Database interrogation using conjunctive queries[⋆]

Michał Bielecki[1] and Jan Van den Bussche[2]

[1] Warsaw University, Poland, mab@mimuw.edu.pl
[2] University of Limburg, Belgium, jan.vandenbussche@luc.ac.be

Abstract. We consider a scenario where a client communicates with a database server by posing boolean conjunctive queries, or more generally, counts of conjunctive queries. We investigate to what extent features such as quantification, negation, or non-equalities are important in such a setting. We also investigate the difference between a setting where the client can pose an adaptive sequence of queries, and a setting where the client must pose a fixed combination of queries.

1 Introduction

Consider a scenario where a client program communicates with a database server through some query language L that is rather limited. Due to the weakness of L, the query Q the client really wants to ask is then often not expressible by a single query in L. Still the client may be able to derive the answer to Q by performing some computation during which a *sequence* of queries, rather than a single one, is posed. We could call this “interrogation” rather than querying. Given the ubiquity of client-server access to databases, studying database interrogation appears to be well-motivated in general. A concrete example of interrogation occurs in database security [5, 3].

Another well-known example of interrogation occurs in information integration using views [12]. There, L is given by a finite number of conjunctive queries called views (possibly parameterized with selection constants, so that L is actually infinite), and Q is another conjunctive query, not in L. To answer Q, one tries to derive a conjunctive query Q' over the views that is equivalent to Q. Executing this Q' is then the computation performed by the client, and indeed involves posing to the database a sequence of queries, namely, the views involved in Q'.

In the above example, the sequence of queries the client poses is *database-independent*; Q' depends only on Q and L but not on the database contents. One can also think of scenarios where the sequence is *adaptive* in that the choice of the next query to be posed depends on the actual answer received to the previous query (and thus depends on the database contents).

In the present paper, we study the more general setting where L is the language of *all* conjunctive queries, or a variation thereof. The big difference with

[⋆] Research supported by Polish KBN grant 7T11C 007 21.

information integration is that the client can now pose any conjunctive query he likes, rather than only those provided as views. However, we have to be careful or this setting becomes trivial: the identity query, which simply downloads the entire database, is a conjunctive query, so a client needs to pose only that single query and then has enough information to compute the answer to any query he wants. Also from a practical server workload or network bandwidth point of view, downloads may be undesirable. Hence, we focus our attention to a setting where the server returns only boolean answers (i.e., the emptiness of the answer), or more generally, counts (i.e., the cardinality of the answer). For example, many search engines on the Internet directly return the cardinality of the search result.

In this setting, we study the effects of features such as quantification, negation, or non-equalities, and we compare database-independent client strategies to adaptive ones. Our contributions can be summarized as follows.

1. For database-independent strategies, non-equality tests matter. More concretely, against a server that answers counts of conjunctive queries, there is a simple conjunctive query extended with a non-equality test whose cardinality cannot be derived in a database-independent way.
2. The same holds for quantification: against a server that answers counts of quantifier-free conjunctive queries, there is a simple conjunctive query whose cardinality cannot be derived in a database-independent way.
3. Complementing the previous negative result with a positive result, we show that against the same server answering counts of quantifier-free conjunctive queries, we can derive the count of any quantifier-free first-order query in a database-independent way.
   Together, these three results yield the following picture:
   $$\#\mathsf{qfCQ} \equiv \#\mathsf{qfFO} < \#\mathsf{CQ} < \#\mathsf{CQ}(\neq)$$
   Here, # stands for 'counts of', and $L_1 < L_2$ ($L_1 \equiv L_2$) means that for database-independent clients, a server answering queries in L_2 provides more (the same) information than a server answering queries in L_1.
4. Moving to adaptive strategies, we improve upon an earlier observation by Tyszkiewicz to note a general positive result: When the server answers boolean conjunctive queries with non-equalities, the client can derive the answer to *any* computable boolean query, using an adaptive strategy. Combining this result with the previous one, this means that for adaptive clients, a server answering just $\#\mathsf{qfCQ}$ queries as above, already provides complete information about the database up to isomorphism.
5. Finally, we consider the case of a server answering boolean conjunctive queries (without non-equalities). We present a number of observations showing that the exact power of adaptive clients in this case is quite intriguing.

The basic setting of our work can be traced back to the framework of reflective relational machines (RRMs) introduced by Abiteboul, Papadimitriou and Vianu [2]. Indeed, RRMs are exactly the right computational model for adaptive clients communicating with a database server. The original model focused on arbitrary boolean first-order queries as the query language; our work restricts this to the popular class of conjunctive queries, and brings counts into the picture. Different restrictions have already been studied by Tyszkiewicz. First, he revisited the complexity properties of the original model in the case of existential queries [8]. Then, he considered RRMs using boolean k-variable non-recursive Datalog queries, and showed that they cannot compute all of k-variable Datalog. He also considered RRMs using boolean k-variable first-order queries (FO^k) with modular counting, and showed that they cannot express all of FO^k with counting [9]. RRMs using boolean FO^k queries have also been studied by Turull-Torres [7], who noted that they can compute precisely all computable queries invariant under FO^k-equivalence. In contrast, we will show here that RRMs using conjunctive queries can *not* compute all computable queries invariant under conjunctive query equivalence.

2 Preliminaries

We work in the relational database model [1, 10, 11].

A *query* is a function Q, mapping databases (over some fixed schema) to relations. We assume familiarity with the class of conjunctive queries, CQ, possibly extended with safe negation (including non-equalities), $\mathsf{CQ}(\neg)$, or just with non-equalities, $\mathsf{CQ}(\neq)$. In a *quantifier-free* conjunctive query, qfCQ, all the variables of the body are also in the head.

Example 1. Over a database schema with a binary relation R and a unary relation S, the following are examples of a CQ, $\mathsf{CQ}(\neq)$, $\mathsf{CQ}(\neg)$, and qfCQ, respectively.

$$\begin{aligned}(x) &\leftarrow R(x,y), R(y,z)\\ (x) &\leftarrow R(x,y), R(y,z),\, x \neq y\\ (x) &\leftarrow R(x,y),\, \neg S(y)\\ (x,y,z) &\leftarrow R(x,y), R(y,z)\end{aligned}$$

3 Database-independent model

A *natural number query* is a function, mapping databases to natural numbers. With any normal query Q we can associate the natural number query $\#Q$, called the *count* of Q, that maps a database D to the cardinality of the relation $Q(D)$.

Definition 1. *Let L be a class of natural number queries, and let Q be another natural number query. We say that Q* can be derived in a database-independent way from *L if there is a finite sequence $Q_1, \dots, Q_k$ of queries in L, and a computable function $f : \mathbb{N}^k \to \mathbb{N}$, such that*

$$Q(D) = f(Q_1(D), \dots, Q_k(D))$$

for any database D.

Example 2. Let L be $\#\mathsf{qfCQ}$: counts of quantifier-free conjunctive queries. Consider the following $\mathsf{qfCQ}(\neg)$ query:

$$Q(x,y) \leftarrow R(x,y),\ \neg S(y)$$

Then $\#Q$ can be derived in a database-independent way from $\#\mathsf{qfCQ}$, simply using the following two queries:

$$\begin{aligned} &Q_1(x,y) \leftarrow R(x,y) \\ &Q_2(x,y) \leftarrow R(x,y), S(y) \end{aligned}$$

Indeed, we have $\#Q = \#Q_1 - \#Q_2$.

The technique illustrated in the above example can be generalized to arbitrary $\mathsf{qfCQ}(\neg)$ queries. Moreover, using the inclusion-exclusion principle $\#(A \cup B) = \#A + \#B - \#(A \cap B)$, we can generalize this further to *unions* of $\mathsf{qfCQ}(\neg)$ queries. We thus obtain:

Proposition 1. *The count of any union of* $\mathsf{qfCQ}(\neg)$ *queries can be derived in a database-independent way from* $\#\mathsf{qfCQ}$*; the derivation function is simply an integer arithmetic expression involving* $+$ *and* $-$.

The language of unions of $\mathsf{qfCQ}(\neg)$ queries can be thought of as the safe fragment of the quantifier-free first-order queries. For example, any relational algebra query involving equijoins, cartesian products, equality selections, unions, and differences, but not projections, is a union of $\mathsf{qfCQ}(\neg)$ queries. Things become simpler when we adopt the active-domain semantics for first-order queries (so that safety is no longer an issue), and are prepared to consider the query $(x) \leftarrow x = x$ a "conjunctive query". Then Proposition 1 can be reformulated by saying that *the count of any quantifier-free first-order query can be derived in a database-independent way from* $\#\mathsf{qfCQ}$.

We note the following variation of Proposition 1. An *injective* qfCQ query, denoted by qfiCQ, is a qfCQ query with modified semantics to the effect that all variables must be instantiated by pairwise distinct values. Alternatively, qfiCQ can be viewed as the fragment of $\mathsf{qfCQ}(\neq)$ where non-equalities *must* be present between every pair of distinct variables. We have:

Proposition 2. *Proposition 1 also holds from* $\#\mathsf{qfiCQ}$ *instead of* $\#\mathsf{qfCQ}$.

This proposition actually follows from Proposition 1, because the count of any qfCQ query can be derived in a database-independent way from $\#\mathsf{qfiCQ}$, as illustrated in the following simple example.

Example 3. The count of $Q(x,y) \leftarrow R(x,y)$ can be derived using the queries $Q_1(x,y) \leftarrow R(x,y),\ x \neq y$ and $Q_2(x) \leftarrow R(x,x)$. Indeed, $\#Q = \#Q_1 + \#Q_2$.

4 Quantification and non-equality

So far, we have seen that in a quantifier-free world, things are pretty simple, and it does not matter whether the server answers merely conjunctive queries, or also supports negation. We will now see that things change when quantifiers come into play. We will first show that for database-independent clients, a server answering counts of arbitrary conjunctive queries truly provides more information than a server answering counts of quantifier-free ones only. We then show that adding even a single non-equality makes the server even more informative.

More concretely, for two classes L_1 and L_2 of natural number queries, we write $L_1 \leqslant L_2$ if any natural number query derivable in a database-independent way from L_1 is this also from L_2. We write $L_1 < L_2$ if $L_1 \leqslant L_2$ but not $L_2 \leqslant L_1$. Then we have:

Theorem 1. $\#\mathsf{qfCQ} < \#\mathsf{CQ}$.

Theorem 2. $\#\mathsf{CQ} < \#\mathsf{CQ}(\neq)$.

We will prove these theorems using the following two queries:

$$\begin{aligned} Q_1(x) &\leftarrow S(x), R(x,y) \\ Q_2(x) &\leftarrow R(x,x), R(x,y),\, x \neq y \end{aligned}$$

Specifically, we will show that $\#Q_1$ is not derivable from $\#\mathsf{qfCQ}$, and $\#Q_2$ not from $\#\mathsf{CQ}$, in a database-independent way.

For any natural number n, define G_n to be the structure over $\{R, S\}$ given by $S = \{0\}$ and $R = \{(0,1), (0,2), \ldots, (0,n)\}$. Any structure isomorphic to a G_n is called a *star*. A database that is a disjoint union of stars is called a *constellation*.

For any natural number $n > 0$, define the following constellations. The constellation A_n consists of $\binom{n}{i}$ copies of G_i for every even $i \leq n$; the constellation B_n consists of $\binom{n}{i}$ copies of G_i for every odd $i \leq n$. We note:

Lemma 1. $\#Q_1(A_n) \neq \#Q_1(B_n)$.

Proof. In any constellation, $\#Q_1$ counts the number of non-G_0's. Now A_n contains one G_0, while B_n contains no G_0's. However, since $\sum_{i \text{ even}} \binom{n}{i} = \sum_{i \text{ odd}} \binom{n}{i}$, the total number of disjoint stars in A_n is the same as in B_n. Hence the lemma follows.

Now suppose, for the sake of contradiction, that $\#Q_1$ is derivable in a database-independent way from qfCQ. Since, by multiplication, counts of qfCQ's with a disconnected body are easily derivable in a database-independent way from counts of qfCQ's with a connected body, we may assume that the qfCQ queries used to derive $\#Q_1$ all have a connected body. Moreover, by Proposition 2, we can actually use qfiCQ queries. Let $\mathcal{C}$ be the finite set of connected qfiCQ queries used to derive $\#Q_1$. Let n be the maximum number of variables used in a query in $\mathcal{C}$. Now observe that the result of applying a connected qfiCQ query with j variables to a constellation equals the set of all embedded G_{j-1}'s. The following lemma, proven by a combinatorial calculation, is therefore crucial:

Lemma 2. *For any $j < n$, the number of embedded G_j's is the same in A_n as in B_n.*

We have now arrived at the desired contradiction. Indeed, Lemma 2 tells us that A_n and B_n are indistinguishable by counts of queries in $\mathcal{C}$. Since $\#Q_1$ was supposed to be derived using exactly these counts, this implies $\#Q_1(A_n) = \#Q_1(B_n)$, which contradicts Lemma 1.

Theorem 1 is thus proven. We move next to the proof of Theorem 2, which is similar, but a bit more complicated due to the lack of a counterpart to Proposition 2 in the case where we have quantification.

First, we need to adapt the notion of "star". For any natural number n, define H_n to be the relation $\{(0,0),(0,1),(0,2),\dots,(0,n)\}$. Any binary relation isomorphic to a H_n is called a *loop-star*. The element corresponding to 0 is called the *center* of the loop-star; the other elements are called *rays*. Any disjoint union of loop-stars is called a *loop-constellation*.

Similarly to A_n and B_n from above, we introduce, for any natural number $n > 0$, the loop-constellation A'_n, which consists of $\binom{n}{i}$ copies of H_i for every even $i \leq n$; and B'_n, which consists of $\binom{n}{i}$ copies of H_i for every odd $i \leq n$. We have the following analogue to Lemma 1:

Lemma 3. $\#Q_2(A'_n) \neq \#Q_2(B'_n)$.

Now suppose, for the sake of contradiction, that $\#Q_2$ is derivable in a database-independent way from CQ. Since, using multiplication, counts of CQ's with a disconnected body are easily derivable in a database-independent way from counts of CQ's with a connected body, we may assume that the CQ queries used to derive $\#Q_2$ all have a connected body. Let $\mathcal{C}$ be the finite set of connected CQ queries used to derive $\#Q_2$. Let n be one more than the maximum number of variables used in a query in $\mathcal{C}$. If we can show that for any $Q \in \mathcal{C}$, we have $\#Q(A'_n) = \#Q(B'_n)$, then, in view of Lemma 3, we have arrived at the desired contradiction and proven Theorem 2.

Take an arbitrary $Q \in \mathcal{C}$, and an arbitrary loop-constellation D. Let m be the number of variables in Q. For any $j \leq m$, define

$$S(j,Q,D) := \{t \in Q(D) \mid t \text{ contains exactly } j \text{ rays}\}.$$

Clearly,

$$\#Q(D) = \sum_{j=0}^{m} \#S(j,Q,D).$$

We therefore need to understand these sets $S(j,Q,D)$ better.

Fix j arbitrarily. Let $\mathcal{E}$ be the set of all embedded images of H_j in D, and let $\varepsilon(j,D)$ be the total number of these. For each $E \in \mathcal{E}$, fix an arbitrary embedding $e_E : H_j \to E$. Now define

$$T(j,Q,D) := \{e_E(u) \mid E \in \mathcal{E} \text{ and } u \in S(j,Q,H_j)\}.$$

We claim:

Lemma 4. $S(j, Q, D) = T(j, Q, D)$.

Proof. The inclusion from right to left is clear. For the converse inclusion, take $t \in S(j, Q, D)$. Writing Q as *head* $\leftarrow$ *body*, this means that there is a homomorphism σ from *body* to D such that $t = \sigma(\mathit{head})$. Since *body* is connected, the image of σ is part of a loop-star H in D. Since t contains j rays, H has at least j rays. Now let E be the image in $\mathcal{E}$ having as rays exactly those of t. Then we can define the following homomorphism σ' from *body* to E:

$$\sigma'(x) := \begin{cases} \sigma(x) & \text{if } \sigma(x) \text{ appears in } t; \\ \text{the center of } E & \text{otherwise.} \end{cases}$$

This is indeed a homomorphism, thanks to the self-loop at the center of E. Since $e_E^{-1} \circ \sigma'$ is then a homomorphism from *body* to H_j, the tuple $u := e_E^{-1}(\sigma'(\mathit{head}))$ is in $Q(H_j)$. Moreover, $u \in S(j, Q, H_j)$, since the number of rays in $\sigma'(\mathit{head}) = t$ equals j. Hence, since $e_E(u) = t$, we conclude that $t \in T(j, Q, D)$ as desired.

We can now complete the proof as follows. Clearly, $\#T(j, Q, D) = \varepsilon(j, D) \times \#S(j, Q, H_j)$. Hence,

$$\#Q(D) = \sum_{j=0}^{m} \varepsilon(j, D) \times \#S(j, Q, H_j).$$

We now note the following analogue to Lemma 2:

Lemma 5. *For any $j < n$, we have $\varepsilon(j, A'_n) = \varepsilon(j, B'_n)$.*

Since the factors $\varepsilon(j, D)$ in the above expression for $\#Q(D)$ are the only that depend on D, we conclude that $\#Q(A'_n) = \#Q(B'_n)$, as desired.

5 The adaptive model

We now move on to the adaptive scenario, where a client, by performing an arbitrarily complex computation during which he interacts with the database server through conjunctive queries, tries to compute the answer to a more complex query. The computational model for this is the reflective relational machine (RRM) [2]. A RRM is a kind of oracle turing machine [4]. There is no input tape; instead, there is a read-only *answer tape* and a write-only *query tape*. Whenever the machine enters some special query state, it receives, on the answer tape, the answer to the query currently written on the query tape.

In the original model, the allowed queries are boolean first-order queries. We consider RRMs with boolean conjunctive queries, or more generally, counts of conjunctive queries.

Our first observation contrasts the adaptive model to our previous database-independent model. While we saw in the previous section that a database-independent client that can ask only counts of qfCQ queries cannot even derive some very simple counting queries, we now have:

Theorem 3. *Every computable natural number query can be computed by a RRM using counts of* qfCQ *queries.*

Here, it is understood, as usual, that computable queries are invariant under database isomorphisms [1].

Theorem 3 is actually a corollary of the following:

Proposition 3. *A RRM using boolean* $\mathsf{CQ}(\neq)$ *queries can construct an isomorphic copy of the database.*

Theorem 3 follows from Proposition 3 because we can get the answer to a boolean $\mathsf{CQ}(\neq)$ query by looking at the count of its quantifier-free version. Counts of $\mathsf{qfCQ}(\neq)$ queries can then in turn be expressed as counts of qfCQ queries, by Proposition 1.

We give:

Proof (of Proposition 3). For simplicity of notation only, we work with a single binary relation R. The proof is an improvement of observations already made by Abiteboul, Papadimitriou and Vianu [2] and Tyszkiewicz [8].[1]

We begin by determining the cardinality of the active domain of the database. If we adopt the active-domain semantics for first-order queries, this is easy: for progressively larger values of n, we ask queries of the form

$$() \leftarrow \bigwedge_{1 \leq i < j \leq n} x_i \neq x_j$$

until the answer becomes false. If, however, we want to work only with safe queries (as we have done so far in this paper), we can still do this: by adding atoms $R(x_i, y_i)$ we can determine the cardinality $\#\pi_1(R)$ of the first projection of R; similarly we can determine $\#\pi_2(R)$, and $\#(\pi_1(R) \cap \pi_2(R))$. The cardinality of the active domain is then $\#\pi_1(R) + \#\pi_2(R) - \#(\pi_1(R) \cap \pi_2(R))$.

Suppose we have determined the active domain to have n elements. Given an order on the active domain, we can write R as an adjacency matrix. There are at most $n!$ possible such adjacency matrices. The following query asks whether at least one of these matrices has a 1 in the first row, first column:

$$() \leftarrow R(x_1, x_1)$$

- If the answer is yes, we ask next whether, of the matrices that have a 1 in the first row, first column, there is one that has also a 1 in the first row, second column:
$$() \leftarrow R(x_1, x_1), R(x_1, x_2), \, x_1 \neq x_2$$
If the answer is yes, we know there is a matrix starting with 11. If no, we know there is a matrix starting with 10 (even more, all matrices starting with 1 continue with 0.)

[1] [2] used full FO, [8] used $\mathsf{CQ}(\neg)$, and both ignored safety.

– If the answer is no, we know all matrices have a 0 in the first row, first column, so now we ask whether there is one that has a 1 in the first row, second column:

$$() \leftarrow R(x_1, x_2),\, x_1 \neq x_2$$

if the answer is yes, we know there is a matrix starting with 01; if no, we know all matrices start with 00.

And so on. After a total of n^2 queries we have recovered a complete adjacency matrix.

Remark 1. In this paper, we have not considered constants in conjunctive queries. In the previous two sections, nothing much changes in the presence of constants. In the adaptive model of the present section, however, constants would allow us to recover the database *exactly*, not up to isomorphism, by first determining the size of the domain as above, then systematically enumerating all possible domains of that size until a match is found, and finally determining the actual tuples in the database. Of course, the number of queries required then becomes totally unpredictable, while the number of queries used in the above proof is polynomial.

Proposition 3 raises the question of what we can do without inequalities. An immediate limitation on the power of a RRM using boolean conjunctive queries, denoted RRM(CQ), is that two databases that are indistinguishable by boolean conjunctive queries will of course be indistinguishable by the RRM as well. Formally, we call two databases A and B (over the same schema) CQ-*equivalent* if there are homomorphisms $f : A \to B$ and $g : B \to A$. It is well known [1, 11] that A and B are CQ-equivalent if and only if they are indistinguishable by boolean conjunctive queries. As a consequence, any boolean or natural number query computable by an RRM(CQ) must be invariant under CQ-equivalence.

Example 4. The binary relations (directed graphs) $\{(1,2),(1,3),(2,4),(3,4)\}$ and $\{(5,6),(6,7)\}$ are CQ-equivalent. This shows that queries depending on precise cardinalities, or on precise in- or out-degrees, or on precise structural properties (such as "is the graph a tree?" or "is the graph a chain?"), are all not computable by a RRM(CQ).

It is natural to conjecture that RRM(CQ)'s can compute *precisely* the computable queries invariant under CQ-equivalence. We can disprove this, however. Consider the boolean query SIMPLECYCLE over binary relations (directed graphs) that asks whether the graph is CQ-equivalent to a simple cycle. We have:

Proposition 4. *SIMPLECYCLE is not computable by a RRM(*CQ*).*

Proof. Suppose there is such a RRM, call it M. Consider the simple cycle $A = \{(1,2),(2,1)\}$. M accepts A; let n be the maximum number of variables used in any query asked by M during its run on A. Pick an odd number $p > n$. Define the graph A' as the disjoint union of A with a simple cycle of length p. Now any boolean conjunctive query Q using at most n variables cannot distinguish A and A'. Hence, M will accept A' as well, which is incorrect because A' is not CQ-equivalent to a simple cycle.

We conclude by showing that nevertheless, some non-trivial structural queries are computable by a RRM(CQ). Consider the following two natural number queries on directed graphs: SHORTCYCLE, mapping G to the length of the shortest cycle if G is cyclic, and to 0 otherwise; and LONGCHAIN, mapping G to the length of the longest chain if G is acyclic, and to 0 otherwise. We have:

Proposition 5. *Natural number queries SHORTCYCLE and LONGCHAIN are computable by a RRM(CQ).*

Proof. We pose the following queries for progressively larger values of n:

$$Q^n_{\text{chain}}() \leftarrow R(x_1, x_2), \ldots, R(x_{n-1}, x_n)$$
$$Q^n_{\text{cycle}}() \leftarrow R(x_1, x_2), \ldots, R(x_{n-1}, x_n), R(x_n, x_1)$$

If the graph is cyclic, Q^n_{cycle} will become true at some point, at which the current value of n gives us the value of SHORTCYCLE; if the graph is acyclic, Q^n_{chain} will become false at some point, at which the current value of n gives us the value of LONGCHAIN.

6 Concluding remarks

A natural open question is, following the notation from Section 4, whether $\#\text{CQ}(\neq) < \#\text{CQ}(\neg)$. More generally, the expressibility of counts of queries in one language using counts of queries in another language seems to give rise to many interesting and mathematically challenging questions, with clear links to finite model theory [6]. We have only scratched the surface.

Another natural open question, given the popularity of conjunctive queries, is to understand the exact power of RRM(CQ).

It could also be worthwhile to study complexity issues in database interrogation using conjunctive queries, such as bounds on the number of queries needed to achieve a certain goal, and the size of these queries. Results of this nature for non-conjunctive queries have already been presented by Abiteboul, Papadimitriou and Vianu [2] and Tyszkiewicz [8, 9].

Acknowledgment

We thank Jerzy Tyszkiewicz and Andrzej Szalas for inspiring discussions.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. S. Abiteboul, C.H. Papadimitriou, and V. Vianu. Reflective relational machines. *Information and Computation*, 143(2):110–136, 1998.
3. N.R. Adam and J.C. Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.
4. M. Davis. *Computability and Unsolvability*. McGraw-Hill, 1958.
5. D. Dobkin, A.K. Jones, and R.J. Lipton. Secure databases: Protection against user influence. *ACM Transactions on Database Systems*, 4(1):97–106, 1979.
6. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, second edition, 1999.
7. J.M. Turull Torres. Reflective relational machines working on homogeneous databases. In *Foundations of Information and Knowledge Systems*, volume 1762 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2000.
8. J. Tyszkiewicz. Queries and algorithms computable by polynomial time existential reflective machines. *Fundamenta Informaticae*, 32:91–105, 1997.
9. J. Tyszkiewicz. Computability by sequences of queries. *Fundamenta Informaticae*, 48:389–414, 2001.
10. J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, 1988.
11. J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume II. Computer Science Press, 1989.
12. J.D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.