# Containment of Conjunctive Queries over Databases With Null Values*

Carles Farré[1], Werner Nutt[2], Ernest Teniente[1], and Toni Urpí[1]

[1] Departament de Llenguatges i Sistemes Informatics
Unversitat Politècnica de Catalunya, c/ Jordi Girona, 1–3
08034-Barcelona, Spain
{farre,teniente,urpi}<at>lsi.upc.edu
[2] Faculty of Computer Science, Free University of Bozen-Bolzano
Dominikanerplatz 3, I-39100 Bozen, Italy
nutt<at>inf.unibz.it

**Abstract.** We study containment of conjunctive queries that are evaluated over databases that may contain tuples with null values. We assume the semantics of SQL for single block queries with a `SELECT DISTINCT` clause. This problem ("null containment" for short) is different from containment over databases without null values and sometimes more difficult.
We show that null-containment for boolean conjunctive queries is NP-complete while it is $\Pi_2^{\mathrm{P}}$-complete for queries with distinguished variables. However, if no relation symbol is allowed to appear more than twice, then null-containment is polynomial, as it is for databases without nulls. If we add a unary test predicate `IS NULL`, as it is available in SQL, then containment becomes $\Pi_2^{\mathrm{P}}$-hard for boolean queries, while it remains in $\Pi_2^{\mathrm{P}}$ for arbitrary queries.

## 1 Introduction

Containment of queries is a key topic in database theory. The main motivation, which was already at the origin of containment studies, is query optimization. In their seminal paper, Chandra and Merlin developed a containment-based technique to minimize the number of joins in a query while retaining equivalence [3]. Other problems for which containment is relevant include transaction management [11], query rewriting [10], and verification of integrity constraints [6].

The study of query containment started off with conjunctive queries. Since then, the work has been extended to a wealth of query types, such as conjunctive queries with comparisons [8, 18], queries with union and difference [15], datalog queries [16], conjunctive queries with negated atoms [11, 17], aggregate queries [5], queries over semistructured data [1, 2], and XPath queries [12].

Containment has been studied under several semantics. In most cases, authors assume that queries are evaluated under set semantics, that is, a query

* This work was supported by the British EPSRC (Grant GR/SS44830/01 MAGIK-I) and the Spanish Ministerio de Educacion y Ciencia (Grant TIN 2005-05406).

returns each answer item only once. Chaudhuri and Vardi considered containment under bag semantics, which allows tuples to occur more than once in an answer and is the semantics implemented in SQL database systems [4]. Another line of research considers the effect of integrity constraints such as functional dependencies or foreign key constraints on containment, which restrict the class of databases to consider [13].

All this work did not take into account null values, which are the means by which incomplete information is represented in SQL databases. In the presence of null values, SQL queries are evaluated in a way that makes the existing theory of containment inapplicable. The semantics of single block `SELECT-FROM-WHERE` queries is as follows [7]:

- a query returns values for those combinations of tuples for which the `WHERE` clause evaluates to *true*;
- an equality or a comparison involving `null` has the logical value *unknown*;
- a conjunction of conditions has the logical value *true* only if all conjuncts evaluate to *true*.

*Example 1.* As an example, consider the two queries $Q$, $Q'$, which use a relation with the schema `residence`(`loc`, `person`):

```
Q: SELECT DISTINCT r1.loc
   FROM   residence r1, residence r2
   WHERE  r1.loc = r2.loc

Q': SELECT DISTINCT r.loc
   FROM   residence r
```

According to the SQL semantics, the second query returns the projection of `residence` onto the attribute `loc`, which may include the value `null`. The first query, however, returns only `loc`-values of `residence` that pass the test "`r1.loc = r2.loc`". In other words, $Q$ returns the non-null values in the projection of `residence` onto `loc`. As a consequence, $Q$ and $Q'$ are equivalent over databases that do not contain nulls. However, in the presence of nulls, $Q$ is contained in $Q'$, but not the other way round. □

In the present paper we study null containment of conjunctive queries, that is, containment under set semantics over databases that contain null values, where conditions involving nulls are evaluated as in SQL. Such queries can be equivalently expressed in SQL as single block queries with the keyword `DISTINCT` in the `SELECT` clause. Note that this is also the semantics of nested subqueries in `EXISTS` or `IN` clauses and of subqueries that are combined by the boolean operators `UNION`, `INTERSECTION`, or `MINUS`.

In Section 2, we fix our notation. Section 3 presents a general criterion for checking null-containment. In Section 4, we introduce J-homomorphisms, and show that the existence of a J-homomorphism is sufficient for null-containment while for boolean queries it is also necessary. We prove in Section 5 that null-containment is $\Pi_2^{\mathrm{P}}$-complete in general, while we show in Section 6 that it is

polynomial for queries with at most two occurrences of each predicate. Finally, in Section 7, we model SQL's built-in predicate `IS NULL` and show that in the presence of such null tests, null-containment becomes $\Pi_2^{\mathrm{P}}$-hard for boolean queries while it remains in $\Pi_2^{\mathrm{P}}$ in the general case.

## 2 Preliminaries

A *term* (like $s$, $t$) is either a *constant* (like $c$, $d$) or a *variable* (like $u$, $v$,..., $z$). A *predicate* symbol (like $p$, $q$, $r$) has an arity, which may be 0. An *atom* has the form $p(s_1, \ldots, s_n)$, where $p$ is a predicate symbol with arity $n$. Denoting *tuples* of terms as $\bar{s}$ (and tuples of constants and variables as $\bar{c}$, $\bar{d}$, and $\bar{u}$, $\bar{v}$, etc.) we sometimes write atoms as $p(\bar{s})$. An atom is *ground* if it does not contain variables. If $a = p(s_1, \ldots, s_n)$ is an atom and $j \in [1, n]$, then $a[j]$ denotes the term occurring at position $j$ in $a$, that is, $a[j] = s_j$.

A *condition* $B$ is a list of atoms, written as $B = a_1, \ldots, a_n$, where $n \geq 0$. We sometimes view a condition as a set of atoms. In particular, if $B'$ and $B$ are conditions, we write $B' \subseteq B$ to express that each atom of $B'$ occurs among the atoms of $B$.

A *conjunctive query* is a rule of the form $q(\bar{x}) \leftarrow B$, where $\bar{x}$ is a tuple of distinct variables. We often identify the query and the head predicate $q$, defined by the query. A conjunctive query is *boolean* if the head predicate has the arity 0. Distinguished and nondistinguished variables of $q$ are defined as usual.

A *database* $\mathcal{D}$ is a finite set of ground atoms. The *carrier* of $\mathcal{D}$ is the set of constants occurring in the atoms of $\mathcal{D}$. An *assignment* over $\mathcal{D}$, say $\delta$, for the query $q(\bar{x}) \leftarrow B$ is a mapping from the set of variables of $q$ to the carrier of $\mathcal{D}$. For a constant $c$ we define $\delta c := c$. Assignments are extended in the obvious way to complex syntactic objects such as tuples, atoms, conditions, etc. An atom $a$ is *satisfied* by $\delta$ over $\mathcal{D}$ if $\delta a \in \mathcal{D}$ and a condition $B$ is *satisfied* by $\delta$ if $\delta B \subseteq \mathcal{D}$. We say that $B$ is satisfied over $\mathcal{D}$ if it is satisfied by some $\delta$ over $\mathcal{D}$. For a boolean query $q() \leftarrow B$, we say that $q$ is satisfied over $\mathcal{D}$ if the body $B$ is satisfied over $\mathcal{D}$.

A tuple of constants $\bar{d}$ is an *answer* over $\mathcal{D}$ to the query $q(\bar{x}) \leftarrow B$ if there exists an assignment $\delta$ such that (1) $\delta$ satisfies $B$, and (2) $\delta\bar{x} = \bar{d}$. The set of all answers to $q$ over $\mathcal{D}$ is denoted as $q^{\mathcal{D}}$. Let $q$, $q'$ be queries with the same arity. Then $q$ is contained in $q'$, written $q \subseteq q'$, if $q^{\mathcal{D}} \subseteq q'^{\mathcal{D}}$ for all databases $\mathcal{D}$.

A *substitution* is a mapping from a set of variables to a set of terms. Substitutions can be naturally extended to atoms and conditions. Let $B$, $B'$ be conditions. A substitution $\gamma$ for the variables of $B'$ is a *condition homomorphism* from $B'$ to $B$ if $\gamma B' \subseteq B$. Suppose that $q$, $q'$ are defined as $q(\bar{x}) \leftarrow B$, $q'(\bar{x}) \leftarrow B'$, respectively. Then $\gamma$ is a *query homomorphism* from $q'$ to $q$ if $\gamma$ is a condition homomorphism from $B'$ to $B$ and $\gamma\bar{x} = \bar{x}$. The Homomorphism Theorem for conjunctive queries says that $q$ is contained in $q'$ if and only if there exists a query homomorphism from $q'$ to $q$ [3].

## 3 Null-Containment

We adapt the framework of Section 2 to capture query evaluation over databases with nulls. We introduce a new constant $\perp$ to model the value `null` in SQL. The value $\perp$ may occur in databases, but not in queries.

Let $B$ be a condition and $y$ be a variable occurring in $B$. We say that $y$ is a *join variable* of $B$ if $y$ has at least two occurrences in $B$ and a *singleton variable* otherwise. An assignment $\delta$ *satisfies* $B$ over $\mathcal{D}$ if (1) $\delta B \subseteq \mathcal{D}$ and (2) $\delta$ *respects join variables,* that is $\delta y \neq \perp$ for every join variable $y$ of $B$. Note that this definition captures SQL's semantics of equalities involving `null`. A variable occurring at two positions in $B$ corresponds in SQL notation to an equality between two (not necessarily distinct) attributes, which is only satisfied if the values of the attributes are identical and not `null`.

The set of answers to a query $q$ over a database with nulls is defined as before and is denoted in the same way as $q^{\mathcal{D}}$. We say that $q$ is *null-contained* in $q'$ and write $q \subseteq_{\perp} q'$ if $q^{\mathcal{D}} \subseteq q'^{\mathcal{D}}$ for all databases $\mathcal{D}$, where $\mathcal{D}$ may contain the value $\perp$.

*Example 2.* The two rule-based queries

$$q(x) \leftarrow \texttt{residence}(x, y), \texttt{residence}(x, w)$$
$$q'(x) \leftarrow \texttt{residence}(x, y)$$

are translations of the SQL queries $Q$, $Q'$ in Example 1. We see that the equality in the `WHERE` clause of $Q$ is reflected by the join variable $x$ in $q$. □

Clearly, null-containment implies containment. However, as seen in Example 1, the converse is not true. This raises the question in which way we can check null-containment and how difficult it is to decide this problem.

In the non-null case, a standard approach to checking whether $q'$ contains $q$ is to turn $q$ into the *canonical database* $\mathcal{D}_q$, obtained from $q$ by "freezing" the variables into constants (see for instance [17]). For instance, the canonical database for the query $q$ in Example 2 is $\mathcal{D}_q = \{\texttt{residence}(x, y), \texttt{residence}(x, w)\}$ where for the sake of simplicity we have identified the variables of $q$ with their frozen versions. We will do so also in the rest of the paper as long as no misunderstanding can arise. Clearly, $q$ returns the tuple consisting of the frozen distinguished variables over $\mathcal{D}_q$. The test criterion says that $q$ is contained in $q'$ if also $q'$ returns this tuple over $\mathcal{D}_q$. Note that for boolean queries this tuple is the empty tuple ().

Example 2 shows that this test cannot be used to decide null-containment because $x \in q^{\mathcal{D}_{q'}}$, although $q$ is not null-contained in $q'$. Fortunately, we can modify the test so that it can be applied to the case of databases with nulls. A *null version* of $\mathcal{D}_q$ is a database $\mathcal{D}$ obtained from $\mathcal{D}_q$ by replacing some frozen variables of $q$ with $\perp$. By some slight abuse of notation we represent null versions of $\mathcal{D}_q$ as instantiations $\theta\mathcal{D}_q$, where $\theta$ is a substitution that replaces some frozen non-join variables of $q$ with $\perp$ and is the identity otherwise.

**Theorem 1 (General Criterion).** *Let $q(\bar{x})$, $q'(\bar{x})$ be conjunctive queries. Then $q$ is null-contained in $q'$ if and only if for every null version $\theta\mathcal{D}_q$ of $\mathcal{D}_q$, we have that $q'$ returns the tuple $\theta\bar{x}$ over $\theta\mathcal{D}_q$.*

*Proof.* The proof of the theorem is straightforward. Clearly, if $q$ is null-contained in $q'$, then $q'$ returns $\theta\bar{x}$ over $\theta\mathcal{D}_q$ because $q$ returns $\theta\bar{x}$ over $\theta\mathcal{D}_q$.

Conversely, suppose $\mathcal{D}$ is an arbitrary database with nulls and $q$ returns an answer $\bar{d}$ over $\mathcal{D}$. Let $B$, $B'$ be the bodies of $q$, $q'$, respectively. Then there is an assignment $\delta$ from the variables of $B$ to the carrier of $\mathcal{D}$ such that $\delta$ respects join variables, $\delta B \subseteq \mathcal{D}$, and $\delta\bar{x} = \bar{d}$. We define a substitution $\theta$ such that $\theta y = \bot$ if $\delta y = \bot$ and $\theta$ is the identity otherwise. Thus, also $\theta$ respects join variables.

Due to our hypothesis, there is an assignment $\eta$ for the variables of $B'$ such that $\eta$ respects join variables, $\eta B' \subseteq \theta\mathcal{D}_q$, and $\eta\bar{x} = \bar{x}$. We can view $\eta$ also as a substitution if we identify variables and their frozen versions. Let $\delta' := \delta\eta$. Then it follows that $\delta'$ respects join variables because $\eta$ does so and because $\delta$ maps no variable in $\theta B$ to $\bot$. Moreover, $\delta' B' = \delta\eta B' \subseteq \delta B \subseteq \mathcal{D}$, and finally $\delta'\bar{x} = \delta\eta\bar{x} = \delta\bar{x} = \bar{d}$. This shows that $\delta'$ satisfies $q'$ over $\mathcal{D}$ and $\delta'\bar{x} = \bar{d}$. □

**Corollary 1 (Upper Complexity Bound).** *Null-containment of conjunctive queries is in $\Pi_2^{\mathrm{P}}$.*

*Proof.* According to Theorem 1, we can check that $q$ is *not* null-contained in $q'$ by exhibiting a null-version $\theta\mathcal{D}_q$ of $\mathcal{D}_q$ where $q'$ does not retrieve $\theta\bar{x}$. Deciding whether $q'$ retrieves a specific tuple over a database with nulls is in NP. Thus, the complement of null-containment is in $\Sigma_2^{\mathrm{P}}$. □

*Example 3.* As a continuation of Example 2, consider the null-version $\mathcal{D}'_0 := \{\texttt{residence}(\bot, \bot)\}$ of $\mathcal{D}_{q'}$. According to our definition of query answers, we have that $q^{\mathcal{D}'_0} = \emptyset$, hence $q'$ is not null-contained in $q$.

## 4 Homomorphisms that Respect Join Variables

The general criterion of Theorem 1 is prohibitively complex. Therefore, we look for a simpler test, which may not completely characterize null-containment, but may serve as a sufficient criterion.

We say that a homomorphism $\gamma$ from condition $B'$ to $B$ *respects join variables* if $\gamma$ maps no join variable of $B'$ to a singleton variable of $B$, that is, $\gamma$ maps join variables to join variables or constants. A query homomorphism from $q'$ to $q$ is a *J-homomorphism* if it respects the join variables of the body of $q'$.

**Proposition 1 (Sufficiency).** *Let $q$, $q'$ be conjunctive queries. If there exists a J-homomorphism from $q'$ to $q$, then $q$ is null-contained in $q'$.*

The proof resembles the one that existence of a homomorphism is a sufficient condition for containment.

**Proposition 2 (NP-Completeness).** *Existence of J-homomorphisms is* NP*-complete. This holds already for boolean conjunctive queries.*

The proof reduces containment to the existence of a J-homomorphism.

For the discussion of boolean queries we introduce some extra notation. For a query $q$, the substitution that maps every singleton variable of $q$ to $\perp$ is denoted as $\theta_\perp$. The corresponding null version $\theta_\perp \mathcal{D}_q$ is denoted as $\mathcal{D}_q^\perp$.

**Theorem 2 (Characterization for Boolean Queries).** *Let $q$, $q'$ be boolean conjunctive queries. Then $q$ is null-contained in $q'$ if and only if there exists a J-homomorphism from $q'$ to $q$.*

*Proof.* We know by Proposition 1 that existence of a J-homomorphism is a sufficient condition. It remains to show the necessity.

Suppose that $q$, $q'$ have the form $q() \leftarrow B$, $q'() \leftarrow B'$, respectively and that $q \subseteq_\perp q'$. By Theorem 1, there exists an assignment $\eta$ for the variables of $q'$ that satisfies $q'$ over $\mathcal{D}_q^\perp$. We will use $\eta$ to construct a J-homomorphism $\gamma$ from $q'$ to $q$.

We identify the substitution $\theta_\perp$ with the mapping that maps every atom $a \in B$ to the atom $\theta_\perp a \in \mathcal{D}_q^\perp$. Similarly, we identify $\eta$ with the mapping that maps every atom $a' \in B'$ to $\eta a' \in \mathcal{D}_q^\perp$. The homomorphism $\gamma$ will be defined in such a way that $\eta = \theta_\perp \gamma$.

We first define $\gamma$ as a mapping from the atoms of $B'$ to the atoms of $B$ and then show that $\gamma$ is induced by a substitution. We choose $\gamma$ as an arbitrary mapping that maps an atom $a' \in B'$ to an atom $a \in B$ such that $\eta a' = \theta_\perp a$. In other words, $\gamma$ has the property that $\gamma a' \in \theta_\perp^{-1}(\eta(a'))$ for every atom $a' \in B'$.

It follows from the definition that the relation symbols of $a'$ and $\gamma(a')$ are identical. Moreover, if $a'[i] = c$ for some constant $c$, then $\eta(a')[i] = c$ and also $a[i] = c$ for every $a \in \theta_\perp^{-1}(\eta(a'))$, hence $\gamma(a')[i] = c$.

Now, consider two distinct atoms $a'$, $b' \in B'$ such that $a'[i] = b'[j] = y$ for some variable $y$. It follows that $\eta(a')[i] = \eta(b')[j] = \eta y$. Since $y$ is a join variable, we have $\eta y = s$ for some term $s \neq \perp$. Let $a := \gamma a'$ and $b := \gamma b'$. If $s$ is a constant $c$, then $a[i] = b[j] = c$ by the definition of $\theta_\perp$. If $s = z$ for a (frozen) variable $z$, then $z$ is a join variable of $B$ and the definition of $\theta_\perp$ implies that $a[i] = b[j] = z$.

Thus, $a'[i] = b'[j]$ implies that $\gamma(a')[i] = \gamma(b')[j]$ for all atoms $a'$, $b' \in B'$ and all positions $i$, $j$. Hence, $\gamma$ is induced by a homomorphism, which we call $\gamma$, too. Also, since $\theta_\perp \gamma = \eta$ and $\eta$ respects join variables, it follows that $\gamma$ respects join variables. □

Combining Proposition 2 and Theorem 2, we can precisely characterize the complexity of null-containment for boolean queries.

**Corollary 2 (Complexity for Boolean Queries).** *For boolean conjunctive queries null-containment is* NP*-complete.*

A closer inspection of the proof of Theorem 2 reveals that for boolean queries the general containment test of Theorem 1 can be simplified to one that uses only a single test database.

**Proposition 3.** *Let $q()$, $q'()$ be boolean conjunctive queries. Then $q$ is null-contained in $q'$ if and only if $q'$ is satisfied by $\mathcal{D}_q^\perp$.*

## 5 Complexity of Null-Containment

The next example shows that the existence of a J-homomorphism is not a necessary condition for null-containment of general conjunctive queries.

*Example 4.* We consider the following two queries:

$$\begin{aligned} q(x) \leftarrow\ & p(x, y_1, z_1),\ p(x_2, y_2, z_2),\ p(x_3, y_3, x_3),\\ & r(y_1, z_1),\ r(y_1, z_2),\ r(y_2, x_3)\\ q'(x) \leftarrow\ & p(x, v_1, w_1),\ p(u, v_2, w_2),\ p(u, v_3, w_3),\\ & r(v_1, w_2). \end{aligned}$$

To simplify our discussion we denote the atoms of $q$ as $a_1$, $a_2$, $a_3$, $b_1$, $b_2$, $b_3$ and the atoms of $q'$ as $a'_1$, $a'_2$, $a'_3$, $b'$, respectively. Thus, the two queries can be written as

$$\begin{aligned} q(x) &\leftarrow a_1,\ a_2,\ a_3,\ b_1,\ b_2,\ b_3\\ q'(x) &\leftarrow a'_1,\ a'_2,\ a'_3,\ b'. \end{aligned}$$

One easily checks that there exist exactly two query homomorphisms from $q'$ to $q$. To see this, note that there is no choice in mapping $a'_1$, since $x$ has to be mapped to $x$. Then, there are two choices to map $b'$, namely either to $b_1$ or to $b_2$. Depending on this choice, there is only one possibility to map $a'_2$ and $a'_3$. In conclusion, the two mappings map the atoms of $q'$ to the atoms of $q$ as follows:

$$\begin{aligned} \gamma_1 &= \{a'_1 \mapsto a_1,\ a'_2 \mapsto a_1,\ a'_3 \mapsto a_1,\ b' \mapsto b_1\}\\ \gamma_2 &= \{a'_1 \mapsto a_1,\ a'_2 \mapsto a_2,\ a'_3 \mapsto a_2,\ b' \mapsto b_2\}. \end{aligned}$$

None of the two mappings preserves the join variable $u$, since $\gamma_1 u = x$, and $\gamma_2 u = x_2$. There exists, however, a third substitution that is a homomorphism from the body of $q'$ to the body of $q$, but which fails to be a query homomorphism, since it maps the distinguished variable $x$ to $x_2$. This is the mapping

$$\gamma_3 = \{a'_1 \mapsto a_2,\ a'_2 \mapsto a_3,\ a'_3 \mapsto a_3,\ b' \mapsto b_3\}.$$

Note that $q'$ has three join variables, namely $u$, $v_1$, and $w_2$, and that all three substitutions map $v_1$ and $w_2$ to join variables of $q$. Moreover, $\gamma_3$ also maps $u$ to the join variable $x_3$.

We will see in the following that $q$ is null-contained in $q'$. Let $\mathcal{D}$ be a database and $d$ an answer retrieved by $q$ over $\mathcal{D}$. Then there is an assignment $\delta$ such that $\delta x = d$ and $\delta$ satisfies the body of $q$. We distinguish between three cases and show that in each case also $q'$ retrieves $d$:

1. If $\delta x \neq \bot$, then $\delta\gamma_1$ satisfies the body of $q'$ and $\delta\gamma_1 x = d$.
2. If $\delta x = \bot$ and $\delta x_2 \neq \bot$, then $\delta\gamma_2$ satisfies the body of $q'$ and $\delta\gamma_2 x = d$.
3. If $\delta x = \bot$ and $\delta x_2 = \bot$, then $\delta\gamma_3$ satisfies the body of $q'$ and $\delta\gamma_3 x = \delta x_2 = \bot = \delta x = d$.

Note that in the third case the reason why $q'$ retrieves $\perp$ is that $\delta$ binds the non-distinguished variable $x_2$ to $\perp$ instead of binding $x$ to $\perp$. □

The queries of Example 4 will be a crucial ingredient for a reduction that proves $\Pi_2^{\mathrm{P}}$-hardness of null-containment for conjunctive queries.

We reduce the problem of deciding the validity of quantified boolean formulas with a prefix of the form $\forall^*\exists^*$ to the null-containment problem. We note that the former problem is already $\Pi_2^{\mathrm{P}}$-complete if the matrix is a conjunction of 3-clauses. Let

$$\Phi = \forall y_1 \ldots \forall y_l \exists z_1 \ldots \exists z_m \phi_1 \wedge \ldots \wedge \phi_n \tag{1}$$

be such a formula, where each $\phi_k$ is a 3-clause containing variables among the $y_i$ and $z_j$.

We denote the values "true" and "false" as $\mathsf{t}$ and $\mathsf{f}$ and we identify the truth values with constants that are interpreted as "true" and "false", respectively.

The validity of $\Phi$ can be rephrased as the satisfiability of a set of formulas derived from $\Phi$. Let $\alpha\colon \{y_1, \ldots, y_l\} \to \{\mathsf{t}, \mathsf{f}\}$ be a truth value assignment. If $\psi$ is a propositional formula, we denote with $\psi^\alpha$ the formula obtained from $\psi$ by replacing each variable $y_i$ with the constant $\alpha y_i$. Let us denote the matrix of $\Phi$ as $\phi := \phi_1 \wedge \ldots \wedge \phi_n$. Then $\Phi$ is valid if and only if for every $\alpha\colon \{y_1, \ldots, y_l\} \to \{\mathsf{t}, \mathsf{f}\}$ the formula $\phi^\alpha$ is satisfiable.

We construct two conjunctive queries $q$, $q'$ such that $q$ is null-contained in $q'$ if and only if $\Phi$ is valid. The two queries have the form

$$q(x_1, \ldots, x_l) \leftarrow G_1, \ldots, G_l, C_1, \ldots, C_n \tag{2}$$
$$q'(x_1, \ldots, x_l) \leftarrow G'_1, \ldots, G'_l, C'_1, \ldots, C'_n, \tag{3}$$

with conditions $G_k$, $G'_k$, $C_j$, $C'_j$. Intuitively, the pair of conditions $C_j$, $C'_j$ encodes which bindings of the variables in the clause $\phi_j$ actually satisfy $\phi_j$ while the pair $G_k$, $G'_k$ together with the distinguished variable $x_i$ generates bindings of $y_i$ to $\mathsf{t}$ and $\mathsf{f}$.

We first define the $C_k$ and $C'_k$. Let $u_1, u_2, u_3$ be the variables occurring in $\phi_k$. We introduce a new ternary relation symbol $cl_k$ and define

$$C'_k := cl_k(u_1, u_2, u_3). \tag{4}$$

Out of the eight possible truth value assignments for the three variables, there are seven, say $\beta_1, \ldots, \beta_7$, that satisfy the clause $\phi_k$. We define

$$C_k := cl_k(\beta_1 u_1, \beta_1 u_2, \beta_1 u_3),\ \ldots,\ cl_k(\beta_7 u_1, \beta_7 u_2, \beta_7 u_3). \tag{5}$$

For instance, if

$$\phi_k = \neg y_2 \vee y_3 \vee \neg z_1,$$

then only the assignment $\{y_2 \mapsto \mathsf{t},\ y_3 \mapsto \mathsf{f},\ z_1 \mapsto \mathsf{t}\}$ does not satisfy $\phi_k$. Hence,

$$\begin{aligned} C'_k &= cl_k(y_2, y_3, z_1) \\ C_k &= cl_k(\mathsf{f},\mathsf{f},\mathsf{f}),\ cl_k(\mathsf{f},\mathsf{f},\mathsf{t}),\ cl_k(\mathsf{f},\mathsf{t},\mathsf{f}),\ cl_k(\mathsf{f},\mathsf{t},\mathsf{t}), \\ &\quad\ cl_k(\mathsf{t},\mathsf{f},\mathsf{f}),\ cl_k(\mathsf{t},\mathsf{t},\mathsf{f}),\ cl_k(\mathsf{t},\mathsf{t},\mathsf{t}). \end{aligned}$$

Consider a substitution $\theta$ for the variables $y_i$, $z_j$, where $i \in [1, l]$ and $j \in [1, m]$, such that $\theta$ maps every variable either to $\mathtt{t}$ or to $\mathtt{f}$. Obviously, such a substitution can be viewed as a truth value assignment for the variables and vice versa. Moreover, by construction we have for each $k \in [1, n]$ that $\theta C'_k \subseteq C_k$ if and only if the corresponding assignment satisfies $\phi_k$.

Next, consider a fixed index $i \in [1, l]$. We define the $G_i$ and $G'_i$ by modifying the bodies of the queries in Example 4. More specifically, the body of $q$ will give rise to $G_i$ and the body of $q'$ to $G'_i$. To this end, we introduce a ternary relation $p_i$ and a binary relation $r_i$ and turn every atom for $p$ and $r$ into one for $p_i$ and $r_i$, respectively. We do so by renaming the output variable $x$ as $x_i$, instantiating $y_1$ and $y_2$ by $\mathtt{t}$ and $\mathtt{f}$, respectively. and renaming $v_1$ as $y_i$. All other variables are renamed by adding the number $i$ to their subscript. Thus, $G_i$ and $G'_i$ look as follows

$$\begin{aligned} G_i &= p_i(\boxed{x_i}, \boxed{\mathtt{t}}, z_{i1}),\ p_i(x_{i2}, \boxed{\mathtt{f}}, z_{i2}),\ p_i(x_{i3}, y_{i3}, x_{i3}),\\ &\quad r_i(\boxed{\mathtt{t}}, z_{i1}),\ r_i(\boxed{\mathtt{t}}, z_{i2}),\ r_i(\boxed{\mathtt{f}}, x_{i3})\\ G'_i &= p_i(\boxed{x_i}, \boxed{y_i}, w_{i1}),\ p_i(u_i, v_{i2}, w_{i2}),\ p_i(u_i, v_{i3}, w_{i3}),\\ &\quad r_i(y_i, w_{i2}), \end{aligned}$$

where we have highlighted the terms that have been introduced as replacements of $x$, $y_1$, $y_2$ and $v_1$. It follows from the discussion of Example 4, that any homomorphism from $G'_i$ to $G_i$ either maps $y_i$ to $\mathtt{t}$ or to $\mathtt{f}$ and that the homomorphisms mapping $y_i$ to $\mathtt{t}$ are exactly the ones that map $x_i$ to $x_i$.

**Lemma 1.** *Let $\Phi$ be a quantified boolean formula as in Equation (1) and $q$, $q'$ be a pair of conjunctive queries encoding $\Phi$ as in Equations (2) and (3). Then*

$$\Phi \text{ is valid} \qquad \Longleftrightarrow \qquad q \subseteq_\perp q'.$$

**Theorem 3 (Complexity of Null-Containment).** *Null-containment of conjunctive queries is $\Pi_2^{\mathrm{P}}$-complete.*

## 6 Binary Queries

A conjunctive query that for every predicate contains at most two atoms with that predicate in its body is called *binary*. Sagiv and Saraiya have shown that containment of binary queries is polynomial [14]. We prove that this extends to null-containment.

We show first that for the class of binary queries the existence of a J-homomorphism is also a necessary condition for null-containment. This holds already if only the containee is binary.

**Theorem 4.** *Let $q$, $q'$ be conjunctive queries such that $q \subseteq_\perp q'$. If $q$ is binary, then there exists a J-homomorphism from $q'$ to $q$.*

*Proof.* If $q$, $q'$ are boolean queries, then the claim follows from Proposition 2. Suppose, therefore, that the queries have the form $q'(\bar{x}) \leftarrow B'$, $q(\bar{x}) \leftarrow B$, where the tuple $\bar{x}$ is nonempty.

Since $q \subseteq_\perp q'$, it follows that $q \subseteq q'$. Hence, there exists a homomorphism from $q'$ to $q$. Let $\gamma_1, \ldots, \gamma_n$ be all the homomorphisms from $q'$ to $q$. We want to show that one of the $\gamma_i$ preserves join variables.

The proof is by contradiction. We assume that each $\gamma_i$ maps some join variable of $B'$ to a singleton variable in $B$. We say a null version $\theta\mathcal{D}_q$ of $\mathcal{D}_q$ is a *witness* for this fact if for every $i \in [1, n]$ there is a join variable $y$ of $B'$ such that $\theta\gamma_i y = \perp$, that is, if every $\gamma_i$ maps some join variable of $B'$ to a singleton variable of $B$ that is mapped to $\perp$ by $\theta$. There is at least one witness, namely the null version $\theta_\perp\mathcal{D}_q$ defined by the substitution $\theta_\perp$ that maps every singleton variable of $B$ to $\perp$.

We introduce a partial order on null versions of $\mathcal{D}_q$ by defining $\theta_1\mathcal{D}_q \preceq \theta_2\mathcal{D}_q$ if $\theta_1 z = \perp$ implies $\theta_2 z = \perp$ for all variables $z$ occurring in $B$. Let $\mathcal{D}_0 := \theta_0\mathcal{D}_q$ be a witness that is minimal with respect to "$\preceq$". Note that, due to the minimality of $\mathcal{D}_0$, for every variable $z$ with $\theta_0 z = \perp$, there is a homomorphism $\gamma_i$ such that $\gamma_i y = z$ for some join variable $y$ of $B'$. If there were a $z$ without this property, then we could redefine $\theta_0 z := z$ while still retaining a witness, contrary to the minimality of $\mathcal{D}_0$.

Clearly, $q$ retrieves $\theta_0\bar{x}$ over $\mathcal{D}_0$. Since $q \subseteq_\perp q'$, there exists an assignment $\eta$ for the variables in $B'$ such that (1) $\eta$ satisfies $B'$ over $\mathcal{D}_0$ and (2) $\eta\bar{x} = \theta_0\bar{x}$. Moreover, since $\mathcal{D}_0$ is a witness, we also have that (3) $\eta \neq \theta_0\gamma_i$ for all $i \in [1, n]$.

The assignment $\eta$ has the property that $\eta B' \subseteq \mathcal{D}_0$ and $\eta$ maps some singleton variables of $B'$ to $\perp$. The database $\mathcal{D}_0$ has been obtained from $\mathcal{D}_q$ by applying $\theta_0$, which maps some frozen singleton variables in $\mathcal{D}_q$ to $\perp$. Thus, each $\perp$ in $\mathcal{D}_0$ replaces a singleton variable in $\mathcal{D}_q$. As a consequence, there is a substitution $\rho$ such that $\rho B' \subseteq B$ and $\eta$ can be factorized as $\eta = \theta_0\rho$.

Thus, $\rho$ satisfies $B'$ over $\mathcal{D}_q$. However, $\rho$ cannot map $\bar{x}$ to $\bar{x}$ because then it would be a query homomorphism. Note that, in general, $\rho$ is not uniquely determined. In summary, the following facts hold for $\rho$: (1) $\rho$ satisfies $B'$ over $\mathcal{D}_q$, (2) $\theta_0\rho$ satisfies $B'$ over $\mathcal{D}_0$, (3) $\theta_0\rho\bar{x} = \theta_0\bar{x}$, and (4) $\rho\bar{x} \neq \bar{x}$.

The above implies that $\rho x \neq x$ for some $x$ in $\bar{x}$. For this $x$ we have that $\theta_0 x = \perp$, since otherwise $\theta_0\rho x = \theta_0 x$ could not hold. Thus, $\theta_0\bar{x}$ has at least one occurrence of $\perp$. Moreover, $\rho x$ is a variable, say $v$, since otherwise $\theta_0\rho x = \theta_0 x = \perp$ could not hold. In summary, there are two singleton variables $x$, $v$ occurring in $B$ such that (1) $x$ occurs in $\bar{x}$, (2) $\rho x = v$, and (3) $\theta_0 x = \theta_0 v = \perp$.

From $\eta x = \theta_0\rho x = \perp$ we conclude that $x$ is also a singleton variable in $B'$. Hence, there is a unique atom $a'$ in $B'$ containing $x$. Given that $x$ is a distinguished variable, there is a unique atom $a_1$ in $B$ such that $a_1 = \gamma_i a'$ for all $i \in [1, n]$, and given that $v \neq x$, there is a unique atom $a_2$ in $B$ such that $a_2 = \rho a'$.

Suppose that $a'$ has the predicate $p$ and that $x$ occurs in $a'$ at position $j$, that is, $a'[j] = x$. Then $a_1$ and $a_2$ have the predicate $p$, too. Moreover, $a_1[j] = x$ and $a_2[j] = v$, which implies that $(\theta_0 a_1)[j] = (\theta_0 a_2)[j] = \perp$ in $\mathcal{D}_0$.

Since $\mathcal{D}_0$ is a minimal witness, $B'$ contains a join variable, say $y$, such that $\gamma_i y = x$ for some $i \in [1, n]$. The variable $x$ being a singleton, we conclude that in $B'$ there is an atom $b'$ such that $b'[j] = y$, and $b'$ has the predicate $p$, too.

Given that $q$ is binary, there are only two atoms in $\mathcal{D}_0$ to which $b'$ can be mapped by the assignment $\eta$, namely $\theta_0 a_1$ and $\theta_0 a_2$. However, both atoms contain the value $\perp$ at position $j$, which contradicts the requirement that $\eta y \neq \perp$, since $y$ is a join variable. □

Next, we introduce a simple transformation of queries that will allow us to reduce the existence check for J-homomorphisms to the one for simple homomorphisms. For every relational query $q(\bar{x})$ we define a query $\hat{q}(\bar{x})$, the *J-transform* of $q$, as follows:

- for every predicate $p$ occurring in the body of $q$ we introduce a new predicate $\hat{p}$ of the same arity;
- for every atom $a = p(\bar{c}, \bar{y}, \bar{z})$ in the body of $q$, where $\bar{c}$ are the constants in $a$, $\bar{y}$ are the join variables in $a$, and $\bar{z}$ are the singleton variables in $a$, we construct the atom $\hat{a} := \hat{p}(\bar{c}, \bar{y}, \bar{w})$, where $\bar{w}$ are fresh variables;
- if $q$ has the body $B$, then $\hat{q}$ has the body $B, \hat{B}$, where $\hat{B}$ contains for every $a \in B$ the corresponding $\hat{a}$.

*Example 5.* Consider the two queries

$$q(x) \leftarrow r(x, z_1)$$
$$q'(x) \leftarrow r(x, z_1), r(x, z_2).$$

The corresponding J-transforms are

$$\hat{q}(x) \leftarrow r(x, z_1),\ \hat{r}(w, w_1)$$
$$\hat{q}'(x) \leftarrow r(x, z_2),\ r(x, z_3),\ \hat{r}(x, w_2),\ \hat{r}(x, w_3).$$

Note that $x$ is a singleton variable in $q$ and therefore the variable $w$ has been introduced as a duplicate of $x$ in $\hat{q}$. □

**Lemma 2 (J-Transform).** *Let $q$, $q'$ be two relational conjunctive queries. There is a J-homomorphism from $q'$ to $q$ if and only if there is a query homomorphism from $\hat{q}'$ to $\hat{q}$.*

**Theorem 5 (Polynomiality for Binary Queries).** *For binary queries, null-containment can be decided in polynomial time.*

*Proof.* By Lemma 2, null-containment of conjunctive queries can be reduced to the containment of their J-transforms. Clearly, the J-transform of a binary query is again binary. As shown in [14], containment can be checked in polynomial time for binary queries. □

## 7 Null Tests

Our SQL-like semantics of conjunctive queries allows us to enforce that a variable $y$ be bound only to non-null values. To see this, suppose that $p(y, \bar{z})$ is an atom in a condition $B$. Let $B'$ be obtained from $B$ by adding an atom $p(y, \bar{w})$, where $\bar{w}$ is a tuple of fresh variables. If $\mathcal{D}$ is a database, then an assignment $\delta$ satisfies $B'$ over $\mathcal{D}$ if and only if $\delta$ satisfies $B$ over $\mathcal{D}$ and $\delta y \neq \bot$.

Also SQL allows one, by writing "`att IS NULL`", to test whether the value of an attribute `att` is null. We model this facility to test for null by introducing a unary built-in predicate *isNull*, which can appear in conditions, but not in databases. Atoms with a predicate other than *isNull* are *relational*. A condition $B$ is *safe* if for every atom *isNull*$(y)$ in $B$ we have that (1) $y$ occurs also in a relational atom of $B$ and (2) $y$ is a singleton variable of $B$. We only consider queries with safe bodies. An assignment $\delta$ *satisfies* *isNull*$(y)$ if $\delta y = \bot$.

Null-containment of conjunctive queries with *isNull* can be decided using null versions of canonical databases as in Theorem 1. The difference is that (1) a canonical database $\mathcal{D}_q$ contains only the frozen *relational* atoms of $q$ and (2) null versions $\theta\mathcal{D}_q$ can only be obtained from substitutions $\theta$ such that $\theta y = \bot$ whenever *isNull*$(y)$ occurs in the body of $q$.

A substitution $\gamma$ is a *homomorphism* or *J-homomorphism* between queries with null tests if $\gamma$ is a homomorphism or J-homomorphism, respectively, when *isNull* is treated like a relational predicate. We say that $\gamma$ is a *relational homomorphism* if $\gamma$ is a homomorphism when we ignore all *isNull* atoms. It is straightforward to check that the existence of a J-homomorphism continues to be a sufficient condition for null-containment.

However, as the following example shows, the existence of a J-homomorphism is no longer a necessary condition for null-containment of boolean queries with null tests. The intuitive reason is that adding null tests allows us to express alternatively that a variable must be bound to null or that it must be bound to a non-null value. This is a form of negation, which forces us to make case analyses when checking containment.

The reader will verify that examples and proofs in this section can be amended in a straightforward way to yield results for other extensions of conjunctive queries that allow one to express the negation of specific atoms. Two examples for such extensions are negation of atoms and comparisons. While it is well-known that containment is $\Pi_2^{\mathrm{P}}$-hard in the presence of comparisons [18], to the best of our knowledge this has not yet been proven for negated atoms.

*Example 6.* Consider the queries

$$\begin{aligned} q() \leftarrow\ & p(c, v_1),\ p(c, v_2),\ p(v_1, v_2),\ p(v_2, v_3), \\ & r(v_1, z_1),\ \mathit{isNull}(z_1), \\ & r(v_2, z_2), \\ & r(v_3, z_3),\ r(z_4, z_3) \end{aligned}$$

$$q'() \leftarrow p(c, u_1),\ p(u_1, u_2),\\ r(u_1, w_1),\ \mathit{isNull}(w_1),\\ r(u_2, w_2),\ r(w_3, w_2).$$

Note that graphically, the first three $p$-atoms of $q$ form a triangle, while the fourth $p$-atom extends the triangle at the variable $v_2$. Each variable $v_i$ is connected to a variable $z_i$ by the predicate $r$. The $p$-atoms in $q'$ form a path of length 2. Analogously to the situation in $q$, each variable $u_j$ is connected by the predicate $r$ to a variable $w_j$. The conditions on $w_1$, $w_2$ in $q'$ require that $u_1$ be connected by $r$ to a null value and $u_2$ to a non-null value. The conditions on $z_1$, $z_2$, $z_3$ in $q$ require that $v_1$ be connected to a null value, $v_3$ to a non-null value, and $v_2$ to a value that may be null, but need not.

Clearly, there is no J-homomorphism from $q'$ to $q$. Such a substitution would have to contain the mappings $[u_1/v_1,\ w_1/z_1,\ u_2/v_2]$. This partial substitution cannot be extended to a J-homomorphism because $w_2$ is a join variable, which cannot be mapped to the singleton variable $z_2$.

Nonetheless, $q \subseteq_\perp q'$, which can be seen by considering the two substitutions

$$\gamma_1 = [u_1/v_1,\ w_1/z_1,\ u_2/v_2,\ w_2/z_2,\ w_3/v_2]$$
$$\gamma_2 = [u_1/v_2,\ w_1/z_2,\ u_2/v_3,\ w_2/z_3,\ w_3/v_3].$$

Clearly, $\gamma_1$ and $\gamma_2$ are relational homomorphisms. Let $\mathcal{D}$ be a database and $\delta$ be an assignment that satisfies $q$ over $\mathcal{D}$. Then one checks that $\delta\gamma_1$ satisfies $q'$ if $\delta z_2 \neq \perp$ and $\delta\gamma_2$ satisfies $q'$ if $\delta z_2 = \perp$. □

The $\Pi_2^{\mathrm{P}}$-hardness proof in this section is based on a similar idea as the one in Section 5. We translate a quantified boolean formula as in Equation 1 into two boolean queries

$$q() \leftarrow H_1, \ldots, H_l, C_1, \ldots, C_n \tag{6}$$
$$q'() \leftarrow H'_1, \ldots, H'_l, C'_1, \ldots, C'_n, \tag{7}$$

where the $C_k$ and $C'_k$ are defined as in Equations (5) and (4), respectively.

The conditions $H_i$, $H'_i$ play the role of generators of assignments for the variables $y_i$ and are defined as modifications of the bodies of the queries in Example 6, obtained by parameterising predicates and variables with the index $i$, and substituting variables $v_2$, $v_3$ in $q$ by $\mathsf{t}$ and $\mathsf{f}$, respectively, and by substituting $u_2$ with $y_i$. Thus, $H_i$ and $H'_i$ look as follows:

$$H_i = p_i(c, v_{i1}),\ p_i(c, \boxed{\mathsf{t}}),\ p_i(v_{i1}, \boxed{\mathsf{t}}),\ p_i(\boxed{\mathsf{t}}, \boxed{\mathsf{f}}),\\ r_i(v_{i1}, z_{i1}),\ \mathit{isNull}(z_{i1}),\ r_i(\boxed{\mathsf{t}}, z_{i2}),\ r_i(\boxed{\mathsf{f}}, z_{i3}),\ r_i(z_{i4}, z_{i3})$$
$$H'_i = p_i(c, u_{i1}),\ p_i(u_{i1}, \boxed{y_i}),\\ r_i(u_{i1}, w_{i1}),\ \mathit{isNull}(w_{i1}),\ r_i(\boxed{y_i}, w_{i2}),\ r_i(w_{i3}, w_{i2}).$$

**Lemma 3.** *Let $\Phi$ be a quantified boolean formula as in Equation (1) and $q$, $q'$ be a pair of boolean queries encoding $\Phi$ as in Equations (6) and (7) Then*

$$\Phi \text{ is valid} \qquad \Longleftrightarrow \qquad q \subseteq_{\perp} q'.$$

**Corollary 3.** *The null-containment problem for boolean conjunctive queries with null tests is $\Pi_2^{\mathrm{P}}$-complete.*

**Corollary 4.** *The containment problem for boolean conjunctive queries with negated atoms is $\Pi_2^{\mathrm{P}}$-hard.*

*Proof.* We modify Example 6 by first dropping all $r$-atoms and all null tests. Then we add in $q$ and in $q'$ the atoms $p'(u_1)$ and $\neg p'(u_2)$. For the new queries, one shows $q \subseteq q'$ by a similar argument as above.

Based on the modified example, we encode a quantified boolean formula as in Equation (1) into a containment problem for conjunctive queries with negated atoms. The reduction and the proof of the corresponding lemma are analogous to the ones for queries with null tests. □

## 8 Conclusion

Query containment has been studied extensively for a variety of query types and semantics. However, the fact that real databases contain null values has been widely ignored by this work. We feel that it is important to understand the effect of null values on containment if one wants to apply containment based techniques in realistic scenarios. Moreover, containment plays a key role in information integration, where it is increasingly likely to encounter data sets with null values after merging heterogeneous data sources.

In the present paper, we have concentrated on relational conjunctive queries because it is in this basic setting that the most crucial differences to the classical non-null results become apparent. A characterization of null-containment in terms of homomorphisms, analogous to the classical case, is only possible for boolean queries, while for queries with distinguished variables null-containment is strictly more complex than containment. A similar characterization, using homomorphisms, exists for queries with at most two atoms per predicate, while an example shows that it no longer holds for queries with three atoms per predicate. Adding an SQL style `IS NULL` test creates a limited form of negation, which resembles the one introduced by comparisons like $y > 1$ and $y \leq 1$, or by the negation of relational atoms. These additional constructs raise complexity to $\Pi_2^{\mathrm{P}}$-completeness, which was already known for comparisons [18].

Containment of conjunctive queries over databases with null values can be reduced to containment of conjunctive queries with disequations over regular databases. The fact that a join variable $x$ cannot be bound to $\perp$ can be expressed by adding to the query body a disequation $x \neq \perp$. Since it is known that containment of conjunctive queries with disequations is $\Pi_2^{\mathrm{P}}$-complete, this yields an alternative proof for Corollary 1 in the present paper. The $\Pi_2^{\mathrm{P}}$ lower bound

for null-containment in Theorem 3 yields also the new result that containment of conjunctive queries with disequations is already $\Pi_2^{\mathrm{P}}$-hard if all disequations are of the form $x \neq c$ with a single constant $c$, which complements the lower bounds in [9, 18].

**Acknowledgment**

We would like to thank an anonymous referee for pointing out the relationship of our work and the one on queries with disequalities.

## References

1. D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. 7th KR*, pages 176–185, 2000.
2. D. Calvanese, G. D. Giacomo, and M. Y. Vardi. Decidable containment of recursive queries. In *Proc. 9th ICDT*, pages 1–18, 2003.
3. A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th STOC*, 1977.
4. S. Chaudhuri and M. Vardi. Optimization of real conjunctive queries. In *Proc. 12th PODS*, 1993.
5. S. Cohen, W. Nutt, and Y. Sagiv. Containment of aggregate queries. In *Proc. 9th ICDT*, 2003.
6. M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Verifying integrity constraints on web-sites. In *Proc. 16th IJCAI*, pages 614–619, 1999.
7. H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Pearson Education International, 2002.
8. A. Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1):146–160, 1988.
9. P. Kolaitis, D. Martin, and M. Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *Proc. 17th PODS*, pages 197–204, 1998.
10. A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. 14th PODS*, pages 95–104, 1995.
11. A. Levy and Y. Sagiv. Queries independent of updates. In *Proc. 19th VLDB*, pages 171–181, 1993.
12. G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. 21st PODS*, pages 65–76, 2002.
13. L. Popa and V. Tannen. An equational chase for path-conjunctive queries, constraints, and views. In *Proc. 7th ICDT*, pages 39–57, 1999.
14. Y. Sagiv and Y. Saraiya. Minimizing restricted-fanout queries. *Discrete Applied Mathematics*, 40:245–264, 1992.
15. Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1981.
16. O. Shmueli. Equivalence of datalog programs is undecidable. *Theoretical Computer Science*, 15(3):231–242, 1993.
17. J. Ullman. Information integration using logical views. In *Proc. 6th ICDT*, pages 19–40, 1997.
18. R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *J. Computer and System Sciences*, 54(1):113–135, 1997.