iff $D_j = \text{Typ}(D_i)$. D can be considered as *well-defined set* of domains, if the graph (D, Typ) is an acyclic structure such that:

$$(\forall D_i \in \text{D})(\neg(\exists D_j \in \text{D})((D_i, D_j) \in \text{Typ}) \Rightarrow D_i \in \text{Dp})) \quad (5)$$

holds. In other words, if D is a well-defined set, then type constraint of a new domain is always specified by using a previously specified domain from D. For each domain, after a finite number of times the relation Typ is applied, a predefined domain must be reached. In this paper, we suppose that D is a well-defined set of domains.

For each domain $D_i \in \text{D} \setminus \text{Dp}$, $\text{Len}(D_i)$ is specified according the following rules. $\text{Len}(D_i) = \Delta$, iff $\text{Typ}(D_i) \in \text{D} \setminus \text{Dp}$, or $\text{Typ}(D_i) \in \text{Dp}$ and $\text{Len}(\text{Typ}(D_i)) = \Delta$. Otherwise, $\text{Len}(D_i) \leq \text{Len}(\text{Typ}(D_i))$, iff $\text{Typ}(D_i) \in \text{Dp}$ and $\text{Len}(\text{Typ}(D_i)) \neq \Delta$.

A logical condition $\text{Con}(D)$ is composed of atomic conditions, related by the logical operators $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, and $\neg$. Each atomic condition is of the form $d\ \theta\ k$, $k\ \theta\ d$, or $d \in \{k_1, ..., k_n\}$, where d is a formal variable, representing a value of the type, $k_1, ..., k_n$ and k are the constants and $\theta \in \{<, >, \leq, \geq, \neq, =\}$. Complete definitions of an atomic condition and a condition $\text{Con}(D)$ can be found in [4] and [6].

A domain constraint $\text{id}(D)$ may be validated for any value d_i. A validation is denoted as $\text{id}(D)(d_i)$, $\text{id}(D)(d_i) \in \{\top, \bot\}$ must hold, where $\top$ stands for *true*, and $\bot$ stands for *false*. In the following text, validation rules for $\text{id}(D)(d_i)$ are presented.

- If d_i is a *null value* (*missing value*), which is denoted by $d_i = \omega$, then $\text{id}(D)(d_i) = \top$. Thus, a null value satisfies each domain constraint by default.
- Validation rules for each $D \in \text{Dp}$ are embedded in a DBMS and no further attention is paid to them.

If $D \in \text{D} \setminus \text{Dp}$, then a validation $\text{id}(D)(d_i)$ is defined by the expression:

$$\text{id}(D)(d_i) = \text{id}(\text{Typ}(D))(d_i) \wedge \text{Len}(D)(d_i) \wedge \text{Con}(D)(d_i). \quad (6)$$

- A validation $\text{id}(\text{Typ}(D))(d_i)$ is performed using the same rules on a domain specified by $\text{Typ}(D)$ in a recursive manner.
- $\text{Len}(D)(d_i)$ is validated only if $\text{id}(\text{Typ}(D))(d_i) = \top$ holds. If $\text{Len}(D) = \Delta$, then $\text{Len}(D)(d_i) = \top$. If $\text{Len}(D) \neq \Delta$, then $\text{Len}(D)(d_i) = \top$ iff the length of d_i is not grater than the specified $\text{Len}(D)$.
- $\text{Con}(D)(d_i)$ is validated only if $\text{Len}(D)(d_i) = \top$ holds. Besides, it is supposed that $\text{Con}(D)$ is built in such a way that all the constants from $\text{Con}(D)$ satisfy both $\text{Typ}(D)$ and $\text{Len}(D)$. If $\text{Con}(D) = \Delta$, then $\text{Con}(D)(d_i) = \top$. If $\text{Con}(D) \neq \Delta$, then $\text{Con}(D)(d_i) = \top$ iff d_i satisfies the logical condition $\text{Con}(D)$, which is transformed in such a way that d_i replaces all the occurrences of a formal variable d. All the relational and logical operators are validated in a usual way.

The relation Typ $\subseteq D^2$ allows defining a notion of the domain chain. Let $Ld(D) = (D_n, ..., D_1)$, $n \geq 2$, be a sequence of domains, where $D, D_1, ..., D_n \in D$ and $D \notin Dp$. $Ld(D)$ is called a *domain chain* for D, if $D_1 \in Dp$, $D_n = D$, and $(\forall i \in \{2, ..., n\})(Typ(D_i) = D_{i-1})$ holds. The following lemma presents a rule how to validate the domain constraint of a user-defined domain. The proof may be found in [6].

Lemma 1. Let D be a user defined domain and let $Ld(D) = (D_n, ..., D_1)$ be a domain chain for D. Then:

$$id(D)(d_p) = id(D_1)(d_p) \wedge Len(D_2)(d_p) \wedge (\bigwedge_{i=2}^{n} Con(D_i)(d_p)) \quad (7)$$

holds. □

A domain integrity constraint type, defined in such a way is an out-of-relation, value constraint type. $T(o)$ is a single element set of the form $T(o) = \{(\Delta, \Delta, Typ(Len), \{(insert, act_1^{i_1}), (update, act_2^{i_2})\})\}$. A logical condition $Con(D)$ is appeared in the specification as *ConCon*.

4.2 The Domain Integrity Constraint of An Attribute in a Universal Set

Let a universal relation scheme (U, C) be given, where $U = \{A_i \mid i \in \{1, ..., n\}\}$ is a universal set of attributes and C is a set of global constraints. Let D be a set of all domains and let $id(D) = \{id(D) \mid D \in D\}$ denote a set of all domain constraints, where $id(D) \subseteq C$ holds. Each constraint from id(D) is specified independently of any attribute from U.

A domain function $Dom_U: U \rightarrow D$ associates each attribute $A \in U$ with a domain from D. Consequently, Dom_U associates the corresponding domain constraint $id(D)$ with A. Let $Dom(U, A)$ denote a domain D for which $Dom_U(A, D)$ holds. Then a domain constraint $id(Dom(U, A))$ will be called the *domain constraint of an attribute in a universal set of attributes*. $id(Dom(U, A))$ may be validated for an attribute A value for each tuple $t \in Tuple(U)$, which is denoted as $id(Dom(U, A))(t[A])$. Validation is performed according to rules for $id(D)$, presented in Lemma1.

4.3 The Domain Integrity Constraint of An Attribute in a Relation Scheme

We suppose that a database schema (S, I) is designed by a decomposition of a universal scheme (U, C), and that it should be logically equivalent to (U, C).

Therefore, the equality of attribute sets $U = \bigcup_{N_j \in S} R_j$ and the equivalence of integrity constraints $C \equiv I \cup \left(\bigcup_{N_j \in S} C_j \right)$ should hold.

Let an attribute $A \in U$ belong to two different relation schemes N_i and N_k from S. It may be supposed that the domains for A in R_i and R_k need not be the same, but both of them should be the subsets of a domain, defined by Dom(U, A). Therefore, a domain function Dom_{R_j}: $R_j \rightarrow D$ is introduced to associate each attribute $A \in R_j$ with a domain from D. Let Dom(N_j, A) denote a domain D for which $Dom_{R_j}(A, D)$ holds. The function Dom_{R_j} must be specified in such a way that Dom(U, A) $\in$ Ld(Dom(N_j, A)) holds, for each A in any N_j from S. It follows that the logical implication id(Dom(N_j, A)) $\models$ id(Dom(U, A)) must hold. The proof of the statement may be found in [6].

Domain constraint id(Dom(N_j, A)) will be called the *domain constraint of an attribute in a relation scheme*. id(Dom(N_j, A)) may be validated for an attribute A value for each tuple $t \in Tuple(R_j)$, which is denoted as id(Dom(U, A))(t[A]). A validation is performed according to rules presented in Lemma 1.

A domain constraint of an attribute in a relation scheme, defined in such a way is also an out-of-relation, value constraint type. T(o) is a single element set of the form T(o) = {(N, Δ, {A}, {(*insert*, $act_1^{i_1}$), (*update*, $act_2^{i_2}$)}). The specification of a logical condition *ConCon* is of the form Dom(N, A).

4.4 The Tuple Integrity Constraint

Let us first introduce the notions of (i) a null value constraint, and (ii) an attribute value constraint, and then formulate a tuple integrity constraint.

For an attribute A one may expect that null values are allowed in one, but disallowed in an other relation scheme. This possibility can additionally restrict a set of allowed values for an attribute in a relation scheme. Let Null(N_j, A) be a logical parameter, which denotes a *null value constraint* of an attribute A in a relation scheme N_j. Thus, Null(N_i, A) $\in$ {$\perp$, $\top$} holds. If it is Null(N_i, A) = $\top$, null values for A in an instance over N_j are allowed. Otherwise, if Null(N_i, A) = $\perp$, null values are disallowed.

In certain cases, a value of Null(N_j, A) may be a consequence of the appropriate null value constraint Null(U, A) in a universal relation scheme only. However, a value of Null(N_j, A) may also depend of the existence of the other constraints, such as keys.

A null value constraint can be validated for any value from the domain of an attribute $A \in R_j$. Validation of a value d_i, which satisfies id(Dom(N_j, A)), is denoted by Null(N_j, A)(d_i). Validation rules are defined as follows:

- Null(N_j, A) = T $\Rightarrow$ Null(N_j, A)(d_i) = T; and
- Null(N_j, A) = $\perp$ $\Rightarrow$ ($d_i = \omega \Leftrightarrow$ Null(N_j, A)(d_i) = $\perp$).

Specifications of a domain constraint of an attribute in a relation scheme and a null value constraint together build a new integrity constraint type, which is called the attribute value constraint. Let $\tau(N_j, A)$ be an *attribute value constraint*. It is a structure given in the following way:

$$\tau(N_j, A) = (\text{Dom}(N_j, A), \text{Null}(N_j, A)). \quad (8)$$

Both components of $\tau(N_j, A)$ are mandatory.

A validation rule for an attribute value constraint is defined as follows: $\tau(N_j, A)(t[A]) = \text{id}(\text{Dom}(N_j, A))(t[A]) \wedge \text{Null}(N_j, A)(t[A])$, where $t[A]$ is an A value of a tuple t over the set of attributes R_j. The set of all values satisfying $\tau(N_j, A)$ is denoted as $Domain(N_j, A)$.

Some attribute values from a tuple over R_j may be, in some cases, logically related. This kind of a relationship is modeled by a logical condition, denoted as Con(N_j). Con(N_j) is composed of atomic conditions, related by the logical operators $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, and $\neg$. Each atomic condition is of the form $\alpha_1 \theta \alpha_2$, where α_1 and α_2 are the terms and $\theta \in \{<, >, \leq, \geq, \neq, =\}$. A term is defined recursively. A term is either (i) a constant $k \in Domain(N_j, A)$, where $A \in R_j$, or (ii) an attribute $A \in R_j$, or (iii) a function $f(\alpha_1,...,\alpha_n)$, where $\alpha_1,...,\alpha_n$ are previously defined terms and f is an n-ary function that is computable on the level of a single tuple over R_j. Each term is obtained by applying aforementioned rules a finite number of times. Con(N_j) may also be an "empty" condition. In this case, it is marked with a symbol Δ. Con(N_j), defined in such a way, is called *the logical condition of a relation scheme*. Complete definitions of an atomic condition and a condition Con(N_j) can be found in [4] and [6].

If a tuple $t \in Tuple(R_j)$ satisfies the attribute value integrity constraint for each $A \in R_j$, then it may be used to validate a logical condition of a relation scheme Con(N_j). A validation is denoted as Con(N_j)(t). Validation is a logical function, i.e. Con(N_j)(t) $\in$ {T, $\perp$} must hold, for each tuple t from the set $\{t \in Tuple(R_j) \mid (\forall A \in R_j)(\tau(N_j, A)(t[A]) = \text{T})\}$. Validation rules are defined as follows.

- If Con(N_j) = Δ, then Con(N_j)(t) = T.

If Con(N_j) $\neq \Delta$, then each attribute A in Con(N_j) is replaced by a corresponding value $t[A]$. After that, the values of all the functions in Con(N_j) are calculated. Finally, Con(N_j)(t) = T will hold iff t satisfies such a transformed condition, where relational and logical operators are validated in a usual way.

Definition 4. A *tuple integrity constraint* $\tau(N_j)$ over a relation scheme N_j is a pair:

$$\tau(N_j) = (\{\tau(N_j, A) \mid A \in R_j\}, \mathrm{Con}(N_j)), \tag{9}$$

where $\{\tau(N_j, A) \mid A \in R_j\}$ is a set of attribute value constraints and $\mathrm{Con}(N_j)$ is a logical condition of a relation scheme. □

A tuple integrity constraint $\tau(N_j)$ may be validated for any tuple t over the set of attributes R_j, which is denoted as $\tau(N_j)(t)$. A validation rule is defined by the logical expression:

$$\tau(N_j)(t) = (\bigwedge_{A \in R_j} \tau(N_j, A)(t[A])) \wedge \mathrm{Con}(N_j)(t), \tag{10}$$

where $t \in Tuple(R_j)$.

By analyzing the syntax rules used to formulate $\mathrm{Con}(N_j)$ and $\mathrm{Con}(D)$, it should be noticed that there are examples of logical constraints, which can be equivalently expressed by means of both $\mathrm{Con}(N_j)$ and some conditions of the type $\mathrm{Con}(\mathrm{Dom}(N_j, A))$, $A \in R_j$. In the same way, it is possible to equivalently express the constraint $\mathrm{Null}(N_j, A) = \bot$, by incorporating the expression $A \neq \omega$ into $\mathrm{Con}(\mathrm{Dom}(N_j, A))$ and leaving $\mathrm{Null}(N_j, A) = \top$. In order to avoid such ambiguousness, we suppose that $\mathrm{Con}(N_j)$ is built in such a way that it expresses only those constraints that could not have been expressed by means of any domain constraint conditions $\mathrm{Con}(\mathrm{Dom}(N_j, A))$, $A \in R_j$. Besides, we suppose that if $\mathrm{Null}(N_j, A) = \top$ holds for an attribute $A \in R_j$, then there must be a tuple $t \in Tuple(R_j)$ for which $\mathrm{Con}(N_j)(t) = \top$ and $t[A] = \omega$ hold. If $\mathrm{Con}(N_j)$ satisfies both aforementioned conditions, it is called a *well-defined condition*. More details may be found in [6].

A tuple integrity constraint, defined in such a way is a single-relation, tuple constraint type. $T(\mathrm{o})$ is a single element set of the form $T(\mathrm{o}) = \{(N, \Delta, R, \{(insert, act_1^{i_1}), (update, act_2^{i_2})\})$. The specification of a logical condition *ConCon* is of the form $\mathrm{Con}(N)$.

4.5 The Uniqueness Integrity Constraint

Definition 5. Let a relation scheme $N_j(R_j, \mathrm{C}_j)$ be given. A *uniqueness integrity constraint* is the expression of the form $\mathrm{Unique}(N_j, X)$, where $X \subseteq R_j$ holds. □

A uniqueness integrity constraint $\mathrm{Unique}(N_j, X)$ may be validated for any relation r_j defined over the attribute set R_j, which is denoted as $\mathrm{Unique}(N_j, X)(r_j)$. Uniqueness integrity constraint is satisfied iff the following two conditions hold:

$$(\forall t_i, t_k \in r_j)((\forall A \in X)(t_i[A] \neq \omega \wedge t_k[A] \neq \omega) \Rightarrow (t_i[X] = t_k[X] \Rightarrow t_i = t_k)), \tag{11}$$

$$(\forall X' \subset X)(\neg(11)). \tag{12}$$

Let the set K_j of all keys of a relation scheme $N_j(R_j, \mathrm{C}_j)$ and a set of attributes $X \subseteq R_j$ be given. According to the definitions of a uniqueness constraint and a key constraint, it follows that the equivalence $(\mathrm{Unique}(N_j, X) \wedge (\forall A \in X)(\mathrm{Null}(N_j, A) = \bot)) \equiv X \in \mathrm{K}_j$ holds ([6]). Hence, we suppose that the set of all relation scheme

uniqueness constraints $\text{Uniq}(N_j) = \{X \subseteq R_j \mid \text{Unique}(N_j, X)\}$ contains only those X for which $(\forall K \in \text{K}_j)(K \not\subseteq X)$ holds.

A uniqueness constraint, defined in such a way is a single-relation, relational constraint type. More details, concerning aforementioned constraint types may be found in [4] and [6].

Conclusions

One of the main problems in a database schema design is how to define and formally specify a set of constraints that faithfully represents a real system and its business rules. The overall aim is to implement all defined constraints on the database server by the DBMS mechanisms. This approach would enable centralization of the consistency control. This control may not be overcome, neither by any program nor by an end-user using the database operations. In this paper, a common specification of the constraints is introduced in order to reach an aforementioned overall aim. By distinguishing between definition and validation scopes, a criteria for constraint classification is established.

References

[1] Codd, E. F. (1990): *The Relational Model for Database Management Version 2*, Addison-Wesley-Publishing-Company, USA

[2] Date, C. J. (1994): *A Guide to the SQL Standard*, Addison-Wesley Publishing Company, USA

[3] Luković, I.; Mogin P. (2000): *On The Role of Subschema as A Component of The Implementation Specification of A Program*, VI Symposium on Computer Science and Information Technologies YUINFO, Kopaonik, Yugoslavia, Proceedings on CD ROM

[4] Mogin, P.; Luković, I.; Govedarica M. (2000): *Database Design Principles*, University of Novi Sad, Faculty of Technical Sciences & MP "Stylos", Novi Sad, Yugoslavia

[5] Mogin, P.; Luković, I. (1999): *An Approach to Database Design*, International Journal of INDUSTRIAL SYSTEMS, Vol. 1, No. 2, Novi Sad, Yugoslavia 59-68

[6] Ristić, S. (2003): *A Research of Subschema Consolidation Problem*, PhD Thesis, University of Novi Sad, Faculty of Economics, Subotica, Yugoslavia

[7] Luković I., Mogin P., Govedarica M., Ristić S., "The Structure of A Subschema and Its XML Specification", in *Proceedings of the XIII International Conference on Information and Intelligent Systems*, Varaždin, Croatia, September 2002, pp. 45-56.

[8] Ristić S., Mogin P., Luković I., "Specifying Database Updates Using A Subschema", in *Proceedings of the VII International Conference On Intelligent Engineering Systems INES 2003*, March 4-6, 2003, Assiut-Luxor, Egypt, pp. 203-212.