INCONSISTENT DATABASES

Leopoldo Bertossi
Carleton University, http://www.scs.carleton.ca/~bertossi

SYNONYMS

None

DEFINITION

An inconsistent database is a database instance that does not satisfy those integrity constraints that have been declared together with the schema of the database.

HISTORICAL BACKGROUND

Already in the classical and seminal paper by E.F. Codd [5] on the relational data model it is possible to find the notions of integrity constraint and consistency of a database. The idea of *consistent query answering*, consisting in characterizing and computing semantically correct answers to queries in inconsistent databases, was explicitly introduced in [1].

SCIENTIFIC FUNDAMENTALS

A database can be seen as a model, i.e. as a simplified, abstract description, of an external reality. In the case of relational databases, one starts by choosing certain predicates of a prescribed arity. The *schema* of the database consists of this set of predicates, possibly *attributes*, which can be seen as names for the arguments of the predicates, together with an indication of the domains where the attributes can take their values. Having chosen the schema, the representation of the external reality is given in terms of relations, which are extensions for the predicates in the schema. This set of relations is called an *instance* of the schema.

For example, relational database for representing information about students of a university might be based on the schema consisting of the predicates $Students(StNum, StName)$ and $Enrollment(StName, Course)$. The attribute *StNum* is expected to take numerical values; *StName*, character string values; and *Course*, alphanumeric string values. In Figure 1 there is a possible instance for this schema.

Students	StuNum	StuName
	101	john bell
	102	mary stein
	104	claire stevens
	107	pat norton

Enrollment	StuNum	Course
	104	comp150
	101	comp100
	101	comp200
	105	comp120

Figure 1: A Database Instance

In order to make the database a more accurate model of the university domain (or to be in a more accurate correspondence with it), certain conditions are imposed on the possible instances of the database. Those conditions are intended to capture more meaning from the outside application domain. In consequence, these conditions are called *semantic constraints* or *integrity constraints* (ICs). For example, a condition could be that, in every instance, the student name functionally depends upon the student number, i.e. a student number is assigned to at most one student name. This condition, called a *functional dependency* (FD), is denoted with $StuNumber \rightarrow StuName$, or $Students : StuNumber \rightarrow StuName$, to indicate that this dependency should hold for attributes of relation *Students*. Actually, in this case, since all the attributes in the relation functionally depend on *StuNum*, the FD is called a *key constraint*.

Students	StuNum	StuName
	101	john bell
	101	joe logan
	104	claire stevens
	107	pat norton

Enrollment	StuNum	Course
	104	comp150
	101	comp100
	101	comp200

Figure 2: Another Instance

Integrity constraints can be declared together with the schema, indicating that the instances for the schema should all satisfy the integrity constraints. For example, if the functional dependency $Students : StuNumber \rightarrow StuName$ is added to the schema, the instance in Figure 1 is consistent, because it satisfies the FD. However, the instance in Figure 2 is *inconsistent*. This is because this instance does not satisfy, or, what is the same, violates the functional dependency (the student number *101* is assigned to two different student names.

Functional dependencies are particular cases of integrity constraints. It is also possible to consider with the schema a *referential integrity constraint* that requires that every student (number) in the relation *Enrollment* appears, associated with a student name, in relation *Students*, the official "table" of students. This is denoted with $Enrollement[StNum] \subseteq Students[StNum]$. If this IC is considered in the schema, the instance in Figure 1 is inconsistent, because student *105* does not appear in relation *Students*. However, if only the referential constraint were in the schema, the instance in Figure 2 would be consistent.

In can be seen that the notion of consistency is relative to a set of integrity constraints; and when a database is said to be inconsistent, it is meant that the particular instance of the database at hand is inconsistent.

The two particular kinds of integrity constraints presented above and also other forms of ICs can be easily expressed in the language of predicate logic. For example, the FD above can be expressed by the symbolic sentence

$$\forall x \forall y \forall z((Students(x, y) \wedge Students(x, z)) \longrightarrow y = z), \tag{1}$$

whereas the referential constraint above can be expressed by

$$\forall x \forall y(Enrollment(x, y) \longrightarrow \exists z Students(x, z)). \tag{2}$$

Notice that this language of predicate logic is determined by the database schema, whose predicates are now being used to write down logical formulas. We may also use "built-in" predicates, like the equality predicate. Thus, ICs can be seen as forming a set Σ of sentences written in a language of predicate logic.

A database instance can be seen as an *interpretation structure* D for the language of predicate logic that is used to express ICs. This is because an instance has an underlying domain and (finite) extensions for the predicates in the schema. Having the database instance as an interpretation structure and the set of ICs as a set of symbolic sentences is crucial, and makes it possible to simply apply the notion of satisfaction of a formula by a structure of first-order predicate logic [6]. In this way, the notion of satisfaction of an integrity constraint by a database instance is a precisely defined notion: the database instance D is consistent if and only if it satisfies Σ, which is commonly denoted with $D \models \Sigma$.

Since it is usually assumed that the set of ICs is consistent as a set of logical sentences, in databases the notion of consistency becomes a condition on the database instance. Thus, this use of the term "consistency" differs from its use in logic, where consistency characterizes a set of formulas.

Inconsistency is an undesirable property for a database. In consequence, one attempts to keep it consistent as it is subject to updates. There are a few ways to achieve this goal. One of them consists in declaring the ICs together with the schema, and the database management system (DBMS) will take care of the database maintenance, i.e. of keeping it consistent. This is done by rejecting transactions that may lead to a violation of the ICs. For example, the DBMS should reject the insertion of the tuple (*101*, *sue jones*) into the instance in Figure 1 if the FD (1) was declared with the schema (as a key constraint). Unfortunately, commercial DBMSs offer limited support this kind of database maintenance.

An alternative way of keeping consistency is based on the use of triggers (or active rules) that are stored in the database. The reaction to a potential violation is programmed as the action of the trigger: if a violation is about to be produced or is produced, the trigger automatically reacts, and its action may reject the violating transaction or compensate it with additional updates, to make sure that at the end, consistency is reestablished. Consistency

can also be enforced through the application programs that interact with the DBMS. However, the correctness of triggers or application programs with respect to (wrt) ensuring database consistency is not guaranteed by the DBMS.

It is the case that, for whatever reasons, databases may become inconsistent, i.e. they may violate certain ICs that are considered to be relevant to maintain for a certain application domain. This can be due to several reasons, e.g. poorly designed or implemented applications that fail to maintain the consistency of the database, or ICs for which a DBMS does not offer any kind of support, or ICs that are not enforced for better performance of application programs or DBMSs, or ICs that are just assumed to be satisfied based on knowledge about the application domain and the kind of updates on the database. It is also possible to have a legacy database on which semantic constraints have to be imposed; or more generally, a database on which imposing new constrains depending on specific needs, e.g. user constraints, becomes necessary.

In the area of data integration the satisfaction of desirable ICs by a database is much more difficult to achieve. One can have different autonomous databases that are separately consistent wrt their own, local ICs. However, when their data is integrated into a single database, either material or virtual, certain desirable global ICs may not be satisfied. For example, two university databases may use the same numbers for students. If their data is put together into an integrated database, a student number might be assigned to two different students.

When trying to use an inconsistent database, the application of some *data cleaning* techniques may be attempted, to cleanse the database from data that participates in the violation of the ICs. This is done sometimes. However, data cleaning is a complex and non-deterministic process; and it may also lead to the loss of information that might be useful. Furthermore, in certain cases like virtual data integration, where the data stays at the autonomous data sources, there is no way to change the data without ownership of the sources.

One might try to live with an inconsistent databases. Actually, most likely one will be forced to keep using it, because there is still useful information in it. It is also likely that most of the information in it is somehow consistent. Thus, the challenge consists in retrieving from the database only information that is consistent. For example, one could pose queries to the database at hand, but expecting to obtain only answers that are semantically correct, i.e. that are consistent with the ICs. This is the problem of *consistent query answering* (CQA).

The notion of consistency of a database is a holistic notion, that applies to the entire database, and not to portions of it. In consequence, in order to pursue this idea of retrieving consistent query answers, it becomes necessary to characterize the consistent data in an inconsistent database first. The idea that was proposed in [1] is as follows: the consistent data in an inconsistent data is the one that is invariant under all possible way of restoring the consistency by performing minimal changes on the initial database. That is, no matter what minimal consistency restoration process is applied to the database, the consistent data stays in the database. Each of the consistent versions of the original instance obtained by minimal changes is called a *minimal repair*, or simply, a *repair*.

It becomes necessary to be more precise about the meaning of minimal change. In between, a few notions have been proposed and studied (cf. [3, 4, 2] for surveys of CQA). Which notion to use may depend on the application. The notion of minimal change can be illustrated using the definition of repair given in [1]. First of all, a database instance D can be seen as a finite set of ground atoms (or database tuples) of the form $P(\bar{c})$, where P is a predicate in the schema, and $\bar{c}$ is a finite sequence of constants in the database domain. For example, $\mathit{Students}(\mathit{101}, \mathit{john\ bell})$ is an atom in the database. Next, it is possible to compare the original database instance D with any other database instance D' (of the same schema) through their symmetric difference $D \Delta D' = \{A \mid A \in (D \smallsetminus D') \cup (D' \smallsetminus D)\}$.

Now, a repair of an instance D wrt a set of ICs Σ is defined as an instance D' that is consistent, i.e. $D' \models \Sigma$, and for which there is no other consistent instance D'' that is closer to D than D', i.e. for which it holds $D \Delta D'' \subsetneq D \Delta D'$. For example, the database in Figure 2 has two repairs wrt the FD (1). They are shown in Figure 3 and are obtained each by deleting one of the two conflicting tuples in relation *Students* (relation *Enrollment* does not change).

Having defined the notion of repair, a *consistent answer* from an instance D to a query $Q(\bar{x})$ wrt a set Σ of ICs is defined as an answer $\bar{c}$ to Q that is obtained from every possible repair of D wrt Σ. That is, if the query Q is posed to each of the repairs, $\bar{c}$ will be returned as a usual answer to Q from each of them.

For example, if the query $Q_1(x, y) : \mathit{Students}(x, y)$, asking for the tuples in relation *Students*, is posed to the instance in Figure 2, then $(\mathit{104}, \mathit{claire\ stevens})$ and $(\mathit{107}, \mathit{pat\ norton})$ should be the only consistent answers wrt the FD (1). Those are the tuples that are shared by the extensions of *Students* in the two repairs. Now, for

the query $Q_2(x) : \exists y Students(x, y)$, i.e. the projection on the first attribute of relation *Students*, the consistent answers are (*101*), (*104*) and (*107*).

Students1	StuNum	StuName
	101	john bell
	104	claire stevens
	107	pat norton

Students2	StuNum	StuName
	101	joe logan
	104	claire stevens
	107	pat norton

Figure 3: Two Repairs

There might be a large number of repairs for an inconsistent database. In consequence, it is desirable to come up with computational methodologies to retrieve consistent answers that use only the original database, in spite of its inconsistency. Such a methodology, that works for particular syntactic classes of queries and ICs, was proposed in [1]. The idea is to take the original query Q that expects consistent answers, and syntactically transform it into a new query Q', such that the *rewritten query* Q', when posed to the original database, obtains as usual answers the consistent answers to query Q. The essential question is, depending on the language in which Q is expressed, what kind of language is necessary for expressing the rewriting Q'. The answer to this question should also depend on the kind of ICs being considered.

The idea behind the rewriting approach presented in [1] can be illustrated by means of an example. The consistent answers to the query $Q_1(x, y) : Students(x, y)$ above wrt the FD (1) can be obtained by posing the query $Q'(x, y) : Students(x, y) \wedge \neg\exists z(Students(x, z) \wedge z \neq y)$ to the database. The new query collects as normal answers those tuples where the value of the first attribute is not associated to two different values of the second attribute in the relation. It can be seen that the answer set for the new query can be computed in polynomial time in the size of the database.

In this example, a query expressed in first-order predicate logic was rewritten into a new query expressed in the same language. It has been established in the literature that, for complexity-theoretic reasons, a more expressive language to do the rewriting of a first-order query may be necessary. For example, it may be necessary to do the rewritings as queries written in expressive extensions of Datalog [2, 3, 4].

If a database is inconsistent wrt referential ICs, like the instance in Figure 1 and the constraint in (2), it is natural to restore consistency by deleting tuples or inserting tuples containing *null values* for the existentially quantified variables in the ICs. For example, the tuple (*105*, *comp120*) could be deleted from *Enrollment* or the tuple (*105*, *null*) could be inserted in relation *Students*. This requires a modification of the notion of repair and a precise semantics for satisfaction of ICs in the presence of null values [3, 4].

KEY APPLICATIONS*

Key applications of *consistent query answering* (CQA) are still missing. Applications to virtual data integration look promising, and also applications to data cleaning.

FUTURE DIRECTIONS

There are many open problems and research directions, among them, and most prominently, the development of key applications of CQA. A more precise characterization of the languages that are needed doing CQA using query rewriting is also missing. It also becomes necessary to shed more light on the right kind of repair semantics to use depending on the application. CQA in a dynamic setting, when the databases is subject to updates, has not been investigated much. Integrity constraints and consistency issues for the relational model of data have been investigated for many years. However, there are other data models, e.g. spatial databases, for which much research of this kind is still necessary.

CROSS REFERENCE*

Relational Theory, Null Values, Logics and Databases, Logical Data Integration, Data Cleaning, Active Databases.

RECOMMENDED READING

[1] Arenas, M., Bertossi, L. and Chomicki, J. (1999): Consistent Query Answers in Inconsistent Databases. In Proc. ACM Symposium on Principles of Database Systems (PODS'99). ACM Press, pp. 68-79.

[2] Bertossi, L. and Chomicki, J. (2003): Query Answering in Inconsistent Databases. In *Logics for Emerging Applications of Databases*, Springer, pp. 43-83.
[3] Bertossi, L. (2006): Consistent Query Answering in Databases. *ACM Sigmod Record*, 35(2):68-76.
[4] Chomicki, J. (2007): Consistent Query Answering: Five Easy Pieces. In Proc. International Conference on Database Theory (ICDT'07), Springer LNCS 4353, pp. 1-17.
[5] Codd, E.F. (1970): A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377-387.
[6] Enderton, H. (2001): *A Mathematical Introduction to Logic*. Academic Press, 2nd edition.

CONSISTENT QUERY ANSWERING

Leopoldo Bertossi
Carleton University, http://www.scs.carleton.ca/~bertossi

SYNONYMS

None

DEFINITION

Consistent query answering (CQA) is the problem of querying a database that is inconsistent, i.e. that fails to satisfy certain integrity constraints, in such a way that the answers returned by the database are consistent with those integrity constraints. This problem involves a characterization of the semantically correct or consistent answers to queries in an inconsistent database.

MAIN TEXT

Databases may be inconsistent in the sense that certain desirable integrity constraints (ICs) are not satisfied. However, it may be necessary to still use the database, because it contains useful information, and, most likely, most of the data is still consistent, in some sense. CQA, as introduced in [1], deals with two problems. First, with the logical characterization of the portions of data that are consistent in the inconsistent database. Secondly, with developing computational mechanisms for retrieving the consistent data. In particular, when queries are posed to the database, one would expect to obtain as answers only those answers that are semantically correct, i.e. that are consistent with the ICs that are violated by the database as a whole.

The consistent data in the database is characterized [1] as the data that is invariant under all the database instances that can be obtained after making minimal changes in the original instance with the purpose of restoring consistency. These instances are the so-called (minimal) *repairs*. In consequence, what is consistently true in the database is what is *certain*, i.e. true in the collection of possible worlds formed by the repairs. Depending on the queries and ICs, there are different algorithms for computing consistent answers. Usually, the original query is transformed into a new query, possibly written in a different language, to be posed to the database at hand, in such a way that the usual answers to the latter are the consistent answers to the former [1]. For surveys of CQA and specific references, c.f. [2,3].

CROSS REFERENCE*

Inconsistent Databases, Database Repairs

REFERENCES*

[1] Arenas, M., Bertossi, L. and Chomicki, J. (1999): Consistent Query Answers in Inconsistent Databases. In Proc. ACM Symposium on Principles of Database Systems (PODS'99). ACM Press, pp. 68-79.

[2] Bertossi, L. (2006): Consistent Query Answering in Databases. *ACM Sigmod Record*, 35(2):68-76.

[3] Chomicki, J. (2007): Consistent Query Answering: Five Easy Pieces. In Proc. International Conference on Database Theory (ICDT'07), Springer LNCS 4353, pp. 1-17.

DATABASE REPAIR

Leopoldo Bertossi
Carleton University, http://www.scs.carleton.ca/~bertossi

SYNONYMS

None

DEFINITION

Given an inconsistent database instance, i.e. that fails to satisfy a given set of integrity constraints, a repair is a new instance over the same schema that is consistent and is obtained after performing minimal changes on the original instance with the purpose of restoring consistency.

MAIN TEXT

Database instances may be inconsistent, in the sense that they may not satisfy certain desirable integrity constraints. In order to make the database consistent, certain updates can be performed on the database instance. However, it is natural to expect that any new consistent instance obtained in this way does not differ too much from the original instance. The notion of repair of the original instance captures this intuition: it is an instance of the same schema that does satisfy the integrity constraints and differs from the original instance by a minimal set of changes. Depending on what is meant by minimal set of changes, different repair semantics can be obtained.

The notion of *repair*, also called *minimal repair*, was introduced in [1]. Database instances can be seen as finite sets of ground atoms. For example, $\mathit{Students}(\mathit{101}, \mathit{joe})$ could be a database atom representing an entry in the relation *Students*. In order to compare two instances of the same schema, it is possible to consider their (set-theoretic) symmetric difference. A repair, as introduced in [1], will make the symmetric difference with the original instance minimal under set inclusion. That is, no other consistent instance differs from the original instance by a proper subset of database tuples. It is implicit in this notion of repair that changes on the original instance are obtained through insertions or deletions of complete database atoms. This notion of repair was used in [1] to characterize the consistent data in an inconsistent database as the data that is invariant under all possible repairs.

In the same spirit, other repairs semantics have also been investigated in the literature. For example, an alternative definition of repair might minimize the cardinality of the symmetric difference. There are also repairs that are obtained via direct updates of attribute values (as opposed to deletions followed by insertions, which might not represent a minimal change). In this case, the number of those local changes could be minimized. A different, more general aggregation function of the local changes could be minimized instead (cf. [2,3] for surveys).

CROSS REFERENCE*

Inconsistent Databases, Consistent Query Answering

REFERENCES*

[1] Arenas, M., Bertossi, L. and Chomicki, J. (1999): Consistent Query Answers in Inconsistent Databases. In Proc. ACM Symposium on Principles of Database Systems (PODS'99). ACM Press, pp. 68-79.

[2] Bertossi, L. (2006): Consistent Query Answering in Databases. *ACM Sigmod Record*, 35(2):68-76.

[3] Chomicki, J. (2007): Consistent Query Answering: Five Easy Pieces. In Proc. International Conference on Database Theory (ICDT'07), Springer LNCS 4353, pp. 1-17.

NULL VALUES

Leopoldo Bertossi
Carleton University, http://www.scs.carleton.ca/~bertossi

SYNONYMS

None

DEFINITION

Null values are used to represent *uncertain data values* in a database instance.

MAIN TEXT

Since the beginning of the relational data model, *null values* have been investigated, with the intention of capturing and representing data values that are uncertain. Depending on the intuitions and cases of uncertainty, different kinds of null values have been proposed, e.g. they may represent information that is withheld, inapplicable, missing, unknown, etc. Thus, in principle, it could be possible to find in a hypothetical database diverse classes of null values, and also several null values of the same class. However, in commercial relational DBMSs and in the SQL Standard, only a single constant, `NULL`, is used to represent the missing values.

Many semantic problems appear when null values are integrated with the rest of the relational data model, which essentially follows the semantics of predicate logic. Among them, (a) the interpretation of nulls values (for a particular intuition); (b) the meaning of relational operations when applied to both null values and certain data values; (c) the characterization of consistency of databases containing null values.

Different formal semantics for null values have been proposed. A common and well-studied semantics for *incomplete databases* uses null values to represent unknown or missing values. Each null value in the database represents a whole set of possible values from the underlying data domain. The combination of concrete values that null values might take generates a class of alternative instances containing certain values. This *possible worlds semantics* makes true whatever is true in every alternative instance. However, the usage of null values in the SQL Standard and commercial DBMS still lacks a clear and complete formal semantics.

CROSS REFERENCE*

Uncertainty and Data Quality Management, Incomplete Information

REFERENCES*

[1] Grahne, G. (1991): *The Problem of Incomplete Information in Relational Databases*. Springer LNCS 554.

[2] Levene, M. and Loizou, G. (1999): A Guided Tour of Relational Databases and Beyond, Springer. Chapter 5.

[3] Van der Meyden, R. (1998): Logical Approaches to Incomplete Information: A Survey. In *Logics for Databases and Information Systems*, J. Chomicki and G. Saake (eds.), Kluwer, pp. 307-356.