Constraint Query Languages

Floris Geerts
University of Edinburgh, Edinburgh, UK

Definition

A constraint query language is a query language for constraint databases.

Historical Background

The field of constraint databases was initiated in 1990 in a paper by Kanellakis, Kuper and Revesz [9]. The goal was to obtain a database-style, optimizable version of constraint logic programming. It grew out of the research on DATALOG and constraint logic programming. The key idea was that the notion of tuple in a relational database could be replaced by a conjunction of constraints from an appropriate language, and that many of the features of the relational model could then be extended in an appropriate way. In particular, standard query languages such as those based on first-order logic and DATALOG could be extended to such a model.

It soon became clear, however, that recursive constraint query languages led to non-effective languages. The focus therefore shifted to non-recursive constraint query languages. The standard query language is the constraint relational calculus (or equivalently, the constraint relational algebra). The study of this query language turned out to lead to many interesting research problems. During the period from 1990 to 2000, the constraint setting has been studied in great generality which led to deep connections between constraint databases and embedded finite model theory. Also, the potential application of constraint databases in the spatial context led to numerous theoretical results and concrete implementations such as the DEDALE and the DISCO systems. The connection with so-called o-minimal geometry underlies many of the results in the spatial setting. The success of this research led to the publication of a comprehensive survey of the area in 2000 [11] and a textbook in 2002 [13].

In recent years, constraint query languages have been studied in new application domains such a strings, spatio-temporal and moving objects.

Foundations

In the constraint model, a database is viewed as a collection of constraints specified by quantifier-free first-order logic formulas over some fixed vocabulary Ω. When interpreted over an Ω-structure $\mathcal{M} = \langle \mathbb{U}, \Omega \rangle$, each constraint corresponds to an $\mathcal{M}$-definable set. Consequently, when interpreted over $\mathcal{M}$, an Ω-constraint database corresponds to a collection of $\mathcal{M}$-definable sets. For instance, consider the vocabulary $\Omega = (+, \cdot, 0, 1, <)$ and $\mathcal{M} = \langle \mathbb{R}, \Omega \rangle$. Constraints in first-order logic over Ω, denoted by FO(Ω), correspond to Boolean combinations of polynomial inequalities with integer coefficients. The corresponding $\mathcal{M}$-definable sets are better known as semi-algebraic sets. Let $\mathcal{R} = \{R, S\}$ be a relational schema consisting of two binary relations R and S and let $\mathbf{D}$ be the constraint database that maps $R \mapsto \varphi_R(x, y) = (x^2 + y^2 \leq 1) \wedge (y - x \geq 0)$ and $S \mapsto \varphi_S(x, y) = (x^2 + y^2 \leq 1) \wedge (-y - x \geq 0)$. The two $\mathcal{M}$-definable sets in $\mathbb{R}^2$ corresponding to φ_R and φ_S are shown in Figs. 1(a) and (b) respectively.

A constraint database can therefore be viewed from two different perspectives: First, one can simply look at the finite representations (constraints) stored in them; Second, one can regard them as a set of definable sets. Whereas in traditional relational databases, a query is simply a mapping that associates with each database an answer relation, in the constraint setting the two different perspectives give rise to two different notions of queries.

Indeed, for a fixed vocabulary Ω, relational schema $\mathcal{R}$ consisting of relation names R_1,...,R_ℓ, where each relation R_i is of arity $n_i > 0$, and natural number k, a *k-ary constraint query* with schema $\mathcal{R}$ over Ω, is a (partial) function Q that maps each Ω-constraint databases $\mathbf{D}$ with schema $\mathcal{R}$ to a k-ary Ω-constraint relation $Q(\mathbf{D})$. That is, a constraint query works entirely on the representational (constraint) level. On the other hand, given an additional Ω-structure $\mathcal{M} = \langle \mathbb{U}, \Omega \rangle$, a *$k$-ary unrestricted query* with schema $\mathcal{R}$ over $\mathcal{M}$ is a (partial) function Q that maps each collection $\mathbf{D}$ of sets in $\mathbb{U}^{n_i}$, for $i \in [1, \ell]$, to a set $Q(\mathbf{D})$ in $\mathbb{U}^k$. Such a collection of sets $\mathbb{U}^{n_i}$, for $i \in [1, \ell]$, is called an *unrestricted database* with schema $\mathcal{R}$ over $\mathcal{M}$.

For instance, consider again $\Omega = (+, \cdot, 0, 1, <)$ and $\mathcal{R} = \{R, S\}$. The mapping Q_1 that associates each Ω-constraint database $\mathbf{D}$ over $\mathcal{R}$ with the binary Ω-constraint relation defined by taking the disjunction of the constraints in R and S, is an example of a 2-ary constraint query over Ω. When applied on the database $\mathbf{D}$ given above, $Q_1(\mathbf{D})$ is mapped to $\varphi_R(x, y) \vee \varphi_S(x, y)$. Similarly, the mapping Q_2 that maps $\mathbf{D}$ to the

Constraint Query Languages. Figure 1. Sets in $\mathbb{R}^2$ defined by $\varphi_R(x, y)$ (a); by $\varphi_S(x, y)$ (b); by $\varphi_R(x, y) \vee \varphi_S(x, y)$ (c); and by $\varphi_1(x, y)$ (d). The set in $\mathbb{R}$ defined by $\varphi_2(x)$ (e). An example of a non-definable set in $\mathbb{R}^2$ (f).

constraint in R or S that contains the polynomial with the largest coefficient (if there is no such unique constraint then Q_2 is undefined) is also a constraint query. It will be undefined on the example database $\mathbf{D}$ since both R and S consist of a polynomial with coefficient one.

So far, only constraint queries have been considered. To relate constraint and unrestricted queries requires some care. Clearly, a constraint query only makes sense if it corresponds to an unrestricted query. In this case, a constraint query is called *consistent*. More formally, a constraint query Q is called consistent if there exists an unrestricted query Q_0 such that for any constraint database $\mathbf{D}$ and any unrestricted database $\mathbf{D}_0$, if $\mathbf{D}$ represents $\mathbf{D}_0$, then $Q(\mathbf{D})$ is defined if and only if $Q_0(\mathbf{D}_0)$ is defined and furthermore, $Q(\mathbf{D})$ represent $Q_0(\mathbf{D}_0)$. One also says that Q *represents* Q_0.

For instance, consider again $\mathbf{\Omega} = (+,\cdot,0,1,<)$, $\mathcal{M} = \langle \mathbb{R}, \mathbf{\Omega} \rangle$ and $\mathcal{R} = \{R, S\}$. The mapping $\tilde{Q}_1$ that assigns to any two sets $A \subseteq \mathbb{R}^2$ and $B \subseteq \mathbb{R}^2$, corresponding to R and S, respectively, their union $A \cup B \subseteq \mathbb{R}^2$ is an unrestricted query. It is clear that Q_1 and $\tilde{Q}_1$ satisfy the condition of consistency and therefore Q_1 is consistent. Fig. 1(c) shows $\tilde{Q}_1(\mathbf{D}_0)$ for the unrestricted database $\mathbf{D}_0$ shown in Fig. 1(a) and (b). This set is indeed represented by the constraint relation $Q_1(\mathbf{D}) \mapsto \varphi_R(x, y) \vee \varphi_S(x, y)$. On the other hand, it is easily verified that Q_2 is not consistent. Indeed, it suffices to consider the behavior of Q_2 on $\mathbf{D}$ defined above and $\mathbf{D}'$ defined by $R \mapsto \varphi'_R(x, y) = (x^2 + y^2 \leq 1) \wedge (6(y - x) \geq 0)$ and $S \mapsto \varphi'_S(x, y) = (x^2 + y^2 \leq 1) \wedge (-y - x \geq 0)$. While both $\mathbf{D}$ and $\mathbf{D}'$ represent the same unrestricted database, note that $Q_2(\mathbf{D})$ is undefined while $Q_2(\mathbf{D}') \mapsto \varphi_R$. Hence, no unrestricted query that is consistent with Q_2 can exist.

Finally, unrestricted queries are defined without any reference to the class of $\mathcal{M}$-definable sets. A desirable property, however, is that when an unrestricted query Q is defined on an unrestricted database $\mathbf{D}$ that consists of $\mathcal{M}$-definable sets, then also $Q(\mathbf{D})$ is an $\mathcal{M}$-definable set. Such unrestricted queries are called *closed*. Note that an unrestricted query that is represented by a consistent constraint query is uniquely defined and moreover is trivially closed. An example of an unrestricted query for $\mathbf{\Omega} = (+,\cdot,0,1,<)$ and $\mathcal{M} = \langle \mathbb{R}, \mathbf{\Omega} \rangle$ that is not closed is the query Q that maps any $\mathcal{M}$-definable set A in $\mathbb{R}^2$ to its intersection $A \cap \mathbb{Q}^2$. Fig. 1(f) shows (approximately) the result of this query on $\tilde{Q}_1(\mathbf{D}_0)$ (i.e., Fig. 1(c)). Since this is not a semi-algebraic set in $\mathbb{R}^2$, it cannot be defined by means of a quantifier-free FO($\mathbf{\Omega}$)-formula. As a consequence, Q is not closed.

Now that the notion of query is defined in the setting of constraint databases, the basic *constraint query language* is introduced. This language, in the same spirit as the relational calculus for traditional relational databases, is the *relational calculus* or first-order logic of the given class of constraints. More specifically, given a vocabulary $\mathbf{\Omega}$ and relational schema $\mathcal{R}$, a *relational calculus formula* over $\mathbf{\Omega}$ is a first-order logic formula over the expanded vocabulary $(\mathbf{\Omega}, \mathcal{R})$ obtained by expanding $\mathbf{\Omega}$ with the relation names (viewed as predicate symbols) of the schema $\mathcal{R}$. This class of queries is denoted by FO$(\mathbf{\Omega}, \mathcal{R})$, or simply FO$(\mathbf{\Omega})$ when $\mathcal{R}$ is understood from the context.

For instance, for $\mathbf{\Omega} = (+, \cdot, 0, 1, <)$ and $\mathcal{R} = \{R, S\}$, the expressions $\varphi_1(x, y) = (R(x, y) \vee S(x, y)) \wedge x > 0$ and $\varphi_2(x) = \exists y \varphi_1(x, y)$ are formulas in FO$(+, \cdot, 0, 1, <, R, S)$.

Given an $\mathbf{\Omega}$-structure $\mathcal{M} = \langle \mathbb{U}, \mathbf{\Omega} \rangle$, formulas in FO$(\mathbf{\Omega}, \mathcal{R})$ express (everywhere defined) unrestricted queries with schema $\mathcal{R}$ over $\mathcal{M}$. Indeed, a formula $\varphi(x_1,\ldots,x_k) \in \text{FO}(\mathbf{\Omega}, \mathcal{R})$ defines the k-ary unrestricted query Q over $\mathcal{M}$ as follows: consider the expansion of $\mathcal{M}$ to a structure $\langle \mathcal{M}, \mathbf{D} \rangle = \langle \mathbb{U}, \mathbf{\Omega}, \mathbf{D} \rangle$ over the expanded vocabulary $(\mathbf{\Omega}, \mathcal{R})$ by adding the sets in the unrestricted database $\mathbf{D}$ to $\mathcal{M}$ for each $R_i \in \mathcal{R}$. Then, $Q(\mathbf{D}) = \{(a_1,\ldots,a_k) \in \mathbb{U}^k \mid 0\langle \mathcal{M}, \mathbf{D} \rangle \models \varphi(a_1,\ldots,a_k)\}$.

For instance, for $\mathbf{\Omega} = (+,\cdot,0,1,<)$ and $\mathcal{M} = \langle \mathbb{R}, \mathbf{\Omega} \rangle$, the formula $\varphi_1(x, y)$ defined above

corresponds to the unrestricted query Q_1 that takes the union of the two sets in $\mathbb{R}^2$ corresponding to R and S, respectively, restricted to those points in $\mathbb{R}^2$ with strictly positive x-coordinate. Similarly for φ_2, but with an additional projection on the x-axis. The results of these two unrestricted queries have been shown in Figs. 1(d), (e), respectively.

The previous example raises the following two questions: (i) are the unrestricted queries expressed by formulas in first-order logic closed, and (ii) if so, can one find a corresponding constraint query that is effectively computable? The fundamental mechanism underlying the use of first-order logic as a constraint query language is the following observation that provides an answer to both questions:

- Every relational calculus formula φ expresses a consistent, effectively computable, total constraint query that represents the unrestricted query expressed by φ, if and only if $\mathcal{M}$ admits *effective quantifier elimination*.

Here, an Ω-structure $\mathcal{M}$ admits effective quantifier elimination if there exists an effective algorithm that transforms any first-order formula in FO(Ω) to an equivalent (in the structure $\mathcal{M}$) *quantifier-free* first-order formula in FO(Ω).

Consider the two FO($+,\cdot,0,1,<$, R, S)-formulas φ_1 and φ_2 given above. It is known that the structure $\langle \mathbb{R}, +, \cdot, 0, 1, < \rangle$ admits effective quantifier-elimination. In case of φ_1 it is easy to see that the result of corresponding constraint query is obtained by "plugging" in the constraints for R (resp. S) as given by the constraint database into the expression for φ_1. That is, on the example database **D**, φ_1 corresponds to the constraint query that maps **D** to $(\varphi_R(x, y) \vee \varphi_S(x, y)) \wedge (x > 0)$, which is a 2-ary Ω-constraint relation. In case of φ_2, however, first plug in the descriptions of the constraints as before, resulting in $\exists y\,(\varphi_R(x, y) \vee \varphi_S(x, y)) \wedge (x > 0)$. In order to obtain an Ω-constraint relation, one needs perform quantifier-elimination. It is easily verified that in this example, a corresponding constraint query is one that maps **D** to $(0 < x) \wedge (x \leq 1)$ which is consistent with Fig. 1(e).

For Ω-structures $\mathcal{M}$ that admit effective quantifier-elimination, this suggests the following effective evaluation mechanism for constraint relational calculus queries φ on a constraint database **D**: (i) plug in the contents of **D** in the appropriate slots (relations). Denote the resulting formula by plug(φ, **D**); and (ii) eliminate the quantifiers in plug(φ, **D**). Since **D** consists of quantifier-free formulas, the number of quantifiers that need to be eliminated is the same as in φ and is therefore independent of **D**. For many structures $\mathcal{M}$ this implies that the evaluation of constraint queries can be done in polynomial data complexity, which is a desirable property for any query language.

It is important to point out that the classical equivalence between the relational calculus and the relational algebra can be easily extended to the constraint setting. That is, for a fixed Ω and schema $\mathcal{R}$, one can define a *constraint relational algebra* and show that every constraint relational calculus formula can be effectively converted to an equivalent constraint relational algebra expression, and vice versa. This equivalence is useful for concrete implementations of constraint database systems.

The study of expressivity of FO($\Omega, \mathcal{R}$) for various Ω-structures $\mathcal{M}$ has led to many interesting results. In particular, the impact of the presence of the "extra" structure on the domain elements in $\mathbb{U}$ has been addressed when **D** consists of an ordinary finite relational database that takes values from $\mathbb{U}$ [3]. In particular, the correspondence between natural and active-domain semantics has been revisited. That is, conditions are identified for $\mathcal{M} = \langle \mathbb{U}, \Omega \rangle$ such that the language FO($\Omega, \mathcal{R}$) is equal to $\mathrm{FO}_{\mathrm{act}}(\Omega, \mathcal{R})$, the query language obtained by interpreting $\forall x$ and $\exists x$ over the active domain of **D** instead of over $\mathbb{U}$. Such structures are said to admit the *natural-active collapse*. Similarly, ordered structures $\mathcal{M}$ are identified that admit the *active-generic collapse*. That is, $\mathrm{FO}_{\mathrm{act}}(\Omega, \mathcal{R})$ is equal to $\mathrm{FO}_{\mathrm{act}}(<, \mathcal{R})$ with respect to the class of generic queries. In other words, every generic query definable under active domain semantics with Ω-constraints is already definable with just order constraints. Finally, structures $\mathcal{M}$ are considered that allow the *natural-generic collapse*. This is the same as the active-generic collapse but with natural domain semantics instead of active domain semantics. The study of these collapse properties for various structures not only sheds light on the interaction of the structure on $\mathbb{U}$ and the query language, it is also helpful to understand the expressiveness of constraint query languages [3,11].

Indeed, let $\Omega = (+, \times, 0, 1, <)$ and $\mathcal{M} = \langle \mathbb{R}, \Omega \rangle$. It can be shown that $\mathcal{M}$ admits all three collapses because it is a so-called o-minimal structure. As a consequence, the query EVEN that returns **yes** if the

cardinality of **D** is even and **no** otherwise, is not expressible in FO($\Omega, \mathcal{R}$). Indeed, if it would be expressible by a query φ in FO($\Omega, \mathcal{R}$) it would already have been expressible by a query in $\mathrm{FO}_{\mathrm{act}}(<, \mathcal{R})$, which is known not to be true in the traditional database setting.

The expressivity of FO($\Omega, \mathcal{R}$) has been studied extensively as well when **D** corresponds to sets of infinite size. In particular, expressiveness questions have been addressed in the spatial setting where $\Omega = (+, \cdot, 0, 1, <)$ and $\mathcal{M} = \langle \mathbb{R}, \Omega \rangle$ (polynomial constraints); and $\Omega' = (+, 0, 1, <)$ and $\mathcal{M} = \langle \mathbb{R}, \Omega' \rangle$ (linear constraints). In this setting, many reductions are presented in [7] to expressiveness questions in the finite case. Combined with the collapse results mentioned above, these reductions were used to show that, for example, topological connectivity of Ω- (resp. Ω'-) constraint databases is not expressible in first-order logic. Indeed, a proof of this results relies on the fact that the EVEN-query is not expressible in FO($\Omega, \mathcal{R}$) (resp. FO($\Omega', \mathcal{R}$)) [7].

An interesting line of work in the spatial context concerns the expressive power of FO($\Omega, \mathcal{R}$) with respect to queries that preserve certain geometrical properties. More formally, let $\mathcal{G}$ be a group of transformations of $\mathbb{R}^k$. A query Q is called $\mathcal{G}$-generic if, for every transformation $g \in \mathcal{G}$, and for any two databases **D** and **D**′, g(**D**) = **D**′ implies g(Q(**D**)) = Q(**D**′). Transformation groups and properties of the corresponding generic queries have been studied for the group of homeomorphisms, affinities, similarities, isometries, among others [8]. Especially the study of the topologically queries (those that are generic under homeomorphisms) has received considerable attention [10,2].

To conclude, both for the historical reasons mentioned above and in view of the limited expressive power of FO($\Omega, \mathcal{R}$), various recursive extensions of FO($\Omega, \mathcal{R}$) have been proposed such as: constraint transitive-closure logic [5], constraint DATALOG [9], and FO($\Omega, \mathcal{R}$) extended with a WHILE-loop [8]. The interaction of recursion with the structure on $\mathbb{U}$ imposed by Ω leads in most cases to computationally complete query languages. Worse still, queries defined in these languages may not be closed or even terminate. To remedy this, special-purpose extensions of FO($\Omega, \mathcal{R}$) have been proposed that guarantee both termination and closure. Characteristic examples include FO($\Omega, \mathcal{R}$) + AVG and FO($\Omega, \mathcal{R}$) + SUM in the context of aggregation [4]. In the spatial setting, extensions of FO($\Omega, \mathcal{R}$) with various connectivity operators have been proposed [1].

Results concerning constraint query languages have been both extended to great generality and applied to concrete settings. Refer to [14] for a gentle introduction and to [11] for a more detailed survey of this research area up to 2000. Some more recent results are included in Chapter 5 of [12] for the general constraint setting and Chapter 12 in [6] for the spatial setting.

Key Applications

Manipulation and querying of constraint databases, querying of spatial data.

Cross-references

- ▶ Complete Query Language
- ▶ Constraint Databases
- ▶ FOL Model
- ▶ FOL Syntax
- ▶ Query Language
- ▶ Relational Calculus and Algebra
- ▶ Relational Model

Recommended Reading

1. Benedikt M., Grohe M., Libkin L., and Segoufin L. Reachability and connectivity queries in constraint databases. J. Comput. Syst. Sci., 66(1):169–206, 2003.
2. Benedikt M., Kuijpers B., Christ of Löding, Van den Bussche J., and Wilke T. A characterization of first-order topological properties of planar spatial data. J. ACM, 53(2):273–305, 2006.
3. Benedikt M. and Libkin L. Relational queries over interpreted structures. J. ACM, 47(4):644–680, 2000.
4. Benedikt M. and Libkin L. Aggregate operators in constraint query languages. J. Comput. Syst. Sci., 64(3):628–654, 2002.
5. Geerts F., Kuijpers B., and Van den Bussche J. Linearization and completeness results for terminating transitive closure queries on spatial databases. SIAM J. Comput., 35(6):1386–1439, 2006.
6. Geerts F. and Kuijpers B. Real algebraic geometry and constraint databases. In Marco Aiello, Ian Pratt-Hartmann, and Johan Van Benthem, editors, Handbook of Spatial Logics. Springer 2007.
7. Grumbach S. and Su J. Queries with arithmetical constraints. Theor. Comput. Sci., 173(1):151–181, 1997.
8. Gyssens M., Van den Bussche J., and Van Gucht D. Complete geometric query languages. J. Comput. Syst. Sci., 58(3):483–511, 1999.
9. Kanellakis P.C., Kuper G.M., and Revesz P.Z. Constraint Query Languages. J. Comput. Syst. Sci., 51(1):26–52, 1995.
10. Kuijpers B., Paredaens J., and Van den Bussche J. Topological elementary equivalence of closed semi-algebraic sets in the real plane. J. Symb. Log., 65(4):1530–1555, 2000.
11. Kuper G.M., Libkin L., and Paredaens J. Constraint Databases. editors. Springer, 2000.

12. Libkin L. Embedded finite models and constraint databases. In Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein, editors, Finite Model Theory and Its Applications. Springer, 2007.
13. Revesz P.Z. Introduction to Constraint Databases. Springer, 2002.
14. Van den Bussche J. Constraint databases. A tutorial introduction. SIGMOD Record, 29(3):44–51, 2000.

Constraint-Driven Database Repair

Wenfei Fan[1,2]
[1]University of Edinburgh, Edinburgh, UK
[2]Bell Laboratories, Murray Hill, NJ, USA

Synonyms

Data reconciliation; Minimal-change integrity maintenance; Data standardization

Definition

Given a set Σ of integrity constraints and a database instance D of a schema R, the problem of *constraint-driven database repair* is to find an instance D' of the same schema R such that (i) D' is *consistent*, i.e., D' satisfies Σ, and moreover, (ii) D' *minimally differs* from the original database D, i.e., it takes a minimal number of repair operations or incurs minimal cost to obtain D' by updating D.

Historical Background

Real life data is often dirty, i.e., inconsistent, inaccurate, stale or deliberately falsified. While the prevalent use of the Web has made it possible, on an unprecedented scale, to extract and integrate data from diverse sources, it has also increased the risks of creating and propagating dirty data. Dirty data routinely leads to misleading or biased analytical results and decisions, and incurs loss of revenue, credibility and customers. With this comes the need for finding repairs of dirty data, and editing the data to make it consistent. This is the data cleaning approach that US national statistical agencies, among others, has been practicing for decades [10].

The notion of constraint-based database repairs is introduced in [1], highlighting the use of integrity constraints for characterizing the consistency of the data. In other words, constraints are used as data quality rules, which detect inconsistencies as violations of the constraints. Prior work on constraint-based database repairs has mostly focused on the following issues. (i) Integrity constraints used for repair. Earlier work considers traditional functional dependencies, inclusion dependencies and denial constraints [1,2,4, 6,12]. Extensions of functional and inclusion dependencies, referred to as conditional functional and inclusion dependencies, are recently proposed in [3,9] for data cleaning. (ii) Repair semantics. Tuple deletion is the only repair operation used in [6], for databases in which the information is complete but not necessarily consistent. Tuple deletion and insertion are considered in [1,4] for databases in which the information may be neither consistent nor complete. Updates, i.e., attribute-value modification are proposed as repair operations in [12]. Cost models for repairs are studied in [2,8]. (iii) Algorithms. The first algorithms for finding repairs are developed in [2], based on traditional functional and inclusion dependencies. Algorithms for repairing and incrementally repairing databases are studied in [8], using conditional functional dependencies. The repair model adopted by these algorithms supports updates as repair operations. (iv) Fundamental issues associated with constraint-based repairs. One issue concerns the complexity bounds on the database repair problem [2,6]. Another issue concerns the static analysis of constraint consistency [3,9] for determining whether a given set of integrity constraints is dirty or not itself.

Constraint-based database repairs are one of the two topics studied for constraint-based data cleaning. The other topic, also introduced in [1], is *consistent query answers*. Given a query Q posed on an inconsistent database D, it is to find tuples that are in the answer of Q over every repair of D. There has been a host of work on consistent query answers [1,4,6,11,12] (see [5,7] for comprehensive surveys).

Foundations

The complexity of the constraint-based database repair problem is highly dependent upon what integrity constraints and repair model are considered.

Integrity Constraints for Characterizing Data Consistency

A central technical issue for data cleaning concerns how to tell whether the data is dirty or clean. Constraint-based database repair characterizes inconsistencies in terms of violations of integrity constraints. Constraints employed for data cleaning include functional dependencies,