Journal of
Information Technology & Software Engineering

Mancas, J Inform Tech Softw Eng 2015, 5:3
DOI: 10.4172/2165-7866.1000e125

Editorial Open Access

On the Paramount Importance of Database Constraints

Christian Mancas*

Department of Computer Science, Bucharest Polytechnic University, Romania

Introduction

Plausible and implausible data and constraints.

Databases (dbs) store data of interest in schemas made out generally of three components: for relational ones, a set of fundamental tables (corresponding to actual object sets - e.g., people, organizations, countries, cities, products, etc.), their columns (corresponding to object set properties - e.g., first name, last name, SSN, birth place, birth date, organization name, type, country name, code, etc.), and a set of constraints (e.g., first and last names are compulsory, birth date should be between Jan. 1st 1900 and today, country names and codes are unique, etc.). The two above components make up the structure of a scheme, while the third does not allow storing any data into this structure, but only desirable one. For graph dbs, the structure is made out of nodes and edges between them.

Some constraints are embedded in the scheme; for example, the fact that a table COUNTRIES has one column Capital restricts countries from having more than one capital. However, these embedded ones are not enough and the constraint sets should contain explicit ones corresponding to all of the actual business rules that are governing the corresponding subuniverse. For example, there may not be either two countries having same names, or two states of a same country having same names, or two people having same SSN, or countries or states without names, etc.

Not adding to a db scheme one constraint corresponding to a business rule of that subuniverse allow storing implausible data. For example, if Capital is not declared as unique (one-to-one) then a same city may be stored for several countries, although, actually, no city may simultaneously be the capital of more than one country.

Dually, adding to a db scheme a constraint which is implausible (or dictatorial, aberrant, i.e., not corresponding to an existing business rule governing that subuniverse) prevents storing plausible data. For example, if you declare StateName as unique (one-to-one) in a STATES table, from the two existing states called "La Rioja" (from Spain and Argentina) only one can be stored at any given moment in time.

To conclude with, all existing business rules should always be enforced by corresponding constraints (for banning implausible data) and no implausible constraint should ever be enforced (for not banning plausible data).

Constraints exist independently of whether we know and/or ignore them: only adding all existing ones in all db schemes may guarantee data plausibility. "Garbage in, garbage out" applies in this context too if there is at least one missing constraint from a db scheme.

Note that, generally, there is no such thing as correct data: for example, almost nobody knows exactly what is the population of a country in any particular moment. However, it is implausible that a country may have less than 800 (Vatican, the smallest one from this point of view, had 839 according to its latest census of July 1, 2012) or more than 2,000,000,000 (China, the most populated one, had 1,372,780,000 at October 23, 2015, for 2030 the UN projection is that India will come first with more than 1,500,000,000, and you should leave room for another 50 years of db service, for example). Consequently, plausibledata range for a Population column of COUNTRIES is (8*102, 2*109).

When enforced, constraints are always satisfied by (or hold in) any corresponding db instance, as any attempt to violate them is rejected. When not enforced, some instances may violate (or do not satisfy) them.

A db instance is said to be consistent (valid) if it satisfies all (table/db) constraints; otherwise, it is called inconsistent (invalid). Obviously, db owners and users would always like to store only consistent (valid) instances.

Coherent and Incoherent Constraints and Constraint Sets

Given any db and its associated set of constraints C, C is coherent(with respect to the db scheme) if for any table of the db there is at least a non-void instance that satisfies C and incoherentotherwise.

For example, if a db scheme contains a MOUNTAIN_PEAKS table and constraints C_1: Altitude≥ 1000 and C_2: Altitude< 1000 (or, equivalently, constraint C: Altitude≥ 1000 andAltitude< 1000), then the only possible instance for MOUNTAIN_PEAKS is obviously the empty set, which means that the corresponding set of constraints C is incoherent.

Constraint C is obviously incoherent and so is any constraint set containingboth C_1 and C_2 above.

Trivially, all constraint sets should be coherent and from any incoherent set a coherent one may be obtained by removing all incoherent constraints and one of any pair of contradicting constraints (C_2 in the above example, as it is implausible).

Fundamental, Redundant, and Trivial Constraints. Minimal Sets of Constraints

Constraints are first order logic formulas[1]; as such, a constraint or a set of constraints may imply other constraints as well: a set of constraints C implies a constraint c if c holds in all instances in which C holds (dually, C does not imply a constraint c if there is an instance for which c does not hold, but C holds).

The standard logical notation for formulas implication is C⊢c; c is called an implied constraint. For example, constraint set {Altitude> 1000, Altitude< 2550} implies constraint Altitude ∈ (1000, 2550); trivially, the vice-versa is also true. Constraints that are not implied are called fundamental, while those implied are called redundant[2].

[1]In particular, closed ones, that is formulas whose variable occurrences are bound to at least one logic quantifier (be it "for any" or "there is"). Dually, open formulas have at least one occurrence of a variable free (that is not bound to any logic quantifier), which, in dbs, are formalizing queries.

[2]In fact, redundant constraints are more precisely defined through the concepts of constraint set closures and equivalences (see, for example, [1]).

***Corresponding author:** Christian Mancas, Department of Computer Science, Bucharest Polytechnic University, Romania, Tel: +40722357078; E-mail: christian.mancas@gmail.com

Received November 08, 2015; **Accepted** November 13, 2015; **Published** December 02, 2015

Citation: Mancas C (2015) On the Paramount Importance of Database Constraints. J Inform Tech Softw Eng 5: e125. doi:10.4172/2165-7866.1000e125

Constraint sets that do not contain any redundant constraint are called minimal.

A particular case of redundant constraints are the trivial ones: constraints that hold in any db instance (i.e., are implied by an empty constraint set). For example,C': Altitude ≥ 1000 or Altitude < 1000 is obviously trivial.

We should never declare implied constraints (be them redundant or trivial) in db schemas: it would only be superfluously time consuming to enforce them too, although this is never actually needed. Consequently, all db constraint sets should be minimal. Testing for constraint sets equivalence and/or constraints' redundancy can be reduced to the well-known implication problem: given any constraint set C and constraint c, does C imply c? (for example, does C= {S⊆T, T⊆U}imply S⊆U? does it imply T⊆S too?).

Note that, depending on the constraint classes, this problem may be solved very quickly (that is linearly), very slowly (that is exponentially), or be impossible to solve (even undecidable[3]).

The Five Basic Relational Constraint Types

Any relational database management system (RDBMS) [2] providesthe following five relational constraint types: (co-) domain (range), not-null (mandatory, compulsory, required, totality), key (minimal uniqueness), foreign key (referential integrity, typed inclusion), and check (tuple). Unfortunately, RDBMSes do not impose to their users declaring any constraint in the dbs they manage. Consequently, you are free to not declare any. In practice, fortunately, almost all db schemas also include some constraints. Unfortunately, very rarely do they include at least all needed relational constraints.

Domain (Range) constraints

Domain (range) constraintsspecify plausible data ranges for column values.

Programmers are acquainted with assigning data types to their variables. DBMSes also provide data types for table columns. Declaring such a type for a column is sometimes enough, for example: BOOLEAN, NUMBER(n), VARCHAR2(n). For most of the time it is not: even if, by definition, all computer data types are finite, they are generally huge and most of their values are implausible for actual db instances.

For example, if you let column BirthDate of a table EMPLOYEES to store DATE values, depending on the DBMS version, users might store birth dates either from year -100 or 10,000, both of them being trivially implausible.

This is why, generally, column values should be restricted to plausible subsets of type [minValue, maxValue] ⊂ data Type. For example, Birth Date above could be restricted to [1/1/1930, SysDate() – 365 * 18] ⊂ DATE (as it is implausible to have, for example in 2015, employees older than 85 and younger than 18).

NOT NULL constraints

Very often, it is the case that some values are not applicable or, at least temporarily, are unknown. For example, although we might not know temporarily the altitude of some mountain peaks, we would however want to store at least their names and the mountains to which they belong. Conventionally, the empty values that are stored, for example, in the column Altitude of such rows are called nulls (or null values).

[3]A problem is said to be undecidable if it is neither provable, nor refutable or for which it is impossible to design an algorithm that always (that is, in any context) correctly answers to its questions (see, for example, [1]).

The ANSI report [3] distinguishes between 14 types of nulls; most of us consider, however, that there are only three basic types (the rest being either particular cases of these three or only due to implementation considerations): non-existent (inapplicable), (temporarily) unknown, and no-information (the logical or between the first two: it is not known whether such a value exists or, if it were existing, nothing is known on its nature). For example, in a column Manager Bonus of a SALARIES table, only managers will have not null data stored, while for all other employees there will be non-existent (inapplicable) nulls.

Consequently, in dbs we have to accommodate with null values too. However, constraints should also be placed on their usage: what would be the meaning of a table row having only nulls? Obviously, at least one column per each table should not accept nulls: not-null (mandatory, compulsory, required, totality) constraints are used to specify such columns.

Please note that there is great misunderstanding on null values: some DBMSes (e.g., MS SQL Server) wrongly assume that there is only one null value, so columns accepting nulls are not allowed in unique constraints; others (IBM DB2, Oracle, and even MS Access) assume correctly that there is a countable infinite number of nulls (so they accept non not null columns into keys).

Key (Uniqueness) constraints

In any subuniverse, there may be both objects whose uniqueness is interesting and objects that we do not need to uniquely identify. For example, in the first category there are people (with their unique e-mail addresses, phone numbers, SSNs, etc.), while in the second are the chairs of a room/apartment/building. In dbs we should always care for uniqueness: for example, if the db should store detailed information on each chair, then they have to be uniquely labeled, e.g., with an inventory number, and these numbers stored for each chair in a CHAIRS table; if not, then we would probably abstract a CHAIR_TYPES table and for each of its rows we would store a unique type name (plus description, total number of chairs per type or, in other tables, per type and room/apartment/building, if needed).

A key constraintis a statement of the type "C_1• … •C_n key", where n>0 is a natural, C_i are columns of a table T, "•" denotes concatenation[4] (which is, generally, omitted) of these columns and key means minimally unique[5] that is it is unique and it does not include any other key. When n = 1, the key is calledsimple; when n> 1 it is calledconcatenated; if a unique column concatenation properly contains a key (that is the included key has smaller than n arity), then it is not a key (as it is not minimal), but a superkey; note that superkeys are of no actual, but only theoretical interest. For example, Capital is a simple key of COUNTRIES, Capital • Population is a superkey, while State Name • Country is a concatenated key of STATES.

Only keys need to be declared, not superkeys[6]: for example, if instead of SSN you declare SSN • Birth Year as a key, then implausible data might be stored (e.g., any number of persons having same SSN, but different birth dates); if both Capitaland Capital • Population are declared as keys, then no implausible data might be stored, but updates to COUNTRIESwould be slower (and the db would need additional disk and memory space to store and process the superkey Capital•Population), as the system would enforce the superkey too.

[4]Mathematically, this means mapping (Cartesian) product

[5]That is minimally one-to-one (see, for example, [1]).

[6]Unfortunately, most RDBMSes (including, for example, Oracle, MS SQL Server and Access, etc.) allow for enforcement of both keys and superkeys!

For any table, you can declare one of its keys as being primary. One of the most important best practice rules in this field is to add to any fundamental table a key integer column (generally having values automatically generated by the DBMS and called ID) with no other meaning than uniquely identifying its rows (called, as such, a surrogate key) and declare it as the primary key.

Please note that uniqueness is not an absolute, but a relative property: in some contexts a column or concatenation of columns are unique, while in others (even in a same subuniverse) they are not. For example, zip codes are unique within a country, but internationally they are not: you need to concatenate them with the country to which they belong in order to be uniquely identifiable.

Keys, just like any other type of fundamental (that is not derived) constraints may only be discovered and declared by humans: there may not ever be any tools, be them hardware, software, conceptual, etc., able to do such a job. Moreover, especially keys are not at all easy to discover: besides their relativeness, as the number of columns increases, the number of their products (theoretically very many of them being possible keys) increases exponentially. This is why [4-6] present algorithms for assisting their discovery.

Referential integrity (Foreign key) constraints

Links between tables, as well as those between a table and itself are done by foreign keys: pointer-type columns whose values should always be among the values of the corresponding referenced columns. For example, in the above COUNTRIES table, Capitalshould be a foreign key referencing some CityID key from table CITIES. Associated constraints are called referential integrities[7] (or (typed) inclusion dependencies).

Foreign keys should have associated domain constraints too exactly matching those of the referenced columns. For example, if CityID is a NUMBER (6) then Capital should also be a NUMBER(6). However, note that, unfortunately, RDBMSes do not automatically derive corresponding domain constraints; even worse, you are sometimes free to oversize and even undersize foreign keys domain constraints.

The referential integrity constraint between foreign key f and corresponding referenced column g it's generally denoted by $f \subseteq g$[8]. Obviously, this is a notational abuse[9]: in fact, its meaning is $Im(f) \subseteq Im(g)$, where Im is the image operator, which computes the set of values taken by its operand (that is the set of all of the values taken by the corresponding mapping). Failing to enforce such constraints may result in storing the so-called dangling pointers: values that point to non-existing values in the set of values of the referenced columns.

Referential integrity constraints might be violated from both sides of the inclusion mathematical relation: as just seen above, from the left one by storing in a foreign key column a value which does not belong to the referenced column, but, dually, from the right one too, by deleting from a table a row which is referenced by at least one row.

The Relational Data Model (RDM) [7-9], allows for concatenated foreign keys and, moreover, for foreign keys referencing any columns, not necessarily being keys. Best practice, however, is to only use single foreign keys, always referencing primary keys, which should always be surrogate-type appropriately range restricted numeric ones.

Note that "foreign" in the "foreign key" syntagm is a false friend: foreign keys may reference a key from the same table (not from a "foreign" one). For example, the "foreign" key Reports To of table EMPLOYEES references its ID column. Moreover, "foreign key" is a double false friend: "key" is a false friend too, as, generally, foreign keys are not also (unique) keys.

For example, in a table COUNTRIES also containing a foreign key Currency (referencing the ID column of a table CURRENCIES), Capital is both a key and a foreign key, Country name is a key but not a foreign key, Currency is a foreign key but not a key (as there are, for example, lot of countries having Euro as currency), whereas Population is neither a key nor a foreign key.

The following Table 1 is summarizing these findings, which prove that the concepts of unique and foreign keys are orthogonal to each other.

Tuple (Check) constraints

For check (tuple) constraints there is no generally agreed definition; from most of the RDBMS implementations, they are first order logic formulas that have to be satisfied by all rows of a table and have (by notational abuse) a simplified, propositional logic type form: they are using parenthesis, the logical not, and, and or operators connecting terms having the form CθD, where all such C and D (from all terms) are columns of a same table, and θ is a standard operator[10]. For example, for a table COMPOSERS, check constraint BirthYear + 3 <PassedAwayYear states that, for each row, corresponding life duration should have been at least 3 years (assuming that no one could ever compose before he/she is at least 3 years old).

In fact, check/tuple constraints are first order logic formulas with only one variable universally quantified, which, by notational abuse, is omitted. For example, the above apparently propositional calculus formula stands for (∀x∈COMPOSERS) (PassedAwayYear(x) — Birth Year(x) between 3 and 120).

The limitation to only one table is essential: no RDBMS is recognizing (and, consequently, enforcing), for example, constraints like Pop Cnstr: (∀x∈COUNTRIES) (∀y∈STATES) (STATES. Country(y)=x⇒COUNTRIES. Population(x) ≥ STATES. Population(y)).

Note that you should not get confused because of theory and RDBMS terminologies: for example, domain (range) and not-null constraints are also considered by Oracleas being check ones.

Non-Relational Constraint Types

As just seen above, the five relational constraint types are not

Column	*Key?*	*Foreign key?*
Population		
Currency		⌉
Country Name	⌉	
Capital	⌉	⌉

Table 1: Summarizes about key and Foreign Key.

[7]Note that some RDBMSes use confusing terminology in this respect too: for example, MS Access table "design" mode considers referential integrity as being only half of this constraint type, the other half being enforceable through the "limit to list" property of foreign keys.

[8]This being the reason why RDM theory also refers to these constraints as being (typed) inclusion dependencies.

[9]Columns being mappings, it is senseless to talk about mapping inclusions.

[10]The set of db standard operators always include the corresponding math ones, plus some additional ones, either derived from them, like between ... and, or generally coming from regular expressions manipulation, as like (which uses, just like in OSes, meta-characters too: ‚%' or ‚*' for any character string, including empty ones, etc.).

enough: constraints like PopCnstrabove should also be enforced in order to reject storing implausible data. For example, the Elementary Mathematical Data Model (EMDM) [6,10-12] is providing more than 30 types of constraints without which data modeling may not be accurate and, consequently, the corresponding dbs are prone to storing implausible data. For example, it allows declaring that the auto-mapping (column) composition Country°Capital should be reflexive (i.e., (∀x∈COUNTRIES)(Country(Capital(x))=x)), which formalizes the constraint "the capital of any country should be a city of that country". Obviously, if such a constraint is not enforced, users may store in a table COUNTRIES, for example, that Toronto is the U.S. capital, Bucharest is the U.K. one, a.s.o.

Only MatBase [5, 6 ,13], a prototype DBMS implementing both the EMDM, the RDM, and the Entity-Relationship Data Model (E-RDM) [14,15], is currently providing such constraints to its users. However, all such constraints should also be discovered, aggregated in a non-relational constraint set associated to the corresponding db schema, and implemented even for relational dbs(rdbs) [16] too, either as RDBMS triggers (using their extended SQL) or as trigger-type methods written in high level programming languages embedding SQL of the software applications that manage corresponding rdbs.

Discovering all such constraints is even harder than discovering key ones. However, EMDM is providing dedicated assistance algorithms too that are guiding its users in this task [6,11]. Moreover, EMDM is also providing algorithms for guaranteeing the coherence and minimality of constraint sets [6,17].

Conclusion

Constraints are of paramount importance in dbs, as they are the only ones that prevent storing implausible data in the corresponding db instances. Besides the constraints embedded in the db scheme structure, a set of explicit constraints corresponding to all and only to the business rules governing the corresponding subuniverse should always be added to a db schema. Such sets should always be coherent and minimal.

RDBMSes provides five relational constraint types: (co-) domain (range), not-null (mandatory, compulsory, required, totality), key (minimal uniqueness), foreign key (referential integrity, typed inclusion), and check (tuple). However, they do not impose to their users declaring any constraint in the dbs they manage. Best practice rules were provided on when and how to use them intelligently.

Unfortunately, these five types of constraints are not enough for accurate data modeling and db design. EMDM provides more than 30 types of constraints badly needed even for very simple db schemes and MatBase, a prototype DBMS implements all of them. When using RDBMSes to manage dbs, the non-relational constraints should also be declared and enforced, either as RDBMS triggers (using their extended SQL) or as trigger-type methods written in high level programming languages embedding SQL of the software applications that manage corresponding rdbs.

EMDM is also providing assistance algorithms that are guiding its users in the process of discovering all existing constraints in any subuniverse of discourse, as well as algorithms for guaranteeing the coherence and minimality of constraint sets, which are implemented in MatBase.

References

1. Mancas C (2015) Conceptual Data Modeling and DB Design. A Fully Algorithmic Approach. I: The Shortest Available Path, Apple Academic Press, NJ.
2. Garcia-Molina H, Ullman JD, Widom J (2014) Database Systems. The Complete Book, 2nd eds. Pearson Education Ltd.: Harlow, U.K.
3. Gorman MM ANSI/X3/SPARC Study Group on Database Management Systems (1975) Interim Report 75: 02-08. FDT-Bulletin ACM SIGMOD 7.
4. Mancas C, Crasovschi L (2003) An Optimal Algorithm for Computer-Aided Design of Key Type Constraints. Proc 1st Balkan Inf Techn BIT Conf Aristotle University Press, Thessaloniki, Greece.
5. Mancas C, Mancas S (2005) MatBase E-R Diagrams Subsystem Metacatalog Conceptual Design. Proc IASTED DBA Conf on DB and App., Innsbruck, Acta Press, Canada.
6. Mancas C (2016) Conceptual Data Modeling and DB Design. A Fully Algorithmic Approach: Refinements for an Expert Path, Apple Academic Press, NJ.
7. Codd EF (1970) A relational model for large shared data banks. CACM 13: 377-387.
8. Abiteboul S, Hull R, Vianu V (1995) Foundations of Databases. Addison-Wesley: Reading, MA.
9. Ullman JD, Widom J (2007) A First Course in Database Systems, 3rd ed. Prentice Hall: Upper Saddle River, NJ.
10. Mancas C (1985) A first introduction in data model based on the elementary theory of sets, relations and functions. Proc. INFO-IAŞI 1: 314-320 Al Cuza University, Iasi, Romania.
11. Mancas C (2002) On Modeling Closed Entity-Relationship Diagrams in an Elementary Mathematical Data Model. Proc. ADBIS, Slovakia 2: 165-174, Slovak Polytechnic Univ., Bratislava, Slovakia.
12. Mancas C (2002) On Knowledge Representation Using an Elementary Mathematical Data Model. Proc. IASTED IKS, Acta Press, Canada.
13. Mancas C, Dragomir S (2004) MatBaseDatalog¬ Subsystem Meta-catalog Conceptual Design. Proc. IASTED Intl. Conf. Softw. Eng. And App., MIT, Cambridge, MA, USA, 34 - 41, Acta Press, Calgary, Canada.
14. Chen PP (1976) The entity-relationship model: Toward a unified view of data. ACM TODS 1: 9-36.
15. Thalheim B (2000) Fundamentals of Entity-Relationship Modeling. Springer-Verlag, Berlin.
16. Date CJ (2013) Relational Theory for Computer Professionals: What Relational Databases are Really All About. Theories in Practice; O'Reilly Media, Inc.: Sebastopol, CA.
17. Mancas C, Dragomir S, Crasovschi, L (2003) On Modeling First Order Predicate Calculus Using the Elementary Mathematical Data Model in MatBase DBMS. Proc. IASTED DBA, 1:1197-1202, Acta Press, Calgary, Canada.