Debugging Database Queries: A Survey of Tools, Techniques, and Users

Sneha Gathani
University of Maryland,
College Park
sgathani@cs.umd.edu

Peter Lim
University of Maryland,
College Park
plim7@terpmail.umd.edu

Leilani Battle
University of Maryland,
College Park
leilani@cs.umd.edu

ABSTRACT
Database management systems (or DBMSs) have been around for decades, and yet are still difficult to use, particularly when trying to identify and fix errors in user programs (or queries). We seek to understand what methods have been proposed to help people debug database queries, and whether these techniques have ultimately been adopted by DBMSs (and users). We conducted an interdisciplinary review of 112 papers and tools from the database, visualization and HCI communities. To better understand whether academic and industry approaches are meeting the needs of users, we interviewed 20 database users (and some designers), and found surprising results. In particular, there seems to be a wide gulf between users' debugging strategies and the functionality implemented in existing DBMSs, as well as proposed in the literature. In response, we propose new design guidelines to help system designers to build features that more closely match users debugging strategies.

Author Keywords
Debugging Databases, Literature Review, Empirical Study, Survey, Visualization

CCS Concepts
•Human-centered computing → User studies; Field studies; •Information systems → Query languages;

INTRODUCTION
Analysts and developers need an efficient way to articulate how they want a computer to process large datasets [62], such as by combining multiple data sources (or tables), clustering specific data records, or calculating aggregate statistics [9]. Database management systems (DBMSs) like PostgreSQL[1] and Microsoft SQL Server[2] enable users to specify their desired analysis output by issuing declarative programs (or queries) on datasets. To process queries, the DBMS first translates them into a logical query plan that represents the operations that must be executed on the data to produce the specified output. Then the logical query plan is compiled into a physical query plan that executes the specified steps efficiently on the underlying hardware (e.g., the user's laptop or a remote server).

[1] https://www.postgresql.org/
[2] https://www.microsoft.com/en-us/sql-server/default.aspx

However, as with any complex programming language, it is rare for users to be able to write "perfect" database queries on their first try (i.e., without any errors). Hence, the *debugging* of queries is a necessary step towards learning to use DBMSs effectively. Debugging a query often requires more than simply fixing syntax issues. Query errors can manifest as vague exceptions thrown by the DBMS, or unexpected behavior like returning zero results or duplicate entries in the results. Debugging these errors requires an intuitive understanding of not only the structure of the corresponding queries but also the DBMS itself, which can be difficult to interpret [9]. Multiple communities have studied how to help users debug queries: the database community has proposed algorithms to detect specific errors (e.g., [22, 77]), the visualization community more intuitive visual representations of queries (e.g., [92, 95]), and the HCI community new mechanisms for real-time feedback as users write programs (e.g., [71]). However, users are unlikely to adopt all these different techniques just to address a single problem. Our communities should rather work together to develop *interdisciplinary tools* to ease the burden on users.

Unfortunately, little work has been done to synthesize and integrate ideas from across these different research areas. To better understand how these efforts can be combined to provide holistic debugging support for end users, we performed an interdisciplinary review of 112 papers and tools of the past 25 years. These works ranged from formalised debugging algorithms to visual query interfaces to interviews with industry analysts to standard debugging features provided by current commercial tools (e.g., breakpoints). We found promising hybrid techniques proposed in the literature (e.g., "why" and "why not" debugging [66, 22], iterative debugging [3]), yet we also saw a lack of adoption of these techniques in industry.

To better understand how techniques in the literature translate into the real world, we conducted an interview study with 20 database users, ranging from students in database courses to full-time industry analysts. Participants shared some of their own queries, as well as tools and evaluation strategies they employed to debug queries. Even though we observed a number of strategies, we found that participants rarely use (and many had never heard of) the proposed debugging techniques from the literature, and seldom used the debugging features in commercial tools. Participants primarily evaluated and fixed their queries manually. We summarize recurring activities and steps used to debug database queries, and identify common pain points in the debugging process, including: vague SQL exceptions, lack of access to the database schema while debugging, and having to systematically test by hand each component of the query (e.g., SQL clause or sub-query).

Our findings uncover several promising techniques in the literature that do seem to match user's debugging strategies, such as illustrating intermediate results (e.g., [45, 9]). Based on the debugging strategies we observed, we believe that certain techniques from the literature could prove very useful to implement in commercial systems, particularly features that support the process of incremental query building and recursive error checking. Furthermore, overall debugging time can be significantly reduced by incorporating few simple utilities into existing tools, such as automated translators from queries written in older SQL standards to newer dialects.

To better understand the industry perspective, we reached out to six database tool designers. We found that even though query debugging is valued in industry, database companies have other more pressing needs, suggesting that researchers could have a big impact by partnering with industry to build the debugging tools that we need today.

LITERATURE REVIEW

In this section, we review the relevant literature in debugging and understanding query behavior. We highlight overlaps between query and general program debugging techniques, as well as alternatives that seek to circumvent the introduction of errors in queries (e.g., intuitive query interfaces).

Methods

Query debugging spans multiple areas, such as databases, visualization, and HCI. To address this challenge, we implemented the steps below to select papers for our literature review. We define our review topic as: *algorithms, interfaces and tools designed to support the debugging of database queries*. A paper or tool was considered relevant if it addressed this topic or a related topic, understanding: query behavior (query and program comprehension); user debugging behavior; user querying and analysis behavior; visualizing queries; or query interfaces; interactive debugging interfaces.

1. Relevant papers from the last five years of conferences in databases (e.g., SIGMOD, VLDB), visualization (VIS, VISSOFT), and HCI (CHI) were added to our review list.
2. We performed targeted online searches to identify papers and tools outside of the venues or dates mentioned above. Relevant search results were added to our review list.
3. We scanned the references of the papers yielded by the previous steps, and added relevant references.
4. We searched for later papers that cite papers yielded by the previous steps. Relevant papers were added to our list.

Our selection process yielded 91 papers and tools. However, 15 papers and 6 tools did not match the above filter criteria, but still provide useful information to understand related concepts (e.g., provenance, static program analysis). We include them as supplements to our analysis, resulting in 112 total papers and tools for our review. Papers and tools were reviewed to understand strategies and processes that users may employ to reason about and debug queries, as well as for existing algorithms and features to aid users in debugging queries.

Understanding User Analysis and Query Behavior

Understanding the Data Analysis Process. Several projects analyze the specific strategies and processes of data analysts and data scientists. These processes can be investigated through interviews (e.g., [62, 6, 63, 93, 65]), as well as through quantitative evaluation of user logs collected by data analysis tools (e.g., [10, 49, 47, 44]). For example, Kandel et al. interviewed 35 analysts and found three different analyst archetypes ("hackers", "scripters", and "application user[s]"), where the "hackers" and "scripters" are typically proficient in writing code and scripts, including queries [62]. Gotz and Zhou analyze user interaction logs, and find that analysts tend to decompose the (visual) analysis process into a hierarchy of more targeted subtasks [44]. These projects provide a broader perspective on the larger data analysis process, like which tasks comprise the core stages of data analysis and data science, and where analysts tend to spend their time, but may lack specifics in terms of how individual stages are carried out (e.g., the stages of the query debugging process). Petrillo et al. present a tool for visualizing logs from users' program debugging sessions, which could shed light on users' debugging strategies, given sufficient data for analysis [84].

Patterns in SQL Usage. Several projects analyze users' query patterns [72, 60]. For example, Jain et al. analyzed query logs collected from years of running the SQLShare system [60]. Their findings suggest that improving ease-of-use can influence a user's continued interaction with DBMSs. However these projects do not discuss how users debug their queries, or how the debugging process could be improved.

Takeaways. Investigating the broader data analysis process can provide insight into how more targeted tasks–in this case query debugging–may generally be structured. For example, Kandel et al. found that a notable number of analysts are comfortable writing queries by hand [62]. Users may decompose a high level debugging task into smaller targeted tasks (i.e., similar to observations by Gotz and Zhou [44]). Analyses of users' querying behavior (e.g., [60]) can also provide insight into how query interfaces are used. However, the granularity of these insights are too coarse to speak to the specific strategies and experiences related to query debugging, and thus are of limited use in determining how best to design debugging tools.

Alternatives to Writing SQL Queries

Many projects develop alternative user interfaces for DBMSs, and fall into two groups: using other (written or spoken) languages or visual interfaces to formulate queries, which map to an existing query language behind the scenes (primarily SQL).

Programming Languages to SQL. Several techniques translate programming languages to SQL [38, 37, 70], for example, DBridge translates imperative code (e.g., Java) into SQL queries, and aims to make the program's execution more efficient, for example by reducing communication and data transfer between a user program and the DBMS [38, 37]. Li et al. propose a variant of SQL called "Schema-free SQL" [70], where users can write queries without perfect recall of the database schema; for example, users can guess forgotten attribute names as they write a query, which are corrected using schema-free SQL, but throw errors using standard SQL.

Object-relational mapping (ORM) languages (e.g., SQLAlchemy[3], Django[4]) and related frameworks (e.g., Ibis[5]) enable programmers to integrate local programming environments (e.g., Pandas[6]) directly with DBMSs.

A range of domain-specific languages have also been developed to query relational data in various contexts beyond just SQL (e.g., MyriaL [102]), and even extend query support to other dataset types (e.g., GraphQL [39], SPARQL [87]).

Natural Language to SQL. Other systems avoid code altogether, and instead translate natural language (e.g., English) directly to SQL queries [73, 69]. For example, the NaLIR system translates English descriptions of queries into proper SQL [69]. However, current methods are still too restrictive for wide adoption, necessitating further research in this area.

Visual Query Interfaces. Many systems provide graphical interfaces for constructing queries using direct manipulation [19]. These interfaces vary across a spectrum from more literal query building interfaces [105, 30, 5, 97], to more abstract exploration interfaces (e.g., [95, 74]). Query builder-style interfaces have users construct a tree or graph representation of the query, where nodes denote input data or query operators, and directed edges give the flow of data through operators [105, 30, 5, 97, 3]. These interfaces can be made more intuitive for specific dataset types [25], such as temporal data (e.g., filtering for specific patterns of events [52, 104, 53, 74, 107]) or network data (e.g., searching for structural patterns in subgraphs [85, 35, 57]). In the case of network data, a graph representation for queries mimics the structure of the graph itself. Zhang et al. generate interactive query interfaces for DBMSs by analyzing query logs: they map groups of queries to interaction widgets that produce equivalent results [110].

On the exploration side of the spectrum, a strong emphasis is placed on *dynamic queries* [92]. Instead of manipulating a representation of the query, dynamic queries allow users to manipulate the data directly [50]. For example, Spotfire supports interactions like brushing and linking, zooming, and range sliders, to perform query operations (e.g., joining, aggregating, and filtering, respectively) [4]. Polaris [95] (now Tableau [99]) uses the VizQL language to map any visualization design in the interface to a corresponding SQL query. To construct visualizations, users drag attributes to encoding "shelves", where each completed interaction translates to a SQL query.

However, complicated queries are difficult to write with visual query interfaces, often requiring many direct manipulation interactions to construct the queries. Furthermore, these graphical interfaces generally lack the debugging infrastructure available when working with programming languages.

Takeaways. Many have observed that traditional query interfaces are overly complex, potentially leading to more (and more complex) errors in users' queries. Many projects and some commercial tools have developed more intuitive interfaces for interacting with database systems. While these tools work very well for some analysts (e.g., the "application user[s]" [62]), it seems that many users still prefer traditional query interfaces (e.g., the "hackers" [62]). Furthermore, data science and database courses may still emphasize traditional query methods (e.g., [75]). Thus, many users may opt out of using these interfaces, but still need debugging support.

[3]https://www.sqlalchemy.org/
[4]https://www.djangoproject.com/
[5]https://docs.ibis-project.org/
[6]https://pandas.pydata.org/

Data Debugging

Sometimes, the underlying data itself may contain errors that need to be fixed [62], which we call "data debugging".

Data Profiling. Several systems provide graphical interfaces to inspect issues with the data. A number of systems support "data profiling", which help users calculate data quality characteristics (e.g., the presence of nulls or duplicate values), and identify potential data errors [2, 62]. For example, Profiler provides an intuitive visual interface for summarizing and interacting with data quality measures for tabular data [62].

Identifying Causes of Data Errors. Some systems aim to explain the causes for data errors. For example, Scorpion searches for patterns that could explain data errors and translates these patterns into queries [108]. QFix seeks to explain how errors occur in a dataset by analyzing the queries that have modified the data in the past, and detecting which queries may have contributed to the errors [103].

Data Cleaning. Several projects suggest best practices [106, 27], and automated features to ease the burden of systematically cleaning large datasets [61, 96, 90]. For example, Holoclean models inconsistencies within a dataset using random variables, and uses probabilistic inference to reason about possible strategies to clean groups of data records [90].

Takeaways. Analysts spend considerable time on fixing data errors, before moving to writing (and debugging) queries [62], and a variety of "data debugging" tools have been developed. However, since data debugging occurs *before* data analysis (and thus query debugging), we consider data debugging to be orthogonal to query debugging, and do not touch on it further.

Query and Program Debugging

We now turn our focus to how systems help users debug their analysis programs, and database queries in particular. These systems generally use one or both of the following strategies, which we use to organize the remainder of our review: 1) modeling the *logical structure* of the query (i.e., using static program analysis methods, e.g., [8, 66]), or 2) modeling *execution output* from the query (i.e., dynamic analysis, e.g., [109, 66]). Systems use these models to illustrate the behavior of the given query, or to make predictions about the cause of the error(s). We also highlight non-database debugging projects that could be relevant to database debugging use cases.

Analyzing Logical Structure.

Many debugging methods aim to infer specific characteristics from the logical structure of queries. Logical structure refers to the organization and meaning of operations within a query (or its corresponding query execution plan). These techniques need only the user's written query (or generated query plan) to facilitate debugging, and avoid executing the query.

Illustrating Logical Structure. One class of techniques illustrates the structure of a query via static analysis of the query [79, 33, 11, 18]. For example, QueryViz summarizes the behavior of queries by generating diagrams to capture the relationships between operators (e.g., execution flow, containment, inclusion/exclusion of tuples) [33]. QueryViz diagrams use nodes and links to denote relationships between operators, similar to query-builder interfaces, and provides example outputs to demonstrate the behavior of query operators. Mou et al. illustrate the execution flow of entire data science workflows, which may consist of multiple black-box components, some of which may not involve a DBMS. VisTrails considers a similar problem, but focuses on visualization workflows [11, 18]. As mentioned previously, Zhang et al. analyze the logical structure of queries, but perform this analysis across full DBMS query sessions, to generate an equivalent interactive user interface for executing similar queries [110].

Several projects apply similar techniques, but for other languages [48, 7]. For example, Scoped depicts the execution scope of variables by analyzing program source code [7]. These principles could be applied to database debugging contexts. For example to visualize relationships between different query components (e.g., SQL clauses, sub-queries).

Inferring Structural Properties. A second class of techniques focuses on inferring specific characteristics of queries through static analysis [26, 101]. For example, Cosette can determine whether two queries are semantically equivalent [26]. These techniques could also be used for debugging purposes, e.g., Cosette could be used to identify a simpler, more succinct query to display when recommending solutions to errors.

Predicting Causes of Errors. A third class of techniques uses static analysis to predict potential causes of errors in programs [81, 94], including queries [17, 16]. These techniques focus on analyzing the "declarative meaning" [81], or the intent behind the query components and not their implementation. These methods often involve searching for structural patterns of programs that denote errors, when compared to a description of program intent specified by the user. However, these techniques may not be intuitive to users, since they require the user to write a second program, in order to debug the first one.

Takeaways. A number of query and program debugging strategies analyze the logical structure of queries without executing them. Some systems visualize or illustrate this logical structure to improve users' understanding of query behavior, while others infer correctness of queries using theoretical proofs or counter-examples. By avoiding execution of queries, these strategies can be scaled to big data use cases. However, users may want a deeper understanding of how the underlying data are affected by different stages of a query's execution, which these techniques are unable to provide.

Analyzing Execution Structure.

In contrast to static analysis techniques, others focus on collecting intermediate results at various stages of query execution, and using this data to model query behavior. These techniques can be thought as provenance-based debugging techniques, given that they require the collection of provenance metadata.

Empty Answer Problem. One class of debugging projects focuses specifically on analyzing a common database query error, known as the "empty answer problem" [77, 78, 100]. This problem occurs when a user writes a very restrictive query. Since DBMSs are designed to return exact answers, overly-restrictive queries often produce zero records, making it difficult for users to identify the cause of the error (i.e., the filter that is eliminating results). The empty answer problem has also been observed in visual analytics (e.g., [47]). Empty answer debugging techniques generate a less restrictive version of the original query, such that fuzzy results are returned to the user, but may not be exactly what the user was looking for. However, the empty answer problem error is only a fraction of the errors that a user encounters with DBMSs. For example, prior work has found that users often run into situations where queries produce non-zero yet undesired results [9].

Recording Query Provenance. Database provenance (or lineage) records the behavior of a query at each stage of execution, and thus can capture the execution behavior of a query. However, constantly recording detailed information about the execution of queries (and programs) can consume considerable time and storage. The database community has extensively studied how to quickly and efficiently record provenance for queries [51, 98, 15, 64, 42, 31, 32, 83] (including over probabilistic datasets [13]), as well as for more general analysis workflows [41, 59, 11, 18]. Database provenance shares some similarities with provenance in the visualization literature [88, 49]. However, the database community focuses more on data provenance (i.e., the history and source of program results), rather than analytic provenance (i.e., the sequence of visualizations, interactions, and insights produced by the user). Database provenance is an active area of research in the database community, but query debugging is only a small fraction of this sub-area, from our observation.

Illustrating Execution Structure. Given recorded provenance data for a query, a second class of debugging projects focus on providing intuitive visual representations of the results [9, 34, 45, 46, 82, 76]. For example, StreamTrace generates visualizations and interactive widgets that enable users to explore the execution of a temporal SQL query, step-by-step [9]. They provide interactive debugging visualizations by rewriting the original query as a series of intermediate queries, and recording which input events and query operators contributed to specific output events. Grust et al. use a similar technique, where instead of an interactive visualization they allow users to click on any query component q_i to observe the intermediate database table produced at that point [45]. RATest records provenance data by comparing execution of a given user query to an existing "solution" query (e.g., database homework problems) [75]. Perfopticon visualizes performance characteristics for each machine and progress across all machines for given query on a distributed (i.e., parallel) database. [76].

There are entire conferences devoted to visualizing programs and software (in particular VISSOFT [1]); we direct readers to the VISSOFT proceedings for more details. However, a subset of these projects visualize software execution behavior for program debugging [71, 89, 86], summarization and comprehension [40, 12]. For example, Lieber et al. use online (i.e., live) provenance data to report line and method execution statistics using in-situ visualizations within the code [71]. Similar principles are used by Hoffswell et al. to help Vega-Lite[91] users debug their visualization designs [54, 55].

Takeaways. Provenance-based debugging tools aim to capture the execution structure of queries and programs, and give the user different ways to "open the hood" and observe how a query or program was executed. Some systems integrate tightly with the underlying DBMS, providing users with a closer connection to the behavior of the DBMS [45, 9, 76]. However, it is still up to the user to interpret the provenance data, identify errors, and devise solutions. Unlike static analysis techniques, dynamic provenance-based debugging relies on a potentially resource-heavy data collection process. As we will see ahead, these techniques can be combined to provide the benefits of both strategies.

Debugging Both Logical and Execution Structure

"Why" and "Why Not" Debugging. A number of systems construct models of queries using both the logical and execution structure of the query [16, 56, 22, 68, 14]. Many of these systems aim to answer "why" and "why not" questions: why a particular result appears in the output, or why a given input data record fails to appear in the final result [21]. For example, Chapman et al. answer "why not" questions by constructing a logically derived directed acyclic graph of the query, and then conducting top-down and bottom-up searches over this graph to identify points in the query where key results are eliminated from the output [22]. As each node in the graph is reached by the "why not" search algorithms, provenance data is collected. The search stops when offending nodes are found, avoiding recording of provenance for irrelevant nodes (i.e., query operators that may not be a cause of the missing "why not" tuples). "Why" and "why not" debugging has been implemented for other languages as well [80, 67, 66, 21]. For example, the WhyLine combines static and dynamic analysis techniques–including provenance recording–to answer "why" and "why not" questions about Java programs [67, 66, 80].

"Why" and "why not" debugging strategies are particularly interesting because they enable users to reason about the behavior of programs and queries at a conceptual (i.e., not code) level. Furthermore, these techniques combine previous algorithmic approaches in ways that magnify their strengths. For example, by constructing an initial graph using logical structures, and then pruning irrelevant query operators, one can spend less time on executing the query and thus improve the speed at which a debugging system can identify errors. However one drawback of these systems is their complexity, which makes them difficult to implement and maintain within already large and complicated development environments.

Query Steering. Some techniques allow users to modify queries by manipulating the execution outputs; called "query steering" [20, 3, 36], also called "query by example". These techniques allow users to add or remove tuples to the output to tell what should (or should not) be in the result. These methods then modify or add query predicates to capture the changes from the augmented results.

Takeaways. Hybrid debugging strategies rely on analysis of both the logical structure and execution structure of erroneous queries (or programs). These techniques answer "why" or "why not" questions about a query's behavior, and use logical query structure to efficiently model and search the space of possible query errors. They strategically collect partial provenance information and avoid executing the full query, saving both computation and storage. However, implementation challenges could pose an issue for commercial adoption.

Commercial Database Debugging Tools

We found several commercial query debugging tools: Embarcadero Technologies RapidSQL [58], Keeptool's PL/SQL Debugger [43], and Microsoft's T-SQL [29]. These are cross-platform tools that can be used with other programming tools like Eclipse[7] and Visual Studio[8]. They can connect to various DBMSs, including PostgreSQL[1], Microsoft SQL Server[2], Oracle[9], etc. Common features of these tools overlap with standard debugging tools, such as the ability to set breakpoints, to check dependencies between variables, and save input variable values to a file for later use.

Microsoft's U-SQL [28], provides performance-based debugging support. U-SQL is a big data query language and execution platform that combines declarative SQL with imperative custom languages, such as C# and Python. U-SQL decomposes a user's query into a workflow of smaller connected jobs. Similar to academic tools like Perfopticon [76], U-SQL reports on various performance characteristics as the query executes across multiple machines. However, this information is not presented to U-SQL users through visualizations.

Takeaways. Interestingly, commercial debugging tools do not seem to implement the state-of-the-art in query or program debugging research. As observed by Reiss for program debugging tools, it seems that database debugging tools also "...have not changed significantly over the last fifty years."[89]

Summary and Discussion

Here, we summarize our findings and highlight research questions for further investigation.

DBMS Users are Commonly Scripters and Programmers. a notable number of analysts use DBMSs regularly for data analysis, where many are comfortable with writing their own queries. Research across databases, visualization and HCI touches on how users interact with and analyze queries, but do not consider the challenge of debugging complex SQL queries.

Users' Database Debugging Strategies are Not Emphasized. Much of the relevant visualization and HCI literature is devoted to intuitive query interfaces. The relevant database literature focuses on efficiently recording execution behavior (i.e., database provenance and lineage), not debugging. Across areas, we see a dearth of rich information about users' database debugging strategies and tools. We also see relatively little work on query debugging tools for more code-savvy users.

Effective Debugging Tools Incorporate Features from Multiple Areas. Query debugging features seem to be more effective if adopted from multiple research areas, such as visualizations of program behaviour, intuitive interaction widgets, and efficient database provenance recording strategies. For

[7]http://eclipsesql.sourceforge.net/
[8]https://visualstudio.microsoft.com/vs/features/ssdt/
[9]https://www.oracle.com/index.html

example, the works of Abouzied et al. [3], Grust et al. [45], Moritz et al. [76] and Battle et al. [9] present systems that leverage database debugging techniques to produce effective human-centered query debugging interfaces. Furthermore, "why" and "why not" techniques seem to provide the best of all worlds in terms of efficiency and debugging support.

Gaps Between the Literature and Commercial Systems. Proposed debugging techniques have been around for over a decade. Yet none of the existing commercial database debugging tools that we reviewed seem to use these techniques. One reason could be that commercial tools assume that users debug queries the same way that they would debug Java or C++ programs (e.g., setting breakpoints, tracking variables, etc.) and thus lack techniques that go beyond this design.

Further Research Questions. From our review, we have the following questions we seek to answer through interviews with database users and designers:

- **R1**: What query interfaces (1) and debugging tools (2) do people use, and how and why did they select these tools?
- **R2**: What kinds of strategies (1) and visual aids (2) do users employ when debugging their database queries?
- **R3**: Why are state-of-the-art database debugging techniques in the literature not used in commercial tools?

To answer **R1** and **R2**, we interview a range of database users. To answer **R3**, we (informally) polled six database experts. In the next section, we discuss our study design and findings.

STUDY DESIGN

Here, we describe the details and intuition for our study design.

Participants

We conducted 18 in-person and two remote interviews with 20 database users total, a number consistent with other similar studies (e.g., [65]). We interviewed three different groups (16 male / 4 female, 18-54 years old):

- **Undergraduate Students** (6) – Participants pursuing or just completed undergraduate degrees in Computer Science or Information Systems.
- **Graduate Students** (4) – Participants pursuing graduate degrees in Computer Science and Business Analytics.
- **Industry Professionals** (10) – Participants working in industry for 3-25 years.

Our aim was to capture a diverse range of database users. In this case, students are relevant because they are *actively learning how to use DBMSs*, and thus may have insights into the debugging process that seasoned analysts no longer have.

We recruited all participants through public-facing online resources (e.g., mailing lists, online contact pages) and professional networks. Most participants were based near our home institution. However, participants came from the USA (12), India (6), China (1) and Croatia (1).

Protocol

Each participant first signed a consent form, then was asked to complete a demographic questionnaire (recording age group, gender, occupation, etc.). We also noted participants' experience levels with databases and SQL. We then followed a semi-structured approach to conduct interviews. First, we asked participants to walk us through two to three queries they had written in the past. We then asked about the strategies they used to debug their queries. Based on the responses, our follow-up questions fell into one of the following categories:

- Strategies for comparing actual and expected query results
- Debugging tools and techniques used to verify query results
- Strengths/weaknesses of mentioned tools and approaches
- Use of visual aids to help with debugging queries

Table 1 lists our interview questions. Interviews typically lasted one hour. Participants were given $20 compensation. This study was approved by our home institution's IRB.

ANALYSIS OF STUDY DATA

We analyzed our interview data (audio transcripts and handwritten notes) using an iterative coding method inspired by grounded theory [23]. We organize our findings around four themes derived from the data, which deviated somewhat from the categories outlined in section 3:

- Usage of general tools for querying DBMSs
- Awareness and usage of query debugging tools
- (High-level) Debugging strategies
- Use of visual aids for debugging queries

We iterated and refined these categories as we gathered more data through our interview studies.

Querying Tools Used (R1-1)

Participants shared a wide range of tools for querying DBMSs, ranging from minimalistic tools like text editors and command line interfaces, to powerful products like SAP Hana[10] and ORM languages (e.g., SQLAlchemy[3]). Other querying tools included database IDE's like MySQL Workbench[11], Oracle SQL Developer[12] and DBViewer[13]. We saw greater variety in tools used to query DBMSs with an increase in experience.

Undergraduates Preferred Simple Tools. We observed that 5 of 6 undergraduates favored the simplest tools for writing SQL queries (i.e., text editors and command line). The remaining participant (P2) was introduced to ORM frameworks in a summer internship, and prefers them over the "direct querying" approach used by other undergraduates. However, queries of undergraduate students were fairly simple (i.e., less than 15 lines long, executed on relatively small datasets), and many shared queries written as part of a database course. Thus, there did not appear to be a great need for powerful database tools.

Graduate Students Preferred IDEs. The 4 graduate students we interviewed showed a mix of tools used for querying, and reportedly used command line interfaces and database IDEs interchangeably. They mentioned that, IDEs saved querying time by providing predictive text and formatting the results in a more readable ways. While most of this group preferred IDEs, participants with additional DBMS experience preferred simpler tools. One participant responded:

> *There is no space for an entire heavy-weighted IDE... when you go into a cluster and need to perform queries like update[s], deletion or migration in stipulated time*

[10]https://www.sap.com/products/hana.html
[11]https://www.mysql.com/products/workbench/
[12]https://www.oracle.com/database/technologies/appdev/sql-developer.html
[13]https://marketplace.eclipse.org/content/dbviewer-plugin

Table 1. Interview Questions

Sr. No.	Category of Questions	Questions
1	Strategies for comparing actual and expected query results	What strategies do you use to evaluate whether database results match your expected results?
		What are some of the tools that help you through this evaluation process?
		Roughly how much time do you spend on debugging and updating your database queries?
2	Debugging tools and techniques used to verify query results	Are you aware of some tools for debugging database queries?
		What are some of the tools that you use to debug your queries?
		(If they do not use the tools) How are you confident of your queries and results?
		(If they do not use the tools) Why would you not use debuggers?
3	Strengths/weaknesses of mentioned tools and approaches	(If they use some tools) What are some of the features you like and some features that you dislike about these tools?
		(If they do not use the tools) What are some of the features you would like in a tool that could help you debug your queries?
		Do you use your own scripts or techniques instead of the debugging tools?
		(If they use scripts or own techniques) What additional benefits do you see in using your techniques? the debugging faster? If so, roughly by how much with respect to using the existing tools?
4	Use of visual aids to help with debugging queries	Can you share examples of your visual aids?
		Can you explain how you used these visual aids during the debugging process?
		How helpful were these visual aids when debugging your queries?
		(If they do not use visual aids) How helpful do you believe visual aids might be when debugging queries?
		(If they do not use visual aids) Can you think of any ways in which visual aids could be used to help you during the debugging process?

to keep the system up-to-date. I would just prefer the command line straight-up. (P4)

A reason for a mix of tools seen in graduate students was perhaps that queries spanned from 30-50 lines long.

Industry Participants Varied Widely in Preference. On the other hand, we see a wide range in usage among the 10 industry participants. Three participants (P15, P16, P17) primarily mentioned using IDEs for querying DBMSs, driven by the conventions of their respective teams at work. Two participants (P1, P14) preferred command line interfaces; these participants were experienced analysts and not developers (i.e., 20+ years industry experience). Two other participants (P5, P9) preferred ORMs, where they could avoid writing SQL; one responded:

Never directly, would I fire up a SQL interface to see the schema because I see it in a more readable format on the ORM! (P5)

Two participants (P18, P19) mentioned using Oracle Hyperion[14] and SAP Hana[6]. These participants are research analysts at large companies, and dealt with querying massive datasets. These participants shared that their queries were generally 200-3000 lines long and handling these large queries that pulled data from many sources and served millions of customers was possible only by these modern DBMSs. One participant said:

I have written reports that are 100-150 pages long. The queries used to generate these reports are very very complex and intertwined in nature. I heavily rely on Hyperion. (P18)

Takeaways. We find that users use and learn tools only with the effort that is absolutely necessary to get their work done. Many of our participants (8 of 20) still preferred command line interfaces for writing and issuing queries, regardless of experience level. In case of students, where they are still learning how to use DBMSs, there is low risk and low reward in putting significant effort into learning new database tools. However, with industry professionals, we see much more variety in their tool usage, because their database needs differ widely. At the extreme, we have serious database users with long, complicated queries which could affect thousands of customers. In this case, we see users preferring top-of-the-line database products to craft and execute their DBMS queries.

Debugging Tools Used (R1-2)

Few Participants Knew of SQL Debuggers. 16 of 20 (80%) participants were unaware of any database debugging tools. Of 4 participants who knew of debuggers, only 2 had used them, while others were unable to name any SQL debuggers. We found these results surprising, given IDEs usage of many participants' and the prevalence of debugging offerings in both the academic literature and across industry tools (section 2).

Undergraduates Were Wary of Debuggers. 5 of 6 undergraduate students were unaware of any database debuggers. The remaining participant stated: *"I am aware that database debuggers exist, but I haven't used any of them."(P2)*. In general, undergraduate students expressed a collective dislike for programming debuggers and used them very rarely.

Graduate Students Knew Program but Not SQL Debuggers. 3 of 4 graduate students reported using debuggers for other languages, and posited that there might be plug-ins for DBMSs (as seen in section 2), but have never seen or used them.

The remaining graduate participant (P11) had worked extensively with the SAP ABAP[15] debugger after her undergraduate. She reported being *"extremely daunted"* by debuggers in undergraduate, and only grew comfortable with debuggers after working in industry as a technical analyst. She describes using the debugger to set breakpoints, check values, change them if incorrect, and repeating this process until all conditions were passed successfully. Even when using the debugger, she recalls the process being tedious and highly repetitive, as she had to manually click buttons many times in the interface, and memorize the schema of different workflows if she was changing some variable values. Nevertheless, she was still grateful for having the debugger and states:

...it would roughly take me an hour and a half to debug [10 queries], but with plain SQL querying on command line, it could take up the whole day or even more... who knows? (P11)

[14] https://www.oracle.com/corporate/acquisitions/hyperion/

[15] https://developers.sap.com/topics/abap-platform.html

Industry Participants were Also Unaware of SQL Debuggers. 7 of 10 industry participants were also oblivious of database debuggers. One participant was *"not even sure if something like that even exists!"(P9)* The remaining 3 participants mentioned being aware of stack traces or debuggers provided by modern frameworks, but never used them.

Takeaways Awareness of database debuggers was low amongst participants (less than 20%). Except for one participant who was a debugging expert, database debuggers (and even program debuggers) were reportedly rarely used.

(High level) Debugging Strategies (R2-1).

Most Participants use Trial-and-Error. Most participants seem to use a crude trial-and-error approach, regardless of database query tool: they try writing or updating their query, and then manually review the raw results from the DBMS. For example, one graduate student said: *"I don't have a systematic way of doing it and I kind of go by trial and error."(P3)* Many industry participants also seemed to rely heavily on raw errors, for example one participant said they only used "*the errors SQL gives [him]*" *(P16)* to debug queries.

Raw DBMS Errors are Difficult to Interpret and Debug. Our participants stressed how painful it is for them to try and interpret raw exceptions from the DBMS, and how little help the exceptions are in understanding the errors in queries. A participant commented on database errors:

> *SQL errors are like just the worst. They just tell you you messed up and not what [is messed up/wrong]? Sometimes, it's also difficult to google it! (P19)*

Scanning Unexpected Results for Common Error Indicators. All 20 participants pointed to some key factors that they checked for in the case where the query didn't throw errors, but still exhibited unexpected behavior. Some participants mentioned that summary metrics output by database query tools (e.g., IDEs) could be useful for debugging, such as how long the query took to execute, or the number of rows returned. Common (manual) debugging metrics included:

- No records are returned (i.e., empty answer, subsection 2.7)
- Presence of NULL entries
- Presence of multiple entries (i.e., duplicates)
- Total rows returned for a query component

Even though these metrics seem simple to report, participants mentioned that none of their chosen tools reported these metrics by default; and they had to manually review the query results to compute them. Participants often used Microsoft Excel[16] for this purpose. They exported the results to a spreadsheet and used filters to evaluate their query results. However, this had clear limitations, for example a participant said that *"this can't work for complex and bigger queries." (P11)*

Debugging Complex Queries Using Nested Error Checking. We asked participants how they circumvented the above limitations with complex DBMS queries. We learned of the nested error checking, as a seemingly universal debugging strategy, used across nearly our entire participant pool. In one variant of this approach, participants break a complex query into a workflow of simpler components (e.g., SQL clauses, inner sub-queries), and debug the simplest component first. Once the first component is returning expected results, then the user moves on to the next simplest component (e.g., another clause, or outer sub-query), and so on. This strategy seems to align well with certain provenance-focused debugging techniques proposed in prior work (e.g., step-by-step illustrations of execution structure and why-not strategies).

We observed a variant of this strategy that involves participants constructing their own unit tests in the form of small, targeted inputs. For example, one participant said:

> *I would go to lower level development environments and work with a mock data which is similar to my actual data but just smaller, to try out my queries there. (P4)*

This debugging strategy has also been observed by Battle et al. for temporal queries [9]. Given a particular unit (represented by one or more records in the output), participants would manually evaluate this unit, to assess the correctness of their query. If the result(s) did not match expectations, the same iterative debugging strategies were used to add conditions to fix the unit. However, with this technique, participants could only hope that they had covered all of the needed test cases to ensure the query's correctness.

Alternative Debugging Methods. Some industry participants use conventional print statements after every query component to visit their query and find possible errors.

An alternative program debugging technique *"rubber duck debugging"* was discussed by an industry professional (P19), where you explain your code logic line by line and in the process understand the error for yourself. He explained:

> *90% of the times you understand what you're doing wrong if you talk. You think you've stared at the query long enough, and it's weird but talking helps logic, since you can't really ask people around you always as they are busy or have headphones on. (P19)*

Many participants also verified their logic by writing pseudocode, as comments situated within their queries and code.

Debugging is not Formally Taught. Both students and industry professionals believe that debuggers are not formally taught in undergrad or grad courses. 4 of 10 industry participants pointed to rely on their colleagues and bosses too to learn how to evaluate their queries. One participant pointed out:

> *I did a lot of pair programming with my boss when querying databases. It was the way I learnt. I still use a lot of the same techniques and behaviours that I learnt from my boss. (P1)*

However, apparently none of our participants' colleagues or senior mentors were seen regularly using debuggers either. One industry participant said:

> *I haven't seen any of my colleagues use debuggers. Also, during my training at [a major tech company], they would have told us if there were some tools much earlier. But, they haven't. (P16)*

Thus, there seems to be a lack of educational infrastructure for debugging in general, which may explain in part the deficiencies in participants' database debugging strategies.

[16] https://office.live.com/start/Excel.aspx

Takeaways. Participants' debugging strategies seem to coalesce around a low-level, trial-and-error process, which can lead to considerable (and unpleasant) effort spent debugging database queries. Even worse, participants seemed to actively dislike using standard program debuggers until developing a mastery of them. Lack of educational support seems to be a key issue highlighted by multiple participants, which future debugging tools could address. However, observed debugging strategies do seem to align with proposed solutions, such as step-by-step illustration of execution structure (e.g., [45, 9] and "why" and "why not" debugging (e.g., [22, 21, 66]).

Visual Aids Used (R2-2)

We asked about any visual aids that might be used by the participants to help them debug their queries. Most observed techniques were based off manual drawing of flowcharts, showing the process to be followed to reach the final results (also observed by Battle et al. [9]). However, participants generally tracked only part of the results to test certain conditions (i.e., the unit-test scenario mentioned above). Some participants would also write down the database schema(s) to have them on hand while writing the queries. In general, 9 of 20 participants created their own visual aids on paper.

ER Diagrams are Bad for Debugging. The visual aid recommended most by participants was the Entity Relationship (ER) diagram [24], which illustrates the structure of a database schema. However, only 50% of the participants, mostly industry participants, actually used this visual aid, and generally not for debugging. ER diagrams were mostly drawn as part of documentation, or by hand on white boards in team meetings (i.e., for communication purposes). A participant notes the limitations of ER diagrams for debugging:

> *ER diagram helps in understanding the schema but I want to see how something in the beginning of the my query affects something at the end of my query. This knowledge is lost in the ER diagram. This is the minute thing I want the visual aid to capture, especially if it is complex! (P8)*

Other Debugging Visualizations from Commercial Tools. Some participants mentioned that powerful tools like Hadoop[17] and SAP Hana[6] provide features of visualising the data as bar charts, though a participant pointed out that:

> *It's not really helpful, as it only visualises up to 10,000 rows. Having a visualization of only 10,000 rows in a database of a million rows, is not really helpful, making the feature useless. (P19)*

Participant Recommendations for Visual Aids. Many of the participants had recommendations of visual aids that could aid their debugging process. Most suggestions involved displaying intermediate results of specific portions of the queries (similar to the approach proposed by Grust et al. [45]), highlighting the trace of where certain tuples came from (similar to the approach proposed by Battle et al. [9]), and having some kind of documentation for keywords, especially when working with different standards of one database language. One participant suggested a visualization showing how certain attributes transformed over the iterations of building the query, which could be interesting future work.

[17] `https://hadoop.apache.org/`

9 of 20 participants mentioned that they derive query logic entirely in their heads. Of these participants, 7 said visual aids could still be helpful, mostly for tracking the nested error checking debugging strategy, which sounds similar to the visualization technique proposed by Battle et al. [9]. The remaining 2 said that visual aids would not help them and they would stick with the command line, as one said, *"It's faster for me to process it in my head than it is for me to put it down somewhere." (P1)*

Takeaways. Some visual aids mentioned by users already have some support in commercial database tools (e.g., very simple charts, ER diagrams, and basic flow charts of low-level query operators). Interestingly, many of the visual aids recommended by our participants have already been developed in the debugging literature, providing further support for these techniques. However, most of these visual aids are *not* supported by existing commercial tools (as observed in section 2).

Summary

Here, we summarise the key takeaways of our interview study.

Lack of Training and Awareness of Database Debugging Tools (R1). 19 of 20 participants we interviewed relied on informal, manual debugging techniques developed or learned on their own regardless of the size and complexity of the queries. 17 of 20 participants were unaware of any kind of database debuggers. Of 20, only one had actually used a database debugger. Many participants mentioned a lack of training in school and their workplace for proper usage of debugging tools in general.

Participants Favor Nested Debugging Strategies (R2-1). All 20 participants used the common strategy of breaking down a query into a workflow of simpler components, and debugging incrementally to get to a final, error-less query. All participants seemed to perform this process manually, oftentimes on paper. However, techniques are proposed in the literature to provide similar visual aids automatically (e.g., [45, 9]).

Participants See Value in Having Better Visual Aids (R2-2). 15 of 20 participants described using some form of visual aid to write and debug queries. The most common visual aids used for querying were ER diagrams, though they were not helpful in debugging. For debugging, participants pointed to vast use of flowcharts, scribbling names of tables and attributes, writing pseudo codes to get to final results and highlighting rows in tables as visual aids. Of the 5 participants that did not actively use visual aids, 4 could still see their value for debugging, especially if queries were long and complex.

FEEDBACK FROM DATABASE EXPERTS (R3)

Our analysis provided interesting insights, however, one question remained: if users want new debugging functionality, and this functionality has already been proposed in the academic literature, why is it not available in commercial tools? We reached out to six database experts with experience in building commercial database tools. We asked them two questions to both sanity check and to provide additional context for our findings: 1) is there interest in industry in database query debugging; and 2) why have proposed techniques not been implemented in commercial tools? The expert's key takeaways:

Query Debugging is Important, But Not the Top Priority. We found that all experts agreed that query debugging is an important research problem to address and there is some industry interest. Most experts suggested that may be more pressing concerns for database companies, such as supporting database automation and performance, which could explain in part why we observe a gap between the literature and commercial tools.

Collaborations With Industry May Be Critical to Improving Debugging Tools. One expert noted that some techniques may be too complicated and too specific to be useful in most real-world debugging contexts. In particular, why not debugging requires users to carefully articulate positive and negative examples, either using a programming language or by manually labeling individual records, neither of which may be any easier or faster to execute over a basic nested debugging strategy.

We see active efforts in developing new debugging methods in academia but limited resources to rigorously test techniques, and limited efforts in industry to improve debugging methods but many opportunities for testing them with real users. Thus there seems to be an opportunity for academics to effect strong positive change in industry through an active collaboration.

SUMMARY AND DISCUSSION

In this paper, we aim to understand how database users debug their database queries, summarize what techniques have been proposed in the literature to support query debugging, and ultimately identify specific ways in which we can better help these users to utilize DBMSs more effectively moving forward. To better understand what query debugging techniques exist in the literature, we reviewed 112 papers and tools across databases, visualization, and HCI. Hybrid techniques from multiple research areas seem to provide better user support *and* improved system performance. However, commercial tools seem to only provide basic program debugging support.

To contextualize proposed techniques with users' experience, we interviewed 20 database users, ranging from undergraduate students just learning how to use DBMSs, to industry veterans leveraging powerful, top-of-the-line DBMSs. However, across expertise levels, we saw roughly the same story: participants were largely unaware of database debugging tools (even commercial ones), and resorted to low-level, manual debugging strategies that relied heavily on raw output from the DBMS. We noted an aversion to using standard debugging tools (even general program debuggers), which participants suggested could stem from a lack of educational infrastructure for how to use these tools. However users' debugging strategies do seem to align well with some existing solutions (e.g., [45, 9]).

Finally, we polled six database experts and learned: debugging support is important but not a high priority in industry; and proposed techniques need more rigorous evaluation to clarify how helpful they truly are in real-world debugging scenarios.

Design Recommendations

From our findings, we make the following suggestions for improving debugging tools in the future:

Design Tools to Integrate into Existing Workflows. Many of our participants preferred their minimalist workflows. New debugging tools may be more effective if they can be integrated with existing command line tools or IDEs. Furthermore, our participants worked with a range of commercial DBMSs, so it may prove useful to support multiple commercial DBMSs, which may have the added benefit of making it easier to establish collaborations in industry to build new tools.

Prioritize Teaching Users How to Debug Queries Effectively. Our participants stressed a lack of formal training with debugging tools. Educational modules could be developed to fill this gap. For example, to help new DBMS users, we can incorporate hover-triggered popups to explain database-specific terms as users interact with queries (and DBMS errors), such as what a relation is or how foreign keys work. However, our participants also seemed loath to learn how to use standard program debuggers, let alone query debuggers. Thus we see an opportunity to make debuggers easy to use and also *easy for users to learn on their own*. For example, we could analyze users' interactions with the DBMS to detect experience levels, and use this information to deploy more learning-focused interactions (e.g., novice users see terminology popups, but expert users do not). Furthermore, *avoiding problematic assumptions about database expertise when designing new features*, e.g., assuming all users are database (and data) experts (e.g., have the schema memorized), could benefit users.

Automate Features to Support Nested Debugging Strategies. Our participants seemed to favor a nested approach to debugging queries, and many hand-drew simple visual aids as part of this process. These visual aids seem straightforward to automate, and have been suggested in the literature (e.g., [45, 9]). These *existing techniques should be prioritized as a starting point* for improving query debugging features. Furthermore, we believe *automated presentation of metadata for debugging purposes* to be a potentially fruitful avenue for future work.

Annotation features could also help users track their debugging progress. For example, color highlighting and text notes could help users keep track of query components already tested so far and employ nested debugging more effectively. Furthermore, these annotations could also be analyzed to develop smarter query testing features, such as generating unit tests automatically for specific query components.

Limitations

One limitation of this work is that we did not poll a large fraction of the database user population. As such, there may exist debugging strategies, tools, or visual aids that are not reflected in our results. However, our findings do corroborate key points and techniques observed in prior work on query debugging, suggesting that our users exhibit some known characteristics observed in other database user populations.

Another potential limitation lies in the framing of our study design. By explicitly asking users in our study about how they incorporated visual aids in their debugging process, we may not have observed the participants' natural usage of visual aids in terms of frequency and significance. However, we do find that users see value in visual aids, and some incorporate visual aids as part of their regular debugging process.

REFERENCES

[1] 2018. VISSOFT 2018. (2018). http://vissoft18.etsii.urjc.es/

[2] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling Relational Data: A Survey. *The VLDB Journal* 24, 4 (Aug. 2015), 557–581. DOI: http://dx.doi.org/10.1007/s00778-015-0389-y

[3] Azza Abouzied, Joseph Hellerstein, and Avi Silberschatz. 2012. DataPlay: Interactive Tweaking and Example-driven Correction of Graphical Database Queries. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 207–218. DOI:http://dx.doi.org/10.1145/2380116.2380144 event-place: Cambridge, Massachusetts, USA.

[4] Christopher Ahlberg. 1996. Spotfire: An Information Exploration Environment. *SIGMOD Rec.* 25, 4 (Dec. 1996), 25–29. DOI: http://dx.doi.org/10.1145/245882.245893

[5] A. Aiken, J. Chen, M. Stonebraker, and A. Woodruff. 1996. Tioga-2: a direct manipulation database visualization environment. In *Proceedings of the Twelfth International Conference on Data Engineering*. 208–217. DOI: http://dx.doi.org/10.1109/ICDE.1996.492109

[6] S. Alspaugh, N. Zokaei, A. Liu, C. Jin, and M. A. Hearst. 2019. Futzing and Moseying: Interviews with Professional Data Analysts on Exploration Practices. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan. 2019), 22–31. DOI: http://dx.doi.org/10.1109/TVCG.2018.2865040

[7] I. Bacher, B. M. Namee, and J. D. Kelleher. 2016. On Using Tree Visualisation Techniques to Support Source Code Comprehension. In *2016 IEEE Working Conference on Software Visualization (VISSOFT)*. 91–95. DOI: http://dx.doi.org/10.1109/VISSOFT.2016.8

[8] Thomas Ball and Sriram K. Rajamani. 2002. The SLAM Project: Debugging System Software via Static Analysis. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '02)*. ACM, New York, NY, USA, 1–3. DOI: http://dx.doi.org/10.1145/503272.503274 event-place: Portland, Oregon.

[9] Leilani Battle, Danyel Fisher, Robert DeLine, Mike Barnett, Badrish Chandramouli, and Jonathan Goldstein. 2016. Making Sense of Temporal Queries with Interactive Visualization. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 5433–5443. DOI: http://dx.doi.org/10.1145/2858036.2858408 event-place: San Jose, California, USA.

[10] Leilani Battle and Jeffrey Heer. 2019. Characterizing Exploratory Visual Analysis: A Literature Review and Evaluation of Analytic Provenance in Tableau. *Computer Graphics Forum* 38, 3 (2019), 145–159. DOI:http://dx.doi.org/10.1111/cgf.13678

[11] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo. 2005. VisTrails: enabling interactive multiple-view visualizations. In *VIS 05. IEEE Visualization, 2005.* 135–142. DOI: http://dx.doi.org/10.1109/VISUAL.2005.1532788

[12] F. Beck, H. A. Siddiqui, A. Bergel, and D. Weiskopf. 2017. Method Execution Reports: Generating Text and Visualization to Describe Program Behavior. In *2017 IEEE Working Conference on Software Visualization (VISSOFT)*. 1–10. DOI: http://dx.doi.org/10.1109/VISSOFT.2017.11

[13] Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. 2006. ULDBs: Databases with Uncertainty and Lineage. In *Proceedings of the 32Nd International Conference on Very Large Data Bases (VLDB '06)*. VLDB Endowment, 953–964. http://dl.acm.org/citation.cfm?id=1182635.1164209 event-place: Seoul, Korea.

[14] Nicole Bidoit, Melanie Herschel, and Katerina Tzompanaki. 2014. Query-Based Why-Not Provenance with NedExplain. In *in "EDBT - 17th International Conference on Extending Database Technology*.

[15] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and Where: A Characterization of Data Provenance. In *Database Theory - ICDT 2001 (Lecture Notes in Computer Science)*, Jan Van den Bussche and Victor Vianu (Eds.). Springer Berlin Heidelberg, 316–330.

[16] Rafael Caballero, Yolanda Garcia-Ruiz, and Fernando Saenz-Perez. 2012a. Algorithmic Debugging of SQL Views. In *Perspectives of Systems Informatics (Lecture Notes in Computer Science)*, Edmund Clarke, Irina Virbitskaite, and Andrei Voronkov (Eds.). Springer Berlin Heidelberg, 77–85.

[17] Rafael Caballero, Yolanda Garcia-Ruiz, and Fernando Saenz-Perez. 2012b. Declarative Debugging of Wrong and Missing Answers for SQL Views. In *Functional and Logic Programming (Lecture Notes in Computer Science)*, Tom Schrijvers and Peter Thiemann (Eds.). Springer Berlin Heidelberg, 73–87.

[18] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos E. Scheidegger, Claudio T. Silva, and Huy T. Vo. 2006. VisTrails: Visualization Meets Data Management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD '06)*. ACM, New York, NY, USA, 745–747. DOI: http://dx.doi.org/10.1145/1142473.1142574 event-place: Chicago, IL, USA.

[19] TIZIANA Catarci, MARIA F. Costabile, STEFANO Levialdi, and CARLO Batini. 1997. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages & Computing* 8, 2 (April 1997), 215–260. DOI:http://dx.doi.org/10.1006/jvlc.1997.0037

[20] Ugur Cetintemel, Mitch Cherniack, Justin DeBrabant, Yanlei Diao, Kyriaki Dimitriadou, Alex Kalinin, Olga Papaemmanouil, and Stan Zdonik. 2013. Query Steering for Interactive Data Exploration *(CIDR '13)*.

[21] Hans Chalupsky and Thomas A. Russ. 2002. WhyNot: Debugging Failed Queries in Large Knowledge Bases. In *Proceedings of the 14th Conference on Innovative Applications of Artificial Intelligence - Volume 1 (IAAI'02)*. AAAI Press, 870–877. http://dl.acm.org/citation.cfm?id=1650083.1650093 event-place: Edmonton, Alberta, Canada.

[22] Adriane Chapman and H. V. Jagadish. 2009. Why Not?. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD '09)*. ACM, New York, NY, USA, 523–534. DOI:http://dx.doi.org/10.1145/1559845.1559901 event-place: Providence, Rhode Island, USA.

[23] Kathy Charmaz and Linda Liska Belgrave. 2015. Grounded Theory. In *The Blackwell Encyclopedia of Sociology*. American Cancer Society. DOI:http://dx.doi.org/10.1002/9781405165518.wbeosg070.pub2

[24] Peter Pin-Shan Chen. 1976. The Entity-relationship Model-Toward a Unified View of Data. *ACM Trans. Database Syst.* 1, 1 (March 1976), 9–36. DOI: http://dx.doi.org/10.1145/320434.320440

[25] Luca Chittaro and Carlo Combi. 2003. Visualizing queries on databases of temporal histories: new metaphors and their evaluation. *Data & Knowledge Engineering* 44, 2 (Feb. 2003), 239–264. DOI: http://dx.doi.org/10.1016/S0169-023X(02)00137-4

[26] Shumo Chu, Chenglong Wang, Konstantin Weitz, and Alvin Cheung. 2017. Cosette: An Automated Prover for SQL.

[27] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data Cleaning: Overview and Emerging Challenges. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 2201–2206. DOI: http://dx.doi.org/10.1145/2882903.2912574 event-place: San Francisco, California, USA.

[28] Microsoft Corporation. 2017a. Get started with U-SQL language in Azure Data Lake Analytics. (June 2017). https://docs.microsoft.com/en-us/azure/data-lake-analytics/data-lake-analytics-u-sql-get-started

[29] Microsoft Corporation. 2017b. Transact-SQL Debugger - SQL Server. (March 2017). https://docs.microsoft.com/en-us/sql/ssms/scripting/transact-sql-debugger

[30] Isabel F. Cruz. 1992. DOODLE: A Visual Language for Object-oriented Databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data (SIGMOD '92)*. ACM, New York, NY, USA, 71–80. DOI: http://dx.doi.org/10.1145/130283.130299 event-place: San Diego, California, USA.

[31] Y. Cui and J. Widom. 2000. Practical lineage tracing in data warehouses. In *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*. 367–378. DOI: http://dx.doi.org/10.1109/ICDE.2000.839437

[32] Y. Cui and J. Widom. 2003. Lineage Tracing for General Data Warehouse Transformations. *The VLDB Journal* 12, 1 (May 2003), 41–58. DOI: http://dx.doi.org/10.1007/s00778-002-0083-8

[33] Jonathan Danaparamita and Wolfgang Gatterbauer. 2011. QueryViz: Helping Users Understand SQL Queries and Their Patterns. In *Proceedings of the 14th International Conference on Extending Database Technology (EDBT/ICDT '11)*. ACM, New York, NY, USA, 558–561. DOI: http://dx.doi.org/10.1145/1951365.1951440 event-place: Uppsala, Sweden.

[34] Wim De Pauw, Mihai Letia, Bugra Gedik, Henrique Andrade, Andy Frenkiel, Michael Pfeifer, and Daby Sow. 2010. Visual Debugging for Stream Processing Applications. In *Runtime Verification (Lecture Notes in Computer Science)*, Howard Barringer, Ylies Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann (Eds.). Springer Berlin Heidelberg, 18–35.

[35] Walter Didimo, Francesco Giacche, and Fabrizio Montecchiani. 2015. Kojaph: Visual Definition and Exploration of Patterns in Graph Databases. In *Graph Drawing and Network Visualization (Lecture Notes in Computer Science)*, Emilio Di Giacomo and Anna Lubiw (Eds.). Springer International Publishing, 272–278.

[36] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. 2014. Explore-by-example: An Automatic Query Steering Framework for Interactive Data Exploration. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*. ACM, New York, NY, USA, 517–528. DOI:http://dx.doi.org/10.1145/2588555.2610523 event-place: Snowbird, Utah, USA.

[37] K. Venkatesh Emani, Tejas Deshpande, Karthik Ramachandra, and S. Sudarshan. 2017. DBridge: Translating Imperative Code to SQL. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. ACM, New York, NY, USA, 1663–1666. DOI: http://dx.doi.org/10.1145/3035918.3058747 event-place: Chicago, Illinois, USA.

[38] K. Venkatesh Emani, Karthik Ramachandra, Subhro Bhattacharya, and S. Sudarshan. 2016. Extracting Equivalent SQL from Imperative Code in Database Applications. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 1781–1796. DOI: http://dx.doi.org/10.1145/2882903.2882926 event-place: San Francisco, California, USA.

[39] Facebook. 2015. GraphQL. (2015). http://facebook.github.io/graphql/

[40] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring. 2013. Live trace visualization for comprehending large software landscapes: The ExplorViz approach. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. 1–4. DOI: http://dx.doi.org/10.1109/VISSOFT.2013.6650536

[41] J. Freire, D. Koop, E. Santos, and C. T. Silva. 2008. Provenance for Computational Tasks: A Survey. *Computing in Science Engineering* 10, 3 (May 2008), 11–21. DOI:http://dx.doi.org/10.1109/MCSE.2008.79

[42] Boris Glavic. 2010. *Perm: efficient provenance support for relational databases*. Ph.D. Dissertation. University of Zurich.

[43] KeepTool GmbH. 2019. PL/SQL Debugger. (2019). https://keeptool.com/pl-sql-debugger.html

[44] David Gotz and Michelle X. Zhou. 2009. Characterizing Users' Visual Analytic Activity for Insight Provenance. *Information Visualization* 8, 1 (Jan. 2009), 42–55. DOI: http://dx.doi.org/10.1057/ivs.2008.31

[45] Torsten Grust, Fabian Kliebhan, Jan Rittinger, and Tom Schreiber. 2011. True Language-level SQL Debugging. In *Proceedings of the 14th International Conference on Extending Database Technology (EDBT/ICDT '11)*. ACM, New York, NY, USA, 562–565. DOI: http://dx.doi.org/10.1145/1951365.1951441 event-place: Uppsala, Sweden.

[46] Torsten Grust, Jan Rittinger, and Jens Teubner. 2007. Data-intensive XQuery Debugging with Instant Replay. In *Proceedings of the 4th International Workshop on XQuery Implementation, Experience and Perspectives (XIME-P '07)*. ACM, New York, NY, USA, 4:1–4:6. DOI:http://dx.doi.org/10.1145/1328158.1328162 event-place: Beijing, China.

[47] H. Guo, S. R. Gomez, C. Ziemkiewicz, and D. H. Laidlaw. 2016. A Case Study Using Visualization Interaction Logs and Insight Metrics to Understand How Analysts Arrive at Insights. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (Jan. 2016), 51–60. DOI: http://dx.doi.org/10.1109/TVCG.2015.2467613

[48] N. Hawes, S. Marshall, and C. Anslow. 2015. CodeSurveyor: Mapping large-scale software to aid in code comprehension. In *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*. 96–105. DOI: http://dx.doi.org/10.1109/VISSOFT.2015.7332419

[49] J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. 2008. Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation. *IEEE Transactions on Visualization and Computer Graphics* 14, 6 (Nov. 2008), 1189–1196. DOI: http://dx.doi.org/10.1109/TVCG.2008.137

[50] Jeffrey Heer and Ben Shneiderman. 2012. Interactive Dynamics for Visual Analysis. *Queue* 10, 2 (Feb. 2012), 30:30–30:55. DOI: http://dx.doi.org/10.1145/2133416.2146416

[51] Melanie Herschel, Ralf Diestelkamper, and Houssem Ben Lahmar. 2017. A Survey on Provenance: What for? What Form? What from? *The VLDB Journal* 26, 6 (Dec. 2017), 881–906. DOI: http://dx.doi.org/10.1007/s00778-017-0486-1

[52] Stacie Hibino and Elke A. Rundensteiner. 1997. User Interface Evaluation of a Direct Manipulation Temporal Visual Query Language. In *Proceedings of the Fifth ACM International Conference on Multimedia (MULTIMEDIA '97)*. ACM, New York, NY, USA, 99–107. DOI: http://dx.doi.org/10.1145/266180.266342 event-place: Seattle, Washington, USA.

[53] Harry Hochheiser and Ben Shneiderman. 2004. Dynamic Query Tools for Time Series Data Sets: Timebox Widgets for Interactive Exploration. *Information Visualization* 3, 1 (March 2004), 1–18. DOI:http://dx.doi.org/10.1057/palgrave.ivs.9500061

[54] Jane Hoffswell, Arvind Satyanarayan, and Jeffrey Heer. 2016. Visual Debugging Techniques for Reactive Data Visualization. *Computer Graphics Forum* 35, 3 (2016), 271–280. DOI:http://dx.doi.org/10.1111/cgf.12903

[55] Jane Hoffswell, Arvind Satyanarayan, and Jeffrey Heer. 2018. Augmenting Code with In Situ Visualizations to Aid Program Understanding. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, 532:1–532:12. DOI: http://dx.doi.org/10.1145/3173574.3174106 event-place: Montreal QC, Canada.

[56] Jiansheng Huang, Ting Chen, AnHai Doan, and Jeffrey F. Naughton. 2008. On the Provenance of Non-answers to Queries over Extracted Data. *Proc. VLDB Endow.* 1, 1 (Aug. 2008), 736–747. DOI: http://dx.doi.org/10.14778/1453856.1453936

[57] Kai Huang, Huey Eng Chua, Sourav S. Bhowmick, Byron Choi, and Shuigeng Zhou. 2019. CATAPULT: Data-driven Selection of Canned Patterns for Efficient Visual Graph Query Formulation. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. ACM, New York, NY, USA, 900–917. DOI:http://dx.doi.org/10.1145/3299869.3300072 event-place: Amsterdam, Netherlands.

[58] Inc. Idera. 2019. Rapid SQL | SQL IDE. (2019). https://www.idera.com/rapid-sql-ide

[59] R. Ikeda, J. Cho, C. Fang, S. Salihoglu, S. Torikai, and J. Widom. 2012. Provenance-Based Debugging and Drill-Down in Data-Oriented Workflows. In *2012 IEEE 28th International Conference on Data Engineering*. 1249–1252. DOI: http://dx.doi.org/10.1109/ICDE.2012.118

[60] Shrainik Jain, Dominik Moritz, Daniel Halperin, Bill Howe, and Ed Lazowska. 2016. SQLShare: Results from a Multi-Year SQL-as-a-Service Experiment. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 281–293. DOI: http://dx.doi.org/10.1145/2882903.2882957 event-place: San Francisco, California, USA.

[61] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 3363–3372. DOI: http://dx.doi.org/10.1145/1978942.1979444 event-place: Vancouver, BC, Canada.

[62] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. 2012. Enterprise Data Analysis and Visualization: An Interview Study. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (Dec. 2012), 2917–2926. DOI: http://dx.doi.org/10.1109/TVCG.2012.219

[63] Y. Kang and J. Stasko. 2012. Examining the Use of a Visual Analytics System for Sensemaking Tasks: Case Studies with Domain Experts. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (Dec. 2012), 2869–2878. DOI: http://dx.doi.org/10.1109/TVCG.2012.224

[64] Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. 2010. Querying Data Provenance. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*. ACM, New York, NY, USA, 951–962. DOI: http://dx.doi.org/10.1145/1807167.1807269 event-place: Indianapolis, Indiana, USA.

[65] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2016. The Emerging Role of Data Scientists on Software Development Teams. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, New York, NY, USA, 96–107. DOI: http://dx.doi.org/10.1145/2884781.2884783 event-place: Austin, Texas.

[66] A. Ko and B. Myers. 2008. Debugging reinvented. In *2008 ACM/IEEE 30th International Conference on Software Engineering*. 301–310. DOI: http://dx.doi.org/10.1145/1368088.1368130

[67] Andrew J. Ko and Brad A. Myers. 2004. Designing the Whyline: A Debugging Interface for Asking Questions About Program Behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 151–158. DOI: http://dx.doi.org/10.1145/985692.985712 event-place: Vienna, Austria.

[68] S. Lee, S. Kohler, B. Ludascher, and B. Glavic. 2017. A SQL-Middleware Unifying Why and Why-Not Provenance for First-Order Queries. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 485–496. DOI: http://dx.doi.org/10.1109/ICDE.2017.105

[69] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proc. VLDB Endow.* 8, 1 (Sept. 2014), 73–84. DOI: http://dx.doi.org/10.14778/2735461.2735468

[70] Fei Li, Tianyin Pan, and Hosagrahar V. Jagadish. 2014. Schema-free SQL. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*. ACM, New York, NY, USA, 1051–1062. DOI: http://dx.doi.org/10.1145/2588555.2588571 event-place: Snowbird, Utah, USA.

[71] Tom Lieber, Joel R. Brandt, and Rob C. Miller. 2014. Addressing Misconceptions About Code with Always-on Programming Visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2481–2490. DOI: http://dx.doi.org/10.1145/2556288.2557409 event-place: Toronto, Ontario, Canada.

[72] Hongjun Lu, Hock Chuan Chan, and Kwok Kee Wei. 1993. A Survey on Usage of SQL. *SIGMOD Rec.* 22, 4 (Dec. 1993), 60–65. DOI: http://dx.doi.org/10.1145/166635.166656

[73] W. S. Luk and Steve Kloster. 1986. ELFS: English Language from SQL. *ACM Trans. Database Syst.* 11, 4 (Dec. 1986), 447–472. DOI: http://dx.doi.org/10.1145/7239.384276

[74] Miro Mannino and Azza Abouzied. 2018. Qetch: Time Series Querying with Expressive Sketches. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. ACM, New York, NY, USA, 1741–1744. DOI: http://dx.doi.org/10.1145/3183713.3193547 event-place: Houston, TX, USA.

[75] Zhengjie Miao, Sudeepa Roy, and Jun Yang. 2019. RATest: Explaining Wrong Relational Queries Using Small Examples. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. ACM, New York, NY, USA, 1961–1964. DOI: http://dx.doi.org/10.1145/3299869.3320236 event-place: Amsterdam, Netherlands.

[76] Dominik Moritz, Daniel Halperin, Bill Howe, and Jeffrey Heer. 2015. Perfopticon: Visual Query Analysis for Distributed Databases. *Computer Graphics Forum* 34, 3 (2015), 71–80. DOI: http://dx.doi.org/10.1111/cgf.12619

[77] Davide Mottin, Alice Marascu, Senjuti Basu Roy, Gautam Das, Themis Palpanas, and Yannis Velegrakis. 2014. IQR: An Interactive Query Relaxation System for the Empty-answer Problem. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*. ACM, New York, NY, USA, 1095–1098. DOI: http://dx.doi.org/10.1145/2588555.2594512 event-place: Snowbird, Utah, USA.

[78] Davide Mottin, Alice Marascu, Senjuti Basu Roy, Gautam Das, Themis Palpanas, and Yannis Velegrakis. 2013. A Probabilistic Optimization Framework for the Empty-answer Problem. *Proc. VLDB Endow.* 6, 14 (Sept. 2013), 1762–1773. DOI: http://dx.doi.org/10.14778/2556549.2556560

[79] X. Mou, H. M. Jamil, and X. Ma. 2017. VisFlow: A Visual Database Integration and Workflow Querying System. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 1421–1422. DOI: http://dx.doi.org/10.1109/ICDE.2017.204

[80] Brad A. Myers, David A. Weitzman, Andrew J. Ko, and Duen H. Chau. 2006. Answering Why and Why Not Questions in User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 397–406. DOI: http://dx.doi.org/10.1145/1124772.1124832 event-place: Montreal, Quebec, Canada.

[81] Lee Naish. 1997. A Declarative Debugging Scheme. *Journal of Functional and Logic Programming* (1997).

[82] N. Noughi and A. Cleve. 2015. Conceptual interpretation of SQL execution traces for program comprehension. In *2015 IEEE 6th International Workshop on Program Comprehension through Dynamic Analysis (PCODA)*. 19–24. DOI: http://dx.doi.org/10.1109/PCODA.2015.7067179

[83] Vicky Papavasileiou, Ken Yocum, and Alin Deutsch. 2019. Ariadne: Online Provenance for Big Graph Analytics. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. ACM, New York, NY, USA, 521–536. DOI: http://dx.doi.org/10.1145/3299869.3300091 event-place: Amsterdam, Netherlands.

[84] F. Petrillo, G. Lacerda, M. Pimenta, and C. Freitas. 2015. Visualizing interactive and shared debugging sessions. In *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*. 140–144. DOI: http://dx.doi.org/10.1109/VISSOFT.2015.7332425

[85] R. Pienta, F. Hohman, A. Endert, A. Tamersoy, K. Roundy, C. Gates, S. Navathe, and D. H. Chau. 2018. VIGOR: Interactive Visual Exploration of Graph Query Results. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (Jan. 2018), 215–225. DOI: http://dx.doi.org/10.1109/TVCG.2017.2744898

[86] S. Podila and Y. Zhu. 2016. A Visualization Tool for 3D Graphics Program Comprehension and Debugging. In *2016 IEEE Working Conference on Software Visualization (VISSOFT)*. 111–115. DOI: http://dx.doi.org/10.1109/VISSOFT.2016.23

[87] Eric Prud'hommeaux and Andy Seaborne. 2008. SPARQL Query Language for RDF. W3C Recommendation. (15 January 2008). http://www.w3.org/TR/rdf-sparql-query/ http://www.w3.org/TR/rdf-sparql-query/.

[88] E. D. Ragan, A. Endert, J. Sanyal, and J. Chen. 2016. Characterizing Provenance in Visualization and Data Analysis: An Organizational Framework of Provenance Types and Purposes. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (Jan. 2016), 31–40. DOI: http://dx.doi.org/10.1109/TVCG.2015.2467551

[89] S. P. Reiss. 2014. The Challenge of Helping the Programmer during Debugging. In *2014 Second IEEE Working Conference on Software Visualization*. 112–116. DOI: http://dx.doi.org/10.1109/VISSOFT.2014.27

[90] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Re. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proc. VLDB Endow.* 10, 11 (Aug. 2017), 1190–1201. DOI: http://dx.doi.org/10.14778/3137628.3137631

[91] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 341–350.

[92] B. Shneiderman. 1994. Dynamic queries for visual information seeking. *IEEE Software* 11, 6 (Nov. 1994), 70–77. DOI:http://dx.doi.org/10.1109/52.329404

[93] Ben Shneiderman. 1996. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages (VL '96)*. IEEE Computer Society, Washington, DC, USA, 336–. http://dl.acm.org/citation.cfm?id=832277.834354

[94] Josep Silva. 2011. A survey on algorithmic debugging strategies. *Advances in Engineering Software* 42, 11 (Nov. 2011), 976–991. DOI: http://dx.doi.org/10.1016/j.advengsoft.2011.05.024

[95] C. Stolte, D. Tang, and P. Hanrahan. 2002. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics* 8, 1 (Jan. 2002), 52–65. DOI: http://dx.doi.org/10.1109/2945.981851

[96] Michael Stonebraker, Daniel Bruckner, Ihab F. Ilyas, Geoorge Beskales, Mitch Cherniack, Stanley Zdonik, Alexander Pagan, and Shan Xu. 2013. Data Curation at Scale: The Data Tamer System.

[97] Michael Stonebraker, Jolly Chen, Nobuko Nathan, Caroline Paxson, Alan Su, and Jiang Wu. 1993. Tioga: A Database-oriented Visualization Tool. In *Proceedings of the 4th Conference on Visualization '93 (VIS '93)*. IEEE Computer Society, Washington, DC, USA, 86–93. http://dl.acm.org/citation.cfm?id=949845.949866 event-place: San Jose, California.

[98] Wang Chiew Tan. 2007. Provenance in databases: past, current, and future. *IEEE Data Eng. Bull.* 30, 4 (2007), 3–12.

[99] Pawel Terlecki, Fei Xu, Marianne Shaw, Valeri Kim, and Richard Wesley. 2015. On Improving User Response Times in Tableau. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*. ACM, New York, NY, USA, 1695–1706. DOI: http://dx.doi.org/10.1145/2723372.2742799 event-place: Melbourne, Victoria, Australia.

[100] E. Vasilyeva, T. Heinze, M. Thiele, and W. Lehner. 2016. DebEAQ - debugging empty-answer queries on large data graphs. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 1402–1405. DOI:http://dx.doi.org/10.1109/ICDE.2016.7498355

[101] Margus Veanes, Nikolai Tillmann, and Jonathan de Halleux. 2010. Qex: Symbolic SQL Query Explorer. In *Logic for Programming, Artificial Intelligence, and Reasoning (Lecture Notes in Computer Science)*, Edmund M. Clarke and Andrei Voronkov (Eds.). Springer Berlin Heidelberg, 425–446.

[102] Jingjing Wang, Tobin Baker, Magdalena Balazinska, Daniel Halperin, Brandon Haynes, Bill Howe, Dylan Hutchison, Shrainik Jain, Ryan Maas, Parmita Mehta, and others. 2017. The Myria Big Data Management and Analytics System and Cloud Services.. In *CIDR*.

[103] Xiaolan Wang, Alexandra Meliou, and Eugene Wu. 2016. QFix: Demonstrating Error Diagnosis in Query Histories. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 2177–2180. DOI: http://dx.doi.org/10.1145/2882903.2899388 event-place: San Francisco, California, USA.

[104] Martin Wattenberg. 2001. Sketching a Graph to Query a Time-series Database. In *CHI '01 Extended Abstracts on Human Factors in Computing Systems (CHI EA '01)*. ACM, New York, NY, USA, 381–382. DOI: http://dx.doi.org/10.1145/634067.634292 event-place: Seattle, Washington.

[105] Wen-Syan Li, K. S. Candan, K. Hirata, and Y. Hara. 1997. IFQ: a visual query interface and query generator for object-based media retrieval. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*. 353–361. DOI: http://dx.doi.org/10.1109/MMCS.1997.609635

[106] Hadley Wickham. 2014. Tidy Data. *Journal of Statistical Software* 59, 10 (2014), 1–23.

[107] Krist Wongsuphasawat, John Alexis Guerra Gomez, Catherine Plaisant, Taowei David Wang, Meirav Taieb-Maimon, and Ben Shneiderman. 2011. LifeFlow: Visualizing an Overview of Event Sequences. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 1747–1756. DOI: http://dx.doi.org/10.1145/1978942.1979196 event-place: Vancouver, BC, Canada.

[108] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining Away Outliers in Aggregate Queries. *Proc. VLDB Endow.* 6, 8 (June 2013), 553–564. DOI: http://dx.doi.org/10.14778/2536354.2536356

[109] Baowen Xu, Ju Qian, Xiaofang Zhang, Zhongqiang Wu, and Lin Chen. 2005. A Brief Survey of Program Slicing. *SIGSOFT Softw. Eng. Notes* 30, 2 (March 2005), 1–36. DOI: http://dx.doi.org/10.1145/1050849.1050865

[110] Qianrui Zhang, Haoci Zhang, Thibault Sellam, and Eugene Wu. 2019. Mining Precision Interfaces From Query Logs. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. ACM, New York, NY, USA, 988–1005. DOI: http://dx.doi.org/10.1145/3299869.3319872 event-place: Amsterdam, Netherlands.