Relational Model and Missing Information

Siniša Jakovljević
Pension and Disability Insurance Fund, Republic of Srpska
jaksi@teol.net

Contribution to the State of the Art
DOI: **10.7251/JIT1201032J** UDC: **005.5:004.032.6**

Abstract: This paper examines possibilities offered by relational model when using missing information. The overview is conducted and possibilities which occur in practial use were analyzed. The use of predicates in which missing values occur has also been analyzed. Possible effects on system performance have been indicated.

Keywords: relational model, missing information, ***null***, three value logic (3VL), integrity, relational operators.

Introduction

More than 40 years after it was revealed, the relational model still exists in its full capacity on the database scene. Behind the amount of data which is stored for both current needs and long-term storage of important data, there are some systems for Relational Database Management System (RDMBS). Modern RDBMS are based on the implementation of the relational model proposed in the early 70s by E.F. Codd [1]. This document has significantly changed the world of database. Simplicity and understandability of the relational model has enabled it to be generally accepted. The close relationship of the model with the perception of the real world was a key factor of success and user's commitment to it, which by its use, model has achieved. The conception, according to which database users should be freed from the knowledge of the internal data presentation on computers, has opened a wide space for parallel development of technology for internal data storage and technology for access and use of this data. In addition to commercial solutions, many free-of-charge and open-source solutions are present on the market. Even leaders in the commercial segment of the market offer very powerful versions which are fully free-of-charge, while functionality is completely preserved. Differences in relation to the commercial version of the same manufacturer are primarily related to certain limitations in the size of the database, and unavailabilty of additional software tools, which the work with data make more improved and efficient.

Relational Model

The relational data model presents an abstract data theory. It is based on proven mathematical aspects, primarily on the set theory and first-order predicate logic [8]. The original model was consisted of three main components: structure, semantics and manipulation. The structure base is formed by relations. Usually, a graphic relation is presented in the form of a two-dimensial table. Basically, the relation is a mathematical determinant for the table of special type [4].

Relation attributes: Relations are defined by their attributes whose range of values is determined by the domain (type) of those attributes. The domain of a certain attribute represents a set of values which that particular attribute is able to accept. Presentation of the relation in the form of the two-dimensional table represents by itself a simplified approach. The level of relation is determined by the number of at-

tributes that relation contains. Suppose we have relation R which has two attributes, which are presented as two columns in the table. Then, it is a binary relation R. Generally, relation R with an n attribute is viewed as an n-ary, where n ≥ 1. Its presentation consists of the display of the table with n columns. Each presentation has heading and body. The relation heading consists of a set of attributes and their domains, while the relation body consists of a set of tuples which meet the heading structure. Each relation has at least one candidate key which by means of its value uniquely determines one and only one tuple of a given relation.

In the relational model, *Integrity* has a special significance. Integrity is achieved at the following levels: domain, entity and referential. Fulfilling integrity conditions at each of the mentioned levels is presented by the result of a certain logic operation whose result must be evaluated as true. At the domain level, integrity is fulfilled in a way that the value of a certain relation attribute must be within the range which is determined by the type of data for that attribute in particular. For example, if it is stipulated that the relation attribute takes values of a numeric type, then the transaction will be completed and data will be stored into the database if and only if that requirement is fulfilled. Also, if it is stipulated that a certain relation attribute is of an alphanumeric type with the length of n characters, then the transaction of data storage into the database will be successfully completed if and only if the attribute value is presented by alphanumeric characters, and the length is ≤ n.

At the entity level, integrity is fulfilled by defining rules in the form of primary key. Each entity could have more unique sets of attributes which can serve as a candidate key. In accordance with the requirements that are to be addressed, the information system designer can freely choose a set of attributes which will uniquely identify the tuple in that particular entity. Such a set of attributes is called the primary key and refers to the relation, rather than individually to a certain tuple . [7].

Referential integrity is established between the relations. If a certain value appears in one context, then it has to appear in another related context [5]. More formally: if the value of the attribute A of the relation R_1 references the value of the attribute B of the relation R_2, then the value of the attribute B of the relation R_2 must exist.

MISSING INFORMATION

Later, by expanding the relational model [2], the way in which the basic relational model relates to the missing information and how it treats that information is presented. What is missing?

As in the real world, it is possible to expect that at the moment of data collection, the value of a certain data will be unknown. It is very rational to plan the possibility that the database enables later data input, at the moment when it is available. Manipulative possibilities for missing information have predicted a three value logic. Truth is treated in the form of three possible values (3VL - three value logic). Truth values, formulated by Boolean algebra, true or false, are supplemented with a new truth value: unknown. This concept has been criticized immediately. It was suggested that the model solves the problem on the basis of previously accepted practice, old-fashioned, but proven method: instead of the missing value, a constant can be input, whose value according to the needs, is determined by a designer. For numeric data, 0 or 1 can be used taking into account the impact of a constant on arithmetic operations which can be performed, i.e. whether to add or multiply a constant will have an influence on the final result. Bearing in mind these types of cases and possible consequences, the old approach does not leave an impression of a secure solution. Therefore, in [3], the validity of approach by means of *unknown* value is argumented, combining that kind of approach with logic and algebra. The concept which is compatible with logic or algebra is taken over: by solving equations, we do not make use of variable values which at specific stages can be changed in time, but we solve equations by the use of logic.

Two essential questions are pointed out:

- Which type of information is missing?
- What is the reason for information missing?

Regarding the first question, it is possible that one part of a tuple is missing, e.g. date of birth of an employee. Also, it is possible that a complete set of data of an employee is missing. Model must deal with both situations. Regarding the second question, it is possible that at the moment of data collection a certain value is not available, but it will be added to the database immediately after becoming known, without any consequences. However, it is also possible that the data can not be related to the entity structure, meaning that the data in particular is not applicable in a given structure. In this case, we have a situation where a complete tuple is missing. Solution is achieved by introducing markers which will at that place hold an empty space for data storage, which meets the requirements of entity semantics. This marker is called the *null* or *null-value*. If the value is known, it will be stored, if not, a space for the storage will be reserved until the storage is possible. The space for the missing value is left, but the problem of manipulation of the missing value in operations that are performed on the data (arithmetic operations with numeric data, manipulation with character data, etc.) appears. Suppose that in the data about the employee we need to use the data about his/her total employment service. After one year of employment service has passed, total employment service must be increased by 1. This operation is trivial, but its result has a significant impact on the employee. What to do if the initial data about the employment service is missing? Situation arises: *null* + 1 = ? What is the solution to the equation? 1? Maybe! 15? Maybe! But maybe is not the answer to the question. The correct answer would be – total value is unknown. It is known that the result would be of a numeric type, but nothing else is known. The solution will be known once the previous value is known, the missing value. A similar situation occurs when the missing value appears in the alphanumeric values. What is the result in the following situation: *null* + 'BCD'? 'BCD'? Maybe! 'ABCD'? Maybe! But maybe is not the answer to this question either.

The missing values in the context of integrity maintenance are a specific problem. Making reference to an unknown value can not be performed until it becomes known. But then, it is not a missing value anymore. Introducing an unknown value into the primary key is opposite to the relational concept. By establishing the primary key over the relation, a firm rule of integrity is being established, thus, the use of an unknown value in the primary key is excluded. The situation is similar for the referential integrity. Referencing to something that is unknown means that there is a possibility that after determining the values of previously unknown, it can happen that the referenced value does not exist. This conflicts with the integrity rule.

Three Value Logic (3VL)

Three value logic introduces a lot of order and unifies the handling of unknown values.

Comparing scalar values of which at least one is an unknown value as a result gives an unknown value. The unknown value of the logical result represents the third value of the logic truth. In the following tables, the truth values for logic operations of conjuction, disjunction and negation are given.

AND	t	u	f
t	t	u	f
u	u	u	f
f	f	f	f

OR	t	u	f
t	t	t	t
u	t	u	f
f	t	u	f

NOT	
t	f
u	u
f	t

In the tables, usual English abbreviations are used: true, unknown, false.

To illustrate the usage of truth table, let us look at an example:

Let X=1, Y=2, Z is unknown.

X > Y	AND	Y > Z	=> false
X > Y	OR	Y > Z	=> unknown
X < Y	OR	Y > Z	=> true
NOT(X=Z)			=> unknown

Null and scalar operators

Let us consider the following example:

Let the value A be unknown.

A + 1	+ A	=> unknown
A – 1	– A	=> unknown
A * 1	* A	=> unknown
A / 1	/ A	=> unknown
+ A	- A	=> unknown

It can be concluded that if at least one operand of a numeric expression is unknown, the result is an unknown value.

If an unknown value is presented with a null, by analyzing the difference or division by zero, we get intuitively an unexpected result:

null - null	=> unknown
null / 0	=> unknown

In the first case, it is obvious that the two *null* values are not treated as equal ones. They are not values but placeholders for the values which will later appear. Therefore, the result of subtraction is unknown.

In the second case, although we would expect a message to try to divide by zero, regardless of when a new value is available, still, if a divisor is *null*, the result of division by zero is unknown.

Null and relational algebra

Relational algebra as part of the relational model has a very clear development of missing information usage in the relational operations. In order to consider the impact of the missing information on the relational operations, it is necessary to bear in mind the following:

- operands of the relational operations are relations
- the result of the relational operations are relations

Projection by definition eliminates tuples that are duplicates. Hence, if a relation contains more identical tuples, meaning that all correspondent attributes of one tuple have the same values as the attributes of another tuple, the result of the projection will display only one occurence of the tuple, regardeless of their number.

For example, let us imagine the relation R, which is a ternary one, meaning that it has 3 attributes: ID#, NAME, NUMBER, and that we have 4 ternary tuples of this relation in total:

ID#	NAME	NUMBER
4001	PETER	5000
4002	PAUL	4000
4003	MARY	3000
4002	PAUL	4000

Projection π(R) produces new relation which has 3 ternary tuples in total:

ID#	NAME	NUMBER
4001	PETER	5000
4002	PAUL	4000
4003	MARY	3000

Ternary tuple of the relation R whose value is ID=4002, NAME=PAUL, NUMBER=4000, in a resulting relation is summarized in one instance of that ternary tuple.

Let us now imagine that in the same example, for ID=4002 we have missing values:

ID#	NAME	NUMBER
4001	PETER	5000
4002	PAUL	NULL
4003	MARY	3000
4002	PAUL	NULL

Applying logic where *null* is equal to nothing, it follows that NULL ≠ NULL, it would be expected that the projection π(R) in this particular case will produce the resulting relation which will contain 4 ternary tuples:

ID#	NAME	NUMBER
4001	PETER	5000
4002	PAUL	NULL
4003	MARY	3000
4002	PAUL	NULL

However, it is determinated that even those tuples that in correspondent attributes have equal values are considered duplicates or that the pairs are established, to whose values, in database, missing values are added. Also, a certain criticism is expressed in terms of removing duplicated tuples because this expanded definition of duplicates does not meet semantic conditions – *null* ≠ *null* [3].

Missing information has no influence on the relational product. [3] [4]

Restriction operation is subjected to the influence of the missing information. The resulting relation contains only those tuples for which the condition of restriction meets the condition of truth, and tuples whose truth is false or unknown are discarded. T h e relational union operation – *union*, eliminates duplicated tuples in the same manner as explained for the projection.

Expansion of the union into the operation *union all* implies that removing of duplicated tuples is not being performed. Relational difference R_1 MINUS R_2 does not include removing of duplicated tuples. Tuple N_x can appear as a result of relational difference R_1 minus R_2 only if N_x is a duplicate of a certain tuple in relation R_1, and is not a duplicate of any tuple in relation R_2.

Intersect between two relations R_1 and R_2, will be the tuple N_x if and only if N_x is a duplicate of a certain tuple and in relation R_1 and R_2.

Join relations which for the requirement of join include attributes whose value allows the existence of an unknown value, will not execute the join. However, if there is a reason for an outer join of two relations, outer relation will contain missing values for tuples which do not meet the requirement of the join, and a designer will use that circumstance in accordance with the solution of a practical problem.

The core of this practical issue is reflected in the two and three value logic. According to their different nature, it is clear that they produce significantly different results.

Null and keys

Implementations of the relational model in commercial systems for database management solve the use of *null* values in a suitable, we could even say a pleasant manner. At the level of physical and logical database design, a designer is left with a choice whether he/she will at the level of domain integrity allow *null* as a possibility or not. Realization of this rule is simple and achieved by a trivial ban of the tuple which contains *null* value of a concrete attribute.

When it comes to the primary key, it by its nature and purpose should perform a unique identification of one tuple, and by doing so, the possibility of *null* value appearing as a part of the primary key is excluded.

Simply put, strictly defined and binding.

However, it is emphasized that each entity can have alternatives for the primary key, i.e. candidate keys. This suggests that there may be different combinations which can provide unity of a certain tuple. It was pointed out that in reality, sometimes at the moment of data storage, the value of each attribute can not be obtained each time. Therefore, such a combination of attributes excludes the possibility of choice for the primary key. However, once the missing value is obtained, and which will definitely ensure unity, such a formed tuple can have its value and can ensure full unity of the tuple. Still, the primary key must be a restrictive one, previously formed and known, so that the option of subsequently fulfilling the requirements of unity will not be an option. The possibility that a certain unity value is required,

starting from the moment when the missing value is available, seems rational. ANSI SQL-92 standard prescribes that the two values are not distinct if they are *null* values and if they fulfill the requirements of the standard prescribed in the clause 8.2. of the standard ANSI SQL-92. From the moment they cease to be *null* values, there is a real chance that they can become distinct.

Based on the above mentioned, there is still a possibility to technically ensure unity, and a definite checking could be performed when a missing value becomes known. At the moment of value recognition, checking is performed whether it is a distinct one and a unique one in a set of all values of that attribute. If this condition is fulfilled, this particular value can be stored into database. For this checking, integrity checking using a *unique* key is introduced.

Implementation of this part of the ANSI SQL-92 standard varies from manufacturer to manufacturer of Database Management System.. Oracle enables establishment of a *unique* key over a certain attribute in a manner that it allows the storage of *null* value by n times. Microsoft SQL server also enables establishment of a *unique* key, but in a more restrictive manner – in one relation, only one tuple which can in a certain attribute accept only one *null* value is enabled. When that specific tuple receives its missing value of that attribute, only then some other tuple can accept *null* value in that same attribute. This example of different limitation usage points out to a large specificity of the missing values and contradictions with which the world of missing values and the relational model is filled.

Null and performance

Satisfaction of a final user of a certain applicative solution that relies on database is reflected in the speed of response at which the user, from the database gets an answer to his/her query. Modern optimizers, implemented in RDBMS to select a set of results from all data, use techniques which rely on very complex mathematical models.

In [6], it is demonstrated that the presence of *null* values does not affect the number of distinct values which are generated in the form of system and object statistics. *Null*s are ignored while creating statistics. However, if the percentage of *null*s' participation is significant, a danger may appear in the form that the optimizer incorrectly interprets data, which would result in incorrect instructions for query execution which arise as a result of optimizer's action. Execution plan of the SQL query, which is based on incorrect optimizer's assumptions, inevitably leads to the performance degradation and a longer period of query execution. In relatively static data, initial optimizer's assumptions will give solid and stable results, however, unless missing values are regularly updated, the appearance of real data instead of *null*s will affect optimizer's work.

The issue of comparing *null* with an existing value is already mentioned. The only checking related to the *null* and which can be described as a trivial is the checking whether a certain attribute has *null* instead of value. Everything except that kind of checking is related to the potentially serious problems. The manner in which the *null* in the predicates of the SQL statement should be treated is an obligation which has to be considered by a designer of the applicative solution. Practical question would be: should *null* be treated as to fulfill the condition of truth or not? The answer depends on business logic on which a designer must create solution. If it is necessary to answer the question how many employees have an income between the values of I_1 and I_2, then a serious confusion may be caused by the result which will not consider the presence of the *null*s in the income data. Does the *null* meet the condition of comparison in this particular example is a question that should be answered by business logic, used by a designer. In the SQL query itself, *null* is processed using functions which treat *null*s in a sense that if the value is unknown, some previously adopted constant is applied. Possible solution is that even *null* does not meet the requirement of comparison. In the first as well as in the second case, it is necessary to further process missing value. Further danger presents the possibility that the total result of comparison ends up being unknown. It seems practical that the *null* should be mapped in some appropriate value. This additional process takes up part of the time which is needed for the result to be produced, but still presents serious

danger which threatens from multiple angles.

As said earlier, system statistics ignore tuples which contain *null*. If the attributes with the *null*s appear in index which is B*tree organized, this implies that

table1.number_of_rows > index1(table1).number_of_rows

which can, again, produce wrong conclusions obtained by the optimizer.

Conclusion

The importance of the relational model lies in the fact that the real world is very practically mapped into a technical form. Even in life, we often have situations where something is missing, out of objective and subjective reasons. Databases are necessities of reality, and therefore, the world of missing values is a completely natural phenomenon. The manner in which the relational model has supported this part of reality is very practical and detailed. It represents a powerful framework in which a designer must adjust a way in which he/she will manage something that is at a given moment unknown. The influence of the missing can be small, insignificant, but at the same time very big. The need exists, solutions also. Maybe missing information will be banned and maybe not. The world of missing information is condemned to a lot of answers in the form of – may be.

Reference

[1] Codd, E. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM , 13* (6), 377-387.

[2] Codd, E. (1979). Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems , 4* (4).

[3] Codd, E. F. (1990). *The Relational Model for DataBase Management Version 2.* Reading, Massachusetts: Addison-Wesley.

[4] Date, C. (2004). *An Introduction to Database Systems* (8th ed.). Boston: Pearson / Addison Wesley.

[5] Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). *Database Systems - The complete book* (2nd ed.). New Jersey: Pearson Prentice Hall.

[6] Lewis, J. (2006). *"Cost-Based Oracle Fundamentals".* New York: Apress.

[7] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). *Database System Concepts, 6th ed.* New York: McGraw-Hill.

[8] Wolfram Research, Inc. (1999-2012). Retrieved from Wolfram MathWorld: http://mathworld.wolfram.com/First-Order-Logic.html

Submitted: May 08, 2012
Accepted: May 30, 2012