Empirical Software Engineering (2025) 30:136
https://doi.org/10.1007/s10664-025-10695-1

# Enhanced SQL error messages facilitate faster error fixing

Toni Taipalus[1] · Hilkka Grahn[2] · Antti Knutas[3]

Accepted: 23 June 2025

**Abstract**
Error messages are one of the primary ways software developers communicate with database management systems in SQL query writing tasks. Even though reading and interpreting error messages is a significant part of a software developer's work, the error messages of SQL compilers have received criticism in terms of readability and their perceived detrimental effects on user experience. Consequently, redesigned SQL error messages have also been proposed to tackle the problems in current error messages. In this study, we examine the effects of enhanced error messages on query writing from several perspectives. The results indicate that when compared to PostgreSQL error messages, the enhanced error messages facilitate faster error fixing, as well as perceived benefits in error finding and error recovery confidence. Our results are applicable in industry, where development time is of significant importance, as well as in educational contexts, where user experience, user confidence, and perceived support in learning play an important role.

**Keywords** SQL · Compiler · Error message · Database management system · Software development · Error recovery

Communicated by: Scott Fleming.

✉ Toni Taipalus
toni.taipalus@tuni.fi

Hilkka Grahn
hilkka.grahn@jyu.fi

Antti Knutas
antti.knutas@lut.fi

1 Faculty of Information Technology and Communication Sciences, Tampere University, Kalevantie 4, Tampere 33014, Finland

2 Faculty of Information Technology, University of Jyväskylä, Mattilanniemi 2, Jyväskylä 40014, Finland

3 Department of Software Engineering, LUT University, Yliopistonkatu 34, Lappeenranta 53850, Finland

## 1 Introduction

Interpreting error messages is a significant aspect of the work undertaken by software developers (Barik et al. 2017). Various studies have emphasized the essential role of the development environment in enhancing the user experience (Donahue 2001), reducing debugging time (Pane et al. 2002), bolstering user confidence (Taipalus and Grahn 2023b), and in learning contexts, fostering student motivation (Cauley and McMillan 2010). Despite the practical importance of this matter, error messages continue to be a source of confusion and frustration (Becker et al. 2019; Kummerfeld and Kay 2003). The state of error messages, particularly those associated with programming language compiler errors, cannot be solely attributed to a lack of scientific research, since the intersection of software engineering and human-computer interaction has seen several studies on enhancing error messages (Karvelas et al. 2020). Furthermore, a recent comprehensive literature review, summarizing numerous studies on programming language compiler error messages (Becker et al. 2019), concluded that the impact of improving compiler error messages on error recovery remains inconclusive.

Despite its age, Structured Query Language (SQL) is still the *lingua franca* of relational database management systems (RDBMS). SQL is taught effectively in all higher education information technology curricula such as computer science (ACM/IEEE 2013), software engineering (ACM 2015), and information systems (ACM/AIS 2020), and remains a highly sought skill in the software industry (Cass 2022). Despite mature RDBMS implementations and wide and in-depth education and training opportunities, SQL remains a challenging language to master, arguably due to the discrepancies between the theoretical foundations (Codd 1970), the SQL Standard (ISO/IEC 2016a, b), and the different RDBMS implementations of SQL (Taipalus 2023b). Similarly to many programming language compiler error messages, error messages of SQL compilers of different RDBMSs leave much to be desired in terms of taking into account human-computer interaction guidelines and usability aspects (Taipalus et al. 2021). Furthermore, compared to the lively research field around programming language compiler error messages, SQL error messages have remained on the sidelines. Due to the declarative and domain-specific nature of SQL, many of the findings in programming language compiler error messages are not applicable to SQL. In fact, it has been implied that SQL-related development tasks are more complex and more time-consuming than other development tasks (Costa et al. 2023), perhaps even more so when SQL is used in conjunction with host languages that have immature or rapidly changing ecosystems, onerous package management, and weak data-I/O integration (Rennels and Chasins 2023).

This study builds upon a set of recent studies that compared and strove to enhance SQL error messages. One of these studies established that when compared to the error messages of several other RDBMSs, the error messages of PostgreSQL are at least marginally more effective for error fixing (Taipalus et al. 2021). Another study redesigned SQL error messages based on qualitative insights from query formulation novices (Taipalus and Grahn 2023a), yet did not test the effectiveness of the redesigned error messages. In this study, we utilize three student cohorts of query formulation novices ($N = 509$) in order to compare the error messages of PostgreSQL to the redesigned SQL error messages presented earlier by Taipalus and Grahn (2023a). We measure error message effectiveness with both subjective metrics of perceived user confidence and perceived error message usefulness, as well as objective metrics of error fixing success and time taken to successfully fix errors. The results indicate that enhanced error messages are perceived to help in finding the erroneous part of the query, they increase error recovery confidence, and they result in approximately 14% faster error fixing. However, it was inconclusive whether the enhanced error messages were *perceived* to help in fixing errors, and if the enhanced error messages actually helped in successfully fixing the errors.

The rest of this study is structured as follows. In the next section, we describe earlier studies on the nature of SQL errors, proposed guidelines for error message design across different computer languages, and the effects of enhanced or redesigned error messages. In Section 3, we detail our research setting, study participants and ethical considerations, data collection and analysis, and present our hypotheses. Section 4 and Appendix A show our results. We discuss our findings, practical implications of the results, critical evaluation of the choices regarding our research setting, and future research opportunities in Section 5. Section 6 concludes the study.

## 2 Background

### 2.1 Errors in SQL Query Formulation

Errors in SQL query formulation have become a subject of increasing scientific attention, particularly within the domain of computing education research (Taipalus and Seppänen 2020). Current investigations (Taipalus et al. 2018; Migler and Dekhtyar 2020; Wang et al. 2024) categorize user errors in SQL queries related to data retrieval into *syntax errors*, identified by the DBMS and leading to a syntax error message, *semantic errors*, typically unnoticed by the DBMS, resulting in inaccurate data in the result table irrespective of the intended query (e.g., a syntactically correct query which will always return an empty result table) Brass and Goldberg (2006), *logical errors*, undetected by the DBMS, resulting in inaccurate data in the result table when considering the query's intent, and *complications*, which may be identified by the DBMS and do not result in incorrect data in the result table but needlessly complicate the query. Syntax errors have been identified as the most prevalent, especially in the formulation of queries by novices (Taipalus and Perälä 2019; Ahadi et al. 2016a). Common syntax errors, depending on the study, arise from factors such as references to undefined tables and columns (Smelcer 1995), grouping issues (Reisner et al. 1975; Reisner 1977), omission of mandatory clauses (Smelcer 1995), data type mismatches (Ahadi et al. 2016a), and misspellings (Welty 1985). Determining the most common SQL syntax errors poses challenges because, with a few exceptions, studies categorize syntax errors based on the DBMS used, rendering results incomparable across studies with different DBMSs, as different DBMSs categorize and identify errors in different ways (Taipalus 2023b). Moreover, studies have shown that distinct SQL concepts, such as grouping, joins, or ordering, invite different types of syntax errors (Taipalus and Perälä 2019), naturally making errors common for queries involving table joins uncommon for queries involving grouping and a single table.

Despite being more common than other error types, syntax errors are arguably easier to fix, possibly because they are caught by the DBMS (Taipalus and Perälä 2019; Ahadi et al. 2016b). This intuitively makes sense, as a syntax error interrupts the query execution, preventing the query writer from obtaining a result table, whereas a logical error does not. A query with a logical error may produce a result table with seemingly correct data that does not align with the query writer's intent. Various factors contributing to query formulation errors have been proposed, encompassing human factors such as cognitive load theory (Smelcer 1995), misconceptions about the language or generalizations (Miedema et al. 2021), self-induced complexity (Miedema et al. 2022b), procedural fixedness (Taipalus 2020), and simple sloppiness (Miedema et al. 2022a; Smelcer 1995). Since syntax errors are typically the only type of error that results in an error message, research on error messages has focused on syntax errors. It has been up to the query writer or writers to spot and fix semantic and logical errors, although there have been efforts towards identifying semantic errors in SQL queries (Taipalus 2023a), as well as automatically repairing some syntax and logical errors (Presler-Marshall et al. 2021; Pasupuleti et al. 2023).

## 2.2 Error Message Design Guidelines

Numerous research endeavors underscore the significance of compiler error messages in the fields of computing education and software development within the context of programming languages (Becker et al. 2016, 2019; Wrenn and Krishnamurthi 2017). However, these studies assert that current compiler error messages suffer from ineffectiveness, attributed to various factors such as confusing and unconstructive phrasing (Becker et al. 2016, 2019), leading users to experience feelings of inadequacy and anxiety when confronted with such messages (Shneiderman et al. 2016). The significance of error messages in the feedback loop between the end-user and the compiler is even more pivotal for novice users, who arguably need more support in error recovery. Such novices may either be novices in computer languages overall or experts in changing from one language to another. Since the quality of error messages significantly influences the overall user experience, improvements in error messages extend benefits to professionals as well (Kantorowitz and Laor 1986). In the context of SQL compilers, various DBMSs offer various error message elements such as identifying error position (e.g., PostgreSQL), providing hints (e.g., VoltDB), identifying several syntax errors within a single query (e.g., SQL Server), and providing error codes (e.g., Oracle Database) (Taipalus and Grahn 2023a).

Several scholars and practitioners have proposed guidelines for enhancing and designing both system messages and error messages. Between 1982 and 2023, studies have proposed that different messages should have qualities such as being brief (Shneiderman 1982; Traver 2010) to reduce cognitive load (Becker et al. 2019), being specific (Shneiderman 1982), showing hints (Becker et al. 2019; Taipalus and Grahn 2023a), and removing unnecessary elements such as error codes and environmental variables (Taipalus and Grahn 2023a). Some of these guidelines are based on expert opinion, others on empirical evidence. Additionally, some of these guidelines concern general system messages, some programming language compiler error messages, and some SQL error messages. Establishing guidelines for error message design proves challenging due to the diversity of the tasks involved. Even when some guidelines are derived from empirical data, conflicts may arise between these guidelines or within the guidelines themselves (Dix et al. 2005). For instance, Shneiderman's (1982) recommendations for *constructiveness* and *brevity* could be perceived as contradictory. The suggestion has been made that adhering strictly to different design guidelines is not necessary. Rather, they serve the purpose of constraining the design space to steer clear of creating unusable systems (Dix et al. 2005). In essence, design guidelines offer a perspective for designers to consider crucial aspects of error messages.

## 2.3 Effects of Enhanced Error Messages

Empirical investigations have demonstrated that improved error messages in programming languages can lead to a decrease in the occurrence of errors (e.g., Becker 2016), and end-users may express a preference for such enhanced error messages (e.g., Barik et al. 2018; Thiselton and Treude 2019). However, several studies have also presented inconclusive or negative outcomes (e.g., Prather et al. 2017; Nienaltowski et al. 2008; Pettit et al. 2017). Conversely, certain approaches, such as adjusting error message spacing, utilizing colors (Dong and Khandwala 2019), and implementing syntax highlighting (Hannebauer et al. 2018), have demonstrated the potential to enhance the effectiveness of error messages. However, the supporting evidence for these techniques has at times been anecdotal (Becker et al. 2019) and, in some cases, contentious (Sarkar 2015; Denny et al. 2014; Pettit et al. 2017; Zhou et al. 2021).

To the best of our knowledge, we are not aware of a single research effort into the *effects of redesigned or enhanced SQL syntax error messages*. Figure 1 summarizes the most relevant set of prior scientific efforts leading up to this study. A study on DBMS-independent SQL error categorization established a set of most common SQL syntax errors (Taipalus et al. 2018). Subsequently, the sixteen most common syntax errors were selected and queries that exhibit these syntax errors were formulated to form an SQL error message test suite (Taipalus et al. 2021). This test suite was used to compare the effectiveness of traditional RDBMS error messages (Taipalus et al. 2021), as well as error messages of novel NewSQL systems (Taipalus and Grahn 2023b). The results indicated differences in the effectiveness of different DBMSs, most importantly that PostgreSQL's error messages are somewhat more effective than those of other traditional RDBMSs. Further, it was shown that general system message qualities poorly explain the effectiveness of SQL error messages, i.e., general system message design guidelines fail to particularize to SQL (Taipalus and Grahn 2023a). Finally, SQL error message design guidelines were formulated based on empirical evidence, and applications of these guidelines were shown in the context of the SQL error message test suite (Taipalus and Grahn 2023a). In this study, we compare the effectiveness of previously redesigned SQL error messages to those of PostgreSQL.

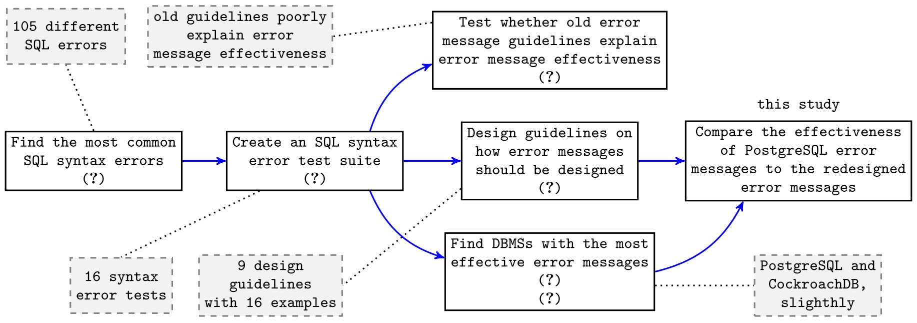

105 different SQL errors
old guidelines poorly explain error message effectiveness
Test whether old error message guidelines explain error message effectiveness (?)
this study
Find the most common SQL syntax errors (?)
Create an SQL syntax error test suite (?)
Design guidelines on how error messages should be designed (?)
Compare the effectiveness of PostgreSQL error messages to the redesigned error messages
Find DBMSs with the most effective error messages (?) (?)
16 syntax error tests
9 design guidelines with 16 examples
PostgreSQL and CockroachDB, slightly

**Fig. 1** The most relevant prior scientific studies affecting the research setting in this study; research goals are presented in rectangles with solid lines, and results in rectangles with dashed lines

## 3 Research Setting

### 3.1 Study Participants

The study participants were recruited from a university database course. The course follows AIS/ACM curriculum guidelines for an undergraduate database course (ACM/AIS 2020) including theory and practical exercises on conceptual modeling, SQL, and database normalization. The course has no particular prerequisite courses, yet it is typically taken by second-year students. In the course, SQL was covered through weekly lectures complemented with lab exercises. SQL concepts relevant to this study included basic statement structures, expressions such as classical comparison operations, `LIKE`, `BETWEEN`, `IS` and `IN` in both `WHERE` and `HAVING` clauses, joins using correlated and non-correlated subqueries and the `JOIN` clause, aggregate functions and grouping, as well as ordering. More advanced concepts such as window functions or common table expressions were not covered. The course strives to teach ANSI/ISO SQL without focusing on a particular DBMS implementation. In the lectures, SQL was covered through theory and practical work regarding reading and writing queries, and the exercises focused on writing SQL queries with an online learning environment that uses SQLite as the underlying DBMS.

After the course had covered SQL, the course students were given the opportunity to earn course credit by filling out a survey. Additionally, the students were given the opportunity to opt in on giving their survey answers to be used in this study. Opting in had no positive or negative effects, and a full privacy statement was presented before choosing whether to participate. The data were collected from three groups of participants, i.e., three student cohorts, henceforth referred to as cohorts A ($n$ = 190), B ($n$ = 152) and C ($n$ = 167). Out of the total of 592 students who answered the survey, 509 (86%) chose to participate. Since the research was based on informed consent, the physical integrity of the participants was not involved, the participants were not minors, the research did not expose the participants to risk of strong stimuli, mental harm, or safety, and no separate ethical committee approval was required as per our institutions' guidelines.

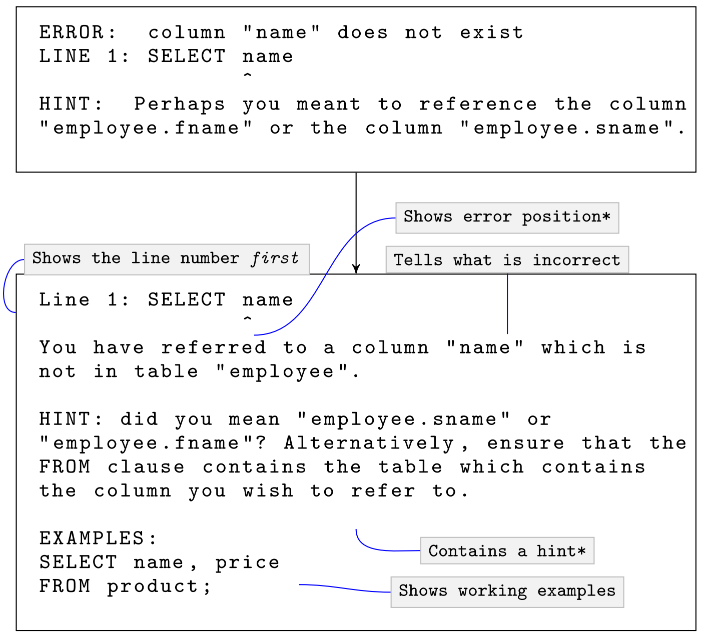
### 3.2 Data Collection

For data collection (i.e., the survey mentioned in the previous section), we utilized a previously formulated SQL test suite (cf. Appendix *S3* in Taipalus et al. (2021)[1]). The test suite consists of sixteen tests, each of which contains a schema diagram of the underlying database, a task expressed in natural language (e.g., "*find all red cars*"), a corresponding SQL query with a syntax error, an error message related to the syntax error, a text-field in which the participant is asked to write the fixed query, and three five-point Likert questions asking how useful the error message was in finding the error, fixing the query, and how confident the error message made the participant feel in error recovery. Each of the erroneous queries in the test suite is related to different syntax errors, e.g., "*using nonstandard operators*" or "*using an aggregate function outside* `SELECT` *or* `HAVING`", which have been found common in an earlier study (Taipalus et al. 2018). The survey instrument also records the duration of attempting to fix a query.

[1] https://ars.els-cdn.com/content/image/1-s2.0-S016412122100131X-mmc3.pdf

For cohort A, we used the test suite that contains error messages produced by PostgreSQL 12.1. For cohort B, we modified the test suite so that instead of PostgreSQL error messages, the test suite contains redesigned error messages. The redesigned error messages were adapted from a previous study (Taipalus and Grahn 2023a) which, according to an empirical mixed-methods analysis, offers nine guidelines for designing SQL error messages, e.g., stating why the error occurs, providing hints for fixing the error, placing most important information first, and removing unnecessary information from the error message. That study also gives sixteen examples of applying the design guidelines to SQL syntax error messages. One example of error message redesign is shown in Fig. 2, which shows how a PostgreSQL error message can be redesigned according to the guidelines: the word ERROR is removed; most important information (i.e., error position) is given first; the error message states in plain English what is incorrect with the query; the error message gives a hint, speculating what the user might want to accomplish; and the error message shows a working example of query concepts identified erroneous. All the redesigned error messages are reported in Appendix B. We utilize these sixteen examples in this study.

Cohort C was randomly divided into two groups, one of which took the same test suite as cohort A, and the other the same test suite as cohort B. In the subsequent analyses, we compare the results of cohort A to cohort B, and the results between the two groups within cohort C. In addition to the sixteen tests, the test suite contains four control questions which are the same for all participants, regardless of cohort. Control questions were used to control the possible skill differences between compared cohorts and groups. Figure 3 summarizes



```
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".
```

```
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;
```

**Fig. 2** An example of an original PostgreSQL error message (above) and a redesigned error message (below); notes show which parts of the error message have been redesigned or repositioned according to the SQL error message redesign guidelines; notes marked with an asterisk pertain to error message elements which were also present in the original error message

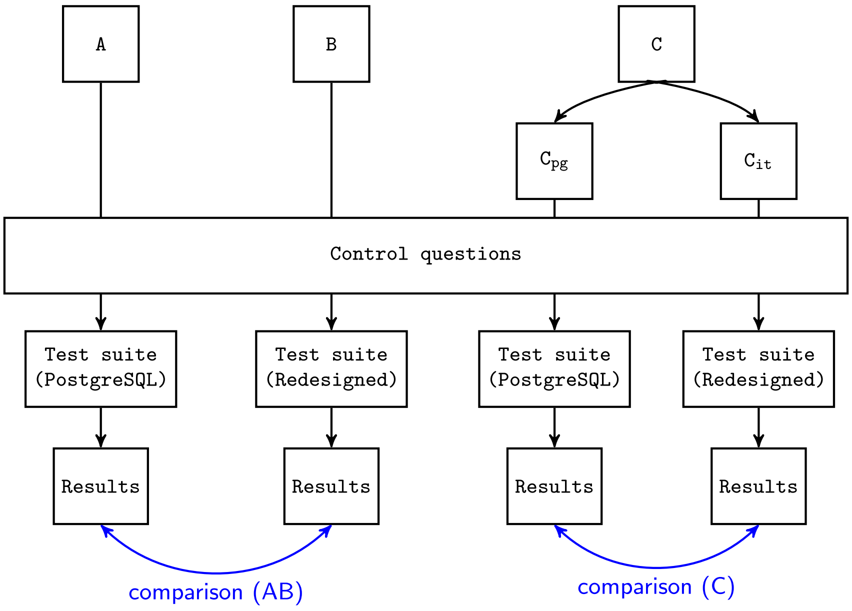


**Fig. 3** Data collection process across the three cohorts (A, B, C); cohort A answered the test suite with PostgreSQL error messages, cohort B answered the test suite with redesigned error messages, and cohort C was divided into two groups, one of which answered the same test suite as cohort A, and the other the same test suite as cohort B

the data collection across the three cohorts. Appendix B contains the error messages of PostgreSQL as well as the redesigned error messages.

## 3.3 Data Preparation

After data collection, we ran the fixed queries the participants had submitted through PostgreSQL to determine which answers were syntactically correct. That is PostgreSQL decided whether a participant's query was syntactically correct or incorrect. Next, among the syntactically correct queries, we manually determined whether the queries were also logically correct equivalents to their corresponding tasks (e.g. *list all red cars*). This posed some subjectivity, as some participants merely fixed a typographic error in the erroneous query, while others chose to almost completely rewrite a query, fixing the syntax error in the process as well. We determined that a logically correct query would return a correct result table. If ordering was not required the order of rows in the correct result table was not relevant. Answers that were both syntactically and logically correct were considered correct, and answers that contained at least one error (regardless of the nature of the error) were considered incorrect. Regarding erroneous query fixing time, we omitted answers in cases where a participant used less than 15 seconds for query fixing, as well as answers in which a participant used more than 1,200 seconds (20 minutes) for fixing a query. These cut-off points were selected based on observations of durations, and chosen to omit cases where participants merely sped through the pages in the survey instrument, or left a page open. Out of the 2,672 observations (167 participants × 16 tests) for cohort C, this resulted in omitting 17 observations. An $\alpha$-level of.05 was chosen for all tests.

### 3.4 Hypotheses

We formulated five hypotheses based on the research setting. All hypotheses test whether the query-fixing process with redesigned error messages is in some way different from the query-fixing process with PostgreSQL's error messages. Hypotheses $H_1$ through $H_4$ are tested with three cohorts (A, B, C), while hypothesis $H_5$ is merely tested with one cohort (C). The first three hypotheses are based on subjective perceptions, while the last two are based on objective metrics.

$H_1$: redesigned error messages are perceived to make finding syntax errors easier.
$H_2$: redesigned error messages are perceived to make fixing syntax errors easier.
$H_3$: redesigned error messages increase user confidence in syntax error recovery.
$H_4$: redesigned error messages make syntax error fixing more successful.
$H_5$: redesigned error messages make syntax error fixing faster.

## 4 Results

### 4.1 Redesigned Error Messages Were Perceived to Help in Finding the Errors

We conducted a Mann-Whitney U test to examine potential differences in participants' subjective evaluations of error message usefulness for error finding, fixing, and recovery confidence between two designs of error messages: PostgreSQL ($n$ = 190 in cohort A, $n$ = 82 in cohort C) and redesigned messages ($n$ = 152 in cohort B, $n$ = 85 in cohort C) in all 16 tests. Upon visual examination, it was observed that the distributions of the scores were not sufficiently similar. Consequently, mean ranks were utilized instead of medians for reporting. The results for all hypotheses are summarized in Table 1.

The results support $H_1$ for all cohorts: for finding the error, error message usefulness scores were statistically significantly greater in the redesigned messages (mean rank 3147.54

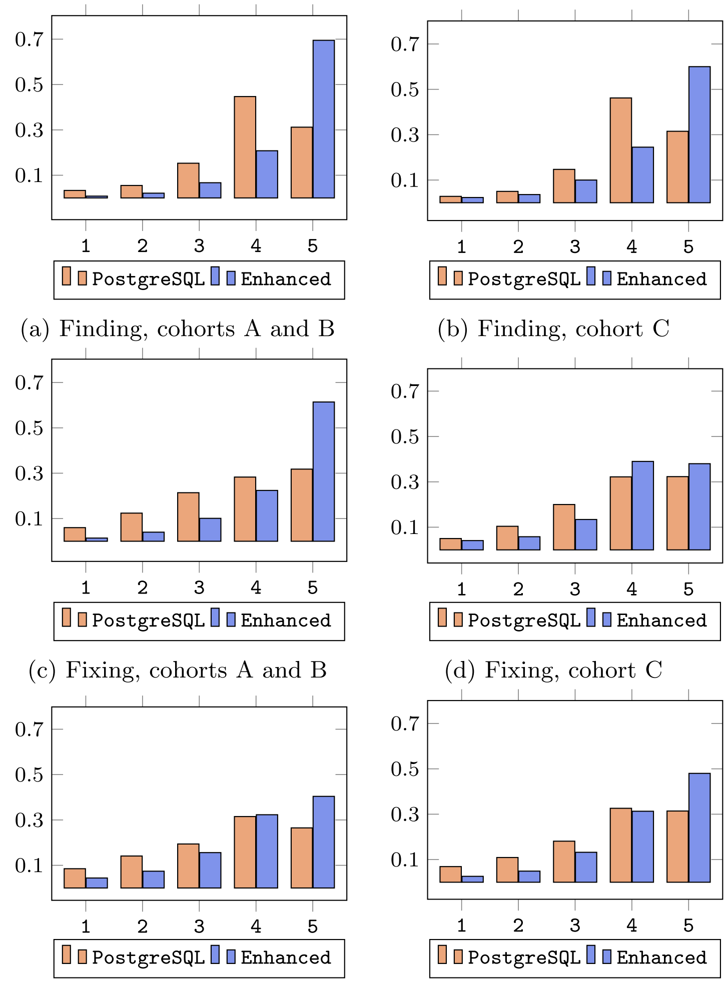
**Table 1** The results indicate that the redesigned error messages are perceived to be more useful in finding the error, facilitate error recovery confidence, and result in faster error fixing; however, we failed to reject the null hypotheses for perceived usefulness in error fixing and actually fixing the error correctly with cohort C; effect sizes are reported for statistically significant results as Cohen's $d$ for $H_1$, $H_2$, $H_3$ and $H_5$, and as Cramér's $V$ for $H_4$

| Hypothesis | Supported | | Test statistic | | Effect size | |
|---|---|---|---|---|---|---|
| | AB | C | AB | C | AB | C |
| $H_1$ error finding | yes | yes | $U = 2696997, z = -19.167, p < .001$ | $U = 796385, z = -4.749, p < .001$ | .518 | .117 |
| $H_2$ error fixing | yes | no | $U = 2405552, z = -23.612, p < .001$ | $U = 848203, z = -1.754, p = .079$ | .668 | |
| $H_3$ recovery confidence | yes | yes | $U = 2955102, z = -13.274, p < .001$ | $U = 842701, z = -2.074, p = .038$ | .362 | .059 |
| $H_4$ query fixed correctly | yes | no | $p < .001$ | $p = .239$ | .098 | |
| $H_5$ time taken | *(n/a)* | yes | *(n/a)* | $t(2285) = 4.053, p < .001$ | | .170 |

for AB, and 1390.39 for C) compared to the PostgreSQL messages (2407.67 for AB, and 1263.16 for C). The Likert distributions are presented in Fig. 4a and b.

## 4.2 It was Inconclusive Whether the Redesigned Error Messages were Perceived to Help in Fixing the Errors

The results support $H_2$ for the AB cohort comparison: for fixing the error, error message usefulness scores were statistically significantly greater in the redesigned messages (3267.38) compared to the PostgreSQL messages (2311.80). However, the results fail to reject the null hypothesis for the comparison between the groups in cohort C. The Likert distributions are presented in Fig. 4c and d.



(a) Finding, cohorts A and B (b) Finding, cohort C

(c) Fixing, cohorts A and B (d) Fixing, cohort C

(e) Confidence, cohorts A and B (f) Confidence, cohort C

**Fig. 4** Comparison of relative Likert scale distributions for PostgreSQL and enhanced error messages for the subjective metrics of *error finding*, *error fixing*, and *recovery confidence*

### 4.3 Redesigned Error Messages were Perceived to Increase Error Recovery Confidence

The results support $H_3$ for all cohorts: for error recovery confidence, error message usefulness scores were statistically significantly greater in the redesigned messages (3041.41 for AB, and 1356.16 for C) compared to the PostgreSQL messages (2492.57 for AB, and 1298.74 for C). The Likert distributions are presented in Fig. 4e and f.

### 4.4 It was Inconclusive Whether the Redesigned Error Messages Helped in Successfully Fixing the Errors

In the comparison between cohorts A and B, 77.1% of the erroneous queries in the PostgreSQL group were fixed correctly ($H_4$), and in the redesigned error messages group, 84.9% of the erroneous queries were fixed correctly. Tested with Pearson's chi-squared, the difference was statistically significant ($p$ <.001). In the comparison between the groups in cohort C, 85.3% of the erroneous queries in the PostgreSQL group were fixed correctly, and in the redesigned error messages group, 86.9% of the erroneous queries were fixed correctly. Tested with Pearson's chi-squared, the difference was not statistically significant ($p$ =.239).

### 4.5 Redesigned Error Messages Resulted in Faster Error Fixing

We used an independent samples t-test to determine if there were differences in the time taken to fix the queries. The results support $H_5$, as it was found that for correctly fixed queries, participants presented with PostgreSQL error messages used statistically significantly more time ($M$ = 225.90 s ± 212.302 s) to fix the error than the participants presented with the redesigned error messages ($M$ = 194.18 s ± 159.546 s), $t(2285) = 4.053$, $p$ <.001. The time taken to fix the query was only collected from cohort C.

### 4.6 Test-by-test Results

To account for the number of tests in the test-by-test comparisons, we adjusted the alpha level ($\alpha$-level =.05) with Bonferroni correction. An alpha-level of $\alpha = \frac{0.05}{16}$ =.003125 was selected for the test-by-test comparisons. Table 2 summarizes the results test-by-test. The test statistics are reported in Appendix A due to their length.

## 5 Discussion

### 5.1 Discussion of the Results

It has been suggested before that measuring error message effectiveness merely by whether error recovery is successful may not be sufficient (Taipalus and Grahn 2023b). Because the syntax error message test suites measure also *perceived* or subjective aspects, we decided to include them in addition to measuring query fixing success. The results show quite uniformly that error finding ($H_1$) was perceived to improve with the enhanced error messages when compared to those of PostgreSQL. The results were more uniform in the inter-cohort

**Table 2** Test-by-test results for each dependent variable (error finding, error fixing, error recovery confidence, success in fixing the query correctly, and time taken to fix the query) for by participant cohort (A, B, C); dark blue cells 🟦 represent test results in which the redesigned error messages were more effective with a statistically significant difference (i.e., supporting the corresponding hypothesis), dark orange cells 🟧 (none present) represent test results in which the PostgreSQL error messages were more effective with a statistically significant difference; light blue and light orange cells represent respective results with no statistically significant difference; the light-colored cells are for descriptive purposes and should not be interpreted as differences

| | $H_1$ Finding | | $H_2$ Fixing | | $H_3$ Confidence | | $H_4$ Correct | | $H_5$ Time |
|---|---|---|---|---|---|---|---|---|---|
| | AB | C | AB | C | AB | C | AB | C | C |
| T01 | 🟦 | 🟦 | 🟦 | 🔹 | 🟦 | 🔹 | 🔹 | 🔹 | 🔹 |
| T02 | 🟦 | 🔸 | 🟦 | 🔸 | 🟦 | 🔸 | 🔹 | 🔹 | 🔹 |
| T03 | 🟦 | 🔹 | 🟦 | 🔹 | 🟦 | 🔹 | 🔹 | 🔸 | 🔸 |
| T04 | 🟦 | 🟦 | 🟦 | 🟦 | 🟦 | 🔹 | 🟦 | 🔹 | 🔸 |
| T05 | 🟦 | 🔸 | 🟦 | 🔸 | 🟦 | 🔸 | 🟦 | 🔹 | 🟦 |
| T06 | 🟦 | 🔹 | 🟦 | 🔹 | 🔹 | 🔸 | 🔹 | 🔹 | 🔸 |
| T07 | 🔸 | 🔹 | 🔸 | 🔹 | 🔸 | 🟦 | 🔹 | 🔹 | 🔸 |
| T08 | 🟦 | 🔹 | 🟦 | 🔹 | 🟦 | 🔸 | 🟦 | 🔹 | 🔹 |
| T09 | 🟦 | 🔹 | 🟦 | 🔸 | 🟦 | 🔹 | 🟦 | 🔹 | 🟦 |
| T10 | 🟦 | 🔹 | 🟦 | 🔸 | 🟦 | 🔸 | 🟦 | 🔹 | 🔸 |
| T11 | 🔹 | 🔹 | 🔹 | 🔹 | 🟦 | 🟦 | 🔹 | 🔸 | 🔸 |
| T12 | 🟦 | 🔸 | 🟦 | 🔸 | 🟦 | 🔸 | 🟦 | 🔸 | 🔸 |
| T13 | 🟦 | 🔹 | 🟦 | 🔸 | 🟦 | 🔸 | 🟦 | 🔹 | 🔹 |
| T14 | 🟦 | 🔹 | 🟦 | 🔹 | 🔹 | 🔹 | 🔸 | 🔸 | 🔹 |
| T15 | 🔹 | 🔹 | 🟦 | 🔹 | 🔹 | 🔹 | 🔹 | 🔹 | 🔸 |
| T16 | 🟦 | 🔸 | 🟦 | 🔸 | 🔹 | 🔹 | 🔹 | 🔸 | 🔹 |

comparisons (AB) than in the intra-cohort comparisons (C). Within the inter-cohort comparisons, thirteen of the sixteen tests showed that the enhanced error messages were perceived to help in error finding more than those of PostgreSQL. PostgreSQL error messages were not seen as more helpful in this regard in any of the tests. In the rest of the tests, the results were not statistically significant. Within the intra-cohort comparisons, the enhanced error messages were perceived better in two of the tests, and PostgreSQL error messages in none of the tests. In most of the tests, the differences were not statistically significant. Overall, these results are in line with previous error message effectiveness comparisons regarding error finding (Taipalus et al. 2021; Taipalus and Grahn 2023b). The test-by-test results show no clear patterns that would match previously studied PostgreSQL error message effectiveness comparisons, which could shed some light on why the results from these tests were statistically significant (Taipalus et al. 2021). Additionally, scientific evidence on what is perceived as helpful in SQL error messages does not always align with what has been objectively measured to be helpful. For example, a quantitative study found that ordering the most critical information first in an SQL error message predicts error-fixing success more accurately than any other message characteristic Taipalus and Grahn (2024), yet one qualitative study showed that novices perceived that SQL error messages that show the erroneous position are the most helpful in error recovery Taipalus et al. (2025).

For perceived usefulness in error fixing ($H_2$), the results are rather similar to $H_1$. Within the inter-cohort comparisons, the enhanced error messages were perceived as more helpful in fixing the error in fourteen of the sixteen tests. PostgreSQL error messages were not seen as more helpful in this regard. Within the intra-cohort comparisons, the enhanced error messages were perceived as more helpful in one of the tests, while other differences were not statistically significant. Overall, as stated in Section 4.2, this means that there is not enough evidence to claim that the enhanced error messages are perceived to help in fixing SQL syntax errors. These mixed findings regarding the inter and intra-cohort comparisons somewhat contest previous observations that the contents of the error message affect perceived usefulness in error fixing (Taipalus et al. 2021; Taipalus and Grahn 2023b).

Again, for error recovery confidence ($H_3$), the results reflect those of $H_1$ and $H_2$. Within the inter-cohort comparisons, the enhanced error messages were perceived to help in error recovery confidence in eleven of the sixteen tests. Other comparisons were not statistically significantly different. Within the intra-cohort comparisons, the enhanced error messages were more helpful in two of the tests. PostgreSQL error messages were not perceived to help with error recovery confidence in either of the comparisons. Overall, the evidence suggests that enhanced error messages help in error recovery confidence. Such effect has been observed in one prior study (Taipalus et al. 2021), but no statistically significant differences have been observed in another (Taipalus and Grahn 2023b).

For successful error fixing ($H_4$), the participants who were shown the enhanced error messages were more successful in fixing the syntax errors in seven of the sixteen tests within the inter-cohort comparisons. Within the intra-cohort comparison, the enhanced error messages facilitated successful error fixing in none of the tests. PostgreSQL error messages were not better in any of the tests in either comparison. Overall, there is not enough evidence to claim that the enhanced error messages facilitate more successful error fixing. This is somewhat in line with what has been previously observed, as PostgreSQL error messages were shown to be more effective than those of MySQL (yet with a small effect size), but not more effective than those of Oracle Database (Taipalus et al. 2021). Another study found that differences between SQL error messages have no effect on successful error fixing (Taipalus and Grahn 2023b).

The time taken to fix the syntax errors ($H_5$) was studied with only one cohort (C). This was due to the fact that this metric was only considered late in the data collection process. In two of the sixteen tests, the results show that the enhanced error messages facilitated faster (successful) syntax error fixing. None of the tests showed a significantly slower syntax error fixing with the enhanced error messages. Overall, the differences were statistically significant, showing that the participants who were shown PostgreSQL error messages used a median time of approximately 226 seconds to fix an erroneous query, while participants with enhanced error messages used approximately 194 seconds (approximately 14% decrease in time taken). Time taken to fix queries has not been studied before, yet in the context of programming languages, error message contents have been observed to affect error recovery time (Seo et al. 2014). This area of error recovery *speed* has been identified as a research dearth before (Becker et al. 2019).

Tests T05 and T09 were the ones with statistically significant differences regarding all hypotheses, in favor of the enhanced error messages. We did not collect qualitative data on *why* some error messages were more effective than others. However, perhaps the reason why the enhanced error message in test T05 (cf. Fig. 9) was effective is that PostgreSQL's error message is rather obscure, given that the error is in using `GROUP BY` instead of `ORDER BY`. In test T09 (cf. Fig. 13), PostgreSQL elects using more jargon than is typical in its error messages, which may explain some of the differences in perceived and actualized effectiveness.

## 5.2 Practical Implications

Overall, the findings highlight the importance of error message design in facilitating user understanding and error recovery. Understanding that enhanced error messages can assist novice query writers in identifying errors within queries better suggests that compiler developers should prioritize improving the clarity and specificity of error messages. The results imply that this may lead to a reduction in debugging time and potentially enhance the overall user experience of RDBMSs in query writing tasks. Additionally, enhanced error messages that facilitate error recovery confidence among novice users can lead to increased user satisfaction and retention. Therefore, RDBMS vendors may consider investing in improving error message systems as part of their product development efforts.

Higher education institutions, especially those offering courses in database management and SQL, should consider integrating the use of enhanced error messages into their curricula. Providing students with exposure to effective error messages can enhance their learning experience and improve their proficiency in query writing tasks. If the DBMS error messages support user experience in error finding and error recovery confidence, it is possible that students feel more encouraged to continue error fixing, which could in turn lead to more successful error fixing through more attempts to fix an error. Enhanced error messages do not seem to facilitate more successful error fixing, but they make error fixing faster among those novices who could fix the errors anyway. Additionally, students are potential future professionals in technical software development tasks, as well as in consultant roles. Their user experiences in RDBMS-related tasks arguably play a role when they choose which tools they prefer to use and recommend to others after their education.

Despite the practical implications listed above, it is worth noting that especially with error recovery confidence ($H_3$) and time taken to fix queries ($H_5$), the effect sizes are weak. Since there are no prior studies on enhanced SQL error messages for comparison with our results, it seems reasonable to argue that there is a possibility of similar findings in SQL error messages as in programming language compiler error messages. That is, more evidence is needed on SQL error messages to see whether results are uniform or conflicting. As discussed in Section 2.3, the effects of enhancing programming language error messages have been conflicted, and error recovery speed has rarely been a variable in such studies. In our experience, query languages operate in such a different environment compared to programming languages that comparisons between error message studies between these two realms are often problematic.

Looking at the rightmost column of Table 2, it seems reasonable to argue that if our test suite had contained more tests similar to T05 and T09, the results overall would have been more clear-cut. Therefore, much of the weight behind the practical implications of the results rests on which types of syntax errors learners and software developers actually need to fix. Despite the fact that the errors in the test suite are based on empirical evidence on the most prominent SQL syntax errors, it is unclear what the proportions of the types of errors are. As a hypothetical example, if the syntax error of T01 accounts for 50% of all encountered syntax errors, the test suite does not represent real situations, as the syntax error of T01 only accounts for approximately 6% ($\frac{1}{16}$) of syntax errors encountered in our dataset. The difficulty here is that the types of syntax errors are highly dependent on the type of query the query writer is writing, and the type of query is highly dependent on the business domain. Therefore, it is arguably not possible to define a test suite that represents all business domains.

In this paper, our contribution to the body of literature was the first evaluation of recommendations towards better SQL error message design through empiricism and objective metrics (as opposed to subjective perceptions of participants or experts). When comparing our findings and recommendations to the wider field of research on programming error messages, our results align with Karvelas et al. (2020) who recommend error messages with better support to novice programmers, and Barik et al. (2018) who found that error messages that build human-readable arguments are more helpful to novices.

## 5.3 Threats to Validity

We recognize that our research setting is prone to several uncontrolled variables. *Prior knowledge* concerning the error messages of a particular DBMS might accustom a participant to those error messages, mitigating the shortcomings or benefits of the messages, and emphasizing familiarization. We utilized SQLite in the database course from which the participants were recruited. This allowed practical exercises with SQL prior to the study, yet without using PostgreSQL or the redesigned error messages, which could have biased the results. Additionally, it is worth noting that by studying novices (who arguably benefit from error messages the most), we focus on novice experiences. This poses the open question of whether the results can be generalized to expert users. Additionally, we did not collect demographic data on the participants, and nuances between e.g., different prior courses taken, participant age or gender are outside the scope of this study.

As our study includes participants from three student cohorts, it is possible that *changes in the teaching environment* affect the results. To control this effect, the database course was taught by the same teacher using exactly the same teaching materials for all three cohorts, so any drift in teaching materials over time was eliminated. However, due to COVID-19, cohorts A and B studied in a remote teaching setting, while cohort C studied in a classroom. For this reason, we only compared remote-remote groups (A and B), and classroom-classroom groups (cohort C divided into two groups). That is, we did not compare remotely-studied participants with classroom-studied participants due to their obviously unequal study conditions. Additionally, cohort C was split at random into two subgroups, so that any unobserved time-oriented drift within that academic term cannot systematically favor one version of the messages.

It is possible that there are *skill differences* both between cohorts (A, B), and within a randomly divided, single cohort (C). In fact, the greater differences in inter-cohort comparisons across hypotheses ($H_1$) through ($H_4$) may suggest that there was an uncontrolled variable affecting the results that are not present in the intra-cohort comparisons. To control this, the test suite includes four control questions similar to the tests proper. Based on how well the participants fixed the syntax errors in the SQL queries in the four control questions, we tested for possible skill differences. Mann-Whitney U tests revealed that there were no statistically significant skill differences between cohorts A and B in error fixing success ($U = 14166.5$, $z = -0.399$, $p = .690$), or between the groups within cohort C in error fixing success ($U = 3171.5$, $z = -0.939$, $p = .348$) or in error fixing time ($U = 53588.0$, $z = -.743$, $p = .458$). This confirms that differences in the timing of data collection (A before the redesign, B after) or the mode of teaching (remote vs. classroom) did not translate into different SQL-debugging abilities before seeing the experimental error messages.

We studied the phenomenon of SQL error recovery in a relatively controlled way, which can be considered an *unnatural environment*. As described in Section 2.2, a query writer typically engages in a feedback loop with the compiler. This compiler can be interacted rather directly with a DBMS's command line interface, or via various integrated development environments cf. e.g.,, (Ahmed et al., 2022). Neither of these considerations were present in our study. The reason for this was to collect error recovery data in a controlled way, i.e., starting from a situation in which a set syntax error had already been committed. This allowed us to focus on certain syntax errors and ensured that the data collection yielded data that could be compared.

Finally, in terms of long-term learning outcomes, it remains unclear whether students benefit from enhanced error messages apart from the immediate positive effects. For example, does the increased support in the error messages foster *learning* SQL concepts more effectively, or does it merely help fix errors?

### 5.4 Future Directions

The field of SQL error messages, as with programming language compiler error messages, needs more scientific evidence. First, it still remains unclear *which elements* of SQL error messages facilitate the desired qualities of perceived helpfulness in error finding and fixing, error recovery confidence, as well as success and speed of error fixing. Despite the fact that helpful elements have been empirically collected and – with this study – their effectiveness tested, the results are not particularly clear-cut. This might also be due to the fact that the enhanced error messages were compared with those of PostgreSQL, instead of RDBMSs that have been shown to produce less effective error messages in prior studies. Second, the field of SQL error message studies needs *more evidence*: replication studies with *(i)* queries representing different syntax errors, *(ii)* participants with different educational and cultural backgrounds, *(iii)* more complex queries to emphasize the effects of complexity on error fixing success, time taken, and error finding, *(iv)* comparisons with error messages from less effective RDBMSs to understand whether the effects shown in this study are more clear-cut with RDBMSs such as Oracle Database or MySQL, *(v)* understanding on how environmental considerations such as the integrated development environments and their features such as syntax highlight and real-time syntax error discovery affect error recovery.

Arguably, the environments in the early 1980 s were different from today's hardware and infrastructure limitations, and considerations for SQL compiler design have probably changed. Unfortunately, despite the age of SQL, both older RDBMS as well as NewSQL vendors have been reluctant to redesign the error messages of SQL compilers. This might be due to several reasons, e.g., error messages are not seen as important features to dedicate vendor time to, or redesigning error messages would require significant alterations to the compiler, making the undertaking too challenging or time-consuming. Fortunately, the popularization of large language models may offer a solution for enhancing error messages without modifying compilers, or being DBMS-specific. Additionally, incorporating generative models into error recovery may relatively effortlessly incorporate personalized error messages, as redesigning error messages would rely simply on the prompts rather than SQL compiler restructuring. However, both performance and energy consumption may be crucial factors in incorporating large language models into DBMSs. We hope that our study provides evidence on the implications of redesigning SQL error messages using the approach introduced in a previous study Taipalus and Grahn (2023a). However, we note that the comparatively smaller sample sizes in cohort C may have reduced the statistical power of these tests. Therefore, more scientific evidence is needed to confirm these results.

## 6 Conclusion

Fixing errors is an important part of every software developer's work, and much of this work is done with the help of compilers, which output error messages to facilitate the process. However, several studies have shown error messages to be ineffective in various ways. In this study, we compared the SQL syntax error messages of PostgreSQL to redesigned error messages via five metrics. The results implied that *(i)* the redesigned error messages were perceived to be more helpful in finding the erroneous part of the queries, *(ii)* the redesigned error messages were perceived to increase error recovery confidence, and *(iii)* the redesigned error messages made fixing SQL syntax errors faster. In contrast, the results showed that *(iv)* the redesigned error messages were not perceived as more helpful in fixing syntax errors, and that *(v)* the redesigned error messages did not facilitate more successful error fixing. The findings of this study help in understanding what types of redesigned SQL error messages are effective, in which regard, and how much. These insights are applicable in redesigning SQL error messages for faster error fixing especially in the software industry, and for increased user experience especially among query writing novices such as students.

## Appendix A: Test Statistics

**Table 3** Test-by-test statistics for each dependent variable (error finding, error fixing, error recovery confidence, success in fixing the query correctly, and time taken to fix the query) by participant cohort (A, B, C)

| | $H_1$ Finding | | $H_2$ Fixing | | $H_3$ Confidence | | $H_4$ Correct | | $H_5$ Time |
|---|---|---|---|---|---|---|---|---|---|
| | AB | C | AB | C | AB | C | AB | C | C |
| T01 | $U = 9853.5$ | $U = 2210.0$ | $U = 8329.5$ | $U = 2704.5$ | $U = 10584.0$ | $U = 3201.0$ | $p = .068$ | $p = .061$ | $U = 2651.0$ |
| | $z = -5.534$ | $z = -4.233$ | $z = -7.052$ | $z = -2.531$ | $z = -4.412$ | $z = -0.936$ | | | $z = -0.705$ |
| | $p < .001$ | $p < .001$ | $p < .001$ | $p = .011$ | $p < .001$ | $p = .349$ | | | $p = .481$ |
| T02 | $U = 11858.0$ | $U = 3359.5$ | $U = 10234.5$ | $U = 3315.0$ | $U = 11490.0$ | $U = 3350.0$ | $p = .069$ | $p = .327$ | $U = 2517.5$ |
| | $z = -3.164$ | $z = -0.303$ | $z = -4.930$ | $z = -0.438$ | $z = -3.431$ | $z = -0.381$ | | | $z = -0.829$ |
| | $p = .002$ | $p = .762$ | $p < .001$ | $p = .661$ | $p < .001$ | $p = .703$ | | | $p = .407$ |
| T03 | $U = 10891.5$ | $U = 3214.0$ | $U = 8346.5$ | $U = 3142.5$ | $U = 9752.5$ | $U = 3239.0$ | $p = .053$ | $p = 1$ | $U = 2688.5$ |
| | $z = -4.519$ | $z = -0.656$ | $z = -7.249$ | $z = -0.887$ | $z = -5.460$ | $z = -0.644$ | | | $z = -0.996$ |
| | $p < .001$ | $p = .512$ | $p < .001$ | $p = .375$ | $p < .001$ | $p = .520$ | | | $p = .319$ |
| T04 | $U = 6962.5$ | $U = 1819.5$ | $U = 5447.0$ | $U = 2113.0$ | $U = 9088.0$ | $U = 3105.5$ | $p < .001$ | $p = .087$ | $U = 1915.5$ |
| | $z = -8.960$ | $z = -5.573$ | $z = -10.331$ | $z = -4.472$ | $z = -6.077$ | $z = -1.177$ | | | $z = -2.161$ |
| | $p < .001$ | $p < .001$ | $p < .001$ | $p < .001$ | $p < .001$ | $p = .239$ | | | $p = .031$ |
| T05 | $U = 7190.0$ | $U = 2990.5$ | $U = 4894.0$ | $U = 3302.5$ | $U = 9968.5$ | $U = 2698.0$ | $p < .001$ | $p = 1$ | $U = 1855.5$ |
| | $z = -8.717$ | $z = -1.422$ | $z = -11.072$ | $z = -0.341$ | $z = -5.173$ | $z = -2.432$ | | | $z = -3.597$ |
| | $p < .001$ | $p = .155$ | $p < .001$ | $p = .733$ | $p < .001$ | $p = .015$ | | | $p < .001$ |
| T06 | $U = 11255.5$ | $U = 2904.0$ | $U = 11804.5$ | $U = 2996.5$ | $U = 13448.0$ | $U = 3443.0$ | $p = .431$ | $p = .138$ | $U = 2621.5$ |
| | $z = -3.718$ | $z = -2.151$ | $z = -3.013$ | $z = -1.687$ | $z = -1.125$ | $z = -0.144$ | | | $z = -0.546$ |
| | $p < .001$ | $p = .031$ | $p = .003$ | $p = .092$ | $p = .261$ | $p = .886$ | | | $p = .585$ |
| T07 | $U = 12750.5$ | $U = 3081.5$ | $U = 12477.0$ | $U = 3425.5$ | $U = 14420.5$ | $U = 2468.5$ | $p = .121$ | $p = .163$ | $U = 3081.5$ |
| | $z = -2.518$ | $z = -1.562$ | $z = -2.648$ | $z = -0.208$ | $z = -0.023$ | $z = -3.512$ | | | $z = -0.250$ |
| | $p = .012$ | $p = .118$ | $p = .008$ | $p = .835$ | $p = .981$ | $p < .001$ | | | $p = .803$ |
| T08 | $U = 7872.0$ | $U = 3429.5$ | $U = 6892.5$ | $U = 3351.5$ | $U = 10955.0$ | $U = 2689.5$ | $p = .002$ | $p = .127$ | $U = 2744.0$ |
| | $z = -7.890$ | $z = -0.218$ | $z = -8.855$ | $z = -0.468$ | $z = -4.121$ | $z = -2.804$ | | | $z = -1.029$ |

**Table 3** (continued)

| | $H_1$ Finding | | $H_2$ Fixing | | $H_3$ Confidence | | $H_4$ Correct | | $H_5$ Time |
|---|---|---|---|---|---|---|---|---|---|
| | AB | C | AB | C | AB | C | AB | C | C |
| | $p<.001$ | $p=.827$ | $p<.001$ | $p=.640$ | $p<.001$ | $p=.005$ | | | $p=.303$ |
| T09 | $U=6969.5$ | $U=3094.0$ | $U=5493.5$ | $U=3118.5$ | $U=9018.5$ | $U=3151.0$ | $p=.001$ | $p=.367$ | $U=1542.0$ |
| | $z=-8.852$ | $z=-1.346$ | $z=-10.330$ | $z=-1.165$ | $z=-6.204$ | $z=-1.029$ | | | $z=-4.181$ |
| | $p<.001$ | $p=.178$ | $p<.001$ | $p=.244$ | $p<.001$ | $p=.303$ | | | $p<.001$ |
| T10 | $U=12022.5$ | $U=3388.0$ | $U=8293.5$ | $U=3442.0$ | $U=11324.0$ | $U=2743.5$ | $p=.002$ | $p=.433$ | $U=1208.0$ |
| | $z=-2.934$ | $z=-0.337$ | $z=-7.087$ | $z=-0.144$ | $z=-3.530$ | $z=-2.515$ | | | $z=-0.415$ |
| | $p=.003$ | $p=.736$ | $p<.001$ | $p=.885$ | $p<.001$ | $p=.012$ | | | $p=.678$ |
| T11 | $U=13017.5$ | $U=2950.5$ | $U=12343.0$ | $U=2964.0$ | $U=10792.0$ | $U=2495.5$ | $p=.883$ | $p=.147$ | $U=2389.5$ |
| | $z=-1.757$ | $z=-1.441$ | $z=-2.503$ | $z=-1.381$ | $z=-4.116$ | $z=-2.971$ | | | $z=-0.943$ |
| | $p=.079$ | $p=.150$ | $p=.012$ | $p=.167$ | $p<.001$ | $p=.003$ | | | $p=.346$ |
| T12 | $U=11055.0$ | $U=2721.0$ | $U=9808.0$ | $U=3012.0$ | $U=9936.0$ | $U=3215.0$ | $p=.026$ | $p=1$ | $U=2851.0$ |
| | $z=-4.555$ | $z=-2.698$ | $z=-5.755$ | $z=-1.496$ | $z=-5.341$ | $z=-0.783$ | | | $z=-1.061$ |
| | $p<.001$ | $p=.007$ | $p<.001$ | $p=.135$ | $p<.001$ | $p=.434$ | | | $p=.289$ |
| T13 | $U=10188.5$ | $U=3217.5$ | $U=8055.5$ | $U=3307.5$ | $U=11733.5$ | $U=3434.5$ | $p<.001$ | $p=.016$ | $U=2331.5$ |
| | $z=-5.278$ | $z=-0.795$ | $z=-7.525$ | $z=-0.464$ | $z=-3.160$ | $z=-0.027$ | | | $z=-2.585$ |
| | $p<.001$ | $p=.427$ | $p<.001$ | $p=.642$ | $p=.002$ | $p=.978$ | | | $p=.010$ |
| T14 | $U=7568.5$ | $U=2692.0$ | $U=9594.5$ | $U=3049.0$ | $U=12916.0$ | $U=2610.5$ | $p=.056$ | $p=.019$ | $U=553.5$ |
| | $z=-7.838$ | $z=-2.711$ | $z=-5.483$ | $z=-1.388$ | $z=-1.724$ | $z=-2.842$ | | | $z=-1.173$ |
| | $p<.001$ | $p=.007$ | $p<.001$ | $p=.165$ | $p=.085$ | $p=.004$ | | | $p=.241$ |
| T15 | $U=13176.5$ | $U=2699.5$ | $U=12128.0$ | $U=3083.5$ | $U=13123.5$ | $U=2610.0$ | $p=1$ | $p=.488$ | $U=2914.0$ |
| | $z=-1.960$ | $z=-2.871$ | $z=-3.159$ | $z=-1.133$ | $z=-1.643$ | $z=-2.730$ | | | $z=-0.703$ |
| | $p=.055$ | $p=.004$ | $p=.002$ | $p=.257$ | $p=.100$ | $p=.006$ | | | $p=.482$ |
| T16 | $U=11295.5$ | $U=3376.0$ | $U=9688.0$ | $U=2950.0$ | $U=12485.5$ | $U=3182.5$ | $p=.348$ | $p=.047$ | $U=1866.5$ |
| | $z=-3.921$ | $z=-0.247$ | $z=-5.587$ | $z=-1.742$ | $z=-2.303$ | $z=-0.884$ | | | $z=-2.652$ |
| | $p<.001$ | $p=.805$ | $p<.001$ | $p=.081$ | $p=.021$ | $p=.377$ | | | $p=.008$ |

## Appendix B: Error Messages

This appendix shows the erroneous queries in the test suites, and corresponding error messages output by PostgreSQL, as well as the enhanced error messages (Taipalus and Grahn 2023a).

```
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 with the error highlighted

```
ERROR:  column reference "name" is
   ambiguous
LINE 1: SELECT  name
                ^
```

(b) PostgreSQL error message

```
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;
```

(c) Enhanced error message

**Fig. 5** Test T01

```
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 with the error highlighted

```
ERROR:  column "qa" does not exist
LINE 9:         WHERE name = QA
                             ^
```

(b) PostgreSQL error message

```
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;
```

(c) Enhanced error message

**Fig. 6** Test T02

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

(a) Erroneous query with the error highlighted

```
ERROR:  syntax error at or near
   "350"
LINE 3: WHERE  price_usd IS 350
                            ^
```

(b) PostgreSQL error message

```
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;
```

(c) Enhanced error message

**Fig. 7** Test T03

```
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 with the error highlighted

```
ERROR:  operator does not exist: character varying
   ~~ record
LINE 3: WHERE  name LIKE ('H%', 'J%', 'K%')
                    ^
HINT:  No operator matches the given name and
   argument types. You might need to add explicit
   type casts.
```

(b) PostgreSQL error message

```
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');
```

(c) Enhanced error message

**Fig. 8** Test T04

```
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 with the error highlighted

```
ERROR:  syntax error at or near "ASC
   "
LINE 6: GROUP BY sname ASC, fname
   ASC;
                       ^
```

(b) PostgreSQL error message

```
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;
```

(c) Enhanced error message

**Fig. 9** Test T05

```
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 with the error highlighted

```
ERROR:  subquery has too many
   columns
LINE 6:         WHERE   project_id
   IN
                                   ^
```

(b) PostgreSQL error message

```
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);
```

(c) Enhanced error message

**Fig. 10** Test T06

```
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 with the error highlighted

```
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".
```

(b) PostgreSQL error message

```
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;
```

(c) Enhanced error message

**Fig. 11** Test T07

```
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 with the error highlighted

```
ERROR:  syntax error at or near "LIKE"
LINE 3: WHRE     (brand LIKE 'S%' OR brand LIKE 'C
   %')
                        ^
```

(b) PostgreSQL error message

```
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;
```

(c) Enhanced error message

**Fig. 12** Test T08

```
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 with the error highlighted

```
ERROR:  invalid input syntax for type boolean: "%
   icloud.com"
LINE 3: WHERE  (s.email LIKE '%gmail.com' OR '%
   icloud.com')
                                            ^
```

(b) PostgreSQL error message

```
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%';
```

(c) Enhanced error message

**Fig. 13** Test T09

```
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 with the error highlighted

```
ERROR:  aggregate functions are not allowed in WHERE
LINE 5: AND    price_usd > AVG(price_usd);
                           ^
```

(b) PostgreSQL error message

```
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);
```

(c) Enhanced error message

**Fig. 14** Test T10

```
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 with the error highlighted

```
ERROR:  column "p.status" must appear in the GROUP BY clause or
    be used in an aggregate function
LINE 1: SELECT   e.city, p.status, COUNT(w.employee_id) AS
   number_of...
                           ^
```

(b) PostgreSQL error message

```
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;
```

(c) Enhanced error message

**Fig. 15** Test T11

```
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 with the error highlighted

```
ERROR:  operator does not exist: character varying == unknown
LINE 3: WHERE  brand == 'Oracle'
                     ^
HINT:  No operator matches the given name and argument types.
   You might need to add explicit type casts.
```

(b) PostgreSQL error message

```
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;
```

(c) Enhanced error message

**Fig. 16** Test T12

```
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 with the error highlighted

```
ERROR:  syntax error at or near "WHERE"
LINE 6: WHERE    p.name = 'HR'
        ^
```

(b) PostgreSQL error message

```
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%';
```

(c) Enhanced error message

**Fig. 17** Test T13

```
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 with the error highlighted

```
ERROR:  syntax error at or near "SELECT"
LINE 8:         (SELECT *
                 ^
```

(b) PostgreSQL error message

```
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';
```

(c) Enhanced error message

**Fig. 18** Test T14

```
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 with the error highlighted

```
ERROR:  column p.price does not exist
LINE 1: SELECT p.name, p.price
                       ^
```

(b) PostgreSQL error message

```
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;
```

(c) Enhanced error message

**Fig. 19** Test T15

```
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 with the error highlighted

```
ERROR:  syntax error at or near "SELECT"
LINE 7:             SELECT id
                    ^
```

(b) PostgreSQL error message

```
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);
```

(c) Enhanced error message

**Fig. 20** Test T16

**Author Contributions** *Toni Taipalus*: Conceptualization; Methodology; Investigation; Writing - Original Draft; Visualization; Project administration. *Hilkka Grahn*: Conceptualization; Methodology; Formal analysis; Writing - Review & Editing. *Antti Knutas*: Conceptualization; Metholodogy; Writing - Review & Editing.

**Funding** Open access funding provided by Tampere University (including Tampere University Hospital).

**Data Availability** The data used in this study is not publicly available due to the nature of the privacy statement agreed upon by the study participants.

## Declarations

**Ethical approval** The research was based on informed consent, the physical integrity of the participants was not involved, the participants were not minors, the research did not expose the participants to risk of strong stimuli, mental harm, or safety, and no separate ethical committee approval was required as per our institutions' guidelines.

**Informed consent** Participation was based on informed consent.

**Conflict of Interest** The authors have no competing interests to declare that are relevant to the content of this article.

**Competing interests** The authors have no competing interests to declare that are relevant to the content of this article.

**Clinical Trial Number** Not applicable.

## References

ACM (2015) Curriculum guidelines for undergraduate degree programs in software engineering. Technical report, New York, NY, USA. The Joint Task Force on Computing Curricula

ACM/AIS (2020) A competency model for undergraduate programs in information systems. Technical report. The Joint ACM/AIS IS2020 Task Force

ACM/IEEE (2013) Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science. Technical report, New York, NY, USA. Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society

Ahadi A, Behbood V, Vihavainen A, Prior J, Lister R (2016a) 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). New York, New York, USA, ACM Press, pp 401–406

Ahadi A, Prior J, Behbood V, Lister R (2016b) 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). New York, New York, USA, ACM Press, pp 272–277

Ahmed T, Ledesma NR, Devanbu P (2022) Synshine: improved fixing of syntax errors. IEEE Trans Softw Eng 49(4):2169–2181

Barik T, Ford D, Murphy-Hill E, Parnin C (2018) How should compilers explain problems to developers? In: Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering, ESEC/FSE 2018. New York, NY, USA, Association for Computing Machinery, pp 633–643

Barik T, Smith J, Lubick K, Holmes E, Feng J, Murphy-Hill E, Parnin C (2017) Do developers read compiler error messages? In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). pp 575–585

Becker BA (2016) An effective approach to enhancing compiler error messages. In: Proceedings of the 47th ACM technical symposium on computing science education, SIGCSE '16. New York, NY, USA, Association for Computing Machinery, pp 126–131

Becker BA, Denny P, Pettit R, Bouchard D, Bouvier DJ, Harrington B, Kamil A, Karkare A, McDonald C, Osera P-M, Pearce, JL, Prather J (2019) Compiler error messages considered unhelpful. In: Proceedings of the working group reports on innovation and technology in computer science education (ITiCSE). ACM

Becker BA, Glanville G, Iwashima R, McDonnell C, Goslin K, Mooney C (2016) Effective compiler error message enhancement for novice programming students. Comput Sci Educ 26(2–3):148–175

Brass S, Goldberg C (2006) Semantic errors in SQL queries: a quite complete list. J Syst Softw 79(5):630–644

Cass S (2022) SQL should be your second language. IEEE Spectr 59(10):20–21

Cauley KM, McMillan JH (2010) Formative assessment techniques to support student motivation and achievement. Clearing House J Educ Strateg Issues Ideas 83(1):1–6

Codd EF (1970) A relational model of data for large shared data banks. Commun ACM 13(6):377–387

Costa DAd, Grattan N, Stanger N, Licorish SA (2023) Studying the characteristics of SQL-related development tasks: an empirical study. Empir Softw Eng 28(3)

Denny P, Luxton-Reilly A, Carpenter D (2014) Enhancing syntax error messages appears ineffectual. In: Proceedings of the 2014 conference on innovation & technology in computer science education, ITiCSE. New York, NY, USA, Association for Computing Machinery, pp 273–278

Dix A, Finlay J, Abowd GD, Beale R (2005) Human-computer interaction. Prentice-Hall

Donahue G (2001) Usability and the bottom line. IEEE Softw 18(1):31–37

Dong T, Khandwala K (2019) The impact of "cosmetic" changes on the usability of error messages. In: Extended abstracts of the 2019 CHI conference on human factors in computing systems. ACM

Hannebauer C, Hesenius M, Gruhn V (2018) Does syntax highlighting help programming novices? In: Proceedings of the 40th international conference on software engineering. ACM

ISO/IEC (2016a). ISO/IEC 9075-1:2016, "SQL - Part 1: Framework"

ISO/IEC (2016b). ISO/IEC 9075-2:2016, "SQL - Part 2: Foundation"

Kantorowitz E, Laor H (1986) Automatic generation of useful syntax error messages. Softw Pract Experience 16(7):627–640

Karvelas I, Li A, Becker BA (2020) The effects of compilation mechanisms and error message presentation on novice programmer behavior. In: Proceedings of the 51st ACM technical symposium on computer science education. pp 759–765

Kummerfeld SK, Kay J (2003) The neglected battle fields of syntax errors. In: Proceedings of the fifth Australasian conference on computing education - vol 20, ACE '03. AUS, Australian Computer Society, Inc, pp 105–111

Miedema D, Aivaloglou E, Fletcher G (2021) Identifying SQL misconceptions of novices: findings from a think-aloud study. In: Proceedings of the 17th ACM conference on international computing education research. New York, NY, USA, Association for Computing Machinery, pp 355–367

Miedema D, Fletcher G, Aivaloglou E (2022a) Expert perspectives on student errors in SQL. ACM Transactions on Computing Education

Miedema D, Fletcher G, Aivaloglou E (2022b) 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. New York, NY, USA, Association for Computing Machinery, pp 122–132

Migler A, Dekhtyar A (2020) Mapping the SQL learning process in introductory database courses. In: Proceedings of the 51st ACM technical symposium on computer science education, SIGCSE '20. New York, NY, USA, Association for Computing Machinery, pp 619–625

Nienaltowski M-H, Pedroni M, Meyer B (2008) Compiler error messages: What can help novices? SIGCSE Bull 40(1):168–172

Pane J, Myers B, Miller L (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

Pasupuleti KK, Li J, Su H, Ziauddin M (2023) Automatic SQL error mitigation in Oracle. Proc VLDB Endowment 16(12):3835–3847

Pettit RS, Homer J, Gee R (2017) Do enhanced compiler error messages help students? results inconclusive. In: Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education, SIGCSE '17. New York, NY, USA, Association for Computing Machinery, pp 465–470

Prather J, Pettit R, McMurry KH, Peters A, Homer J, Simone N, Cohen, M (2017) On novices' interaction with compiler error messages: a human factors approach. In: Proceedings of the 2017 ACM conference on international computing education research, ICER '17. New York, NY, USA, Association for Computing Machinery, pp 74–82

Presler-Marshall K, Heckman S, Stolee K (2021) (SQLRepair: identifying and repairing mistakes in student-authored SQL queries. In: 2021 IEEE/ACM 43rd international conference on software engineering: software engineering education and training (ICSE-SEET). IEEE, pp 199–210

Reisner P (1977) Use of psychological experimentation as an aid to development of a query language. IEEE Trans Softw Eng SE–3(3):218–229

Reisner P, Boyce RF, Chamberlin DD (1975) Human factors evaluation of two data base query languages. In: Proceedings of the national computer conference and exposition AFIPS '75. ACM Press

Rennels L, Chasins SE (2023) How domain experts use an embedded DSL. Proc ACM Program Lang 7(OOPSLA2)

Sarkar A (2015) The impact of syntax colouring on program comprehension. In: Proceedings of the 26th annual conference of the psychology of programming interest group (PPIG 2015)

Seo H, Sadowski C, Elbaum S, Aftandilian E, Bowdidge R (2014) Programmers' build errors: a case study (at Google). In: Proceedings of the 36th international conference on software engineering. pp 724–734

Shneiderman B (1982) Designing computer system messages. Commun ACM 25(9):610–611

Shneiderman B, Plaisant C, Cohen MS, Jacobs S, Elmqvist N, Diakopoulos N (2016) Designing the user interface: strategies for effective human-computer interaction. Pearson

Smelcer JB (1995) User errors in database query composition. Int J Hum Comput Stud 42(4):353–381

Taipalus T (2020) Explaining causes behind SQL query formulation errors. In: 2020 IEEE frontiers in education conference (FIE). pp 1–9

Taipalus T (2023a) Query execution plans and semantic errors: usability and educational opportunities. In: Extended abstracts of the 2023 CHI conference on human factors in computing systems, CHI EA '23. New York, NY, USA, Association for Computing Machinery

Taipalus T (2023b) SQL: a Trojan horse hiding a decathlon of complexities. In: Proceedings of the 2nd international workshop on data systems education, DataEd '23. New York, NY, USA, Association for Computing Machinery, pp 9–13

Taipalus T, Grahn H (2023a) Framework for SQL error message design: a data-driven approach. ACM Trans Softw Eng Methodol 33

Taipalus T, Grahn H (2023) New SQL database management system compiler errors: effectiveness and usefulness. Int J Human-Comput Interact 39:3936–3947

Taipalus T, Grahn H (2024) Building blocks towards more effective SQL error messages. In: Proceedings of the 2024 on innovation and technology in computer science education V. 1, ITiCSE 2024. New York, NY, USA, Association for Computing Machinery, pp 241–247

Taipalus T, Grahn H, Ghanbari H (2021) Error messages in relational database management systems: a comparison of effectiveness, usefulness, and user confidence. J Syst Softw 181:111034

Taipalus T, Grahn H, Ritonummi S, Siitonen V, Vartiainen T, Zhidkikh D (2025) Novice perceptions on effective elements of PostgreSQL error messages. ACM Transactions on Computing Education, Just Accepted

Taipalus T, Perälä P (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. New York, NY, USA, ACM, pp 198–203

Taipalus T, Seppänen V (2020) SQL education: a systematic mapping study and future research agenda. ACM Trans Comput Educ 20(3)

Taipalus T, Siponen M, Vartiainen T (2018) Errors and complications in SQL query formulation. ACM Trans Comput Educ 18(3):15:1-15:29

Thiselton E, Treude C (2019) Enhancing Python compiler error messages via Stack. In: 2019 ACM/IEEE international symposium on empirical software engineering and measurement (ESEM). pp 1–12

Traver VJ (2010) On compiler error messages: What they say and what they mean. Advances in Human-Computer Interaction, pp 1–26

Wang J, Chen S, Tang Z, Lin P, Wang Y (2024) False positives and deceptive errors in SQL assessment: a large-scale analysis of online judge systems. ACM Trans Comput Educ, Just Accepted

Welty C (1985) Correcting user errors in SQL. Int J Man Mach Stud 22(4):463–477

Wrenn J, Krishnamurthi S (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

Zhou Z, Wang S, Qian Y (2021) Learning from errors: exploring the effectiveness of enhanced error messages in learning to program. Front Psychol 12