# Preservation of Codd semantics in databases with SQL nulls

*Konrad Pijanowski*


MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2023

# Abstract

Research in database theory usually represents missing values as *marked nulls*, whereas SQL-based systems use a single syntactic object `NULL`. To reconcile the mismatch in the expressivity of the two approaches, SQL nulls are often interpreted as *Codd nulls* - non-repeating marked nulls. However, if SQL nulls could truly be interpreted as Codd nulls, the Codd semantics should also be preserved by queries in their answers, which are incomplete databases themselves. Unfortunately, this is often not the case. In fact, it was shown that the class of relational algebra queries preserving Codd semantics is not recursively enumerable. Therefore, the best we can do is to recognise real-life queries preserving Codd semantics using various syntactic criteria.

The available sufficient conditions for Codd semantics preservation suffer from one crucial issue. Namely, the result of the verification depends on the exact formulation of the query. That is, the decision can be different for two equivalent relational algebra queries differing only in the order of the executed operations. In this project, we mitigate this issue for chained unions by coming up with a new weaker condition that guarantees formulation independent verification. Furthermore, we study the interpretation of SQL nulls which allows marked nulls to be repeated, for now, only among records that are duplicated in a table. We devise new sufficient conditions ensuring preservation of this new semantics and compare them to the conditions for the preservation of Codd semantics. Finally, we observe that both interpretations are equivalent when queries are evaluated using set semantics. Hence, we use these new conditions to propose weaker conditions for the preservation of Codd semantics over sets.

# Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Konrad Pijanowski*)

# Acknowledgements

I would like to express my gratitude to my supervisor, Paolo Guagliardo, for his invaluable guidance, encouragement, and unwavering support throughout the journey of completing my project. His expertise and constructive feedback helped me immensely in refining my research work.

I would also like to extend my sincere appreciation to my parents for their unconditional love and support, which have been the source of my strength and motivation during my time at the University of Edinburgh. Their constant encouragement and belief in me have been vital in shaping me into the person I am today. *Dziękuję!*

Furthermore, I am immensely grateful to all my friends. The time spent together gave me all the energy and motivation I needed to get through this demanding journey.

Lastly, I would like to thank all the individuals who have contributed in any way towards the successful completion of my project. Without you, this accomplishment would not have been possible.

# Table of Contents

# Chapter 1

# Introduction

## 1.1 Background, motivations and related work

Representing and handling incomplete information has been actively studied since the birth of relational database theory [1, 6]. Theoretical research usually represents missing values as *marked*, or *labelled*, *nulls* [2, 7]. However, this model is more expressive than the one employed by real-life SQL-based systems. In particular, using marked nulls we can express the fact that two nulls are equal to each other. On the other hand, this is impossible in SQL which represents nulls with the same syntactic symbol **NULL**. For example, $\perp_1 = \perp_1$ is always *true*, while **NULL** = **NULL** in SQL evaluates to *unknown*.

To reconcile this mismatch, the database community often models SQL nulls as *Codd nulls* - non-repeating marked nulls [2, 3]. The idea is that together with the adjusted comparison semantics that mimics the three-valued logic of SQL [4], a single **NULL** symbol is no longer a problem as each null is interpreted as a distinct null anyway. Although, this approach ignores the role of queries in database systems. If SQL nulls could truly be interpreted as Codd nulls, the Codd semantics should be preserved by queries in their answers, which are incomplete databases themselves [5].

The intuition behind this argument is depicted in Figure 1. To explain it here, let $\mathsf{codd}(D)$ be an interpretation of a SQL database $D$ in which all **NULL** values are replaced with fresh marked nulls. Technically, $\mathsf{codd}(D)$ is a set of isomorphic databases as the labels of the unique marked nulls can be chosen arbitrarily (e.g. $\{\perp_x, \perp_y\}$ instead of $\{\perp_1, \perp_2\}$).

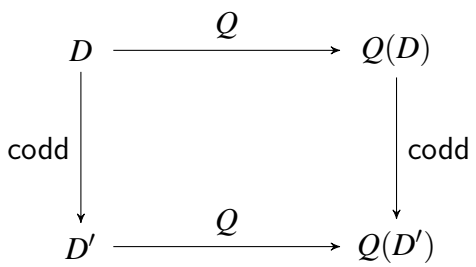



Figure 1: Condition for the preservation of Codd semantics for SQL nulls. Source: [5].

Then we say that the query $Q$ preserves Codd semantics on all SQL databases $D$, if the answer to $Q$ on a database $D' \in \text{codd}(D)$ can be obtained from the answer to $Q$ on $D$ by replacing all SQL nulls with Codd nulls.

To give an example of how these two can differ, let $D$ be a SQL database with two relations: $R = \{A : 1, \texttt{NULL}\}$ and $S = \{B : \texttt{NULL}\}$. Also, let $D'$ be an interpretation of $D$ in which all nulls were replaced by distinct marked nulls, e.g., $R = \{A : 1, \bot_R\}$ and $S = \{B : \bot_S\}$. Then, the answers to the query $R \times S$ evaluated on these databases are:

$Q(D)$:

| $A$ | $B$ |
|---|---|
| 1 | NULL |
| NULL | NULL |

$Q(D')$:

| $A$ | $B$ |
|---|---|
| 1 | $\bot_S$ |
| $\bot_R$ | $\bot_S$ |

Looking at the answer to the query on the database $D$, we could infer that the output contains three distinct marked nulls. This is because assuming Codd semantics, we interpret each NULL as a distinct labelled null. However, if we execute the same query on the database $D'$, we can reach a conflicting conclusion. Namely, we get the table with two duplicated marked nulls ($\bot_S$) for the attribute $B$, even though the initial table did not contain any repeated marked nulls. This leads to the inconsistency in the interpretation of the nulls in the results, and therefore we say that $Q$ does not preserve Codd semantics.

To close this gap between theory and practice, two main approaches have emerged. The first one calls for the introduction of marked nulls in SQL. Some attempts have already been made in this direction by [10]. However, as noted in the recent survey concerned with database practitioners' understanding of SQL nulls: "any changes could also lead to horrendous version control issues" [11]. On the other hand, the second approach, developed in [5], aims to identify relational algebra (RA) queries which preserve Codd semantics. This is achieved using a set of sufficient conditions (introduced later), which if satisfied, guarantee the preservation.

In this MInf Project, we expanded on the latter solution. Our work focused primarily on the issue of the formulation-dependent verification of the aforementioned conditions. That is the fact that given two equivalent RA queries, the conditions might be satisfied by one ordering of operations, but not the other. Ideally, we would like to be certain that given a set of conditions either all or none of the equivalent formulations of a query satisfy them. However, since any given query can be rewritten in infinitely many (often "creative") ways, providing such guarantees seems extremely unlikely. Nevertheless, there are cases when this inconsistency in the verification of the Codd semantics preservation is especially flagrant. To be precise, we have shown that the exact formulation of intersections or unions of multiple operands can affect the verdict when using the conditions from [5]. For example, it might be the case that the query $Q_1 \cup (Q_2 \cup Q_3)$ satisfies the conditions, whilst the query $(Q_1 \cup Q_2) \cup Q_3$ does not. This is highly undesirable as these RA operations are both associative and commutative giving the author of a query full freedom to shuffle the order of intermediate operations. Therefore, finding new weaker conditions and a procedure for verifying them that produce formulation-independent results for these operations is a big step forward in recognising the preservation of Codd semantics.

Moreover, we also started exploring interpretations of SQL nulls in which marked nulls

can repeat. This could allow us to utilise the more expressive nature of labelled nulls in computations based on SQL nulls. In general, we do not know if finding conditions for the preservation of null semantics with an arbitrary codd (null replacement) procedure is possible. Therefore, to begin with, we considered an interpretation in which records that are repeated in a table are assigned the same marked nulls. This led to a study of corresponding conditions and gave birth to questions such as: are these two interpretations comparable? Are these new conditions weaker or stronger? Do they differ under the bag and set semantics? We answer all of those in this report.

## 1.2 Contributions of this MInf Project

Motivated by the described research problems, over the last two years, we introduced a number of improvements to the process of recognising Codd semantics preserving queries described in [5].

To be precise, **last year in Part 1** [9]:

- we showed how the information about which attributes are certain to be made non-nullable by selection operations can be derived from their conditions;
- we introduced a variadic intersection operation together with a corresponding sufficient condition that enabled us to identify more intersections of multiple operands that preserve Codd semantics (by abstracting away their exact ordering information);
- we suggested a normalisation procedure that transforms the query in question into an equivalent query which is more likely to satisfy the sufficient conditions;
- we implemented the verification of sufficient conditions in relational algebra queries as a Java library *coddifier* [8].

Following up on these advancements, **this year in Part 2**:

- we devise a weaker sufficient condition for unions, *disjoint effective nullable attributes* (`DJEN`), which enables us to identify even more Codd semantics preserving queries;
- we prove that `DJEN` on its own guarantees formulation-independent recognition of Codd semantics preserving unions of multiple operands;
- we present sufficient conditions for the preservation of semantics of SQL nulls in which records that repeat in a table are assigned the same marked nulls;
- we compare the conditions for the two interpretations and show that the conditions for the original Codd semantics are weaker;
- we use the new conditions to derive sufficient conditions for RA queries evaluated under set semantics that are weaker than those proposed in [5].

# Chapter 2

# Preliminaries

Before we present any concrete findings, let us introduce the data model and the relational algebra query language based on which the sufficient conditions for the preservation of Codd semantics were derived. To be consistent, we repeat most of this chapter as written in Part 1 [9] of this project. In doing so, we use the same definitions and follow the same conventions as in [5].

## 2.1 Data model, schemas, and (incomplete) databases

We start by defining a *bag* as an unordered collection of objects in which instances of the same element, unlike in sets, can repeat. We say that an element $e$ in a bag $B$ has multiplicity $k$, denoted as $\#(e,B) = k$ or $e \in_k B$, if $e$ appears $k$ times in $B$. Similarly, we can write $e \in B$ and $e \notin B$ to state a general fact that $e$ is or is not in $B$, respectively. We also use notation $B \subseteq B'$, if $\#(e,B) \leq \#(e,B')$ for every $e \in B$. Finally, we define four bag operations: *union* $\cup$, *intersection* $\cap$, *difference* $-$, and *duplicate elimination* $\varepsilon$. If $e \in_m B$ and $e \in_n B'$, then: $\#(e,B \cup B') = m+n$, $\#(e,B \cap B') = \min(m,n)$, $\#(e,B-B') = \max(0,m-n)$, and $\#(e,\varepsilon(B)) = 1$ if $e \in B$, otherwise $\#(e,\varepsilon(B)) = 0$.

Now, let us take two countably infinite and disjoint sets of *names* and *values*. Any finite subset of names can be a *signature*. Then, a *record* is a map from some signature to values. Using these concepts we define a *table* as a bag of *records* over the same signature. $\mathsf{sig}(r)$ / $\mathsf{sig}(T)$ denotes the signature of a record $r$ / table $T$, respectively.

The *projection* of a record $r$ on a subset $\alpha$ of its signature is the restriction of $r$ on $\alpha$, denoted by $\pi_\alpha(r)$. For two records $r$ and $s$ of disjoint signatures, the *product* of $r$ with $s$, denoted by $r \times s$, is the record over $\mathsf{sig}(r) \cup \mathsf{sig}(s)$ whose projections on $\mathsf{sig}(r)$ and $\mathsf{sig}(s)$ are $r$ and $s$, respectively. For a record $r$, given $N \in \mathsf{sig}(r)$ and $N' \notin \mathsf{sig}(r)$, we define the following *renaming* operation:

$$\rho_{N \to N'}(r) \stackrel{\text{def}}{=\!=} \pi_{\mathsf{sig}(r)-N}(r) \times \{N' \mapsto r(N)\}\,.$$

The operations on records described above extend naturally to tables. The bag operations $\cup$, $\cap$ and $-$ can be applied to tables of the same signature, which ensures the result is a table. Duplicate elimination $\varepsilon$ applies without restrictions.

Next, a relational schema is a set of relation names together with a function $\mathsf{sig}$ which associates every relation name $R$ with a set of attributes $\mathsf{sig}(R)$ - its signature. Then, a database $D$ maps each relation name $R$ from the schema with a table $[\![R]\!]_D$ over the same signature as $R$. Each database instance can store values from only two countably infinite and disjoint sets of *constants* ($\mathsf{Const}$) and nulls ($\mathsf{Null}$). The $\mathsf{Null}$ set contains a special value $\mathtt{N}$ that is used to represent SQL's `NULL` object. Other nulls are denoted $\bot$, potentially with some subscript. By $\mathsf{Const}(D)$ and $\mathsf{Null}(D)$ we denote sets of constants and nulls present in a database $D$.

Real-life databases support several constraints on data in a table. We are specifically interested in constraints that mark attributes as `NOT NULL`. To reflect them in our model, we partition the signature of a relation $R$ into *nullable* and *non-nullable* signatures, denoted by $\mathsf{n\text{-}sig}(R)$ and $\mathsf{c\text{-}sig}(R)$ respectively. The non-nullable signature of $R$ is a set of its non-nullable attributes, that is, attributes that are allowed to take only constant values (e.g., because of the `NOT NULL` or `PRIMARY KEY` constraint). On the other hand, there is no such restriction for nullable attributes in the nullable signature of $R$ which can take both null and constant values.

Depending on the presence of nulls in a database as a whole, we can recognize four types of databases. A database $D$ is a:

- *Complete* database if $D$ does not contain any nulls - $\mathsf{Null}(D) = \varnothing$.
- *Naive* database if $D$ does not contain any SQL nulls - $\mathtt{N} \notin \mathsf{Null}(D)$.
- *SQL* database if all nulls in $D$ are SQL nulls - $\mathsf{Null}(D) = \{\mathtt{N}\}$.
- *Codd* database if $D$ is a naive database in which each null is different.

*Remark.* We can say that a table is a complete, naive, SQL, or Codd table if it satisfies equivalent conditions.

Directly related to the notion of SQL and Codd databases is the idea of Codd interpretation of SQL nulls. Rephrasing what we said in the introduction using the newly defined concepts, theoreticians model incomplete databases using naive databases, whereas in practice SQL operates on SQL databases. To bridge this gap, SQL databases are usually interpreted as Codd databases. This is achieved by replacing each SQL null $\mathtt{N}$ with a unique element of $\mathsf{Null} - \{\mathtt{N}\}$ that is not yet present in the database.

To formalise this idea, given a record $r$, we denote by $\mathsf{sql}(r)$ the record $r'$ over $\mathsf{sig}(r)$ such that:

$$r'(A) = \begin{cases} r(A) & \text{if } r(A) \in \mathsf{Const}, \\ \mathtt{N} & \text{otherwise} \end{cases}$$

Moreover, by $\mathsf{sql}^{-1}(r)$ we denote the set of all records $r'$ such that $\mathsf{sql}(r') = r$. The $\mathsf{sql}$ and $\mathsf{sql}^{-1}$ notation can be further extended to tables and databases. Namely, for a table $T$ over $\mathsf{sig}(T)$, $\mathsf{sql}(T)$ is a table over the same signature that consists of records $r$, such that:

$$\#\big(r, \mathsf{sql}(T)\big) = \sum_{s \in \mathsf{sql}^{-1}(r)} \#(s, T)$$

For a database $D$, $\mathsf{sql}(D)$ denotes a database, having the same schema as $D$, where:

$$[\![R]\!]_{\mathsf{sql}(D)} = \mathsf{sql}([\![R]\!]_D)$$

The $\mathsf{sql}^{-1}(T)$ and $\mathsf{sql}^{-1}(D)$ denote sets of all tables $T'$ and databases $D'$, respectively, such that $\mathsf{sql}(T') = T$ and $\mathsf{sql}(D') = D$.

Finally, we can formalise what we mean by the Codd interpretation of an SQL database. For that, we define $\mathsf{codd}(D)$ to be a set of all Codd databases in $\mathsf{sql}^{-1}(D)$. Even though this set may be infinite as the set of Null is countably infinite, all databases in it are isomorphic since they differ only in the names of the nulls. For that reason, we allow ourselves to talk about a single interpretation that is unique up to the renaming of nulls.

## 2.2 Query language

The sufficient conditions for Codd semantics preservation considered in this report apply to queries written in relational algebra for bags. To be consistent, we follow the syntax and semantics of the language as described in [5]. The syntax consists of two main constructs, that is *expressions* $E$ and *conditions* $\theta$, whose semantics are summarised in Figure 2.

A *term* $t$ is either a name or a value, and its semantics $[\![t]\!]_r$ is given with respect to a record $r$: if $t$ is a name in $\mathsf{sig}(r)$, then $[\![t]\!]_r = r(t)$, otherwise, if $t$ is a value, $[\![t]\!]_r = t$.

Atomic conditions consist of equality/inequality comparisons between terms and tests which determine whether a term is null or constant. Complex conditions are constructed from atomic ones by means of conjunction and disjunction. There is no explicit negation, as it can be propagated all the way down to atoms. The signature of a condition $\theta$, denoted by $\mathsf{sig}(\theta)$, is the set of names appearing in it. Its semantics $[\![\theta]\!]_r$ is defined with respect to a record $r$ such that $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(r)$: it can be either $\mathbf{t}$ (true) or $\mathbf{f}$ (false), as determined by the rules in Figure 2b.

$$
\begin{aligned}
[\![R]\!]_D &\text{ is given for every } R \\
[\![E_1 \,\mathsf{op}\, E_2]\!]_D &\stackrel{\text{def}}{=\!=} [\![E_1]\!]_D \,\mathsf{op}\, [\![E_2]\!]_D \\
&\quad \text{for } \mathsf{op} \in \{\times, \cup, \cap, -\} \\
[\![\pi_\alpha(E)]\!]_D &\stackrel{\text{def}}{=\!=} \pi_\alpha\big([\![E]\!]_D\big) \\
[\![\sigma_\theta(E)]\!]_D &\stackrel{\text{def}}{=\!=} \sigma_\theta\big([\![E]\!]_D\big) \\
[\![\varepsilon(E)]\!]_D &\stackrel{\text{def}}{=\!=} \varepsilon\big([\![E]\!]_D\big) \\
[\![\rho_{N\to N'}(E)]\!]_D &\stackrel{\text{def}}{=\!=} \rho_{N\to N'}\big([\![E]\!]_D\big)
\end{aligned}
$$

(a) EXPRESSIONS

$$
\begin{aligned}
[\![t_1 = t_2]\!]_r = \mathbf{t} &\iff [\![t_1]\!]_r = [\![t_2]\!]_r \in \mathsf{Const} \\
[\![t_1 \neq t_2]\!]_r = \mathbf{t} &\iff [\![t_1 = t_2]\!]_r \neq \mathbf{t} \\
[\![\mathsf{null}(t)]\!]_r = \mathbf{t} &\iff [\![t]\!]_r \in \mathsf{Null} \\
[\![\mathsf{const}(t)]\!]_r = \mathbf{t} &\iff [\![t]\!]_r \in \mathsf{Const} \\
[\![\theta_1 \wedge \theta_2]\!]_r = \mathbf{t} &\iff [\![\theta_1]\!]_r = [\![\theta_2]\!]_r = \mathbf{t} \\
[\![\theta_1 \vee \theta_2]\!]_r = \mathbf{t} &\iff [\![\theta_1]\!]_r = \mathbf{t} \;\vee\; [\![\theta_2]\!]_r = \mathbf{t}
\end{aligned}
$$

(b) CONDITIONS

Figure 2: Semantics of relational algebra. Source: [5].

For a table $T$ and a condition $\theta$ such that $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(T)$, we can then define the following *selection* operation:

$$\sigma_\theta(T) \stackrel{\text{def}}{=\!=} \{\, \underbrace{r, \ldots, r}_{k \text{ times}} \mid r \in_k T,\ [\![\theta]\!]_r = \mathbf{t} \,\}$$

By $\mathsf{c\text{-}sig}(\theta)$ we denote a set of all attributes in $\mathsf{sig}(\theta)$ for which the selection operation $\sigma_\theta$ is guaranteed to remove all records mapping any of these attributes to a null value [9]. It is defined inductively as follows:

$$\begin{aligned}
\mathsf{c\text{-}sig}(t_1 \neq t_2) &= \mathsf{c\text{-}sig}(\mathsf{null}(t)) = \varnothing \\
\mathsf{c\text{-}sig}(t_1 = t_2) &= \mathsf{sig}(t_1 = t_2) \\
\mathsf{c\text{-}sig}(\mathsf{const}(t)) &= \mathsf{sig}(\mathsf{const}(t)) \\
\mathsf{c\text{-}sig}(\theta_1 \wedge \theta_2) &= \mathsf{c\text{-}sig}(\theta_1) \cup \mathsf{c\text{-}sig}(\theta_2) \\
\mathsf{c\text{-}sig}(\theta_1 \vee \theta_2) &= \mathsf{c\text{-}sig}(\theta_1) \cap \mathsf{c\text{-}sig}(\theta_2)
\end{aligned}$$

As for expressions, these are names of base relations present in the schema that can be further composed using standard operations of union $\cup$, intersection $\cap$, difference $-$, Cartesian product $\times$, selection $\sigma$, projection $\pi$, renaming $\rho$, and duplicate elimination $\varepsilon$. The *base* of an expression $E$, denoted by $\mathsf{base}(E)$, is the set of relation names that appear in it (i.e., the set of its atomic subexpressions). The signature of each expression is defined recursively as follows:

$$\begin{aligned}
\mathsf{sig}(R) &\text{ is given for every } R \\
\mathsf{sig}(E_1 \,\mathsf{op}\, E_2) &= \mathsf{sig}(E_1) \text{ for } \mathsf{op} \in \{\cup, \cap, -\} \\
\mathsf{sig}(E_1 \times E_2) &= \mathsf{sig}(E_1) \cup \mathsf{sig}(E_2) \\
\mathsf{sig}\big(\sigma_\theta(E)\big) &= \mathsf{sig}\big(\varepsilon(E)\big) = \mathsf{sig}(E) \\
\mathsf{sig}\big(\pi_\alpha(E)\big) &= \alpha \\
\mathsf{sig}\big(\rho_{A \to B}(E)\big) &= \big(\mathsf{sig}(E) - \{A\}\big) \cup \{B\}
\end{aligned}$$

In a similar manner, we define the nullable signature of each expression:

$$\begin{aligned}
\mathsf{n\text{-}sig}(R) &\text{ is given for every } R \\
\mathsf{n\text{-}sig}(E_1 \,\mathsf{op}\, E_2) &= \mathsf{n\text{-}sig}(E_1) \cup \mathsf{n\text{-}sig}(E_2) \text{ for } \mathsf{op} \in \{\cup, \times\} \\
\mathsf{n\text{-}sig}(E_1 \cap E_2) &= \mathsf{n\text{-}sig}(E_1) \cap \mathsf{n\text{-}sig}(E_2) \\
\mathsf{n\text{-}sig}(E_1 - E_2) &= \mathsf{n\text{-}sig}(E_1) \\
\mathsf{n\text{-}sig}\big(\sigma_\theta(E)\big) &= \mathsf{n\text{-}sig}(E) - \mathsf{c\text{-}sig}(\theta) \\
\mathsf{n\text{-}sig}\big(\varepsilon(E)\big) &= \mathsf{n\text{-}sig}(E) \\
\mathsf{n\text{-}sig}\big(\pi_\alpha(E)\big) &= \mathsf{n\text{-}sig}(E) \cap \alpha \\
\mathsf{n\text{-}sig}\big(\rho_{A \to B}(E)\big) &= \mathsf{n\text{-}sig}(E)[A/B]
\end{aligned}$$

where $\mathsf{n\text{-}sig}(E)[A/B]$ means that attribute $A$ is replaced by $B$ in $\mathsf{n\text{-}sig}(E)$.

A relational algebra query over some schema is an expression that is well-defined with respect to this schema. One can recursively determine whether an expression is well-defined using the following rules:

- An atomic expression $R$ is well-defined if $R$ is a relation name in the schema.
- $E_1 \ \mathsf{op} \ E_2$, for $\mathsf{op} \in \{\cup, \cap, -\}$, is well-defined if both $E_1$ and $E_2$ are well-defined and $\mathsf{sig}(E_1) = \mathsf{sig}(E_2)$.
- $E_1 \times E_2$ is well-defined if $E_1$ and $E_2$ are well-defined and $\mathsf{sig}(E_1) \cap \mathsf{sig}(E_2) = \varnothing$.
- $\sigma_\theta(E)$ is well-defined if E is well defined and $\mathsf{sig}(\theta) \subseteq \mathsf{sig}(E)$.
- $\pi_\alpha(E)$ is well-defined if $E$ is well defined and $\mathsf{sig}(\alpha) \subseteq \mathsf{sig}(E)$.
- $\rho_{A \to B}(E)$ is well-defined if $E$ is well defined, $A \in \mathsf{sig}(E)$, and $B \notin \mathsf{sig}(E) - \{A\}$.
- $\varepsilon(E)$ is well-defined if $E$ is well-defined.

Closely related to the relational algebra query is a notion of its syntax tree.

**Definition 1** ([5])**.** The *syntax tree* of an RA query $Q$ is a binary (ordered) tree constructed as follows:

- Each relation symbol $R$ is a single node labelled $R$.
- For each unary operation symbol $\mathsf{op}_1$, the syntax tree of $\mathsf{op}_1(Q)$ has root labelled $\mathsf{op}_1$ and the syntax tree of $Q$ rooted at its single child.
- For each binary operation symbol $\mathsf{op}_2$, the syntax tree of $Q \ \mathsf{op}_2 \ Q'$ has root labelled $\mathsf{op}_2$ and the syntax trees of $Q$ and $Q'$ rooted at its left child and right child, respectively.

*Remark.* Each node in the syntax tree of $Q$ defines a subquery of $Q$, so we can associate properties of such queries with properties of syntax tree nodes.

## 2.3 Queries preserving Codd semantics

We have already said that in order to be able to interpret SQL nulls as Codd nulls it must be the case that this interpretation not only applies to input databases but is also preserved by query answers. Figure 1 in the introduction depicts the intuition behind the condition for the Codd semantics preservation, which is formally defined below.

**Definition 2** ([5])**.** A query $Q$ preserves Codd semantics if for every SQL database $D'$ it holds that $[\![Q]\!]_D \in \mathsf{codd}\left([\![Q]\!]_{D'}\right)$, where $D$ is the Codd interpretation of $D'$. This can be equivalently formulated as:

$$\mathsf{sql}\left([\![Q]\!]_D\right) = [\![Q]\!]_{\mathsf{sql}(D)} \qquad \text{(1a)} \qquad\qquad \text{and } [\![Q]\!]_D \text{ is a Codd table.} \qquad \text{(1b)}$$

Unfortunately, [5] showed that the class of queries preserving Codd semantics is not recursively enumerable. For that reason, we can only come up with syntactic restrictions that guarantee this property. Guagliardo and Libkin introduced in [5] a number of conditions which can be satisfied by a node in a syntax tree.

**Definition 3** ([5])**.** A node in the syntax tree of a query satisfies:

`NNC` *(non-nullable child)* - if one of its children is non-nullable.

`NNA` *(non-nullable ancestor/self)* - if either itself or one of its ancestors is non-nullable.

`DJN` *(disjoint nullable attributes)* - if its children have no common nullable attributes.

`DJB` *(disjoint bases)* - if its children have bases with no relation names in common.

Using these constraints they came up with sufficient conditions for the preservation of Codd semantics.

**Theorem 1** ([5])**.** *Let $Q$ be an RA query whose syntax tree is such that:*

a) *each* $\varepsilon$ *node satisfies* `NNC`*;*

b) *each* $\cap$ *and* $-$ *node satisfies* `DJN`*;*

c) *each* $\times$ *node satisfies* `NNA`*;*

d) *each* $\cup$ *node satisfies* `NNC` *or* `DJB` *or* `NNA`*.*

*Then, $Q$ preserves Codd semantics.*

Theorem 1 enables us to quickly verify whether a query is guaranteed to preserve Codd semantics. We will illustrate this process on the query $\pi_{A,B}(T \times S) \cap ((R-T) \cup \varepsilon(U))$ whose syntax tree is presented in Figure 3. The signature of each subexpression is given to the left of the node (underlined attributes are non-nullable). The respective sufficient conditions satisfied by the nodes are marked on the right. Now, in the right subquery, the $\varepsilon$ node satisfies `NNC` as relation $U$ is non-nullable. The $-$ node satisfies `DJN`, as its children have non-overlapping nullable signatures. The union satisfies all respective conditions, although one is enough for the preservation of Codd semantics. In the left subquery, the $\times$ node satisfies the `NNA` condition because the root of the query is non-nullable, which is the case, as the $\cap$ node satisfies `DJN`.

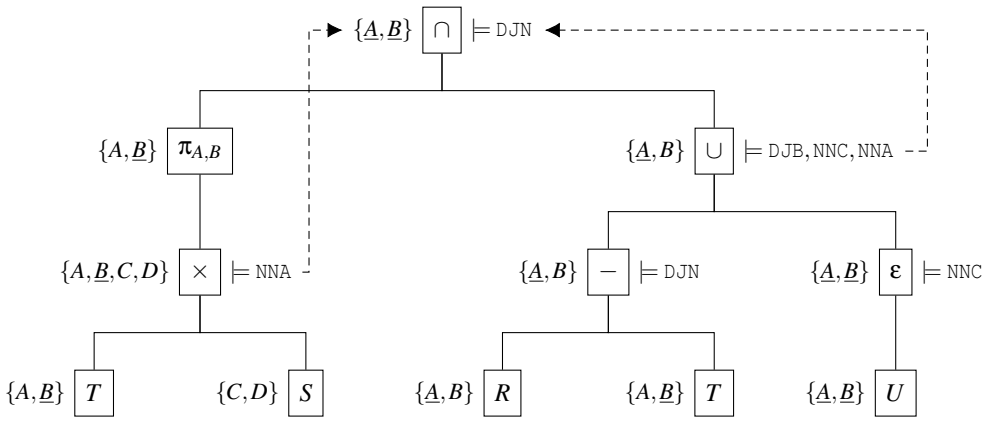



Figure 3: Syntax tree of the query $\pi_{A,B}(T \times S) \cap ((R-T) \cup \varepsilon(U))$. Each node is marked with the sufficient conditions it satisfies. The underlined attributes are non-nullable.

# Chapter 3

# New Condition for Union

As we have already discussed in Part 1 [9], recognising a RA query as Codd semantics preserving does not depend only on its semantics, but also on its formulation. We argued that this could be especially problematic for queries involving operations that are both associative and commutative, that is intersection and union. When multiple such operations are chained together, their exact ordering may impact the properties of individual nodes in a syntax tree, and hence the satisfiability of the sufficient conditions. To mitigate this problem for intersections, we:

1. introduced a variadic intersection operation that abstracts away the ordering information while capturing the information about all operands participating in the intersection,
2. used query transformations to normalise the query to a form that is more likely to satisfy the conditions.

Combining the two solutions helped us to capture the Codd semantics preservation of queries involving chained intersections, regardless of their exact ordering.

Following up on the previous work, in this chapter, we will focus on another associative and commutative operation - the union. To make our considerations more precise, let us first define the concept of a chain of unions that will be the subject of this study.

**Definition 4.** A *chain of unions* is a query $Q_\cup$ complying with the grammar $Q_\cup := Q \cup Q$ and $Q := Q_\cup \mid Q'$, where $Q'$ is a query that does not have $\cup$ as its root.

Equipped with this new definition, we can illustrate the problem with chains of unions using the following example.

**Example 1.** Consider a union of base relations $R_1$, $R_1$ (duplicated on purpose), $R_2$, and $R_3$, where $R_i$, for $i \in \{1,2,3\}$, is over the attribute $A$ and only $R_1$ has $A$ marked as `NOT NULL`. Figure 4 presents syntax trees of two possible formulations of such union. The signature of each node is displayed to its left with non-nullable attributes underlined. Now, even though this query must be Codd semantics preserving, the chain of unions on the left does not capture this property as the $\cup^1$ node fails to satisfy any of the conditions. The bases of $\cup^2$ and $\cup^3$ are not disjoint ($R_1$ appears in both of them),

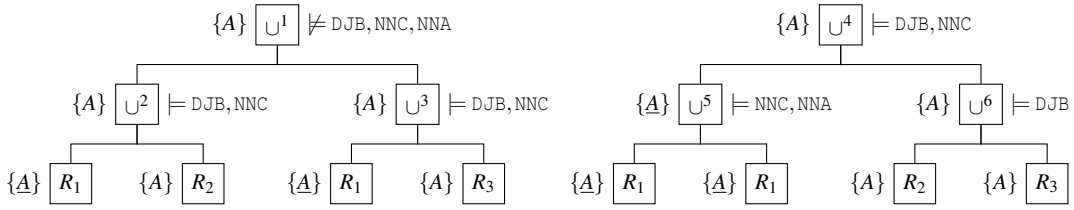



Figure 4: Annotated syntax trees of two possible formulations of the union from Example 1.

hence DJB is not satisfied. Also, none of these union nodes is non-nullable so the NNC and NNA conditions cannot be met. Meanwhile, the formulation on the right can be recognised as Codd semantics preserving, since all union nodes satisfy appropriate conditions.

Interestingly, even though the structure of the two chains is the same, swapping the two operands, $R_1$ and $R_2$, completely changes the number and the type of conditions satisfied by each $\cup$ node. Most notably, the $\cup^1$ node on the left-hand side does not satisfy any conditions, while $\cup^4$ on the right-hand side satisfies both DJB and NNC which are concerned with seemingly unrelated properties of the subqueries. This conflicts with the intuition that the order of operations should not matter for the preservation of Codd semantics. Ideally, the preservation should be captured regardless of the exact formulation of a union chain.

To solve this problem, in this chapter, in sections 3.1 and 3.2, we introduce a new variadic union operation together with the PDJB condition that guarantees the preservation of Codd semantics by union nodes. Next, in section 3.3, we identify the limitations of PDJB that make it unnecessarily restrictive. Then, in sections 3.4 and 3.5, we mitigate the recognised issues of PDJB, which leads us to the formulation of the weaker DJEN condition that enables us to Capture more queries preserving Codd semantics. Finally, in section 3.6, we present the formal proof that with DJEN we do not need the variadic union to guarantee the formulation-independent verification process for chains of unions.

## 3.1 Variadic union

Our first attempt to capture Codd semantics preservation of various chains of unions is to represent them using a single variadic operation. Technically, this only introduces yet another possible formulation of the same query and leaves the problem for a chain of binary unions intact. However, as we will show, every chain of unions is equivalent to some variadic union operation, therefore having weaker conditions for the $n$-ary operator will help us to classify all equivalent chains of unions as Codd semantics preserving queries.

To begin, let us extend the language of relational algebra with a new operation: *n-ary union* $\bigcup(E_1, \ldots, E_n)$. It is well-defined with respect to a schema when:

- $E_1, \ldots, E_n$ are well-defined expressions;
- $\text{sig}(E_1) = \cdots = \text{sig}(E_n)$

The signature of a well-defined $n$-ary union $\bigcup(E_1,...,E_n)$ is defined as follows:

$$\text{sig}\Big(\bigcup(E_1,\ldots,E_n)\Big) \stackrel{\text{def}}{=\!=} \text{sig}(E_1) = \cdots = \text{sig}(E_n)$$

The set of nullable attributes is given by:

$$\text{n-sig}\big(\bigcup(E_1,...,E_n)\big) \stackrel{\text{def}}{=\!=} \bigcup_{i=1}^{n} \text{n-sig}(E_i)$$

The semantics of an $n$-ary union $\bigcup(E_1,\ldots,E_n)$ w.r.t. a database $D$ is the following:

$$\Big[\!\Big[\bigcup(E_1,\ldots,E_n)\Big]\!\Big]_D \stackrel{\text{def}}{=\!=} [\![E_1]\!]_D \cup \cdots \cup [\![E_n]\!]_D$$

*Remark.* The RHS of the previous definition is well-defined since the union operation is associative and commutative.

For notational convenience, we also define the $n$-ary union of bags $B_1,\ldots,B_n$ as $\bigcup(B_1,\ldots,B_n) = B_1 \cup \cdots \cup B_n$. Then, the following holds trivially:

$$\Big[\!\Big[\bigcup(E_1,\ldots,E_n)\Big]\!\Big]_D = \bigcup([\![E_1]\!]_D,\ldots,[\![E_n]\!]_D) \tag{2}$$

The definition of the syntax tree is also expanded to accommodate the $n$-ary operator:

- For each operator $\bigcup$, the syntax tree of $\bigcup(Q_1,...,Q_n)$ has the root labelled with $\cup$ and syntax trees of $Q_1,...,Q_n$ as its children. The order of operands is preserved, meaning that the syntax trees of $Q_1$ and $Q_n$ are rooted as the leftmost and rightmost child nodes respectively.

Clearly, all of the above definitions are generalisations of the corresponding definitions for the binary union operation. For that reason, it should not be a surprise that $Q_1 \cup Q_2 \equiv \bigcup(Q_1,Q_2)$. As expected, the semantics of these queries, as well as their properties (e.g., the signature, the nullable signature, etc.), are the same without regard to whether the rules for the binary union or the $n$-ary union are used.

Next, we present relevant properties of the newly defined $n$-ary union, which we will later use in proofs.

**Lemma 1** (Algebraic Properties of the $n$-ary Union of Bags)**.** *Let $B_1,\ldots,B_n$, for $2 \leq k < n$, be bags. Then:*

*a) The order of bags in an n-ary union does not affect the result;*

*b)* $\bigcup\big(\bigcup(B_1,\ldots,B_k),B_{k+1},\ldots,B_n\big) = \bigcup(B_1,\ldots,B_n)$.

These follow directly from the definition of the $n$-ary union of bags.

*Remark.* Lemma 1 presents the properties of the $n$-ary union of bags. However, because of the relation (2), all of these properties can be lifted to the query level as long as the RA expression $\bigcup(E_1,\ldots,E_n)$ is well-defined.

Knowing the basic properties of the $n$-ary union, it should be clear that every chain of unions $Q_{\cup}$ combining expressions $E_1$ to $E_n$ is equivalent to the expression $\bigcup(E_1, \ldots, E_n)$. To put it in another way, regardless of the exact order in which we decide to combine these expressions, the outcome will be always the same as if we combined them "simultaneously" using a single $n$-ary union.

## 3.2 Pairwise disjoint bases of nullable children

The problem that a union poses for the preservation of Codd semantics is related to the condition (1b). This is because marked nulls can be duplicated by a union even if its operands are Codd tables [5]. Take a query $R \cup R$ as an example. It will not produce a Codd table if the relation $R$ contains any nulls. The immediate observation for the binary operation is that:

- only nullable operands can pose problems for the preservation of Codd semantics, or equivalently, that non-nullable cannot - which is reflected by the `NNC` constraint;
- the union can produce new repetitions of a marked null if and only if it comes from a base relation that is mentioned in both operands. This is because the sets of marked nulls present in two different base relations must be disjoint in every Codd database - hence we have the `DJB` condition.

Projecting these observations onto the realm of multiple operands, we can observe that non-nullable children are not a problem and we can safely focus only on nullable ones. Thus, to guarantee that no marked nulls are duplicated in the output of a variadic union, we can require that bases of nullable operands are pairwise disjoint.

**Definition 5.** A node in the refined syntax tree of a query satisfies `PDJB` (*pairwise disjoint bases of nullable children*) if the bases of its nullable children are pairwise disjoint.

We can now expand the sufficient conditions for the preservation of Codd semantics from Theorem 1 with the new sufficient condition for the $n$-ary union:

**Theorem 2.** *Let Q be an RA query whose syntax tree is such that:*

a-d) *As in Theorem 1*

e) *Each* $\bigcup$ *(variadic union) node satisfies* `PDJB` *or* `NNA`.

*Then, Q preserves Codd semantics.*

Now, before we prove the theorem, let us introduce one technical result from [5].

**Proposition 1** (Proposition 4 in [5])**.** *Every RA query whose syntax tree is such that:*

- *each* $\varepsilon$ *node satisfies* `NNC`*,*
- *each* $\cap$ *and* $-$ *node satisfies* `DJN`

*satisfies the condition* (1a) *on all databases (not only Codd ones).*

See Appendix B for the extension of the proof from [5] covering the variadic union operator. Finally, with the help of this proposition, we can prove our theorem.

**Proof of Theorem 2.** Let $Q$ be a query whose nodes in the syntax tree satisfy one of the conditions required by the theorem, that is: every $\varepsilon$ node satisfies `NNC`; every $\cap$ and $-$ node satisfies `DJN`; every $\times$ node satisfies `NNA`; every $\cup$ (binary union) node satisfies `NNA` or `NNC` or `DJB`; and every $\bigcup$ (variadic union) node satisfies `PDJB` or `NNA`. To prove the theorem we need to show that conditions (1a) and (1b) hold for every Codd database $D$. Clearly, $Q$ satisfies (1a) by Proposition 1. In order to show that $Q$ satisfies the condition (1b), we need to ensure that for every Codd database $D$, $[\![Q]\!]_D$ is a Codd table. To this end, we proceed with the proof by induction on the structure of $Q$:

**Base cases** ([5]):

- $Q$ is a relation name $R$. In such a case, $[\![Q]\!]_D$ is trivially a Codd table since $D$ is a Codd database.
- $Q$ is non-nullable. Then $[\![Q]\!]_D$ is a complete table which is also a Codd table.

**Induction**:

Here, we only extend the proof of Theorem 1 from [5] with the inductive steps for the new $n$-ary union. For the remaining operations and conditions, see the proof in [5].

First, note that the case when $Q = \bigcup(Q_1, \ldots, Q_n)$ satisfies the `NNA` condition is already covered in the base case as the condition implies that $Q$ is non-nullable.

- $Q$ is $\bigcup(Q_1, \ldots, Q_n)$ and satisfies `PDJB`:

Let us partition the operands of the $n$-ary union into two sets of nullable and non-nullable children. Without loss of generality, let us assume that, for some $k \leq n$, $Q_1$ to $Q_k$ are nullable and the remaining subqueries $Q_{k+1}$ to $Q_n$ are not. Clearly, if some null were to repeat in $[\![Q]\!]_D$, it must come from the nullable children, as $\mathsf{Null}([\![Q_{k+1}]\!]_D) = \cdots = \mathsf{Null}([\![Q_n]\!]_D) = \varnothing$.

By the induction hypothesis, $[\![Q_1]\!]_D$ to $[\![Q_k]\!]_D$ are Codd tables. Thus, if some null was duplicated in $[\![Q]\!]_D$, it must come from two different children. So, we need to show that $\mathsf{Null}([\![Q_i]\!]_D) \cap \mathsf{Null}([\![Q_j]\!]_D) = \varnothing$ for all distinct $i, j \in [1, k]$. For this, recall that:

$$\mathsf{Null}([\![Q_i]\!]_D) \subseteq \bigcup_{R \in \mathsf{base}(Q_i)} \mathsf{Null}([\![R]\!]_D)$$

The `PDJB` condition guarantees that bases of nullable children are disjoint, that is $\mathsf{base}(Q_i) \cap \mathsf{base}(Q_j) = \varnothing$ for all pairs of $i$ and $j$, therefore

$$\bigcup_{R \in \mathsf{base}(Q_i)} \mathsf{Null}([\![R]\!]_D) \cap \bigcup_{R \in \mathsf{base}(Q_j)} \mathsf{Null}([\![R]\!]_D) = \varnothing$$

and thus

$$\mathsf{Null}([\![Q_i]\!]_D) \cap \mathsf{Null}([\![Q_j]\!]_D) \subseteq \bigcup_{R \in \mathsf{base}(Q_i)} \mathsf{Null}([\![R]\!]_D) \cap \bigcup_{R \in \mathsf{base}(Q_j)} \mathsf{Null}([\![R]\!]_D) = \varnothing$$

completing the proof. □

*Remark.* The attentive reader might notice that all conditions for variadic union nodes can also apply to binary union nodes since $\bigcup(Q_1, Q_2) \equiv Q_1 \cup Q_2$. Nevertheless, PDJB is not listed as a condition for the latter. This is because for nodes with two children, PDJB is equivalent to the disjunction of NNC and DJB (see Proposition 10 in Appendix A), which is already included in the existing conditions for a binary union node.

Now, we would like to use the variadic union together with its corresponding conditions to capture the preservation of Codd semantics of chained unions regardless of their exact formulation. To do so, we can replace a chain of unions in a query with an equivalent $n$-ary union and then check for the conditions of Theorem 2. Meanwhile, the next proposition guarantees that the resulting variadic union node will satisfy PDJB whenever there exists any equivalent chain of unions whose all union nodes satisfy PDJB as well.

**Proposition 2.** *An n-ary union node* $\bigcup$ *satisfies the* PDJB *condition if and only if there exists an equivalent chain of (binary) unions in which every* $\cup$ *node satisfies a* DJB *or* NNC *condition.*

We include the proof in Appendix B.

Coming back to the query from Example 1, we could express the union of operands $R_1$, $R_1$ (duplicated on purpose), $R_2$, $R_3$ as $(R_1 \cup R_2) \cup (R_1 \cup R_3)$. Unfortunately, as we showed before, this formulation does not satisfy the sufficient conditions (left syntax tree in Figure 4). However, now we can represent this chain of unions as $\bigcup(R_1, R_1, R_2, R_3)$. Keeping in mind that $R_1$ was non-nullable, the only pair of nullable operands is $R_2$ and $R_3$ whose bases are clearly disjoint, therefore this union node satisfies PDJB. As a result, the whole union can be recognised as Codd semantics preserving even though the original chain of unions was not recognised as such.

## 3.3 Problems with PDJB

In the previous section, we suggested a solution to the problem of formulation-dependent verdicts for queries containing chained unions. However, Proposition 2 on which the whole process relies, also tells us that we cannot identify in this way any ”semantically new” chains of unions preserving the Codd semantics. That is, by using variadic unions with the PDJB condition, we are only able to recognise formulations of chains of unions for which there already existed some formulation satisfying the conditions of Theorem 1. Or in other words, if all formulations of a union of multiple operands could not be identified as Codd semantics preserving before, then they still cannot be recognised as such now. For this reason, in this and the following sections, we will identify what makes the PDJB condition unnecessarily restrictive and propose a more relaxed condition which will allow us to identify more queries preserving Codd semantics.

We have already said that the union poses problems with preserving Codd semantics when the same null value is propagated from two or more different operands. To avoid it, we have required nullable operands to have disjoint bases (PDJB condition). This approach effectively treats the relation name as a proxy for the possible presence of a

marked null from the base relation in the answer to the query. Namely, any marked null from some relation $R$ can appear in an answer to a query $Q$ if and only if $R \in \mathsf{base}(Q)$. Hence, there exists a possibility of duplicating a marked null from $R$ if it appears in more than one child. This heuristic works in many cases, although we have identified three problems that make its logic unnecessarily restrictive.

Firstly, the `PDJB` condition does not account for non-nullable relations in the bases of the union's children. That is to say, for the query $Q = Q_1 \cup Q_2$, if all relations in the set $\mathsf{base}(Q_1) \cap \mathsf{base}(Q_2)$ are non-nullable, then $Q$ also preserves Codd semantics. This is because non-nullable relations do not contain any nulls which could be duplicated.

Moreover, not all nullable relations in a base of a query can propagate their null values to the query's output. For example, the query $Q = S - R$ has both $S$ and $R$ in its base, while no nulls from $[\![R]\!]_D$ can ever be found in $[\![Q]\!]_D$, regardless of the database $D$. For that reason, the query $R \cup (S - R)$ preserves the Codd semantics, despite the fact that `DJB` can never be satisfied by the union node.

Finally, using a relation name as a proxy for the presence of a marked null is too coarse and can be made more precise. For instance, consider relation $R$ over nullable attributes $A$ and $B$. Clearly, marked nulls mapped to the attribute $A$ in $R$ must differ from marked nulls mapped to the attribute $B$. More formally:

$$\mathsf{Null}(\pi_A([\![R]\!]_D)) \cap \mathsf{Null}(\pi_B([\![R]\!]_D)) = \varnothing$$

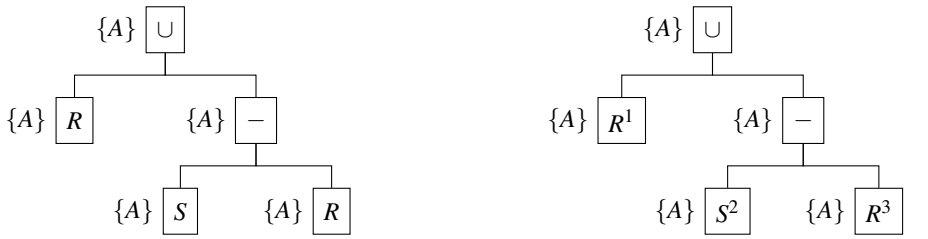
for all Codd databases $D$. This is because marked nulls in a Codd table must be unique across the attributes. Therefore, the query $\pi_A(R) \cup \rho_{B \to A}(\pi_B(R))$ will also always preserve Codd semantics even though the `DJB` condition cannot be satisfied.

## 3.4 Tracking value propagation

The three problems described before are caused by the fact that the base of a query does not do a good job of tracking how values from the base relation propagate through the query to the output table. However, before we introduce a better method of doing exactly that, we need a way to distinguish multiple appearances of the same relation in the query. To illustrate why, consider the query $R \cup (S - R)$, whose syntax tree is presented in Figure 5a. We would like to be able to say that the relation $R$ that is the left child of the union node can propagate nullable values to the query answer, while the same relation $R$ that is the right child of the difference node cannot.

Currently, we cannot differentiate the two because they are referenced using the same relation name $R$. For that reason, we add a unique superscript index to each "use" of the relation name in the query. We read such notation $R^i$ as *the instance $R^i$ of the relation $R$*. Clearly, $[\![R^i]\!]_D = [\![R]\!]_D$ for all instances of $R$. Using the new terminology and the instance annotations as in Figure 5b, we can unambiguously state that instances $R^1$ and $S^2$ can propagate their values to the query answer, while the instance $R^3$ cannot.

Moreover, for notational convenience, we overload the meaning of the base of a query. When we talk about the base of a query in the context of base relations, we consider the base to include names of relations only, e.g., $R$, $S$, etc. Whereas, when we talk about




(a) Without relation instance annotations (b) With relation instance annotations

Figure 5: Syntax trees of the query $R \cup (S - R)$.

the base of a query in the context of relation instances, we consider the base to include names of all relation instances, i.e., $R^1$, $S^2$, $R^3$, etc. So, the variable $P$ in the statement $\forall P \in \mathsf{base}(R^1 \cup (S^2 - R^3))$ would take a value of relations $R$ and $S$, while the variable $P^i$ in the statement $\forall P^i \in \mathsf{base}(R^1 \cup (S^2 - R^3))$ would take a value of relation instances $R^1$, $S^2$, and $R^3$. Despite this ambiguity, we ensure that the meaning of the base will be always clear from the context.

At last, being able to distinguish between individual uses of the relation name in the query, we introduce a concept of an *effective signature map*. A function that will help us to track the propagation of values from leaves to the root of a query's syntax tree.

**Definition 6** (*Effective Signature Map*)**.** Let $R$ be a relation in a database $D$ and $Q$ be an RA query. We will write $\phi_{R^i}(Q)(A) = B$ to denote that values associated with an attribute $A$ in the instance $R^i$ of the relation $R$ can be propagated to and appear among values associated with an attribute $B$ in $[\![Q]\!]_D$. The effective signature map is defined recursively in the following manner:

$$
\begin{aligned}
\phi_{R^i}(R^i) &\stackrel{\text{def}}{=} \{\ (A,A) \mid A \in \mathsf{sig}(R)\ \} \\
\phi_{R^i}(R^j) &\stackrel{\text{def}}{=} \varnothing, \text{ where } R^j \text{ is any instance of any relation other than } R^i \\
\phi_{R^i}(\rho_{N\to N'}(Q)) &\stackrel{\text{def}}{=} \begin{cases} \phi_{R^i}(Q)[(A,N)\backslash(A,N')] & \text{if } (A,N) \in \phi_{R^i}(Q), \\ \phi_{R^i}(Q) & \text{otherwise} \end{cases} \\
\phi_{R^i}(\pi_\alpha(Q)) &\stackrel{\text{def}}{=} \{\ (A,B) \in \phi_{R^i}(Q) \mid B \in \alpha\ \} \\
\phi_{R^i}(\sigma_\theta(Q)) &\stackrel{\text{def}}{=} \phi_{R^i}(Q) \\
\phi_{R^i}(\varepsilon(Q)) &\stackrel{\text{def}}{=} \phi_{R^i}(Q) \\
\phi_{R^i}(Q_1 \,\mathsf{op}\, Q_2) &\stackrel{\text{def}}{=} \{\ (A,B) \mid (A,B) \in \phi_{R^i}(Q_1) \text{ or } (A,B) \in \phi_{R^i}(Q_2)\ \} \\
&\quad \text{for } \mathsf{op} \in \{\times, \cup, \cap\} \\
\phi_{R^i}(Q_1 - Q_2) &\stackrel{\text{def}}{=} \phi_{R^i}(Q_1)
\end{aligned}
$$

Defined in this way, $\phi_{R^i}(Q)$ is a partial function from $\mathsf{sig}(R)$ to $\mathsf{sig}(Q)$. It is partial as values from some attributes of $R^i$ might be dropped by a query via projection operation, in which case we lose the mapping for the removed attributes.

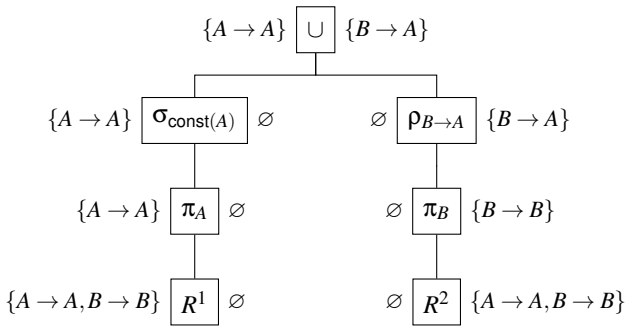



Figure 6: Analysis of the propagation of values from Example 2.

For convenience, we will write $\phi_{R^i}(Q)(X)$, where $X \subseteq \mathsf{sig}(R)$, to represent a set:

$$\{\phi_{R^i}(Q)(A) \mid A \in X \text{ and } \phi_{R^i}(Q)(A) \text{ is defined}\}$$

Furthermore, note that $\phi_{R^i}(Q)$ is injective because all renamings in $Q$ must be well-defined. For that reason, we can define $\phi^{-1}_{R^i}(Q) = \{\ (B,A) \mid (A,B) \in \phi_{R^i}(Q)\ \}$ which denotes that values of attribute $B$ in $Q$ can consist of values associated with attribute $A$ in $R^i$. We will abuse the nomenclature and refer to $\phi^{-1}_{R^i}(Q)$ as the "inverse" of $\phi_{R^i}(Q)$.

**Example 2.** An example of how the effective signature map keeps track of the value propagation through the query is depicted in Figure 6. We will refer to this query using the symbol $Q$. We assume that the relation $R$ is over nullable attributes $A$ and $B$. Then, annotations to the left of a node represent effective signature mappings between the instance $R^1$ and the node, and to the right between the instance $R^2$ and the node. Analysing the mappings for the root node, we see that the attribute $A$ in the query answer can consist of values from both attributes $A$ and $B$ in the relation $R$ as $\phi^{-1}_{R^1}(Q)(A) = A$ and $\phi^{-1}_{R^2}(Q)(A) = B$, respectively.

*Remark.* Observe that the map contains no entries if the relation instance is not in the subtree rooted at the node with respect to which the map is computed (denoted by the $\varnothing$ symbol in the diagram).

Now, with the help of the effective signature map, we can keep track of the propagation of values through the query at a more granular level. Then, using its inverse we can find the corresponding attributes in the particular instance of the relation. This solves the first discussed problem with the `PDJB` condition - the fact that not all relations propagate all values to the output of a query. However, what we really want to keep track of is what attributes in a relation instance can propagate nulls to the query output. One naive approach to achieving this could be using the inverse of the effective signature map:

$$\mathsf{n\text{-}sig}(R^i) \cap \phi^{-1}_{R^i}(Q)(\mathsf{n\text{-}sig}(Q)) \tag{3}$$

The left-hand side of this formula is the set of all nullable attributes in $R^i$. The right-hand side represents the set of attributes in $R^i$ that can propagate values to nullable attributes in $Q$. Consequently, the set of attributes which can contribute null values to the nullable attributes in the output is the intersection of the two sets.

However, this approach is not without faults. Most importantly, it misses the fact that attributes that are nullable in $R^i$ and propagate to nullable attributes in $Q$ may

have their null values removed in the process. For example, this happens in the query presented in Figure 6 that we analysed before (again, we refer to this query as $Q$). It can be easily verified that: $\mathsf{n\text{-}sig}(R^1) = \{A,B\}$, $\mathsf{n\text{-}sig}(Q) = \{A\}$, and $\phi^{-1}_{R^1}(Q) = \{(A,A)\}$. Using equation (3), we could infer that the attribute $A$ of $R^1$ can propagate null values to the answer:

$$\mathsf{n\text{-}sig}(R^1) \cap \phi^{-1}_{R^1}(Q)(\mathsf{n\text{-}sig}(Q)) = \{A,B\} \cap \phi^{-1}_{R^1}(Q)(\{A\}) = \{A,B\} \cap \{A\} = \{A\}$$

However, this is not the case since the selection on the constant value of $A$ precedes the union. Therefore, no null values from $R^1$ can propagate to the query output. Only the attribute $B$ in $R^2$ can contribute null values to the answer.

For this reason, we need to adjust the definition of the effective signature map so that it explicitly filters out the attributes that can no longer contain nullable values.

**Definition 7** (*Effective Nullable Signature Map*)**.** Let $R$ be a relation in a database $D$ and $Q$ be an RA query. We will write $\psi_{R^i}(Q)(A) = B$ to denote that null values associated with a nullable attribute $A$ in the instance $R^i$ of relation $R$ can be propagated to and appear among null values associated with a nullable attribute $B$ in $[\![Q]\!]_D$. The effective nullable signature map is defined recursively in the following manner:

$$\begin{aligned}
\psi_{R^i}(R^i) &\stackrel{\text{def}}{=} \{\ (A,A) \mid A \in \mathsf{n\text{-}sig}(R)\ \} \\
\psi_{R^i}(R^j) &\stackrel{\text{def}}{=} \varnothing, \text{ where } R^j \text{ is any instance of any relation other than } R^i \\
\psi_{R^i}(\rho_{N\to N'}(Q)) &\stackrel{\text{def}}{=} \begin{cases} \psi_{R^i}(Q)[(A,N)\setminus(A,N')] & \text{if } (A,N) \in \psi_{R^i}(Q), \\ \psi_{R^i}(Q) & \text{otherwise} \end{cases} \\
\psi_{R^i}(\pi_\alpha(Q)) &\stackrel{\text{def}}{=} \{\ (A,B) \in \psi_{R^i}(Q) \mid B \in \alpha\ \} \\
\psi_{R^i}(\sigma_\theta(Q)) &\stackrel{\text{def}}{=} \{\ (A,B) \in \psi_{R^i}(Q) \mid B \in \mathsf{n\text{-}sig}(\sigma_\theta(Q))\ \} \\
\psi_{R^i}(\varepsilon(Q)) &\stackrel{\text{def}}{=} \psi_{R^i}(Q) \\
\psi_{R^i}(Q_1\ \mathsf{op}\ Q_2) &\stackrel{\text{def}}{=} \{\ (A,B) \in \big(\psi_{R^i}(Q_1) \cup \psi_{R^i}(Q_2)\big)\ \} \\
&\quad \text{for } \mathsf{op} \in \{\times, \cup\} \\
\psi_{R^i}(Q_1 \cap Q_2) &\stackrel{\text{def}}{=} \{\ (A,B) \in \big(\psi_{R^i}(Q_1) \cup \psi_{R^i}(Q_2)\big) \mid B \in \mathsf{n\text{-}sig}(Q_1 \cap Q_2)\ \} \\
\psi_{R^i}(Q_1 - Q_2) &\stackrel{\text{def}}{=} \psi_{R^i}(Q_1)
\end{aligned}$$

*Remark.* The properties of and conventions for the effective nullable signature map are the same as in the case of the effective signature map.

With the help of the effective nullable signature map, we can directly model the propagation of nullable attributes through the query.

**Example 3.** Figure 7 depicts the effective nullable signature maps for the query from Example 2, assuming that both attributes of $R$ are nullable. Again, the map between $R^1$ and any given node is presented to the left of that node and to the right for $R^2$. The key difference is, that this time, we removed the mapping for the attribute $A$ at the node $\sigma_{\mathsf{const}(A)}$ as it filters out all null values that propagated from that attribute. As a result,

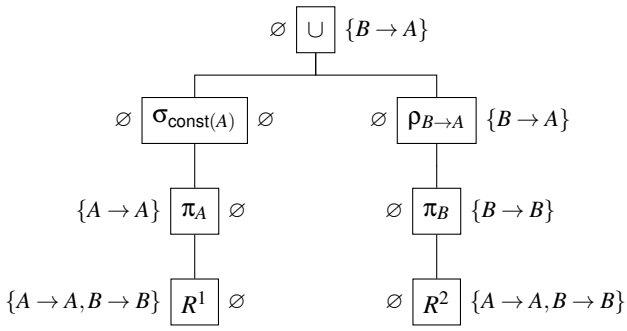



Figure 7: Analysis of the propagation of nulls from Example 3.

we can correctly identify that only null values from the attribute $B$ of $R^2$ can propagate to the query answer.

*Remark.* We introduced the definitions of both effective signature and effective nullable signature maps instead of proceeding directly with the latter as the former will be used in Chapter 4.

## 3.5 Disjoint effective nullable attributes

In section 3.3, we argued that using relation names in the base of a query is too coarse as a proxy for the presence of nulls. Instead, we noticed that marked nulls associated with different relation attributes must also be distinct. Thus, the union operation should preserve Codd semantics as long as each relation appearing in a base of at least one child contributes nulls from different attributes than the same relation in the bases of other children. To formalize this intuition, we first introduce the concept of effective nullable signature.

**Definition 8** (Effective Nullable Signature)**.** Effective nullable signature of the relation $R$ w.r.t. the query $Q$, denoted by $\mathsf{en\text{-}sig}(R, Q)$, is the set of all nullable attributes of $R$ such that for each attribute $A$ in $\mathsf{en\text{-}sig}(R, Q)$, there exists some instance of $R$ in $Q$ which can propagate null values from its attribute $A$ to the query answer. It can be computed as:

$$\mathsf{en\text{-}sig}(R, Q) = \bigcup_{R^i \in \mathsf{base}(Q)} \psi^{-1}_{R^i}(Q)\big(\mathsf{n\text{-}sig}(Q)\big)$$

Having defined the effective nullable signature, we propose a new condition for the $n$-ary union.

**Definition 9.** A node in the refined syntax tree of a query satisfies the `DJEN` (*disjoint effective nullable signature*) condition if for every pair of its children $Q_i$ and $Q_j$, it holds that:

$$\forall R \in \big(\mathsf{base}(Q_i) \cap \mathsf{base}(Q_j)\big)\ \mathsf{en\text{-}sig}(R, Q_i) \cap \mathsf{en\text{-}sig}(R, Q_j) = \varnothing \tag{4}$$

In other words, for all pairs of children, $Q_i$ and $Q_j$, we require that the effective nullable signatures w.r.t. $Q_i$ and $Q_j$ must be disjoint for all relations shared by $Q_i$ and $Q_j$.

*Remark.* To clarify, from now on, when we consider a pair of operands/children, $Q_i$ and $Q_j$, then $Q_i$ and $Q_j$ cannot represent the same operand/child. However, $Q_i$ and $Q_j$ can be the same query. For example, in the query $R \cup R$, the two children nodes $R$ of the $\cup$ node are considered distinct children even though they are the same query.

Now, since we want to use the new condition to capture strictly more queries preserving Codd semantics, we must ensure that every $n$-ary union node satisfying `PDJB` satisfies `DJEN` as well. This turns out to be the case for every node in the syntax tree of a query.

**Proposition 3.** `DJEN` *is strictly weaker than* `PDJB`.

We include the proof in Appendix B.

Satisfied with the above result, we can extend the sufficient conditions for the preservation of Codd semantics from Theorem 1 with the `DJEN` condition for the $n$-ary union:

**Theorem 3.** *Let $Q$ be an RA query whose syntax tree is such that:*

a-c) *As in Theorem 1*

d) *Each $\cup$ (binary union) node satisfies* `NNC` *or* `DJB` *or* `NNA` *or* `DJEN`.

d) *Each $\bigcup$ (variadic union) node satisfies* `DJEN` *or* `NNA`.

*Then, $Q$ preserves Codd semantics.*

**Proof.** Let $Q$ be a query whose nodes in the syntax tree satisfy at least one of the conditions required by the theorem. To prove the theorem we need to show that conditions (1a) and (1b) hold for every Codd database $D$. As before, $Q$ satisfies (1a) by Proposition 1. In order to show that $Q$ satisfies the condition (1b), we need to ensure that for every Codd database $D$, $[\![Q]\!]_D$ is a Codd table. To this end, we proceed with the proof by induction on the structure of $Q$:

**Base cases** ([5]):

- $Q$ is a relation name $R$. In such a case, $[\![Q]\!]_D$ is trivially a Codd table since $D$ is a Codd database.
- $Q$ is non-nullable. Then $[\![Q]\!]_D$ is a complete table which is also a Codd table.

**Induction**:

Here, we only extend the proof of Theorem 1 from [5] with the inductive steps for the new $n$-ary union. For the remaining operations and conditions, see the proof in [5].

First of all, note that the case when $Q = \bigcup(Q_1, \ldots, Q_n)$ satisfies the `NNA` condition is already covered in the base case as the condition implies that $Q$ is non-nullable. Secondly, the case when $Q$ is $Q_1 \cup Q_2$ and satisfies `DJEN` is covered by the proof for the case when $Q$ is $\bigcup(Q_1, \ldots, Q_n)$ (below) as $Q_1 \cup Q_2 \equiv \bigcup(Q_1, Q_2)$.

- $Q$ is $\bigcup(Q_1, \ldots, Q_n)$ and satisfies the `DJEN` condition:

By the induction hypothesis, $[\![Q_1]\!]_D$ to $[\![Q_n]\!]_D$ are Codd tables. If some marked null were to be repeated in $[\![Q]\!]_D$, it must come from two different operands as Codd tables cannot

contain duplicated nulls. Therefore, we need to show that $\mathsf{Null}([\![Q_i]\!]_D) \cap \mathsf{Null}([\![Q_j]\!]_D) = \varnothing$ for all pairs of $i, j \in [1,n], i \neq j$.

Clearly, if the same null were to appear in both $[\![Q_i]\!]_D$ and $[\![Q_j]\!]_D$, it must come from the same base relation. This is because $\mathsf{Null}([\![R]\!]_D) \cap \mathsf{Null}([\![S]\!]_D) = \varnothing$ for any two distinct relations $R$ and $S$ in a Codd database $D$. Therefore, we only need to focus on relations $R \in \big(\mathsf{base}(Q_i) \cap \mathsf{base}(Q_j)\big)$.

The set of null values in $[\![Q_i]\!]_D$ and $[\![Q_j]\!]_D$ that come from the relation $R$ can be expressed as $\mathsf{Null}([\![Q_i]\!]_D) \cap \mathsf{Null}([\![R]\!]_D)$ and $\mathsf{Null}([\![Q_j]\!]_D) \cap \mathsf{Null}([\![R]\!]_D)$, respectively. By the definition of the effective nullable signature, we know that:

$$\mathsf{Null}([\![Q_i]\!]_D) \cap \mathsf{Null}([\![R]\!]_D) \subseteq \mathsf{Null}(\pi_{\mathsf{en\text{-}sig}(R,Q_i)}([\![R]\!]_D)) \tag{5}$$

$$\mathsf{Null}([\![Q_j]\!]_D) \cap \mathsf{Null}([\![R]\!]_D) \subseteq \mathsf{Null}(\pi_{\mathsf{en\text{-}sig}(R,Q_j)}([\![R]\!]_D)) \tag{6}$$

Because the union satisfies the `DJEN` condition, we know that the nullable signatures of $R$ w.r.t. $Q_i$ and $Q_j$ are disjoint for each pair of children and for every base relation $R$ in $\mathsf{base}(Q_i) \cap \mathsf{base}(Q_j)$. Thus, we can conclude that

$$\mathsf{Null}(\pi_{\mathsf{en\text{-}sig}(R,Q_i)}([\![R]\!]_D)) \cap \mathsf{Null}(\pi_{\mathsf{en\text{-}sig}(R,Q_j)}([\![R]\!]_D)) = \varnothing \tag{7}$$

as well. This is because, by the induction hypothesis, $[\![R]\!]_D$ is a Codd table, hence nulls associated with different attributes must be distinct.

Finally, combining equations (5), (6), and (7) we can conclude that the set of nulls from any relation $R$ shared by the two operands must be empty:

$$\begin{aligned}
&\big(\mathsf{Null}([\![Q_i]\!]_D) \cap \mathsf{Null}([\![Q_j]\!]_D)\big) \cap \mathsf{Null}([\![R]\!]_D) = \\
&= \big(\mathsf{Null}([\![Q_i]\!]_D) \cap \mathsf{Null}([\![R]\!]_D)\big) \cap \big(\mathsf{Null}([\![Q_j]\!]_D) \cap \mathsf{Null}([\![R]\!]_D)\big) \\
&\subseteq \mathsf{Null}(\pi_{\mathsf{en\text{-}sig}(R,Q_i)}([\![R]\!]_D)) \cap \mathsf{Null}(\pi_{\mathsf{en\text{-}sig}(R,Q_j)}([\![R]\!]_D)) && \text{(by eq. 5 and 6)} \\
&= \varnothing && \text{(by eq. 7)}
\end{aligned}$$

This proves that no two operands can share a null, so the $n$-ary union must preserve Codd semantics.

□

The importance of the above theorem lies in the fact that the conditions of Theorem 3 enable us to capture strictly more RA queries that preserve Codd semantics than the condition from Theorem 2. This is because the requirements of Theorem 3 are satisfied whenever the requirements of Theorem 2 are met as the `DJEN` condition is weaker than `PDJB`. At the same time, the converse is not necessarily true as demonstrated in the following example.

**Example 4.** Let our query be:

$$\bigcup(\sigma_{\mathsf{const}(B)}(\rho_{A\to C}(R)), \sigma_{\mathsf{const}(B)}(\rho_{A\to B}(\rho_{B\to C}(R))), \rho_{A\to C}(S-R))$$

Its annotated syntax tree is presented in Figure 8. The query references two base relations $R$ and $S$ over attributes $A, B$. Let us assume that both $A$ and $B$ are nullable in

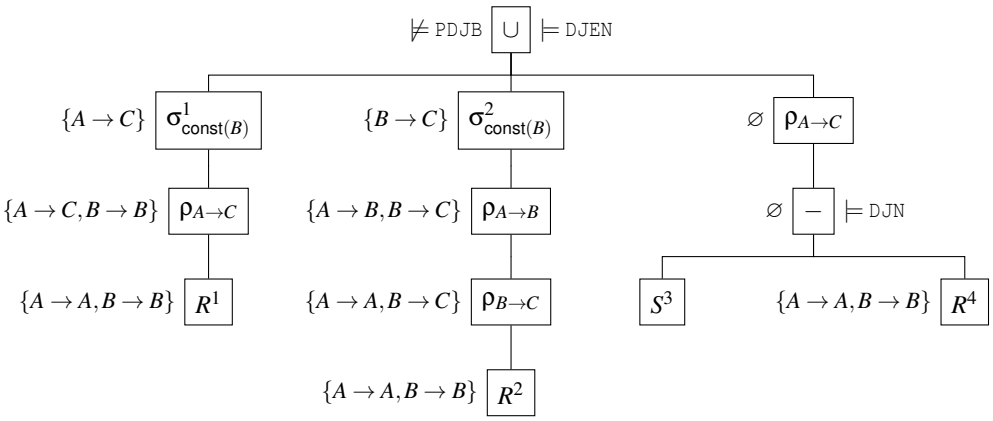



Figure 8: Analysis of the propagation of nulls in the query from Example 4.

$R$ and that the relation $S$ is non-nullable (so that the $-$ node satisfies DJN). We are not able to capture the preservation of Codd semantics of the $\bigcup$ node with the help of the PDJB condition alone since the relation $R$ appears in two nullable children ($\sigma^1$ and $\sigma^2$).

However, using DJEN, we can allow the null values to propagate from the same base relation in two different operands as long as the null values come from different attributes of the relation. Analysing the propagation of null values from instances of $R$ to $\sigma^1$, $\sigma^2$ and $-$ nodes (annotations to the left of nodes in Figure 8), we can observe that effective nullable signatures of $R$ with respect to these nodes are $\{A\}$, $\{B\}$, and $\varnothing$, respectively. For $\sigma^1$, the attribute $B$ is removed from the en-sig by the selection operation. For $\sigma^2$, the selection filters out nulls from attribute $B$ as well, but at this stage, the values in $B$ come originally from the attribute $A$ in $R$ (represented by the $A \to B$ mapping). Hence, only the attribute $B$ of $R$ is in the effective nullable signature. Finally, no nulls from $R$ can be found in the output of the $-$ node, so its effective nullable signature is empty. Clearly, the three sets are disjoint, so there is no risk of creating repeated nulls. The DJEN condition is satisfied and the union can be correctly recognised as Codd semantics preserving.

*Remark.* As a result of the previous theorem, we now have four sufficient conditions for the preservation of Codd semantics by the binary union operation. That is NNA, DJB, NNC, and DJEN. However, it should be recognised that DJB and NNC are special cases of DJEN (as shown formally in the proof of Proposition 10 in Appendix B). On the other hand, a node can satisfy DJEN without satisfying NNC or DJB as is the case in Example 2. Consequently, DJEN is weaker than the disjunction of NNC and DJB and it enables us to capture more queries that preserve Codd semantics even in the case of a single binary union. This, however, should not be surprising as we previously said that the disjunction of NNC and DJB is equivalent to PDJB (in the binary case), which was shown to be stricter than DJEN. Therefore, we can simplify the condition for binary union nodes in Theorem 3 to the form:

d) Each $\cup$ (binary union) node satisfies DJEN or NNA.

## 3.6 Redundancy of variadic union

In the previous section, we introduced the new DJEN condition for union nodes. We also demonstrated that the DJEN condition enables us to capture strictly more queries preserving Codd semantics. Now, we will show how it can be used to identify chains of unions preserving Codd semantics regardless of their exact formulation.

With the PDJB condition, we relied on Proposition 2. It guaranteed that we can safely rewrite chains of unions using equivalent $n$-ary unions, which would then satisfy PDJB whenever any equivalent formulation satisfied the premises of Theorem 2. This time, it turns out we can completely avoid replacing chains of unions with variadic unions. Namely, we show that it is enough to check whether each union node in the query, as given, satisfies DJEN. To this end, we first propose the following claim.

**Proposition 4.** *Let $Q_1$ to $Q_n$ be RA queries with the same signature. Then the following statements are equivalent:*

*(a)* $\bigcup(Q_1, \ldots, Q_n)$ *satisfies* DJEN.

*(b) In all chains of binary unions that combine queries $Q_1$ to $Q_n$, every $\cup$ node satisfies* DJEN.

*(c) There exists a chain of binary unions that combines queries $Q_1$ to $Q_n$ in which every $\cup$ node satisfies* DJEN.

**Proof.** We prove the equivalence of statements using the circular chain of implications.

- $(a) \rightarrow (b)$: Let us assume that $\bigcup(Q_1, \ldots, Q_n)$ satisfies DJEN. Under this assumption, we will show that every $\cup$ node in all chains of binary unions that combine queries $Q_1$ to $Q_n$ satisfies DJEN.

  For that, let $P_l$ and $P_r$ be two disjoint non-empty subsets of $\{Q_1, \ldots, Q_n\}$. Let $Q_l$ and $Q_r$ be some chains of unions merging together queries from $P_l$ and $P_r$, respectively. Also, let $\cup^*$ be the union node in the query $Q_l \cup^* Q_r$. Since sets $P_l$ and $P_r$ can be chosen arbitrarily, the same as the structure of $Q_l$ and $Q_r$, the $\cup^*$ node effectively represents all possible union nodes in all chains of unions that are equivalent to $\bigcup(Q_1, \ldots, Q_n)$.

  Now, the $\cup^*$ node satisfies the DJEN condition if for all relations $R$ that appear both in the base of $Q_l$ and in the base of $Q_r$, it holds that:

$$\text{en-sig}(R, Q_l) \cap \text{en-sig}(R, Q_r) = \varnothing$$

  We can rewrite the above equation using the fact that $Q_l$ and $Q_r$ are just chains of unions and therefore the effective nullable signatures w.r.t. $Q_l$ and $Q_r$ are made up of attributes from the nullable signatures w.r.t. the operands they combine:

$$\Big( \bigcup_{Q_i \in P_l} \text{en-sig}(R, Q_i) \Big) \cap \Big( \bigcup_{Q_j \in P_r} \text{en-sig}(R, Q_j) \Big) = \varnothing$$

  or equivalently:

$$\bigcup_{Q_i \in P_l} \bigcup_{Q_j \in P_r} \Big( \text{en-sig}(R, Q_i) \cap \text{en-sig}(R, Q_j) \Big) = \varnothing$$

Now, this is indeed the case as, by our assumption, $\bigcup(Q_1,\ldots,Q_n)$ satisfies the DJEN condition, so $\textsf{en-sig}(R,Q_i)\cap\textsf{en-sig}(R,Q_j)=\varnothing$ for all pairs of operands $Q_i,Q_j\in\{Q_1,\ldots,Q_n\}$ and for all $R\in\textsf{base}(Q_i)\cap\textsf{base}(Q_j)$:

$$\bigcup_{Q_i\in P_l}\bigcup_{Q_j\in P_r}\Big(\textsf{en-sig}(R,Q_i)\cap\textsf{en-sig}(R,Q_j)\Big)=\bigcup_{Q_i\in P_l}\bigcup_{Q_j\in P_r}(\varnothing)=\varnothing$$

Thus, $\cup^*$ satisfies DJEN and so do all $\cup$ nodes in all possible chains of unions.

- $(b)\rightarrow(c)$: The conclusion follows trivially from the premise because there always exists at least one chain of binary unions that combines queries $Q_1$ to $Q_n$.
- $(c)\rightarrow(a)$: Let us assume that there exists a chain of binary unions that combines queries $Q_1$ to $Q_n$ in which every $\cup$ node satisfies DJEN. Under this assumption, we will show that $\bigcup(Q_1,\ldots,Q_n)$ satisfies DJEN as well.

  To this end, consider a chain that satisfies our initial assumption. Then, for any pair of operands $Q_i$ and $Q_j$, where $i,j\in[1,n]$, let $\cup^*$ be their lowest common ancestor node in the chain of unions. Without loss of generality, we can assume that $Q_i$ and $Q_j$ are subqueries of $\cup^*$'s left and right children, $Q_l$ and $Q_r$, respectively.

  Using the fact that $Q_l$ must be $Q_i$ itself or a chain of unions containing $Q_i$, we know that $\textsf{en-sig}(R,Q_i)\subseteq\textsf{en-sig}(R,Q_l)$ for all relations $R$ that appear in the base of $Q_i$. Similarly for $Q_r$ and $Q_j$. Also, as $\cup^*$ satisfies DJEN by our assumption, we know that: $\textsf{en-sig}(R,Q_l)\cap\textsf{en-sig}(R,Q_r)=\varnothing$. Combining all of these observations together, we can conclude that for each pair of queries $Q_i$ and $Q_j$ and for all relations $R\in(\textsf{base}(Q_i)\cap\textsf{base}(Q_j))$, it holds that:

  $$\textsf{en-sig}(R,Q_i)\cap\textsf{en-sig}(R,Q_j)\subseteq\textsf{en-sig}(R,Q_l)\cap\textsf{en-sig}(R,Q_r)=\varnothing$$

  which is exactly what is required by the DJEN condition for $\bigcup(Q_1,\ldots,Q_n)$.

□

Having proved the equivalence of the statements from Proposition 4, we can state the main result of this section.

**Corollary 1.** *Let $Q_1$ to $Q_n$ be RA queries with the same signature. Then the following statements are equivalent:*

*a)* $\bigcup(Q_1,\ldots,Q_n)$ *does not satisfy* DJEN

*b) there exists a chain of binary unions that combines queries $Q_1$ to $Q_n$ in which some $\cup$ node does not satisfy* DJEN.

*c) there does not exist a chain of binary unions that combines queries $Q_1$ to $Q_n$ in which every $\cup$ node satisfies* DJEN.

**Proof.** Each statement is the negation of the corresponding statement from Propositions 4, therefore they must be equivalent too. □

The significance of the above corollary is the following. Firstly, by Theorem 3, if all $\cup$ nodes in a chain of binary unions satisfy DJEN, then we know that it preserves Codd

semantics. Otherwise, it means that we have found a chain of binary unions in which some $\cup$ node does not satisfy DJEN. As a consequence, by Corollary 1, we can be certain that no other formulation of the chain nor the equivalent $n$-ary union satisfies the conditions of Theorem 3. This is a very strong result as before we were not able to discard this possibility. Therefore, we can limit ourselves to verifying the preservation of the Codd semantics directly on the chains of unions as formulated in the query.

*Remark.* Technically, an equivalent formulation could be also identified as Codd semantics preserving by satisfying the NNA condition. Although, it can be easily shown that in a chain of unions satisfying the conditions of Theorem 3 either all union nodes satisfy DJEN or all satisfy NNA. In the latter case, all union nodes in the original formulation would satisfy NNA as well, so there is no need to check other formulations.

As a consequence, it turns out that we do not need the variadic union at all. This is because any variadic union can be represented using solely binary unions and all the checks can be performed on the binary unions directly. Therefore, for the remainder of the report, we will consider only RA queries without the variadic union. This leads us to the final version of the conditions for the preservation of Codd semantics.

**Theorem 4.** *Let Q be an RA query whose syntax tree is such that:*

a) *each* $\varepsilon$ *node satisfies* NNC*;*

b) *each* $\cap$ *and* $-$ *node satisfies* DJN*;*

c) *each* $\times$ *node satisfies* NNA*;*

d) *each* $\cup$ *node satisfies* NNA *or* DJEN*.*

*Then, Q preserves Codd semantics.*

# Chapter 4

# Preserving Duplicated Marked Nulls

## 4.1 Redefining Codd semantics

So far we have defined the Codd interpretation of an SQL database $D$, i.e. $\mathsf{codd}(D)$, as a result of replacing each SQL null in $D$ with a fresh marked null. Under such semantics, no marked null can occur twice in a database nor in an answer to a query. Now, we would like to relax this constraint and allow duplicated labelled nulls in tables. Ideally, in the extreme case, we would like to be able to replace SQL nulls in $D$ with any marked null. Unfortunately, such an interpretation introduces another degree of non-determinism to the null replacement procedure. As a result, the codd function could produce a set of databases that are no longer all isomorphic to each other. For example, the SQL table on the left (below) could be interpreted as each of the tables to its right.

| $A$ | $B$ |
|---|---|
| 1 | **N** |
| 2 | **N** |
| 2 | **N** |
| 2 | **N** |

| $A$ | $B$ |
|---|---|
| 1 | $\perp_1$ |
| 2 | $\perp_2$ |
| 2 | $\perp_3$ |
| 2 | $\perp_4$ |

| $A$ | $B$ |
|---|---|
| 1 | $\perp_1$ |
| 2 | $\perp_2$ |
| 2 | $\perp_2$ |
| 2 | $\perp_3$ |

| $A$ | $B$ |
|---|---|
| 1 | $\perp_1$ |
| 2 | $\perp_2$ |
| 2 | $\perp_2$ |
| 2 | $\perp_2$ |

This poses multiple problems. The biggest one is the fact that even if a query $Q$ preserves the semantics of nulls under such interpretation, it might not be immediately clear which replacement of nulls the answer represents. In simple terms, we cannot talk anymore about the single interpretation of an SQL table. This makes the semantics of the query's output ambiguous which is undesirable in real-life systems. Therefore, we focus on the interpretation of SQL nulls in which **N** values are still replaced by fresh marked nulls but this time all records that are repeated within the SQL table are associated with the same marked nulls. This is to say that the multiplicity of each record in a table after replacing SQL nulls with marked nulls is preserved by the transformation.

*Remark.* The newly introduced interpretation of SQL nulls is obviously different from the original Codd interpretation. However in this chapter, for consistency with the notation and nomenclature introduced before, we will keep using the terms Codd semantics, Codd table, Codd database, codd function, etc., when referring to the

corresponding objects under the new interpretation. In particular, from now on we will call:

- Codd semantics - the interpretation of nulls in which records that are repeated in a table are assigned the same marked null,
- a Codd table - a naive table in which each marked null is different except in repeated records; similarly for the Codd database,
- the codd function - a function that assigns fresh marked nulls to distinct records and the same nulls to records that are repeated in a table,

Moreover, whenever we compare the two interpretations, we will clearly distinguish between them in order to avoid any confusion.

Coming back, an example of the new Codd interpretation is presented below:

SQL:

| $A$ | $B$ | $C$ |
|---|---|---|
| 1 | **N** | **N** |
| 2 | **N** | **N** |
| 2 | **N** | **N** |

Codd:

| $A$ | $B$ | $C$ |
|---|---|---|
| 1 | $\perp_1$ | $\perp_2$ |
| 2 | $\perp_3$ | $\perp_4$ |
| 2 | $\perp_3$ | $\perp_4$ |

This approach yields a codd procedure that produces a set of databases which are all isomorphic to each other. Hence, once again, we can talk about the unambiguous Codd interpretation of SQL database/table that is unique up to the renaming of nulls.

While the definition of the preservation of the semantics of nulls remains intact, it is no longer the case that a query $Q$ preserves the new Codd semantics if conditions (1a) and (1b) from Definition 2 hold. This is demonstrated in the following example.

**Example 5.** Consider the query $Q = R \cup S$ evaluated on a Codd database $D$ where:

$[\![R]\!]_D =$

| $A$ | $B$ |
|---|---|
| 1 | $\perp_1$ |

;

$[\![S]\!]_D =$

| $A$ | $B$ |
|---|---|
| 1 | $\perp_2$ |

In such case, the answers to the query $Q$ on databases $D$ and $\mathsf{sql}(D)$ are:

$[\![Q]\!]_D =$

| $A$ | $B$ |
|---|---|
| 1 | $\perp_1$ |
| 1 | $\perp_2$ |

;

$[\![Q]\!]_{\mathsf{sql}(D)} =$

| $A$ | $B$ |
|---|---|
| 1 | **N** |
| 1 | **N** |

Clearly, $\mathsf{sql}([\![Q]\!]_D) = [\![Q]\!]_{\mathsf{sql}(D)}$ as well as $[\![Q]\!]_D$ is a Codd table. However, $[\![Q]\!]_D$ is not a valid Codd interpretation of $[\![Q]\!]_{\mathsf{sql}(D)}$ as it would need to have two identical records, whereas the query evaluated on Codd database produces the table with two distinct records.

Most importantly, notice that in the new Codd semantics not every Codd database/table is a Codd interpretation of some SQL database/table (as was the case in the original Codd semantics). Therefore, we need an additional condition for the preservation of the new Codd semantics that captures the fact that the modified codd function must keep multiplicities of translated records the same. We achieve that by requiring that no two records in $[\![Q]\!]_D$ differ only in null values. This condition prevents two records in $[\![Q]\!]_D$ to be transformed into the same record in $\mathsf{sql}([\![Q]\!]_D)$. Consequently, the multiplicity

of a record $r$ in $[\![Q]\!]_D$ must be the same as the multiplicity of a corresponding record $\mathsf{sql}(r)$ in $\mathsf{sql}([\![Q]\!]_D)$ and vice-versa. We formalise the above discourse in the following proposition.

**Proposition 5.** *Let $T$ be a table and let $T' = \mathsf{sql}(T)$. Then, all the following statements are equivalent:*

*(a) No two records in $T$ differ only in nulls.*

*(b) For every record $r$ in $T$, $\#(r,T) = \#(\mathsf{sql}(r), T')$.*

*(c) For every record $r'$ in $T'$, there exists a unique record $r$ in $T$ such that $r \in \mathsf{sql}^{-1}(r')$ and $\#(r,T) = \#(r',T')$.*

The proof of the above proposition is given in Appendix B.

As a result, the definition of the preservation of Codd semantics, using the new codd function, can be equivalently formulated as:

**Definition 10.** A query $Q$ preserves the new Codd semantics if for every SQL database $D'$ it holds that $[\![Q]\!]_D \in \mathsf{codd}\left([\![Q]\!]_{D'}\right)$, where $D$ is the Codd interpretation of $D'$, which can be equivalently formulated as:

$$\mathsf{sql}([\![Q]\!]_D) = [\![Q]\!]_{\mathsf{sql}(D)}, \tag{2a}$$

$$[\![Q]\!]_D \text{ is a Codd table,} \tag{2b}$$

$$\text{and no two records in } [\![Q]\!]_D \text{ differ only in null values.} \tag{2c}$$

## 4.2 Sufficient conditions with new requirement

Having identified the new requirement (2c), we need to consider which RA operations can be affected by it. Clearly, renaming just changes the signature of a table so it cannot interfere with (2c). Intersection, difference, selection, and duplicate elimination produce a table that is always contained in one of input tables. So, the output can contain two records differing only in null values only if the input tables contain such records too. Similarly, records produced by the Cartesian product can differ only in null values only if there are at least two such records in some input table as well - see the formal argument in the proof of Proposition 6.

Therefore, the only problematic operations with respect to the new condition are projection and union. The former can produce two records differing only in null values when it removes all the attributes that differentiated the two records on the constant values in the input table. We demonstrate this scenario in the following example.

**Example 6.** Consider the query $Q = \pi_B(R)$ evaluated on a Codd database $D$

where $[\![R]\!]_D =$

| $A$ | $B$ |
|---|---|
| 1 | $\perp_1$ |
| 2 | $\perp_2$ |

. The answer to $Q$ on $D$ is $[\![Q]\!]_D =$

| $B$ |
|---|
| $\perp_1$ |
| $\perp_2$ |

.

Clearly, the two records in $[\![Q]\!]_D$ differ only in nulls ($\perp_1$ and $\perp_2$) even though $[\![R]\!]_D$ satisfies the condition (2c). This is because the attribute $A$ that differentiated the two records on constant values was removed by the projection.

Nevertheless, we can still allow such a situation to happen as long as such records will be eventually removed from the output, which can be ensured by the NNA condition.

On the other hand, the union may not satisfy the requirement (2c) when two records differing only in nulls come from two separate children. This is because the two records are then put together in one output table (as we saw in Example 5). There is a number of ways we can prevent that from happening. First, we could require the NNA condition and ignore the problem since the problematic records will be eventually discarded. Alternatively, we can require the NNC condition which ensures that at least one child is a complete table, hence it cannot contain any nullable records. Following similar reasoning, we can also require the DJN condition. The intuition is that if two records do not share nullable attributes then they cannot differ only in nulls. This is because one record is guaranteed to have a constant value for the attribute that is nullable in the other. Putting all the above observations together we can formulate the following proposition.

**Proposition 6.** *Every RA query whose syntax tree is such that all $\pi_\alpha$ nodes satisfy the* NNA *condition and every $\cup$ node satisfies* NNA *or* DJN *or* NNC *conditions satisfies* (2c) *on all Codd interpretations of SQL databases.*

See the proof in Appendix B.

Furthermore, we can utilise the new property of Codd semantics allowing marked nulls to be duplicated among repeated records to come up with a new sufficient condition which ensures that the result of the union operation is a Codd table.

**Definition 11.** A node $Q$ in the syntax tree of a query satisfies the SESM (*same effective signature map*) condition if for every relation $R \in \big(\mathsf{base}(Q_1) \cap \mathsf{base}(Q_2)\big)$, where $Q_1$ and $Q_2$ are children of $Q$, the following statement holds:

For all $R^i, R^j \in \mathsf{base}(Q)$, if $\phi_{R^i}(Q)$ and $\phi_{R^j}(Q)$ are total and surjective functions (effective signature maps) from $\mathsf{sig}(R)$ to $\mathsf{sig}(Q)$, then $\phi_{R^i}(Q) = \phi_{R^j}(Q)$.

The condition is based on the fact that if a relation instance contributes values to the output of a query from a record which is either restricted by an intermediate projection or extended by an intermediate Cartesian product, then the propagated values must be constant. This is because if $\pi$ and $\times$ nodes satisfy the NNA condition, then the nullable records they produce must be removed from the output at some point. We formalise this observation in the following lemma.

**Lemma 2.** *Let $Q$ be a RA query whose syntax tree is such that all $\pi_\alpha$ and $\times$ nodes satisfy the* NNA *condition. An instance of a relation $R^i$ can propagate null values to the query output only if $\phi_{R^i}(Q)$ is a total and surjective function from $\mathsf{sig}(R)$ to $\mathsf{sig}(Q)$.*

See the proof in Appendix B.

For that reason, only records which are neither truncated nor extended can potentially contribute null values to the output. Hence, by ensuring that the same attributes from all instances of a base relation are mapped to the same attributes in the output table,

we are able to guarantee that propagated records have the same "shape" and only the same records contain the same marked nulls. For the formal argument see the proof of Theorem 5.

Taking into account all the aforementioned findings, we now present the conditions for the preservation of the Codd semantics with the new codd function.

**Theorem 5.** *Let $Q$ be an RA query whose syntax tree is such that:*

a) *each* $\varepsilon$ *node satisfies* `NNC`*;*

b) *each* $\cap$ *and* $-$ *node satisfies* `DJN`*;*

c) *each* $\times$ *and* $\pi_\alpha$ *node satisfies* `NNA`*;*

d) *each* $\cup$ *node satisfies (*`NNA` *or* `DJN` *or* `NNC`*) AND (*`NNA` *or* `DJEN` *or* `SESM`*).*

*Then, $Q$ preserves the new Codd semantics.*

**Proof.** Let $Q$ be an RA query whose syntax tree satisfies the conditions of Theorem 5. To prove the theorem we need to show that conditions (2a), (2b), and (2c) hold for every Codd database $D$. Clearly, $Q$ satisfies condition (2a) by Proposition 1 as it guarantees that $\mathsf{sql}(\llbracket Q \rrbracket_D) = \llbracket Q \rrbracket_{\mathsf{sql}(D)}$ holds for all databases $D$. Moreover, $Q$ satisfies condition (2c) by Proposition 6.

Therefore, we only need to show that $Q$ satisfies the condition (2b). We prove it by the induction on the structure of $Q$.

**Base cases** ([5]):

- $Q$ is a relation name $R$. In such a case, $\llbracket Q \rrbracket_D$ is trivially a Codd table since $D$ is a Codd database.
- $Q$ is non-nullable. Then $\llbracket Q \rrbracket_D$ is a complete table which is also a Codd table.

**Induction**:

First of all, note that the case when $Q$ is $\varepsilon(Q_1)$ and satisfies `NNC` is already covered in the base case as the condition implies that $Q$ is non-nullable. Similarly when $Q$ is $Q_1 \cap Q_2$ or $Q_1 - Q_2$ and satisfies `DJN` and when $Q$ is $Q_1 \times Q_2$ or $Q_1 \cup Q_2$ or $\pi_\alpha(Q_1)$ and satisfies `NNA`.

- $Q$ is $\sigma(Q_1)$:

The proof follows from the fact that $\llbracket Q \rrbracket_D \subseteq \llbracket Q_1 \rrbracket_D$ and, by the induction hypothesis, $\llbracket Q_1 \rrbracket_D$ is a Codd table.

- $Q$ is $\rho_{A \to B}(Q_1)$:

The claim trivially follows from the fact that $\rho_{A \to B}$ only changes the name of attribute A in each record in $\llbracket Q_1 \rrbracket_D$, which is a Codd table by the induction hypothesis, but not the value this attribute is mapped to.

- $Q$ is $Q_1 \cup Q_2$ and satisfies `DJEN`:

By the induction hypothesis, $[\![Q_1]\!]_D$ and $[\![Q_2]\!]_D$ must be Codd tables. Then, $\varepsilon([\![Q_1]\!]_D)$ and $\varepsilon([\![Q_2]\!]_D)$ are Codd tables in the original sense (no repeated nulls). In the proof of Theorem 3, we show that if the input tables are Codd tables (in the original sense) and the union satisfies `DJEN`, then the output table is also a Codd table (in the original sense). Hence, we know that $\varepsilon([\![Q_1]\!]_D) \cup \varepsilon([\![Q_2]\!]_D)$ is a Codd table (in the original sense). Furthermore, it is the case that $\varepsilon([\![Q_1]\!]_D \cup [\![Q_2]\!]_D) \subseteq \varepsilon([\![Q_1]\!]_D) \cup \varepsilon([\![Q_2]\!]_D)$. Therefore, $\varepsilon([\![Q_1]\!]_D \cup [\![Q_2]\!]_D)$ must also be a Codd table (in the original sense) and $[\![Q_1]\!]_D \cup [\![Q_2]\!]_D$ must be a Codd table as defined in this chapter (duplicated nulls only among repeated records). Thus, $[\![Q]\!]_D$ is also a Codd table.

- $Q$ is $Q_1 \cup Q_2$ and satisfies `SESM`:

By the induction hypothesis, $[\![Q_1]\!]_D$ and $[\![Q_2]\!]_D$ are Codd tables. Therefore, if we show that the nulls they share are repeated only among identical records in both children, then we can conclude that $[\![Q_1 \cup Q_2]\!]_D$ is also a Codd table. To this end, we will show that there cannot exist two distinct records $r_1 \in [\![Q_1]\!]_D$ and $r_2 \in [\![Q_2]\!]_D$ that contain the same null. That is, $r_1$ and $r_2$ such that $r_1 \neq r_2$ and $r_1(A) = r_2(B) = \bot \in \mathsf{Null}$, for some attributes $A, B \in \mathsf{sig}(Q)$. As a consequence, the output table cannot contain two different records with the same marked null, which proves that $[\![Q]\!]_D$ is a Codd table.

*Remark.* Note that if $r_1 = r_2$, then $\#(r_1, [\![Q_1 \cup Q_2]\!]_D) = \#(r_1, [\![Q_1]\!]_D) + \#(r_2, [\![Q_2]\!]_D)$. Such records do not cause problems with the condition (2b) as the nulls are duplicated in the output only among identical records. Therefore, only the case when $r_1 \neq r_2$ needs to be considered.

To begin with, let us notice that $\bot$ must come from the same base relation in both children. This is because $\mathsf{Null}([\![R]\!]_D) \cap \mathsf{Null}([\![S]\!]_D) = \varnothing$ for any two different relations $R$ and $S$. Thus, we can only focus on nulls that come from some relation $R$ that is in the base of both $Q_1$ and $Q_2$.

Next, we observe that all instances of a base relation $R$ in the base of $Q$ can be partitioned into two sets based on whether $\phi_{R^i}(Q)$ is a total and surjective function from $\mathsf{sig}(R)$ to $\mathsf{sig}(Q)$ or not. By Lemma 2, the instances belonging to the latter category cannot propagate null values to the query output. Summing up, only records from instances of the same base relation that have total and surjective maps w.r.t. $Q$ can cause problems with the preservation of Codd semantics. In the remaining part of the proof, we show that if the condition `SESM` is satisfied, then all records $r_1 \in [\![Q_1]\!]_D$ and $r_2 \in [\![Q_2]\!]_D$ such that $r_1(A) = r_2(B) = \bot \in \mathsf{Null}$ must be the same and, therefore, $[\![Q_1 \cup Q_2]\!]_D$ must be a Codd table.

We prove the statement by contradiction. Let us assume that $r_1$ and $r_2$ are indeed distinct and that originally they come from $R^i \in \mathsf{base}(Q_1)$ and $R^j \in \mathsf{base}(Q_2)$, respectively. For that reason, let $f_1 = \phi_{R^i}(Q_1) = \phi_{R^i}(Q)$ and $f_2 = \phi_{R^j}(Q_2) = \phi_{R^j}(Q)$. By our assumption, $f_1$ and $f_2$ are total and surjective, which means that records from $R^i$ and $R^j$ are not truncated nor extended by projection or Cartesian product operations. Therefore, it must be the case that $r_1 \in \rho_{f_1}([\![R]\!]_D)$ and $r_2 \in \rho_{f_2}([\![R]\!]_D)$, where $\rho_{f_1}$ and $\rho_{f_2}$ is an abuse of notation for the renaming operation that applies all mappings from the effective signature map to its input. Moreover, since the $\cup$ node satisfies the `SESM` condition, we know that $f_1 = f_2$, so $\rho_{f_1}([\![R]\!]_D) = \rho_{f_2}([\![R]\!]_D)$. As a result, both $r_1$ and $r_2$ must come

from the same table $\rho_{f_1}(\llbracket R \rrbracket_D)$. It is important to observe at this point that $\rho_{f_1}(\llbracket R \rrbracket_D)$ must be a Codd table because, by the induction hypothesis, $\llbracket R \rrbracket_D$ is a Codd table. But this leads to a contradiction since we just showed that $\rho_{f_1}(\llbracket R \rrbracket_D)$ contains two distinct records $r_1$ and $r_2$ which share the same marked null $\perp$. As a consequence, such two distinct $r_1$ and $r_2$ cannot exist, which completes the proof.

□

The last thing that we would like to notice here is that the condition for the union can be equivalently stated as: each $\cup$ node satisfies NNA or NNC or (DJN and DJEN) or (DJN and SESM). This is because NNC is a special case of DJEN. However, it turns out that this set of clauses is not minimal due to the following proposition.

**Proposition 7.** *Let $Q$ be an RA query whose syntax tree is such that every $\pi_\alpha$ and $\times$ node satisfies the* NNA *condition. Then, every $\cup$ node in $Q$ that does not satisfy* NNA *but satisfies* DJN *and* SESM *must also satisfy* DJEN*.*

See the proof in Appendix B.

As a consequence, we know that if a $\cup$ node satisfies NNA, then it satisfies the conditions of Theorem 5. Otherwise, if $\cup$ does not satisfy NNA, then it cannot satisfy DJN and SESM without satisfying DJEN. Hence, we can safely rewrite the clause (DJN and SESM) as (DJN and SESM and DJEN) which in turn subsumes (DJN and DJEN). Thus, the minimal formulation of the condition for union nodes is:

d) *each $\cup$ node satisfies* NNA *or* NNC *or (*DJN *and* DJEN*)*

## 4.3 Comparison of conditions for different null interpretations

So far, we have analysed and refined the sufficient conditions for the preservation of null semantics for two interpretations of an SQL table/database. That is, when codd function assigns:

1. a fresh marked null to each SQL null - original Codd semantics;
2. a fresh marked null to each SQL null in different records, but repeats the same marked nulls in the same records within a table - Codd semantics as defined in this chapter

The conditions for the preservation of these semantics are captured by Theorems 4 and 5, respectively. The key differences between the two are the following:

- for interpretation 2, all $\pi_\alpha$ nodes have to satisfy the NNA condition;
- for interpretation 2, if NNA is not satisfied, then $\cup$ nodes cannot satisfy DJEN on its own (as it is for interpretation 1) and are required to satisfy NNC or (DJN and DJEN).

Furthermore, observe that DJEN is subsumed by both NNC and (DJN $\wedge$ DJEN) conditions. This is because DJEN needs to be satisfied for (DJN $\wedge$ DJEN) to be satisfied, while NNC

is a special case of DJEN. Therefore, due to stricter conditions for $\pi_\alpha$ and $\cup$ nodes for interpretation 2, we can state the following proposition.

**Proposition 8.** *The conditions of Theorem 4 are strictly weaker than the conditions of Theorem 5.*

To demonstrate this, in the next examples, we present two queries which preserve interpretation 1 but not interpretation 2.

**Example 7.** Let $Q$ be the query whose annotated syntax tree is presented in Figure 9. Also, let relation $R$ be over nullable attributes $A$ and $B$. The annotations to the left of a node represent the effective nullable signature map of the instance of $R$ that is present in the subquery with respect to the given node. Based on these, we can see that the instance $R^1$ can contribute nulls to the answer of the query only from the attribute $A$, while $R^2$ only from the attribute $B$. Therefore, clearly, $\cup$ satisfies DJEN. On the other hand, $\mathsf{n\text{-}sig}(\sigma^1) = \mathsf{n\text{-}sig}(\sigma^2) = \{C\}$, so DJN is not satisfied. For the same reason, neither $\sigma^1$ nor $\sigma^2$ is non-nullable so the $\cup$ node does not satisfy NNC nor NNA. Thus, the conditions of Theorem 4 are satisfied, while those of Theorem 5 are not.

*Remark.* Note that in this example the $\cup$ node does not satisfy SESM as well. The effective signature maps of $R^1$ and $R^2$ with respect to the $\cup$ node are $\phi_{R^1}(Q) = \{A \to C, B \to B\}$ and $\phi_{R^2}(Q) = \{A \to B, B \to C\}$, respectively. Clearly, both of them are total and surjective functions from $\mathsf{sig}(R)$ to $\mathsf{sig}(Q)$, but they are not equal to each other, so SESM is not satisfied.

**Example 8.** Consider the query $Q = \pi_A(R)$. Clearly, $Q$ preserves the original Codd semantics on all SQL databases for which $Q$ is well-defined as Theorem 4 does not impose any restrictions on projections. On the other hand, $Q$ satisfies the conditions of Theorem 5 if and only if attribute $A$ of $R$ is marked as **NOT NULL**.

## 4.4 Evaluation under set semantics

Whilst the two interpretations of nulls differ when queries are evaluated using bag semantics, they turn out to be equivalent when applied under set semantics. This is

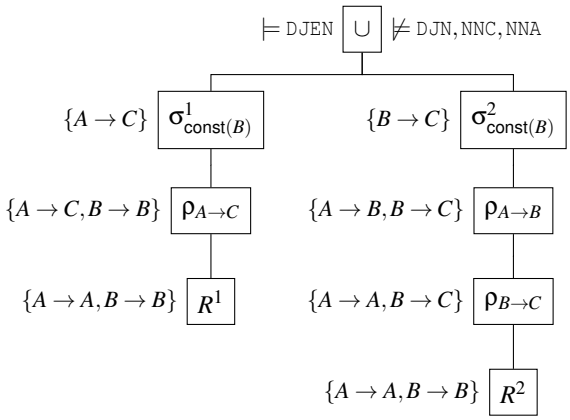



Figure 9: Analysis of the propagation of nulls in the query from Example 7.

because, by definition, set tables do not contain any duplicated records. Therefore, the new codd function will always replace every SQL null with a fresh marked null. For that reason, it is impossible to talk about the conditions for two different interpretations. Instead, we can use the new conditions for bags to devise new conditions for the preservation of Codd semantics over sets.

For the purpose of this analysis, let $\mathrm{RA}^{\mathrm{set}}$ be a fragment of RA without $\varepsilon$. Moreover, we define a *set database* to be a database where each record appears only once in a table. Then, the *set semantics* of an $\mathrm{RA}^{\mathrm{set}}$ query $Q$ on the set database $D$, denoted by $[\![Q]\!]^{\mathrm{set}}_D$ is recursively defined in the following manner:

$$
\begin{aligned}
[\![R]\!]^{\mathrm{set}}_D &\stackrel{\mathrm{def}}{=\!=} [\![R]\!]_D \\
[\![Q_1 \,\mathsf{op}\, Q_2]\!]^{\mathrm{set}}_D &\stackrel{\mathrm{def}}{=\!=} [\![Q_1]\!]^{\mathrm{set}}_D \,\mathsf{op}\, [\![Q_2]\!]^{\mathrm{set}}_D \text{ for } \mathsf{op} \in \{\cap, -, \times\} \\
[\![Q_1 \cup Q_2]\!]^{\mathrm{set}}_D &\stackrel{\mathrm{def}}{=\!=} \varepsilon\big([\![Q_1]\!]^{\mathrm{set}}_D \cup [\![Q_2]\!]^{\mathrm{set}}_D\big) \\
[\![\pi_\alpha(Q')]\!]^{\mathrm{set}}_D &\stackrel{\mathrm{def}}{=\!=} \varepsilon\big(\pi_\alpha([\![Q']\!]^{\mathrm{set}}_D)\big) \\
[\![\rho_{A\to B}(Q')]\!]^{\mathrm{set}}_D &\stackrel{\mathrm{def}}{=\!=} \rho_{A\to B}([\![Q']\!]^{\mathrm{set}}_D) \\
[\![\sigma_\theta(Q')]\!]^{\mathrm{set}}_D &\stackrel{\mathrm{def}}{=\!=} \sigma_\theta([\![Q']\!]^{\mathrm{set}}_D)
\end{aligned}
$$

The above is well-defined because $D$ is a set database and because each record is guaranteed to appear only once in a table (despite the fact that the semantics of each subexpression formally returns bag) [5].

Subsequently, we can define the notion of the preservation of the Codd semantics over sets.

**Definition 12** ([5])**.** An $\mathrm{RA}^{\mathrm{set}}$ query $Q$ preserves Codd semantics over sets if, for every SQL set database $D'$ and for every $D \in \mathsf{codd}(D')$, it holds that $[\![Q]\!]^{\mathrm{set}}_D \in \mathsf{codd}\big([\![Q]\!]^{\mathrm{set}}_{D'}\big)$. This can be equivalently formulated as:

$$
\mathsf{sql}\big([\![Q]\!]^{\mathrm{set}}_D\big) = [\![Q]\!]^{\mathrm{set}}_{D'} \qquad \text{(3a)} \qquad\qquad \text{and } [\![Q]\!]^{\mathrm{set}}_D \text{ is a Codd table.} \qquad \text{(3b)}
$$

Even though the condition (2c) is not strictly required for the preservation over sets, we can still use it to come up with weaker conditions. To understand how, consider the following situation. In [5], authors argue that `DJB` is no longer sufficient for the union node to preserve Codd semantics, which they demonstrate using the following example.

**Example 9** ([5])**.** Consider a schema with $R$ and $S$ over a single nullable attribute $A$. The $\mathrm{RA}^{\mathrm{set}}$ query $Q = R \cup S$ trivially satisfies `DJB`, but it does not preserve Codd semantics over sets. To see this, consider the Codd database $D$ and its SQL equivalent $D'$, where

$$
[\![R]\!]_D = \begin{array}{|c|}\hline B \\ \hline \bot_1 \\ \hline \end{array} \quad ; \quad [\![S]\!]_D = \begin{array}{|c|}\hline B \\ \hline \bot_2 \\ \hline \end{array} \quad ; \quad [\![R]\!]_{D'} = \begin{array}{|c|}\hline B \\ \hline \mathbf{N} \\ \hline \end{array} \quad ; \quad [\![S]\!]_{D'} = \begin{array}{|c|}\hline B \\ \hline \mathbf{N} \\ \hline \end{array}
$$

Obviously, $D \in \mathsf{codd}(D')$. However, we have:

$$[\![R \cup S]\!]^{\text{set}}_{D} = \begin{array}{|c|}\hline B \\ \hline \bot_1 \\ \bot_2 \\ \hline \end{array} \quad ; \quad [\![R \cup S]\!]^{\text{set}}_{D'} = \begin{array}{|c|}\hline B \\ \hline \mathbf{N} \\ \hline \end{array}$$

Clearly, $[\![Q]\!]^{\text{set}}_{D} \notin \mathsf{codd}([\![Q]\!]^{\text{set}}_{D'})$. However, in this example, the problem is not that DJB fails to guarantee that $[\![Q]\!]^{\text{set}}_{D}$ is a Codd table (its original purpose), but rather that it does not ensure that $\mathsf{sql}\big([\![Q]\!]^{\text{set}}_{D}\big) = [\![Q]\!]^{\text{set}}_{D'}$. Therefore, instead of discarding the sufficient conditions for (3b) which do not happen to ensure (3a) as well, we suggest finding new conditions on top of them which will guarantee the satisfiability of the condition (3a) - on their own or with conjunction with other conditions. As it turns out, requiring the condition (2c) to hold achieves exactly that. But before proving it, we need to introduce another technical result derived in [5].

**Lemma 3** (Lemma 9 in [5])**.** *Let $Q$ be an RA$^{\text{set}}$ query whose syntax tree is such that every $\cap$ and $-$ node satisfies* DJN*. Then, for all set databases $D$ and $D'$ such that $D' = \mathsf{sql}(D)$, it holds that:*

$$\varepsilon\big(\mathsf{sql}([\![Q]\!]^{\text{set}}_{D})\big) = [\![Q]\!]^{\text{set}}_{D'}$$

As a consequence, for all RA$^{\text{set}}$ queries $Q$ satisfying the premises of Lemma 3, proving that the condition (3a) is satisfied comes down to showing that $\mathsf{sql}([\![Q]\!]^{\text{set}}_{D})$ is a set [5]. With this in mind, we can now formally present our results.

**Proposition 9.** *Let $Q$ be an RA$^{\text{set}}$ query such that $\varepsilon\big(\mathsf{sql}([\![Q]\!]^{\text{set}}_{D})\big) = [\![Q]\!]^{\text{set}}_{D'}$ for every SQL set database $D'$ and for every $D \in \mathsf{codd}(D')$. If no two records in $[\![Q]\!]^{\text{set}}_{D}$ differ only in null values, then $\mathsf{sql}([\![Q]\!]^{\text{set}}_{D}) = [\![Q]\!]^{\text{set}}_{D'}$.*

**Proof.** Let $D'$ be an SQL set database and $D$ be its Codd interpretation, that is $D \in \mathsf{codd}(D')$. Also, let $Q$ be an RA$^{\text{set}}$ query such that (‡) $\varepsilon\big(\mathsf{sql}([\![Q]\!]^{\text{set}}_{D})\big) = [\![Q]\!]^{\text{set}}_{D'}$ and in which no records differ only in null values. Now, since $D$ is a set database, it follows that $[\![Q]\!]^{\text{set}}_{D}$ is a set table. Thus, for every record $r$ in $[\![Q]\!]^{\text{set}}_{D}$ we know that $\#(r, [\![Q]\!]^{\text{set}}_{D}) = 1$. Consequently, by the equivalence of statements (a) and (b) in Proposition 5, we get that $\#(\mathsf{sql}(r), \mathsf{sql}([\![Q]\!]^{\text{set}}_{D})) = 1$ meaning that (†) $\mathsf{sql}([\![Q]\!]^{\text{set}}_{D})$ must be a set as well. Hence:

$$\mathsf{sql}([\![Q]\!]^{\text{set}}_{D}) \overset{(\dagger)}{=} \varepsilon(\mathsf{sql}([\![Q]\!]^{\text{set}}_{D})) \overset{(\ddagger)}{=} [\![Q]\!]^{\text{set}}_{D'}$$

which completes the proof. □

As a result, the above proposition acts like a ”proxy” letting us find sufficient conditions for (3a) by coming up with sufficient conditions for (2c) and combining them with conditions of Lemma 3, which we believe is easier as (2c) is more ”concrete” than (3a).

Moreover, it should be noted that the conditions of Proposition 6 can also be used to ensure (2c) under set semantics. This is because every RA$^{\text{set}}$ query $Q$ evaluated on set databases under set semantics can be rewritten as the RA query evaluated under bag semantics by immediately following each operation in $Q$ with duplicate elimination. As Proposition 6 does not impose any restrictions on the added $\varepsilon$ nodes, it means that the conditions must be the same for the two evaluation modes.

Building on these observations we present new sufficient conditions for the preservation of Codd semantics over sets.

**Theorem 6.** *Let $Q$ be an RA$^{\text{set}}$ query whose syntax tree is such that:*

a) *each $\cap$ and $-$ node satisfies* DJN*;*

b) *each $\times$ and $\pi_\alpha$ node satisfies* NNA

c) *each $\cup$ node satisfies (*NNA *or* DJN *or* NNC*) AND (*NNA *or* DJEN *or* SESM*)*

*Then, $Q$ preserves Codd semantics over sets.*

**Proof of Theorem 6.** Let $Q$ be an RA$^{\text{set}}$ query whose syntax tree satisfies the conditions of the theorem. Also, let $D'$ be an SQL set database and $D$ be its Codd interpretation, that is $D \in \mathsf{codd}(D')$. We need to show (3a) and (3b).

We start by proving (3a). To this end, notice that each $\cap$ and $-$ node satisfies DJN, hence by Lemma 3, it holds that $\varepsilon\big(\mathsf{sql}([\![Q]\!]^{\text{set}}_D)\big) = [\![Q]\!]^{\text{set}}_{D'}$. Moreover, every $\pi_\alpha$ node satisfies the NNA condition and every $\cup$ node satisfies NNA or DJN or NNC condition, therefore by Proposition 6, we know that $[\![Q]\!]^{\text{set}}_D$ does not contain any two records that differ only in nulls. Then, combining these two observations together, by Proposition 9, we can conclude that $\mathsf{sql}([\![Q]\!]^{\text{set}}_D) = [\![Q]\!]^{\text{set}}_{D'}$ completing the first part of the proof.

As for (3b), we prove it by the induction on the structure of $Q$.

**Base cases:** ([5]):

- $Q$ is a relation name $R$. Then, $[\![Q]\!]^{\text{set}}_D = [\![R]\!]_D$, which is a Codd table because $D$ is a Codd database by assumption.
- $Q$ is non-nullable. Then, $[\![Q]\!]^{\text{set}}_D$ is a complete set, so it is a Codd table.

**Induction**:

Note that cases when $Q$ is $\pi_\alpha(Q_1)$ or $Q_1 \times Q_2$ or $Q_1 \cup Q_2$ and satisfies NNA are covered in the base cases as NNA implies that $Q$ is non-nullable. Similarly, when $Q$ is $Q_1 \cap Q_2$ and satisfies DJN as it also enforces non-nullability.

- $Q$ is $Q_1 \cup Q_2$ and satisfies DJEN:

We have $[\![Q]\!]^{\text{set}}_D = \varepsilon([\![Q_1]\!]^{\text{set}}_D \cup [\![Q_2]\!]^{\text{set}}_D)$. By the induction hypothesis, $[\![Q_1]\!]^{\text{set}}_D$ and $[\![Q_2]\!]^{\text{set}}_D$ are Codd tables (without repeated nulls). In the proof of Theorem 3, we showed that if that is the case and $\cup$ node satisfies DJEN, then $[\![Q_1]\!]^{\text{set}}_D \cup [\![Q_2]\!]^{\text{set}}_D$ is also a Codd table (without repeated nulls). Consequently, $[\![Q]\!]^{\text{set}}_D$ must be a Codd table as well.

- $Q$ is $Q_1 \cup Q_2$ and satisfies SESM:

We have $[\![Q]\!]^{\text{set}}_D = \varepsilon([\![Q_1]\!]^{\text{set}}_D \cup [\![Q_2]\!]^{\text{set}}_D)$. By the induction hypothesis, $[\![Q_1]\!]^{\text{set}}_D$ and $[\![Q_2]\!]^{\text{set}}_D$ are Codd set tables. In the proof of Theorem 5, we showed that if that is the case and $\cup$ node satisfies SESM, then $[\![Q_1]\!]^{\text{set}}_D \cup [\![Q_2]\!]^{\text{set}}_D$ is a Codd table with potentially repeated nulls among duplicated records. Consequently, $\varepsilon([\![Q_1]\!]^{\text{set}}_D \cup [\![Q_2]\!]^{\text{set}}_D)$ get rids of duplicated records, and therefore of repeated marked nulls, thus $[\![Q]\!]^{\text{set}}_D$ must be a Codd set table.

- $Q$ is $\mathsf{op}_1(Q_1)$ for $\mathsf{op}_1 \in \{\sigma_\theta, \rho_{A\to B}\}$ or $Q_1 - Q_2$:

The proof is exactly the same as the proof for the corresponding case in Theorem 3 in [5]. □

Having found the new sufficient conditions for sets, we can compare them with the sufficient conditions for the preservation of Codd semantics presented in [5] (see Theorem 7 in Appendix A). Clearly, the two differ only in restrictions imposed on $\cup$ nodes. To contrast them, recall that our restrictions for union nodes can be equivalently formulated as: $\texttt{NNA} \vee \texttt{NNC} \vee (\texttt{DJN} \wedge \texttt{DJEN})$. Obviously, these are weaker than $\texttt{NNA} \vee \texttt{NNC}$ required by Theorem 7. And since there are many examples of unions satisfying $(\texttt{DJN} \wedge \texttt{DJEN})$ without satisfying NNA or NNC (e.g. $\sigma_{\mathsf{const}(A)}(R) \cup \sigma_{\mathsf{const}(B)}(R)$, where $R$ is over nullable attributes $A$ and $B$), we can conclude that our conditions are strictly weaker than those proposed in [5].

# Chapter 5

# Conclusions and Future Work

In this report, we set out several improvements to the process of recognising Codd semantics queries, which can be roughly divided into three categories.

First, we presented the derivation process of the DJEN condition that improves on the identified shortcomings of PDJB. Moreover, we proved that DJEN is strictly weaker than PDJB. This enabled us to identify chains of unions preserving Codd semantics that could not be recognised as such before. Also, we showed that with the help of DJEN, we can guarantee formulation-independent recognition of chains of unions preserving Codd semantics. In other words, we proved that if some chain of unions does not satisfy sufficient conditions, then its other equivalent formulations cannot satisfy them as well. While we showed that the use of a variadic union is completely unnecessary in the verification process, its ability to abstract away the ordering information happened to be immensely useful in proving that the satisfiability of DJEN does not depend on the exact formulation of a chain. We believe that a similar technique could be applied to find a formulation-independent condition for chains of intersection. This would remove the need for replacing chains of intersections with single variadic intersections nodes as is currently suggested in Part 1.

Secondly, we studied the conditions for the preservation of the interpretation of nulls in which the codd function assigns the same marked nulls to repeated records. We found out that for such semantics to be preserved, queries must satisfy the extra condition (2c). We came up not only with the sufficient conditions for (2c) but also with the new sufficient condition, SESM, for (2b). Unfortunately, we also showed that currently SESM does not allow us to capture any new queries preserving Codd semantics. This does not mean that its discovery is pointless though. At the moment, its potential is limited by quite strict conditions for (2c) that are required alongside it. However, when new weaker sufficient conditions for (2c) are found, it might be the case that SESM will enable us to capture queries that would not be recognised by any other condition. Eventually, we showed that the conditions ensuring the preservation of this new interpretation of nulls are stricter than those guaranteeing the preservation of the original Codd semantics. Again, this is mainly due to quite strict conditions for (2c).

Last but not least, we showed that the two interpretations of nulls are equivalent when

queries are evaluated on sets. This allowed us to use the new sufficient conditions to derive weaker requirements for the preservation of Codd semantics over sets. This time, the process was actually facilitated by the new findings related to the condition (2c). This is because we showed that when (2c) holds it is easier to prove that the requirement (3a) is satisfied as well.

Given all of that, there are still many directions one could explore beyond finding even weaker conditions for the preservation of the Codd semantics. The first thing that comes to mind would be to check whether real-life queries written in a suitable fragment of SQL are Codd semantics preserving. To achieve that, SQL queries from some datasets could be translated to relational algebra as described in [4] and then verified using the results presented in this MInf Project (Parts 1 & 2). The results of such evaluation could tell us whether the assumption about the Codd interpretation of SQL nulls does in fact hold in the majority of real-life queries. However, even if the results of the study are not positive, it would not immediately mean that most queries do not preserve the semantics of nulls. This is because the presented conditions are only sufficient and not strictly necessary.

With respect to the above considerations, at least two things come to mind which can hinder such a study. Firstly, the exact translation rules may influence the results as the verification process is still formulation dependent - even with all the presented improvements. We hope that this issue can be mitigated by first normalising queries to a form that is more likely to satisfy the conditions (as explained in Part 1). Therefore, it is important to further improve this transformation procedure. For example, one could think about propagating selections or projections towards the leaves of a syntax tree to narrow down the nullable signature of a larger number of nodes. Secondly, many real-life queries include groupings and aggregations which are not currently studied in the context of the preservation of Codd semantics. Doing that could greatly increase the number of eligible SQL queries that could be checked during the study.

# Bibliography


[1] Marco Console, Paolo Guagliardo, Leonid Libkin, and Etienne Toussaint. Coping with Incomplete Data: Recent Advances. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS'20, pages 33–47, New York, NY, USA, June 2020. Association for Computing Machinery.

[2] Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. In Torsten Grust, Hagen Höpfner, Arantza Illarramendi, Stefan Jablonski, Marco Mesiti, Sascha Müller, Paula-Lavinia Patranjan, Kai-Uwe Sattler, Myra Spiliopoulou, and Jef Wijsen, editors, *Current Trends in Database Technology – EDBT 2006*, pages 278–296, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[3] Paolo Guagliardo and Leonid Libkin. Correctness of SQL Queries on Databases with Nulls. *ACM SIGMOD Record*, 46(3):5–16, October 2017.

[4] Paolo Guagliardo and Leonid Libkin. A formal semantics of sql queries, its validation, and applications. *Proc. VLDB Endow.*, 11(1):27–39, sep 2017.

[5] Paolo Guagliardo and Leonid Libkin. On the codd semantics of SQL nulls. *Inf. Syst.*, 86:46–60, 2019.

[6] Tomasz Imieliński and Witold Lipski. Incomplete Information in Relational Databases. *Journal of the ACM*, 31(4):761–791, September 1984.

[7] Witold Lipski. On relational algebra with marked nulls preliminary version. In *Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems*, PODS '84, pages 201–203, New York, NY, USA, April 1984. Association for Computing Machinery.

[8] Konrad Pijanowski. coddifier. https://github.com/kopi22/coddifier, 2022. Retrieved April 2, 2023.

[9] Konrad Pijanowski. Preservation of codd semantics in databases with sql nulls. Bachelor's thesis, University of Edinburgh, 2022.

[10] Peter Storeng. Implementing marked nulls in postgresql. Master's thesis, University of Edinburgh, 2016.

[11] Etienne Toussaint, Paolo Guagliardo, Leonid Libkin, and Juan Sequeda. Troubles with nulls, views from the users. *Proceedings of the VLDB Endowment*, 15(11):2613–2625, July 2022.

# Appendix A

# Additional and External Results

## A.1 Theorems

**Theorem 7** (Theorem 3 in [5])**.** *Let $Q$ be an $RA^{\text{set}}$ query whose syntax tree is such that:*

a) *each $\cap$ and $-$ node satisfies* `DJN`*;*

b) *each $\times$ and $\pi_\alpha$ node satisfies* `NNA`*;*

c) *each $\cup$ node satisfies* `NNC` *or* `NNA`*.*

*Then, $Q$ preserves Codd semantics over sets.*

## A.2 Propositions

**Proposition 10.** *A node in the syntax tree of a query satisfies* `PDJB` *if and only if every pair of its distinct children, $Q_i$ and $Q_j$, satisfies* `DJB` *or* `NNC` *condition, that is:*

- *$Q_i$ or $Q_j$ is non-nullable*
- *or their bases are disjoint*

See Appendix B for the proof.

## A.3 Lemmas

**Lemma 4.** *Let $T_1$ to $T_n$ be tables. Then* $\mathsf{sql}(\bigcup(T_1,\ldots,T_n)) = \bigcup(\mathsf{sql}(T_1),\ldots,\mathsf{sql}(T_n))$.

The proof is given in Appendix B.

# Appendix B

# Additional Proofs

## B.1 Propositions

**Proof of Proposition 1 (extending the proof of Proposition 4 in [5]).**
Let $Q$ be an RA query whose syntax tree is such that every $\cap$ and $-$ node satisfies DJN, and every $\varepsilon$ node satisfies NNC. Let $D$ be a database and let $D' = \mathsf{sql}(D)$. By induction on the structure of $Q$, we will show that $[\![Q]\!]_{D'} = \mathsf{sql}([\![Q]\!]_D)$.

**Base case:** $Q$ is a relation $R$. Then, obviously, $[\![R]\!]_{D'} = \mathsf{sql}([\![R]\!]_D)$ since $D' = \mathsf{sql}(D)$.

**Induction:**

- $Q$ is $\bigcup(Q_1, \ldots, Q_n)$:

  By applying (†) the semantics of the variadic intersection in queries, (‡) the induction hypothesis, (††) Lemma 4, we get:

$$\begin{aligned}
\Big[\!\Big[\bigcup(Q_1,\ldots,Q_n)\Big]\!\Big]_{D'} &\overset{(\dagger)}{=} \bigcup([\![Q_1]\!]_{D'},\ldots,[\![Q_n]\!]_{D'}) \\
&\overset{(\ddagger)}{=} \bigcup\big(\mathsf{sql}\big([\![Q_1]\!]_D\big),\ldots,\mathsf{sql}\big([\![Q_n]\!]_D\big)\big) \\
&\overset{(\dagger\dagger)}{=} \mathsf{sql}\Big(\bigcup([\![Q_1]\!]_D,\ldots,[\![Q_n]\!]_D)\Big) \\
&\overset{(\dagger)}{=} \mathsf{sql}\Big(\Big[\!\Big[\bigcup(Q_1,\ldots,Q_n)\Big]\!\Big]_D\Big)
\end{aligned}$$

- $Q$ is $\mathsf{op}_1(Q_1)$ for $\mathsf{op}_1 \in \{\varepsilon, \pi_\alpha, \sigma_\theta, \rho_{A\to B}\}$ or $Q_1\,\mathsf{op}_2\,Q_2$ for $\mathsf{op}_2 \in \{\times, \cup, \cap, -\}$:

  The proof is exactly the same as in the proof for the corresponding case in Proposition 4 in [5].

□

**Proof of Proposition 2.** Let $Q_1$ to $Q_n$ be queries to be combined using a chain of unions or the $n$-ary union operation. Also, let us assume, without loss of generality, that for some $k \leq n$, $Q_1$ to $Q_k$ are nullable and $Q_{k+1}$ to $Q_n$ are not.

We start by showing that there always exists a chain of binary unions in which each union node satisfies either DJB or NNC condition, if we assume that $\bigcup(Q_1,\ldots,Q_n)$ satisfies PDJB, that is, $Q_1$ to $Q_k$ have pairwise disjoint bases. Under this assumption, consider the equivalent chain of unions:

$$((((((Q_1 \cup^2 Q_2) \cup^3 Q_3)...) \cup^k Q_k) \cup^{k+1} Q_{k+1})...) \cup^n Q_n$$

where we marked each binary union with a superscript $i$ and the children of $\cup^i$ are subqueries $Q_l = ((Q_1 \cup Q_2)\cdots) \cup Q_{i-1}$ and $Q_r = Q_i$. Now, we need to show that all nodes $\cup^i$ for $2 \leq i \leq n$ satisfy either DJB or NNC condition.

Firstly, for $2 \leq i \leq k$, the union $\cup^i$ is combining only nullable operands (by assumption). We show that each such node $\cup^i$ satisfies the DJB condition:

$$\begin{aligned}
\mathsf{base}(Q_l) \cap \mathsf{base}(Q_r) &= \Big(\bigcup_{p=1}^{i-1} \mathsf{base}(Q_p)\Big) \cap \mathsf{base}(Q_i) \\
&= \bigcup_{p=1}^{i-1} \big(\mathsf{base}(Q_p) \cap \mathsf{base}(Q_i)\big) && \text{(distributive law)} \\
&= \bigcup_{p=1}^{i-1} \varnothing && \text{(PDJB condition)} \\
&= \varnothing
\end{aligned}$$

Secondly, for $k < i \leq n$, the union $\cup^i$ has a non-nullable query $Q_i$ as its right child, therefore it must satisfy the NNC condition.

Combining the two observations we get that nodes $\cup^2$ to $\cup^k$ satisfy the DJB condition and nodes $\cup^{k+1}$ to $\cup^n$ satisfy the NNC condition, thus all union nodes in the chain satisfy either DJB or NNC.

To prove the statement in the other direction, we show that $Q_1$ to $Q_k$ must have pairwise disjoint bases whenever there exists a chain in which every union satisfies DJB or NNC. To this end, consider a chain that satisfies this assumption. Then, for any pair of distinct nullable operands $Q_i$ and $Q_j$, let $\cup^*$ be their lowest common ancestor node in the chain of unions. Without loss of generality, we can assume that $Q_i$ and $Q_j$ are subqueries of its left and right children, $Q_l$ and $Q_r$, respectively.

Using the fact that $Q_l$ must be $Q_i$ itself or a chain of unions containing $Q_i$, we know that $\mathsf{n\text{-}sig}(Q_i) \subseteq \mathsf{n\text{-}sig}(Q_l)$. Similarly for $Q_r$ and $Q_j$. Now, because $Q_i$ and $Q_j$ are nullable, so must be $Q_l$ and $Q_r$. Therefore, $\cup^*$ cannot satisfy the NNC condition and must satisfy DJB as we assumed it satisfies at least one of these two.

Finally, combining the facts that $\mathsf{base}(Q_i) \subseteq \mathsf{base}(Q_l)$ and $\mathsf{base}(Q_j) \subseteq \mathsf{base}(Q_r)$ with the observation that $\mathsf{base}(Q_l) \cap \mathsf{base}(Q_r) = \varnothing$ which is guaranteed by the DJB condition, we can conclude that

$$\mathsf{base}(Q_i) \cap \mathsf{base}(Q_j) \subseteq \mathsf{base}(Q_l) \cap \mathsf{base}(Q_r) = \varnothing$$

proving that each pair of distinct nullable operands must have disjoint bases. This finishes the proof of Proposition 2. □

**Proof of Proposition 3.** Both PDJB and DJEN are conditions that impose their restrictions on the node's children in a pairwise fashion. Therefore, to prove that DJEN is strictly weaker than PDJB it suffices to show that the condition enforced by PDJB on any pair of children guarantees that the same pair satisfies the requirement of DJEN as well. To this end, let $Q_i$ and $Q_j$ be two children of the node that satisfies PDJB. Consequently, it must be the case that $\mathsf{base}(Q_i) \cap \mathsf{base}(Q_j) = \varnothing$ or either $Q_i$ or $Q_j$ is non-nullable (as the requirement applies to nullable children only). We will consider these two cases separately.

- Bases of $Q_i$ and $Q_j$ are disjoint:

  For DJEN to be satisfied, equation (4) must hold for all $R$ in $\mathsf{base}(Q_i) \cap \mathsf{base}(Q_j)$. Since there are no such relations, as bases of $Q_i$ and $Q_j$ are disjoint, DJEN is trivially satisfied.

- $Q_i$ or $Q_j$ is non-nullable:

  Without loss of generality, let us assume that $Q_i$ is the non-nullable child, so $\mathsf{n\text{-}sig}(Q_i) = \varnothing$. Consequently, (†) for every $R \in \mathsf{base}(Q_i)$ it is the case that $\mathsf{en\text{-}sig}(R, Q_i)$ is equal to $\varnothing$:

$$\begin{aligned}
\mathsf{en\text{-}sig}(R, Q_i) &= \bigcup_{R^i \in \mathsf{base}(Q_i)} \psi^{-1}_{R^i}(Q_i)\big(\mathsf{n\text{-}sig}(Q_i)\big) && \text{(definition)} \\
&= \bigcup_{R^i \in \mathsf{base}(Q_i)} \psi^{-1}_{R^i}(Q_i)\big(\varnothing\big) && (Q_i \text{ is non-nullable}) \\
&= \bigcup_{R^i \in \mathsf{base}(Q_i)} \varnothing = \varnothing
\end{aligned}$$

  Therefore, the equation (4) is always satisfied for all $R$ in $\mathsf{base}(Q_i) \cap \mathsf{base}(Q_j)$:

$$\mathsf{en\text{-}sig}(R, Q_i) \cap \mathsf{en\text{-}sig}(R, Q_j) \overset{(\dagger)}{=} \varnothing \cap \mathsf{en\text{-}sig}(R, Q_j) = \varnothing$$

  As required by the DJEN condition.

To show that converse is not necessarily the case, that is DJEN does not imply PDJB, consider the following query: $\sigma_{\mathsf{const}(A)}(R) \cup \sigma_{\mathsf{const}(B)}(R)$, where relation $R$ is over nullable attributes $A$ and $B$. Now, let $Q_1 = \sigma_{\mathsf{const}(A)}(R)$ and $Q_2 = \sigma_{\mathsf{const}(B)}(R)$ be the nullable children of the $\cup$ node. Clearly, $\mathsf{base}(Q_1) = \mathsf{base}(Q_2) = \{R\}$, so the PDJB condition is not satisfied as the bases are not disjoint. On the other hand, one can quickly verify that $\mathsf{en\text{-}sig}(R, Q_1) = \{B\}$, while $\mathsf{en\text{-}sig}(R, Q_2) = \{A\}$. So DJEN is satisfied because the effective nullable signatures of $R$ w.r.t. $Q_1$ and $Q_2$ are disjoint. This proves that DJEN is strictly weaker than PDJB.

□

**Proof of Proposition 5.** Let $T$ be a table and let $T' = \mathsf{sql}(T)$. We prove the proposition using the circular chain of implications.

- $(a) \to (c)$: Let us assume that no two records in $T$ differ only in nulls. We prove the first part of the conclusion - "for every record $r'$ in $T'$, there exists a unique record $r$ in $T$ such that $r \in \mathsf{sql}^{-1}(r')$" - by contradiction.

To begin with, let $r'$ be a record in $T'$. Also, let $r_1$ and $r_2$ be two distinct records in $\mathsf{sql}^{-1}(r')$ and let $r_1, r_2 \in T$. We have to consider two cases. If $r'$ does not contain any nulls, then $\mathsf{sql}^{-1}(r') = \{r'\}$, so $r_1$ and $r_2$ cannot be distinct which contradicts our assumption. Otherwise, $r_1$ and $r_2$ must differ only in nulls since constant values are unaffected by the $\mathsf{sql}^{-1}$ operation. This in turn contradicts our premise - the statement $(a)$. Therefore, we can conclude that it is impossible for two such records to be in $T$ and there must exist a unique record $r$ in $T$ such that $r \in \mathsf{sql}^{-1}(r')$

As for the second part of the conclusion, it must hold because $(\dagger)$ $r$ is the only record which is both in $\mathsf{sql}^{-1}(r')$ and in $T$, so:

$$\#(r', T') = \sum_{s \in \mathsf{sql}^{-1}(r')} \#(s, T) \overset{(\dagger)}{=} \#(r, T)$$

- $(c) \to (b)$: Let us assume that $(\dagger)$ for every record $r'$ in $T'$, there exists a unique record $r$ in $T$ such that $r \in \mathsf{sql}^{-1}(r')$, in which case $(\ddagger)$ $\#(r, T) = \#(r', T')$.

  Now, clearly, for every record $s \in T$, the record $s' = \mathsf{sql}(s)$ is in $T'$. Thus, by $(\dagger)$, $s$ is the unique record in $T$ which is also in $\mathsf{sql}^{-1}(s')$. Consequently:

$$\#(s, T) \overset{(\ddagger)}{=} \#(s', T') = \#(\mathsf{sql}(s), T')$$

- $(b) \to (a)$: First, observe that two records containing nulls, call them $p$ and $q$, differ only in nulls if and only if:

  1. $p \neq q$, and
  2. $\mathsf{sql}(p) = \mathsf{sql}(q)$, and
  3. $p, q \in \mathsf{sql}^{-1}(\mathsf{sql}(p)) = \mathsf{sql}^{-1}(\mathsf{sql}(q))$.

  Therefore, we will prove that no two records in $T$ differ only in nulls, assuming that for every record $r$ in $T$ it is the case that $\#(r, T) = \#(\mathsf{sql}(r), T')$, by showing that no record other than $r$ belongs to both $\mathsf{sql}^{-1}(\mathsf{sql}(r))$ and $T$ at the same time.

  For that, let $r' = \mathsf{sql}(r)$. Then:

$$\#(r', T') = \sum_{s \in \mathsf{sql}^{-1}(r')} \#(s, T) = \#(r, T) + \sum_{s \in (\mathsf{sql}^{-1}(r') - \{r\})} \#(s, T) = \#(r', T') + \sum_{s \in (\mathsf{sql}^{-1}(r') - \{r\})} \#(s, T)$$

  The last equality holds because of our assumption we know that $\#(r, T) = \#(r', T')$. Now, subtracting $\#(r', T')$ from both sides of the above equation we get that:

$$\sum_{s \in (\mathsf{sql}^{-1}(r') - \{r\})} \#(s, T) = 0$$

  So no record from $\mathsf{sql}^{-1}(r')$ other than $r$ is present in $T$, which completes the proof.

□

**Proof of Proposition 6.** Let Q be an RA query whose syntax tree is such that every $\pi_\alpha$ node satisfies the `NNA` condition and every $\cup$ node satisfies one of `NNA`, `DJN`, or `NNC`

conditions. Let $D'$ be an SQL database and let $D$ be its Codd interpretation, that is $D \in \mathsf{codd}(D')$. We proceed with the proof by induction on the structure of $Q$.

**Base cases**:

- $Q$ is a relation name $R$.

  By definition, $[\![R]\!]_D \in \mathsf{codd}([\![R]\!]_{D'})$. Now, towards contradiction, let us assume that there exist two distinct records $r_1$ and $r_2$ in $[\![R]\!]_D$ that differ only in nulls. Consequently, $\mathsf{sql}(r_1)$ and $\mathsf{sql}(r_2)$ in $[\![R]\!]_{D'}$ are exactly the same. On the other hand, codd maps the same records in $[\![R]\!]_{D'}$ to the same records in $\mathsf{codd}([\![R]\!]_{D'})$. Therefore, $r_1$ and $r_2$ would have to be the same in $[\![R]\!]_D$, which contradicts our initial assumption. Hence there cannot exist two such records in $[\![Q]\!]_D$.

- $Q$ is non-nullable. Then $[\![Q]\!]_D$ is a complete table so it cannot contain any records with null values.

Note that cases when $Q$ is $\pi_\alpha(Q_1)$ and when $Q$ is $Q_1 \cup Q_2$ and satisfies `NNA` are already covered by the base case as the `NNA` condition forces $Q$ to be non-nullable.

**Inductive steps**:

- $Q$ is $\rho_{A\to B}(Q_1)$:

  The renaming operation only changes the name of the attribute $A$ to $B$ in each record in $[\![Q_1]\!]_D$. It does not affect any values associated with that attribute. Since by the induction hypothesis $[\![Q_1]\!]_D$ does not contain two records which differ only in null values, there cannot be such records in $[\![Q]\!]_D$ as well.

- $Q$ is $\mathsf{op}_1(Q_1)$ for $\mathsf{op}_1 \in \{\sigma_\theta, \varepsilon\}$ or $Q_1\ \mathsf{op}_2\ Q_2$ for $\mathsf{op}_2 \in \{\cap, -\}$:

  The claim follows from the fact that $[\![Q]\!]_D \subseteq [\![Q_1]\!]_D$. By the induction hypothesis, $[\![Q_1]\!]_D$ does not contain two records that differ only in nulls. Therefore, it must be the case for $[\![Q]\!]_D$ as well.

- $Q$ is $Q_1 \times Q_2$:

  Let $r_1, s_1$ be records in $[\![Q_1]\!]_D$ and $r_2, s_2$ be records in $[\![Q_2]\!]_D$. Then, the records $r = r_1 \times r_2$ and $s = s_1 \times s_2$ must be in $[\![Q]\!]_D$. If $r$ and $s$ were to differ only in nulls in $[\![Q]\!]_D$, then it would have to be the case that either $r_1$ and $s_1$ or $r_2$ and $s_2$ must also differ in nulls only. This is because $r$ and $s$ can differ only in nulls iff:

  a) $r_1 = s_1$ and $r_2$ and $s_2$ differ only in nulls;

  a) $r_1$ and $s_1$ differ only in nulls and $r_2 = s_2$;

  a) $r_1$ and $s_1$ differ only in nulls and $r_2$ and $s_2$ differ only in nulls.

  Although, by the induction hypothesis, we know that neither $[\![Q_1]\!]_D$ nor $[\![Q_2]\!]_D$ can have two records differing only in nulls. Therefore, $[\![Q]\!]_D$ cannot contain two records that differ only in nulls as well.

- $Q$ is $Q_1 \cup Q_2$ and satisfies `NNC`:

Let us assume without loss of generality that $Q_1$ is the non-nullable child. Also, by the induction hypothesis, $[\![Q_2]\!]_D$ does not contain two records that differ only in nulls. Since $[\![Q_2]\!]_D$ is the only child that can contribute records with nulls to the query answer, it follows that $[\![Q]\!]_D$ does not contain two such records as well.

- $Q$ is $Q_1 \cup Q_2$ and satisfies DJN:

  Since the $\cup$ node satisfies the DJN condition, we know that for any two records $r_1 \in [\![Q_1]\!]_D$ and $r_2 \in [\![Q_2]\!]_D$, whenever $r_1(A) \in \mathsf{Null}$ for an attribute $A \in \mathsf{n\text{-}sig}(Q)$, it must be the case that $r_2(A) \in \mathsf{Const}$ and vice-versa. So it is impossible for $r_1$ and $r_2$ to differ only in nulls because they never take a null value for the same attribute. Therefore, it follows that two such records from separate children cannot be put together in the output table and violate the condition.

  Moreover, by the induction hypothesis, each of the tables $[\![Q_1]\!]_D$ and $[\![Q_2]\!]_D$ does not contain two records that differ only in nulls so such records cannot propagate to the output from the same child. Combining the two results, we conclude that $[\![Q]\!]_D$ cannot contain such two records, thus it satisfies the condition (2c).

□

**Proof of Proposition 7.** Let $Q'$ be an RA query whose syntax tree is such that every $\pi_\alpha$ and $\times$ node satisfies the NNA condition. Under this assumption, we want to prove that for every $\cup$ node in $Q'$, it holds that:

$$(\cup \models \texttt{DJN}, \texttt{SESM} \wedge \cup \not\models \texttt{NNA}) \implies \cup \models \texttt{DJEN}$$

We prove the statement by contradiction. To this end, let us assume that $Q = Q_1 \cup Q_2$ is a subquery of $Q'$ and that this $\cup$ node satisfies DJN and SESM, while it does not satisfy NNA and DJEN. Therefore, we know that:

1. $Q$ and all of its ancestors are nullable - as $Q$ does not satisfy NNA.

   Consequently, as every $\times$ and $\pi_\alpha$ node in $Q'$ satisfies NNA, it follows that (†) all, if any, $\times$ and $\pi_\alpha$ nodes must be descendants of $Q$. Otherwise, $Q$ would satisfy NNA itself.

2. $\mathsf{n\text{-}sig}(Q_1) \cap \mathsf{n\text{-}sig}(Q_2) = \varnothing$ - as $Q$ satisfies DJN;

3. there exists a base relation $R$ that appears in $Q_1$ and $Q_2$ for which

   $$\mathsf{en\text{-}sig}(R, Q_1) \cap \mathsf{en\text{-}sig}(R, Q_2) \neq \varnothing$$

   as $Q$ does not satisfy DJEN.

Expanding the equation from point (3) using the definition of en-sig we get that:

$$\bigcup_{R^i \in \mathsf{base}(Q_1)} \psi_{R^i}^{-1}(Q_1)\big(\mathsf{n\text{-}sig}(Q_1)\big) \cap \bigcup_{R^j \in \mathsf{base}(Q_2)} \psi_{R^j}^{-1}(Q_2)\big(\mathsf{n\text{-}sig}(Q_2)\big) \neq \varnothing$$

Consequently, there must exist instances $R^i \in \mathsf{base}(Q_1)$ and $R^j \in \mathsf{base}(Q_2)$ of this relation $R$, such that:

$$\big[\psi_{R^i}^{-1}(Q_1)\big(\mathsf{n\text{-}sig}(Q_1)\big)\big] \cap \big[\psi_{R^j}^{-1}(Q_2)\big(\mathsf{n\text{-}sig}(Q_2)\big)\big] \neq \varnothing$$

It can be easily verified using definition that $\psi^{-1}_{R^i}(Q_1) = \psi^{-1}_{R^i}(Q)$ and $\psi^{-1}_{R^j}(Q_2) = \psi^{-1}_{R^j}(Q)$, so the equation B.1 can be rewritten as:

$$\left[\psi^{-1}_{R^i}(Q)\big(\mathsf{n\text{-}sig}(Q_1)\big)\right] \cap \left[\psi^{-1}_{R^j}(Q)\big(\mathsf{n\text{-}sig}(Q_2)\big)\right] \neq \varnothing \tag{9}$$

Now, recall that we write $\phi_{R^i}(Q)(\mathsf{n\text{-}sig}(Q_1))$ to represent a set:

$$\{\phi_{R^i}(Q)(A) \mid A \in \mathsf{n\text{-}sig}(Q_1) \text{ and } \phi_{R^i}(Q)(A) \text{ is defined}\}$$

For that reason, let $X_1$ be a subset of $\mathsf{n\text{-}sig}(Q_1)$ for which $\psi^{-1}_{R^i}(Q)$ is defined and $X_2$ be a subset of $\mathsf{n\text{-}sig}(Q_2)$ for which $\psi^{-1}_{R^j}(Q)$ is defined. Then, we can equivalently rewrite equation (9) as:

$$\left[\psi^{-1}_{R^i}(Q)\big(X_1\big)\right] \cap \left[\psi^{-1}_{R^j}(Q)\big(X_2\big)\right] \neq \varnothing \tag{10}$$

Furthermore, observe that for all sets $X$ such that $\psi^{-1}_{R^i}(Q)$ is defined on elements of $X$, $\psi^{-1}_{R^i}(Q)(X) = \phi^{-1}_{R^i}(Q)(X)$ as $\psi^{-1}_{R^i}(Q)$ is a restriction of $\phi^{-1}_{R^i}(Q)$ . Similarly for $\psi^{-1}_{R^j}(Q)$ and $\phi^{-1}_{R^j}(Q)$. Hence, we can rewrite equation (10) as:

$$\left[\phi^{-1}_{R^i}(Q)\big(X_1\big)\right] \cap \left[\phi^{-1}_{R^j}(Q)\big(X_2\big)\right] \neq \varnothing \tag{11}$$

Now, we know that $R^i$ and $R^j$ must be able to propagate nullable values to the output of the $\cup$ node (otherwise $X_1$ and $X_2$ would be empty sets and the previous equation would not hold). Thus, by Lemma 2, its conditions are satisfied because of (†), it follows that $\phi_{R^i}(Q)$ and $\phi_{R^j}(Q)$ are total and surjective functions from $\mathsf{sig}(R)$ to $\mathsf{sig}(Q)$. Moreover, since $\cup$ satisfies SESM, we know that $\phi_{R^i}(Q) = \phi_{R^j}(Q)$ and consequently $\phi^{-1}_{R^i}(Q) = \phi^{-1}_{R^j}(Q)$. Thus, equation (11) can be written as:

$$\left[\phi^{-1}_{R^i}(Q)\big(X_1\big)\right] \cap \left[\phi^{-1}_{R^i}(Q)\big(X_2\big)\right] \neq \varnothing \tag{12}$$

Finally, applying $\phi_{R^i}(Q)$ map to both sides of the equation (12) and manipulating it using basic properties of a function on sets, we get:

$$\begin{aligned}
\phi_{R^i}(Q)\left(\left[\phi^{-1}_{R^i}(Q)\big(X_1\big)\right] \cap \left[\phi^{-1}_{R^i}(Q)\big(X_2\big)\right]\right) &\neq \phi_{R^i}(Q)(\varnothing) \\
\phi_{R^i}(Q)\left(\left[\phi^{-1}_{R^i}(Q)\big(X_1\big)\right] \cap \left[\phi^{-1}_{R^i}(Q)\big(X_2\big)\right]\right) &\neq \varnothing \\
\phi_{R^i}(Q)\left(\left[\phi^{-1}_{R^i}(Q)\big(X_1\big)\right]\right) \cap \phi_{R^i}(Q)\left(\left[\phi^{-1}_{R^i}(Q)\big(X_2\big)\right]\right) &\neq \varnothing \\
X_1 \cap X_2 &\neq \varnothing
\end{aligned}$$

Now, by definition, $X_1$ and $X_2$ are subsets of $\mathsf{n\text{-}sig}(Q_1)$ and $\mathsf{n\text{-}sig}(Q_2)$, respectively. Hence it has to follow that $\mathsf{n\text{-}sig}(Q_1) \cap \mathsf{n\text{-}sig}(Q_2) \neq \varnothing$, which contradicts our initial assumption that the DJN condition is satisfied. This completes the proof that:

$$(\cup \models \texttt{DJN}, \texttt{SESM} \wedge \cup \not\models \texttt{NNA}) \implies \cup \models \texttt{DJEN}$$

□

**Proof of Proposition 10.**
Let $Q$ be a node in the syntax tree of a query and let $Q_1$ to $Q_n$ be its children. We present the proof in both directions.

• PDJB $\implies$ pairwise (DJB or NNC):

First, let us assume that $Q$ satisfies PDJB. Under this assumption, we will show that every pair of children $Q_i$ and $Q_j$, for $i, j \in [1,n], i \neq j$, satisfies NNC or DJB. Obviously, if either $Q_i$ or $Q_j$ is non-nullable, then such a pair satisfies the NNC condition. Therefore, we are only left with pairs where both $Q_i$ and $Q_j$ are nullable. However, by our initial assumption, we know that all the nullable children must have pairwise disjoint bases and thus each such pair must satisfy the DJB condition.

• Pairwise (DJB or NNC) $\implies$ PDJB:

Now, let us assume that for every pair of children $Q_i$ and $Q_j$, for $i, j \in [1,n], i \neq j$, either:

- $Q_i$ or $Q_j$ is non-nullable,
- or their bases are disjoint.

Based on this assumption, we will show that all pairs of distinct nullable children satisfy the DJB condition. This conclusion follows naturally, as the children nodes in question are not non-nullable, hence it must be the case that their bases are disjoint. This completes the proof. □

## B.2 Lemmas

**Proof of Lemma 2.** Let $Q$ be an RA query whose syntax tree is such that every $\pi_\alpha$ and $\times$ node satisfies the NNA condition. Also, let $R^i$ be an instance of some relation $R \in \mathsf{base}(Q)$. Finally, let $f$ be an effective signature map of $R^i$ w.r.t. $Q$. We prove our statement by the contraposition. That is, we show that if $f$ is not a total and surjective function from $\mathsf{sig}(R)$ to $\mathsf{sig}(Q)$, then $R^i$ cannot propagate null values to the query output.

Before we start, let $\mathsf{Dom}(f)$ denote the domain of definition (or natural domain) of $f$, that is the set of all elements for which $f$ is actually defined (as $f$ can be a partial function). Now, note that $f$ is a total and surjective function if and only if $\mathsf{Dom}(f) = \mathsf{sig}(R)$ ($f$ is total) and $|\mathsf{Dom}(f)| = |\mathsf{sig}(Q)|$ ($f$ is surjective - and bijective since the map is injective by definition). Therefore, one can verify that there are only two cases when $f$ is not total and surjective:

- $\mathsf{Dom}(f) \subset \mathsf{sig}(R)$:

  $f$ is a partial function. This can only happen as a result of an intermediate projection operation. Since all projections satisfy the NNA condition, the result of the projection is either a table without nulls (a complete table) or all of its records containing null values will be eventually discarded. This applies to records containing the nulls from $[\![R^i]\!]$ as well. Consequently, no nulls from $[\![R^i]\!]$ can propagate to $[\![Q]\!]_D$.

- $\mathsf{Dom}(f) = \mathsf{sig}(R)$ and $|\mathsf{Dom}(f)| < |\mathsf{sig}(Q)|$:

  If $|\mathsf{Dom}(f)| < |\mathsf{sig}(Q)|$, then we can be sure that there must be at least one Cartesian product node in the syntax tree of $Q$ on the path from $R^i$ to the root. This is because the query $Q$ must be well-defined so all the records in the query output must have the same signature. And the only way to extend the signature of a record is by using the

Cartesian product operation. Since all $\times$ nodes satisfy the NNA condition, we use the same reasoning as before to conclude that no nulls from $[\![R^i]\!]$ can propagate to $[\![Q]\!]_D$.

□

**Proof of Lemma 4 (based on the proof of Lemma 3a in [5]).**
Let $T_1$ to $T_n$ be tables over the same signature. Clearly, both tables $\mathsf{sql}(\bigcup(T_1,\ldots,T_n))$ and $\bigcup(\mathsf{sql}(T_1),\ldots,\mathsf{sql}(T_n))$ are well-defined and they are both over $\mathsf{sig}(T_1)$. Then, for every record $r$ over $\mathsf{sig}(T_1)$, we get:

$$\begin{aligned}
\#\Big(r,\mathsf{sql}\Big(\bigcup(T_1,\ldots,T_n)\Big)\Big) &= \#(r,\mathsf{sql}(T_1\cup\cdots\cup T_n))\\
&\overset{(\dagger)}{=} \sum_{s\in\mathsf{sql}^{-1}(r)}\#\big(s,T_1\cup\cdots\cup T_n\big)\\
&\overset{(\ddagger)}{=} \sum_{s\in\mathsf{sql}^{-1}(r)}\big(\#(s,T_1)+\cdots+\#(s,T_n)\big)\\
&= \sum_{s\in\mathsf{sql}^{-1}(r)}\#(s,T_1)+\cdots+\sum_{s\in\mathsf{sql}^{-1}(r)}\#(s,T_n)\\
&\overset{(\dagger)}{=} \#(r,\mathsf{sql}(T_1))+\cdots+\#(r,\mathsf{sql}(T_n))\\
&\overset{(\ddagger)}{=} \#(r,\mathsf{sql}(T_1)\cup\cdots\cup\mathsf{sql}(T_n))\\
&= \#(r,\bigcup(\mathsf{sql}(T_1),\ldots,\mathsf{sql}(T_n)))
\end{aligned}$$

where (†) are by the definition of $\mathsf{sql}$ on tables and (‡) are by the definition of the union operation on tables. □