Troubles with Nulls, Views from the Users

Etienne Toussaint
University of Edinburgh
etienne.toussaint@ed.ac.uk

Paolo Guagliardo
University of Edinburgh
paolo.guagliardo@ed.ac.uk

Leonid Libkin
University of Edinburgh & ENS Paris
l@libk.in

Juan Sequeda
data.world
juan@data.world

ABSTRACT

Incomplete data, in the form of null values, has been extensively studied since the inception of the relational model in the 1970s. Anecdotally, one hears that the way in which SQL, the standard language for relational databases, handles nulls creates a myriad of problems in everyday applications of database systems. To the best of our knowledge, however, the actual shortcomings of SQL in this respect, as perceived by database practitioners, have not been systematically documented, and it is not known if existing research results can readily be used to address the practical challenges.

Our goal is to collect and analyze the shortcomings of nulls and their treatment by SQL, and to re-evaluate existing research in this light. To this end, we designed and conducted a survey on the everyday usage of null values among database users. From the analysis of the results we reached two main conclusions. First, null values are ubiquitous and relevant in real-life scenarios, but SQL's features designed to deal with them cause multiple problems. The severity of these problems varies depending on the SQL features used, and they cannot be reduced to a single issue. Second, foundational research on nulls is misdirected and has been addressing problems of limited practical relevance. We urge the community to view the results of this survey as a way to broaden the spectrum of their researches and further bridge the theory-practice gap on null values.

PVLDB Reference Format:
Etienne Toussaint, Paolo Guagliardo, Leonid Libkin, and Juan Sequeda. Troubles with Nulls, Views from the Users. PVLDB, 15(11): 2613 - 2625, 2022.
doi:10.14778/3551793.3551818

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/etouss/NULLSurvey.

1 INTRODUCTION

Managing incomplete data in relational databases has been an academic challenge studied extensively since the early papers of the 1970s [17, 18, 49]. From the theoretical perspective, correctness is usually associated with the notion of certain answers: answers we can be sure about no matter how we interpret the incomplete information present in the database. This approach, first proposed around 40 years ago [35, 40, 49] is now dominant in the literature and it is standard in academic studies where incomplete information appears (data integration, data exchange, ontology-based data access, data cleaning, etc.).

From a practical perspective, to the best of our knowledge, there is a lack of research to understand the expectations of database practitioners with respect to incomplete data. Common anecdotes are the following:

- RDBMSs are criticized for producing counter-intuitive and even incorrect answers when handling incomplete data.
- One often finds statements like "*those SQL features are* [. . .] *fundamentally at odds with the way the world behaves*" [24], or even "*you can never trust the answers you get from a database with nulls*" [23].
- Behavior is blamed on SQL's three-valued logic (3VL); indeed, it is commonly assumed that programmers tend to think in terms of the familiar two-valued logic, while 3VL underlies the implementation of SQL's null-related features.

Our research goal *is to improve our understanding of database practitioner's expectations and approaches to incomplete information in relational databases, namely* **`NULL`** *values*, in order to know how we can further bridge the gap between theory and practice. We wish to validate or refute the common assumptions made by the database research community about the causes and potential solutions to the problem. Towards achieving our goal, we seek to answer four questions:

(Q1) How commonly are SQL's **`NULL`** and related features used?
(Q2) What do nulls mean to users?
(Q3) Are users satisfied with SQL's handling of nulls and, if not, why?
(Q4) Are there readily available solutions to mitigate the problems?

As a first milestone to find answers to these questions, we designed an online survey which ran for four months and attracted 175 participants, with three quarters of them being database practitioners and one quarter academics.

Our contributions are two-fold. The first contribution is the result of the survey and its analysis. Our major findings, in response to the above questions, are summarized below.

SQL's **`NULL`** *features usage.* Our participants acknowledge that **`NULL`** values appear often in their databases. They use some **`NULL`**-specific operators of SQL (most commonly **`IS NULL`** tests), but rely

Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097.
doi:10.14778/3551793.3551818

more on schema constraints to rule out **NULL** values. It appears that **NULL** values are mostly perceived as an inconvenience rather than a feature that one can take advantage of.

Meaning of **NULL** *values.* **NULL** values appear for many reasons, and our participants often ascribe different meanings to them. While there is a near consensus that nulls can represent non-applicable values, only a quarter of respondents think that this is the only interpretation. A majority think that a **NULL** can also represent some unknown value, which may or may not exist. The meaning of **NULL** could also be 0, empty-string, or another constant, but this is rare.

SQL handling of nulls is not satisfactory. SQL's rules for handling **NULL** values are not fully satisfactory. While for simple queries (positive fragment of relational algebra) most of our respondents accept SQL's behavior, for more complex queries, involving either aggregation or negation, many are not satisfied with SQL answers.

No readily available solutions and more research is needed. There is no consensus among the respondents as to what a better behavior of SQL could be. The desired behaviors are diverse, and independent of the users' view of what nulls mean. Some users want the answers to contain more tuples (moving in the direction of possible answers), others want fewer tuples (but with more guarantees), and yet others simply wish for a warning or error message to be given. These different approaches also indicate that the focus of the academic literature, which typically concentrated on the missing value model of nulls and certain answers as the holy grail, is addressing only a very narrow spectrum of the database practitioner's needs.

The second contribution is the survey itself. While the responses we collected and analyzed can only provide indications, the scope of the mismatch between the common assumption made by the research community and the participants' answers provide strong evidence that a problem does exist and deserves to be studied with more resources. Obtaining definitive answers to questions (Q1)–(Q4), requires access to a larger sample of participants and a more in-depth protocol that can include random sampling, interviews, web-data analysis (Twitter, Stack Overflow, Reddit, etc.) [13, 26]. Our questionnaire design can serve as the basis for such further studies. Moreover, the design of a social study experiment necessitates some prior information about the target population to minimize bias and maximize efficiency; our survey design and analysis methodology allow to gather such information [25]. An obvious step in this direction would be to run our survey on a larger sample, although we note that our sample of 175 respondents is twice as large as those that have previously been seen in database research of this kind [58] and has a higher proportion of practitioners. A different possibility is to run the survey on a more focused sample (e.g., database professionals working in a particular industry, specific DBMS users) to study the problems that different application domains may experience and reduce non-response bias [9]. For that reason, the survey and the software for analyzing its results have been made available in the GitHub repository.

Organization. Section 2 describes the methodology behind the design of the survey. Sections 3–6 analyze answers to questions (Q1)–(Q4). Section 7 presents our conclusions and Section 8 gives recommendations to the research community based on these conclusions.

2 DESIGN AND METHODOLOGY

In this section, we explain the design of the survey and the methodology used to analyze each type of question. We then describe how participants were recruited, and we analyze the respondents' demographic and their level of engagement with the survey.

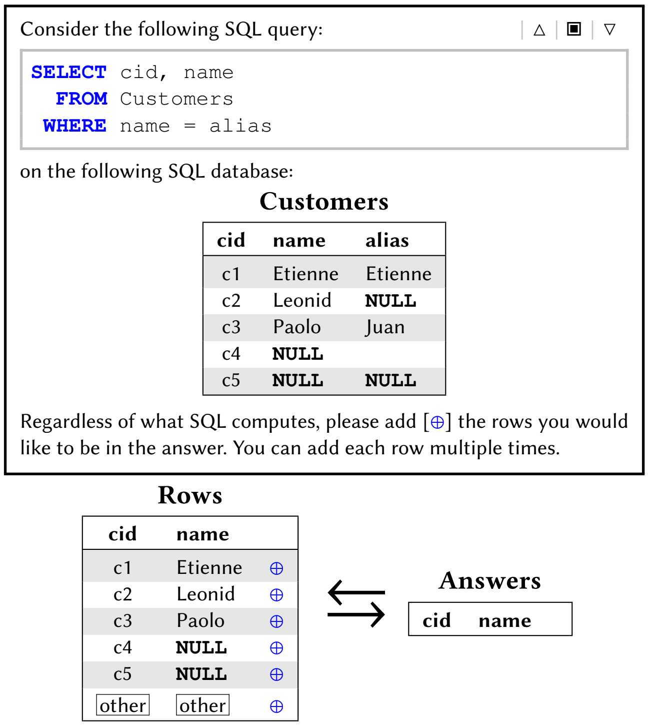
2.1 Question Types and Analysis Methodology

The survey is an online structured questionnaire consisting of 34 items. To reduce bias and improve the experience of respondents, the design of the survey has been reviewed by a selected group of practitioners and academics. However, biases are inherent to the survey tool, and while some are reduced by our process of answers analysis, we advise readers to discuss our findings.

Multiple-choice. These questions were used when there is a fixed number of possible answers, with an option labeled "other" always made available as a fallback. For this kind of questions, we wish to study the prevalence of choices; therefore, for each available option, we compute the proportion of respondents who selected it. When a question allows more than one option to be chosen, we also compute the proportion of participants who selected each subset of all available options. In our analysis, the data collected through multiple-choice questions is mostly used to get demographic information about the participants, and the results are presented as a pie or bar chart.

Frequency scale. These are "how-often" questions whose answer is a frequency chosen from an ordinal Likert scale [48] with options *never*, *infrequently*, *occasionally*, *often*, and *regularly*, which are ordered from the least to the most frequent. Answers to different frequency scale questions can be compared with one another, but we cannot assume that the frequency differential between each subsequent option in the scale is constant; thus, numerical analysis that computes an average frequency score on a single question is largely meaningless [57]. For each such question and each frequency option, we compute the proportion of participants who selected that option in the question under consideration. As the number of available options is limited, these statistics can be effectively exploited. Moreover, since the same scale is used for several questions, we can compare the relative frequency of some events. The data collected through frequency scale questions is mostly used to obtain information about the participants' demographic as well as to answer question (Q1).

Build the answer. In these questions (Figure 1a) participants are presented with a relational database and an SQL query, and then asked to construct, by adding default or custom rows to the output table, *the answer they would like to obtain.* This design allows us to gather information about our respondents' expectations from the evaluation of each query. From an abstract point of view, this type of question is a multiple-choice question. The default rows can be seen as the choices of the question, and adding them to the answer is equivalent to selecting the choice. However, due to the nature of the task, even if the prevalence of each row can be of interest, we argue that more emphasis should be put towards SQL's



Consider the following SQL query:

```
SELECT cid, name
  FROM Customers
 WHERE name = alias
```

on the following SQL database:

Customers

cid	name	alias
c1	Etienne	Etienne
c2	Leonid	NULL
c3	Paolo	Juan
c4	NULL	
c5	NULL	NULL

Regardless of what SQL computes, please add [⊕] the rows you would like to be in the answer. You can add each row multiple times.

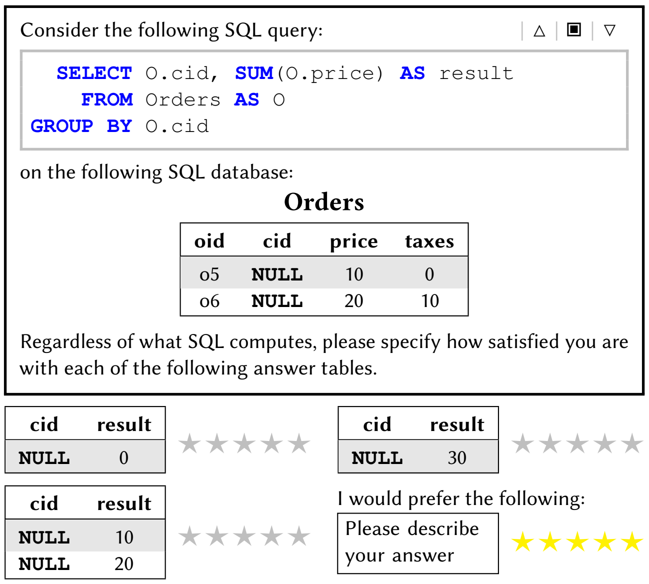
Rows

cid	name	
c1	Etienne	⊕
c2	Leonid	⊕
c3	Paolo	⊕
c4	NULL	⊕
c5	NULL	⊕
other	other	⊕

Answers

cid	name

(a) Build the answer



Consider the following SQL query:

```
   SELECT O.cid, SUM(O.price) AS result
     FROM Orders AS O
GROUP BY O.cid
```

on the following SQL database:

Orders

oid	cid	price	taxes
o5	NULL	10	0
o6	NULL	20	10

Regardless of what SQL computes, please specify how satisfied you are with each of the following answer tables.

cid	result
NULL	0

☆☆☆☆☆

cid	result
NULL	30

☆☆☆☆☆

cid	result
NULL	10
NULL	20

☆☆☆☆☆

I would prefer the following:

Please describe your answer

★★★★★

(b) Likert interval scale

Figure 1: Examples of questions in our online survey.

default answers. Therefore, we partition the participants' answers into four groups, according to whether the answer (1) matches the SQL answer, (2) is a subset of the SQL answer, (3) is a superset of the SQL answer, or (4) is not comparable with the SQL answer. We then compute the proportion of answers in each of the groups and report the results in a table. The data collected through this kind of questions are mostly used to answer the question (Q3).

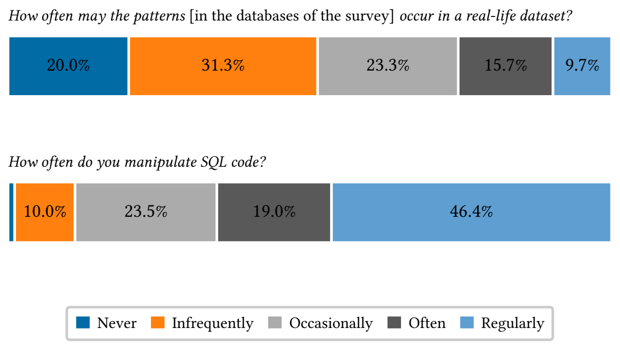
Interval Likert scales. In these questions (Figure 1b) participants are presented with a relational database, a *value-inventing* SQL query (i.e., producing a value not already present in the database, such as an aggregate value over a column) and several tables. The task is to score each table with a value between 0 and 5 stars (in half-star increments) based on how satisfied the participant would be if the scored table were the answer to the query on the given database. This design lets us gather information about our respondents' expectations for the evaluation of value-inventing queries. Each option is displayed with an initial satisfaction score of 0. The respondents can also use an option labeled "other", with an initial score of 5, to provide a better alternative to those presented. We only consider the score of a custom answer if the participant has provided one. For each query we report the average and the quartile values of the satisfaction score obtained by the SQL answer, and we also compute the proportion of participants who would be more satisfied with a different answer. This data is used to answer the question (Q3).

In designing a survey which asks the respondents to evaluate or construct query answers, it is important to ensure that the sample relational databases represent realistic real-life scenarios rather than data patterns that are unlikely to occur. The latter would render the results of the survey less valuable. To this end, for each of the three databases used in the survey, we asked the participants the following frequency question: *How often may the pattern in the given database occur in a real-life dataset?* As shown in the summary (the average for the three databases used) of responses in Figure 2a, 80% of our respondents are of the opinion that such patterns can occur in real-life databases, with a varying degree of frequency; about half are of the opinion that such patterns occur frequently.

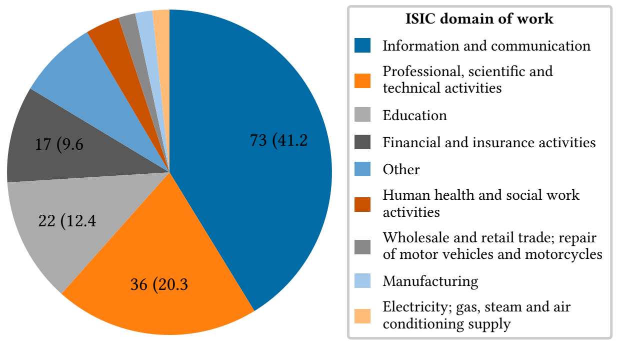
For all types of questions, depending on the quality of the respondents' sample, the results can be either considered as a representation of the general population (high-quality sample) or, in the case of a small sample, used to identify trends or outliers (choices for multiple-choice questions, events for frequency scale questions, and categories for build the answer and interval Likert scale questions) [25]. We will discuss the results we obtained from the participants we managed to recruit, but since the quality of the conclusions one can draw is heavily dependent on the size and quality of the sample, anyone is encouraged to run the survey and its analyses on their own community. To this end, both the survey and the code are publicly available in the GitHub repository. Moreover, throughout the discussion and analysis of our results, we acknowledge (in retrospect, having analyzed the results) that some elements of the design could have been done differently, be improved; readers interested in conducting the survey are advised to contact the authors for further details.

2.2 Sample of Respondents

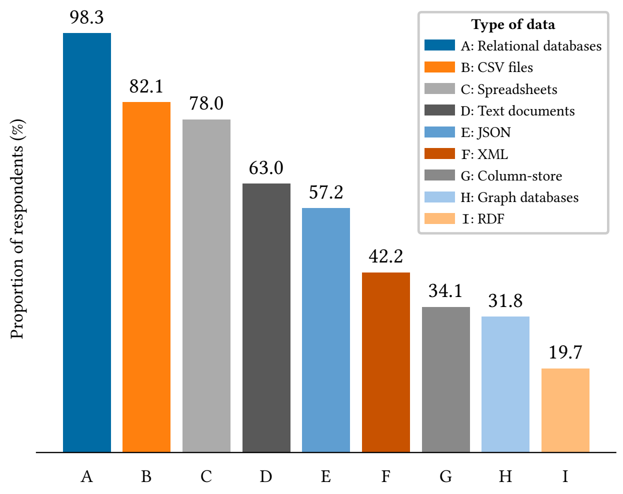
We conducted the survey during a four-month period, and used several different methods to recruit participants. First, we posted a summary of our research on the data.world blog [59] to advertise the survey. We also sent both the survey and the blog post to the data.world mailing list through the monthly digest, reaching their



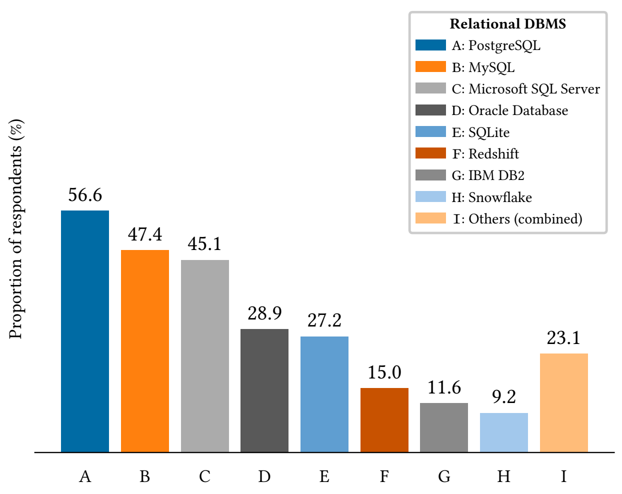
(a) Database patterns and manipulation of SQL code.



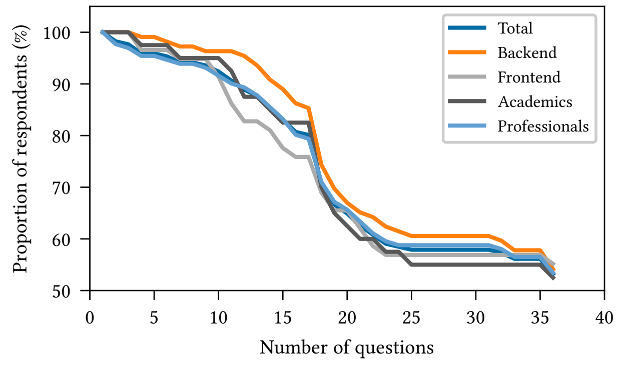
(b) Participants' ISIC domain of work.



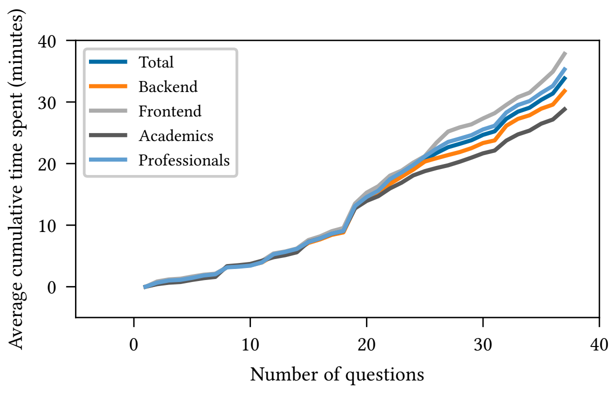
(c) Types of data users work with.



(d) Popularity of Relational DBMSs.



(e) Proportion of respondents for each question.



(f) Average time spent on each question.

Figure 2

user base. We used mailing lists, such as dbworld, as well as Reddit posts, and several Slack channels dedicated to specific DBMSs.

All participants accessed the survey using the same online link, so we cannot tell how many participants were recruited via each individual channel. In the end, there were 175 participants who took the survey, among which 94 completed it in full. Below, we discuss the participants' demographics and how their engagement with the questions evolved during the completion of the survey.

Demographic information. We first asked the participants which domain they work in, according to the ISIC classification [53]. As shown in Figure 2b, the participants indicated 14 different domains, demonstrating that nulls matter for a wide variety of fields.

Next, we asked the participants what best describes their role in their organization. According to the answers, we split participants into “practitioners” and “academics”, where the latter are those who chose *Education* as a domain of activities, or *Professor* or *Student* as a role in their organization. As intended, the sample has a prevalence of practitioners (73%), which our study is geared for.

We asked how often participants manipulate SQL code, to assess their familiarity with the language itself. The answers are shown in Figure 2a. We split our participants into two groups:

- *Front-end users*, who manipulate SQL code only occasionally or infrequently; they make up 34% of the respondents (with 1% saying never).
- *Back-end users*, who manipulate SQL code often or regularly; they account for 65% of the respondents with the largest group, almost half, saying regularly; thus, we have reached our target audience.

Finally, we asked what type of data and which relational DBMSs our participants use. As shown in Figure 2c and Figure 2d, they mostly deal with relational data, and use a variety of systems, with a preference for PostgreSQL, MySQL, and Microsoft SQL Server (followed by Oracle Database, SQLite, and others).[1]

Participants’ engagement. The complexity and the extent of our study was too ambitious to be kept within the recommended 10-minute survey format [56]. Despite our best efforts, we knew that our survey would take longer than that to complete. It is surprising that so many participants spent so much time on it. Figure 2e and Figure 2f show the number of participants and the average time they spent on the survey, respectively; we report the results for each question and each group of participants.

We observe that the engagement is similar for each group of participants, and a significant drop-off in the number of respondents occurs after question 18, which in fact corresponds to the 10-minute mark of average time spent on the survey. In retrospect, the design of question 18 could have been improved: although not complex to answer, it is bulky and cumbersome, and this may have discouraged some participants from continuing.

We also observe that, despite the average completion time being rather high (around 45 minutes), a high proportion (60%) of participants finished the survey. This shows that database practitioners have a strong interest in the problem of nulls in SQL.

3 SQL’S NULL FEATURES USAGE

The importance of studying nulls stems from their ubiquity in everyday applications; this was confirmed by the survey. Figure 3a shows how frequently users encounter nulls; more than 80% of them see **NULL** often or regularly, and less than 20% fall into the *infrequently* and *occasionally* categories. Even if **NULL** does not appear in a dataset, it can be generated by queries, in particular outer joins. Figure 3d shows that users frequently deal with **LEFT**

[1]The survey erroneously offered both “DB2” and “IBM DB2” as options; 20 respondents chose at least one of them, with 4 selecting both, so we considered 16 as the combined total.

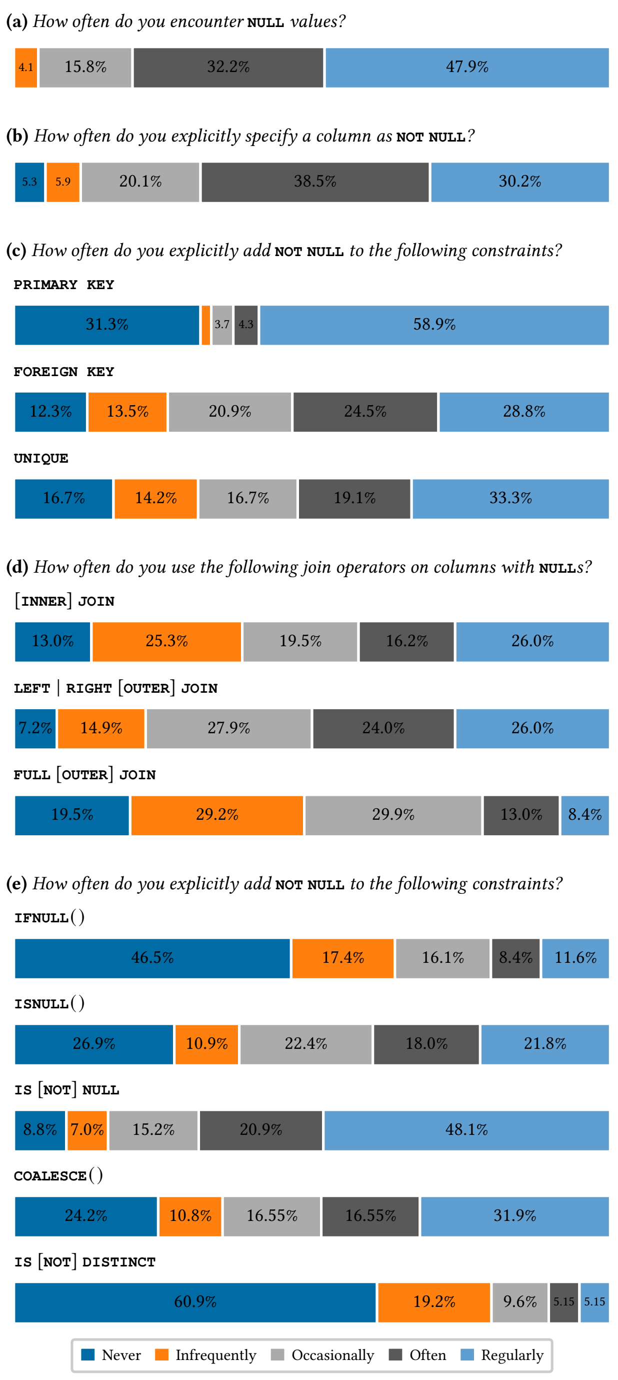


Figure 3

and **RIGHT JOIN**; in fact, these are more common than explicitly specified inner joins. Full outer joins are also quite common.

The features offered by RDBMSs to handle **NULL** can be subdivided into the following two categories:

- SQL’s DDL constraints, such as **NOT NULL**, to prevent **NULL** from appearing in columns;

- null-specific tests and functions, such as `IS [NOT] NULL`, `IS [NOT] DISTINCT`, `COALESCE()`, `ISNULL()`, and `IFNULL()`.

Figure 3b and Figure 3c show the prevalence of using null-prohibiting constraints in the DDL. The use of `NOT NULL` is very common, with almost 70% of respondents regularly or often declaring columns, and just over 5% avoiding the practice. These are also frequently added to keys, foreign keys, and `UNIQUE` declarations. It is interesting to note that, in the case of primary keys, `NOT NULL` is superfluous; yet, the majority of users nevertheless includes it explicitly. Adding `NOT NULL` to foreign keys also appears to be a common practice, in all likelihood aimed at avoiding three-valued logic in joins. On the other hand, it is also noteworthy that a non-negligible minority, around a quarter of respondents, never or almost never use null-prohibiting declarations.

In Figure 3e we see how commonly SQL's null-related features are used in queries. By far the most common one is checking whether a value is `NULL`. To this end, the `IS NULL` condition, or its negation, are used by almost 70% of the users, often or regularly, and completely avoided by fewer than 10%. The next most common feature is `COALESCE`, with almost half the respondents using it at least often; other operations are less common. We remark that `ISNULL()` and `IFNULL()`, as opposed to `IS [NOT] NULL`, are non-standard functions only available in some systems.

Conclusion: `NULL` is a common occurrence in relational databases, and users are well aware of it. This manifests itself most commonly in the use of SQL's DDL, by frequently declaring some columns as `NOT NULL`. Null-related features are also fairly common in queries.

"The prevalence of dirty and missing data ought not to be underestimated. For many years, I was in charge of systems for data collection of medical data. Even such a regularized domain frequently had problems managing missing, dirty and suspicious data."

A participant

4 MEANINGS OF NULLS

Now that we confirmed the ubiquity of nulls and analyzed operations on nulls, both in the DDL and the query language, we move to the next question: what does `NULL` mean? The question has been addressed in the research literature, going all the way back to [65] which defined three types of nulls: non-applicable, no-information, and those representing existing but currently unknown values (two of those, non-applicable and unknown values, were adopted early by relational databases [19]). To see what options we can give to the users asking them a multiple choice question on the meaning of nulls, consider a hypothetical example: we have a table with information about employees, and the salary of the CEO is given as `NULL`. This could have different meanings:

- *Non-applicable* (**NA**). The CEO may not receive a regular salary and use another remuneration scheme. Then `NULL` indicates a field that is non-applicable, for which a value does not exist.
- *Existing unknown* (**EU**). The CEO salary cannot be disclosed for privacy reasons. In such cases, `NULL` denotes an existing but currently unknown constant.
- *Existing known constant* (**C**). The CEO salary may not be disclosed because it depends on changing financial results of company operations. Here `NULL` denotes an existing, and known, value.

Table 1

Sem.	Options presented in the survey
NA	*"the value does not exist"* *"non-applicable field"*
EU	*"the value exists and could be anything"* *"the value exists and is equal to an unknown constant"*
C	*"the value exists and is equal to a known constant"*
D	*"there is a bug"* *"the data is dirty"*
NI	*"nothing is known about the value"*

- *Dirty* (**D**). The CEO may receive a regular salary but the data source from which the table is populated may have been dirty.
- *No-information* (**NI**). We may be in a situation when we know nothing at all about the reasons why that `NULL` is in the database.

While the different semantics described above are easy to understand in a particular example, a single universally accepted description is not particularly easy to formulate in natural language. Thus, we introduced some redundancy with the options we presented to the users, as shown in Table 1.

Figure 4a provides the frequencies with which the participants chose each option describing a possible meaning of `NULL`. The **NA** semantics is the most popular one, selected by over 85% of respondents. It is followed by **NI**, which was chosen by more than 60% of the respondents. The **EU** semantics is considered by nearly 40% of the respondents. Finally, the **D** and **C** semantics were chosen by nearly 35% and 20% of the respondents, respectively.

The data in Figure 4a does not provide any information about *combinations* of different semantics chosen by the participants. If we take that into account, we obtain 23 different groups, as shown in the pie chart of Figure 4b. Not surprisingly, **NA** dominates: alone or in combination with **NI** and **NI+EU**, it accounts for more than half of all the combinations of the semantics seen. We remark that 13% of participants proposed an alternative semantics that does not coincide with any of those we have discussed (the curated comments and custom answers can be found in the GitHub repository). In light of this, our list of possible interpretations of `NULL`, while not exhaustive, covers most of the cases one meets in real-life situations.

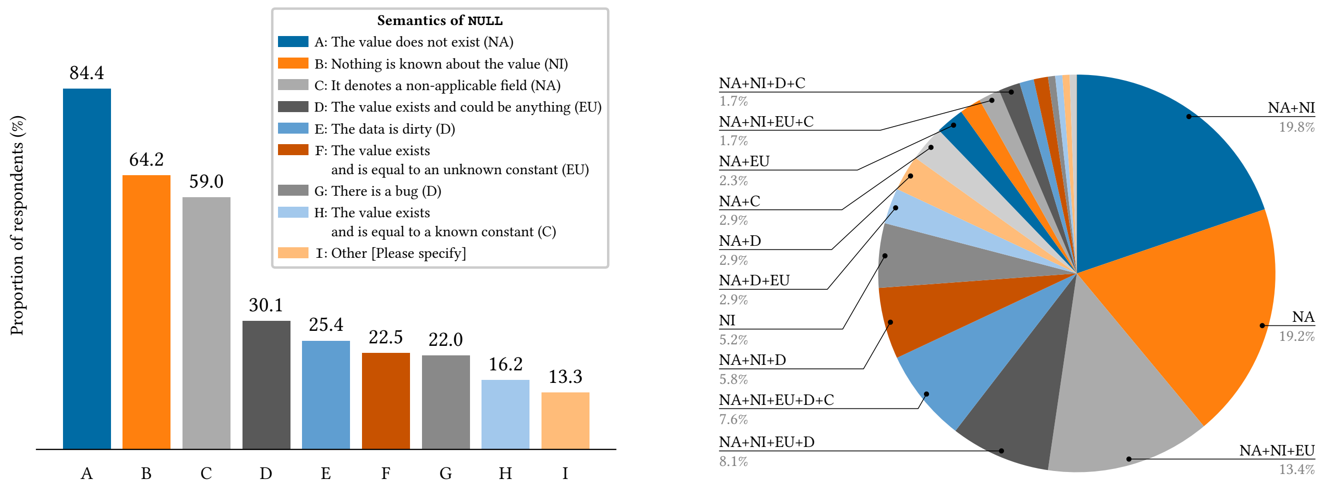
Conclusion: The meaning of a `NULL` value is highly varied, and therefore there is no consensus on any one specific interpretation. However, the non-applicable semantics seems to dominate.

"It is impossible to know what NULL means without understanding the intent of the one who put it there in the first place. Unfortunately, I've seen it used for all of the above."

A participant

5 SQL'S HANDLING OF NULLS

In order to understand SQL's handling of `NULL` we organize our findings in two parts. Section 5.1 addresses data manipulation queries that stay within the standard textbook version of relational



(a) (b)

Figure 4: (a) Popularity of different semantics of `NULL`; (b) Combinations of `NULL` semantics chosen by the participants.

algebra, known as *generic* queries [3, 5]. These types of queries do not generate new values by means of function application or aggregates. On the other hand, Section 5.2 focuses on how **`NULL`** is dealt with when applying arithmetic functions and aggregates.

5.1 Generic Queries

We look at three different aspects of such queries with respect to **`NULL`** handling: (1) comparisons involving **`NULL`**, (2) positive queries without **`NOT`**, and (3) negative queries using **`NOT`**.

The latter are more likely to cause issues given what we know from the literature [44, 46], due to the way the 3-valued logic is handled in query evaluation.

Comparisons involving **`NULL`**. We would like to see whether survey participants were satisfied with the way SQL evaluated comparisons between **`NULL`** and

- a constant equal to: 0 for numerical types, and empty string for character types;
- another constant;
- another **`NULL`**.

We use the results of four build-the-answer questions (Figure 1a) to study both equality and disequality comparisons, and are interested in the percentage of participants who *disagree* with the way SQL handles such comparisons; the higher the value is, the more problematic such a comparison. The results, summarized in Table 2, show that the desired behavior does not depend on the data-type. Moreover, contrary to a popular belief neither the empty string or the integer value 0 are treated differently than any other constant (even though the former is used as a substitute for **`NULL`** in some RDBMSs). Around 20% of our respondents would want SQL to view two **`NULL`** values as equal (i.e., use syntactic equality for them), while around 10% would like them to be different. Finally around 40% would like for SQL to evaluate disequality comparisons between **`NULL`** and a constant to *false* rather than *unknown*.

Table 2: Proportion of answers that differ from SQL.

Operator	(a): 0 or ''	(b): Constant	(c): **`NULL`**
Equality String	5.1	3.8	**22.2**
Equality Integer	3.3	2.6	**21.5**
Inequality String	**41.6**	**48.7**	11.0
Inequality Integer	**41.6**	**43.0**	8.72

When it comes to queries, we aim to learn whether the answer the participants would like to see returned in the presence of **`NULL`**

- coincides with the SQL answer, or
- contains the SQL answer, or
- is contained in the SQL answer, or
- is incomparable with the SQL answer.

Positive queries. In these queries there is no negation; they are built using joins as well as **`IN`** and **`EXISTS`** subqueries. For each individual query testing a particular feature, more than 90% of our participants say that the answer should be the same as the one SQL returns. At the same time, the percentage of participants who say that the result should be the same as SQL for *every* query using those features is 84%. If we consider the whole positive fragment of SQL generic queries, and therefore also take into account the answers obtained with filtering queries on equality conditions (Table 2), this percentage drops to 68%. Thus even for fairly uncontroversial queries it is hard to find a uniform agreement across a query workload. But things quickly get much more problematic when we consider the interaction of **`NULL`** with queries that involve negation.

Table 3: Proportions of respondents' answers vs SQL.

Query	A=SQL	A⊂SQL	SQL⊂A	A≠ SQL
`NOT IN`	15.4	0	**84.5**	0
`NOT EXISTS`	78.3	15.8	1.7	4.1
`EXCEPT`	53.0	10.4	**27.0**	9.6
`EXCEPT ALL`	10.7	**76.8**	8.9	3.6

Queries with negation. When queries have negation, the disagreement with SQL query results increases significantly. In the survey we used queries containing **NOT IN** and **NOT EXISTS** subqueries, as well the difference query expressed either via **EXCEPT** under set semantics or **EXCEPT ALL** under bag semantics. In Table 3 we report the views of the participants under the four possibilities shown above, with A standing for the answer a participant wanted, and SQL standing for the answer returned for by SQL. For example, the first row in that table says that for a query with a **NOT IN** subquery, 15.4% of participants agree with the result that SQL returned, while 84.5% of participants preferred an output that was a proper superset of SQL's query evaluation.

Looking at **NOT IN** and **NOT EXISTS** subqueries, we see that a large majority are unsatisfied with **NOT IN** and think that it filters out too many results, while the users are generally satisfied with **NOT EXISTS** subqueries. In fact those who are unsatisfied with **NOT IN** and want it to output more results, want it to behave in the spirit of **NOT EXISTS**. We note that **IN** subqueries follow the rules of three-valued logic (the condition in **WHERE** could evaluate to *unknown*) while **NOT EXISTS** subqueries follow the familiar two-valued logic (only *true* and *false* are possible), thus suggesting that the users are more comfortable with two truth values.

Regarding **EXCEPT**, we see that there is a significant variation in results wanted by the users. Recall that set operations treat **NULL** syntactically, so for the purpose of these operations two **NULL** values will be equal, even if testing them explicitly for equality does not return *true*. This sudden change of the semantics may well be at play here. The results for **EXCEPT ALL** seem to be more about handling duplicates than handling nulls: analyzing them in more detail we see that most outputs preferred by survey participants match SQL answers except for duplicates. The desire to have a smaller result, expressed by over 75%, appears to be the desire to eliminate duplicates from the result.

5.2 Value-Inventing Queries

We now look at queries that may output values not found in a database. Examples of such would be queries whose **SELECT** clauses contain arithmetic expressions like `A+B` or aggregates such as **SUM** or **AVG**. For such queries we want to see how satisfied the participants are with the results that SQL produces, and whether they have alternatives. Thus, we ask about users' expectations with respect to queries that evaluate expressions such as `30`+**NULL** or **NULL**+**NULL** (whose values, returned by SQL, is **NULL**), or the same expression in the **SUM** aggregate (which, perhaps confusingly for some users, would return 30 for **SUM**(30,**NULL**) and **NULL** for **SUM**(**NULL**,**NULL**)).

The results are presented in Table 4. In the left column we have a key feature of a query that depends on handling nulls. For example, we may ask how adding two nulls will behave (**NULL**+**NULL**) or adding a null and a constant (**NULL**+30), or how the same operations are performed under the aggregate **SUM** (**SUM**(**NULL**,30) and **SUM**(**NULL**,**NULL**) respectively). We also ask about queries in which the summation aggregate is applied to values obtained as the result of arithmetic expressions (**SUM**(**NULL**+30,**NULL**+0)), and look at queries with **GROUP BY** where grouping is done on an attribute whose value is **NULL**.

In all of these questions we ask the participants to tell us how satisfied they are with SQL answers on the 0-to-5 scale, with 0 being the least satisfied and 5 the most satisfied. The additional columns in Table 4 have the following meaning:

Quartiles/Median This is a box plot displaying the minimum, first quartile, median, third quartile, and maximum. We draw a box from the first quartile to the third quartile. A vertical line goes through the box at the median. The whiskers go from each quartile to the minimum or maximum. For example, looking at the first row of Table 4, we see that 75% of respondents gave answers 3, 4, or 5, indicating that they were rather or very satisfied with the fact that SQL evaluated 30+**NULL** as **NULL**. The next row shows that more than 75% were satisfied with the way SQL evaluated **NULL**+**NULL**. The vertical bar in those results gives the *median* answer to the question; when it is not visible within the blue-colored rectangle, it means the median is 5 (complete satisfaction).

Mean This column provides the mean answer.

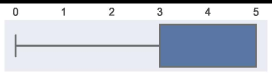
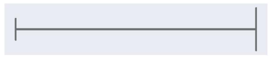
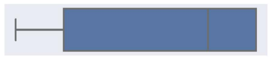
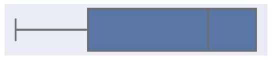
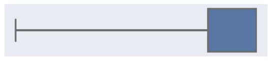
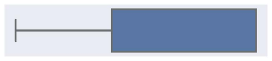
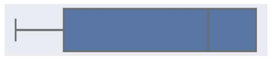
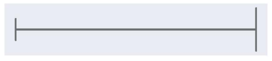
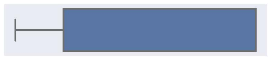
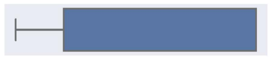
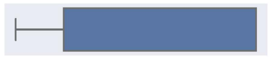
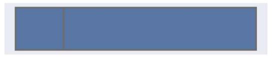
Agreement and Controversy This is our informal interpretation of the results; agreement indicates to what extent people agree with SQL answers and controversy indicates how much of the difference of opinion there is; the possible scores are high, medium, and low. The higher the average and the median score are the higher the agreement of the participants is. The larger the difference between the median value and the lower quartile value is, the higher is the controversy. For example, in the first row of Table 4, there is generally high level of agreement as the mean and median tend to be high, with a moderate amount of controversy as for 75% of participants they range from 3 to 5.

Alternative Participants were asked to score several tables for each query (Figure 1b). The "Alternative" column gives the percentage of respondents who scored the table produced by SQL lower than (at least one of) the others.

We see from Table 4a that the way SQL applies functions whose arguments could be **NULL** (using the rule that the value of any such function is **NULL** then) is fairly uncontroversial and accepted by most. The agreement with SQL's way of handling **NULL** slightly decreases when aggregates are applied to columns containing **NULL** (in which case the rule is not to output **NULL** but rather to ignore all nulls and then compute the aggregate). In the case of aggregates, a significantly higher proportion of participants would be more satisfied with an alternative answer. Most disagreement with SQL occurs in the case of queries with grouping on a column that may contain **NULL**. In this case SQL uses the syntactic equality of nulls, i.e., two **NULL**s are equal, which of course differs from the treatment of **NULL**s elsewhere and thus causes additional problems.

We then look at similar queries but with one important difference. Now **NULL**s are not present in the database but rather created by an

Table 4: Results obtained with (a) value-inventing queries and `NULL`s that occur in the database, and (b) value inventing queries and computational `NULL`s.

Query	Quartiles/Median	Mean	Agreement	Controversy	Alternative	
30 + **NULL**		3.8	high	medium	21.6%	
NULL + **NULL**		4.4	high	low	2.1%	
SUM(30, **NULL**)		3.4	medium	high	33.3%	
MIN(30, **NULL**)		3.4	medium	medium	30.5%	(a)
SUM(**NULL**, **NULL**)		4.1	high	low	9.5%	
SUM(30 + **NULL**, 0 + **NULL**)		3.6	high	high	23.2%	
GROUP BY(**NULL**, **NULL**)		3.1	medium	high	29.5%	
NULL + **NULL**		4.3	high	low	7.5%	
SUM(30, **NULL**)		3.1	high	high	35.5%	
MIN(30, **NULL**)		3.3	high	high	31.5%	(b)
SUM(**NULL**)		3.2	high	high	37.9%	
GROUP BY(**NULL**, **NULL**)		2.0	low	high	63.2%	

OUTER JOIN query. Then we ask questions about similar queries involving arithmetic, aggregation, and grouping on **NULL**. Results, in the same way as before, are presented in Table 4b. While the trends are the same, we see three notable differences. First, the level of agreement with SQL results is higher: the median in all the cases except grouping on **NULL** moves to the highest score 5. Second, the level of controversy increases too (indicated by the widened blue shaded area in the diagram: more different scored are picked by more participants). And third, the percentage of users who prefer an answer that differs from SQL's is now much higher and peaks (at almost two thirds) for the grouping on **NULL**.

Conclusion: SQL's way of handling **NULL**s for the positive generic queries satisfies a vast majority of the users. For negative queries, the amount of dissatisfaction increases significantly. The users do not seem to like that SQL's 3VL filters out so many tuples. Users are reasonably accustomed to SQL's way of handling **NULL**s in arithmetic operations, and slightly less so in aggregation, where the rule changes and **NULL**s are ignored. Their level of dissatisfaction increases when **NULL**s come not from a database but generated by a query. What they do not like is the use of syntactic equality (saying that two **NULL**s are equal) for generating groups using **GROUP BY**.

"The NOT IN vs NOT EXISTS trap does nail people in real life, fairly often. I understand why it is the way it is, but I wish SQL had defined NOT IN as a syntactic transform to NOT EXISTS. GROUP BY NULL should cause the user and/or data designer to vanish from existence immediately."

A participant

6 ARE THERE READY SOLUTIONS?

The results of our survey revealed both agreement and significant disagreement with SQL's handling of nulls, as well as much variety when it comes to the interpretation of **NULL**. The first question we address here is whether some of the users' dissatisfaction can be tackled with solutions from the existing academic literature. Towards that goal, we give a quick overview of existing work and conclude that most academic research was focused on issues that are of rather limited practical interest.

With this knowledge, we asked ourselves whether there could be an association between the participants' demographic data (industry vs academia, familiarity with SQL, etc.), their view of the semantics of **NULL** and their preferred query behavior. To much of our surprise, this data analysis (using the standard statistical and machine learning toolkit) revealed no correlation whatsoever.

6.1 Existing Work

The subject of incomplete data has been addressed in the literature, starting from foundational papers in the 1970s [17, 35, 49], and several surveys are available; e.g., [22, 36, 63]. In this overview of related work, we outline what has been done for the **EU** semantics of nulls, which is the subject of most of the material in the aforementioned surveys, and concentrate on the interpretations that users found most common.

***EU** and certain answers.* Theoretical work on incomplete databases has traditionally focused on the interpretation of nulls as missing values, in the sense of the **EU** semantics: a value exists, but it is currently unknown. The latest survey in that area is only two years old, and we refer the reader to it for multiple additional details and references, while we here outline only the basics. The prevalent model in this line of research is that of so-called *marked nulls*, first systematically studied in [40], where each missing value is represented by a special null symbol with an associated identifier.

There is a clear mismatch between the model of nulls used in theoretical database research and that of SQL; even if we interpret SQL nulls solely under the **EU** semantics, marked nulls are more expressive, because they allow co-reference of nulls, which is not possible in SQL. It is folklore in the database theory community that SQL nulls can be modeled by non-repeating marked nulls (i.e., with distinct identifiers). This, however, is not the case, as was demonstrated in [38]. Nonetheless, the marked nulls model dominated not only research on incomplete information, but also work in areas where incomplete databases naturally arise. Examples of these include ontology-based data access [55] and data exchange [27]. We refer the reader to specific surveys of these areas [6, 10], with hundreds of further references, and detailed explanations of the usage of marked nulls under the **EU** interpretation.

For marked nulls, the preferred model of query answering is that of *certain answers* [40, 49]. These are answers that are true under all interpretations of nulls, i.e., all possible ways of assigning known values to them. The definition of certainty itself is not unique [45, 50], but in all cases query answering is computationally hard; e.g., coNP-complete for relational algebra queries [4]. The standard query evaluation (a.k.a. *naïve*), where nulls are treated like regular constants, does sometimes produce answers that come with certainty guarantees; this is the case for unions of conjunctive queries (i.e., select-project-join-union queries) [40], or their extension with relational algebra division [33]. Given the general hardness of computing certain answers, recent efforts concentrated on finding efficient approximations [28, 29, 37]. However, as seen from the results of our survey, users do not seem to be preoccupied with certainty (in fact, they tend not to like it when too few answers are given, which is a tendency with the notion of certainty). Furthermore, there are no developed algorithms, nor even a generally accepted definition, for extending certainty to queries with aggregation, which are extremely common in applications.[2]

[2] A well-founded definition of certain answers for value-inventing queries under bag semantics was only recently proposed in [61].

Thus, despite much solid research in this area, its outcomes are often not directly applicable to how nulls are seen in the real world.

Nulls in SQL. Even prior to the development of SQL, an ANSI committee working on the then new relational model suggested 14 different interpretations of nulls [1]. Most of these fall into the categories our respondents provided, with some exceptions that have been made obsolete by subsequent developments. For example, access policies are managed explicitly and not via specific nulls, while some of the earlier interpretations implicitly assumed manual data entry. While [1] did not provide any semantics of operations on nulls, nor a logic for their handling, this was suggested in two later documents [2, 16] from the SQL standardization process; these proposals date back to the late 1980s, but no action with respect to the Standard was taken. They proposed user-defined nulls, a model moving in the direction of marked nulls, with user-defined semantics, and SQL's **`NULL`** as the fallback position.

As already mentioned, shortcomings of SQL's **`NULL`**s are anecdotally known in the community, with only few exceptions trying to systematize them in the research literature. Among those are [11], which looks into issues experienced by SQL programmers in general, not with nulls directly, and [54], which studies how aggressive optimization techniques result in incorrect query evaluation in the presence of nulls.

***NA** and **NI** nulls.* While we have seen that the **NA** and **NI** interpretations of nulls are the dominant ones, research on them is not nearly as extensive as for the **EU** semantics. Some early work on query evaluation with nulls under the **NI** semantics was done in [65], which proposed the trichotomy of **NI**, **NA**, and **EU** nulls. This was followed by work on **NA** in [43], essentially treating them as problems with the database design. There are different lines of work on **NA** and **NI**, each going in its own direction. For example, [34] extended truth tables and arithmetic operations to the **NA** null, and [8, 39] looked at functional dependencies over databases with **NA** and **NI** nulls. One direction of work is to capture the semantics of different nulls by extending the 3-valued logic of SQL with more values and even paraconsistent logics [7, 20, 32, 60, 64]. However it was shown recently that SQL's 3-valued logic is the most rational choice of a propositional logic that does not hamper query evaluation [21].

On the query evaluation side, we mention [15, 30, 42]. The approach of [42] is based on the notion of subsumption of tuples with nulls, already present in [14, 65], and defined the semantics of **EU** and **NI** by different ways of lifting this subsumption from tuples to relations. This led to a well-defined notion of query answering and approximation for the **EU** semantics, but not for the **NI** semantics, where the discussion stopped at the level of investigating a number of concrete SQL queries. A step further was made in [15], which looked at **NI** added as either a substitute for a value, or a disjunction of several values (this, by itself, is a model of incompleteness known as *or-sets* [41, 47]). These can be expressed by means of extra constraints attached to tuples in the spirit of conditional tables [40]. The paper shows then how to use such conditions to evaluate relational algebra queries; the result is again a conditional table, which can complicate understanding the output from a user's perspective. Last but not least, [30] formally shows that answering queries (in relational algebra and calculus) on databases where nulls are interpreted under the **NA** semantics is equivalent to naively evaluating a rewritten query on a decomposed database without nulls, in which the "absence" of values is implicitly encoded in the schema. From a practical point of view, however, no concrete implementation and experimental evaluation of such rewriting and decomposition algorithms were devised.

***EU** and **D**: imputation.* A well-established line of research that concerns these two interpretations (which together occur in about 20% of our respondents' answers) is to impute null values. That is, nulls are replaced with actual constants according to some statistical model. In the end, this gives us a database without nulls that can be queried and analyzed assuming the data is clean and complete. Most of this work is seen in the statistical and data cleaning literature [31, 62] and these approaches are fundamentally geared towards machine learning tasks on data derived from a database.

To summarize, most research efforts of the last four decades have been directed to the investigation of semantics that, while appearing in applications, are not the most common ones in the eyes of database practitioners.

6.2 Solutions vs Demographics

We now move to the next question: whether there exists any correlation between the demographics for our respondents, and their preferences for the behavior of SQL queries. To address this, we performed a statistical analysis of the received responses. It revealed that, somewhat surprisingly, *there is no correlation* between the users' demographics and their responses to the questions about SQL. This makes the task of fixing SQL even more daunting, if at all possible.

To perform the analysis, we defined eight binary parameters that take into account users' information (academic vs practitioner, front-end vs back-end) and the preferred semantics of **`NULL`**; we refer to these parameters collectively as "demographic data". We looked at how they correlate with participants' agreement with SQL answers for each of the queries, which is simply labeled as *true* or *false*. For generic queries, participants were considered to agree with SQL if they built precisely the answer SQL produces. For value-inventing queries, participants agreed with SQL if the answers produced by SQL were their most satisfying options.

The first measurement we report is the *uncertainty coefficient* – also called *proficiency coefficient* or *Theil's U* [66] – between the demographic data and the responses for each query. This measure, for two random variables x and y, is the ratio of the mutual information of x and y to the entropy of x. Informally, this is the proportion of the information content of the agreement data that can be explained by the demographics. The demographic data would always correlate with some of the information of the agreement data for each query, simply because it is multi-dimensional while the latter, by definition, is a Boolean value. Thus, to see the real meaning of the uncertainty coefficient, we also compute its value for *randomly* generated data that uses the same distribution as the demographic and agreement data.

We also use machine learning techniques to analyze possible correlations. We study the average accuracy of a Logistic Regression [51], a Random Forest [12], and a Multi-layer Perceptron (Neural Network) [52] classifier trained on 80% of our data, and then assessing the remaining 20%. To evaluate the quality of the prediction, we compare the accuracy of these classifiers with a dummy classifier algorithm that always returns the highest probability outcome from the training set.

The results, for both generic and value-inventing queries, are presented in Table 5. Startlingly, they show that the uncertainty coefficient does not produce better results than those obtained on random data, that is, the real life correlation between demographics and answers is noticeably lower than the one obtained from random data. Likewise, the logistic regression, the random forest and the multi-layer perceptron classifiers fail to outperform the dummy classifier.

Therefore, we have strong evidence that the *demographic data is not sufficient to predict, with a reasonable degree of certainty, the user's preferred behavior of SQL queries in the presence of* **`NULL`**. As a consequence, the paradigm – prevalent in the academic literature – of deriving answers from the semantics of **`NULL`** cannot satisfy all users.

> *"Though some of these scenarios give the opportunity to address common frustrations, they have also illustrated how difficult it would be to change the way NULLs are handled without significantly changing how other SQL syntax works. Any changes could also lead to horrendous version control issues."*
>
> *A participant*

7 CONCLUSIONS

Based on the results of the survey, we arrive at the following.

Conclusion 1: `NULL` values are problematic. The data management community has come to this conclusion anecdotally. To the best of our knowledge, ours is the first study that provides strong evidence that this is actually the case in practice. The study provides clues and indications about users' (dis)satisfaction with SQL features:

- The **`NULL`** values appearing in a single database can have a variety of meanings; we identified over 20 semantic combinations.
- SQL's three-valued logic filters out too many results, which increases users' disagreement with SQL on queries with negation.
- When **`NULL`** is considered as a syntactic value, as in **`GROUP BY`** or set operations, dissatisfaction increases.
- While **`NULL`** as a function argument tends to behave as expected by users, the use of **`NULL`** in aggregates is more controversial and would benefit from further explanations.

Conclusions 2: Handling `NULL`s in RDBMSs is an open problem. Even though we have identified concrete issues with **`NULL`** values, there are no satisfying solutions to handle them yet. Most existing research is not addressing the problems that database practitioners are encountering. What exacerbates the problem, is that neither demographics nor the semantics of **`NULL`** can explain what users want. The bottom line is that more socio-technical studies (e.g., interviews, web-data analysis, online surveys, etc.) are required to address the problem of **`NULL`** with users in mind.

Table 5: Association and prediction scores obtained with (a) generic queries, (b) value inventing queries and `NULL` values in the database, and (c) value inventing queries and computational `NULL`s.

Query	Theil's U	Theil's U Random	LR Classifier	RF Classifier	MLP Classifier	Dummy Classifier	
`=`	48%	69%	69%	63%	65%	72%	
`<>`	45%	73%	61%	51%	50%	52%	
`NOT IN`	61%	79%	84%	79%	78%	84%	(a)
`NOT EXISTS`	45%	78%	78%	72%	72%	78%	
`EXCEPT`	51%	78%	56%	60%	55%	55%	
`EXCEPT ALL`	54%	92%	89%	85%	87%	89%	
30 + `NULL`	55%	80%	84%	81%	80%	84%	
`NULL` + `NULL`	51%	100%	98%	97%	97%	98%	
`SUM`(30, `NULL`)	61%	76%	63%	64%	62%	68%	
`MIN`(30, `NULL`)	44%	83%	70%	66%	65%	72%	(b)
`SUM`(`NULL`, `NULL`)	49%	90%	92%	91%	91%	92%	
`SUM`(30 + `NULL`, 0 + `NULL`)	51%	79%	81%	77%	78%	81%	
`GROUP BY`(`NULL`, `NULL`)	57%	81%	71%	61%	56%	73%	
`NULL` + `NULL`	55%	95%	93%	90%	89%	93%	
`SUM`(30, `NULL`)	39%	76%	70%	64%	64%	70%	
`MIN`(30, `NULL`)	40%	80%	74%	70%	68%	74%	(c)
`SUM`(`NULL`)	49%	85%	66%	65%	68%	67%	
`GROUP BY`(`NULL`, `NULL`)	38%	82%	54%	46%	46%	51%	

Conclusion 3: The research spectrum needs to broaden. The research on incomplete information undertaken by the data management community is insufficient, given the problematic state of `NULL` values that practitioners continue to tolerate today. One could argue that research has not yet properly translated to practice. However, research has mostly focused on the **EU** semantics, which only a minority of users encounter, rather than the **NA** and **NI** semantics, which are much more prevalent in practice. Furthermore, for a good part of the last 40 years, research has been based on the assumption that, in the presence of `NULL` values and for a specific semantics, all users should expect the same answers. The results of our survey provide evidence that this assumption does not hold in practice. Thus, we need to tackle the problem in a different way.

8 RECOMMENDATIONS

In light of these conclusions, we make the following recommendations to the data management community.

I) We suggest that research on nulls refocus to concentrate on its non-applicable (**NA**) and no-information (**NI**) interpretations that are assumed by most users. We need to further study and design models of query answering that play the same role for them as certain answers played under the existing-unknown interpretation.

II) Research on nulls should look at broader classes of queries, especially value-inventing (e.g., aggregate) queries. They can cause many problems and yet basic research on them is lacking.

III) We need to address research challenges brought to life by SQL design decisions, such as the use of 3VL and mixing different semantics of `NULL` (e.g., syntactic equality of nulls in grouping or set operations, and 3VL interpretation in conditions). It is important to implement new ideas that circumvent some of these problems (e.g., two-valued logic instead of 3VL, or proper marked nulls) in existing RDBMSs and evaluate their usefulness; this could lead to either alternative treatments of nulls, or new ways of treating them in query languages that are being designed.

IV) We need flexible query answering systems that have users' requirements in mind. For example, instead of concentrating on a single one-size-fits-all solution, we could even use the existing criticized semantics while explaining to the user why the answer is what it is and how nulls affected it.

Finally, and more generally, we would like to advocate the use of this socio-technical approach to data management problems in other areas of database research, to ensure that the topics of most research interest are indeed the ones that real-life users care about and need.

ACKNOWLEDGMENTS

This work was supported by data.world, EPSRC grants N023056 and S003800, as well as Edinburgh's EPSRC Center for Doctoral Training in Data Science. This work was only possible thanks to all the survey participants.

REFERENCES

[1] 1975. Interim Report: ANSI/X3/SPARC Study Group on Data Base Management Systems. *FDT - Bulletin of ACM SIGMOD* 7, 2 (1975).

[2] 1990. *Introduce named null definitions.* Technical Report. ISO/IEC JTC1 SC21 WG3 LON-111.

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases.* Addison-Wesley.

[4] S. Abiteboul, P. Kanellakis, and G. Grahne. 1991. On the representation and querying of sets of possible worlds. *Theoretical Computer Science* 78, 1 (1991), 158–187.

[5] Marcelo Arenas, Pablo Barceló, Leonid Libkin, Wim Martens, and Andreas Pieris. 2022. *Database Theory: Querying Data.* Freely available from github.com/pdm-book/community.

[6] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. 2014. *Foundations of Data Exchange.* Cambridge University Press.

[7] Ofer Arieli, Arnon Avron, and Anna Zamansky. 2010. Maximally Paraconsistent Three-Valued Logics. In *Principles of Knowledge Representation and Reasoning (KR).* AAAI Press.

[8] Paolo Atzeni and Nicola M. Morfuni. 1984. Functional Dependencies in Relations with Null Values. *Inform. Process. Lett.* 18, 4 (1984), 233–238.

[9] Anol Bhattacherjee. 2012. *Social science research: Principles, methods, and practices.* University of South Florida.

[10] Meghyn Bienvenu and Magdalena Ortiz. 2015. Ontology-Mediated Query Answering with Data-Tractable Description Logics. In *Reasoning Web.* 218–307.

[11] Stefan Brass and Christian Goldberg. 2006. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software* 79, 5 (2006), 630–644.

[12] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[13] Alan Bryman. 2016. *Social research methods.* Oxford university press.

[14] Peter Buneman, Achim Jung, and Atsushi Ohori. 1991. Using powerdomains to generalize relational databases. *Theoretical Computer Science* 91, 1 (1991), 23–55.

[15] K. Selçuk Candan, John Grant, and V. S. Subrahmanian. 1997. A Unified Treatment of Null Values Using Constraints. *Information Sciences* 98, 1-4 (1997), 99–156.

[16] S.G. Cannan, E.G. Dee, and J.M. Kerridge. 1987. *A proposal to provide support for multiple NULL states.* Technical Report. ISO/TC97/SC21/WG3-DBL-AMS51.

[17] E. F. Codd. 1975. Understanding relations (Installment #7). *FDT - Bulletin of ACM SIGMOD* 7, 3 (1975), 23–28.

[18] E. F. Codd. 1979. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems* 4, 4 (1979), 397–434.

[19] E. F. Codd. 1986. Missing Information (Applicable and Inapplicable) in Relational Databases. *SIGMOD Record* 15, 4 (1986), 53–78.

[20] Marco Console, Paolo Guagliardo, and Leonid Libkin. 2016. Approximations and Refinements of Certain Answers via Many-Valued Logics. In *KR.* AAAI Press, 349–358.

[21] Marco Console, Paolo Guagliardo, and Leonid Libkin. 2022. Propositional and predicate logics of incomplete information. *Artificial Intelligence* 302 (2022), 103603.

[22] Marco Console, Paolo Guagliardo, Leonid Libkin, and Etienne Toussaint. 2020. Coping with Incomplete Data: Recent Advances. In *Proceedings of the 39th ACM Symposium on Principles of Database Systems, PODS 2020.* ACM, 33–47.

[23] Chris J. Date. 2005. *Database in Depth - Relational Theory for Practitioners.* O'Reilly.

[24] C. J. Date and H. Darwen. 1996. *A Guide to the SQL Standard.* Addison-Wesley.

[25] Joel R Evans and Anil Mathur. 2005. The value of online surveys. *Internet research* (2005).

[26] Joel R Evans and Anil Mathur. 2018. The value of online surveys: A look back and a look ahead. *Internet research* (2018).

[27] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336, 1 (2005), 89–124.

[28] Su Feng, Aaron Huber, Boris Glavic, and Oliver Kennedy. 2019. Uncertainty Annotated Databases - A Lightweight Approach for Approximating Certain Answers. In *SIGMOD Conference.* ACM, 1313–1330.

[29] Nicola Fiorentino, Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. 2018. ACID: A System for Computing Approximate Certain Query Answers over Incomplete Databases. In *SIGMOD Conference.* ACM, 1685–1688.

[30] Enrico Franconi and Sergio Tessaris. 2012. On the Logic of SQL Nulls. In *AMW (CEUR Workshop Proceedings)*, Vol. 866. CEUR-WS.org, 114–128.

[31] Yunjun Gao and Xiaoye Miao. 2018. *Query Processing over Incomplete Databases.* Morgan & Claypool Publishers.

[32] G. H. Gessert. 1990. Four Valued Logic for Relational Database Systems. *SIGMOD Record* 19, 1 (1990), 29–35.

[33] Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. 2014. Naïve evaluation of queries over incomplete databases. *ACM Transactions on Database Systems* 39, 4 (2014), 31:1–31:42.

[34] Georg Gottlob and Roberto V. Zicari. 1988. Closed World Databases Opened Through Null Values. In *VLDB.* Morgan Kaufmann, 50–61.

[35] John Grant. 1977. Null Values in a Relational Data Base. *Inform. Process. Lett.* 6, 5 (1977), 156–157.

[36] Sergio Greco, Cristian Molinaro, and Francesca Spezzano. 2012. *Incomplete Data and Data Dependencies in Relational Databases.* Morgan & Claypool Publishers.

[37] Paolo Guagliardo and Leonid Libkin. 2016. Making SQL Queries Correct on Incomplete Databases: A Feasibility Study. In *PODS.* ACM, 211–223.

[38] Paolo Guagliardo and Leonid Libkin. 2019. On the Codd semantics of SQL nulls. *Information Systems* 86 (2019), 46–60.

[39] Sven Hartmann and Sebastian Link. 2012. The implication problem of data dependencies over SQL table definitions: Axiomatic, algorithmic and logical characterizations. *ACM Transactions on Database Systems* 37, 2 (2012), 13:1–13:40.

[40] Tomasz Imielinski and Witold Lipski. 1984. Incomplete information in relational databases. *J. ACM* 31, 4 (1984), 761–791.

[41] Tomasz Imielinski, Shamim A. Naqvi, and Kumar V. Vadaparty. 1991. Incomplete Objects - A Data Model for Design and Planning Applications. In *SIGMOD.* ACM Press, 288–297.

[42] Hans-Joachim Klein. 2001. Null Values in Relational Databases and Sure Information Answers. In *Semantics in Databases (Lecture Notes in Computer Science)*, Vol. 2582. Springer, 119–138.

[43] Nadine Lerat and Witold Lipski Jr. 1986. Nonapplicable Nulls. *Theoretical Computer Science* 46, 3 (1986), 67–82.

[44] Leonid Libkin. 2014. Incomplete information: what went wrong and how to fix it. In *PODS.* 1–13.

[45] Leonid Libkin. 2016. Certain answers as objects and knowledge. *Artificial Intelligence* 232 (2016), 1–19.

[46] Leonid Libkin. 2016. SQL's Three-Valued Logic and Certain Answers. *ACM Transactions on Database Systems* 41, 1 (2016), 1:1–1:28.

[47] Leonid Libkin and Limsoon Wong. 1996. Semantic Representations and Query Labguages for Or-Sets. *J. Comput. System Sci.* 52, 1 (1996), 125–142.

[48] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of Psychology* 22, 140 (1932), 5–55.

[49] W. Lipski. 1979. On Semantic Issues Connected with Incomplete Information Databases. *ACM Transactions on Database Systems* 4, 3 (1979), 262–296.

[50] Witold Lipski. 1984. On Relational Algebra with Marked Nulls. In *PODS.* 201–203.

[51] Peter McCullagh and John A Nelder. 2019. *Generalized linear models.* Routledge.

[52] Fionn Murtagh. 1991. Multilayer perceptrons for classification and regression. *Neurocomputing* 2, 5 (1991), 183–197.

[53] United Nations. 2007. *International Standard Industrial Classification of All Economic Activities (ISIC) Revision 4.* United Nations, New York. https://unstats.un.org/unsd/classifications/Econ/isic

[54] Thomas Neumann. 2018. Reasoning in the Presence of NULLs. In *34th IEEE International Conference on Data Engineering.* IEEE Computer Society, 1682–1683.

[55] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. 2008. Linking Data to Ontologies. *Journal on Data Semantics* 10 (2008), 133–173.

[56] Melanie Revilla and Carlos Ochoa. 2017. Ideal and Maximum Length for a Web Survey. *International Journal of Market Research* 59, 5 (2017), 557–565.

[57] Judy Robertson. 2012. Likert-type scales, statistical methods, and effect sizes. *Commun. ACM* 55, 5 (2012), 6–7.

[58] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. 2017. The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. *Proceedings of the VLDB Endowment* 11, 4 (2017), 420–431.

[59] Juan Sequeda. 2020. *Understanding NULL values: a research partnership with the University of Edinburgh.* Retrieved July 17, 2022 from https://data.world/blog/understanding-null-values-a-research-partnership-with-the-university-of-edinburgh/

[60] Bernhard Thalheim and Klaus-Dieter Schewe. 2010. NULL 'Value' Algebras and Logics. In *20th European-Japanese Conference on Information Modelling and Knowledge Bases (EJC) (Frontiers in Artificial Intelligence and Applications)*, Vol. 225. IOS Press, 354–367.

[61] Etienne Toussaint, Paolo Guagliardo, and Leonid Libkin. 2020. Knowledge-Preserving Certain Answers for SQL-like Queries. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning.* 758–767.

[62] Mark van der Loo and Edwin de Jonge. 2018. *Statistical Data Cleaning.* Wiley.

[63] Ron van der Meyden. 1998. Logical Approaches to Incomplete Information: A Survey. In *Logics for Databases and Information Systems.* 307–356.

[64] Kwok-bun Yue. 1991. A More General Model for Handling Missing Information in Relational Databases Using a 3-valued Logic. *SIGMOD Record* 20, 3 (1991), 43–49.

[65] Carlo Zaniolo. 1984. Database Relations with Null Values. *J. Comput. System Sci.* 28, 1 (1984), 142–166.

[66] Arnold Zellner and Henri Theil. 1992. Three-stage least squares: simultaneous estimation of simultaneous equations. In *Henri Theil's Contributions to Economics and Econometrics.* Springer, 147–178.