The Algebra and the Logic for SQL Nulls

Enrico Franconi and Sergio Tessaris

Free University of Bozen-Bolzano, Italy
lastname@inf.unibz.it

Abstract The logic of nulls in databases has been subject of investigation since their introduction in Codd's Relational Model, which is the foundation of the SQL standard. In the logic based approaches to modelling relational databases proposed so far, nulls are considered as representing unknown values. Such existential semantics fails to capture the behaviour of the SQL standard. We show that, according to Codd's Relational Model, a SQL null value represents a non-existing value; as a consequence no indeterminacy is introduced by SQL null values. We show that the domain independent fragment of the extension of first-order logic accounting for predicates with missing arguments is equivalent to Codd's relational algebra with SQL nulls. Moreover, we illustrate a faithful encoding of the logic into standard first-order logic. At the end, we show how to capture in this framework the UNIQUE, PRIMARY KEY, and FOREIGN KEY constraints as defined in the SQL:1999 standard.

1 Relational Databases and SQL Null Values

Consider a database instance with null values over the relational schema $\{R/2\}$, and an SQL query asking for the tuples in R being equal to themselves:

R :

1	2
a	b
b	**N**

```
SELECT * FROM R
WHERE R.1 = R.1 AND R.2 = R.2 ;
```

$\Longrightarrow$

```
 1 | 2
---+---
 a | b
(1 row)
```

Figure 1.

In SQL, the query above returns the table `R` if and only if the table `R` does not have any null value, otherwise it returns just the tuples not containing a null value, i.e., in this case only the first tuple $\langle a, b\rangle$. Informally this is due to the fact that an SQL null value is never equal (or not equal) to anything, including itself. How can we formally capture this behaviour?

We introduce a formal semantics for SQL null values in order to capture exactly the behaviour of SQL queries and SQL constraints in presence of null values. We restrict our attention to the first-order fragment of SQL – e.g., we do not consider aggregates, and both queries and constraints should be set based (as opposed to bag/multi-set based): this fragment can be expressed, in absence of null values, into the standard relational algebra $\mathcal{RA}$.

To the best of our knowledge, there has been no attempt to formalise in logic a relational algebra with *SQL* null values. It is well known that SQL null values require a special semantics. Indeed, if they were treated as standard database constants, the direct translation in the standard relational algebra $\mathcal{RA}$ of the above SQL query would be equivalent to the *identity* expression for R: $\sigma_{1=1}$ $\sigma_{2=2}$ R, giving as an answer the table R itself, namely the tuples $\{\langle a, b\rangle, \langle b, \mathbf{N}\rangle\}$. However, we have seen that the expected answer to this query is different.

The most popular semantics in the literature for null values is the one interpreting null values with an existential meaning, namely a null value denotes an object which exists but has an unknown identity: this is the semantics of *naive* tables (see [1]). It is easy to see that also with this semantics, the direct translation of the above SQL query in the standard relational algebra $\mathcal{RA}$ over naive tables would be equivalent to the *identity* expression for R: $\sigma_{1=1}$ $\sigma_{2=2}$ R, giving as an answer the table R itself, namely the tuples $\{\langle a, b\rangle, \langle b, \mathbf{N}\rangle\}$. And again, we have seen above that the expected answer to this query is different.

2 Database Instances with Null Values

We introduce the notions of tuple, relation, and database instance in presence of null values. We consider the unnamed (positional) perspective for the attributes of tuples: elements of a tuple are identified by their position within the tuple.

Given a set of domain values Δ, an n-tuple is a *total* function from integers from 1 up to n (the position of the element within the tuple) into the set of domain values Δ augmented by the special term $\mathbf{N}$ (the null value): if we denote the set $\{i \in \mathbb{N} \mid 1 \leq i \wedge i \leq n\}$, possibly empty if $n = 0$, as $[1 \cdots n]$, then an n-tuple is a total function $[1 \cdots n] \mapsto \Delta \cup \{\mathbf{N}\}$. Note that the zero-tuple is represented by a *constant* zero-ary function that we call $\{\}$. For example, the tuple $\langle b, \mathbf{N}\rangle$ is represented as $\{1 \mapsto b, 2 \mapsto \mathbf{N}\}$, while the zero-tuple $\langle\rangle$ is represented as $\{\}$. A relation of arity n is a set of n-tuples; if we want to specify that n is the arity of a given relation R, we write the relation as R/n. Note that a relation of arity zero is either empty or it includes only the zero-tuple $\{\}$. A relational schema $\mathcal{R}$ includes a set of relation symbols with their arities and a set of constant symbols $\mathcal{C}$. A database instance $\mathcal{I}^{\mathbf{N}}$ associates to each relation symbol R of arity n from the relational schema $\mathcal{R}$ a set of n-tuples $\mathcal{I}^{\mathbf{N}}(R)$, and to each constant symbol in $\mathcal{C}$ a domain value in Δ. Usually, in relational databases all constant symbols are among the domain values and are associated in the database instance to themselves – this is called the Standard Name Assumption. As an example of a database instance $\mathcal{I}^{\mathbf{N}}$ consider Figure 2(a).

In our work we consider two alternative representations of a database instance $\mathcal{I}^{\mathbf{N}}$, where null values do not appear explicitly: $\mathcal{I}^{\varepsilon}$ and $\mathcal{I}^{\wp}$. We will show that they are isomorphic to $\mathcal{I}^{\mathbf{N}}$. The representations share the same constant symbols, domain values, and mappings from constant symbols to domain values. The differences are in the way null values are encoded within a tuple.

Compared with a database instance $\mathcal{I}^{\mathbf{N}}$, a corresponding database instance $\mathcal{I}^{\varepsilon}$ differs only in the way it represents n-tuples: an n-tuple is a *partial* function

$$R/2: \{\{1 \mapsto a, 2 \mapsto b\}, \{1 \mapsto b, 2 \mapsto \mathbf{N}\}\}$$

(a) instance $\mathcal{I}^{\mathbf{N}}$

$$R/2: \{\{1 \mapsto a, 2 \mapsto b\}, \{1 \mapsto b\}\}$$

(b) instance $\mathcal{I}^{\varepsilon}$

$$\begin{aligned} \widetilde{R}_{\{1,2\}}/2 &: \{\{1 \mapsto a, 2 \mapsto b\}\} \\ \widetilde{R}_{\{1\}}/1 &: \{\{1 \mapsto b\}\} \\ \widetilde{R}_{\{2\}}/1 &: \{\} \\ \widetilde{R}_{\{\}}/0 &: \{\} \end{aligned}$$

(c) instance $\mathcal{I}^{\wp}$

Figure 2. The three representations of the database instance of Figure 1

from integers from 1 up to n into the set of domain values - the function is undefined exactly for those positional arguments which are otherwise defined and mapped to a null value in $\mathcal{I}^{\mathbf{N}}$. As an example of a database instance $\mathcal{I}^{\varepsilon}$ consider Figure 2(b).

Compared with a database instance $\mathcal{I}^{\mathbf{N}}$, a corresponding database instance $\mathcal{I}^{\wp}$ differs in the way relation symbols are interpreted: a relation symbol of arity n is associated to a set of null-free tuples of dishomogeneous arities up to n. Given a database instance $\mathcal{I}^{\mathbf{N}}$ defined over a relational schema $\mathcal{R}$, the corresponding database instance $\mathcal{I}^{\wp}$ is defined over the *decomposed* relational schema $\widetilde{\mathcal{R}}$: for each relation symbol $R \in \mathcal{R}$ of arity n and for each (possibly empty) subset of its positional arguments $A \subseteq [1 \cdots n]$, the decomposed relational schema $\widetilde{\mathcal{R}}$ includes a predicate $\widetilde{R}_A$ with arity $|A|$. The correspondence between $\mathcal{I}^{\mathbf{N}}$ and $\mathcal{I}^{\wp}$ is based on the fact that each $|A|$-tuple in the relation $\mathcal{I}^{\wp}(\widetilde{R}_A)$ corresponds exactly to the n-tuple in $\mathcal{I}^{\mathbf{N}}(R)$ having non-null values only in the positional arguments in A, with the same values and in the same relative positional order. This corresponds to the notion of lossless *horizontal decomposition* in databases [2]. As an example of a database instance $\mathcal{I}^{\wp}$ consider Figure 2(c).

In absence of null values, the $\mathcal{I}^{\mathbf{N}}$ and $\mathcal{I}^{\varepsilon}$ representations of a database instance coincide, and they coincide also with the $\mathcal{I}^{\wp}$ representation if we equate each n-ary R relation symbol in $\mathcal{I}^{\mathbf{N}}$ and $\mathcal{I}^{\varepsilon}$ with its corresponding $\widetilde{R}_{[1 \cdots n]}$ relation symbol in $\mathcal{I}^{\wp}$ – indeed in absence of null values $\mathcal{I}^{\mathbf{N}}(R) = \mathcal{I}^{\varepsilon}(R) = \mathcal{I}^{\wp}(\widetilde{R}_{[1 \cdots n]})$ for every n-ary relation R, and $\mathcal{I}^{\wp}(\widetilde{R}_A) = \emptyset$ for every relation of arity $|A| < n$. Given the discussed isomorphisms, in the following – whenever the difference in the representation of null values is not ambiguous – we will denote as $\mathcal{I}$ the database instance represented in any of the above three forms $\mathcal{I}^{\mathbf{N}}$, $\mathcal{I}^{\varepsilon}$, or $\mathcal{I}^{\wp}$.

3 Relational Algebra with Null Values

We introduce in this Section the formal semantics of the relational algebra dealing with null values, corresponding (modulo the zero-ary relations) to the first-order fragment of SQL.

Let's first recall the notation of the standard relational algebra $\mathcal{RA}$ (see, e.g., [3] for details). Standard relational algebra expressions over a relational schema $\mathcal{R}$ are built according to the inductive formation rules in the boxed expressions of

Atomic relation - R **-** (where $R \in \mathcal{R}$)

$$R(\mathcal{I}) = \mathcal{I}^{\mathsf{N}}(R).$$

Constant singleton - $\langle v \rangle$ **-** (where $v \in \mathcal{C}$)

$$\langle v \rangle(\mathcal{I}) = \{1 \mapsto v\}.$$

Selection - $\sigma_{i=v}\ e$, $\sigma_{i=j}\ e$ **-** (where $v \in \mathcal{C}$, ℓ is the arity of e, and $i, j \leq \ell$)

$$\begin{aligned}\sigma_{i=v}\ e(\mathcal{I}) &= \{s \text{ is a } \ell\text{-tuple} \mid s \in e(\mathcal{I}) \land s(i) = v\},\\ \sigma_{i=j}\ e(\mathcal{I}) &= \{s \text{ is a } \ell\text{-tuple} \mid s \in e(\mathcal{I}) \land s(i) = s(j)\}.\end{aligned}$$

Projection - $\pi_{i_1,\ldots,i_k} e$ **-** (where ℓ is the arity of e, and $\{i_1, \ldots, i_k\} \subseteq [1 \ldots \ell]$)

$$\pi_{i_1,\ldots,i_k} e\ (\mathcal{I}) = \{s \text{ is a } k\text{-tuple} \mid \text{ exists } s' \in e(\mathcal{I}) \text{ s.t. for all } 1 \leq j \leq k.\ s(j) = s'(i_j)\}.$$

Cartesian product - $e \times e'$ **-** (where n, m are the arities of e, e')

$$\begin{aligned}(e \times e')(\mathcal{I}) = \{s \text{ is a } (n+m)\text{-tuple} \mid\ & \text{exists } t \in e(\mathcal{I}), t' \in e'(\mathcal{I}) \text{ s.t.}\\ & \text{for all } \quad 1 \leq j \leq n. \qquad s(j) = t(j) \land\\ & \text{for all } 1+n \leq j \leq (n+m).\ s(j) = t'(j-n)\}.\end{aligned}$$

Union/Difference - $e \cup e'$, $e - e'$ **-** (where ℓ is the arity of e and e')

$$\begin{aligned}(e \cup e')(\mathcal{I}) &= \{s \text{ is a } \ell\text{-tuple} \mid s \in e(\mathcal{I}) \lor s \in e'(\mathcal{I})\},\\ (e - e')(\mathcal{I}) &= \{s \text{ is a } \ell\text{-tuple} \mid s \in e(\mathcal{I}) \land s \notin e'(\mathcal{I})\}.\end{aligned}$$

Derived operators - where $v \in \mathcal{C}$, $\ell \leq \mathsf{min}(m, n)$, m, n are the arities of e, e', $i, j, i_1, \ldots, i_\ell \leq m$, and $k_1, \ldots, k_\ell \leq n$

$$\begin{aligned}&\sigma_{i<>v}\ e \equiv e - \sigma_{i=v}\ e,\\ &\sigma_{i<>j}\ e \equiv e - \sigma_{i=j}\ e,\\ &e \underset{\substack{i_1=k_1\\ \cdots\\ i_\ell=k_\ell}}{\bowtie} e' \equiv \pi_{([1\cdots m+n]\setminus\{m+k_1,\ldots,m+k_\ell\})}\sigma_{i_1=m+k_1} \ldots \sigma_{i_\ell=m+k_\ell}\ (e \times e')\ .\end{aligned}$$

Figure 3. The standard relational algebra $\mathcal{RA}$

Figure 3. Also in Figure 3 the semantics of an algebra expression e is inductively defined as the transformation of database instances $\mathcal{I}$ – with the Standard Name Assumption – to a set of tuples $e(\mathcal{I})$.

We extend the standard relational algebra to deal with SQL nulls; we refer to it as *Null Relational Algebra* ($\mathcal{RA}^{\mathsf{N}}$). In order to deal with null values in the relational model, Codd [4] included the special null value in the domain and adopted a three-valued logic having the third truth value *unknown* together with *true* and *false*. The comparison expressions in Codd's algebra are evaluated to *unknown* if they involve a null value, while in set operations, tuples otherwise identical but containing null values are considered to be different. SQL uses this three-valued logic for the evaluation of `WHERE` clauses [5]. In order to define

Null singleton - $\langle \mathsf{N} \rangle$ **-**

$$\langle \mathsf{N} \rangle(\mathcal{I}) = \{1 \mapsto \mathsf{N}\}.$$

Selection - $\sigma_{i=j}\ e$ **-** (where ℓ is the arity of e, and $i, j \leq \ell$)

$$\sigma_{i=j}\ e(\mathcal{I}) = \{s \text{ is a } \ell\text{-tuple} \mid\ s \in e(\mathcal{I}) \wedge s(i) = s(j) \wedge s(i) \neq \mathsf{N} \wedge s(j) \neq \mathsf{N}\}.$$

Derived operators - where $v \in \mathcal{C}$, $\ell \leq \min(m, n)$, m, n are the arities of e, e', $i, j, i_1, \ldots, i_\ell \leq m$, and $k_1, \ldots, k_\ell \leq n$

$$\begin{aligned}
\sigma_{i<>j}\ e &\equiv \sigma_{i=i}\ \sigma_{j=j}\ e - \sigma_{i=j}\ e,\\
\sigma_{\mathsf{isNull}(i)}\ e &\equiv e - \sigma_{i=i}\ e,\\
\sigma_{\mathsf{isNotNull}(i)}\ e &\equiv \sigma_{i=i}\ e,\\
\sigma_{i=\mathsf{N}}\ e &\equiv e - e,\\
\sigma_{i<>\mathsf{N}}\ e &\equiv e - e,\\
e \underset{\substack{i_1=k_1\\ \cdots\\ i_\ell=k_\ell}}{⟕}\ e' &\equiv (e \underset{\substack{i_1=k_1\\ \cdots\\ i_\ell=k_\ell}}{\bowtie} e') \cup (e - \pi_{1,\ldots,m}(e \underset{\substack{i_1=k_1\\ \cdots\\ i_\ell=k_\ell}}{\bowtie} e')) \times (\underbrace{\langle \mathsf{N} \rangle \times \cdots \times \langle \mathsf{N} \rangle}_{n-\ell}).
\end{aligned}$$

Figure 4. The null relational algebra $\mathcal{RA}^{\mathsf{N}}$ defined only in the parts different from $\mathcal{RA}$

$\mathcal{RA}^{\mathsf{N}}$ from the standard relational algebra, we adopt the $\mathcal{I}^{\mathsf{N}}$ representation of a database instance where null values are explicitly present as possible elements of tuples, and with the Standard Name Assumption. All the $\mathcal{RA}$ expressions are valid $\mathcal{RA}^{\mathsf{N}}$ expressions, and maintain the same semantics, with the only change in the semantics of the *selection* expressions $\sigma_{\mathbf{i=j}}\ \mathbf{e}$ and $\sigma_{\mathbf{i<>j}}\ \mathbf{e}$ with equality or inequality: in these cases the semantic definitions make sure that the elements to be tested for equality (or inequality) are both different from the null value in order for the equality (or inequality) to succeed. In other words, it is enough to let equality (and inequality) fail whenever null values are involved, since its evaluation would be *unknown*.

The syntax of $\mathcal{RA}^{\mathsf{N}}$ expressions extends the standard $\mathcal{RA}$ syntax only for the additional *null singleton* expression $\langle \mathsf{N} \rangle$ (and for the derived operators involving null values). Figure 4 introduces the syntax and semantics of $\mathcal{RA}^{\mathsf{N}}$ expressed in terms of its difference with $\mathcal{RA}$, including the derived operators which are added or defined differently in $\mathcal{RA}^{\mathsf{N}}$. It is easy to show that the definition of the null relational algebra exactly matches the (informal) definition given to SQL with null values, and it generates the same practical behaviour [5].

Given a tuple t and a $\mathcal{RA}^{\mathsf{N}}$ expression e, we call *models* of the expression e with the answer t all the database instances $\mathcal{I}$ such that $t \in e(\mathcal{I})$. Sometimes an n-ary algebra expression is intended to express a boolean statement over a database instance in the form of a *denial constraint*: this is done by checking whether its evaluation over the database instance returns the empty n-ary relation or not. an n-ary denial constraint e can always be reduced into a zero-ary constraint, whose boolean value is meant to be **true** if its evaluation contains the zero-tuple $\{\}$ and **false** if it is empty, by considering its zero-ary projection $(\pi_\emptyset e)$.

Example 1. Consider the schema $\{R/2, S/2\}$ with the following data and $\mathcal{RA}^{\mathsf{N}}$ constraints:

$R:$

1	2
a	b
b	**N**

$S:$

1	2
a	a
a	**N**
N	a
N	**N**

– UNIQUE constraint for $R.1$: $\sigma_{1=3}\ \sigma_{2\neq 4}\ (R \times R) = \emptyset$;
– NOT-NULL constraint for $R.1$: $\sigma_{\mathsf{isNull}(1)}\ R = \emptyset$;
– UNIQUE constraint for $S.1$: $\sigma_{1=3}\ \sigma_{2\neq 4}\ (S \times S) = \emptyset$,
– FOREIGN KEY constraint from $S.2$ to $R.1$: $\pi_2\sigma_{\mathsf{isNotNull}(2)}\ S - \pi_1\sigma_{\mathsf{isNotNull}(1)}\ R = \emptyset$.

It is easy to see that these constraints are all satisfied: they behave in the same way as the corresponding SQL constraints with null values. Similarly, the query considered in the previous Section ($\sigma_{1=1}\ \sigma_{2=2}\ R$), now behaves as in SQL and it returns correctly $\{\langle a, b\rangle\}$.

4 First-order Logic with Null Values

The *Null Relational Calculus* ($\mathcal{FOL}^{\varepsilon}$) is a first-order logic language with an explicit symbol **N** representing the null value, and where predicates denote tuples over *subsets* of the arguments instead of just their whole set of arguments. It extends classical first-order logic in order to take into account the possibility that some of the arguments of a relation might not exist.

Given a set of predicate symbols each one associated with an arity together with the special equality binary predicate =, and a set $\mathcal{C}$ of constants – together forming the relational schema (or *signature*) $\mathcal{R}$ – and a set of variable symbols, terms of $\mathcal{FOL}^{\varepsilon}$ are variables, constants, and the special *null* symbol **N**, and formulae of $\mathcal{FOL}^{\varepsilon}$ are defined by the following rules:

1. if $t_1, \ldots, t_n$ are terms and R is a predicate symbol in $\mathcal{R}$ (different from the equality) of arity n, $R(t_1, \ldots, t_n)$ is an atomic formula;
2. if t_1, t_2 are terms different from **N**, $=(t_1, t_2)$ is an atomic formula;
3. if φ and φ' are formulae, then $\neg\varphi$, $\varphi \wedge \varphi'$ and $\varphi \vee \varphi'$ are formulae;
4. if φ is a formula and x is a variable, then $\exists x\varphi$ and $\forall x\varphi$ are formulae.

The semantics of $\mathcal{FOL}^{\varepsilon}$ formulae is given in terms of database instances of type $\mathcal{I}^{\varepsilon}$, called *interpretations*. As usual, an interpretation $\mathcal{I}^{\varepsilon}$ includes an interpretation domain Δ and it associates each relation symbol R of arity n in the signature to a set of n-tuples $\mathcal{I}^{\varepsilon}(R)$ – i.e., a set of *partial* functions with range in Δ – and each constant symbol in $\mathcal{C}$ to a domain value in Δ (we do not consider here the Standard Name Assumption). The equality predicate is interpreted as the classical equality over the domain Δ. It is easy to see that if n-tuples were just *total* functions, then an interpretation $\mathcal{I}^{\varepsilon}$ would correspond exactly to a classical first-order interpretation.

The definition of satisfaction and entailment in $\mathcal{FOL}^{\varepsilon}$ is the same as in classical first-order logic with equality over the signature $\mathcal{R}$, with the only difference in the truth value of atomic formulae which use partial functions instead of total functions, because of the possible presence of null values. As usual, an interpretation $\mathcal{I}^{\varepsilon}$ satisfying a formula is called a *model* of the formula.
An interpretation $\mathcal{I}^{\varepsilon}$ and a valuation function for variable symbols α satisfy

an atomic formula – written $\mathcal{I}^\varepsilon, \alpha \models_{\mathcal{FOL}^\varepsilon} R(t_1, \ldots, t_n)$ – iff there is an n-tuple $\tau \in \mathcal{I}^\varepsilon(R)$ such that for each $i \in [1 \cdots n]$: $\tau(i) = \mathcal{I}^\varepsilon(c)$ if t_i is a constant symbol c, $\tau(i) = \alpha(t_i)$ if t_i is a variable symbol, and $\tau(i)$ is undefined if $t_i = \mathsf{N}$. It is easy to see that the satisfiability of a $\mathcal{FOL}^\varepsilon$ formula without any occurrence of the null symbol N doesn't depend on partial tuples; so its models can be characterised by classical first-order semantics: in each model the interpretation of predicates would include only tuples represented as total functions.

Example 2. The models of the $\mathcal{FOL}^\varepsilon$ formula $R(a, b) \wedge R(b, \mathsf{N})$ are the interpretations $\mathcal{I}^\varepsilon$ such that $\mathcal{I}^\varepsilon(R)$ includes the tuples $\{1 \mapsto a, 2 \mapsto b\}$ and $\{1 \mapsto b\}$.

4.1 Characterisation in classical First-order Logic

Given a signature $\mathcal{R}$, let's consider a classical first-order logic language with equality ($\mathcal{FOL}$) over the *decomposed* signature $\widetilde{\mathcal{R}}$, as it has been defined in Section 2; $\mathcal{FOL}$ has a classical semantics with models of type $\mathcal{I}^\wp$ and it does not deal with null values directly. In this Section we show that $\mathcal{FOL}^\varepsilon$ over $\mathcal{R}$ and $\mathcal{FOL}$ over $\widetilde{\mathcal{R}}$ are equally expressive, namely that for every formula in $\mathcal{FOL}^\varepsilon$ over the signature $\mathcal{R}$ there is a corresponding formula in $\mathcal{FOL}$ over the decomposed signature $\widetilde{\mathcal{R}}$, such that the two formulae have isomorphic models, and that for every formula in $\mathcal{FOL}$ over the decomposed signature $\widetilde{\mathcal{R}}$ there is a corresponding formula in $\mathcal{FOL}^\varepsilon$ over the signature $\mathcal{R}$, such that the two formulae have isomorphic models. As we discussed before, the isomorphism between the interpretations $\mathcal{I}^{\mathsf{N}}$ and $\mathcal{I}^\wp$ is based on the fact that each $|A|$-tuple in the relation $\mathcal{I}^\wp(\widetilde{R}_A)$ corresponds exactly to the n-tuple in $\mathcal{I}^{\mathsf{N}}(R)$ having non-null values *only* in the positional arguments specified in A, with the same values and in the same relative positional order.

In order to relate the two logics, we define a bijective translation function $\Omega_f(\cdot)$ (and its inverse $\Omega_f^{-1}(\cdot)$) which maps $\mathcal{FOL}^\varepsilon$ formulae into $\mathcal{FOL}$ formulae (and vice-versa).

Definition 1 (Bijective translation Ω_f).
$\Omega_f(\cdot)$ is a bijective function from $\mathcal{FOL}^\varepsilon$ formulae over the signature $\mathcal{R}$ to $\mathcal{FOL}$ formulae over the signature $\widetilde{\mathcal{R}}$, defined as follows.

- $\Omega_f(R(t_1, \ldots, t_n)) = \widetilde{R}_{\{i_1,\ldots,i_k\}}(t_{i_1}, \ldots, t_{i_k})$,
  where $R \in \mathcal{R}$ an n-ary relation, $\{i_1, \ldots, i_k\} = \{j \in [1 \cdots n] \mid t_j$ is not $\mathsf{N}\}$.
  Obviously: $\Omega_f^{-1}(\widetilde{R}_{\{i_1,\ldots,i_k\}}(t_{i_1}, \ldots, t_{i_k})) = R(t'_1, \ldots, t'_n)$,
  where $\{i_1, \ldots, i_k\} \subseteq [1 \cdots n]$ and $t_{i_j} = t'_j$ for $j = 1, \ldots, k$, and t'_j is N for $j \in [1 \cdots n] \setminus \{i_1, \ldots, i_k\}$.
  In both the direct and inverse cases we assume $i_1, \ldots, i_k$ in ascending order.
- *The translation of equality atoms and non atomic formulae is the identity transformation inductively defined on top of the above translation of atomic formulae.*

Example 3. The $\mathcal{FOL}^\varepsilon$ formula $\exists x.R(a, x) \wedge R(x, \mathsf{N}) \wedge R(\mathsf{N}, \mathsf{N})$ over the signature R is translated as the $\mathcal{FOL}$ formula $\exists x.\widetilde{R}_{\{1,2\}}(a, x) \wedge \widetilde{R}_{\{1\}}(x) \wedge \widetilde{R}_{\{\}}$ over the decomposed signature $\widetilde{\mathcal{R}}$, and vice-versa.

The above bijective translation preserves the models of the formulae, modulo the isomorphism among models presented in Section 2.

Theorem 1. *Let φ be a $\mathcal{FOL}^\varepsilon$ formula over the signature $\mathcal{R}$, and $\widetilde{\varphi}$ a $\mathcal{FOL}$ formula over the signature $\widetilde{\mathcal{R}}$. Then for any (database) instance $\mathcal{I}$:*

$$\mathcal{I}^\varepsilon, \alpha \models_{\mathcal{FOL}^\varepsilon} \varphi \quad \textit{if and only if} \quad \mathcal{I}^\wp, \alpha \models_{\mathcal{FOL}} \Omega_f(\varphi).$$

4.2 Domain Independent fragment of $\mathcal{FOL}^\varepsilon$ with Standard Names

In this Section we introduce the domain independent fragment of $\mathcal{FOL}^\varepsilon$ with the Standard Name Assumption, and we analyse its properties.

Definition 2 (Domain Independence). *A $\mathcal{FOL}^\varepsilon$ closed formula φ is domain independent if for every two interpretations $\mathcal{I} = \langle \Delta^\mathcal{I}, \mathcal{I}(\cdot)\rangle$ and $\mathcal{J} = \langle \Delta^\mathcal{J}, \mathcal{J}(\cdot)\rangle$, which agree on the interpretation of relation symbols and constant symbols – i.e. $\mathcal{I}(\cdot) = \mathcal{J}(\cdot)$ – but disagree on the interpretation domains $\Delta^\mathcal{I}$ and $\Delta^\mathcal{J}$:*

$$\mathcal{I} \models \varphi \quad \textit{if and only if} \quad \mathcal{J} \models \varphi.$$

The domain independent fragment of $\mathcal{FOL}^\varepsilon$ includes only the $\mathcal{FOL}^\varepsilon$ domain independent formulae.

It is easy to see that the domain independent fragment of $\mathcal{FOL}^\varepsilon$ can be characterised with the *safe-range* syntactic fragment of $\mathcal{FOL}^\varepsilon$: intuitively, a formula is safe-range if and only if its variables are bounded by positive predicates or equalities – for the syntactical definition see, e.g., [3]. Due to the strong semantic equivalence expressed in Theorem 1 and to the fact the the bijection $\Omega_f(\cdot)$ preserves the syntactic structure of the formulae, we can reuse the results about safe-range transformations and domain independence holding for classical $\mathcal{FOL}$.

Theorem 2. *Any safe-range $\mathcal{FOL}^\varepsilon$ formula is domain independent, and any domain independent $\mathcal{FOL}^\varepsilon$ formula can be transformed into a logically equivalent safe-range $\mathcal{FOL}^\varepsilon$ formula.*

We observe that an interpretation is a model of a formula in the domain independent fragment of $\mathcal{FOL}^\varepsilon$ with the Standard Name Assumption *if and only if* the interpretation which agrees on the interpretation of relation and constant symbols but with the interpretation domain equal to the set of standard names $\mathcal{C}$ is a model of the formula. Therefore, in the following when dealing with the domain independent fragment of $\mathcal{FOL}^\varepsilon$ with the Standard Name Assumption we can just consider interpretations with the interpretation domain equal to $\mathcal{C}$.

We can weaken the Standard Name Assumption by assuming *Unique Names* instead. An interpretation $\mathcal{I}$ satisfies the Unique Name Assumption if $\mathcal{I}(a) \neq \mathcal{I}(b)$ for any different $a, b \in \mathcal{C}$. An interpretation is a model of a $\mathcal{FOL}^\varepsilon$ formula with the Standard Name Assumption *if and only if* the interpretation obtained by homomorphically transform the standard names with arbitrary domain elements is a model of the formula; this latter interpretation satisfies the Unique

Name Assumption. It is possible therefore to interchange the Standard Name and the Unique Name Assumptions; this is of practical advantage, since the Unique Name Assumption can be encoded in first-order logic and it is natively present in most description logic reasoners.

5 Equivalence of Algebra and Calculus

We show that the $\mathcal{RA}^{\mathsf{N}}$ relational algebra with nulls and the domain independent fragment of $\mathcal{FOL}^{\varepsilon}$ with the Standard Name Assumption are equally expressive. The notion of *equal expressivity* is captured by the following two theorems.

Theorem 3. *Let e be an arbitrary $\mathcal{RA}^{\mathsf{N}}$ expression of arity n, and t an arbitrary n-tuple as a total function with values taken from the set $\mathcal{C} \cup \{\mathsf{N}\}$. There is a function $\Omega(e,t)$ translating e with respect to t into a closed safe-range $\mathcal{FOL}^{\varepsilon}$ formula, such that for any instance $\mathcal{I}$ with the Standard Name Assumption:*

$$t \in e(\mathcal{I}^{\mathsf{N}}) \quad \textit{if and only if} \quad \mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \Omega(e,t).$$

Theorem 4. *Let φ be an arbitrary safe-range $\mathcal{FOL}^{\varepsilon}$ closed formula. There is a $\mathcal{RA}^{\mathsf{N}}$ expression e, such that for any instance $\mathcal{I}$ with the Standard Name Assumption:*

$$\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \varphi \quad \textit{if and only if} \quad e(\mathcal{I}^{\mathsf{N}}) \neq \emptyset.$$

This means that there exists a reduction from the membership problem of a tuple in the answer of a $\mathcal{RA}^{\mathsf{N}}$ expression over a database instance with null values into the satisfiability problem of a closed safe-range $\mathcal{FOL}^{\varepsilon}$ formula over the same database (modulo the isomorphism among database instances presented in Section 2); and there exists a reduction from the satisfiability problem of a closed safe-range $\mathcal{FOL}^{\varepsilon}$ formula over a database instance with null values into the emptiness problem of the answer of a $\mathcal{RA}^{\mathsf{N}}$ expression over the same database (modulo the isomorphism among database instances).

This is the translation mapping $\mathcal{RA}^{\mathsf{N}}$ expressions into safe-range $\mathcal{FOL}^{\varepsilon}$ formulae.

Definition 3 (From $\mathcal{RA}^{\mathsf{N}}$ to safe-range $\mathcal{FOL}^{\varepsilon}$). *Let e be an arbitrary $\mathcal{RA}^{\mathsf{N}}$ expression, and t an arbitrary tuple of the same arity as e, as a total function with values taken from the set $\mathcal{C} \cup \{\mathsf{N}\} \cup \mathcal{V}$, where $\mathcal{V}$ is a countable set of variable symbols. The function $\Omega(e,t)$ translates e with respect to t into a $\mathcal{FOL}^{\varepsilon}$ formula according to the following inductive definition:*

- *for any $R/_{\ell} \in \mathcal{R}$, $\Omega(R,t) \rightsquigarrow R(t(1),\ldots,t(\ell))$*
- $\Omega(\langle v \rangle, t) \rightsquigarrow \begin{cases} =(t(1),v) & \textit{if } t(1) \neq \mathsf{N} \\ \textsf{false} & \textit{otherwise} \end{cases}$
- $\Omega(\langle \mathsf{N} \rangle, t) \rightsquigarrow \begin{cases} \textsf{false} & \textit{if } t(1) \neq \mathsf{N} \\ \textsf{true} & \textit{otherwise} \end{cases}$

– $\mathit{\Omega}(\sigma_{i\,=\,v}\, e, t) \rightsquigarrow \begin{cases} \mathit{\Omega}(e,t) \wedge {=}(t(i), v) & \textit{if } t(i) \neq \mathbf{N} \\ \textbf{false} & \textit{otherwise} \end{cases}$
– $\mathit{\Omega}(\sigma_{i\,=\,j}\, e, t) \rightsquigarrow \begin{cases} \mathit{\Omega}(e,t) \wedge {=}(t(i), t(j)) & \textit{if } t(i), t(j) \neq \mathbf{N} \\ \textbf{false} & \textit{otherwise} \end{cases}$
– $\mathit{\Omega}(\pi_{i_1 \cdots i_k} e, t) \rightsquigarrow \exists x_1 \cdots x_n \bigvee_{H \subseteq \{1 \cdots n\} \setminus \{i_1 \cdots i_k\}} \mathit{\Omega}(e, t_H)$
where x_i are fresh variable symbols and t_H is a sequence of n terms defined as:

$$t_H(i) \doteq \begin{cases} t(i) & \textit{if } i \in \{i_1, \ldots, i_k\} \\ \mathbf{N} & \textit{if } i \in H \\ x_i & \textit{otherwise} \end{cases}$$

– $\mathit{\Omega}(e_1 \times e_2, t) \rightsquigarrow \mathit{\Omega}(e_1, t') \wedge \mathit{\Omega}(e_2, t'')$
where n_1, n_2 are the arity of e_1, e_2 respectively,
where t' is a n_1-ary tuple function s.t. $t'(i) = t(i)$ for $1 \leq i \leq n_1$,
and t'' is a n_2-ary tuple function s.t. $t''(i) = t(n_1 + i)$ for $1 \leq i \leq n_2$
– $\mathit{\Omega}(e_1 \cup e_2, t) \rightsquigarrow \mathit{\Omega}(e_1, t) \vee \mathit{\Omega}(e_2, t)$
– $\mathit{\Omega}(e_1 - e_2, t) \rightsquigarrow \mathit{\Omega}(e_1, t) \wedge \neg \mathit{\Omega}(e_2, t)$

Example 4. Given some database instance, checking whether the tuple $\langle a, b \rangle$ or the tuple $\langle b, \mathbf{N} \rangle$ are in the answer of the $\mathcal{RA}^{\mathbf{N}}$ query $\sigma_{1=1}\ \sigma_{2=2}\ R$ (discussed in Section 1) corresponds to check the satisfiability over the database instance of the $\mathcal{FOL}^{\varepsilon}$ closed safe-range formula $R(a, b)$ in the former case, or of the formula **false** in the latter case. You can observe that, as expected, also when translated in first-order logic, it turns out that the tuple $\langle b, \mathbf{N} \rangle$ is not in the answer of the query $\sigma_{1=1}\ \sigma_{2=2}\ R$ for any database.

Example 5. Let's consider the "UNIQUE constraint for $R.1$" from Example 1 expressed in $\mathcal{RA}^{\mathbf{N}}$ as $\sigma_{1=3}\ \sigma_{2\neq 4}\ (R \times R) = \emptyset$. The $\mathcal{RA}^{\mathbf{N}}$ expression is translated as the closed safe-range $\mathcal{FOL}^{\varepsilon}$ formula: $\forall x, y, z.\ R(x, y) \wedge R(x, z) \rightarrow y = z$, which corresponds to the way to express a *unique* constraint in first-order logic.

Let's see the translation in three steps from safe-range $\mathcal{FOL}^{\varepsilon}$ formulae to $\mathcal{RA}^{\mathbf{N}}$ expressions. First, the $\mathcal{FOL}^{\varepsilon}$ formula over the signature $\mathcal{R}$ is first translated into a safe-range $\mathcal{FOL}$ formula over the signature $\widetilde{\mathcal{R}}$, as explained in Section 4.1. Then, the safe-range $\mathcal{FOL}$ formula over the signature $\widetilde{\mathcal{R}}$ is translated into a $\mathcal{RA}$ expression over the signature $\widetilde{\mathcal{R}}$, using the classical translation of classical relational algebra into safe-range first-order logic [3]. Finally, the $\mathcal{RA}$ expression over the signature $\widetilde{\mathcal{R}}$ is translated into a $\mathcal{RA}^{\mathbf{N}}$ expression over the signature $\mathcal{R}$, where each basic relation $\widetilde{R}_{\{i_1, \ldots, i_k\}}$ with R of arity n is substituted according to the rule:

$$\widetilde{R}_{\{i_1, \ldots, i_k\}} \rightsquigarrow\ \pi_{i_1, \ldots, i_k}(\sigma_{\mathsf{isNotNull}(\{i_1, \ldots, i_k\})}\ (\sigma_{\mathsf{isNull}([1 \cdots n] \setminus \{i_1, \ldots, i_k\})}\ R))$$

where $\sigma_{\mathsf{isNull}(i_1, \ldots, i_k)}\ e$ is a shorthand for $\sigma_{\mathsf{isNull}(i_1)} \ldots \sigma_{\mathsf{isNull}(i_k)}\ e$.

Example 6. The safe-range closed $\mathcal{FOL}^{\varepsilon}$ formula $\exists x.R(x, \mathsf{N})$ is translated as the zero-ary $\mathcal{RA}^{\mathsf{N}}$ statement $\sigma_{\mathsf{isNotNull}(1)}\ \sigma_{\mathsf{isNull}(2)}\ R \neq \emptyset$.

Example 7. Let's consider what would be the classical way to express the "FOREIGN KEY from $S.2$ to $R.1$" constraint from Example 1 in first-order logic:

$$\forall x, y.\ S(x, y) \rightarrow \exists z.\ R(y, z) \quad \equiv \quad \neg\exists y.\ (\exists x.\ S(x, y)) \wedge \neg(\exists z.\ R(y, z)).$$

The formula is translated as the $\mathcal{RA}^{\mathsf{N}}$ statement: $\pi_2\sigma_{\mathsf{isNotNull}(2)}\ S - \pi_1\sigma_{\mathsf{isNotNull}(1)}\ R = \emptyset$, which is the $\mathcal{RA}^{\mathsf{N}}$ statement we considered in Example 1.

6 Semantic Integrity with Null Values in SQL:1999

In this Section we consider the main integrity constraints involving a specific behaviour for null values as defined in the ANSI/ISO/IEC standard SQL:1999, and we show how these can be naturally captured using $\mathcal{FOL}^{\varepsilon}$. We focus on *unique and primary key constraints* and on *foreign key constraints* as defined in [6]. We will see how the actual definitions of the unique, primary key, and foreign key constraints are a bit more involved than we have seen before: this is because more complex cases than the simple examples above may happen involving null values.

Unique and primary key constraints. As specified in [6], a uniqueness constraint $\textbf{UNIQUE}(u_1, \ldots, u_n)$ holds for a table R of arity $m > n$ in a database if and only if there are no two rows r_1, r_2 in R such that the values of all their uniqueness columns u_i match and are not null. More formally, in tuple-relational calculus with explicit null values in the domain (fixing the incomplete definition in [6]):

$$\forall r_1, r_2 \in R.\ \left(r_1 \neq r_2 \wedge \bigwedge_{i=1}^{n} r_1.u_i \neq \mathsf{N} \wedge r_2.u_i \neq \mathsf{N}\right) \rightarrow \left(\bigvee_{i=1}^{n} r_1.u_i \neq r_2.u_i\right).$$

In $\mathcal{FOL}^{\varepsilon}$ this constraint can be written as:

$$\forall \overline{x}.\ \left(\bigwedge_{\{i_1\cdots i_k\}\subseteq\{1\cdots m\}\setminus\{u_1\cdots u_n\}} \forall \overline{y}, \overline{z}.\ (\mathit{\Pi}_{u_1,\ldots,u_n,i_1,\ldots,i_k} R(\overline{x}, \overline{y}) \wedge \mathit{\Pi}_{u_1,\ldots,u_n,i_1,\ldots,i_k} R(\overline{x}, \overline{z})) \rightarrow \overline{y} = \overline{z}\right) \wedge$$

$$\forall \overline{x}.\ \left(\bigwedge_{\{i_1\cdots i_k\}\subseteq\{1\cdots m\}\setminus\{u_1\cdots u_n\}} \exists \overline{y}.\ \mathit{\Pi}_{u_1,\ldots,u_n,i_1,\ldots,i_k} R(\overline{x}, \overline{y})\right) \rightarrow \bot$$

where $\mathit{\Pi}_{i_1,\ldots,i_k} R(t_1, \ldots, t_k)$ is a shorthand for $R(t'_1, \ldots, t'_m)$ where $t'_j = t_j$ for $j = i_1 \ldots i_k$ and $t'_j = \mathsf{N}$ for all other j.

A *primary key* constraint is a combination of a *uniqueness* constraint and one or more *not null* constraints. A constraint $\textbf{PRIMARY KEY}(u_1, \ldots, u_n)$ holds for a table R if and only if the following holds, in tuple-relational calculus with explicit null values in the domain (fixing the incomplete definition in [6]):

$$\forall r \in R.\ \left(\bigwedge_{i=1}^{n} r.u_i \neq \mathsf{N}\right) \wedge \forall r_1, r_2 \in R.\ r_1 \neq r_2 \rightarrow \left(\bigvee_{i=1}^{n} r_1.u_i \neq r_2.u_i\right).$$

Note that no column shall occur more than once within the same unique/primary key definition; furthermore, each table can have at most one primary key.
In $\mathcal{FOL}^{\varepsilon}$ this constraint can be written as the conjunction of a uniqueness constraint as above and a not null constraint as follows for each key attribute u_i:

$$\neg\left(\bigvee_{\{i_1\cdots i_k\}\subseteq\{1\cdots m\}\setminus\{u_i\}} \exists\overline{y}.\ \Pi_{i_1,\ldots,i_k} R(\overline{y})\right).$$

Foreign key constraints. Foreign key constraints (or referential constraints) express dependencies among (portions of) rows in tables. Given a *referenced* (or *parent*) table and a *referencing* (or *child*) table, a subset $f_i, \ldots, f_k$ of the columns of the referencing table builds the foreign key and refers to the unique/primary key columns $u_j, \ldots, u_l$ of the referenced table. As specified in [6], the *simple match* is the default foreign key constraint implemented by all DBMS vendors. For each row r of the referencing table R (child table), either at least one of the values of the referencing columns $f_1, \ldots, f_n$ is a null value or the value of each referencing columns $f_i, 1 \leq i \leq n$, is equal to the value of the corresponding referenced column u_i for some row s of the referenced table S. More formally, in tuple-relational calculus with null values in the domain [6]:

$$\forall r \in R.\ \left(\bigwedge_{i=1}^{n} r.f_i \neq \mathsf{N}\right) \rightarrow \exists s \in S.\ \left(\bigwedge_{i=1}^{n} r.f_i = s.u_i\right).$$

In $\mathcal{FOL}^{\varepsilon}$ this constraint can be written as:

$$\forall\overline{x}.\ \left(\bigvee_{\{i_1\cdots i_k\}\subseteq\{1\cdots m\}\setminus\{u_1\cdots u_n\}} \exists\overline{y}.\ \Pi_{u_1,\ldots,u_n,i_1,\ldots,i_k} S(\overline{x},\overline{y})\right) \rightarrow$$
$$\left(\bigvee_{\{i_1\cdots i_k\}\subseteq\{1\cdots m\}\setminus\{u_1\cdots u_n\}} \exists\overline{y}.\ \Pi_{u_1,\ldots,u_n,i_1,\ldots,i_k} R(\overline{x},\overline{y})\right).$$

We observe that, if the database does not contain null values, the SQL:1999 definitions of unique, not null, and foreign key constraints (with simple match) involving null values reduce to the well known classical $\mathcal{FOL}$ definitions of these constraints without null values.

7 Conclusions

Since their inception, SQL null values have been at the centre of long discussions about their real meaning and their formal semantics (see, e.g., [7]). The vast majority of logic based approaches consider nulls as values with an unknown interpretation (i.e., a value exists but it is not known) and they model them as existential variables (e.g. naive tables [1] and the works inspired by [8]). In spite of the fact that these works have their merits and provide a well founded characterisation of incomplete information in databases, they diverge from SQL standard. We have shown that null values – when defined as in SQL with the three-valued logic – should be interpreted as *nonexistent* values, and that such null values do not introduce any incompleteness in the data. In this paper we extend Codd's theorem stating the equivalence of the relational algebra with the relational calculus, to the case in which SQL null values are added to the language.

References

1. Imieliński, T., Lipski, Jr., W.: Incomplete information in relational databases. J. ACM **31**(4) (September 1984) 761–791
2. Date, C.J., Darwen, H.: Database Explorations: Essays on the Third Manifesto and Related Topics. Trafford Publishing (2010)
3. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
4. Codd, E.F.: Extending the database relational model to capture more meaning. ACM Trans. Database Syst. **4**(4) (1979) 397–434
5. Date, C.: An Introduction to Database Systems. 8 edn. Addison-Wesley (2003)
6. Türker, C., Gertz, M.: Semantic integrity support in SQL:1999 and commercial (object-)relational database management systems. The VLDB Journal **10**(4) (2001) 241–269
7. Grant, J.: Null values in SQL. SIGMOD Rec. **37**(3) (September 2008) 23–25
8. Zaniolo, C.: Database relations with null values. Journal of Computer and System Sciences **28**(1) (1984) 142–166