On the Logic of 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.
In this paper we introduce an extension of first-order logic accounting for predicates with missing arguments. We show that the domain independent fragment of this logic is equivalent to Codd's relational algebra with SQL nulls. Moreover, we prove a faithful encoding of the logic into standard first-order logic, so that we can employ classical deduction machinery.

1 Relational Databases and SQL Null Values

Consider a database instance with null values over the relational schema $\{R/2\}$, and a 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 a SQL null value is never equal (or not equal) to anything, including itself. How can we formally capture this behaviour?

In this paper 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 $\mathcal{RA}$. Note that the

standard relational algebra $\mathcal{RA}$ is more expressive than the null-free first-order fragment of SQL, since it can express relations of arity zero [1].

To the best of our knowledge, there has been no attempt so far to formalise in logic a relational algebra with proper 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 above 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 [2]). 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.

This paper is organised as follows. We first introduce three equivalent different representations of databases with null values: they will be used in different contexts to prove the equivalence of the different semantics given to SQL null values. Then, we introduce the relational algebra with null values $\mathcal{RA}^{\mathbf{N}}$, as an obvious extension to the standard relational algebra $\mathcal{RA}$ and compatible with the documents defining SQL. We introduce an extension to standard first-order logic $\mathcal{FOL}$, called $\mathcal{FOL}^{\varepsilon}$, which takes into account the possibility that some of the arguments of a relation might not exist. We then show that $\mathcal{FOL}^{\varepsilon}$ is equally expressive in a strong sense to the standard first-order logic $\mathcal{FOL}$, which allows us to devise simple implementations of $\mathcal{FOL}^{\varepsilon}$ by relying to standard first-order provers. At the end, we prove our main result: $\mathcal{RA}^{\mathbf{N}}$ is equally expressive to the domain independent fragment of $\mathcal{FOL}^{\varepsilon}$ with the standard name assumption, a result which parallels in an elegant way the classical equivalence result by Codd for the standard relational algebra $\mathcal{RA}$ and the domain independent fragment of first-order logic $\mathcal{FOL}$ with the standard name assumption.

2 Database Instances with Null Values

We introduce here the notions of tuple, of relation, and of 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

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

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

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

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

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

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

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

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 given 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 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 $\mathcal{I}^{\mathbf{N}}$.
That is, if $R \in \mathcal{R}$ is an n-ary relation, then

$$\mathcal{I}^{\varepsilon}(R) = \{t \in ([1 \cdots n] \mapsto \mathcal{C}) \mid \exists t' \in \mathcal{I}^{\mathbf{N}}(R).\, t'(i) \neq \mathbf{N} \rightarrow t(i) = t'(i) \wedge t'(i) = \mathbf{N} \rightarrow t(i)\ \textit{undefined}\}$$

Obviously, also the inverse correspondence exists:

$$\mathcal{I}^{\mathbf{N}}(R) = \{t' \in ([1 \cdots n] \mapsto \mathcal{C} \cup \{\mathbf{N}\}) \mid \exists t \in \mathcal{I}^{\varepsilon}(R).\, t(i)\ \textit{defined} \rightarrow t'(i) = t(i) \wedge t(i)\ \textit{undefined} \rightarrow t'(i) = \mathbf{N}\}$$

As an example of a database instance $\mathcal{I}^{\varepsilon}$ consider Figure 2(b). From the figure is clear that this representation takes a different approach w.r.t. the standard relational model: i.e., relations may include tuples of dishomogeneous arity. We will introduce the third kind of representation in order to demonstrate that this

approach can be embedded into a standard relational model by considering an extended vocabulary.

Compared with a database instance $\mathcal{I}^{\varepsilon}$, 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 novel relation symbols having the arity less or equal to n. The main idea is that a tuple with null arguments ends up in the corresponding relation which excludes those arguments (see Figure 2(c)). The concept is analogous to the notion of lossless *horizontal decomposition* as described, e.g., in [1].

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. That is, if $R \in \mathcal{R}$ is an n-ary relation, then

$$\begin{aligned}\mathcal{I}^{\wp}(\widetilde{R}_{\{i_1,\ldots,i_k\}}) = \{t \in ([1 \cdots k] \mapsto \mathcal{C}) \mid \exists\, t' \in \mathcal{I}^{\mathbf{N}}(R).\ & \\ \forall j \in ([1 \cdots n] \setminus \{i_1,\ldots,i_k\}).t'(j) = \mathbf{N} \wedge & \\ \forall j \in \{i_1,\ldots,i_k\}.t'(j) \neq \mathbf{N} \wedge & \\ \forall j \in \{1,\ldots,k\}.t(j) = t'(i_j)\}&\end{aligned}$$

Obviously, also the inverse correspondence exists:

$$\begin{aligned}\mathcal{I}^{\mathbf{N}}(R) = \bigcup_{\{i_1,\ldots,i_k\} \subseteq [1 \cdots n]} \{t' \in\ & ([1 \cdots n] \mapsto \mathcal{C} \cup \{\mathbf{N}\}) \mid \\ & \exists\, t \in \mathcal{I}^{\wp}(\widetilde{R}_{\{i_1,\ldots,i_k\}}). \\ & \quad \forall j \in \{1,\ldots,k\}.t'(i_j) = t(j) \wedge \\ & \quad \forall j \in ([1 \cdots n] \setminus \{i_1,\ldots,i_k\}).t'(j) = \mathbf{N}\}\end{aligned}$$

Abusing the notation for the sake of simplicity, we assume that if $k = 0$ then $\{1,\ldots,k\} = \{i_1,\ldots,i_k\} = \emptyset$.

In absence of null values, clearly the $\mathcal{I}^{\mathbf{N}}$ and $\mathcal{I}^{\varepsilon}$ representations of a database instance coincide. The third representation cannot be directly compared because of the different signature, but it can be noted that, in absence of null values, for every n-ary relation R in $\mathcal{R}$, $\mathcal{I}^{\wp}(\widetilde{R}_A) = \emptyset$ if $|A| < n$. Therefore for each n-ary relation in $\mathcal{R}$, $\mathcal{I}^{\mathbf{N}}(R) = \mathcal{I}^{\varepsilon}(R) = \mathcal{I}^{\wp}(\widetilde{R}_{[1 \cdots n]})$.

Given the discussed isomorphisms, in the following – whenever the difference in the representation of null values does not generate ambiguity – 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 $\mathcal{I}^{\wp}$.

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

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

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 $i, j \leq \ell$)

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

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')

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

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

$$(e \cup e')(\mathcal{I}) = \{s \text{ is a } \ell\text{-tuple} \mid s \in e(\mathcal{I}) \vee s \in e'(\mathcal{I})\},$$
$$(e - e')(\mathcal{I}) = \{s \text{ is a } \ell\text{-tuple} \mid s \in e(\mathcal{I}) \wedge s \notin e'(\mathcal{I})\}.$$

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 $k_1, \ldots, k_\ell \leq n$)

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

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

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 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 $e(\mathcal{I})$.

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

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

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

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

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 $k_1, \ldots, k_\ell \leq n$)

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

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

We now extend the standard relational algebra to deal with SQL nulls; we refer to it as *Null Relational Algebra* ($\mathcal{RA}^{\mathbf{N}}$).

Null values have been introduced in the relational model since its inception. In order to deal with null values, 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. This view is exactly what we find in SQL where three-valued logic is used for the evaluation of `WHERE` clauses but independent occurrences of the null value may be identified in set operations – see, e.g., [5].

In order to define $\mathcal{RA}^{\mathbf{N}}$ from the standard relational algebra, we adopt the $\mathcal{I}^{\mathbf{N}}$ representation of a database instance where null values are explicitly present as possible elements of tuples, and with the Standard Name Assumption. Then, it is enough to let equality (and inequality) fail whenever null values are involved, since its evaluation would be *unknown*.

All the $\mathcal{RA}$ expressions are valid $\mathcal{RA}^{\mathbf{N}}$ 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 among elements in a tuple: 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. The syntax of $\mathcal{RA}^{\mathbf{N}}$ expressions extends the standard $\mathcal{RA}$ syntax only for the additional *null singleton* expression $\langle \mathbf{N} \rangle$ (and for the derived operators involving null values). Figure 4 introduces the syntax and semantics of $\mathcal{RA}^{\mathbf{N}}$ expressed in terms of its difference with $\mathcal{RA}$, including the derived operators which are added or defined differently in $\mathcal{RA}^{\mathbf{N}}$. 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 relational schema $\{R/2, S/2\}$ with the following data and the following constraints expressed in $\mathcal{RA}^{\mathsf{N}}$:

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.

Note that the syntax of $\mathcal{FOL}^{\varepsilon}$ corresponds to a classical first-order logic with equality, with the only addition of the special *null* symbol **N** which may appear in atomic formulae (but not in equalities). As usual, a formula is ground if it does not contain any variable symbol.

The semantics of $\mathcal{FOL}^{\varepsilon}$ formulae is given in terms of database instances of type $\mathcal{I}^{\varepsilon}$, also 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 exactly the same as in classical first-order logic with equality over the signature $\mathcal{R}$, with the only difference lying on the truth value of atomic formulae which involve partial functions instead of total functions because of the possible presence of null values in atomic formulae. 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 $t_i = \mathbf{N}$.
It is easy to see that the satisfiability of a $\mathcal{FOL}^{\varepsilon}$ formula without any occurrence of the null symbol $\mathbf{N}$ 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 $R(a, b) \wedge R(b, \mathbf{N})$ are all the interpretations $\mathcal{I}^{\varepsilon}$ such that $\mathcal{I}^{\varepsilon}(R)$ includes at least 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 $\mathcal{I}^{\mathbf{N}}$ 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 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, $\{i_1,\ldots,i_k\} = \{j \in [1\cdots n] \mid t_i$ is not $\mathbf{N}\}$.
  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 $\mathbf{N}$ 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.*

It is easy to see that the size of a translated formula (via either the $\Omega_f(\cdot)$ or the $\Omega_f^{-1}(\cdot)$ translation) is linearly bounded by the size of the input formula, assuming the signatures $\mathcal{R}$ and $\widetilde{\mathcal{R}}$ fixed; the signature $\widetilde{\mathcal{R}}$ can be exponentially larger than the signature $\mathcal{R}$, the exponent being the maximum arity of the relations in $\mathcal{R}$.

Example 3. The $\mathcal{FOL}^{\varepsilon}$ formula $\exists x.R(a,x) \wedge R(x,\mathbf{N}) \wedge R(\mathbf{N},\mathbf{N})$ 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.

Let's show now that 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 $\mathcal{I}$:*

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

Since $\Omega_f(\cdot)$ is a bijection, we can equivalently write:

$$\mathcal{I}^{\varepsilon}, \alpha \models_{\mathcal{FOL}^{\varepsilon}} \Omega_f^{-1}(\widetilde{\varphi}) \quad \text{if and only if} \quad \mathcal{I}^{\wp}, \alpha \models_{\mathcal{FOL}} \widetilde{\varphi}.$$

Proof. The statements of the theorem are proved by induction on the syntax of the formulae φ and $\widetilde{\varphi}$. Since syntax and semantics of non-atomic formulae is the same for both $\mathcal{FOL}^{\varepsilon}$ and $\mathcal{FOL}$, we show that the statement hold for the atomic cases. The case of equality can be easily proved by noticing that $\mathbf{N}$ is not allowed to appear among its arguments.
We focus on ground atomic formulae, since the valuation function for variable symbols α is the same in the pre- and the post-conditions.

1. Let $R(t_1,\ldots,t_n)$ be a $\mathcal{FOL}^{\varepsilon}$ ground atomic formula where $A = \{i_1,\ldots,i_k\} \subseteq [1\cdots n]$ is the set of its positional arguments for which t_i is not $\mathbf{N}$. Then $\Omega_f(R(t_1,\ldots,t_n)) = \widetilde{R}(t_{i_1},\ldots,t_{i_k})$.
   By the assumption $\mathcal{I}^{\varepsilon}, \alpha \models_{\mathcal{FOL}^{\varepsilon}} R(t_1,\ldots,t_n)$; therefore there is a tuple $\tau \in \mathcal{I}^{\varepsilon}(R)$ such that $\tau(j) = \mathcal{I}^{\varepsilon}(t_j)$ for $j \in A$, and τ is undefined for $j \notin A$. By definition of $\mathcal{I}^{\wp}$, $(\tau(i_1),\ldots,\tau(i_k)) \in \mathcal{I}^{\wp}(\widetilde{R})$ therefore $\mathcal{I}^{\wp}, \alpha \models \widetilde{R}_A(t_{i_1},\ldots,t_{i_k})$.

2. Let $\widetilde{R}_{\{i_1,\ldots,i_k\}}(t_1,\ldots,t_k)$ be a $\mathcal{FOL}^{\varepsilon}$ ground atomic formula with $\{i_1,\ldots,i_k\} \subseteq [1\cdots n]$; then $\Omega_f^{-1}(\widetilde{R}_{\{i_1,\ldots,i_k\}}(t_1,\ldots,t_k)) = R(t'_1,\ldots,t'_n)$, where $\{i_1,\ldots,i_k\} \subseteq [1\cdots n]$ and $t'_{i_j} = t_j$ for $j = 1,\ldots,k$ and $t'_j = \mathbf{N}$ for $j \in [1\cdots n] \setminus \{i_1,\ldots,i_k\}$. By assumption $\mathcal{I}^{\wp}, \alpha \models \widetilde{R}_{\{i_1,\ldots,i_k\}}(t_1,\ldots,t_k)$, therefore $(\mathcal{I}^{\wp}(t_1),\ldots,\mathcal{I}^{\wp}(t_k)) \in \mathcal{I}^{\wp}(\widetilde{R}_{\{i_1,\ldots,i_k\}})$. By definition of $\mathcal{I}^{\varepsilon}$ there is a tuple $\tau \in \mathcal{I}^{\varepsilon}(R)$ s.t. $\tau(i_j) = \mathcal{I}^{\wp}(t_j)$ for $j \in \{i_1,\ldots,i_k\}$ and undefined otherwise. Therefore $\mathcal{I}^{\varepsilon}, \alpha \models_{\mathcal{FOL}^{\varepsilon}} R(t'_1,\ldots,t'_n)$. □

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 domain independent formulae of $\mathcal{FOL}^{\varepsilon}$.

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 exact syntactical definition see, e.g., [3]. Due to the strong semantic equivalence expressed in Theorem 1 and due 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 usually holding for classical $\mathcal{FOL}$. To check whether a formula is safe-range, the formula is transformed into a logically equivalent *safe-range normal form* and its *range restriction* is computed according to a set of syntax based rules; the range restriction of a formula is a subset of its free variables, and if it coincides with the free variables then the formula is said to be safe-range [3]. In addition to that, the following theorem on domain independence holds.

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 focus now on the domain independent fragment of $\mathcal{FOL}^{\varepsilon}$ *with* the Standard Name Assumption. 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 now show that we can weaken the Standard Name Assumption by just assuming *Unique Names*, without changing the certain answers. An interpretation $\mathcal{I}$ satisfies the Unique Name Assumption if $\mathcal{I}(a) \neq \mathcal{I}(b)$ for any different $a, b \in \mathcal{C}$. We observe that 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 clearly satisfies the Unique Name Assumption. This observation allows us to freely interchange the Standard Name and the Unique Name Assumptions; this is of practical advantage, since we can easily encode the Unique Name Assumption in a classical first-order logic reasoner, and most description logics reasoners do have a native Unique Name Assumption.

5 Equivalence of Algebra and Calculus

In this Section we prove the strong relation between $\mathcal{RA}^{\mathbf{N}}$ and $\mathcal{FOL}^{\varepsilon}$.

Theorem 3. *The $\mathcal{RA}^{\mathbf{N}}$ 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 propositions.

Proposition 1. *Let e be an arbitrary $\mathcal{RA}^{\mathbf{N}}$ expression of arity n, and t an arbitrary n-tuple as a total function with values taken from the set $\mathcal{C} \cup \{\mathbf{N}\}$. 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:*

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

Proposition 2. *Let φ be an arbitrary safe-range $\mathcal{FOL}^{\varepsilon}$ closed formula. There is a $\mathcal{RA}^{\mathbf{N}}$ expression e, such that for any instance $\mathcal{I}$ with the Standard Name Assumption:*

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

This means that there exists a reduction from the membership problem of a tuple in the answer of a $\mathcal{RA}^{\mathbf{N}}$ 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 $\mathcal{RA}^{\mathbf{N}}$ expression over the same database (modulo the isomorphism among database instances).

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 $\sigma_{1=1}\ \sigma_{2=2}\ R$. A constructive way to get the $\mathcal{FOL}^{\varepsilon}$ formulae is discussed in Section 5.1.

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 classical way to express a *unique* constraint in first-order logic. A constructive way to get the $\mathcal{FOL}^{\varepsilon}$ formula is discussed in Section 5.1.

Example 6. The safe-range closed $\mathcal{FOL}^{\varepsilon}$ formula $\exists x.R(x, \mathbf{N})$ is translated as the zero-ary $\mathcal{RA}^{\mathbf{N}}$ statement $\sigma_{\mathsf{isNotNull}(1)}\ \sigma_{\mathsf{isNull}(2)}\ R \neq \emptyset$. A constructive way to get the $\mathcal{RA}^{\mathbf{N}}$ statement is discussed in Section 5.2.

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 $\mathcal{RA}^{\mathbf{N}}$ statement: $\pi_2\sigma_{\mathsf{isNotNull}(2)}\ S - \pi_1\sigma_{\mathsf{isNotNull}(1)}\ R = \emptyset$, which is the $\mathcal{RA}^{\mathbf{N}}$ statement we considered in Example 1. A constructive way to get the $\mathcal{RA}^{\mathbf{N}}$ statement is discussed in Section 5.2.

5.1 From Algebra to Calculus

To show that the safe-range fragment of $\mathcal{FOL}^{\varepsilon}$ with the Standard Name Assumption captures the expressivity of $\mathcal{RA}^{\mathbf{N}}$ queries, we first define explicitly a translation function which maps $\mathcal{RA}^{\mathbf{N}}$ expressions into safe-range $\mathcal{FOL}^{\varepsilon}$ formulae.

Definition 3 (From $\mathcal{RA}^{\mathbf{N}}$ to safe-range $\mathcal{FOL}^{\varepsilon}$). *Let e be an arbitrary $\mathcal{RA}^{\mathbf{N}}$ expression, and t an arbitrary tuple of the same arity as e, as a total function with values taken from the set $\mathcal{C} \cup \{\mathbf{N}\} \cup \mathcal{V}$, 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 $R/_{\ell} \in \mathcal{R}$, $\Omega(T, t) \rightsquigarrow R(t(1), \ldots, t(\ell))$*
- $\Omega(\langle v \rangle, t) \rightsquigarrow \begin{cases} =(t(1), v) & \text{if } t(1) \neq \mathbf{N} \\ \mathbf{false} & \text{otherwise} \end{cases}$
- $\Omega(\langle \mathbf{N} \rangle, t) \rightsquigarrow \begin{cases} \mathbf{false} & \text{if } t(1) \neq \mathbf{N} \\ \mathbf{true} & \text{otherwise} \end{cases}$
- $\Omega(\sigma_{i\,=\,v}\ e, t) \rightsquigarrow \begin{cases} \Omega(e, t) \wedge =(t(i), v) & \text{if } t(i) \neq \mathbf{N} \\ \mathbf{false} & \text{otherwise} \end{cases}$
- $\Omega(\sigma_{i\,=\,j}\ e, t) \rightsquigarrow \begin{cases} \Omega(e, t) \wedge =(t(i), t(j)) & \text{if } t(i), t(j) \neq \mathbf{N} \\ \mathbf{false} & \text{otherwise} \end{cases}$

– $\Omega(\pi_{i_1 \cdots i_k} e, t) \rightsquigarrow \exists x_1 \cdots x_n \bigvee_{H \subseteq \{1 \cdots n\} \setminus \{i_1 \cdots i_k\}} \Omega(e, t_H)$
where x_i are fresh variable symbols and t_H is a sequence of n terms defined as:

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

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

We can now proceed with the proof of Proposition 1.

Proof (Proposition 1). We prove the proposition by induction on the expression e that $t \in e(\mathcal{I}^{\mathbf{N}})$ iff $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \Omega(e, t)$. For the sake of brevity we demonstrate the arguments for two of the significant cases.

Let e be $R \in \mathcal{R}$ of arity n and t a tuple s.t. $t \in R(\mathcal{I}^{\mathbf{N}})$ where $\{i_1, \ldots, i_k\}$ are the indexes for which $t(i) = \mathbf{N}$. By Definition 3, $\Omega(R, t) \rightsquigarrow R(t)$. Since $t \in R(\mathcal{I}^{\mathbf{N}})$, then $t \in \mathcal{I}^{\mathbf{N}}(R)$, so there is a partial function $t' \in \mathcal{I}^{\varepsilon}(R)$ s.t. $t'(i)$ is undefined for $i \in \{i_1, \ldots, i_k\}$ and $t'(i) = t(i)$ otherwise. Therefore $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} R(t)$. On the other hand, if $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} R(t)$ then there is a partial function $t' \in \mathcal{I}^{\varepsilon}(R)$ s.t. $t'(i)$ is undefined for $i \in \{i_1, \ldots, i_k\}$ and $t'(i) = \mathcal{I}^{\varepsilon}(t(i))$ otherwise. Since $\mathcal{I}$ is an interpretation with the Standard Name Assumption, $\mathcal{I}^{\varepsilon}(t(i)) = t(i)$; therefore $t \in \mathcal{I}^{\mathbf{N}}(R)$ and $t \in R(\mathcal{I}^{\mathbf{N}})$.

Let $t \in (\pi_{i_1 \cdots i_k} e)(\mathcal{I}^{\mathbf{N}})$, then there is t' of arity n s.t. $t' \in e(\mathcal{I}^{\mathbf{N}})$ and $t(j) = t'(i_j)$ for $i = 1 \ldots k$. By the recursive hypothesis, $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \Omega(e, t')$. Let $H' \subseteq [1 \cdots n] \setminus \{i_1, \cdots, i_k\}$ be the sets of indexes for which $t'(\cdot)$ is $\mathbf{N}$; then $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \exists x_1 \cdots x_n \Omega(e, t_{H'})$, so $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \exists x_1 \cdots x_n \bigvee_{H \subseteq \{1 \cdots n\} \setminus \{i_1 \cdots i_k\}} \Omega(e, t_H)$ because H' is one of these disjuncts. For the other direction, let us assume that $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \Omega(\pi_{i_1 \cdots i_k} e, t)$, then $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \exists x_1 \cdots x_n \bigvee_{H \subseteq [1 \cdots n] \setminus \{i_1 \cdots i_k\}} \Omega(e, t_H)$, so there is $H' \subseteq [1 \cdots n] \setminus \{i_1 \cdots i_k\}$ and assignment α for variables in $t_{H'}$ s.t. $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \Omega(e, \alpha(t_{H'}))$. By the inductive hypothesis there is a tuple t' corresponding to $\alpha(t_{H'})$ s.t. $t' \in e(\mathcal{I}^{\mathbf{N}})$; by construction of $t_{H'}$, $t(i) = \alpha(t_{H'}(i))$ for $i \in \{i_1, \ldots, i_k\}$ therefore $t \in \pi_{i_1 \cdots i_k} e\ (\mathcal{I}^{\mathbf{N}})$. □

5.2 From Calculus to Algebra

To show that $\mathcal{RA}^{\mathbf{N}}$ queries capture the expressivity of the safe-range fragment of $\mathcal{FOL}^{\varepsilon}$ with the Standard Name Assumption, we first establish two intermediate results supporting that $\mathcal{FOL}^{\varepsilon}$ queries can be expressed in (standard) relational algebra.

Lemma 1. *Let $\widetilde{e}$ be an expression in the standard relational algebra $\mathcal{RA}$ over the decomposed relational schema $\widetilde{\mathcal{R}}$. There is a $\mathcal{RA}^{\mathbf{N}}$ expression e over the relational schema $\mathcal{R}$, such that for any instance $\mathcal{I}$:*

$$\widetilde{e}(\mathcal{I}^{\wp}) = e(\mathcal{I}^{\mathbf{N}})$$

Proof. The $\mathcal{RA}^{\mathbf{N}}$ expression e is the same expression $\widetilde{e}$ where each basic relation $\widetilde{R}_{\{i_1,\ldots,i_k\}}$ with R of arity n is substituted by the expression

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

where $\sigma_{\mathsf{isNull}(i_1,\ldots,i_k)}\ e$ is a shorthand for $\sigma_{\mathsf{isNull}(i_1)}\ \ldots\ \sigma_{\mathsf{isNull}(i_k)}\ e$.
The proof is by induction on the expression e where the basic cases are the unary singleton - which is the same as $\mathcal{RA}^{\mathbf{N}}$ since there are no nulls - and any basic relation $\widetilde{R}_{\{i_1,\ldots,i_k\}}$ where $R \in \mathcal{R}$ an n-ary relation. Compound expressions satisfy the equality because of the inductive assumption and the fact that the algebraic operators (in absence of null values) in $\mathcal{RA}^{\mathbf{N}}$ and $\mathcal{RA}$ have the same definition.
Therefore we just need to show that

$$\widetilde{R}_{\{i_1,\ldots,i_k\}}(\mathcal{I}^{\wp}) = \pi_{i_1,\ldots,i_k}(\sigma_{\mathsf{isNotNull}(i_1,\ldots,i_k)}\ (\sigma_{\mathsf{isNull}(j_1,\ldots,j_\ell)}\ R))(\mathcal{I}^{\mathbf{N}})$$

however, is true by the definition of $\mathcal{I}^{\wp}$ and $\mathcal{I}^{\mathbf{N}}$ as shown in Section 2:

$$\begin{aligned}\mathcal{I}^{\wp}(\widetilde{R}_{\{i_1,\ldots,i_k\}}) = \{t \in\ & ([1\cdots k] \mapsto \mathcal{C}) \mid \\ & \exists t' \in \mathcal{I}^{\mathbf{N}}(R).\ \forall j \in ([1\cdots n]\setminus\{i_1,\ldots,i_k\}).\ t'(j) = \mathbf{N} \wedge \\ & \qquad \forall j \in \{i_1,\ldots,i_k\}.\ t'(j) \neq \mathbf{N} \wedge \\ & \qquad \forall j \in \{1,\ldots,k\}.\ t(j) = t'(i_j)\}.\end{aligned}$$

□

Lemma 2. *Let ψ be a safe-range $\mathcal{FOL}$ closed formula. There is an $\mathcal{RA}$ expression $\widetilde{e}$ of arity zero, such that for any instance $\mathcal{I}^{\wp}$ with the Standard Name Assumption:*

$$\mathcal{I}^{\wp} \models_{\mathcal{FOL}} \psi \quad \textit{if and only if} \quad \widetilde{e}(\mathcal{I}^{\wp}) \neq \emptyset$$

Proof. Since null values are not present neither in $\mathcal{FOL}$ nor $\mathcal{RA}$, the lemma derives from the equivalence between safe-range relational calculus queries and relational algebra (see e.g. [3]). □

Now we can prove Proposition 2 using the above lemmata.

Proof (Proposition 2). Given a safe-range $\mathcal{FOL}^{\varepsilon}$ closed formula φ, by Theorem 1, there is a safe-range $\mathcal{FOL}$ formula $\widetilde{\varphi}$ such that $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \varphi$ iff $\mathcal{I}^{\wp} \models_{\mathcal{FOL}} \widetilde{\varphi}$.
By restricting to instances with the Standard Name Assumption, we can use Lemma 2 to conclude that there is an $\mathcal{RA}$ expression $\widetilde{e}$ of arity zero such that $\mathcal{I}^{\wp} \models_{\mathcal{FOL}} \widetilde{\varphi}$ iff $\widetilde{e}(\mathcal{I}^{\wp}) \neq \emptyset$; therefore $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \varphi$ iff $\widetilde{e}(\mathcal{I}^{\wp}) \neq \emptyset$
Finally we use Lemma 1 to show that there is an $\mathcal{RA}^{\mathbf{N}}$ expression e such that $\widetilde{e}(\mathcal{I}^{\wp}) \neq \emptyset$ iff $e(\mathcal{I}^{\mathbf{N}}) \neq \emptyset$; which enables us to conclude that $\mathcal{I}^{\varepsilon} \models_{\mathcal{FOL}^{\varepsilon}} \varphi$ iff $e(\mathcal{I}^{\mathbf{N}}) \neq \emptyset$. □

6 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., the recent thread

in SIGMOD Record [6,7]). 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 [2] and all 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 so they are outside the scope of our approach. 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.

To the best of our knowledge this is the first proposal to characterise the SQL semantics of null values by means of relational calculus. Moreover, we do properly extend Codd's theorem, stating the equivalence of the relational algebra with the relational calculus, to the case in which null values with the original SQL semantics are added to the languages.

It is worth to notice that the use of sub-relations to model the absence of values (shown in Section 4.1) is similar in spirit to the *horizontal decomposition* summarised e.g. in [1]. However, classically horizontal decomposition is more a modelling paradigm and must be performed "by hand". In our case, the decomposition is transparent to the modeller/user; in fact it is rather a technical mean in order to reduce our novel logic $\mathcal{FOL}^{\varepsilon}$ to standard first-order logic $\mathcal{FOL}$.

The SQL semantics for null values is also reflected in the satisfiability of integrity constraints (e.g. foreign keys), expressed usually as denial constraints using the query algebra itself. However, SQL:1999 introduced the possibility of specifying alternative semantics for the constraints (not based on the actual evaluation of SQL queries). In particular, foreign keys may be given optionally a semantic where null values would be interpreted as unknown values as opposed to nonexistent (see [9]). The formalisation of this alternative semantics of null values only within constraints (but not queries) will require some more work. Note that commercial relational database systems do not support this extension.

References

1. Date, C.J., Darwen, H.: Database Explorations: Essays on the Third Manifesto and Related Topics. Trafford Publishing (2010)
2. Imieliński, T., Lipski, Jr., W.: Incomplete information in relational databases. J. ACM **31**(4) (September 1984) 761–791
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. Rubinson, C.: Nulls, three-valued logic, and ambiguity in SQL: critiquing Date's critique. SIGMOD Record **36**(4) (December 2007) 13–17
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
9. 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