Troubles with Nulls, Views from the Users

Citation for published version:
<https://vldb.org/pvldb/volumes/15/paper/Troubles%20with%20Nulls%2C%20Views%20from%20the%20Users>

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of the VLDB Endowment (PVLDB)

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Troubles with Nulls, Views from the Users

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

Leonid Libkin  
University of Edinburgh & ENS Paris  
libkin@ed.ac.uk

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

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:

PVLDB Artifact Availability:
The survey, 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, 48]. 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, 39, 48] 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 practitioners’ 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 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 [57] 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 [47] 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 [56]. 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 (Fig. 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
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 (Fig. 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 Fig. 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 [58] to advertise the survey. We also sent both the survey and the blog post to the
How often may the patterns [in the databases of the survey] occur in a real-life dataset?

- 20.0%
- 31.3%
- 23.3%
- 15.7%
- 9.7%

How often do you manipulate SQL code?

- 10.0%
- 23.5%
- 19.0%
- 46.4%

(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

data.world mailing list through the monthly digest, reaching their 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 [52]. As shown in Fig 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 Fig. 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 Fig. 2c and Fig. 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).

Participants' engagement. The complexity and the extent of our study was too ambitious to be kept within the recommended 10-minute survey format [55]. 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. Fig. 2e and Fig. 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. Fig. 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. Fig. 3d shows that users frequently deal with LEFT and RIGHT JOIN; in fact, these are more common than explicitly specified inner joins. Full outer joins are also quite common.

<table>
<thead>
<tr>
<th>(a) How often do you encounter NULL values?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
</tr>
<tr>
<td>15.8%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(b) How often do you explicitly specify a column as NOT NULL?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
</tr>
<tr>
<td>13%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(c) How often do you explicitly add NOT NULL to the following constraints?</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIMARY KEY</td>
</tr>
<tr>
<td>FOREIGN KEY</td>
</tr>
<tr>
<td>UNIQUE</td>
</tr>
<tr>
<td>(d) How often do you use the following join operators on columns with NULLs?</td>
</tr>
<tr>
<td>[INNER JOIN]</td>
</tr>
<tr>
<td>LEFT</td>
</tr>
<tr>
<td>FULL [OUTER JOIN]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(e) How often do you explicitly add NOT NULL to the following constraints?</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFNULL()</td>
</tr>
<tr>
<td>ISNULL()</td>
</tr>
<tr>
<td>IS [NOT] NULL</td>
</tr>
<tr>
<td>COALESCE()</td>
</tr>
<tr>
<td>IS [NOT] DISTINCT</td>
</tr>
</tbody>
</table>

![Figure 3](image-url)
Fig. 3b and Fig. 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 Fig. 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 [64] 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 the users asking them a multiple choice question on the meaning of those, non-applicable and unknown values, were adopted early and those representing existing but currently unknown values (two addressed in the research literature, going all the way back to [64]). The CEO may receive a regular salary but the data source from which the table is populated may have been dirty. However, the non-applicable semantics seems to dominate.

**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 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 [43, 45], 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 (Fig. 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 nor 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 treat 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.

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.

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>(a): 0</th>
<th>(b): Constant</th>
<th>(c): NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality String</td>
<td>5.1</td>
<td>3.8</td>
<td>22.2</td>
</tr>
<tr>
<td>Equality Integer</td>
<td>3.3</td>
<td>2.6</td>
<td>21.5</td>
</tr>
<tr>
<td>Inequality String</td>
<td>41.6</td>
<td>48.7</td>
<td>11.0</td>
</tr>
<tr>
<td>Inequality Integer</td>
<td>41.6</td>
<td>43.0</td>
<td>8.72</td>
</tr>
</tbody>
</table>

Table 2: Proportion of answers that differ from SQL.
We now look at queries that may output values not found in a table. We may ask how adding two nulls will behave (e.g., \( \text{SUM}(\text{NULL} + \text{NULL}) \)) or the same expression \( \text{SUM}(\text{NULL} + 30) \) and \( \text{SUM}(\text{NULL} + \text{NULL}) \)). We also ask about queries in which the summation aggregate is applied to values obtained as the result of arithmetic expressions \( \text{SUM}(\text{NULL} + 30, \text{NULL} + 0) \)), and look at queries with \text{GROUP BY} where grouping is done on an attribute whose value is \text{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 \( 30 + \text{NULL} = \text{NULL} \). The next row shows that more than 75% were satisfied with the way SQL evaluated \( \text{NULL} + \text{NULL} \). 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 (Fig. 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 \text{NULL} (using the rule that the value of any such function is \text{NULL} then) is fairly uncontroversial and accepted by most. The agreement with SQL’s way of handling \text{NULL} slightly decreases when aggregates are applied to columns containing \text{NULL} (in which case the rule is not to output \text{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 \text{NULL}. In this case SQL uses the syntactic equality of nulls, i.e., two \text{NULL}s are equal, which of course differs from the treatment of \text{NULL}s elsewhere and thus causes additional problems.

We then look at similar queries but with one important difference. Now \text{NULL}s are not present in the database but rather created by an \text{OUTER JOIN} query. Then we ask questions about similar queries

---

<table>
<thead>
<tr>
<th>Query</th>
<th>A=SQL</th>
<th>A≤SQL</th>
<th>A≠SQL</th>
<th>A≥SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NOT IN</strong></td>
<td>15.4</td>
<td>0</td>
<td>84.5</td>
<td>0</td>
</tr>
<tr>
<td><strong>NOT EXISTS</strong></td>
<td>78.3</td>
<td>15.8</td>
<td>1.7</td>
<td>4.1</td>
</tr>
<tr>
<td><strong>EXCEPT</strong></td>
<td>53.0</td>
<td>10.4</td>
<td>27.0</td>
<td>9.6</td>
</tr>
<tr>
<td><strong>EXCEPT ALL</strong></td>
<td>10.7</td>
<td>76.8</td>
<td>8.9</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Table 3: Proportions of respondents’ answers vs SQL.
Table 4: Results obtained with (a) value-inventing queries and NULLs that occur in the database, and (b) value inventing queries and computational NULLs.

<table>
<thead>
<tr>
<th>Query</th>
<th>Quartiles/Median</th>
<th>Mean</th>
<th>Agreement</th>
<th>Controversy</th>
<th>Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 + NULL</td>
<td><img src="image1" alt="Bar Chart" /></td>
<td>3.8</td>
<td>high</td>
<td>medium</td>
<td>21.6%</td>
</tr>
<tr>
<td>NULL + NULL</td>
<td><img src="image2" alt="Bar Chart" /></td>
<td>4.4</td>
<td>high</td>
<td>low</td>
<td>2.1%</td>
</tr>
<tr>
<td>SUM(30, NULL)</td>
<td><img src="image3" alt="Bar Chart" /></td>
<td>3.4</td>
<td>medium</td>
<td>high</td>
<td>33.3%</td>
</tr>
<tr>
<td>MIN(30, NULL)</td>
<td><img src="image4" alt="Bar Chart" /></td>
<td>3.4</td>
<td>medium</td>
<td>medium</td>
<td>30.5%</td>
</tr>
<tr>
<td>SUM(NULL, NULL)</td>
<td><img src="image5" alt="Bar Chart" /></td>
<td>4.1</td>
<td>high</td>
<td>low</td>
<td>9.5%</td>
</tr>
<tr>
<td>SUM(30 + NULL, 0 + NULL)</td>
<td><img src="image6" alt="Bar Chart" /></td>
<td>3.6</td>
<td>high</td>
<td>high</td>
<td>23.2%</td>
</tr>
<tr>
<td>GROUP BY(NULL, NULL)</td>
<td><img src="image7" alt="Bar Chart" /></td>
<td>3.1</td>
<td>medium</td>
<td>high</td>
<td>29.5%</td>
</tr>
<tr>
<td>NULL + NULL</td>
<td><img src="image8" alt="Bar Chart" /></td>
<td>4.3</td>
<td>high</td>
<td>low</td>
<td>7.5%</td>
</tr>
<tr>
<td>SUM(30, NULL)</td>
<td><img src="image9" alt="Bar Chart" /></td>
<td>3.1</td>
<td>high</td>
<td>high</td>
<td>35.5%</td>
</tr>
<tr>
<td>MIN(30, NULL)</td>
<td><img src="image10" alt="Bar Chart" /></td>
<td>3.3</td>
<td>high</td>
<td>high</td>
<td>31.5%</td>
</tr>
<tr>
<td>SUM(NULL)</td>
<td><img src="image11" alt="Bar Chart" /></td>
<td>3.2</td>
<td>high</td>
<td>high</td>
<td>37.9%</td>
</tr>
<tr>
<td>GROUP BY(NULL, NULL)</td>
<td><img src="image12" alt="Bar Chart" /></td>
<td>2.0</td>
<td>low</td>
<td>high</td>
<td>63.2%</td>
</tr>
</tbody>
</table>

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

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 NULLs 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 NULLs in arithmetic operations, and slightly less so in aggregation, where the rule changes and NULLs are ignored. Their level of dissatisfaction increases when NULLs come not from a database but generated by a query. What they do not like is the use of syntactic equality (saying that two NULLs 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, 48], and several surveys are available; e.g., [22, 36, 62]. 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 [39], 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 in 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 [54] 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 [39, 48]. 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 [44, 49], 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) [39], 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.1

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 NULLS 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 [53], 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 [64], which proposed the trichotomy of NI, NA, and EU nulls. This was followed by work on NA in [42], 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] looked at functional dependencies over databases with such 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, 59, 63]. 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, 41]. The approach of [41] is based on the notion of subsumption of tuples with nulls, already present in [14, 64], 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* [40, 46]). These can be expressed by means of extra constraints attached to tuples in the spirit of conditional tables [39]. 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

---

1 A well-founded definition of certain answers for value-inventing queries under bag semantics was only recently proposed in [60].
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
corns 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, 61] 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 cor-
relation between the demographics for our respondents, and their
preferences for the behavior of SQL queries. To address this, we per-
formed 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 [65] – 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 informa-
tion 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 ex-
plained 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 cor-
rrelations. We study the average accuracy of a Logistic Regression
[50], a Random Forest [12], and a Multi-layer Perceptron (Neural
Network) [51] classifier trained on 80% of our data, and then assess-
ing 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 manage-
ment 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 pro-
vides 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 combina-
tions.
• SQL’s three-valued logic filters out too many results, which
  increases users’ disagreement with SQL on queries with nega-
tion.
• 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 ex-
  pected by users, the use of NULL in aggregates is more contro-
versial and would benefit from further explanations.

Conclusions 2: Handling NULLs in RDBMSs is an open prob-
lem. 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 prac-
titioners 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.

Conclusion 3: The research spectrum needs to broaden. The
research on incomplete information undertaken by the data man-
agement community is insufficient, given the problematic state
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 NULLs.

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

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


