# Framework for SQL Error Message Design: A Data-Driven Approach

TONI TAIPALUS and HILKKA GRAHN, University of Jyväskylä, Finland

Software developers use a significant amount of time reading and interpreting error messages. However, error messages have often been based on either anecdotal evidence or expert opinion, disregarding novices, who arguably are the ones who benefit the most from effective error messages. Furthermore, the usability aspects of Structured Query Language (SQL) error messages have not received much scientific attention. In this mixed-methods study, we coded a total of 128 error messages from eight database management systems (DBMS), and using data from 311 participants, analysed 4,796 queries using regression analysis to find out if and how acknowledged error message qualities explain SQL syntax error fixing success rates. Additionally, we performed a conventional content analysis on 1,505 suggestions on how to improve SQL error messages, and based on the analysis, formulated a framework consisting of nine guidelines for SQL error message design. The results indicate that general error message qualities do not necessarily explain query fixing success in the context of SQL syntax errors and that even some novel NewSQL systems fail to account for basic error message design guidelines. The error message design framework, and examples of its practical applications shown in this study, are applicable in educational contexts, as well as by DBMS vendors in understanding novice perspectives in error message design.

CCS Concepts: • **Software and its engineering → Compilers**; • **Information systems → Relational database query languages**; • **Human-centered computing → Empirical studies in HCI**.

Additional Key Words and Phrases: Structured Query Language, SQL, compiler, error message, database management system, human-computer interaction, human factor, usability, readability

## 1 INTRODUCTION

A common view is that computer error messages are confusing and unhelpful, even for professionals [9, 27], and that the difficulty of reading error messages is similar to the difficulty of reading source code [5]. As developers use a significant amount of time reading and interpreting error messages [5], it seems natural that both the software industry and academia are interested in the qualities of error messages. Consequently, several studies have shown that a more usable software development environment results in increased productivity [36, 38], and that end-users benefit from increased usability in general [17]. From a pedagogical point of view, feedback – or formative assessment - during the learning process is a powerful way to improve novices' sense of achievement and motivation [e.g., 13]. Furthermore, feedback is effective especially in focusing on treating mistakes or errors as learning opportunities. This view of seeing mistakes as opportunities means that novices are shown regularly occurring mistakes and errors they make and then, utilizing feedback, are advised how to fix these mistakes [13]. Explaining mistakes and errors encourages learning [23].

Several studies agree that learning Structured Query Language (SQL) can be challenging, possibly due to the declarative nature [44, 63]. Since feedback is seen vital in learning, the database management systems (DBMS) that novices use to learn SQL should provide constructive and useful feedback in their error messages. Unfortunately, SQL compiler error messages are rarely clear or helpful from a novice viewpoint, although there are differences between DBMSs [53, 54]. In contrast to a professional, a novice often sees a particular error message for the first time, which makes the quality of the error message even more important [60]. Hence, previous literature has

Authors' address: Toni Taipalus, toni.taipalus@jyu.fi; Hilkka Grahn, hilkka.grahn@jyu.fi, University of Jyväskylä, P.O. Box 35, Jyväskylä, Finland, FI-40014.

1049-331X/2023/7-ART
https://doi.org/10.1145/3607180

made attempts to examine [7] and enhance [40] error messages of programming language compilers, or system error messages in general [47]. However, literature regarding SQL error messages is still almost entirely lacking. Additionally, research dealing with programming language compiler error messages is often based on either anecdotal evidence, or expert opinions [7], yet it has been questioned whether experts can reliably understand novice viewpoints [44]. This study attempts to fulfill the needs for both the data-driven approach and the novice perspective for SQL error message design.

In this study, we explore if and how general error message guidelines apply in the context of SQL error messages and query fixing success rates. This study also presents a framework specifically for SQL error message design. The guidelines in the framework are based on an analysis of novice feedback on the sixteen most common SQL syntax errors and corresponding syntax error messages from eight DBMSs. Additionally, we apply our framework to present examples of modified SQL error messages.

The structure of this study is as follows. In the next section, we discuss the theoretical background behind SQL query formulation, error messages in the context of general system messages as well as programming language compilers, and error message qualities. In Section 3, we describe our research goals, methods, and data collection. Sections 4 and 5 as well as Appendix A present the results of the study, i.e., results from the statistical analyses, the error message design framework, and some applications of said framework. In Section 6, we discuss the implications of our results in light of previous studies, as well as the implications for industry and education. Section 7 concludes the study.

## 2 THEORETICAL BACKGROUND

### 2.1 Query Execution

A query writer usually communicates with a relational database via a DBMS. Depending on the scenario, a query may be written, e.g., by using a DBMS interface, or by embedding SQL into a host language such as Java. When the DBMS receives the query, the query is processed and executed, and then a set of data or some form of feedback is returned. How the query is processed and executed is dependent on a particular DBMS internals [21, 22]. Because of these different internals, a query may be deemed erroneous by one DBMS, while executed by another [42]. It follows that the feedback the DBMS provides to the query writer is dependent on the DBMS.

After the query is sent, the query processing done by the DBMS is largely a black box to a novice query writer [61]. After the query writer receives an error message instead of a result table or a result table that does not meet the query writer's expectations, the process typically repeats. This is called a feedback loop between the query writer and the DBMS. Apart from the query written, the DBMS has limited means to help the query writer to formulate the query they want to formulate. Although the query writer usually cannot communicate with the DBMS in natural language, the DBMS can relate natural language messages to the query writer, bridging the gap between the two. This is typically done in one of two ways. First, if the DBMS deems the query syntactically incorrect, the DBMS outputs a natural language syntax error message [1, 11], which usually helps the query writer to pinpoint the error and even fix it. Second, if the query is deemed syntactically correct, the query writer may obtain more information on how the query was executed through a query execution plan [58]. As reading query execution plans requires both that the query is syntactically correct, as well as considerable knowledge on DBMS internals and physical database design [20, 61], they are typically used by developers for query optimization, and not by novices in undergraduate database courses [58]. For these reasons, this study focuses on syntax error messages as the means of communication between the user and the DBMS.

### 2.2 SQL Syntax Errors

Different SQL errors have received growing scientific attention, especially in computing education research [56]. Current research [11, 57] divides SQL errors regarding data retrieval into *(i)* syntax errors, which are identified by the DBMS, and which result in a syntax error message, *(ii)* semantic errors, which are typically not identified by the DBMS, and which result in incorrect data in the result table regardless of what the query is supposed to retrieve, *(iii)* logical errors, which are not identified by the DBMS, and which result in incorrect data in the result table when the intent of the query is considered, and *(iv)* complications, which may be identified by the DBMS, and which do not result in incorrect data in the result table, but unnecessarily complicate the query.

Syntax errors have been shown as the most common errors, especially in novice query formulation [1, 55]. Depending on the study, common syntax errors are caused by e.g., references to undefined tables and columns [49], problems with grouping [43, 45], data type mismatches [1], misspellings [62], omitting mandatory clauses [49], as well as illegal aggregate function placement and duplicate clauses [57]. The identification of the most common SQL syntax errors is challenging due to the fact that, with a few exceptions, studies categorize SQL syntax errors depending on the DBMS used, making results incomparable to studies with a different DBMS. Furthermore, it has been shown that, rather intuitively, different SQL concepts such as grouping, joins or ordering invite different types of syntax errors [55], making, e.g., syntax errors common for queries involving a table join uncommon for queries involving grouping and a single table.

Despite their commonness over other types of errors, and probably because syntax errors are caught by the DBMS, syntax errors have been shown to be easier to fix than other types of errors [2, 55]. This is arguably intuitive, as a syntax error halts the execution of a query and prevents the query writer from receiving a result table, while e.g., a logical error does not. A query with a logical error may return a result table with even seemingly correct data, yet this data do not adhere to the query writer's intent. Causes behind query formulation errors in general have been explained by both human factors such as cognitive load theory [49], different misconceptions regarding the language or generalizations [28], self-induced complexity [30], procedural fixedness [51], and simple sloppiness [29, 49], as well as environmental considerations such as database structure complexity [50] and database normal form [10].

## 2.3 Error Message Qualities

As syntax errors are typically the only type of errors identified by a DBMS, it follows that a natural way to enhance the communication between the query writer and the DBMS is with more effective syntax error messages. Despite the scientific attention given to SQL errors and programming language error messages, SQL error messages have not received much scientific attention. For this reason, we discuss programming language error message research here. It is worth noting that the declarative nature, as well as the purpose of SQL, is different from programming languages such as C# or Python, which poses challenges to the comparison of the results of programming language compiler error messages and SQL error messages.

Previous studies have shown that programming language compiler error messages are often considered confusing and unhelpful [7, 8], and that users are likely to feel inadequate and anxious when encountering error messages [48]. The role of error messages in the feedback loop is even more crucial when the user is a novice, yet as the quality of error messages affects the overall user experience, better error messages benefit professionals as well [25].

For the reasons above, previous literature has formed guidelines for designing error messages [7, 47, 60]. These guidelines have focused on either general system error messages or programming language compiler error messages, but not on SQL or query language error messages. However, since the domains are related, and previous SQL error message guidelines are unavailable in scientific literature, it is reasonable to inspect those related guidelines here.

The renowned guidelines for designing computer system messages by Shneiderman in 1982 [47] consists of five suggestions that system messages should meet: *(i)* be brief, *(ii)* be positive, *(iii)* be constructive, *(iv)* be specific, and *(v)* be comprehensible. Later, other authors have formed guidelines for programming language compiler error messages as well. For instance, Traver [60] suggested eight programming language error message qualities, reflected against Nielsen's heuristics [34, 35] and his own experience as a programmer and an educator, for error message design. In addition, Becker et al. [7] published a comprehensive review of papers on programming language error messages, categorizing the studies to historical, anecdotal, or empirical research, and presenting a compilation of ten guidelines for programming language error message design.

These three guidelines, understandably, share similarities. Shneiderman's first suggestion, *be brief*, is discussed by both Traver [60] as *clarity and brevity* as well as Becker et al. [7] as *reduce cognitive load*. This guideline is, depending on the source, effectively realised by aesthetic and minimalist design [34, 35], meaning that error messages should not be cryptic, long, or hard to interpret [60], and also by enhancing simplicity in the error messages [7]. It also has many other meanings in the literature, such as removing jargon, using complete sentences, and using simple vocabulary [16].

Second, *be positive*, more broadly expressed as *proper phrasing* [60] is related to Nielsen's heuristic of *match between system and the real world*, which refers to the positive tone of the error message, guidance to help fix the error, use of simple language, as well as using similar words in the system when referring to similar concepts in the real world. In the compiled guidelines [7], the counterpart is phrased as *use a positive tone*, referring to avoiding negative words, such as illegal, incorrect, or invalid.

Third, Shneiderman's [47] *be constructive* is included under *proper phrasing* as *constructive guidance* in Traver's guidelines [60], and broadly interpreted counterparts in Becker et al. [7] are called *provide scaffolding* and *show solutions or hints*. Effectively, these guidelines suggest that the error message should provide explanations to the user on why they received the error message and giving support on how to proceed, rather than simply stating that there is an error. The error message should also suggest solutions for how the error can be fixed.

Fourth, *be specific* is formulated as *specificity* by Traver [60], and is related to two of Nielsen's [34, 35] heuristics, *recognition rather than recall*, and *help user to recognize diagnose, and recover from errors*. Specifically, the guideline means that error messages should not be too general, since a general error message makes locating the erroneous position difficult. Traver's [60] *locality* is related to this guideline, suggesting that the error message should indicate the true origin of the error. Related to this guideline, Becker et al. [7] suggest that the error message should *provide context*, meaning that there should be information about the programming code relevant to the error which helps understand and address the error.

Fifth, the guideline *be comprehensible* can be mapped to *clarity and brevity* [60], and *increase readability* [7]. This is perhaps one of the more subjective guidelines, as comprehensibility is closely dependent on the user reading the message. In addition to Shneiderman's [47] five guidelines and their broadly interpreted counterparts, subsequent studies have suggested additional or more precise guidelines for programming language error messages. For example, it has been suggested that error messages should be *context-insensitive*, meaning that the same error results in the same error message [60], and that the error message conveys a logical train of thought to the user of why the error occurred by *using logical argumentation* [7] and nonanthropomorphic messages [60]. That is, the message should not use language which implies that the syntax was checked by a human-like actor.

Furthermore, Traver [60] suggests that the error message is divided into three levels: a short message first, then — if the user needs — a brief explanation or examples regarding the error message, and finally, possible corrective actions. Additionally, the environment in general should use colors and fonts to notify the user of errors as early in the writing process as possible [7, 60]. Finally, the message should show the user examples of similar errors to improve the understanding of why the error occurred [7]. As can be seen, all these above guidelines share similarities with Shneiderman's [47] renowned guidelines regarding computer system messages, as well as with each other.

Table 1. Sixteen most common syntax errors [57] and corresponding tests [54]

| Test | Syntax error name | Test | Syntax error name |
|---|---|---|---|
| T01 | ambiguous column | T09 | failure to specify column name twice |
| T02 | omitting quotes around character data | T10 | using an aggregate function outside `SELECT` or `HAVING` |
| T03 | IS where not applicable | T11 | grouping error: extraneous grouping column |
| T04 | confusing the syntax of keywords | T12 | nonstandard operators |
| T05 | confusing the logic of keywords | T13 | using `WHERE` twice |
| T06 | too many columns in subquery | T14 | nonstandard keywords or standard keywords in wrong context |
| T07 | undefined column | T15 | synonyms |
| T08 | misspellings | T16 | curly, square or unmatched brackets |

## 3 RESEARCH SETTING

### 3.1 Research Scope and Goals

Based on the research gaps identified in the previous sections, we formulated four research goals (RG), and collected both quantitative and qualitative data to reach these goals. We chose to limit the scope of our study to the sixteen most common syntax errors reported in a previous study [57] (cf. Table 1).

We chose four "traditional" RDBMS as well as four NewSQL [39] RDBMS syntax error messages. NewSQL systems are RDBMSs built from the ground up in the 2010s to account for the innovations and technical development introduced by NoSQL systems, while also catering to the needs of RDBMS users by, e.g., using SQL and a strong consistency model [39]. The reason for including these eight DBMSs was that we wanted to inspect the state of error messages in long-running DBMSs as well as DBMSs developed in the 2010s, while limiting the study to DBMSs that can execute SQL to the degree the test suite requires. The test suite and the selected DBMSs are detailed in the next section.

As discussed in Section 2.3, scientific literature has made many recommendations and guidelines on how user interfaces and system messages in general, or programming language error messages in particular should be formulated. We chose to use the guidelines for system messages proposed by Shneiderman [47], because they are general, as opposed to guidelines proposed for programming language error messages [e.g., 7, 60], which are particular for a different purpose. Furthermore, with our choice of older guidelines, we wished to highlight how even some modern DBMSs disregard guidelines proposed in the early 1980s.

***RG1****: Find out if and how previously identified system message qualities (i.e., whether the message is brief, positive, constructive, specific, and comprehensible) affect SQL query fixing success in the error messages of the eight selected DBMSs.* The results are presented in Section 4.

***RG2****: Formulate an SQL error message design framework consisting of guidelines derived from the collected data.* The framework is presented in Section 5.1.

***RG3****: Investigate how query writers would improve syntax error messages of eight relational DBMSs both in general, and specific to each of the most common syntax errors.* The results are introduced in Section 5.2 and presented in detail in Appendix A.

***RG4****: Based on the formulated framework, propose examples of error messages pertaining to the sixteen most common syntax errors.* The examples are presented in Appendix A.

### 3.2 Data Collection

We created our data collection form around a previously reported syntax error test suite [54], which was in turn based on the sixteen most common SQL syntax errors detailed in Table 1. This test suite provided us with a concrete database structure, sixteen SQL data demands, and the corresponding erroneous queries. We then ran the erroneous queries on eight DBMSs (MySQL 8.0.12 with InnoDB storage engine, Oracle Database 19c Enterprise Edition 19.5.0.0, PostgreSQL 12.1, SQL Server 2019 Developer, CockroachDB 19.2.2, SingleStore 7.0.10 with InnoDB storage engine, NuoDB 4.0.4-2, and VoltDB Community 9.2.2) to capture corresponding syntax error messages. These versions were the most recent, stable available versions at the time of data collection. Based on the error messages, we created eight data collection forms — one for each DBMS — consisting of the database schema, a data demand, a corresponding erroneous SQL query, and the corresponding syntax error message, and two free text input fields in which the participant was asked to write a fixed SQL query, and with their own words describe how to improve the error message. All eight data collection forms were the same with the exception of the syntax error messages, and each data collection form consisted of sixteen pages, one for each syntax error detailed in Table 1.

We recruited study participants from a database course given in the authors' university. Prior to participation, the participants were given lectures and mandatory exercises on topics recommended in AIS/ACM curriculum guidelines [59] for an undergraduate database course. Each topic (separated by semicolons) included approximately 4 hours of lectures and 7 hours of exercises: conceptual modeling; the relational model; relational calculus; data manipulation language using SQLite with simple operators, inner and outer joins, and ordering; data manipulation language using SQLite with aggregate functions, grouping, and correlated and uncorrelated subqueries; and data definition language using SQLite. After these topics, this study was introduced. After this study, the course continued with topics such as transaction management and normalization theory. When a participant decided to participate in the study, they were randomly presented with one of the eight data collection forms, e.g., a participant assigned to a data collection form with syntax errors from SQL Server filled out the input fields based on their perceptions of SQL Server error messages. For each participant, the sixteen pages were shown in random order. Participation was voluntary, and the participants were shown a data privacy statement prior to their decision on whether to participate. Out of the 363 respondents, 311 (86%) chose to participate.

## 3.3 Data Preparation

We coded the 128 syntax error messages (16 tests × 8 DBMSs) according to the five system message guidelines described by Shneiderman [47], according to which a system message should be *brief*, *positive*, *constructive*, *specific*, and *comprehensible*. Because the message qualities given by Shneiderman are general regarding the scope of our study (*system* messages, as opposed to *SQL error* messages), and because Shneiderman gives no example on, e.g., what is a brief message, we defined a rubric (Table 2) according to which we coded the SQL error messages. We coded the same subset (20%) of the error messages individually using the rubric, and compared our results. All the codings were similar. The first author then coded the rest of the error messages. The original error messages of the eight DBMSs and respective coding are reported in Appendix B.

After data collection, we executed the 4,976 SQL queries (311 participants × 16 tests) the participants had attempted to fix on the corresponding DBMS, e.g., if a participant had been shown VoltDB error messages, and therefore attempted to fix erroneous queries based on VoltDB error messages, we executed their fixed queries on VoltDB. Additionally, the syntactically correct queries were manually checked to determine if they were also logical equivalents to the corresponding data demand. MySQL tolerated the syntax errors in tests T05 and T09, and SingleStore tolerated the syntax errors in tests T09 and T11. In these data collection forms, the error messages were made up by us, and the corresponding participant answers were omitted from the analyses. This left a total of 124 error messages for coding, and 4,796 SQL queries for statistical analyses.

Table 2. The rubric created for error message coding; the numerical codes were converted to empty (white), half, and full circles (black) for readability; the cutoffs for the brevity of the messages were determined based on tertiles of the length of the messages studied

| Quality | Code | Description |
|---|---|---|
| brief | ○ | More than 171 characters, not counting the possibly replicated query or parts thereof, or parts replicated from environmental variables. |
| | ◐ | Between 100 and 171 characters, not counting the possibly replicated query or parts thereof, or parts replicated from environmental variables. |
| | ● | 99 characters or less, not counting the possibly replicated query or parts thereof, or parts replicated from environmental variables. |
| positive | ○ | Tone is negative, aggressive, dramatic or discouraging, using words such as illegal, invalid, error, or incorrect. |
| | ● | Tone is positive or neutral, the message contains no negative words. |
| constructive | ○ | Offers no advice. |
| | ◐ | Offers advice on what causes the error. |
| | ● | Offers constructive advice on how to fix the error. General advice such as “refer to manual” or “see help on SELECT” was not considered constructive advice. |
| specific | ○ | Provides no error position. |
| | ◐ | Replicates a relatively large part of the erroneous query or replicates the query in full, or otherwise provides an approximate error position. |
| | ● | Specifically shows the position of the error. |
| comprehensible | ○ | The error message is almost incomprehensible or generally unhelpful. |
| | ◐ | The error message contains unnecessary jargon. |
| | ● | Not including error codes, SQL keywords, or common relational database terms, the message reads closely to plain English. |

## 3.4 Data Analyses

In order to analyze the effects of error message qualities, we constructed binomial logistic regression models for each of the sixteen tests. The data contained 311 answers per test, with the limitations concerning MySQL and SingleStore described in Section 3.3. The independent variables in the model were the error message qualities, i.e., whether the error message was brief, positive, constructive, specific, and comprehensible. The dependent variable in the model was query fixing success, which was binomial, 0 meaning that the query was not fixed, and 1 meaning that the participant succeeded in fixing the query.

Participants also suggested improvements for the error messages using a free text input field. Some participants did not suggest improvements for all error messages, while others suggested several improvements for each of the sixteen error messages they were shown. In total, the participants gave 1,505 answers, which we analysed using conventional content analysis [24]. Effectively, the method groups similar content, or *themes*, into groups that are derived from the data rather than theory or prior literature. The goal of the method is to generalize or reduce data to a form that is easier to interpret. The participant answers were interpreted as is, resulting in both general themes that span across most error messages, and themes that are specific to a given error message.

We first collectively analyzed approximately 5% of the 1,505 answers, and derived example codes from the data, such as *error message should show line number* and *error message should suggest a fix*. We then chose a portion of 10% which we both then coded individually. After the individual coding, we compared our results and deemed that all our codings were similar. These agreements most likely stemmed from the fact that all the answers were relatively short, typically containing one or two sentences. The first author then proceeded to code the rest of the data. After this step, we convened to discuss whether the more specific categories should be merged, and categorized the most frequent codings into a higher-level framework for SQL error message design. That is, our results show three levels of abstraction. On the lowest level, we show suggestions for improvements *per error message for the 16 tests*. On the middle level, we show suggested improvements regarding *all error messages*. On the highest level, we categorize these suggested improvements regarding all error messages into *themes*.

## 4 FACTORS AFFECTING ERROR MESSAGE EFFECTIVENESS

Only in three tests, the binomial logistic regression model was statistically significant (alpha level .05). In test T01 ($\chi^2(3) = 13.339$, $p = .004$), the model explained 8.0% (Nagelkerke $R^2$) of the variance of fixing the query and correctly classified 87.8% of cases. Of the five predictor variables (i.e., error message qualities), only one was statistically significant: specificity of the error message ($p = .012$). Error messages being specific had 2.03 higher odds of being fixed successfully.

In test T11 ($\chi^2(5) = 14.003$, $p = .016$), the model explained 7.3% (Nagelkerke $R^2$) of the variance of fixing the query and correctly classified 82.6% of cases. Of the five predictor variables, only one was statistically significant: error message being brief ($p = .020$). The errors with long error messages had 1.79 higher odds of being fixed successfully.

In test T16 ($\chi^2(4) = 10.243$, $p = .037$), the model explained 6.9% (Nagelkerke $R^2$) of the variance of fixing the query and correctly classified 90.4% of cases. Of the five predictor variables, only one was statistically significant: error message being positive ($p = .010$). The errors with positive error messages had 1.95 lower odds of being fixed successfully. In the rest of the tests, the binomial logistic regression models did not identify the result as statistically significant (significance levels ranging from $p = .101$ to $p = .969$).

## 5 ERROR MESSAGE DESIGN FRAMEWORK AND MODIFIED ERROR MESSAGES

### 5.1 Error Message Design Framework

Using conventional content analysis, and regardless of the test, we identified nine recurring suggestions for error message improvements in the data. Five suggested, general improvements received more than one hundred mentions in the 1,505 answers: *specify the line number of the erroneous part* (191 mentions, approximately 13%), *suggest how to fix the error* (181, 12%), *remove unnecessary information* (141, 9%), *explain what causes the error and why* (141, 9%), and *specify the error position* (111, 7%). It is worth noting that these mentions are only general mentions, not counting e.g., more specific suggestions on how to fix an error (e.g., "*suggest single quotes around character strings*" or "*suggest replacing* `IS` *with =*"). Among others, these abstracted suggestions comprise the SQL error message design framework presented in Table 3. The following list shows some selected quotations from the data.

- [on CockroachDB, T04] "*The error message tries to say that the* `LIKE` *operator does not understand lists, but it says this in a very difficult way. A typical those-who-know-just-know type of message. Isn't the point of error messages to help us, rather than further separate us from professional users?*"
- [on MySQL, T08] "*It is unnecessary to state multiple times, or even once, that the query is erroneous. If it weren't erroneous, I would receive results instead of a message.*"
- [on MySQL, T08] "*It is astounding how the message cannot pinpoint such an obvious typo.*"
- [on VoltDB, T09] "*Simply providing a line number would have been more helpful than this long error message.*"
- [on NuoDB, T10] "*The error was easy to locate, but fixing it just requires skill which I do not have, and the error message is not helpful in this regard.*"
- [on NuoDB, T12] "*The error message was so comprehensive it almost fixed the error for me.*"
- [on Oracle Database, T13] "*Perhaps the message is technically the correct way to describe the error, but from a human perspective, this seems incomprehensible.*"

Table 3. The SQL error message design framework consists of three high-level themes consisting of a total of nine guidelines

| | |
|---|---|
| Where | **Provide line number**: as accurately as possible, show the user on which line the error is.<br>**Specify the error position**: point to the position of the error on the erroneous line. |
| What | **Explain what causes the error**: describe what is missing, extraneous, ill-placed, or incorrect.<br>**Explain why the error occurs**: describe what principle is violated.<br>**Place the most important information first**: let the user choose whether to read further. |
| How | **Provide suggestions on how to fix the error**: use reserved wording, as the intent of the user is unknown.<br>**Provide working examples of similar query concepts**: show how a query concept is used as a part of a query. |
| **Remove unnecessary elements**: remove error codes, host names etc., or move them to the end of the message. | |
| **Use plain English**: use well-understood terms, or explain complex terms using simple natural language words. | |

First, two guidelines in the framework are closely related to *where* the error occurs. The data suggest that providing the line number (and providing it correctly) was one of the most frequent suggestions for improvements. Additionally, and while line breaks are not something SQL enforces, the data suggest that a line number is not always enough to specify the error position accurately. One solution to such suggestion (also suggested by the data) is to replicate the erroneous query or parts thereof, and specify the erroneous position using a free-standing circumflex like some DBMSs already do (e.g., PostgreSQL in Fig. 17).

Second, three guidelines are closely associated with *what* elements should form the body of the error message. The data suggest that novices want to know what causes the compiler to halt the interpretation of the query, and providing the position of the error is not informative enough. Additionally, a frequent suggestion for improvements was that the error message should explain why the error occurs. For example, placing a semicolon in the middle of a query should result in an error message showing the *line number* and exact *position* of the error, stating that there is a semicolon in the middle of a query (the *what*), and explaining that a semicolon is used to terminate an SQL statement, and that placing a semicolon in the middle of a query is against this principle (the *why*). Finally, the error message should be structured in a way that the most important parts for the user are placed first. This arguably helps professional users, who are arguably not as interested in hints or SQL examples as novices.

Third, two guidelines concern *how* the user could or should proceed in fixing the erroneous query. The error message should suggest how the user could fix the error. As the DBMS does not understand the intent of the user, the wording in these suggestions should be reserved, as a suggested fix may point the user towards a fix that is not the correct fix for the particular intent. Next, the error message should show examples of correctly using the SQL concepts associated with the error, e.g., an erroneous expression should result in an error message showing examples of correctly written expressions.

Finally, general themes in the data were the removal of unnecessary elements and the use of plain English (or whatever the language is if the error messages are provided in some other language). For example, error codes, host names, and the statement that the query is erroneous were seen as unnecessary information. The use of plain English was also frequently suggested, as the environment is complex even without seemingly unnecessarily convoluted sentences.

## 5.2 Suggested Improvements and Modifications

We present the results from the conventional content analysis for each of the sixteen tests in Appendix A due to their length. Each of these figures consists of subfigures illustrating *(a)* the erroneous query with the erroneous part highlighted for readability, *(b)* respective fixed query, *(c)* suggested improvements derived with conventional content analysis, with the number of occurrences, and *(d)* an example of how an error message could be reformulated based on the proposed error message design framework. It is worth noting that suggestions with fewer than two mentions are not reported, as we deemed that one mention did not constitute a category. Despite the highlights in Appendix A, the erroneous parts were not highlighted for the study participants, and that the typographic details concerning, e.g., line breaks may differ from the tests proper due to horizontal space limitations here. The example queries shown in the modified error messages are intended as static examples, i.e., we do not intend the DBMS to generate dynamic examples based on the underlying database schema, although with recent advances in large language models, this might be a feature to consider. All corresponding error messages are presented in Appendix B.

## 6 DISCUSSION

### 6.1 General Discussion

This study pursued to examine what qualities of error messages explain the rate the participants succeeded to correct the query they were shown and how would they improve the error messages. Utilizing the participants' suggestions, we modified the error messages used in this study and compiled a set of guidelines for error message design (Table 3).

As can be observed, the queries with common syntax errors are simple in the test suite. Additionally, as the test suite is based on previously identified common syntax errors, the empirical observations underneath also show that novices commit simple syntax errors [57]. Despite this, the error messages do not reflect the simple nature of the errors. Based on the error messages listed in Appendix B, it seems justified to argue that many error messages fail to identify the nature of the error correctly, identify the error position incorrectly, or both. This arguably highlights the rather unfortunate state of error messages in many modern DBMSs, instead of begging the question of why the test suite only considers simple syntax errors. Despite what a reader thinks about the error message guidelines presented in this study or of those presented previously in scientific literature, we argue that all the DBMSs subject to this study have error messages that contain at least some elements that seem unintuitive in facilitating query fixing. Table 4 lists characteristics typical to each DBMS. The table arguably shows that many (if not all) of the design guidelines presented in this study have been implemented in at least one of the DBMSs studied.

Research Goal 1, presented in Section 3.1, was concerned with previously identified system message qualities and how they affect SQL query fixing success. This was analyzed with binomial logistic regression. The results of the regression analyses presented in Section 4, with three exceptions, failed to reject the null hypothesis. This may indicate, at least with the data available, that general error message qualities do not explain SQL error fixing, i.e., the general guidelines fail to particularize. In addition, it should be noted that all the percentages of how much the three statistically significant models explained the success rate were very low. The results from the qualitative analysis, however, suggest that the participants value the error message qualities proposed by Shneiderman [47], with the exception that error messages should be positive. In a sense, the results from the quantitative analyses are not in line with the results from the qualitative analyses. That is, the regression analyses suggest that general system error message qualities do not affect query fixing, but nevertheless the results from the content analyses rather uniformly suggest that if the error message qualities tested in the regression model were not present in the error messages, the participants suggested adding these qualities.

Research Goal 2 was to formulate an SQL error message design framework derived from the data. Table 5 compares the guidelines presented in this study to those presented by Shneiderman [47], Traver [60], and Becker et al. [7]. The table shows that most of our guidelines map to most of the guidelines presented in previous studies, indicating that our study participants suggest improvements for error messages presented in previous studies unknown to them. The only clear omission in our guidelines is that the error message should be positive. According to our coding of the eight DBMS error messages, all DBMSs had at least one error message that was

Table 4. Typical characteristics of the SQL error messages of eight DBMSs; it is worth noting that these are typical characteristics based only on the sixteen types of errors studied

| DBMS | Characteristics of SQL error messages |
|---|---|
| MySQL (with InnoDB) | Sometimes contain error codes at the beginning of the message; both brief and wordy messages; general suggestions to check the manual; line numbers sometimes present; sometimes replicates a part of the query; non-uniform error messages. |
| Oracle Database | Error codes at the beginning of the message; brief messages; no line numbers; general messages. |
| PostgreSQL | No error codes; line numbers; specific error position is indicated by a free-standing circumflex; sometimes provides hints; replicates the erroneous line; complete sentences. |
| SQL Server | Error codes and additional environmental variable information at the beginning of the message; line numbers; a single message may identify multiple errors; replicates the erroneous position. |
| CockroachDB | No error codes; does not explicitly state that there is an error; both brief and wordy messages; no line numbers; sometimes the specific error position is indicated by a free-standing circumflex; sometimes provides general hints; sometimes replicates the query up to the position of the error. |
| SingleStore (with InnoDB) | Error codes at the beginning of the message; general suggestions to check the manual; line numbers sometimes present; sometimes replicates a part of the query. |
| NuoDB | Error codes at the beginning of the message; sometimes replicates parts of the query; sometimes the specific error position is indicated by a free-standing circumflex; sometimes explains what was expected at the erroneous position. |
| VoltDB | No error codes; no line numbers; replicates the whole query; sometimes provides hints; sometimes explains what was expected at the erroneous position. |

not positive, meaning that all 311 participants were exposed to at least one non-positive error message. Our data contained only three mentions (all from a single participant, approximately 0.3% of all participants) that the error message could be rephrased without the use of dramatic words. This observation may be biased due to the fact that the participants were recruited from a single university. Becker et al. [7] summarize that in the context of programming language error messages, the effects of a positive tone have been empirically tested by merely two studies, while 14 studies on the subject are of historical or anecdotal nature. A closer inspection of the two studies reveals that in the first study [31], 29% of the 77 participants observed that the word *illegal* may intimidate the user. In the second study [32], six of the thirteen participants claimed that the error messages under observation were *friendly*, yet it was not further discussed whether *friendly* was considered helpful. Both our quantitative and qualitative results contest the recommendation that error messages should be positive. In fact, in test T16, a positive tone even reduced the odds of successful query fixing. Our interpretation of this result is that the positive error messages in test T16 simply had some other quality which hindered query fixing, rather than positive tone being detrimental to query fixing.

Research Goal 3 was to investigate how the error messages should be improved according to the participants, and Research Goal 4 was to propose examples of modified error messages based on the suggested improvements. The modified error messages are presented in Appendix A. We listed *specify the error position* as one of the guidelines. It seems both crucial and needless to expand that the error message must provide the error position *correctly*, something that both programming language error messages [60] as well as SQL error messages (e.g., VoltDB in Fig. 24 in Appendix B) sometimes fail to do. In cases when reliably pinpointing the error is not possible, the error message should provide a *near* position like some DBMSs already do, although even these positions are not always accurate, or replicate a part of the query which does not contain the error (e.g., SingleStore in Fig. 24 in Appendix B). As demonstrated with a quotation in Section 5.1, at least one participant raised a concern that simply providing the line number of the erroneous part would be more informative than the error message provided. Although the consensus view seems to be that feedback with examples is more useful than binary feedback (correct/incorrect) [14], even this might not hold true when the error message provides unnecessary

Table 5. The SQL error message design framework guidelines and their broadly interpreted counterparts in previous literature

| **SQL error message design guidelines (this study)** | General system message guidelines described by Shneiderman [47] | Programming language error message guidelines presented by Traver [60] | Programming language error message guidelines compiled by Becker et al. [7] |
|---|---|---|---|
| Provide line number | Specific | Specificity; Locality | Provide context |
| Specify the error position | Specific | Specificity; Locality | Provide context |
| Explain what causes the error | Constructive | *(not present)* | Provide scaffolding |
| Explain why the error occurs | *(not present)* | *(not present)* | Allow dynamic interaction; Provide scaffolding; Use logical argumentation |
| Place the most important information first | *(not present)* | Extensible help | Reduce cognitive load |
| Provide suggestions on how to fix the error | Constructive | Constructive guidance | Show solutions or hints |
| Provide working examples of similar query concepts | *(not present)* | Extensible help | Show examples |
| Remove unnecessary elements | Brief | Clarity and brevity | Reduce cognitive load |
| Use plain English | Comprehensible | Clarity and brevity; Programmer language | Increase readability |
| *(not present)* | Positive | Positive tone | Use a positive tone |
| *(not present)* | *(not present)* | *(not present)* | Provide errors at the right time |
| *(not present)* | *(not present)* | Context-insensitivity; Nonanthropomorphism; Consistency; Visual design | *(not present)* |

or incorrect feedback. For example, the error message of VoltDB shown in Fig. 22 in Appendix B replicates the erroneous query in its entirety, yet does not provide the error position, demonstrating an error message that is neither brief nor specific. Also from the perspective of specificity and comprehensiveness, the error messages of VoltDB and SQL Server in Fig. 22 in Appendix B are different. Considering that the error in the query in Fig. 22(a) is that the subquery returns too many columns for the `IN` operator which is in this case only expecting values from one column, the cause of the error in VoltDB in Fig. 22 in Appendix B is ambiguous, stating "row column count mismatch", which seems to imply that that the error is somehow related to rows. The error can be fixed by making sure that there is the same number of columns in the upper-level query's expression concerning `IN`, as in the subquery's `SELECT` clause. The error message of SQL Server in Fig. 22 in Appendix B, however, uses a complete sentence and conveys the cause of the error more accurately.

Regarding suggestions on how to fix errors, Marceau et al. [26] suggest that novices can follow suggestions on how to fix an error without understanding what causes the error and whether the suggestion is even the right fix. This may be problematic from an educational perspective, as the goal is arguably not to fix an error per se but to learn how to write queries. Although making mistakes is part of any learning process, simply committing errors for the sake of receiving hints and suggestions on how to correctly write a query seems counterproductive to learning. We believe that by explaining what causes the error and why the error occurs, the error message can provide a more deep-rooted understanding to a novice, as opposed to merely providing a suggestion on how to fix the error.

Both Shneiderman [47] and Traver [60] have suggested that error messages should be brief, and the need for brevity is usually argued with the need to reduce cognitive load [46, 51]. Four of our guidelines, *provide line number*, *specify the error position*, *place the most important information first*, and *remove unnecessary elements* can be viewed as means to reduce cognitive load, yet it is worth noting that our data suggest very few observations on error message brevity per se. Based on the data, it seems reasonable to argue that brevity in itself is not a desirable goal for an SQL error message, as it can be seen to contradict guidelines such as using plain English with complete sentences, or with the guidelines of providing hints and suggestions. Additionally, some errors arguably cannot be described with both clarity and brevity, as the situations in which the errors occur, or why the error occurs may be complex. Therefore, we present that cognitive load should be reduced with other means such as the removal of unnecessary elements and ordering of information, rather than with brief error messages.

Previous results from scientific efforts toward more effective programming language error messages have been inconclusive, or not implemented by the industry, which somewhat diminishes the framework presented in this study. For example, some studies have tested enhanced error messages with novices, yet concluded that there are no positive effects [15, 40]. In contrast, at least one study has shown that the utilization of enhanced error messages results in fewer errors, and fewer repetitions of a similar error [6]. Nevertheless, it has been criticized that over several decades, error message guidelines revolve around similar themes, and despite the rise of new programming languages, the same problems persist [7]. In our opinion, there have also been exceptions, such as the error messages presented in the programming language Rust. From a critical perspective, this study is also yet another one proposing guidelines for error message design, albeit in the novel context of SQL. Despite the criticism presented about programming language error messages, Table 5 shows that there are fundamental differences between the error messages in different DBMSs and that the effects of these differences have also been shown to affect the perceived usefulness for error finding and fixing [54]. Therefore, it seems reasonable to argue that enhancing SQL error messages is a desirable goal for both industry and education, even though our quantitative analysis does not support the view that some error message qualities affect query fixing success rates. In other words, the effects of enhanced error messages may be explained with other variables besides success, not captured in our data or regression model.

Additionally, online learning environments have been proposed for SQL learning for decades [12], and several learning environments that provide enhanced error messages have been studied in programming education [8, 40]. Implementing enhanced error messages into learning environments seems like a natural and relatively fast way of helping novices, as well as acquiring empirical findings on the effectiveness of said error messages. The potential problems with such learning environments are the maintenance overhead and the fact that many of such environments are closed or proprietary, and to our knowledge not widely utilized in industry. Furthermore, in terms of SQL, which can be implemented by one of several different DBMSs with different internals, either the maintenance of the learning environment is even more laborious than that of programming language learning environments, or the SQL learning environment only supports a small subset of DBMSs. One workaround to maintenance could be to check the query syntax on the learning environment's side, which in turn risks the situation where the learning environment evaluates the query syntax error-free, yet the underlying DBMS detects a syntax error (or vice versa). Given these considerations, we deem it beneficial for education that the DBMS vendors undertake the task of enhancing SQL error messages to consider design guidelines. That being said, the task is not as straightforward as changing the character strings provided by the compiler [4], and arguably requires refactoring of the query parsing process with, e.g., reclassification [64] or perhaps by utilizing the work done on automatic error correction in SQL [37], or with large language models. However, it is unclear how DBMSs identify error messages, how different these implementations are between different DBMSs [52], and how technically difficult it is to implement modified error messages. Finally, industry may be (rightfully) concerned about investing in enhancing error messages which have not been scientifically shown to affect the general user experience.

## 6.2 Limitations and Threats to Validity

The limitations concerning the scope of this study are that the error message design framework is solely based on data retrieval, and not on other types of SQL statements such as data insertion or updating and that we only considered the sixteen most common syntax errors. The main reason behind limiting the scope of this study is the extent of previous studies. In terms of different SQL statements, data retrieval is the most well-studied in human-centered contexts [56], and provided us with a limited yet scientifically justified starting point with reports on which errors are the most common [57], and how these common errors can be tested [54]. Another limitation is that many of the syntax errors can be interpreted in multiple ways (e.g., T08 misspellings), yet we only tested each syntax error with one query. Furthermore, all the syntax errors were tested with relatively simple queries. Arguably, more complex queries emphasize the error message qualities even more, e.g., specifying the error position in the error message is more valuable in a query spanning 50 lines when compared to a query spanning 5 lines. Finally, we only tested the syntax errors using novice participants. This could be seen as a limitation affecting the generalizability of the results, yet given that many previously reported guidelines have been based on expert opinion [47, 60], and that this expert opinion has been critiqued [7], we believe the use of novices is a justified approach towards filling an identified research gap, rather than a limitation. The use of appropriate study participants has been argued for in detail in several studies [18, 19].

Regarding the regression analyses, it is possible that there may be a hidden factor or factors (i.e., predictors) not present in the data that affect query fixing success. Furthermore, the dependent variable (i.e., query fixing success) may not be a fitting metric for error message effectiveness. In the context of programming languages, it has been speculated that time taken to fix an error might be such a metric [3, 41], but time was not measured in this study due to the shortcomings of our data collection instrument. In hindsight, measuring time would have been informative and should be taken into account in further studies if the effects of the modified error messages are studied. Finally, it is possible that our coding of the error messages (Appendix B) captures Shneiderman's [47] error message qualities incorrectly. We have explained the general nature of said guidelines in Section 3.3, and based our coding on a rubric reported in the same section to mitigate this threat and make the coding more transparent.

Another threat to validity is the unnatural environment in which the participants fixed their queries. As explained in Section 2.1, the user typically engages in a feedback loop with the compiler. In this study, however, the participants fixed queries written by someone other than themselves (i.e., us), and received no feedback on whether their fixes were at least syntactically correct. As we wanted to base the error message design framework on previously identified common syntax errors, designing the research setting in another way would have introduced other threats to validity. Nevertheless, the results should be interpreted while taking the environment into account.

## 6.3 Future Directions

Although our framework was constructed based on empirical findings, this study provides no empirical evidence if these modified error messages actually facilitate, e.g., error fixing success rate, the time required to fix errors or user experience in general. For this reason, we have refrained from calling the new error message examples *enhanced* or *improved*. An intuitive topic for future research is to test the effectiveness of these messages using several metrics and iterate the messages based on empirical findings.

A potential — and to our knowledge little studied — topic is the suggestions given by compilers. In our data, several participants criticized the error messages for giving misleading suggestions, or identifying the erroneous position incorrectly. Although we did not systematically examine such situations in our data, it seems justifiable to speculate whether certain error messages are even detrimental to error fixing. Future research could categorize the queries the participants had fixed as, e.g., *more incorrect*, *still incorrect*, *incorrect in a different way*, *more correct*, and *correct*, and examine whether the error message plays a part in the evolution of the originally erroneous query.

Finally, both Becker et al. [7] and Traver [60] briefly discuss the interaction between the human user and the compiler through a more interactive user interface than plain text. For example, such a simple modification as hyperlinks in the error message pointing to more extensive documentation is something our participants also suggested. External online documentation would also make fixes and updates to error messages more effortless, without requiring updating the DBMS. Additionally, the error messages may be provided in a form other than textual, if the environment allows [33]. While more rich feedback may arguably present problems in development contexts (as opposed to learning contexts) when, e.g., the DBMS error message is replicated in a plain text error stack, from an educational and human-computer interaction perspective, such richer error messages are an interesting future topic.

## 7 CONCLUSION

In this study, we set out to investigate if and how general system error message qualities explain SQL syntax error fixing success rates. The results indicate, at least in the scope of this study, that the general error messages qualities do not explain query fixing success. We also analyzed qualitative data regarding suggestions on how to improve SQL error messages, and formulated a framework for SQL error message design. The framework guides error message design towards specifying where the error occurs, what causes the error and why, providing suggestions on how to fix the error, and showing examples of similar query concepts. Additionally, the framework emphasizes the ordering of information in the error message, the removal of unnecessary elements, and the use of plain English. Finally, and based on the formulated framework, we applied the framework and showed examples of how to design error messages for the sixteen most common SQL syntax errors. We suggest the industry to follow either these or previously published general system message guidelines.

## ACKNOWLEDGMENTS

The authors thank all who participated in the study, as well as the anonymous reviewers for their insights on how to improve the paper.

## REFERENCES

[1] Alireza Ahadi, Vahid Behbood, Arto Vihavainen, Julia Prior, and Raymond Lister. 2016. Students' Syntactic Mistakes in Writing Seven Different Types of SQL Queries and its Application to Predicting Students' Success. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE)*. ACM Press, New York, New York, USA, 401–406. https://doi.org/10.1145/2839509.2844640

[2] Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. 2016. Students' Semantic Mistakes in Writing Seven Different Types of SQL Queries. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Press, New York, New York, USA, 272–277. https://doi.org/10.1145/2899415.2899464

[3] Umair Z. Ahmed, Renuka Sindhgatta, Nisheeth Srivastava, and Amey Karkare. 2019. Targeted Example Generation for Compilation Errors. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. https://doi.org/10.1109/ase.2019.00039

[4] Andrei Alexandrescu. 1999. Better template error messages. *C/C++ Users Journal* 17 (1999), 37–47.

[5] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. 2017. Do Developers Read Compiler Error Messages?. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE. https://doi.org/10.1109/icse.2017.59

[6] Brett A. Becker. 2016. An Effective Approach to Enhancing Compiler Error Messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE)*. ACM Press. https://doi.org/10.1145/2839509.2844584

[7] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler Error Messages Considered Unhelpful.

In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM. https://doi.org/10.1145/3344429.3372508

[8] Brett A. Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. 2016. Effective compiler error message enhancement for novice programming students. *Computer Science Education* 26, 2-3 (2016), 148–175. https://doi.org/10.1080/08993408.2016.1225464

[9] Brett A. Becker, Cormac Murray, Tianyi Tao, Changheng Song, Robert McCartney, and Kate Sanders. 2018. Fix the First, Ignore the Rest: Dealing with Multiple Compiler Error Messages. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 634–639. https://doi.org/10.1145/3159450.3159453

[10] A. Faye Borthick, Paul L. Bowen, S.T Liew, and Fiona H. Rohde. 2001. The effects of normalization on end-user query errors: An experimental evaluation. *International Journal of Accounting Information Systems* 2, 4 (2001), 195 – 221. https://doi.org/10.1016/S1467-0895(01)00023-9

[11] Stefan Brass and Christian Goldberg. 2006. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software* 79, 5 (2006), 630–644. https://doi.org/10.1016/j.jss.2005.06.028

[12] Peter Brusilovsky, Sergey Sosnovsky, Michael V. Yudelson, Danielle H. Lee, Vladimir Zadorozhny, and Xin Zhou. 2010. Learning SQL Programming with Interactive Tools: From Integration to Personalization. *ACM Transactions on Computing Education* 9, 4, Article 19 (2010), 15 pages. https://doi.org/10.1145/1656255.1656257

[13] Kathleen M. Cauley and James H. McMillan. 2010. Formative Assessment Techniques to Support Student Motivation and Achievement. *The Clearing House: A Journal of Educational Strategies, Issues and Ideas* 83, 1 (2010), 1–6. https://doi.org/10.1080/00098650903267784

[14] Loris D'antoni, Dileep Kini, Rajeev Alur, Sumit Gulwani, Mahesh Viswanathan, and Björn Hartmann. 2015. How Can Automatic Feedback Help Students Construct Automata? *ACM Transactions on Computer-Human Interaction* 22, 2, Article 9 (2015), 24 pages. https://doi.org/10.1145/2723163

[15] Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. 2014. Enhancing Syntax Error Messages Appears Ineffectual. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE)*. Association for Computing Machinery, New York, NY, USA, 273–278. https://doi.org/10.1145/2591708.2591748

[16] Paul Denny, James Prather, Brett A. Becker, Catherine Mooney, John Homer, Zachary C Albrecht, and Garrett B. Powell. 2021. On Designing Programming Error Messages for Novices: Readability and its Constituent Factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3411764.3445696

[17] G.M. Donahue. 2001. Usability and the bottom line. *IEEE Software* 18, 1 (2001), 31–37. https://doi.org/10.1109/52.903161

[18] Davide Falessi, Natalia Juristo, Claes Wohlin, Burak Turhan, Jürgen Münch, Andreas Jedlitschka, and Markku Oivo. 2017. Empirical software engineering experts on the use of students and professionals in experiments. *Empirical Software Engineering* 23, 1 (2017), 452–489. https://doi.org/10.1007/s10664-017-9523-3

[19] Robert Feldt, Thomas Zimmermann, Gunnar R. Bergersen, Davide Falessi, Andreas Jedlitschka, Natalia Juristo, Jürgen Münch, Markku Oivo, Per Runeson, Martin Shepperd, Dag I. K. Sjøberg, and Burak Turhan. 2018. Four commentaries on the use of students and professionals in empirical software engineering experiments. *Empirical Software Engineering* 23, 6 (2018), 3801–3820. https://doi.org/10.1007/s10664-018-9655-0

[20] Mrunal Gawade and Martin Kersten. 2012. Stethoscope: A Platform for Interactive Visual Analysis of Query Execution Plans. *Proceedings of the VLDB Endowment* 5, 12 (2012), 1926–1929. https://doi.org/10.14778/2367502.2367539

[21] Goetz Graefe. 1993. Query Evaluation Techniques for Large Databases. *Comput. Surveys* 25, 2 (1993), 73–169. https://doi.org/10.1145/152610.152611

[22] Joseph M. Hellerstein, Michael Stonebraker, and James Hamilton. 2007. Architecture of a Database System. *Foundations and Trends in Databases* 1, 2 (2007), 141–259. https://doi.org/10.1561/1900000002

[23] Richard Higgins, Peter Hartley, and Alan Skelton. 2002. The Conscientious Consumer: Reconsidering the role of assessment feedback in student learning. *Studies in Higher Education* 27, 1 (2002), 53–64. https://doi.org/10.1080/03075070120099368

[24] Hsiu-Fang Hsieh and Sarah E Shannon. 2005. Three Approaches to Qualitative Content Analysis. *Qualitative Health Research* 15, 9 (2005), 1277–1288. https://doi.org/10.1177/1049732305276687

[25] E. Kantorowitz and H. Laor. 1986. Automatic generation of useful syntax error messages. *Software: Practice and Experience* 16, 7 (1986), 627–640. https://doi.org/10.1002/spe.4380160703

[26] Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi. 2011. Mind Your Language: On Novices' Interactions with Error Messages. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. Association for Computing Machinery, New York, NY, USA, 3–18. https://doi.org/10.1145/2048237.2048241

[27] Davin McCall and Michael Kölling. 2019. A New Look at Novice Programmer Errors. *ACM Transactions on Computing Education* 19, 4, Article 38 (2019), 30 pages. https://doi.org/10.1145/3335814

[28] Daphne Miedema, Efthimia Aivaloglou, and George Fletcher. 2021. Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study. In *Proceedings of the 17th ACM Conference on International Computing Education Research*. Association for Computing Machinery, New York, NY, USA, 355–367. https://doi.org/10.1145/3446871.3469759

[29] Daphne Miedema, George Fletcher, and Efthimia Aivaloglou. 2022. Expert Perspectives on Student Errors in SQL. *ACM Transactions on Computing Education* (2022). https://doi.org/10.1145/3551392

[30] Daphne Miedema, George Fletcher, and Efthimia Aivaloglou. 2022. So Many Brackets! An Analysis of How SQL Learners (Mis)Manage Complexity during Query Formulation. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension (ICPC '22)*. Association for Computing Machinery, New York, NY, USA, 122–132. https://doi.org/10.1145/3524610.3529158

[31] Rolf Molich and Jakob Nielsen. 1990. Improving a Human-Computer Dialogue. *Commun. ACM* 33, 3 (1990), 338–348. https://doi.org/10.1145/77481.77486

[32] Christian Murphy, Eunhee Kim, Gail Kaiser, and Adam Cannon. 2008. Backstop: A Tool for Debugging Runtime Errors. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*. Association for Computing Machinery, 173–177. https://doi.org/10.1145/1352135.1352193

[33] Emerson Murphy-Hill and Andrew P. Black. 2012. Programmer-Friendly Refactoring Errors. *IEEE Transactions on Software Engineering* 38, 6 (2012), 1417–1431. https://doi.org/10.1109/TSE.2011.110

[34] Jakob Nielsen. 1994. Enhancing the Explanatory Power of Usability Heuristics. In *Conference Companion on Human Factors in Computing Systems* (Boston, Massachusetts, USA) *(CHI '94)*. Association for Computing Machinery, New York, NY, USA, 210. https://doi.org/10.1145/259963.260333

[35] Jakob Nielsen and Rolf Molich. 1990. Heuristic Evaluation of User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Seattle, Washington, USA) *(CHI '90)*. Association for Computing Machinery, New York, NY, USA, 249–256. https://doi.org/10.1145/97243.97281

[36] Marie-Hélène Nienaltowski, Michela Pedroni, and Bertrand Meyer. 2008. Compiler error messages. *ACM SIGCSE Bulletin* 40, 1 (2008), 168–172. https://doi.org/10.1145/1352322.1352192

[37] Shunsuke Otawa, Kento Goto, and Motomichi Toyama. 2021. Automatic Correction of Syntax Errors in SuperSQL Queries. In *Proceedings of the 22nd International Conference on Information Integration and Web-Based Applications & Services (iiWAS '20)*. Association for Computing Machinery, New York, NY, USA, 28–33. https://doi.org/10.1145/3428757.3429131

[38] J.F. Pane, B.A. Myers, and L.B. Miller. 2002. Using HCI techniques to design a more usable programming system. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*. IEEE Computing Society. https://doi.org/10.1109/hcc.2002.1046372

[39] Andrew Pavlo and Matthew Aslett. 2016. What's Really New with NewSQL? *SIGMOD Record* 45, 2 (2016), 45–55. https://doi.org/10.1145/3003665.3003674

[40] Raymond S. Pettit, John Homer, and Roger Gee. 2017. Do Enhanced Compiler Error Messages Help Students? Results Inconclusive. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 465–470. https://doi.org/10.1145/3017680.3017768

[41] James Prather, Raymond Pettit, Kayla Holcomb McMurry, Alani Peters, John Homer, Nevan Simone, and Maxine Cohen. 2017. On Novices' Interaction with Compiler Error Messages: A Human Factors Approach. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. Association for Computing Machinery, New York, NY, USA, 74–82. https://doi.org/10.1145/3105726.3106169

[42] Gary B. Randolph. 2003. The Forest and the Trees: Using Oracle and SQL Server Together to Teach ANSI-standard SQL. In *Proceedings of the 4th ACM Conference on Information Technology Curriculum (CITC) (CITC4 '03)*. ACM, New York, NY, USA, 234–236. https://doi.org/10.1145/947121.947174

[43] Phyllis Reisner. 1977. Use of Psychological Experimentation as an Aid to Development of a Query Language. *IEEE Transactions on Software Engineering* SE-3, 3 (1977), 218–229. https://doi.org/10.1109/tse.1977.231131

[44] Phyllis Reisner. 1981. Human Factors Studies of Database Query Languages: A Survey and Assessment. *Comput. Surveys* 13, 1 (March 1981), 13–31. https://doi.org/10.1145/356835.356837

[45] Phyllis Reisner, Raymond F. Boyce, and Donald D. Chamberlin. 1975. Human factors evaluation of two data base query languages. In *Proceedings of the national computer conference and exposition AFIPS '75*. ACM Press. https://doi.org/10.1145/1499949.1500036

[46] Shin-Shing Shin. 2020. Structured Query Language Learning: Concept Map-Based Instruction Based on Cognitive Load Theory. *IEEE Access* 8 (2020), 100095–100110. https://doi.org/10.1109/ACCESS.2020.2997934

[47] Ben Shneiderman. 1982. Designing computer system messages. *Commun. ACM* 25, 9 (1982), 610–611. https://doi.org/10.1145/358628.358639

[48] Ben Shneiderman, Catherine Plaisant, Maxine S Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. 2016. *Designing the user interface: strategies for effective human-computer interaction*. Pearson.

[49] John B. Smelcer. 1995. User errors in database query composition. *International Journal of Human-Computer Studies* 42, 4 (1995), 353–381. https://doi.org/10.1006/ijhc.1995.1017

[50] Toni Taipalus. 2020. The effects of database complexity on SQL query formulation. *Journal of Systems and Software* 165 (2020), 110576. https://doi.org/10.1016/j.jss.2020.110576

[51] Toni Taipalus. 2020. Explaining Causes Behind SQL Query Formulation Errors. In *2020 IEEE Frontiers in Education Conference (FIE)*. 1–9. https://doi.org/10.1109/FIE44824.2020.9274114

[52] Toni Taipalus. 2023. Query Execution Plans and Semantic Errors: Usability and Educational Opportunities. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI EA '23)*. Association for Computing Machinery, New York, NY, USA, Article 239, 6 pages. https://doi.org/10.1145/3544549.3585794

[53] Toni Taipalus and Hilkka Grahn. 2023. NewSQL Database Management System Compiler Errors: Effectiveness and Usefulness. *International Journal of Human–Computer Interaction* (2023), 1–12. https://doi.org/10.1080/10447318.2022.2108648 arXiv:https://doi.org/10.1080/10447318.2022.2108648

[54] Toni Taipalus, Hilkka Grahn, and Hadi Ghanbari. 2021. Error messages in relational database management systems: A comparison of effectiveness, usefulness, and user confidence. *Journal of Systems and Software* 181 (2021), 111034. https://doi.org/10.1016/j.jss.2021.111034

[55] Toni Taipalus and Piia Perälä. 2019. What to Expect and What to Focus on in SQL Query Teaching. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE) (SIGCSE '19)*. ACM, New York, NY, USA, 198–203. https://doi.org/10.1145/3287324.3287359

[56] Toni Taipalus and Ville Seppänen. 2020. SQL Education: A Systematic Mapping Study and Future Research Agenda. *ACM Transactions on Computing Education* 20, 3, Article 20 (2020), 33 pages. https://doi.org/10.1145/3398377

[57] Toni Taipalus, Mikko Siponen, and Tero Vartiainen. 2018. Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education* 18, 3, Article 15 (2018), 29 pages. https://doi.org/10.1145/3231712

[58] Jess Tan, Desmond Yeo, Rachael Neoh, Huey-Eng Chua, and Sourav S Bhowmick. 2022. MOCHA: A Tool for Visualizing Impact of Operator Choices in Query Execution Plans for Database Education. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3602–3605. https://doi.org/10.14778/3554821.3554854

[59] Heikki Topi, Kate M. Kaiser, Janice C. Sipior, Joseph S. Valacich, J. F. Nunamaker, Jr., G. J. de Vreede, and Ryan Wright. 2010. *Curriculum Guidelines for Undergraduate Degree Programs in Information Systems.* Technical Report. New York, NY, USA. https://dl.acm.org/citation.cfm?id=2593310

[60] V. Javier Traver. 2010. On Compiler Error Messages: What They Say and What They Mean. *Advances in Human-Computer Interaction* (2010), 1–26. https://doi.org/10.1155/2010/602570

[61] Weiguo Wang, Sourav S. Bhowmick, Hui Li, Shafiq Joty, Siyuan Liu, and Peng Chen. 2021. Towards Enhancing Database Education: Natural Language Generation Meets Query Execution Plans. In *Proceedings of the 2021 International Conference on Management of Data.* Association for Computing Machinery, New York, NY, USA, 1933–1945. https://doi.org/10.1145/3448016.3452822

[62] C. Welty. 1985. Correcting user errors in SQL. *International Journal of Man-Machine Studies* 22, 4 (1985), 463–477. https://doi.org/10.1016/s0020-7373(85)80051-1

[63] Charles Welty and David W. Stemple. 1981. Human factors comparison of a procedural and a nonprocedural query language. *ACM Transactions on Database Systems* 6, 4 (1981), 626–649. https://doi.org/10.1145/319628.319656

[64] John Wrenn and Shriram Krishnamurthi. 2017. Error messages are classifiers: a process to design and evaluate error messages. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software.* ACM. https://doi.org/10.1145/3133850.3133862

## A SUGGESTED IMPROVEMENTS AND MODIFIED ERROR MESSAGES

```
SELECT name
FROM   supplier
JOIN   delivery
ON     (supplier.id = delivery.supplier_id)
JOIN   product
ON     (delivery.product_id = product.id)
WHERE  product.price_usd > 50
AND    product.brand = 'Apple';
```

(a) Erroneous query

```
SELECT supplier.name
FROM   supplier
JOIN   delivery
ON     (supplier.id = delivery.supplier_id)
JOIN   product
ON     (delivery.product_id = product.id)
WHERE  product.price_usd > 50
AND    product.brand = 'Apple';
```

(b) Fixed query

| Suggested improvement | Count (out of 108) |
|---|---|
| Suggest how to fix the error | 25 (23%) |
| Rephrase the error message in plain English | 19 (18%) |
| Specify the line number of the erroneous part | 18 (17%) |
| Specify error position and cause more accurately | 10 (9%) |
| Tell which tables contain the column that causes the error | 6 (6%) |
| Explain that there are multiple column with a similar name | 5 (5%) |
| Rephrase the error message in a way that the important parts come first | 5 (5%) |
| Explain that table alias or identifier is missing | 3 (3%) |
| Remove unclear terms (“field list”) | 2 (2%) |

(c) Suggestions for improvements

```
Line 1: SELECT name
               ^
There are multiple columns named 'name'.

HINT: Did you mean column 'name' in table 'supplier' or in table 'product'? Use identifiers to refer to
     columns with similar names. Typical identifiers are table names with the following syntax: <table>.<
     column>.

EXAMPLES:
SELECT product.name
FROM product;
```

(d) Modified error message

Fig. 1. Test T01 (ambiguous column)

```
SELECT fname, sname
FROM   employee
WHERE  id IN
    (SELECT employee_id
    FROM    works
    WHERE   project_id IN
        (SELECT id
        FROM project
        WHERE name = QA
        OR    name = HR)
    );
```

(a) Erroneous query

```
SELECT fname, sname
FROM   employee
WHERE  id IN
    (SELECT employee_id
    FROM    works
    WHERE   project_id IN
        (SELECT id
        FROM project
        WHERE name = 'QA'
        OR    name = 'HR')
    );
```

(b) Fixed query

| Suggested improvement | Count (out of 122) |
|---|---|
| Suggest single quotes around strings | 38 (31%) |
| Mention all errors | 23 (19%) |
| Suggest how to fix the error | 18 (15%) |
| Remove unnecessary information | 14 (11%) |
| Tell why the query is erroneous | 14 (11%) |
| Specify the line number of the erroneous part | 8 (7%) |
| Rephrase the error message in plain English | 7 (6%) |
| Rephrase the error message in a way that the important parts come first | 7 (6%) |
| Remove dramatic choices of words | 2 (2%) |
| Include a link to documentation | 2 (2%) |

(c) Suggestions for improvements

```
Line 9: WHERE name = QA
                     ^
Part of the query is not recognized.

HINT: If you are searching for the string "QA", it should be enclosed in single quotation marks.
     Alternatively, if you are joining two tables and "QA" is a column name, check spelling. There seem to
     be no columns with a name that is "QA" or similar.

EXAMPLES:
SELECT *
FROM product
WHERE name = 'machine';

SELECT product.id, product.price
FROM product, order
WHERE product.id = order.product_id;
```

(d) Modified error message

Fig. 2. Test T02 (omitting quotes around character data)

```
SELECT brand, model
FROM   product
WHERE  price_usd IS 350
AND    id IN
        (SELECT product_id
        FROM delivery
        WHERE amount > 100);
```

(a) Erroneous query

```
SELECT brand, model
FROM   product
WHERE  price_usd = 350
AND    id IN
        (SELECT product_id
        FROM delivery
        WHERE amount > 100);
```

(b) Fixed query

| Suggested improvement | Count (out of 141) |
|---|---|
| Suggest replacing IS with = | 53 (38%) |
| Remove unnecessary information | 40 (28%) |
| Specify the error position | 14 (10%) |
| Explain what causes the error and why the error occurs | 12 (9%) |
| Suggest how to fix the error | 9 (6%) |
| Specify the line number of the erroneous part | 7 (6%) |
| Rephrase the error message in a way that the important parts come first | 6 (4%) |
| Error message should point to the erroneous position | 6 (4%) |

(c) Suggestions for improvements

```
Line 3: WHERE  price_usd IS 350
                         ^
You have used IS to compare a number.

HINT: The keyword IS is used in finding empty values (NULL). If you are searching for empty values, use IS.
      Alternatively, if you are searching for rows with specific values, use appropriate operators such as
     comparison operators or LIKE.

EXAMPLES:
SELECT *
FROM product
WHERE price IS NULL;

SELECT *
FROM product
WHERE price = 100;
```

(d) Modified error message

Fig. 3. Test T03 (IS where not applicable)

```
SELECT id, name, status
FROM   project
WHERE  name LIKE ('H%', 'J%', 'K%')
AND    manager_id IN
        (SELECT id
         FROM    employee
         WHERE   sname = 'Smith');
```

(a) Erroneous query

```
SELECT id, name, status
FROM   project
WHERE  (name LIKE 'H%'
OR     name LIKE 'J%'
OR     name LIKE 'K%')
AND    manager_id IN
        (SELECT id
         FROM    employee
         WHERE   sname = 'Smith');
```

(b) Fixed query

| Suggested improvement | Count (out of 129) |
|---|---|
| Specify that the erroneous part concerns LIKE | 39 (30%) |
| Specify the line number of the erroneous part | 17 (13%) |
| Suggest how to fix the error | 14 (11%) |
| Rephrase the error message in plain English | 13 (10%) |
| Specify error position, a reference to a comma is ambiguous | 10 (8%) |
| Remove unnecessary information | 9 (7%) |
| Rephrase the error message in a way that the important parts come first | 5 (4%) |
| Should not point to parentheses, now gives incorrect error position | 5 (4%) |
| Explain that LIKE expects one value, not a list | 4 (3%) |
| Suggest to use logical operators AND or OR | 3 (2%) |

(c) Suggestions for improvements

```
Line 3: name LIKE ('H%', 'J%', 'K%')
                       ^
LIKE cannot be used with a list of values.

HINT: If you are using wildcards, consider using logical operators AND or OR between expressions.
      Alternatively, if you are not using wildcards, try replacing LIKE with IN.

EXAMPLES:
SELECT *
FROM product
WHERE name LIKE 'A%'
OR name LIKE 'B%';

SELECT *
FROM product
WHERE name IN ('Alpha', 'Beta');
```

(d) Modified error message

Fig. 4. Test T04 (confusing the syntax of keywords)

```
SELECT    e.sname, e.fname
FROM      employee e
JOIN      supplier s ON (e.city = s.city)
WHERE     s.id = 409
OR        s.id = 309
GROUP BY e.sname ASC, e.fname ASC;
```

(a) Erroneous query

```
SELECT    e.sname, e.fname
FROM      employee e
JOIN      supplier s ON (e.city = s.city)
WHERE     s.id = 409
OR        s.id = 309
ORDER BY e.sname ASC, e.fname ASC;
```

(b) Fixed query

| Suggested improvement | Count (out of 75) |
|---|---|
| Explain that ASC cannot be used in GROUP BY | 34 (45%) |
| Explain what causes the error and why the error occurs | 12 (16%) |
| Suggest how to fix the error | 10 (13%) |
| Specify error position | 6 (8%) |
| Rephrase the error message in plain English | 5 (7%) |
| Identify the error position correctly | 3 (4%) |
| Remove unnecessary information | 3 (4%) |

(c) Suggestions for improvements

```
Line 5: GROUP BY e.sname ASC, e.fname ASC
                         ^
You have used ASC or DESC in a GROUP BY clause.

HINT: ASC and DESC are used in the ORDER BY clause to specify the ordering of rows in the result table, and
      GROUP BY is used to group rows together. Did you mean "ORDER BY e.sname ASC, e.fname ASC"?

EXAMPLES:
SELECT price, COUNT(*)
FROM product
GROUP BY price;

SELECT name, price
FROM product
ORDER BY name ASC, price DESC;
```

(d) Modified error message

Fig. 5. T05 (confusing the logic of keywords)

```
SELECT id, fname, sname
FROM   employee
WHERE  id IN
        (SELECT employee_id
        FROM    works
        WHERE   project_id IN
            (SELECT id, manager_id
            FROM    project
            WHERE   manager_id =
                (SELECT id
                FROM    employee
                WHERE   city = 'Paris')
            )
        );
```

(a) Erroneous query

```
SELECT id, fname, sname
FROM   employee
WHERE  id IN
        (SELECT employee_id
        FROM    works
        WHERE   project_id IN
            (SELECT id
            FROM    project
            WHERE   manager_id =
                (SELECT id
                FROM    employee
                WHERE   city = 'Paris')
            )
        );
```

(b) Fixed query

| Suggested improvement | Count (out of 102) |
|---|---|
| Specify the error position | 41 (40%) |
| Specify the line number of the erroneous part | 32 (31%) |
| Suggest how to fix the error | 9 (9%) |
| Rephrase the error message in plain English | 8 (8%) |
| Identify the error position correctly | 5 (5%) |
| Tell why the query is erroneous | 5 (5%) |
| Tell that a subquery formulated with IN accepts only one column | 5 (5%) |
| Remove unnecessary information | 3 (3%) |

(c) Suggestions for improvements

```
Line 7: (SELECT id, manager_id
                    ^
You have used IN followed by a subquery that returns several columns.

HINT: IN is expecting one list of values (i.e., values from one column), but cannot handle several lists (i
     .e., values from several columns). Try specifying only one column in the subquery's SELECT clause,
     depending on which column you wish to use to join the tables.

EXAMPLES:
SELECT *
FROM project
WHERE id IN
    (SELECT project_id
    FROM order);
```

(d) Modified error message

Fig. 6. Test T06 (too many columns in subquery)

```
SELECT name
FROM   employee
WHERE  (city = 'New York'
OR     city = 'Minneapolis')
AND    id IN
    (SELECT manager_id
    FROM project
    WHERE status = 0);
```

(a) Erroneous query

```
SELECT fname, sname
FROM   employee
WHERE  (city = 'New York'
OR     city = 'Minneapolis')
AND    id IN
    (SELECT manager_id
    FROM project
    WHERE status = 0);
```

(b) Fixed query

| Suggested improvement | Count (out of 64) |
|---|---|
| Specify the line number of the erroneous part | 20 (31%) |
| Suggest how to fix the error | 13 (20%) |
| Replace uncommon nomenclature (message reads "field list" instead of "`SELECT` clause") | 10 (16%) |
| Explain what causes the error and why the error occurs | 7 (11%) |
| Remove unnecessary error codes | 5 (8%) |
| Rephrase the error message in plain English | 5 (8%) |
| Should not show parts of the replicated query with altered letter case | 4 (6%) |
| Propose close matches for column names | 3 (5%) |
| Tell which tables contain likely candidates | 2 (3%) |

(c) Suggestions for improvements

```
Line 1: SELECT name
               ^
You have referred to a column "name" which is not in table "employee".

HINT: did you mean "employee.sname" or "employee.fname"? Alternatively, ensure that the FROM clause
     contains the table which contains the column you wish to refer to.

EXAMPLES:
SELECT name, price
FROM product;
```

(d) Modified error message

Fig. 7. Test T07 (undefined column)

```
SELECT   name, price_usd, brand, model
FROM     product
WHRE     (brand LIKE 'S%' OR brand LIKE 'C%')
AND      picture IS NULL
ORDER BY name DESC;
```

(a) Erroneous query

```
SELECT   name, price_usd, brand, model
FROM     product
WHERE    (brand LIKE 'S%' OR brand LIKE 'C%')
AND      picture IS NULL
ORDER BY name DESC;
```

(b) Fixed query

| Suggested improvement | Count (out of 140) |
|---|---|
| Identify the error position correctly | 30 (21%) |
| Remove unnecessary information | 18 (13%) |
| Explain what causes the error and why the error occurs | 17 (12%) |
| Identify WHRE as a typographic error in one the common clause keywords | 9 (6%) |
| Should not suggest a misleading fix | 7 (5%) |
| Specify the line number of the erroneous part | 6 (4%) |

(c) Suggestions for improvements

```
Line 3: WHRE    (brand LIKE 'S%' OR brand LIKE 'C%')
        ^
You have written an expression without specifying a proper clause, or misspelled an SQL keyword.

HINT: Expressions in the form of <column> <operator> <value> are typically placed in a WHERE or HAVING
     clause. Check that you have a WHERE or a HAVING clause. You have written "WHRE". Did you mean "WHERE"?

EXAMPLES:
SELECT *
FROM product
WHERE price > 100;

SELECT color, AVG(price)
FROM product
GROUP BY color
HAVING AVG(price) > 100;
```

(d) Modified error message

Fig. 8. Test T08 (misspellings)

```
SELECT s.id, s.name, s.email
FROM   supplier s
WHERE  (s.email LIKE '%gmail.com'
OR      '%icloud.com')
AND    EXISTS
        (SELECT *
        FROM delivery d
        WHERE s.id = d.supplier_id);
```

(a) Erroneous query

```
SELECT s.id, s.name, s.email
FROM   supplier s
WHERE  (s.email LIKE '%gmail.com'
OR     s.email LIKE '%icloud.com')
AND    EXISTS
        (SELECT *
        FROM delivery d
        WHERE s.id = d.supplier_id);
```

(b) Fixed query

| Suggested improvement | Count (out of 75) |
|---|---|
| Specify that the expression is incomplete | 19 (25%) |
| Specify the error position | 19 (25%) |
| Rephrase the error message in plain English | 11 (15%) |
| Suggest how to fix the error | 10 (13%) |
| Specify the line number of the erroneous part | 9 (12%) |
| Explain what causes the error and why the error occurs | 6 (8%) |
| Explain that as many `LIKE`s are required as there a columns in the `WHERE` clause | 3 (4%) |

(c) Suggestions for improvements

```
Line 4: OR      '%icloud.com')
                ^
The expression is incomplete.

HINT: Expressions typically take the form of <column> <operator> <value>. You can specify multiple
     expressions in a WHERE clause by using logical operators AND and OR. Did you mean "s.email LIKE '%
     icloud.com'"?

EXAMPLES:
SELECT *
FROM product
WHERE name LIKE 'A%'
OR name LIKE 'B%';
```

(d) Modified error message

Fig. 9. Test T09 (failure to specify column name twice)

```
SELECT name, brand, model
FROM   product
WHERE  brand IN ('Google', 'Microsoft')
AND    picture IS NOT NULL
AND    price_usd > AVG(price_usd);
```

(a) Erroneous query

```
SELECT name, brand, model
FROM   product
WHERE  brand IN ('Google', 'Microsoft')
AND    picture IS NOT NULL
AND    price_usd >
      (SELECT AVG(price_usd)
      FROM product);
```

(b) Fixed query

| Suggested improvement | Count (out of 92) |
|---|---|
| Specify that aggregate functions cannot be placed in the WHERE clause | 24 (26%) |
| Specify the line number of the erroneous part | 23 (25%) |
| Explain where aggregate functions can be placed | 14 (15%) |
| Suggest how to fix the error | 10 (11%) |
| Rephrase the error message in plain English | 6 (7%) |
| Specify which function causes the error | 5 (5%) |
| Replace uncommon nomenclature (message reads “group function” instead of “aggregate function”) | 5 (5%) |
| Explain what causes the error and why the error occurs | 4 (4%) |

(c) Suggestions for improvements

```
Line 5: AND    price_usd > AVG(price_usd);
                           ^
You have used an aggregate function (AVG) in a WHERE clause. Aggregate functions are not allowed in the
     WHERE clause.

HINT: depending on what you are trying to accomplish, consider either moving the aggregate function in a
     HAVING clause or a SELECT clause. In the latter case, you may need to use a subquery.

EXAMPLES:
SELECT COUNT(*)
FROM product;

SELECT COUNT(*), color
FROM product
GROUP BY color
HAVING COUNT(*) > 10;

SELECT name
FROM product
WHERE price >
  (SELECT AVG(price)
  FROM product);
```

(d) Modified error message

Fig. 10. Test T10 (using an aggregate function outside SELECT or HAVING)

```
SELECT   e.city
       , p.status
       , COUNT(w.employee_id) AS number_of_emp
FROM      employee e, project p, works w
WHERE     e.id = w.employee_id
AND       p.id = w.project_id
AND       p.id BETWEEN 1000 AND 2000
GROUP BY e.city;
```

(a) Erroneous query

```
SELECT   e.city
       , p.status
       , COUNT(w.employee_id) AS number_of_emp
FROM      employee e, project p, works w
WHERE     e.id = w.employee_id
AND       p.id = w.project_id
AND       p.id BETWEEN 1000 AND 2000
GROUP BY e.city, p.status;
```

(b) Fixed query

| Suggested improvement | Count (out of 55) |
|---|---|
| Specify the line number of the erroneous part | 12 (22%) |
| Suggest how to fix the error | 11 (20%) |
| Remove unnecessary information | 9 (16%) |
| Explain that all grouping columns much also appear in the `GROUP BY` clause | 9 (16%) |
| Rephrase the error message in plain English | 8 (15%) |
| Explain what causes the error and why the error occurs | 4 (7%) |
| Confusingly, this message also specifies the database name “test.p.status” | 3 (5%) |

(c) Suggestions for improvements

```
Line 8: GROUP BY e.city;
               ^
SELECT clause contains grouping columns not present in the GROUP BY clause.

HINT: Depending on what you are trying to accomplish, you may want to add the column "p.status" to the
     GROUP BY clause. Alternatively, you may want to remove the column "p.status" from the SELECT clause.

EXAMPLES:
SELECT COUNT(*)
FROM product;

SELECT COUNT(*), color
FROM product
GROUP BY color;

SELECT COUNT(*), color, price
FROM product
GROUP BY color, price;
```

(d) Modified error message

Fig. 11. Test T11 (grouping error: extraneous grouping column)

```
SELECT name, price_usd
FROM   product
WHERE  brand == 'Oracle'
AND    id IN
    (SELECT product_id
     FROM    delivery
     WHERE   project_id IN
         (SELECT id
          FROM project
          WHERE name LIKE 'data%')
    );
```

(a) Erroneous query

```
SELECT name, price_usd
FROM   product
WHERE  brand = 'Oracle'
AND    id IN
    (SELECT product_id
     FROM    delivery
     WHERE   project_id IN
         (SELECT id
          FROM project
          WHERE name LIKE 'data%')
    );
```

(b) Fixed query

| Suggested improvement | Count (out of 84) |
|---|---|
| Remove unnecessary information | 14 (17%) |
| Should be more specific | 13 (15%) |
| Specify that the operator is erroneous | 13 (15%) |
| Suggest how to fix the error | 11 (13%) |
| Explain what causes the error and why the error occurs | 11 (13%) |
| Specify the line number of the erroneous part | 9 (11%) |
| Suggestion to fix the error should not cause uncertainty | 7 (8%) |
| Should explain that == is not a valid comparison operator, and suggest using = instead | 5 (6%) |
| Identify the error position correctly | 4 (5%) |
| Rephrase the error message in plain English | 3 (4%) |
| Error message should be shorter | 3 (4%) |
| Rephrase the error message in a way that the important parts come first | 2 (2%) |

(c) Suggestions for improvements

```
Line 3: WHERE  brand == 'Oracle'
                     ^
The operator '==' is not recognized.

HINT: For equal comparison, SQL uses the '=' operator, which can be used to compare data types such as
      numbers and strings. Did you mean "brand = 'Oracle'"?

EXAMPLES:
SELECT *
FROM product
WHERE price = 100;
```

(d) Modified error message

Fig. 12. Test T12 (nonstandard operators)

```
SELECT   s.name, s.email, s.city, s.tel
FROM     supplier s
JOIN     delivery d ON (s.id = d.supplier_id)
JOIN     project p ON (p.id = d.project_id)
WHERE    s.name = 'Athens'
WHERE    p.name = 'HR'
ORDER BY s.city ASC, s.name ASC;
```

(a) Erroneous query

```
SELECT   s.name, s.email, s.city, s.tel
FROM     supplier s
JOIN     delivery d ON (s.id = d.supplier_id)
JOIN     project p ON (p.id = d.project_id)
WHERE    s.name = 'Athens'
AND      p.name = 'HR'
ORDER BY s.city ASC, s.name ASC;
```

(b) Fixed query

| Suggested improvement | Count (out of 101) |
|---|---|
| Explain that multiple WHERE clauses are not allowed | 37 (37%) |
| Suggest how to fix the error | 17 (17%) |
| Explain what causes the error and why the error occurs | 13 (13%) |
| Specify the line number of the erroneous part | 6 (6%) |
| Simplify the error message because the error is simple | 6 (6%) |
| Rephrase the error message in plain English | 4 (4%) |
| Should be more specific | 4 (4%) |
| Rephrase the error message in a way that the important parts come first | 3 (3%) |
| Remove unhelpful error codes | 2 (2%) |

(c) Suggestions for improvements

```
Line 6: WHERE     p.name = 'HR'
        ^
The query seems to contain several WHERE clauses, yet a query may only contain one WHERE clause.

HINT: this is typically fixed by replacing the latter WHERE keyword with AND or OR, depending on what you
     are trying to accomplish. You may use logical operators (AND, OR) to write multiple expressions in a
     WHERE clause.

EXAMPLES:
SELECT *
FROM product
WHERE name LIKE 'A%'
OR name LIKE 'B%';
```

(d) Modified error message

Fig. 13. Test T13 (using WHERE twice)

```
SELECT p.name, p.status
FROM   project p
WHERE  10 =
    (SELECT COUNT(w.employee_id)
    FROM    works w
    WHERE   p.id = w.project_id
    AND     JOIN
        (SELECT *
        FROM    employee e
        WHERE   e.id = w.employee_id
        AND     e.city = 'London')
    );
```

(a) Erroneous query

```
SELECT p.name, p.status
FROM   project p
WHERE  10 =
    (SELECT COUNT(w.employee_id)
    FROM    works w
    WHERE   p.id = w.project_id
    AND     EXISTS
        (SELECT *
        FROM    employee e
        WHERE   e.id = w.employee_id
        AND     e.city = 'London')
    );
```

(b) Fixed query

| Suggested improvement | Count (out of 84) |
|---|---|
| Suggest how to fix the error | 15 (18%) |
| Specify that `JOIN` requires `ON` | 9 (11%) |
| Explain what causes the error and why the error occurs | 9 (11%) |
| Should be more specific | 9 (11%) |
| Remove unnecessary information | 9 (11%) |
| Show examples of how different table joins are used | 7 (8%) |
| Message should not be long without being specific | 7 (8%) |
| Specify error position | 7 (8%) |
| Specify the line number of the erroneous part | 4 (5%) |
| Specify that `JOIN` cannot be used in a `WHERE` clause | 4 (5%) |
| Suggest using `EXISTS` instead of `JOIN` | 2 (2%) |

(c) Suggestions for improvements

```
Line 7:     AND     JOIN
                    ^
The JOIN clause does not contain an ON keyword which is used to specify which tables and which columns are
     used for a table join.

HINT: Syntactically, the JOIN clause is placed between a FROM and a WHERE clause (if a WHERE clause is
     present). The JOIN keyword is followed by a table name, which is typically followed by the keyword ON
     (or alternatively, USING) which is used to specify the columns used for a table join. A query may
     contain several JOIN clauses.

EXAMPLES:
SELECT product.id
FROM product
JOIN order
ON (product.id = order.product_id);

SELECT product.id
FROM product
JOIN order
ON (product.id = order.product_id)
JOIN customer
ON (order.customer_id = customer.id)
WHERE customer.sname = 'Smith';
```

(d) Modified error message

Fig. 14. Test T14 (nonstandard keywords or standard keywords in wrong context)

```
SELECT p.name, p.price
FROM   product p
JOIN   delivery d ON (p.id = d.product_id)
JOIN   project j ON (d.project_id = j.id)
WHERE  p.picture IS NULL
AND    j.status = 1;
```

(a) Erroneous query

```
SELECT p.name, p.price_usd
FROM   product p
JOIN   delivery d ON (p.id = d.product_id)
JOIN   project j ON (d.project_id = j.id)
WHERE  p.picture IS NULL
AND    j.status = 1;
```

(b) Fixed query

| Suggested improvement | Count (out of 48) |
|---|---|
| Propose close matches for column names | 14 (29%) |
| Specify the line number of the erroneous part | 10 (21%) |
| In addition to erroneous column name, specify which table did not contain this column | 7 (15%) |
| Suggest how to fix the error | 5 (10%) |
| Remove unhelpful error codes | 5 (10%) |
| Specify the error position | 4 (8%) |
| Replace uncommon nomenclature (message reads “field list” instead of “SELECT clause”) | 4 (8%) |
| Explain what causes the error and why the error occurs | 3 (6%) |
| Rephrase the error message in plain English | 2 (4%) |
| Should not show parts of the replicated query with altered letter case | 2 (4%) |

(c) Suggestions for improvements

```
Line 1: SELECT p.name, p.price
                         ^
There is no column "price" in table "product".

HINT: Table "product" contains the following columns quite similar to "price": "price_usd". Perhaps you
     meant to write "price_usd" instead of "price"?

EXAMPLES:
SELECT product.id, order.id
FROM product, order
WHERE product.id = order.product_id;
```

(d) Modified error message

Fig. 15. Test T15 (synonyms)

```
SELECT name, manager_id
FROM   project
WHERE  id IN
        (SELECT project_id
        FROM    delivery
        WHERE   supplier_id IN
            SELECT id
            FROM    supplier
            WHERE   city = 'Sydney')
        );
```

(a) Erroneous query

```
SELECT name, manager_id
FROM   project
WHERE  id IN
        (SELECT project_id
        FROM    delivery
        WHERE   supplier_id IN
            (SELECT id
            FROM    supplier
            WHERE   city = 'Sydney')
        );
```

(b) Fixed query

| Suggested improvement | Count (out of 85) |
|---|---|
| Specify that the parentheses are unmatched | 40 (47%) |
| Explain what causes the error and why the error occurs | 24 (28%) |
| Remove unnecessary information | 10 (12%) |
| Specify the line number of the erroneous part | 10 (12%) |
| Suggest how to fix the error | 4 (5%) |
| Rephrase the error message in plain English | 3 (4%) |

(c) Suggestions for improvements

```
Line 7:     SELECT id
            ^
The query contains unmatched parentheses. It seems like at least one opening parenthesis is missing.

HINT: check that each parenthesis that is opened is also closed. Also check that all subqueries start with
     a parenthesis.

EXAMPLES:
SELECT product.id
FROM product
WHERE EXISTS
  (SELECT *
  FROM order
  WHERE product.id = order.product_id);
```

(d) Modified error message

Fig. 16. Test T16 (curly, square or unmatched brackets)

## B ERROR MESSAGE CODING

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_NON_UNIQ_ERROR: Column 'name' in field list`<br>`is ambiguous` | ● | ○ | ◐ | ● | ● |
| Oracle Database | `ORA-00918: column ambiguously defined` | ● | ● | ◐ | ○ | ● |
| PostgreSQL | `ERROR:  column reference "name" is ambiguous`<br>`LINE 1: SELECT  name`<br>`                ^` | ● | ○ | ◐ | ● | ● |
| SQL Server | `Msg 209, Level 16, State 1, Server q7410, Line 2`<br>`Ambiguous column name 'name'.` | ● | ● | ◐ | ● | ● |
| CockroachDB | `pq: column reference "name" is ambiguous`<br>`(candidates: supplier.name, product.name)` | ● | ● | ● | ● | ● |
| Singlestore | `ERROR 1052 ER_NON_UNIQ_ERROR: Column 'name'`<br>`in field list is ambiguous` | ● | ○ | ◐ | ● | ● |
| NuoDB | `Error 42000: field "NAME" is ambiguous` | ● | ○ | ◐ | ● | ● |
| VoltDB | `SQL error while compiling query: Error in`<br>`"SELECT  name`<br>`FROM    supplier`<br>`JOIN    delivery`<br>`ON (supplier.id = delivery.supplier_id)`<br>`JOIN    product`<br>`ON (delivery.product_id = product.id)`<br>`WHERE   product.price_usd > 50`<br>`AND     product.brand = 'Apple';"`<br>`Column "NAME" is ambiguous.`<br>`It's in tables: PRODUCT, SUPPLIER` | ○ | ○ | ● | ● | ● |

Fig. 17. Error messages and coding for test T01

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_BAD_FIELD_ERROR: Unknown column 'QA' in`<br>`'where clause'` | ● | ○ | ◐ | ● | ● |
| Oracle Database | `ORA-00904: "HR": invalid identifier` | ● | ○ | ◐ | ● | ◐ |
| PostgreSQL | `ERROR:  column "qa" does not exist`<br>`LINE 9:         WHERE name = QA`<br>`                             ^` | ● | ○ | ◐ | ● | ● |
| SQL Server | `Msg 207, Level 16, State 1, Server q7410, Line 10`<br>`Invalid column name 'QA'.`<br>`Msg 207, Level 16, State 1, Server q7410, Line 11`<br>`Invalid column name 'HR'.` | ◐ | ○ | ◐ | ● | ◐ |
| CockroachDB | `pq: column "qa" does not exist` | ● | ● | ◐ | ● | ● |
| SingleStore | `ERROR 1054 ER_BAD_FIELD_ERROR: Unknown column 'QA'`<br>`in 'where clause'` | ● | ○ | ◐ | ● | ● |
| NuoDB | `Error 58000: can't resolve field "HR"` | ● | ○ | ◐ | ● | ◐ |
| VoltDB | `SQL error while compiling query: Error in`<br>`"SELECT fname, sname`<br>`FROM   employee`<br>`WHERE  id IN`<br>`    (SELECT employee_id`<br>`    FROM    works`<br>`    WHERE   project_id IN`<br>`        (SELECT id`<br>`        FROM project`<br>`        WHERE name = QA`<br>`        OR    name = HR)`<br>`    );" - object not found: QA` | ● | ○ | ◐ | ● | ◐ |

Fig. 18. Error messages and coding for test T02

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_PARSE_ERROR: You have an error in your SQL`<br>`syntax; check the manual that corresponds to your`<br>`MySQL server version for the right syntax to use near`<br>`'350 AND id IN (SELECT product_id FROM delivery WHERE`<br>`' at line 3` | ◐ | ○ | ○ | ● | ○ |
| Oracle Database | `ORA-00908: missing NULL keyword` | ● | ● | ◐ | ○ | ○ |
| PostgreSQL | `ERROR:  syntax error at or near "350"`<br>`LINE 3: WHERE  price_usd IS 350`<br>`                            ^` | ● | ○ | ○ | ● | ○ |
| SQL Server | `Msg 102, Level 15, State 1, Server q7410, Line 4`<br>`Incorrect syntax near '350'.` | ● | ○ | ○ | ◐ | ○ |
| CockroachDB | `invalid syntax: statement ignored: at or near "350":`<br>`syntax error`<br>`DETAIL: source SQL:`<br>`SELECT brand, model`<br>`FROM   product`<br>`WHERE  price_usd IS 350`<br>`                    ^`<br>`HINT: try \h SELECT` | ◐ | ○ | ○ | ● | ○ |
| SingleStore | `ERROR 1064 ER_PARSE_ERROR: You have an error in your`<br>`SQL syntax; check the manual that corresponds to your`<br>`MySQL server version for the right syntax to use near`<br>`'350`<br>`AND    id IN`<br>`        (SELECT product_id`<br>`        FROM delivery`<br>`        WH' at line 3` | ○ | ○ | ○ | ● | ○ |
| NuoDB | `Error 42000: syntax error on line 3`<br>`WHERE  price_usd IS 350`<br>`                    ^ expected NOT or NULL got 350` | ● | ○ | ◐ | ● | ◐ |
| VoltDB | `SQL error while compiling query: SQL Syntax error in`<br>`"SELECT brand, model`<br>`FROM   product`<br>`WHERE  price_usd IS 350`<br>`AND    id IN`<br>`        (SELECT product_id`<br>`        FROM delivery`<br>`        WHERE amount > 100);" unexpected token: 350` | ● | ○ | ◐ | ● | ○ |

Fig. 19. Error messages and coding for test T03

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_OPERAND_COLUMNS: Operand should contain 1`<br>`column(s)` | ● | ○ | ◐ | ○ | ○ |
| Oracle Database | `ORA-00907: missing right parenthesis` | ● | ● | ◐ | ○ | ○ |
| PostgreSQL | `ERROR:  operator does not exist: character varying`<br>`~~ record`<br>`LINE 3: WHERE  name LIKE ('H%', 'J%', 'K%')`<br>`                    ^`<br>`HINT:  No operator matches the given name and`<br>`argument types. You might need to add explicit`<br>`type casts.` | ○ | ○ | ● | ● | ◐ |
| SQL Server | `Msg 102, Level 15, State 1, Server q7410, Line 4`<br>`Incorrect syntax near ','.` | ● | ○ | ○ | ◐ | ○ |
| CockroachDB | `pq: unsupported comparison operator: <varchar> LIKE`<br>`<tuple{string, string, string}>` | ● | ● | ◐ | ◐ | ◐ |
| SingleStore | `ERROR 1064 ER_PARSE_ERROR: You have an error in your`<br>`SQL syntax; check the manual that corresponds to your`<br>`MySQL server version for the right syntax to use near`<br>`', 'J%', 'K%')`<br>`AND    manager_id IN`<br>`        (SELECT id`<br>`        FROM    employe' at line 3` | ○ | ○ | ○ | ◐ | ○ |
| NuoDB | `Error 0A000: multi-column value only allowed in`<br>`comparison operators` | ● | ○ | ◐ | ○ | ○ |
| VoltDB | `SQL error while compiling query: SQL Syntax error in`<br>`"SELECT id, name, status`<br>`FROM   project`<br>`WHERE  name LIKE ('H%', 'J%', 'K%')`<br>`AND    manager_id IN`<br>`        (SELECT id`<br>`        FROM    employee`<br>`        WHERE   sname = 'Smith');" unexpected token:`<br>`, required: )` | ● | ○ | ◐ | ◐ | ○ |

Fig. 20. Error messages and coding for test T04

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| Oracle Database | `ORA-00933: SQL command not properly ended` | ● | ○ | ◐ | ○ | ○ |
| PostgreSQL | `ERROR:  syntax error at or near "ASC"`<br>`LINE 6: GROUP BY e.sname ASC, e.fname ASC;`<br>`^` | ● | ○ | ○ | ● | ○ |
| SQL Server | `Msg 156, Level 15, State 1, Server q7410, Line 7`<br>`Incorrect syntax near the keyword 'ASC'.` | ● | ○ | ○ | ◐ | ○ |
| CockroachDB | `invalid syntax: statement ignored: at or near`<br>`"asc": syntax error`<br>`DETAIL: source SQL:`<br>`SELECT   e.sname, e.fname`<br>`FROM     employee e`<br>`JOIN     supplier s ON (e.city = s.city)`<br>`WHERE    s.id = 409`<br>`OR       s.id = 309`<br>`GROUP BY e.sname ASC, e.fname ASC`<br>`^` | ◐ | ○ | ○ | ● | ○ |
| SingleStore | `ERROR 1064 ER_PARSE_ERROR: You have an error in your`<br>`SQL syntax; check the manual that corresponds to your`<br>`MySQL server version for the right syntax to use near`<br>`'ASC, e.fname ASC' at line 6` | ○ | ○ | ○ | ● | ○ |
| NuoDB | `Error 42000: syntax error on line 6`<br>`GROUP BY e.sname ASC, e.fname ASC;`<br>`^ expected end of statement got ASC` | ● | ○ | ◐ | ● | ○ |
| VoltDB | `SQL error while compiling query: Error in`<br>`"SELECT   e.sname, e.fname`<br>`FROM     employee e`<br>`JOIN     supplier s ON (e.city = s.city)`<br>`WHERE    s.id = 409`<br>`OR       s.id = 309`<br>`GROUP BY e.sname ASC, e.fname ASC;" - expression`<br>`not in aggregate or GROUP BY columns: E.FNAME` | ● | ○ | ◐ | ● | ◐ |

Fig. 21. Error messages and coding for test T05; MySQL tolerated the syntax error

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_OPERAND_COLUMNS: Operand should contain 1 column(s)` | ● | ○ | ◐ | ○ | ○ |
| Oracle Database | `ORA-00913: too many values` | ● | ● | ◐ | ○ | ◐ |
| PostgreSQL | `ERROR:  subquery has too many columns`<br>`LINE 6:         WHERE   project_id IN`<br>`                                    ^` | ● | ○ | ◐ | ● | ● |
| SQL Server | `Msg 116, Level 16, State 1, Server q7410, Line 14`<br>`Only one expression can be specified in the select list when the subquery is not introduced with EXISTS.` | ◐ | ● | ◐ | ○ | ● |
| CockroachDB | `pq: unsupported comparison operator: <int> IN <tuple{tuple{int AS id, int AS manager_id}}>` | ● | ● | ◐ | ◐ | ◐ |
| SingleStore | `ERROR 1241 ER_OPERAND_COLUMNS: Operand should contain 1 column(s)` | ● | ○ | ◐ | ○ | ○ |
| NuoDB | `Error 0A000: multi-column subquery not supported` | ● | ○ | ◐ | ○ | ◐ |
| VoltDB | `SQL error while compiling query: Error in`<br>`"SELECT id, fname, sname`<br>`FROM   employee`<br>`WHERE  id IN`<br>`        (SELECT employee_id`<br>`        FROM    works`<br>`        WHERE   project_id IN`<br>`            (SELECT id, manager_id`<br>`            FROM    project`<br>`            WHERE   manager_id =`<br>`                (SELECT id`<br>`                FROM    employee`<br>`                WHERE   city = 'Paris')`<br>`            )`<br>`        );" - row column count mismatch` | ● | ○ | ◐ | ○ | ○ |

Fig. 22. Error messages and coding for test T06

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_BAD_FIELD_ERROR: Unknown column 'name' in 'field list'` | ● | ○ | ◐ | ● | ◐ |
| Oracle Database | `ORA-00904: "NAME": invalid identifier` | ● | ○ | ◐ | ● | ◐ |
| PostgreSQL | `ERROR: column "name" does not exist LINE 1: SELECT name ^ HINT: Perhaps you meant to reference the column "employee.fname" or the column "employee.sname".` | ◐ | ○ | ● | ● | ● |
| SQL Server | `Msg 207, Level 16, State 1, Server q7410, Line 2 Invalid column name 'name'.` | ● | ○ | ◐ | ● | ◐ |
| CockroachDB | `pq: column "name" does not exist` | ● | ● | ◐ | ● | ● |
| SingleStore | `ERROR 1054 ER_BAD_FIELD_ERROR: Unknown column 'name' in 'field list'` | ● | ○ | ◐ | ◐ | ◐ |
| NuoDB | `Error 58000: can't resolve field "NAME"` | ● | ○ | ◐ | ● | ◐ |
| VoltDB | `SQL error while compiling query: Error in "SELECT name FROM employee WHERE (city = 'New York' OR city = 'Minneapolis') AND id IN (SELECT manager_id FROM project WHERE status = 0);" - object not found: NAME` | ● | ○ | ○ | ● | ◐ |

Fig. 23. Error messages and coding for test T07

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_PARSE_ERROR: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '(brand LIKE 'S%' OR brand LIKE 'C%') AND picture IS NULL ORDER BY name DESC' at line 3` | ◐ | ○ | ○ | ◐ | ○ |
| Oracle Database | `ORA-00933: SQL command not properly ended` | ● | ○ | ◐ | ○ | ○ |
| PostgreSQL | `ERROR: syntax error at or near "LIKE" LINE 3: WHRE (brand LIKE 'S%' OR brand LIKE 'C%') ^` | ● | ○ | ○ | ● | ○ |
| SQL Server | `Msg 321, Level 15, State 1, Server q7410, Line 4 "brand" is not a recognized table hints option.` | ● | ● | ◐ | ◐ | ○ |
| CockroachDB | `invalid syntax: statement ignored: at or near "like": syntax error DETAIL: source SQL: SELECT name, price_usd, brand, model FROM product WHRE (brand LIKE 'S%' OR brand LIKE 'C%') ^ HINT: try \h <SOURCE>` | ◐ | ○ | ○ | ● | ○ |
| SingleStore | `ERROR 1064 ER_PARSE_ERROR: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '(brand LIKE 'S%' OR brand LIKE 'C%') AND picture IS NULL ORDER BY name DE' at line 3` | ○ | ○ | ○ | ◐ | ○ |
| NuoDB | `Error 42000: syntax error on line 3 WHRE (brand LIKE 'S%' OR brand LIKE 'C%') ^ expected end of statement got parenthesis` | ● | ○ | ◐ | ● | ○ |
| VoltDB | `SQL error while compiling query: SQL Syntax error in "SELECT name, price_usd, brand, model FROM product WHRE (brand LIKE 'S%' OR brand LIKE 'C%') AND picture IS NULL ORDER BY name DESC;" unexpected token: LIKE required: )` | ● | ○ | ◐ | ◐ | ○ |

Fig. 24. Error messages and coding for test T08

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| Oracle Database | `ORA-00920: invalid relational operator` | ● | ○ | ◐ | ○ | ○ |
| PostgreSQL | `ERROR:  invalid input syntax for type boolean:`<br>`"%icloud.com"`<br>`LINE 4: WHERE  (s.email LIKE '%gmail.com'`<br>`        OR     '%icloud.com')`<br>`              ^` | ● | ○ | ◐ | ● | ○ |
| SQL Server | `Msg 4145, Level 15, State 1, Server q7410, Line 4`<br>`An expression of non-boolean type specified in a`<br>`context where a condition is expected, near ')'.` | ◐ | ● | ◐ | ◐ | ○ |
| CockroachDB | `pq: could not parse "%icloud.com" as type bool:`<br>`invalid bool value` | ● | ○ | ◐ | ● | ○ |
| NuoDB | `Error 22000: error converting to boolean from`<br>`'%icloud.com' of type string` | ● | ○ | ◐ | ◐ | ○ |
| VoltDB | `SQL error while compiling query: Error in`<br>`"SELECT s.id, s.name, s.email`<br>`FROM   supplier s`<br>`WHERE  (s.email LIKE '%gmail.com'`<br>`OR     '%icloud.com')`<br>`AND    EXISTS`<br>`        (SELECT *`<br>`        FROM delivery d`<br>`        WHERE s.id = d.supplier_id);"`<br>`- data type of expression is not boolean` | ● | ○ | ◐ | ○ | ○ |

Fig. 25. Error messages and coding for test T09; MySQL and SingleStore tolerated the syntax error

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_INVALID_GROUP_FUNC_USE: Invalid use of group function` | ● | ○ | ◐ | ○ | ◐ |
| Oracle Database | `ORA-00934: group function is not allowed here` | ● | ● | ◐ | ○ | ● |
| PostgreSQL | `ERROR:  aggregate functions are not allowed in WHERE`<br>`LINE 5: AND    price_usd > AVG(price_usd);`<br>`                          ^` | ● | ○ | ◐ | ● | ● |
| SQL Server | `Msg 147, Level 15, State 1, Server q7410, Line 6`<br>`An aggregate may not appear in the WHERE clause unless it is in a subquery contained in a HAVING clause or a select list, and the column being aggregated is an outer reference.` | ○ | ● | ◐ | ◐ | ● |
| CockroachDB | `pq: avg(): aggregate functions are not allowed in WHERE` | ● | ● | ◐ | ◐ | ● |
| SingleStore | `ERROR 1111 ER_INVALID_GROUP_FUNC_USE: Invalid use of group function` | ● | ○ | ◐ | ○ | ◐ |
| NuoDB | `Error 42000: Aggregate functions not allowed in WHERE clause` | ● | ○ | ◐ | ● | ● |
| VoltDB | `SQL error while compiling query: Error in`<br>`"SELECT name, brand, model`<br>`FROM   product`<br>`WHERE  brand IN ('Google', 'Microsoft')`<br>`AND    picture IS NOT NULL`<br>`AND    price_usd > AVG(price_usd);"`<br>`- invalid WHERE expression` | ● | ○ | ◐ | ◐ | ◐ |

Fig. 26. Error messages and coding for test T10

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_WRONG_FIELD_WITH_GROUP: Expression #2 of`<br>`SELECT list is not in GROUP BY clause and contains`<br>`nonaggregated column 'test.p.status' which is not`<br>`functionally dependent on columns in GROUP BY clause;`<br>`this is incompatible with sql_mode=only_full_group_by` | ○ | ○ | ◐ | ● | ● |
| Oracle Database | `ORA-00979: not a GROUP BY expression` | ● | ● | ◐ | ○ | ◐ |
| PostgreSQL | `ERROR:  column "p.status" must appear in the`<br>`GROUP BY clause or be used in an aggregate function`<br>`LINE 2:        , p.status`<br>`                 ^` | ● | ○ | ● | ● | ● |
| SQL Server | `Msg 8120, Level 16, State 1, Server q7410, Line 2`<br>`Column 'project.status' is invalid in the select list`<br>`because it is not contained in either an aggregate`<br>`function or the GROUP BY clause.` | ◐ | ○ | ◐ | ● | ● |
| CockroachDB | `pq: column "status" must appear in the GROUP BY`<br>`clause or be used in an aggregate function` | ● | ● | ◐ | ◐ | ● |
| NuoDB | `Error 42000: column P.STATUS must appear in the`<br>`GROUP BY clause or be used in an aggregate function` | ● | ○ | ◐ | ● | ● |
| VoltDB | `SQL error while compiling query: Error in`<br>`"SELECT   e.city`<br>`       , p.status`<br>`       , COUNT(w.employee_id) AS number_of_emp`<br>`FROM     employee e, project p, works w`<br>`WHERE    e.id = w.employee_id`<br>`AND      p.id = w.project_id`<br>`AND      p.id BETWEEN 1000 AND 2000`<br>`GROUP BY e.city;" - expression not in aggregate`<br>`or GROUP BY columns: P.STATUS` | ● | ○ | ◐ | ● | ◐ |

Fig. 27. Error messages and coding for test T11; SingleStore tolerated the syntax error

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_PARSE_ERROR: You have an error in your SQL`<br>`syntax; check the manual that corresponds to your`<br>`MySQL server version for the right syntax to use near`<br>`'== 'Oracle' AND id IN (SELECT product_id`<br>`FROM delivery WHERE ' at line 3` | ◐ | ○ | ○ | ◐ | ○ |
| Oracle Database | `ORA-00936: missing expression` | ● | ● | ◐ | ○ | ○ |
| PostgreSQL | `ERROR:  operator does not exist: character varying`<br>`== unknown`<br>`LINE 3: WHERE  brand == 'Oracle'`<br>`                     ^`<br>`HINT:  No operator matches the given name and`<br>`argument types. You might need to add explicit`<br>`type casts.` | ○ | ○ | ● | ● | ● |
| SQL Server | `Msg 102, Level 15, State 1, Server q7410, Line 4`<br>`Incorrect syntax near '='.` | ● | ○ | ○ | ◐ | ◐ |
| CockroachDB | `invalid syntax: statement ignored: at or near`<br>`"=": syntax error`<br>`DETAIL: source SQL:`<br>`SELECT name, price_usd`<br>`FROM   product`<br>`WHERE  brand == 'Oracle'`<br>`              ^`<br>`HINT: try \h SELECT` | ◐ | ○ | ○ | ● | ○ |
| SingleStore | `ERROR 1064 ER_PARSE_ERROR: You have an error in your`<br>`SQL syntax; check the manual that corresponds to your`<br>`MySQL server version for the right syntax to use near`<br>`'=' at line 3` | ○ | ○ | ○ | ◐ | ○ |
| NuoDB | `Error 42000: syntax error on line 3`<br>`WHERE  brand == 'Oracle'`<br>`             ^ unexpected =` | ● | ○ | ◐ | ● | ● |
| VoltDB | `SQL error while compiling query: Error in`<br>`"SELECT name, price_usd`<br>`FROM   product`<br>`WHERE  brand == 'Oracle'`<br>`AND    id IN`<br>`    (SELECT product_id`<br>`    FROM    delivery`<br>`    WHERE   project_id IN`<br>`        (SELECT id`<br>`        FROM project`<br>`        WHERE name LIKE 'data%')`<br>`    )" - object not found: =` | ● | ○ | ◐ | ◐ | ○ |

Fig. 28. Error messages and coding for test T12

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_PARSE_ERROR: You have an error in your SQL`<br>`syntax; check the manual that corresponds to your`<br>`MySQL server version for the right syntax to use near`<br>`'WHERE p.name = 'HR' ORDER BY s.city ASC, s.name ASC'`<br>`at line 6` | ◐ | ○ | ○ | ◐ | ○ |
| Oracle Database | `ORA-00933: SQL command not properly ended` | ● | ○ | ◐ | ○ | ○ |
| PostgreSQL | `ERROR:  syntax error at or near "WHERE"`<br>`LINE 6: WHERE    p.name = 'HR'`<br>`        ^` | ● | ○ | ○ | ● | ○ |
| SQL Server | `Msg 156, Level 15, State 1, Server q7410, Line 7`<br>`Incorrect syntax near the keyword 'WHERE'.` | ● | ○ | ○ | ◐ | ○ |
| CockroachDB | `invalid syntax: statement ignored: at or near`<br>`"where": syntax error`<br>`DETAIL: source SQL:`<br>`SELECT   s.name, s.email, s.city, s.tel`<br>`FROM     supplier s`<br>`JOIN     delivery d ON (s.id = d.supplier_id)`<br>`JOIN     project p ON (p.id = d.project_id)`<br>`WHERE    s.name = 'Athens'`<br>`WHERE    p.name = 'HR'`<br>`^` | ● | ○ | ○ | ● | ○ |
| SingleStore | `ERROR 1064 ER_PARSE_ERROR: You have an error in your`<br>`SQL syntax; check the manual that corresponds to your`<br>`MySQL server version for the right syntax to use near`<br>`'WHERE    p.name = 'HR'`<br>`ORDER BY s.city ASC, s.name ASC' at line 6` | ○ | ○ | ○ | ◐ | ○ |
| NuoDB | `Error 42000: syntax error on line 6`<br>`WHERE    p.name = 'HR'`<br>`^ expected end of statement got WHERE` | ● | ○ | ◐ | ● | ○ |
| VoltDB | `SQL error while compiling query: SQL Syntax error in`<br>`"SELECT   s.name, s.email, s.city, s.tel`<br>`FROM     supplier s`<br>`JOIN     delivery d ON (s.id = d.supplier_id)`<br>`JOIN     project p ON (p.id = d.project_id)`<br>`WHERE    s.name = 'Athens'`<br>`WHERE    p.name = 'HR'`<br>`ORDER BY s.city ASC, s.name ASC;" unexpected token:`<br>`         WHERE` | ● | ○ | ◐ | ◐ | ◐ |

Fig. 29. Error messages and coding for test T13

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_PARSE_ERROR: You have an error in your SQL syntax;`<br>`check the manual that corresponds to your MySQL server version`<br>`for the right syntax to use near`<br>`'JOIN (SELECT * FROM employee e WHERE e.id = w.emplo' at line 7` | ◐ | ○ | ○ | ◐ | ○ |
| Oracle Database | `ORA-00936: missing expression` | ● | ● | ◐ | ○ | ○ |
| PostgreSQL | `ERROR:  syntax error at or near "SELECT"`<br>`LINE 8:         (SELECT *`<br>`                 ^` | ● | ○ | ○ | ● | ○ |
| SQL Server | `Msg 156, Level 15, State 1, Server q7410, Line 8`<br>`Incorrect syntax near the keyword 'JOIN'.`<br>`Msg 102, Level 15, State 1, Server q7410, Line 13`<br>`Incorrect syntax near ')'.` | ◐ | ○ | ○ | ◐ | ○ |
| CockroachDB | `invalid syntax: statement ignored: at or near "select": syntax error`<br>`DETAIL: source SQL:`<br>`SELECT p.name, p.status`<br>`FROM   project p`<br>`WHERE  10 =`<br>`    (SELECT COUNT(w.employee_id)`<br>`    FROM    works w`<br>`    WHERE   p.id = w.project_id`<br>`    AND     JOIN`<br>`        (SELECT *`<br>`         ^` | ● | ○ | ○ | ● | ○ |
| SingleStore | `ERROR 1064 ER_PARSE_ERROR: You have an error in your SQL syntax;`<br>`check the manual that corresponds to your MySQL server version`<br>`for the right syntax to use near`<br>`'JOIN`<br>`        (SELECT *`<br>`        FROM    employee e`<br>`        WHERE   e.id = w.em' at line 7` | ○ | ○ | ○ | ◐ | ○ |
| NuoDB | `Error 42000: syntax error on line 7`<br>`AND     JOIN`<br>`        ^ unexpected JOIN` | ● | ○ | ◐ | ● | ◐ |
| VoltDB | `SQL error while compiling query: SQL Syntax error in`<br>`"SELECT p.name, p.status`<br>`FROM   project p`<br>`WHERE  10 =`<br>`    (SELECT COUNT(w.employee_id)`<br>`    FROM    works w`<br>`    WHERE   p.id = w.project_id`<br>`    AND     JOIN`<br>`        (SELECT *`<br>`        FROM    employee e`<br>`        WHERE   e.id = w.employee_id`<br>`        AND     e.city = 'London')`<br>`    );" unexpected token: JOIN` | ● | ○ | ◐ | ◐ | ◐ |

Fig. 30. Error messages and coding for test T14

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_BAD_FIELD_ERROR: Unknown column 'p.price' in 'field list'` | ● | ○ | ◐ | ● | ● |
| Oracle Database | `ORA-00904: "P"."PRICE": invalid identifier` | ● | ○ | ◐ | ● | ◐ |
| PostgreSQL | `ERROR:  column p.price does not exist LINE 1: SELECT p.name, p.price ^` | ● | ○ | ◐ | ● | ● |
| SQL Server | `Msg 207, Level 16, State 1, Server q7410, Line 2 Invalid column name 'price'.` | ● | ○ | ◐ | ● | ● |
| CockroachDB | `pq: column "p.price" does not exist` | ● | ● | ◐ | ● | ● |
| SingleStore | `ERROR 1054 ER_BAD_FIELD_ERROR: Unknown column 'p.price' in 'field list'` | ● | ○ | ◐ | ● | ● |
| NuoDB | `Error 58000: can't resolve field "P.PRICE"` | ● | ○ | ◐ | ● | ◐ |
| VoltDB | `SQL error while compiling query: Error in "SELECT p.name, p.price FROM   product p JOIN   delivery d ON (p.id = d.product_id) JOIN   project j ON (d.project_id = j.id) WHERE  p.picture IS NULL AND    j.status = 1;" - object not found: P.PRICE` | ● | ○ | ◐ | ● | ◐ |

Fig. 31. Error messages and coding for test T15

| DBMS | Error message | brief | positive | constructive | specific | comprehensible |
|---|---|---|---|---|---|---|
| MySQL | `Error: ER_PARSE_ERROR: You have an error in your SQL syntax; check`<br>`the manual that corresponds to your MySQL server version for the`<br>`right syntax to use near`<br>`'SELECT id FROM supplier WHERE city = 'Sydney') '`<br>`at line 7` | ◐ | ○ | ○ | ◐ | ○ |
| Oracle Database | `ORA-00936: missing expression` | ● | ● | ◐ | ○ | ○ |
| PostgreSQL | `ERROR:  syntax error at or near "SELECT"`<br>`LINE 7:             SELECT id`<br>`                     ^` | ● | ○ | ○ | ● | ○ |
| SQL Server | `Msg 156, Level 15, State 1, Server q7410, Line 8`<br>`Incorrect syntax near the keyword 'SELECT'.`<br>`Msg 102, Level 15, State 1, Server q7410, Line 10`<br>`Incorrect syntax near ')'.` | ◐ | ○ | ○ | ◐ | ○ |
| CockroachDB | `invalid syntax: statement ignored: at or near "select": syntax error`<br>`DETAIL: source SQL:`<br>`SELECT name, manager_id`<br>`FROM   project`<br>`WHERE  id IN`<br>`        (SELECT project_id`<br>`        FROM    delivery`<br>`        WHERE   supplier_id IN`<br>`            SELECT id`<br>`            ^`<br>`HINT: try \h SELECT` | ◐ | ○ | ○ | ● | ○ |
| SingleStore | `ERROR 1064 ER_PARSE_ERROR: You have an error in your SQL syntax; check the`<br>`manual that corresponds to your MySQL server version for the right syntax`<br>`to use near`<br>`          'SELECT id`<br>`            FROM    supplier`<br>`            WHERE   city = 'Sydney')`<br>` ' at line 7` | ○ | ○ | ○ | ◐ | ○ |
| NuoDB | `Error 42000: syntax error on line 7`<br>`SELECT id`<br>`^ expected parenthesis got SELECT` | ● | ○ | ◐ | ● | ◐ |
| VoltDB | `SQL error while compiling query: SQL Syntax error in "SELECT name, manager_id`<br>`FROM   project`<br>`WHERE  id IN`<br>`        (SELECT project_id`<br>`        FROM    delivery`<br>`        WHERE   supplier_id IN`<br>`            SELECT id`<br>`            FROM    supplier`<br>`            WHERE   city = 'Sydney')`<br>`        );" unexpected token: SELECT`<br>`            required: (` | ● | ○ | ◐ | ◐ | ◐ |

Fig. 32. Error messages and coding for test T16