JYU DISSERTATIONS 283

Toni Taipalus

Persistent Errors in Query Formulation

UNIVERSITY OF JYVÄSKYLÄ
FACULTY OF INFORMATION TECHNOLOGY

JYU DISSERTATIONS 283

Toni Taipalus

Persistent Errors in Query Formulation

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi syyskuun 29 päivänä 2020 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
on September 29, 2020 at 12 o'clock noon.

JYVÄSKYLÄN YLIOPISTO
UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2020

Editors
Marja-Leena Rantalainen
Faculty of Information Technology, University of Jyväskylä
Timo Hautala
Open Science Centre, University of Jyväskylä

Permanent link to this publication: http://urn.fi/URN:ISBN:978-951-39-8290-4

ISBN 978-951-39-8290-4 (PDF)
URN:ISBN:978-951-39-8290-4
ISSN 2489-9003 Jyväskylä University Printing House, Jyväskylä 2020

ABSTRACT

Taipalus, Toni
Persistent Errors in Query Formulation
Jyväskylä: University of Jyväskylä, 2020, 40 p. (+included articles)
(JYU Dissertations
ISSN 2489-9003; 283)
ISBN 978-951-39-8290-4 (PDF)

We use the internet daily to query data from a myriad of databases; every search term entered in a search engine, every movie watched, every song listened, every newspaper article read online. Although we as end-users only see the relatively effortless user interfaces as we query data, someone has had to formalize our queries into a language the software understands. The most common of these so called query languages is Structured Query Language (SQL). In order for us as end-users to retrieve exactly the data we want, it is crucial that the software developers responsible for writing the underlying queries have written the queries without errors. Educational SQL research, however, has not yet thoroughly addressed issues related to understanding query formulation errors or some technical factors which influence the process of learning SQL. This doctoral dissertation makes the following contributions for increased understanding of SQL education: *(i)* a systematic overview of SQL teaching practices proposed in scientific literature, *(ii)* a creation of a wide taxonomy of errors committed in SQL learning, *(iii)* a description of which types of errors halt query formulation, and which types of encountered errors are usually fixed, *(iv)* evidence on the effects of database complexity on query formulation success rates, and *(v)* a creation of a planning notation designed to mitigate errors in query formulation. Contribution *(ii)* presents practical implications for research by allowing the comparison of results of different SQL error studies when the taxonomy is used, and extending and generalizing prior SQL error studies. While contributions *(i)* and *(v)* may be directly applied in teaching SQL, contributions *(iii)* and *(iv)* may be considered when making an informed decision on what kind of databases are the most suitable for practicing SQL.

Keywords: Structured Query Language (SQL); computing education; database; relational; error; logical complexity; planning; notation;

TIIVISTELMÄ (ABSTRACT IN FINNISH)

Taipalus, Toni
Pysyvät virheet tietokantakyselyissä
Jyväskylä: University of Jyväskylä, 2020, 40 s. (+artikkelit)
(JYU Dissertations
ISSN 2489-9003; 283)
ISBN 978-951-39-8290-4 (PDF)

SQL-kieli (Structured Query Language) on yleisin kyselykielistä, joita käytetään tiedon hakemiseen tietokannoista. Jotta kyselykielellä kirjoitetut kyselyt palauttaisivat loppukäyttäjälle hänen haluamaansa oikeellista tietoa, täytyy kysely kirjoittaa virheettömästi. SQL-kielen opetuksen tutkimuksessa erilaiset virheet eivät kuitenkaan ole saaneet laajaa tai syvällistä tieteellistä huomiota. Lisäksi joidenkin SQL-kielen oppimiseen liittyvien seikkojen vaikutusta ei ole tieteellisesti tutkittu. Tässä väitöskirjassa *(i)* tehdään systemaattinen kirjallisuuskartoitus SQL-kielen opetuksessa käytetyistä menetelmistä, *(ii)* muodostetaan laaja SQL-kielen kyselyiden kirjoittamisessa tehtyjen virheiden taksonomia, *(iii)* selitetään mitkä virhetyypeistä ovat pysyviä, eli jäävät tavallisesti kyselyn kirjoittajalta korjaamatta ja mitkä virhetyypeistä tavallisesti korjataan, *(iv)* esitetään harjoitustietokannan monimutkaisuuden vaikutukset kyselyn kirjoittamiselle ja *(v)* kuvaillaan suunnittelunotaatio, jonka tarkoituksena on pysyvien virheiden lukumäärän vähentäminen. Edellä luetelluista kontribuutioista *(ii)* auttaa taksonomiaa käyttävien tieteellisten tutkimusten tuloksien vertailussa, sekä luo pohjan esimerkiksi virheiden ja virhetyyppien paikantamiseen koneoppimismenetelmin. Kontribuutioita *(i)* ja *(v)* voidaan käyttää sellaisenaan SQL-kielen opetuksessa, ja kontribuutiot *(iii)* ja *(iv)* auttavat päätöksenteossa sopivan monimutkaisen harjoitustietokannan valinnassa.

Avainsanat: Structured Query Language (SQL); tietokanta; virhe; kysekykieli; loogisen rakenteen monimutkaisuus; suunnittelu; notaatio;

Author

Toni Taipalus
Faculty of Information Technology
University of Jyväskylä
Finland

Supervisors

Dr. Mikko Siponen
Faculty of Information Technology
University of Jyväskylä
Finland

Dr. Tero Vartiainen
Department of Computer Science
University of Vaasa
Finland

Reviewers

Dr. Jan Erik Moström
Department of Computing Science
Umeå University
Sweden

Dr. Julia Prior
School of Computer Science
Faculty of Engineering and Information Technology
University of Technology Sydney
Australia

Opponent

Dr. Antti Knutas
School of Engineering Science
Department of Software Engineering
Lappeenranta University of Technology
Finland

ACKNOWLEDGEMENTS

My most sincere thanks to my supervisors and co-authors, professors Mikko Siponen and Tero Vartiainen, for your support and trust over the years. Thank you reviewers, Dr. Jan Erik Moström and Dr. Julia Prior for your kind comments and constructive criticism, and Dr. Antti Knutas for taking the time and effort as the opponent in my defence. Thank you, co-authors Dr. Ville Seppänen and Piia Perälä. Thank you, professors Pekka Abrahamsson, Tuomo Rossi and Lauri Kettunen for your trust and for always being supportive. You have given me the opportunity to focus on research and teaching I am genuinely passionate about. Thank you, Dr. Henri Pirkkalainen for your guidance, especially in the beginning of my studies. Thank you, Dr. Mauri Leppänen for your contagious enthusiasm for databases, and for your examples in teaching. Thank you, numerous co-workers, especially co-conspirer Hilkka Grahn for continuously assuring me to keep on trying. Without you, I do not know how I could have survived all the one-of-those-days at the office, the desk rejects, those countless faculty parties, or the popular culture cues in the crossword puzzles. Thank you, researchers in the Computer Science Education research group, especially Dr. Ville Isomöttönen for always leading us with a professional and positive attitude. Thank you, family, friends and critters home and abroad for just being there for me. Over the years, you commendably endured my ramblings about research you had absolutely no interest in. And thank you, who are already with the summer sun and brighter sands.

Toni Taipalus
Jyväskylä
August 2020

LIST OF FIGURES

FIGURE 1 A relation represented as a table 14
FIGURE 2 An example of a conceptual, logical, and physical representation of the same database structure 15
FIGURE 3 An example of a basic SQL learning environment 16
FIGURE 4 A summary of factors affecting query formulation success rates 19

CONTENTS

ABSTRACT
TIIVISTELMÄ (ABSTRACT IN FINNISH)
ACKNOWLEDGEMENTS
LIST OF FIGURES
CONTENTS
LIST OF INCLUDED ARTICLES

1 INTRODUCTION 11

2 THEORETICAL BACKGROUND 13
2.1 The relational model and SQL 13
2.2 SQL in education 15
2.3 Query formulation errors 16
2.4 Causes behind errors 18

3 SUMMARY OF ARTICLES 21
3.1 Overview and motivation 21
3.2 Contributions 23
3.2.1 Article PI 23
3.2.2 Article PII 24
3.2.3 Article PIII 25
3.2.4 Article PIV 25
3.2.5 Article PV 26

4 DISCUSSION 27
4.1 Practical implications 27
4.2 Limitations and threats to validity 28
4.3 Ethical considerations 30
4.4 Future agenda 31

5 CONCLUSIONS 32

YHTEENVETO (SUMMARY IN FINNISH) 33

REFERENCES 34

INCLUDED ARTICLES

LIST OF INCLUDED ARTICLES

PI Toni Taipalus and Ville Seppänen. SQL education: A systematic mapping study and future research agenda. *ACM Transactions on Computing Education*, 20(3), Article 20, 2020.

PII Toni Taipalus, Mikko Siponen, and Tero Vartiainen. Errors and complications in SQL query formulation. *ACM Transactions on Computing Education*, 18(3), Article 15, 2018.

PIII Toni Taipalus and Piia Perälä. What to expect and what to focus on in SQL query teaching. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19).*, ACM, New York, 198-203, 2019.

PIV Toni Taipalus. The effects of database complexity on SQL query formulation. *Journal of Systems and Software*, 165, Article 110576, 2020.

PV Toni Taipalus. Teaching tip: A notation for planning SQL queries. *Journal of Information Systems Education*, 30(3), 160-166, 2019.

1 INTRODUCTION

Structured Query Language (SQL) is the most common way to define databases and manipulate data. As arguably all information systems consist of a database, SQL is a crucial tool for software developers. It follows that SQL is an integral part of ICT education, and understanding common mistakes and factors influencing SQL learning is an important step in further developing SQL teaching. One factor in teaching SQL is the exercise database against which students execute SQL queries in order to learn the language. It is rather common that the exercise database is relatively simple (e.g., Elmasri and Navathe, 2016; Kroenke and Auer, 2016), although real life databases are usually complex (Cleve et al., 2015). Some scholars have argued that a simple exercise databases are beneficial because they facilitate the focus of learning to SQL semantics, and away from the complexity of the database (Wagner et al., 2003). Others, however, have argued for the use of complex databases in order to better help students cope in their future work environments (Jukic and Gray, 2008a; Watson and Hoffer, 2003). In order to facilitate more effective SQL teaching, we need to provide students the most suitable environment, while taking into account limitations concerning teacher workload, i.e., it might not be feasible to tailor the environment for individual students. Before making an informed decision regarding a suitable environment, we need to understand factors that influence query writing.

Studies have identified several factors that influence query writing success rates, e.g., natural language and database representation considerations. However, scientific evidence on the effects of database complexity on query formulation is lacking. Arguably, if database complexity has no negative effects on learning, it would be beneficial for students to learn SQL in an environment which resembles their future work. However, in order to both measure success in query writing, and to understand the causes for failure, we as researchers need to understand what an error is, and what types of errors are possible in query formulation. Although errors have been studied extensively in the realm of programming languages (Smith and Rixner, 2019), SQL has remained in the sidelines in this regard. Studies concerning SQL errors have largely discussed errors from the perspective of a single database management system (e.g., Ahadi et al., 2016a), and no broad

error categorization has been attempted. Finally, the assumption of practicing in a realistic environment extends to query complexity as well; by facilitating the learning of complex queries, we may be able to more effectively prepare students for their future work, where complex queries are often required.

To facilitate SQL teaching, we present an aggregated body of knowledge on SQL teaching practices proposed in scientific literature, and systematically map SQL education research topics and publication fora. We formulate an extensive database management system independent SQL error categorization based on prior studies and empirical analysis. Subsequently, we focus on errors which are more difficult to fix than others and seem to halt query formulation. Based on this groundwork, we measure the effects of database complexity on query formulation, and present that an increase in database complexity results in a decrease in query formulation success rates. Finally, we introduce a notation for planning SQL queries, which we have designed to mitigate query formulation errors originating from cognitive factors.

Rather than repeating the contents of the included articles in this research article based doctoral dissertation, we provide an overview, briefly summarize the articles, and discuss the implications of this line of research as a whole. The rest of this dissertation is structured as follows. In Chapter 2, we present the theoretical background necessary to understand the following sections. In Chapter 3 we discuss the motivation behind the articles, and present our research questions and contributions. In Chapter 4, we consider the practical implications of our results, threats to validity, and future research agenda. Chapter 5 concludes the dissertation, and is followed by the original articles.

2 THEORETICAL BACKGROUND

In this chapter, we define and discuss key terms regarding this line of research. While some terms are described in detail in the included articles, we focus here on terms which are crucial in later understanding the motivation and contributions of the articles as a whole. The narrative advances from the general to more and more on the research problems addressed by this study.

2.1 The relational model and SQL

Databases follow one or more *data models*, or paradigms, which dictate what kind of data structures are available, what kind of data can be stored, and sometimes what kind of operations are available to process the database data. Data models are usually divided into *conceptual*, *logical*, and *physical*, from the highest to lowest the level of abstraction. Prime examples conceptual and logical level paradigms are the Entity-Relationship (ER) model (Chen, 1976) and the relational model (Codd, 1970), respectively. In general, conceptual level data models are used to describe data item characteristics and interrelations on a high level, and independently of the underlying paradigm. Consequently, logical level data models are used to describe data items in relation to a selected paradigm, but independently of the physical implementation. In order to understand SQL, it is crucial to understand the theoretical foundations behind the underlying data model.

In this study, we focus on the relational model. The relational model is a logical level data model, and consists of data structures and set theory based operations (Codd, 1971a, 1972). The most important data structures defined by the relational model are *attributes*, which consist of an attribute *name* and a corresponding atomic, i.e., indivisible *value* (e.g., name-value pairs $\langle color : red \rangle$ and $\langle brand : Toyota \rangle$ in Fig. 1). Next, *tuples* group attributes into representations of an entity or an event in the business domain. Furthermore, *relations* group tuples representing similar entities or events together, and a relation consists of a header, i.e., *intention* of what kind of data the relation holds, and the data proper,

CAR

plateno	brand	model	color
7C88505	Opel	Corsa	red
1A10290	Toyota	Corolla	black
6C88471	Toyota	Corolla	red

FIGURE 1 A relation represented as a table

i.e., *extension*. Importantly, the relational model describes two main integrity constraints, namely *primary* and *foreign key*. As all tuples in a relation must be distinct (Codd, 1970), it follows that in a relation there is at least one set of attributes that has different values for each tuple of the relation, and one of these sets is selected as the relation's primary key. Finally, a foreign key represents a set of attributes, which is used to enforce that the values of the foreign key attributes are also present in some other relation, usually as that relation's primary key.

The most important operations in the relational model are projection (i.e., selecting attributes), selection (i.e., selecting tuples), and join (i.e., finding common values in two sets of attributes). One particularly common way to implement these operations is SQL, and the SQL standard (ISO/IEC, 2016a,b) provides a description of how these – and additional – operations could be or should be implemented. SQL adopts the secondary, perhaps more intuitive, nomenclature used by Codd (1970): rough equivalents of sets of attributes of the same name are referred to as columns, tuples as rows, and relations as tables. SQL also abandons the property of the relational model which requires all rows to be distinct Date (1983). While the operations defined by Codd (1970) retrieve, i.e., *query* data from a relational database, SQL extends to other types of operations as well. In addition to data retrieval, SQL can be used to manipulate data, define data structures and other *database objects*, e.g., users, indices, and functions, define user privileges, and control database transactions.

In contrast to the SQL standard and the relational model which are more theoretical foundations, the relational model and SQL are implemented by the many database management systems (DBMS), and there are several key things to note in understanding the physical implementations. First, even though SQL is defined by the SQL standard, DBMSs implement SQL differently (Buitendijk, 1988; Randolph, 2003). Second, SQL has been from its very inception contradictory to some aspects of the relational model, e.g., in handling duplicate values (Date, 1983), the implentation of missing values (Chamberlin, 2012), and in nomenclature. Third, although the relational model is a logical data model, SQL is not limited to the logical representation level, and can also be used to define physical database aspects, e.g., data types and indices. Fourth, SQL has evolved from the relational model to concepts which are, by definition, in violation the rules of the first normal form, e.g., non-atomic values (Eisenberg et al., 2004; Kulkarni and Michels, 2012), or are adopted from different data models altogether (Eisenberg and Melton, 1999).

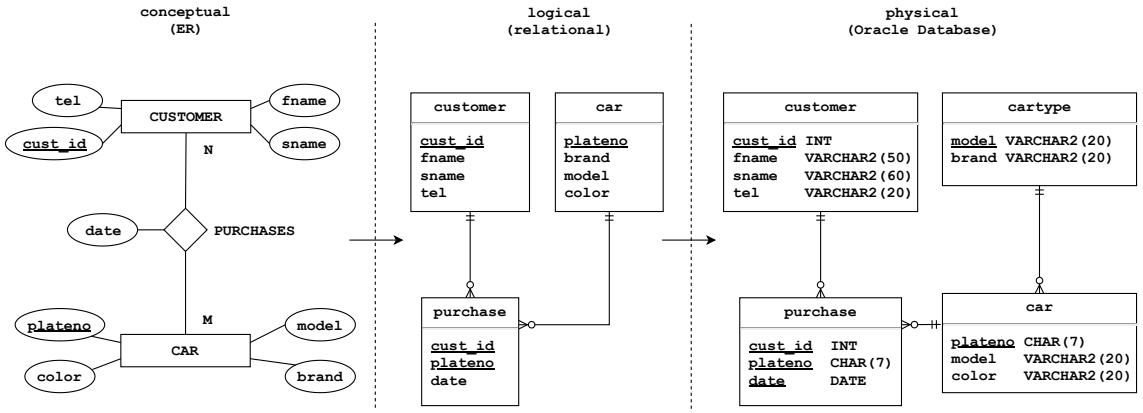


FIGURE 2 An example of a conceptual, logical, and physical representation of the same database structure

2.2 SQL in education

SQL is part of several ICT curricula, e.g., in the ACM/AIS undergraduate information systems curriculum guidelines (Topi et al., 2010), SQL is one of the few specifically mentioned computer languages. Furthermore, SQL is mentioned in the ACM/IEEE computer science (ACM/IEEE, 2013) and software engineering (ACM/IEEE, 2015) curricula guidelines. These guidelines vary in terms of the level of detail, but seem to agree that data retrieval and manipulation, and database structure definition are among the key concepts students should learn.

SQL is not taught in isolation, but usually as a part of a database course. In our experience, three most important topics that should accompany SQL in a database course are the ER and relational data models, and normalization theory, all of which are among the database course topics in, e.g., the undergraduate information systems curricula guidelines (Topi et al., 2010). First, learning conceptual modeling helps students to understand of what is relevant in the business domain in terms of data, and how domain relevant data are linked. Although the common approach for conceptual modeling is the ER model, Watson (2006) argues that the notation is a secondary concern. As SQL was designed as a query language for relational databases, it is natural that the theoretical foundations of the underlying data model are taught before SQL (although opposing views also exist). Next, the conceptual data model is transformed into a logical data model (Fig. 2).

At this stage, it is natural to introduce relational database design practices, i.e., normalization theory (Codd, 1970, 1971b, 1972). In layperson's terms, normalization is an iterative design process which rearranges the attributes and relations in a way that aims to minimize data redundancy and hence data anomalies. As a simple example, in Fig. 2, normalization is applied in the transition between the logical and physical levels, and the relation *car* is decomposed to reduce redundancy with the assumption that no car model is associated with more than one car brand. After normalization is applied to a level suitable to the needs of the business domain, the database may be implemented on the physical level using a

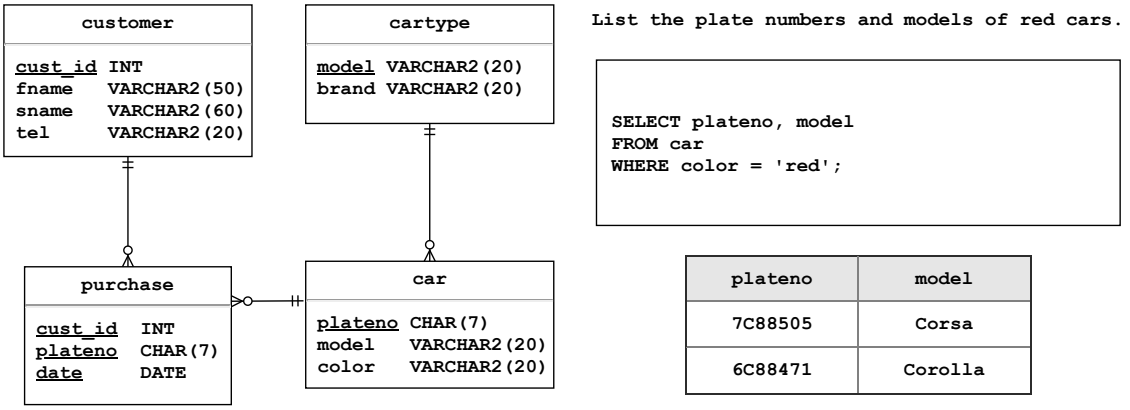


FIGURE 3 An example of a basic SQL learning environment

selected DBMS, e.g., Oracle Database. After the database has been implemented, SQL may be utilized to query data and to receive result tables. In addition to aspects presented in the data model in Fig. 2, the physical data model may contain additional information, e.g., database indices.

In addition to lectures, it is common that students learn practical SQL skills using a digital learning environment (Brusilovsky et al., 2010). These environments are aplenty (Mitrovic, 1998, 2003; Permpool et al., 2019; Prior, 2014), and have different advanced features designed to support SQL learning, e.g., team forming and collaborative learning (Abelló et al., 2016), personalized or adaptive feedback (Mitrovic, 2003; Kenny and Pahl, 2005; Nalintippayawong et al., 2017), and query execution visualization (Garner and Mariani, 2015). In contrast to these advanced features, the digital learning environments typically consist of four basic elements: a representation of the exercise database against which the query is executed by the DBMS, a *data demand* expressed in natural language (e.g., "List the plate numbers and models of red cars"), an input field into which the corresponding SQL query needs to be written, and the result table output by the DBMS after the execution of the query (Fig. 3). Additionally, the environment may contain auxiliary elements, e.g., data demand in pseudo-SQL instead of natural language (Casterella and Vijayasarathy, 2013), or the correct result table the query should return (Reilly, 2018), which may help the student to formulate the query or validate the result. If the result table is incorrect, or the DBMS returns an error message instead, the query contains an error.

2.3 Query formulation errors

In any language, errors may be committed in a myriad of ways. While some errors are archetypal and common among different query writers, others are chimeric and subjective, possibly pertaining to different language related misconceptions, database object name misreads, typographical errors, or simply ignorance. For the sake of narrative, we explore the development of scientific literature around query formulation errors, namely *what errors occur*, in three steps.

At the *first step* in understanding query formulation errors, errors have been divided into *syntax* and *semantic* errors (Buitendijk, 1988; Smelcer, 1995). This is an intuitive starting point, as DBMSs check the syntax of the submitted query, and return a syntax error if the query is syntactically incorrect.

Syntax errors violate the rules of SQL, and even one prevents the execution of the query. Archetypal syntax errors are, e.g., misspellings of SQL keywords (Buitendijk, 1988), incorrect ordering of keywords (Welty, 1985), unmatched parentheses, and keywords used in incorrect contexts (Ahadi et al., 2016a). Chimeric syntax errors, being highly diverse, are usually grouped together and not discussed in detail (e.g., Ahadi et al., 2016a). A key thing worth noting, especially in the context of syntax errors, is that different DBMSs implement different syntax checks. What might be considered a syntax error in one DBMS might be tolerated by another. Consequently, a specific DBMS is often used to categorize syntax errors, and these categorizations are based on a single DBMS's point of view (Ahadi et al., 2016a). This poses a problem in terms of generalizability, when results from two studies using different DBMSs need to be compared.

According to the level of understanding at this step, if the result table is returned but incorrect, the query contains a semantic error. In other words, semantic errors are a equivalent to *errors other than syntactic*, as semantic errors are not checked by DBMSs, or if they are, the query writer is not notified whether the query contains a semantic error. Queries with one or more semantic error usually retrieve a result table which does not correspond to the data demand, or corresponds to the data demand, but only with the current exercise database data. Examples of archetypal semantic errors are missing conditions (e.g., listing all cars instead of red cars), retrieving unneeded columns, and failing to sort the result table according to some attribute value (Ahadi et al., 2016b).

Buitendijk (1988) expressed a tangential - perhaps even axiomatic - notion that a query should correspond to a specific data demand. This notion was later explored in detail, and Brass and Goldberg (2006) further divided queries with semantic errors to *always incorrect* and *incorrect in regards to the data demand*. This marked the *second step* in understanding query formulation errors. While examples of the latter class have been provided in the previous paragraph, an *always incorrect* query would try, inconsistently, to retrieve, e.g., all cars which are both red *and* black from a database which only allows one color per car. The key detail here is the subtle yet distinct difference between retrieving cars which are both red and black, and retrieving red cars and black cars. For clarity, we henceforth call the former type of semantic errors (e.g., inconsistencies and tautologies) semantic, and the latter class of semantic errors (e.g., missing conditions) *logical* errors.

At the *third step*, in addition to the three aforementioned error classes, research extends to the concept of *complications* (Brass and Goldberg, 2006). Queries that exhibit complications could be formulated in a simpler fashion. It is worth noting that this is not a subjective consideration, e.g., violations of established practices regarding uppercase keywords, indentation, or uniform suffixes are not considered complications. Rather, a table join, or a condition is a complication if

its omittance would not affect the result table, for any data in the database. While the presence of complications does not affect the result table, and consequently complications are not errors in the strict sense, we address them as such. Similarly to syntax errors, it is possible for DBMSs to recognize semantic errors and complications (Brass and Goldberg, 2006), but not logical errors before DBMSs can understand natural language data demands. When DBMSs can recognize whether a query logically corresponds to a data demand, DBMSs arguably can reliably convert data demands into SQL, thus making SQL a less important educational topic. For now, this is not the case, although attempts have been reported (Li and Jagadish, 2014).

Although this division of errors into four classes seems justified, researchers have not widely adapted this point of view, and the few studies on SQL errors still operate according to the first step (Ahadi et al., 2015, 2016a,b). However, as a necessary sidetrack to this narrative, we have noticed that industry recognizes the existence of complications and semantic errors, but show it only for a small subset of DBMS users. The most popular five[1] relational DBMSs Oracle Database, MySQL, SQL Server, PostgreSQL, and DB/2 all display warnings or indications of some inconsistencies in queries in respective query execution plans. Although this does not benefit application developers or writers of *ad hoc* queries, personnel working on database optimization may see these indications. If nothing else, the implementations of semantic error and complication recognition are evidence that DBMS vendors can and want to utilize these considerations in practice.

2.4 Causes behind errors

A natural step in understanding query formulation errors is moving from *what* errors occur to *why* errors occur (Reisner, 1981), before these problems can be fixed (Smelcer, 1995). For the sake of structure, we first discuss the causes behind query formulation errors stemming from the query writer, and then causes stemming from the surrounding environment. These two categories of causes are not disjoint, but merely consider the causes behind errors from different viewpoints. Additionally, causes may stem from aspects we as teachers have no feasible control over, e.g., the design of the SQL language itself (Reisner, 1981).

From the query writer perspective, causes behind query formulation errors have mostly been studied prior the year 2000. On a high level, errors are caused by insufficient knowledge of the data, the database structure, or the query language (Ogden et al., 1986, as cited in Smelcer, 1995). This level, however, is too general to explain causes behind errors concerning different SQL concepts (Smelcer, 1995). Based on Reisner (1977, 1981, 1988), Smelcer (1995) introduced a model of four cognitive explanations for SQL errors: *(i)* overloading working memory, *(ii)* absence of a clue, *(iii)* procedural fixedness, and *(iv)* ignorance. The model is based on the presupposition that query formulation is a cognitive pro-

[1] https://db-engines.com/en/ranking

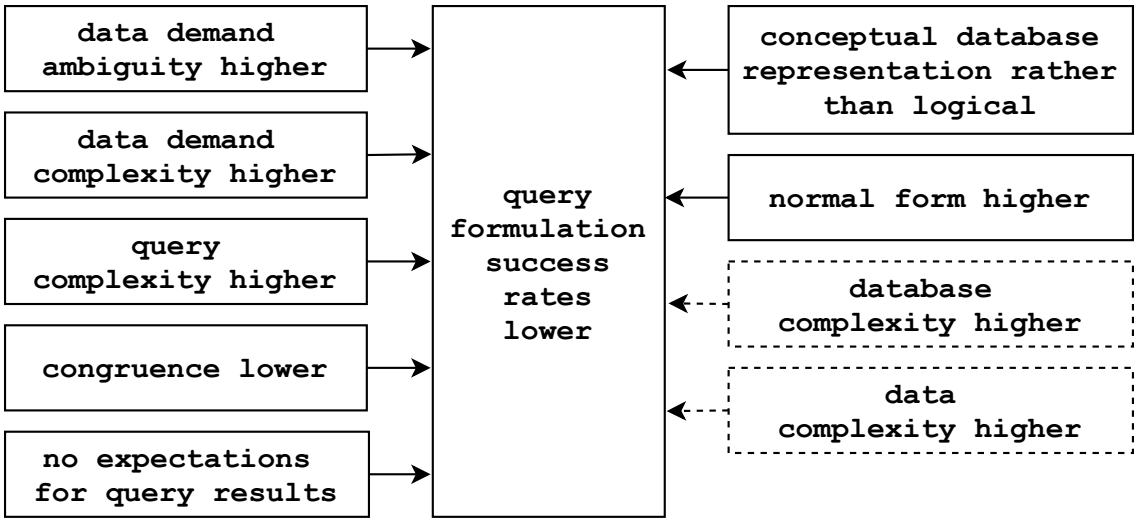


FIGURE 4 A summary of factors affecting query formulation success rates

cess of translating a natural language data demand into an SQL query, and problems in the translation process lead to errors in the query.

First, if the number of items, e.g., database object names, SQL keywords, or conditions, exceeds the query writer's subjective working memory capacity, certain items are omitted, which in turn leads to query formulation errors. Second, if the data demand is imprecise or lacks a clue (or a cue), the translation process requires additional cognitive effort, and possibly additional interpretation. Third, procedural fixedness is related to incorrectly utilizing translation procedures associated with preceding translations, e.g., querying a single table as in previous query formulation tasks, although a multi-table query is required. Fourth, ignorance, or absence of procedural knowledge simply means that the query writer lacks sufficient skill to translate the data demand. Smelcer (1995) regards this explanation as "not theoretically interesting", which is arguably a dismissive statement in the context of educational research, yet captures phenomena such as students simply trying to write SQL without first familiarizing themselves with the language on a theoretical level.

From the environment's perspective, causes behind query formulation errors have been studied extensively. Intuitively, the factors that have the potential to affect query formulation success rates are related to the data demand, query, database, or database representation. These factors are summarized in Fig. 4, and solid lines represent hypotheses supported by scientific evidence, while dotted lines represent untested hypotheses or mere speculation.

As data demand ambiguity (cf. Ashkanasy et al., 2007) increases, query formulation success rates decrease (Axelsen et al., 2001; Borthick et al., 2001a; Casterella and Vijayasarathy, 2013, 2019). With Smelcer's (1995) presupposition of the absence of clues in mind, this is an expected development, as query writers need to interpret an ambiguous data demand, in addition to translating it into a query. As data demand complexity increases, query formulation success rates decrease (Borthick et al., 2001a). Similarly, an increase in query complexity causes success rates to decrease (Borthick et al., 2001a; Chan et al., 1999). It is worth noting that

a complex data demand does not necessarily result in a more complex query or vice versa. In some cases, depending on the data demand, database, and the natural language used, it is possible to express data demands simply, even for those that require complex queries. Congruence refers to how well the database objects correspond to their real world equivalents. If congruence decreases, so do success rates (Borthick et al., 2001a). It has also been shown that if the query writer has expectations on how much or what kind of data the query should return, they can more accurately formulate queries and correct them if necessary, as opposed to query writers who have no expectations (Robb et al., 2012). If the database structure is presented on a conceptual level (e.g., as an ER diagram), query formulation success rates are lower than with a database presentation on a logical level (e.g., as a database schema diagram) (Davis, 1990; Leitheiser and March, 1996), although contradictory evidence has also been presented (Chan et al., 1999; Siau et al., 2004). Additionally, it has been shown that higher normal forms, i.e., further normalization from the first normal form results in a decrease in query formulation rates, at least in some types of queries (Borthick et al., 2001b; Bowen et al., 2004).

Rather surprisingly, neither the effects of the complexity of the database structure (e.g., a few tables versus many tables) or data (e.g., a few rows versus many rows) have been studied, although the latter has received arguments in favor of complex data which arguably better prepares students for their future work (Gudivada et al., 2007; Ortiz et al., 2012; Wagner et al., 2003). Furthermore, it has been shown that students find more complex databases more interesting and useful (Yue, 2013).

3 SUMMARY OF ARTICLES

In this chapter, we first discuss the motivation and assumptions behind the premises of this line of research, and our research questions. Next, we briefly summarize Articles PI through PV, their aims and contributions, and the author's role in each article.

3.1 Overview and motivation

The line of research presented in the included articles rests on four assumptions which are argued for next.

Assumption 1. We as educators and researchers need to understand student errors in query formulation if we are to facilitate learning.

First, as hinted in previous chapters, errors in SQL query formulation have not been systematically mapped. With the exception of Brass and Goldberg (2006), the error categorizations presented in prior studies have rather been presented as *a priori* established vehicles for answering respective research questions, rather than results of those studies. Furthermore, the errors presented in previous studies are by design a small subset of SQL errors (e.g., Smelcer, 1995), and sometimes error categorizations have been based on a single DBMS's point of view (e.g., Ahadi et al., 2016a). Indirectly inspired by Gregor's (2006) taxonomy of theory development in information systems, we deemed that before trying to understand more complex questions of *why something happens*, *what will happen*, or *how to do something*, we needed to first study *what* errors occur, as we considered prior scientific literature on the topic insufficient. As Metcalfe (2017) points out, understanding student errors helps teachers focus instruction on difficult concepts, and what aspects of the task possibly cause students to miss correct solutions. By understanding what types of errors novices commit enables us as researchers to move towards quantifying those errors in different research settings.

Assumption 2. Although committing errors is a part of the learning process, *persistent* errors in particular are a desirable research focus.

Second, simply avoiding errors in various learning situations may be counterproductive to learning (Metcalfe, 2017), whereas simply succeeding in all tasks may result in learning not taking place at all (Wilson et al., 2019). As Metcalfe (2017) points out, not all errors are equal. Therefore, we focused on persistent, i.e., never corrected errors, and considered them an indication of either decisively or prolongedly halted learning experience.

Assumption 3. Understanding what aspects in the environment cause persistent query formulation errors is an important step in choosing the most appropriate learning environment parameters.

Third, as underlined in Subsection 2.4, the effects of database complexity on SQL query formulation success rates have not been studied, while many other aspects in the environment have received scientific attention. To better prepare students for their career, many studies have argued for utilizing *natural* environments, i.e., environments which more accurately mirror students' future work (Ahadi et al., 2015; AL-Salmi, 2018; Yue, 2013). As textbook examples often utilize relatively simple exercise database structures (e.g., Elmasri and Navathe, 2016; Connolly and Begg, 2015; Kroenke and Auer, 2016), those databases do not necessarily represent databases students will likely encounter in their work. Although Wagner et al. (2003), Jukic and Gray (2008b), and Yue (2013) argue for using either more complex database structures or datasets, the negative implications of such environment parameters have not been scrutinized. Therefore, if more complex databases indeed have no discernible downsides to learning, educators should utilize them in teaching SQL to better prepare students for their future work. However, if such databases present a threat to the learning process, caution should be exercised when choosing exercise database complexity.

Assumption 4. Understanding more and more of the aspects that affect query formulation is an important step towards making an informed decision regarding which of those aspects should be mitigated and how.

Fourth, the effects of different environment parameters in query formulation success rates may be explained by different cognitive explanations presented by Smelcer (1995), but only after these effects, or lack thereof, have been demonstrated. Furthermore, in order to decide whether or not some environment parameter is beneficial to learning, we as educators should understand these effects. Finally, if some negative aspects can be mitigated with relative ease, these preceding steps are useful in developing such solutions. Next, we discuss how the research was conducted on a general level.

Chronologically, we first set out to find research gaps in SQL education. This was not a systematic process, but revealed that the rather intuitive consideration of the effects of database complexity on query formulation success rates had not received scientific attention (Article PIV). However, before studying success rates

in query formulation, we needed to understand what success is, i.e., what constitutes in making a query correct or incorrect. DBMS specific implementations of the SQL languages presented a threat in this regard, as prior studies usually utilized a specific DBMS to study errors. Therefore, we needed a DBMS independent error definitions, and used the SQL standard to guide our error categorization framework (Article PII). During the course of the error categorization, it became apparent that not all errors committed are equal, as some are usually corrected by query writers, while others halt the query writing process, resulting in a failure in query formulation. Rather than merely understanding *what* is incorrect (Article PII), we wanted to understand some aspects of *why* query formulation is unsuccessful (Article PIII). Finally, based on the preliminary literature review, we conducted a systematic mapping study to compile the varied SQL teaching practices proposed in scientific literature, and to propose future research topics for the educational SQL research field (PI). It is worth noting that the included articles are not presented in a chronological order. The primary research questions for the included articles are presented below.

RQ1: What SQL teaching practices have been proposed in scientific literature? (Article PI)

RQ2: What types of errors novices commit in SQL query formulation? (Article PII)

RQ3: What types of errors are persistent, i.e., have a tendency to remain unfixed? (Article PIII)

RQ4: What are the effects of database complexity on query formulation? (Article PIV)

Additionally, we developed a query planning notation to mitigate the root causes for query formulation errors in the translation process as proposed by Smelcer (1995) (Article PV). As the notation was not among intended results of our line of research, it may be considered a by-product rather than an answer to an established research question.

3.2 Contributions

3.2.1 Article PI

Toni Taipalus and Ville Seppänen. SQL education: A systematic mapping study and future research agenda. *ACM Transactions on Computing Education* 20(3), Article 20, 2020.

Aim Using systematic mapping as proposed in Petersen et al. (2008, 2015), we set out to collect SQL teaching practices and considerations from scientific literature. As the preliminary, non-systematic literature review for Article PII revealed

numerous teaching considerations, we deemed a systematic map using a wider viewpoint a proper fit to explore the topic. A systematic map was also a natural fit, as to our knowledge there are no systematic literature reviews or mapping studies regarding SQL education.

Main contributions We present a systematic map of SQL education research publication fora and publications by year over a time frame of 30 years. We categorize SQL education research into six topics, namely *(i)* student errors in query formulation; *(ii)* characteristics and presentation of the exercise database; *(iii)* specific and *(iv)* non-specific teaching approach suggestions; *(v)* patterns and visualization; and *(vi)* easing teacher workload. Additionally, we list 66 teaching considerations from the selected 89 primary studies into these six topics, and present a future research agenda based on the insights from the mapping process.

Minor contributions We adjusted and particularized the research type facet classification scheme proposed in Wieringa et al. (2005) and Petersen et al. (2008) for systematic mapping studies. The adjustment was done in order to more unambiguously map educational research into research type facets. This was not done according to one scientific method, but rather based on the experiences of the authors.

Author's contribution The author formulated the research goals and aims, collected and analyzed research data, and wrote the initial draft of the paper. The co-authors participated in validation and discussion of the data analysis, and contributed to reviewing and editing the text. The final version of the paper was created by collaboration of all the authors.

3.2.2 Article PII

Toni Taipalus, Mikko Siponen, and Tero Vartiainen. Errors and complications in SQL query formulation. *ACM Transactions on Computing Education* 18(3), Article 15, 2018.

Aim Utilizing both directed and conventional content analyses as described in Hsieh and Shannon (2005), we set out to formulate an SQL query error taxonomy based on SQL query data from a cohort of students. Some SQL errors had been identified in prior studies, yet error categorization had not been the goal of those studies.

Main contributions We present a DBMS independent error categorization of syntax, semantic, and logical errors, and complications in SQL query formulation. The error categorization is presented as a taxonomy of three levels, i.e., the aforementioned four error classes, which are comprised of a total of 18 error categories, which are further comprised of 105 different errors. Furthermore, we present descriptive statistics on the frequency of the 18 errors types throughout the 15 SQL exercises.

Minor contributions Based on our teaching experiences, we structured a query concept framework of 15 exercises and 19 query concepts, which was then used to guide our data collection. The query concept framework was formulated before data collection.

Author's contribution The author formulated the research goals and aims, and the query concept framework, collected, coded and analyzed research data, and wrote the initial draft of the paper. The co-authors supervised the research process, participated in validation and discussion of the data analysis, and contributed to reviewing and editing the text. The final version of the paper was created by collaboration of all the authors.

3.2.3 Article PIII

Toni Taipalus and Piia Perälä. What to expect and what to focus on in SQL query teaching. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19).* ACM, New York, 198-203, 2019.

Aim Utilizing the query concept framework and error taxonomy presented previously in Article PII, we proceed to focus on persistent (i.e., never corrected) errors in SQL query formulation. Using queries from three student cohorts, we separate common errors from common persistent errors, and describe what types of errors certain query concepts seem to invite.

Main contributions We present descriptive statistics on error categories with high relative frequencies. High relative frequencies indicate errors that are difficult for students to fix, as opposed to high frequencies presented in Article PII, which indicate errors that are common, but not necessarily difficult to fix. Using the highest level of the error taxonomy, we also present estimated means for each error class for each exercise, revealing that logical errors are the most prominent class of persistent errors in all exercises, regardless of the query concepts tested.

Minor contributions Abstracted from the data analysis, we present an estimation of which query concepts invite which types of errors. This estimation does not cover all of the 19 tested query concepts, as most concepts display no clear patterns in relation to any error types.

Author's contribution The author formulated the research goals and aims, collected and coded research data, and wrote the initial draft of the paper. The co-authors conducted the data analysis, and contributed to reviewing and editing the text. The final version of the paper was created by collaboration of all the authors.

3.2.4 Article PIV

Toni Taipalus. The effects of database complexity on SQL query formulation. *Journal of Systems and Software* 165, Article 110576, 2020.

Aim Utilizing the query concept framework and error taxonomy presented earlier in Article PII, and the insights on persistent errors in Article PIII, we proceed to study how logical complexity affects query formulation success rates. Although prior scientific literature suggests positive effects on student motivation and preparation for future work, the potential negative aspects of more complex databases have not been studied.

Main contributions Based on analyses of queries executed against three databases of different logical complexity, we present that as database complexity increases, *(i)* query formulation success rates decrease; *(ii)* the number of complications increase; *(iii)* the number of syntax errors increase, but not with a statistically significant effect; *(iv)* the number of semantic errors show no statistically significant difference; and *(v)* the number of logical errors display no distinct pattern between databases of different logical complexity.

Author's contribution The author was the sole author of the paper.

3.2.5 Article PV

Toni Taipalus. Teaching tip: A notation for planning SQL queries. *Journal of Information Systems Education* 30(3), 160-166, 2019.

Aim Inspired by the notion that human working memory constraints, procedural fixedness, and lack of cues hinder query formulation, especially with more complex queries, we proceed to emphasize the importance of *a priori* planning by developing an SQL query planning notation. To the best of our knowledge, this was the only SQL query planning notation published in a scientific forum, although tangential work has been published before (Koutrika et al., 2010; Danaparamita and Gatterbauer, 2011) and simultaneously (Sundin and Cutts, 2019).

Main contributions We present an SQL query planning notation inspired by graph theory. The notation covers a wide range of query concepts such as selection, restriction, joins, grouping and sorting. Additionally, the notation may be applied to complex update and delete statements with minor alterations.

Author's contribution The author was the sole author of the paper.

4 DISCUSSION

In this chapter, we first discuss the practical implications of the articles as a whole, as opposed to individual implications discussed in the articles proper. Next, we consider the limitations of this line of research, argue for the chosen research methods, discuss threats to validity, and explain some ethical considerations. Finally, we present potential future research directions.

4.1 Practical implications

The line of research presented in Articles PII through PIV contribute to increased understanding of errors in SQL learning, and shed light on the effects of database complexity on query formulation errors. For research, we presented an SQL error categorization which was designed to be DBMS independent. This categorization is a contribution to any SQL research studying errors, and has the potential benefit of making results of different studies that utilize the categorization comparable with each other. Additionally, we presented a query concept framework to be utilized in courses and other research settings. The error frequencies presented in Article PII and investigated in more detail in Article PIII also provide guidelines for teachers who want to mitigate natural language ambiguities in data demands by showing students correct result tables. These frequencies can be utilized in the design of exercise database data in a way that the data account for most common errors. As presented in Article PIII, certain query concepts invite certain types of errors, and exercise database data should be designed so that queries with the most common errors return a different result table than the correct query. Accounting for all possible errors, however, is not feasible, and the most common errors should be considered first. As these studies show, logical errors are the most common class of errors, and the expected errors may be used to guide the development of intelligent tutoring systems and the feedback they provide to the student about incorrect queries. Regarding SQL teaching focus, we presented errors which are common, errors which are persistent, and errors

which are both. As errors which are common but not persistent, students are able to fix on their own, teachers should focus on errors which are both common and persistent when designing exercise database data.

In terms of research, many factors affecting query formulation success rates have been studied, and in Article PIV, we present evidence on the negative effects of increased database complexity on query formulation. In terms of teaching, higher student engagement remains an important factor when more realistic databases are used. Based on the preceding studies and results in Article PIV, we suggest that there might be at least two directions worth studying, if we as teachers want to both use realistic exercise databases, and see high query formulation success rates. First, we might be able to convince students of the fact that a more realistic database is not necessarily more complex. Second, we could use method to try to mitigate the causes of procedural fixedness, working memory constraints, and lack of cues while still utilizing complex exercise databases. As a potential solution for the second consideration, we describe a planning notation in Article PV, which can be utilized by teachers simultaneously to teaching SQL.

Although educational SQL research is scarce when compared to programming languages, SQL has been a part of research and teaching for a relatively long time, and time, rather than research focus, has resulted in a number of scientific articles on educational SQL. In article PI, we aggregate a body of knowledge of teaching practices to a relatively accessible format for teachers to utilize. For research, we provide a systematic mapping of SQL education literature. Furthermore, we summarize and discuss educational SQL research on a high level, providing an overview of potential research dearths and trending research directions.

4.2 Limitations and threats to validity

According to the SQL standard, SQL is an extensive language. This line of research, particularly Articles PII through PV, covers only a relatively small subset of the SQL language, although according to different curricula guidelines, this subset is a salient part of SQL. However, even though this line of research is limited to data retrieval, which also is only a part of SQL, not all data retrieval concepts and keywords are addressed in the articles. This is by design to limit the scope of the studies.

Data analysed in Articles PII through PIV originates from participants who took one single course and were taught SQL by one single teacher. This may have influenced the diversity of different errors reported in Article PII, and participants taught by another teacher could have committed errors not found in that study. Although the error categorization reported in Article PII was designed to be DBMS independent, SQLite was used in data collection. As mentioned in Article PIII, syntax error discovery by a single DBMS is unreliable in regards to error categorization. Although using a single DBMS in the research setting in Article

PIV is arguably a necessity to mitigate the unwanted effects of different DBMSs on query formulation, using a single DBMS in Article PII to formulate the error taxonomy may limit the errors discovered, and skew error frequencies. We would detail that SQLite is relatively lenient in syntax error detection, and does not enforce strict grouping. This has possibly increased the number of reported syntax errors, grouping related syntax errors in particular, in Articles PIII and PIV. Furthermore, an iota of caution is advised when interpreting the DBMS independence in the error categorization, as no categorization is thoroughly DBMS independent, because the standard does not always unambiguously dictate what is an error.

We discuss many of the threats to validity concerning control variables in article PIV that extend to the preceding Articles PII and PIII as well. Additionally, there are some considerations regarding the two types of qualitative content analysis used but not discussed in Article PII. According to Hsieh and Shannon (2005), directed content analysis "presents challenges to the naturalistic paradigm" with strong researcher bias originating from existing theory. Therefore, some syntax errors, semantic errors, and complications that we recognized as previously discovered errors in Article PII may indeed have exhibited novel types of errors we failed to identify due to our preconceptions. As summarized in Hsieh and Shannon (2005), one of the main challenges in conventional content analysis is that researchers miss key information. Although we discussed that prolonged exposure and closeness to the teaching process should mitigate this in Article PII, it is possible that we have missed some logical errors in our data coding and analysis. With this in mind, the main threat to validity in Articles PII through PIV is that the query formulation error coding and taxonomy development was effectively based on a single researcher's qualitative analysis, however close their involvement to the teaching and data collection processes.

Choosing an appropriate statistical method is arguably an important factor for validity. Regarding the methods used in Articles PIII and PIV, we feel necessary to expand the discussion on the nature of data analysed to justify our chosen methods. In Article PIII, we estimated means of errors by error class per exercise using negative binomial regression. As our dependent variable was the number of errors committed by a participant by error class by exercise, we considered the scale of our dependent variable count. Our two independent variables (error class and exercise) were measured on nominal scale, and we proceeded by further investigating the suitability of Poisson regression. However, we found that our data were overdispersed, and therefore considered negative binomial regression, according to which variance and mean are not equivalent.

In Article PIV, we used both chi-square test of homogeneity and Kruskal-Wallis H test to determine if there were statistically significant differences between query formulation success rates and database complexity, and between numbers of different errors belonging to an error class and database complexity. We chose chi-square test of homogeneity, because *(i)* our dependent variable was measured on dichotomous scale (success or failure in formulating a query), *(ii)* we had one independent variable and it had three categorical and indepen-

dent groups (simple, semi-complex, and complex database), *(iii)* we had independence of observations by design, and *(iv)* we had a sufficiently large sample size. We chose the Kruskal-Wallis H test, because *(i)* we had one continuous dependent variable (number of errors committed by a participant within an error class), *(ii)* we had one independent variable that had three categorical and independent groups (simple, semi-complex, and complex database), *(iii)* we had independence of observations, *(iv)* data were not normally distributed for each group of the independent variable, and *(v)* groups sizes were not equal. We first considered running a one-way ANOVA, but due to points *(iv)* and *(v)*, we chose a non-parametric test instead, as to our understanding, Kruskal-Wallis H test is not affected by outliers to the same degree as one-way ANOVA. For the Kruskal-Wallis H tests, post hoc analysis (pairwise comparisons) were performed using Dunn's (1964) procedure with a Bonferroni correction for multiple comparisons. Because the group sizes were not equal, and because the homogeneity of variance was violated, we chose to use this particular post hoc analysis.

Article PIV only considers query formulation success rate as a measure of successful query formulation. In hindsight, it would have been more informative to also measure query formulation time, number of attempts, and participant confidence as done by Chan et al. (1999) and Borthick et al. (2001a) in their respective studies. However, measuring query formulation time reliably would have required a more controlled research setting altogether.

Key assumptions that guided the development of the query planning notation in Article PV lean on the results of Smelcer (1995). However, it is unclear how Smelcer (1995) mapped different query formulation errors to their particular cognitive explanations. Additionally, Smelcer's (1995) analysis is based on queries from 17 participants, and, as discussed in article PI, new versions of the SQL standard have since introduced new and alternative features to the language. Finally, although our experiences and informal feedback regarding the notation have been positive, Article PV is effectively a solution proposal without empirical evidence to support the usefulness of the notation.

4.3 Ethical considerations

For Articles PI and PV, no personal data were used due to the nature of the research. For Articles PII, PIII and PIV, we followed current scientific recommendations regarding data privacy and ethical considerations regarding research participation. In summary, the participants were informed of what data were collected and how it would be used in our research. The participants were informed that the data would only be used for the purposes of our research, and not passed to third parties outside the scope of the researchers listed as authors of the papers. The participants were given the opportunity to participate or to opt out with no advantages or disadvantages regarding, for example, course grade. Besides user names, the data contained no personal information. The data were anonymized

before the analyses, and reported in a form of statistical information, where the identification of individuals was not possible.

4.4 Future agenda

Based on the systematic mapping, we presented future research avenues for SQL education in general in article PI. Additionally, for a more comprehensive SQL error categorization, our query concept framework could be further developed to encompass more of data retrieval concepts. One such extension is already available (Migler and Dekhtyar, 2020), yet it remains open whether these new query concepts introduce novel errors. Amongst all our student cohorts, logical errors were the most prominent, and this suggests that logical errors should be further investigated. Especially regarding logical errors, the usefulness of our query planning notation remains an open question, and it would be beneficial to investigate whether the notation mitigates the causes behind query formulation failure it was designed to mitigate.

The effects of dataset complexity on query formulation have not been studied. This is closely related to the notion of natural and unnatural learning environments presented in Articles PII and PIV; what kind of learning environment in terms of database and dataset complexity, and data demand ambiguity is the most beneficial to students? In this regard, we feel that SQL education is ready to more systematically start moving in the direction of explaining *why* and proposing solutions based on scientific evidence on query formulation success factors.

5 CONCLUSIONS

In this doctoral dissertation, we answered our four research questions. We focused on extending prior knowledge on errors in query formulation, namely, what types of errors are committed, what types of errors are difficult to fix, and what factors in the learning environment cause errors which are difficult to fix. Additionally, we presented a systematic map of educational SQL research and teaching practices proposed, a future research agenda, and a notation for planning more complex SQL queries.

Our results indicate that there are numerous different kinds of errors committed in query formulation, and the frequencies of different errors are highly dependent on the type of query being formulated. While some errors were frequent, we also considered errors which are difficult to fix, and halt the query formulation process. On a general level, logical errors were most prominent in query formulation, but also statistically the most difficult to fix. These results support previously suggested cognitive explanations behind query formulation errors, namely human working memory constraints, fixed query writing habits, and natural language interpretation difficulties. Our results also revealed that an increase in database complexity results in a decrease in query formulation success rates, indicating that complex databases, although realistic, may not be the most suitable environments for practicing SQL.

Additionally, we collected SQL teaching practices from scientific literature, and systematically mapped SQL education research over a time frame of 30 years. This listing of teaching practices may be directly utilized in SQL education. Finally, designed around the suggested cognitive explanations behind query formulation errors, we presented a query planning notation especially for more complex SQL retrieval statements. The notation can be used in both teaching and industry to mitigate logical errors in query formulation.

YHTEENVETO (SUMMARY IN FINNISH)

Haemme päivittäin tietoa tietokannoista, vaikka tietokannan olemassaolo ei välttämättä olekaan meille kuluttajille ilmiselvää. Jokainen verkon hakukoneella tehty avainsanahaku, jokainen musiikkikappale tai elokuva suoratoistopalvelusta ja jokainen verkkolehdessä luettu artikkeli saa aikaan kyselyn tietokantaan. Kyselyn kirjoittaminen on tavallisesti yleistetty meille kuluttajille vaivattomaksi: pelkkä avainsanan kirjoittaminen riittää. Ohjelmistokehittäjän on kuitenkin järjestelmää kehittäessään täytynyt muuttaa kysely sellaiselle formaalille kielelle, jota myös ohjelmisto ymmärtää. Yleisin näistä ns. kyselykielistä on SQL-kieli. Jotta kuluttajina saisimme järjestelmästä juuri sen tiedon mitä haluamme, on tärkeää että ohjelmistokehittäjä on kirjoittanut vastaavan lauseen kyselykielellä virheettömästi. SQL-kieltä tarkastelevassa tieteellisessä kirjallisuudessa ei ole kuitenkaan käsitelty laaja-alaisesti niitä virheitä, joita kyselyiden muodostamisessa tavallisesti tehdään. Lisäksi kaikkien keskeisten tietokantaan liittyvien aspektien vaikutusta kyselyiden kirjoittamiseen ei ole tutkittu. Tässä väitöskirjassa *(i)* kartoitettiin systemaattisesti SQL-kielen opetukseen liittyvä tieteellinen tutkimus 30 vuoden ajalta ja koottiin yhteen tieteellisessä kirjallisuudessa esitetyt tavat opettaa SQL-kieltä, *(ii)* muodostettiin laaja-alainen SQL-kyselyiden kirjoittamisessa tapahtuvien virheiden taksonomia, *(iii)* eroteltiin virheet, jotka kyselyn kirjoittaja pystyy tavallisesti havaitsemaan ja korjaamaan sellaisista virheistä, jotka tavallisesti jäävät lopulliseen kyselyyn, *(iv)* havaittiin, että tietokannan rakenteen monimutkaisuus on verrannollinen sellaisten virheiden määrään, joita kyselyn kirjoittaja ei tavallisesti pysty havaitsemaan tai korjaamaan, ja *(v)* muodostettiin monimutkaisempien SQL-kyselyiden suunnitteluun notaatio, jonka tarkoituksena on kyselyä kirjoitettaessa tehtyjen virheiden lukumäärän vähentäminen.

REFERENCES

ACM/IEEE 2013. Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. doi:10.1145/2534860. (999133).

ACM/IEEE 2015. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. ⟨URL:https://dl.acm.org/citation.cfm?id=2965631⟩.

AL-Salmi, A. 2018. A web-based semi-automatic assessment tool for formulating basic SQL statements: Point-and-click interaction method. In Proceedings of the 10th International Conference on Computer Supported Education (CSEDU), Vol. 1, 191-198. doi:doi.org/10.5220/0006671501910198.

Abelló, A., Burgués, X., Casany, M. J., Martín, C., Quer, C., Rodríguez, M. E., Romero, Ó. & Urpí, T. 2016. A software tool for e-assessment of relational database skills. International Journal of Engineering Education 32 (3A), 1289–1312. ⟨URL:http://hdl.handle.net/2117/89668⟩.

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, 401–406. doi:10.1145/2839509.2844640.

Ahadi, A., Prior, J., Behbood, V. & Lister, R. 2015. A Quantitative Study of the Relative Difficulty for Novices of Writing Seven Different Types of SQL Queries. In Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE). New York, New York, USA: ACM Press, 201–206. doi:10.1145/2729094.2742620.

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, 272–277. doi:10.1145/2899415.2899464.

Ashkanasy, N., Bowen, P. L., Rohde, F. H. & Wu, C. Y. A. 2007. The effects of user characteristics on query performance in the presence of information request ambiguity. Journal of Information Systems 21 (1), 53-82. doi:10.2308/jis.2007.21.1.53.

Axelsen, M., Borthick, A. F. & Bowen, P. L. 2001. A model for and the effects of information request ambiguity on end-user query performance. In ICIS 2001 Proceedings, 68. ⟨URL:http://aisel.aisnet.org/icis2001/68⟩.

Borthick, A., Bowen, P. L., Jones, D. R. & Tse, M. H. K. 2001a. The effects of information request ambiguity and construct incongruence on query development. Decision Support Systems 32 (1), 3–25. doi:10.1016/s0167-9236(01)00097-5.

Borthick, A., Bowen, P., Liew, S. & Rohde, F. 2001b. The effects of normalization on end-user query errors: An experimental evaluation. International Journal of Accounting Information Systems 2 (4), 195 - 221. doi:https://doi.org/10.1016/S1467-0895(01)00023-9.

Bowen, P. L., Rohde, F. H. & Basford, J. 2004. Ex ante evaluations of alternate data structures for end user queries. Journal of Database Management 15 (4), 45–70. doi:10.4018/jdm.2004100103.

Brass, S. & Goldberg, C. 2006. Semantic errors in SQL queries: A quite complete list. Journal of Systems and Software 79 (5), 630–644. doi:10.1016/j.jss.2005.06.028.

Brusilovsky, P., Sosnovsky, S., Yudelson, M. V., Lee, D. H., Zadorozhny, V. & Zhou, X. 2010. Learning SQL programming with interactive tools: From integration to personalization. ACM Transactions on Computing Education 9 (4), 19:1–19:15. doi:10.1145/1656255.1656257.

Buitendijk, R. B. 1988. Logical errors in database SQL retrieval queries. Computer Science in Economics and Management 1 (2), 79–96. doi:10.1007/bf00427157.

Casterella, G. I. & Vijayasarathy, L. 2013. An Experimental Investigation of Complexity in Database Query Formulation Tasks. Journal of Information Systems Education 24 (3), 211–221. ⟨URL:http://jise.org/Volume24/24-3/pdf/Vol24-3pg211.pdf⟩.

Casterella, G. I. & Vijayasarathy, L. 2019. Query Structure and Data Model Mapping Errors in Information Retrieval Tasks. Journal of Information Systems Education 30 (3), 178–190. ⟨URL:http://jise.org/Volume30/n3/JISEv30n3p178.pdf⟩.

Chamberlin, D. D. 2012. Early history of SQL. IEEE Annals of the History of Computing 34 (4), 78-82. doi:10.1109/MAHC.2012.61.

Chan, H. C., Tan, B. C. & Wei, K.-K. 1999. Three important determinants of user performance for database retrieval. International Journal of Human-Computer Studies 51 (5), 895–918. doi:10.1006/ijhc.1999.0272.

Chen, P. P.-S. 1976. The entity-relationship model - toward a unified view of data. ACM Transactions on Database Systems 1 (1), 9–36. doi:10.1145/320434.320440.

Cleve, A., Gobert, M., Meurice, L., Maes, J. & Weber, J. 2015. Understanding database schema evolution: A case study. Science of Computer Programming 97 (P1), 113–121. doi:10.1016/j.scico.2013.11.025.

Codd, E. F. 1970. A relational model of data for large shared data banks. Communications of the ACM 13 (6), 377–387. doi:10.1145/362384.362685.

Codd, E. F. 1971a. A data base sublanguage founded on the relational calculus. In Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control - SIGFIDET '71. ACM Press. doi:10.1145/1734714.1734718.

Codd, E. F. 1971b. Further normalization of the data base relational model. IBM Research Report, San Jose, California RJ909.

Codd, E. F. 1972. Relational completeness of data base sublanguages. IBM Research Report RJ987. (republished on "ACM SIGMOD Anthology").

Connolly, T. & Begg, C. 2015. Database Systems (6th. ed.). Pearson.

Danaparamita, J. & Gatterbauer, W. 2011. Queryviz: Helping users understand sql queries and their patterns. In Proceedings of the 14th International Conference on Extending Database Technology. New York, NY, USA: ACM. EDBT/ICDT '11, 558–561. doi:10.1145/1951365.1951440.

Date, C. J. 1983. Critique of the SQL database language. SIGMOD Record 14 (3). doi:10.1145/984549.984551.

Davis, J. S. 1990. Experimental investigation of the utility of data structure and E-R diagrams in database query. International Journal of Man-Machine Studies 32 (4), 449 - 459. doi:10.1016/S0020-7373(05)80142-7.

Dunn, O. J. 1964. Multiple comparisons using rank sums. Technometrics 6 (3), 241–252. ⟨URL:https://www.jstor.org/stable/1266041⟩.

Eisenberg, A., Melton, J., Kulkarni, K., Michels, J.-E. & Zemke, F. 2004. SQL:2003 has been published. ACM SIGMOD Record 33 (1), 119. doi:10.1145/974121.974142.

Eisenberg, A. & Melton, J. 1999. SQL:1999, formerly known as SQL3. SIGMOD Record 28 (1), 131–138. doi:10.1145/309844.310075.

Elmasri, R. & Navathe, S. B. 2016. Fundamentals of Database Systems (7th. ed.). Pearson.

Garner, P. & Mariani, J. A. 2015. Learning SQL in steps. Journal of Systemics, Cybernetics and Informatics 13 (4), 19–24.

Gregor, S. 2006. The nature of theory in information systems. MIS Quarterly 30 (3), 611. doi:10.2307/25148742.

Gudivada, V. N., Nandigam, J. & Tao, Y. 2007. Enhancing student learning in database courses with large data sets. In 2007 37th annual Frontiers in Education conference (FIE). IEEE. doi:10.1109/fie.2007.4418135.

Hsieh, H.-F. & Shannon, S. E. 2005. Three approaches to qualitative content analysis. Qualitative Health Research 15 (9), 1277-1288. doi:10.1177/1049732305276687. (PMID: 16204405).

ISO/IEC 2016a. ISO/IEC 9075-1:2016 - SQL Part 1: Framework. ⟨URL:https://www.iso.org/standard/63555.html⟩.

ISO/IEC 2016b. ISO/IEC 9075-2:2016 - SQL - Part 2: Foundation. ⟨URL:https://www.iso.org/standard/63556.html⟩.

Jukic, N. & Gray, P. 2008a. Teradata university network: A no cost web-portal for teaching database, data warehousing, and data-related subjects. Journal of Information Systems Education 19 (4), 395-402. ⟨URL:http://jise.org/Volume19/n4/JISEv19n4p395.html⟩.

Jukic, N. & Gray, P. 2008b. Using real data to invigorate student learning. SIGCSE Bull. 40 (2), 6–10. doi:10.1145/1383602.1383604.

Kenny, C. & Pahl, C. 2005. Automated tutoring for a database skills training environment. In Proceedings of the 36th ACM Technical Symposium on Computer Science Education (SIGCSE). New York, NY, USA: ACM. SIGCSE '05, 58–62. doi:10.1145/1047344.1047377.

Koutrika, G., Simitsis, A. & Ioannidis, Y. E. 2010. Explaining structured queries in natural language. In 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010). IEEE. doi:10.1109/icde.2010.5447824.

Kroenke, D. & Auer, D. J. 2016. Database Processing: Fundamentals, Design, and Implementation (14th edition). Pearson Education.

Kulkarni, K. & Michels, J.-E. 2012. Temporal features in SQL:2011. ACM SIGMOD Record 41 (3), 34. doi:10.1145/2380776.2380786.

Leitheiser, R. L. & March, S. T. 1996. The influence of database structure representation on database system learning and use. Journal of Management Information Systems 12 (4), 187–213. doi:10.1080/07421222.1996.11518106.

Li, F. & Jagadish, H. V. 2014. Constructing an interactive natural language interface for relational databases. Proceedings of the VLDB Endowment 8 (1), 73–84. doi:10.14778/2735461.2735468.

Metcalfe, J. 2017. Learning from errors. Annual Review of Psychology 68 (1), 465-489. doi:10.1146/annurev-psych-010416-044022. (PMID: 27648988).

Migler, A. & Dekhtyar, A. 2020. Mapping the SQL learning process in introductory database courses (in press). In Proceedings of the 51th ACM Technical Symposium on Computer Science Education (SIGCSE). ACM. SIGCSE '20.

Mitrovic, A. 2003. An intelligent SQL tutor on the web. International Journal of Artificial Intelligence in Education 13 (2-4), 173-197.

Mitrovic, A. 1998. Learning SQL with a computerized tutor. In Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education. New York, NY, USA: ACM. SIGCSE '98, 307–311. doi:10.1145/273133.274318.

Nalintippayawong, S., Atchariyachanvanich, K. & Julavanich, T. 2017. DBLearn: Adaptive e-learning for practical database course - an integrated architecture approach. In 2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 109–114. doi:10.1109/SNPD.2017.8022708.

Ogden, W. C., Korenstein, R. & Smelcer, J. B. 1986. An Intelligent Front-End for SQL. IBM General Products Division.

Ortiz, J., Dietrich, S. W. & Chaudhari, M. B. 2012. Learning from database performance benchmarks. Journal of Computing Sciences in Colleges 27 (4), 151–158. ⟨URL:http://dl.acm.org/citation.cfm?id=2167431.2167457⟩.

Permpool, T., Nalintippayawong, S. & Atchariyachanvanich, K. 2019. Interactive SQL learning tool with automated grading using MySQL Sandbox. In Interactive SQL Learning Tool with Automated Grading using MySQL Sandbox, 928-932. doi:10.1109/IEA.2019.8715175.

Petersen, K., Feldt, R., Mujtaba, S. & Mattsson, M. 2008. Systematic mapping studies in software engineering. In Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering. Swindon, UK: BCS Learning & Development Ltd. EASE'08, 68–77.

Petersen, K., Vakkalanka, S. & Kuzniarz, L. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology 64, 1 - 18. doi:10.1016/j.infsof.2015.03.007.

Prior, J. R. 2014. AsseSQL: An online, browser-based SQL skills assessment tool. In Proceedings of the 2014 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE). New York, NY, USA: ACM, 327–327. doi:10.1145/2591708.2602682.

Randolph, G. B. 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). New York, NY, USA: ACM, 234–236. doi:10.1145/947121.947174.

Reilly, C. F. 2018. Experience with active learning and formative feedback for a SQL unit. In 2018 IEEE Frontiers in Education Conference (FIE). IEEE. doi:10.1109/fie.2018.8659173.

Reisner, P. 1977. Use of psychological experimentation as an aid to development of a query language. IEEE Transactions on Software Engineering SE-3 (3), 218–229. doi:10.1109/tse.1977.231131.

Reisner, P. 1988. Query languages. In M. Helander (Ed.) Handbook of Human-Computer Interaction. New York: Elsevier, 257–280.

Reisner, P. 1981. Human factors studies of database query languages: A survey and assessment. ACM Computing Surveys 13 (1), 13–31. doi:10.1145/356835.356837.

Robb, D. A., Bowen, P. L., Borthick, A. F. & Rohde, F. H. 2012. Improving new users' query performance. Journal of Data and Information Quality 3 (4), 1–22. doi:10.1145/2348828.2348829. ⟨URL:https://doi.org/10.1145\%2F2348828.2348829⟩.

Siau, K. L., Chan, H. C. & Wei, K. K. 2004. Effects of query complexity and learning on novice user query performance with conceptual and logical database interfaces. IEEE Trans. Syst., Man, and Cybernetics - Part A: Syst. and Hum. 34 (2), 276–281. doi:10.1109/TSMCA.2003.820581.

Smelcer, J. B. 1995. User errors in database query composition. International Journal of Human-Computer Studies 42 (4), 353–381. doi:10.1006/ijhc.1995.1017.

Smith, R. & Rixner, S. 2019. The error landscape: Characterizing the mistakes of novice programmers. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education - SIGCSE '19. ACM Press. doi:10.1145/3287324.3287394.

Sundin, L. & Cutts, Q. 2019. Is it feasible to teach query programming in three different languages in a single session?: A study on a pattern-oriented tutorial and cheat sheets. In Proceedings of the 1st ACM UK & Ireland Computing Education Research Conference. New York, NY, USA: ACM. UKICER, 7:1–7:7. doi:10.1145/3351287.3351293.

Topi, H., Kaiser, K. M., Sipior, J. C., Valacich, J. S., Nunamaker, Jr., J. F., de Vreede, G. J. & Wright, R. 2010. Curriculum Guidelines for Undergraduate Degree Programs in Information Systems. ⟨URL:https://dl.acm.org/citation.cfm?id=2593310⟩.

Wagner, P. J., Shoop, E. & Carlis, J. V. 2003. Using scientific data to teach a database systems course. In Proceedings of the 34th ACM Technical Symposium on Computer Science Education (SIGCSE). New York, NY, USA: ACM, 224–228. doi:10.1145/611892.611975.

Watson, H. J. & Hoffer, J. A. 2003. Teradata university network: A new resource for teaching large data bases and their applications. Communications of the Association for Information Systems 12. doi:10.17705/1cais.01209.

Watson, R. T. 2006. The essential skills of data modeling. Journal of Information Systems Education 17 (1), 39-42. ⟨URL:http://jise.org/Volume17/n1/JISEv17n1p39.pdf⟩.

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

Wieringa, R., Maiden, N., Mead, N. & Rolland, C. 2005. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requirements Engineering 11 (1), 102–107. doi:10.1007/s00766-005-0021-6.

Wilson, R. C., Shenhav, A., Straccia, M. & Cohen, J. D. 2019. The eighty five percent rule for optimal learning. Nature Communications 10 (1), 4646. doi:10.1038/s41467-019-12552-4.

Yue, K.-B. 2013. Using a semi-realistic database to support a database course. Journal of Information Systems Education 24 (4), 327-336. ⟨URL:http://jise.org/Volume24/n4/JISEv24n4p327.pdf⟩.

ORIGINAL PAPERS

PI

SQL EDUCATION: A SYSTEMATIC MAPPING STUDY AND FUTURE RESEARCH AGENDA

by

Toni Taipalus and Ville Seppänen 2020

ACM Transactions on Computing Education, 20(3), Article 20

SQL education: A systematic mapping study and future research agenda

TONI TAIPALUS and VILLE SEPPÄNEN*, University of Jyvaskyla, Finland

Structured Query Language (SQL) skills are crucial in software engineering and computer science. However, teaching SQL effectively requires both pedagogical skill and considerable knowledge of the language. Educators and scholars have proposed numerous considerations for the betterment of SQL education, yet these considerations may be too numerous and scattered among different fora for educators to find and internalize, as no systematic mappings or literature reviews regarding SQL education have been conducted. The two main goals of this mapping study are to provide an overview of educational SQL research topics, research types and publication fora, and to collect and propagate SQL teaching practices for educators to utilize. Additionally, we present a short future research agenda based on insights from the mapping process. We conducted a systematic mapping study complemented by snowballing techniques to identify applicable primary studies. We classified the primary studies according to research type, and utilized directed content analysis to classify the primary studies by their topic. Out of our selected 89 primary studies, we identified six recurring topics: *(i)* student errors in query formulation; *(ii)* characteristics and presentation of the exercise database; *(iii)* specific and *(iv)* non-specific teaching approach suggestions; *(v)* patterns and visualization; and *(vi)* easing teacher workload. We list 66 teaching approaches the primary studies argued for (and in some cases against). For researchers, we provide a systematic map of educational SQL research, and future research agenda. For educators, we present an aggregated body of knowledge on teaching practices in SQL education over a time frame of 30 years. In conclusion, we suggest that replication studies, studies on advanced SQL concepts, and studies on aspects other than data retrieval are needed to further educational SQL research.

CCS Concepts: • **Information systems → Query languages**; • **Social and professional topics → Computing education**; **Computer science education**; **Software engineering education**.

Additional Key Words and Phrases: Structured Query Language (SQL), education, database, query language, student

ACM Reference Format:
Toni Taipalus and Ville Seppänen. 20xx. SQL education: A systematic mapping study and future research agenda. 1, 1 (May 20xx), 31 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Among the core topics in software engineering, computer science, and information systems curricula in higher education are databases and Structured Query Language (SQL) [62, 103, 105]. Since SQL is prevalent in database systems, SQL skills are also valued in the software industry, and consequently, teaching SQL effectively is essential in training future software professionals. However, teaching databases requires considerable subject knowledge in addition to pedagogical skill [102]. Additionally, there are several approaches to teaching SQL, and especially for an inexperienced database course teacher, differentiating between patterns (i.e., an effective teaching approach) and anti-patterns (i.e., what merely looks like an effective teaching approach) is difficult [89].

Authors' address: Toni Taipalus, toni.taipalus@jyu.fi; Ville Seppänen, ville.r.seppanen@jyu.fi, University of Jyvaskyla, Faculty of Information Technology, Agora, P.O. Box 35, Jyvaskyla, Finland, FI-40014.

Over the years of teaching SQL, we have come across numerous teaching practices proposed in scientific literature. Yet, these numerous studies may be overly onerous for educators to find and internalize, even superficially. Furthermore, it may not be clear whether a proposed teaching approach has been supported or contested by other scholars, as there are no systematic mapping studies or literature reviews regarding SQL education. To that end, and inspired by the propagation of ideas proposed by Bort et al. [13], we collected 66 teaching approaches from 89 primary studies over the course of 30 years with respective arguments for and against for educators to utilize. For researchers, we present a systematic mapping of SQL education with classifications regarding both the nature and the topic of the studies. Finally, based on the mapping, we present considerations for future research on *what* to study and *how* to study SQL education.

The rest of this study is structured as follows. In Section 2 we discuss the background of this study, i.e., SQL and its role in higher education. In Section 3 we describe the systematic mapping process, the research type and topic classifications, and threats to validity. In Sections 4 and 5 we present the results of our study, i.e., the systematic mapping and the lists of teaching practices, respectively. In Section 6 we discuss our results, and in Section 7 present the future research agenda. Finally, in Section 8 we conclude the study. In Appendices A, B, and C, we present the list of primary studies, a more detailed primary study classifications, and a list of the number of participants in each primary study, respectively.

2 BACKGROUND

2.1 SQL

The evolution of SQL (formerly SEQUEL, or Structured English Query Language [29, 30]) from the theoretical foundations of relationally complete query languages [38] to what SQL is today has been driven by both standardization, and vendors behind database management systems (DBMSs). The first available implementations of SQL were introduced in the late 1970s and early 1980s, and the first SQL standard, SQL-86 in 1986 by the standard groups ANSI and ISO [28]. Since SQL-86, the standard has received eight revisions, SQL:2016 being the latest. Each revision has added additional and alternative features. Despite its age, SQL remains the de facto query language in relational database management systems.

The SQL language features in the SQL standard are divided into *mandatory* (i.e., *core*) and *optional* features, and DBMSs implement both mandatory and optional features to varying degrees. Additionally, DBMSs may implement features differently to how they are described in the SQL standard, and most DBMSs have additional, vendor specific features [87]. Finally, SQL standard conformance testing of SQL implementations by the U.S. National Institute of Standards and Technology (NIST) has been discontinued in 1996, and a feature being *mandatory* is not a guarantee for a feature's implementation. Despite all these points, at least basic SQL statements remain portable with little or no modifications between DBMSs.

SQL is a versatile language, and allows users to retrieve, store, modify and delete data, create, modify and delete database objects (e.g, tables, columns, procedures, users), grant and revoke user privileges, and group statements into transactions. An SQL command is called a *statement*, and a statement which retrieves data from the database a *query*. SQL statements consist of *clauses* (e.g., SELECT, FROM, WHERE), which mainly contain database object names, *predicates* (e.g., LIKE, BETWEEN, EXISTS), *operators* (e.g., AND, OR, NOT), *quantifiers* (e.g., ANY, ALL, UNION), and *functions* (e.g, COUNT, SUM, AVG) [59, 60]. We call these collectively *concepts*, when it is not necessary to differentiate between, for example, clauses and predicates.

SQL is commonly divided into at least two sublanguages, Data Manipulation Language (DML, e.g., SELECT, INSERT, UPDATE, DELETE) and Data Definition Language (DDL, e.g, CREATE, ALTER, DROP) [28, 60]. Additionally, as the revisions of the SQL standard have introduced more features, two more sublanguages, Data Control Language (DCL, e.g., GRANT, REVOKE) and Transaction Control Language (TCL or TxCL, e.g., BEGIN, COMMIT, ROLLBACK), are sometimes discussed in literature. The origins of the sublanguage names DCL and TCL remain unclear, as these names are not explicitly mentioned in the SQL standard. Nevertheless, we have found this division into four sublanguages helpful and rather intuitive, and utilize it in this study.

2.2 SQL in Higher Education

SQL teaching in higher education is both long-lived and widespread. In the information technology subfields, SQL is explicitly mentioned in software engineering (SE) [103], computer science (CS) [62], and information systems (IS) undergraduate curricula guidelines [105], and additionally in areas such as business analytics [113]. These three information technology curricula guidelines expectedly overlap [64], and recommend SQL education on a relatively high level. SE guidelines recommend DML, DDL, and indexes, views, sequences, joins, and triggers in the context of database design. IS guidelines recommend DML, DDL, DCL, and transactions. CS guidelines provide the finest level of detail among the three, and recommend DDL, primary and foreign key attribute and schema definition, query formulation, UPDATE, integrity constraints, selection, projection, aggregate functions, GROUP BY, subqueries, division, stored procedures, and transaction control.

As these guidelines are merely guidelines, and presented at a high level, it is unclear how comprehensively and in depth SQL is covered in courses. Our impression, based on the primary studies and our teaching experience, is that basic DML is commonly discussed. Basic DML includes SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, INSERT INTO, VALUES, UPDATE and DELETE clauses, different types of joins, including the different variations of JOIN, certain predicates like IN, EXISTS, LIKE, BETWEEN, and IS NULL, and standard SQL aggregate functions MIN, MAX, AVG, SUM, and COUNT. However, advanced concepts like recursion, common table expressions, or derived tables are seldom included. It is unclear whether these advanced features are not widely known to educators and curricula designers, or have they been omitted from course contents by design. Basic DDL is commonly discussed, and includes CREATE, ALTER, and DROP statements on tables, views, and users (i.e., *roles*). Table creation includes column name and data type definitions, primary and foreign keys, and the CHECK constraint. However, more advanced concepts such as assertion, trigger, and procedure manipulations are rare. DCL is discussed to a lesser extent than DML and DCL, even though this sublanguage is relatively small and simple. If TCL is included in a course, it is often discussed with examples outside SQL, such as simple *read(a)*, *write(b)* [e.g., 39, p. 669ff.], even though transactions are often defined using SQL. After SQL, database education may focus on non-relational extensions [107], other data models, and data analytics [109].

2.3 Learning Context

In the aforementioned information technology curricula guidelines, SQL is not taught in isolation, but as a part of a database course. Before learning SQL, students need to know, at the very least, about the relational data model, and possibly the theoretical foundations of relational query languages. Nowadays, students learn SQL in digital, interactive environments using an exercise database [23]. This kind of environment can simply be a DBMS to which a student submits queries and receives output.

Alternatively, the environment may be a DBMS's SQL interface embedded in a web page, and the web page fitted with auxiliary elements, for example, a representation of the underlying database schema, a natural language request (i.e., *data demand*) to which a students must write an SQL equivalent, and the correct result table [101]. The database may be represented at the conceptual level as an Entity–Relationship (ER) diagram [37], or at the logical level as a database schema diagram [43]. For both levels, numerous additional or alternative notations can be utilized, for example, Unified Modeling Language class diagrams, enhanced/extended ER, and Logical Data Structures. When a student submits a query, the DBMS outputs either a result table, or an error message. Commonly, SQL errors have been divided into syntax and semantic errors [95]. More recently, research has identified the concept of complications, for example, tautologies and unnecessary elements in queries, which do not affect the result table but performance and readability [17]. Furthermore, semantic errors have been further divided into errors which are evident without knowledge of the underlying data demand, and errors which are only recognizable if the data demand is known [17]. The former kind of semantic errors are called *semantic*, and the latter *logical* [101].

Finally, more advanced environments may be used [19, 20]. These environments provide different additional features, for example, non-binary grading of queries [1], personalized feedback [77], and visualized query execution [49]. These more advanced environments are outside the scope this study.

3 RESEARCH METHOD

3.1 Research Questions

We divided our research questions into two categories. Research questions 1 and 2 are closely related to typical outcomes of the systematic mapping process in software engineering [83], and research question 3 and 4 related to the proposed SQL teaching practices, and to the nature of the proposing studies:

> *RQ1: In which fora is SQL education research published?* There are no publication fora which are specifically focused on SQL education, or even SQL in general. However, both computing education and database research fora in general are plentiful. Answers are presented in Section 4.1.
>
> *RQ2: What types of research are represented and to what extent?* Educational research is diverse by nature, and while some studies test clearly formulated hypotheses, others report opinions and experiences. We want to understand the SQL education landscape to identify potential dearths in research. Answers are presented in Section 4.2 and Appendix B.
>
> *RQ3: Which practices have been proposed for teaching SQL?* In addition to lectures, textbooks, and practical exercises, studies have identified and proposed practices (e.g., new tools, teaching methods, or increased emphasis on specific topics) to more effectively teach SQL. Answers are presented in Section 5.
>
> *RQ4: What kind of evidence is presented to support the proposed practices?* Whereas some SQL education studies report practices as results, others merely suggest different practices in their respective discussion sections. We want to differentiate between educated opinions and scientifically supported (or contested) propositions. Answers are presented in Section 5 and Appendix C.

3.2 Search Strategy

We searched four digital libraries without applying date or publication type restrictions: ACM Digital Library, IEEExplore, ISI Web of Science, and Scopus, which include arguably the most recognized computing science education research fora such as *ACM Transactions on Computing Education*, *Computer Science Education*, and the SIGCSE and ITiCSE

Table 1. Search strings

Database	Search string
ACM DL	("structured query language" OR SQL) AND (education OR teaching OR student OR students OR learning)
IEEExplore	(("structured query language" OR SQL) AND (education OR teaching OR student OR students OR learning))
Web of Science	TS=(("structured query language" OR SQL) AND (education OR teaching OR student OR students OR learning))
Scopus	TITLE-ABS-KEY(("structured query language" OR SQL) AND (education OR teaching OR student OR students OR learning))
AIS eLibrary	(SQL OR "structured query language")
JISE	SQL

conferences. Due to the pervasive nature of SQL in the information and communication technology (ICT) field, we also searched two information systems focused databases: AIS eLibrary and the database of *Journal of Information Systems Education* (JISE). Search strings are presented in Table 1.

The database searches yielded a total of 2,709 studies, 414 from ACM Digital Library, 646 from IEEExplore, 228 from Web of Science, 1,361 from Scopus, 46 from AIS eLibrary, and 14 from JISE. Additionally, we had 16 papers [4–6, 16, 17, 25, 40–42, 70, 71, 94, 98, 100, 101, 121] which we knew well enough to deem them suitable for closer criteria evaluation (cf. Table 2). The AIS eLibrary search was limited to peer-reviewed repositories. Both the AIS eLibrary and JISE search strings were more inclusive than the others due to the relative small size of the databases, although the former is only small if limited to peer-reviewed repositories. Google Scholar was considered, but a preliminary search returned too many results to inspect in a feasible timeframe.

The study selection process is illustrated in Fig. 1. The rectangles labeled A1 and A2 indicate the authors who performed the corresponding step. We performed backward snowballing (i.e., following reference lists to find relevant studies) [114] twice. Both authors studied the papers independently, and marked them both according to the research type facet classification (Table 3), and whether the paper should be included or excluded and why. We then compared our notes. In case of disagreement, we discussed until we reached a consensus on whether to include or exclude a paper, and how to classify the paper according to the research type facets. The comparison and discussion step was performed twice, and 89 primary studies were selected.

3.3 Study Selection

The searches yielded many papers concerning machine learning, SQL injection, and SQL learning environments. These papers were excluded because different learning environments were outside the scope of this study, *learning* in *machine learning* is not related closely enough to human learning in the context of this study, and education regarding SQL injection is more concerned with the design of the application program rather than SQL. To a lesser extent, the searches returned papers concerned with procedural extensions of SQL (e.g., T-SQL and PL/SQL), query optimization, NoSQL, data warehousing, and web development, all of which were excluded. We were relatively unanimous in our study inclusion/exclusion discussions, yet one study [106] was particularly difficult. The study explores the effects of task complexity and time limitations on query writing, and gives every implication that the query language

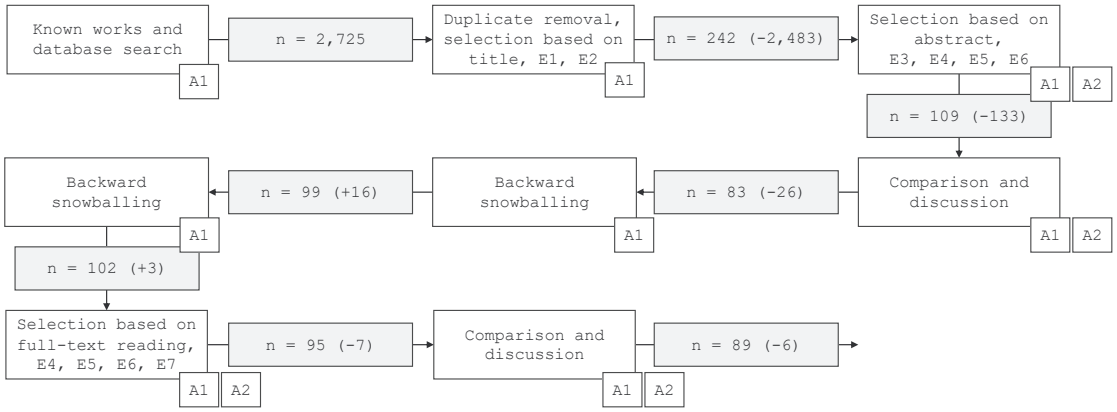


Fig. 1. Study selection process - A1 and A2 refer to the authors, *E* refers to an exclusion criterion described in Table 2, and *n* indicates the number of included papers

Table 2. Inclusion (I) and exclusion (E) criteria

ID	Criterion	Example studies
I1	present considerations on teaching or learning SQL	
I2	published online during the time frame 1989 to Sep. 2019	
E1	published in non-peer reviewed forum	[48]
E2	not written in English	
E3	full text we could not find or download	[36, 68, 85, 120, 123]
E4	do not mention SQL, or SQL is merely an example or a vehicle	[9, 11, 27, 55, 67, 92, 93, 96, 111]
E5	concerned with SQL alternatives rather than teaching SQL	[22, 26, 31, 33, 35, 81]
E6	focus on describing an SQL learning environment	[2, 19, 20, 76, 78, 82, 86]
E7	lack sufficient detail to suggest a detailed teaching approach	[3, 52, 57, 69, 79, 116–118, 122]

under study is indeed SQL. However, SQL is not mentioned in the paper, with the exception of a table summarizing prior work. The study was not included, along with several other borderline exclusions [7, 11, 19, 20, 23, 27, 31–33, 45, 46, 50, 52, 55, 61, 63, 73, 79, 93, 97, 104]. Finally, we recognize that it is increasingly common for ICT research to report implications for research, industry, and teaching. It follows that there are most likely papers that report implications to SQL education, but were not found by our search criteria. As indications for finding these implications are not often found in abstracts, reading, for example, all relational database related research was not feasible, and not done in this study.

Inclusion (I1–I2) and exclusion (E1–E7) criteria are presented in Table 2. We decided to exclude papers published before 1989. Although this year marked the publication of the first revision to the SQL standard, SQL-89, this was not an educated choice as much as conveniently including 30 years of SQL education research.

3.4 Data Extraction

We extracted basic reference information from the database searches: names of the authors, title, publication year, name of the publication forum, and issue number, volume, and page numbers, where applicable. Once the primary studies were selected, we classified each paper according to the research type and research topic facets. We also marked why a paper was excluded according to our predefined exclusion criteria, the number of citations from Google Scholar, and the number of participants in each study.

3.5 Classification

Wieringa et al. [112] propose six classes (or research type facets, as Petersen et al. [83] summarize) for requirements engineering papers. As we began categorizing our primary studies according to these facets, it became clear that they are not a natural fit with educational research papers. Two particularly problematic classes were validation and evaluation research. To summarize Wieringa et al. [112], validation research in requirements engineering is effectively prototyping, simulation, and experiments (i.e., *in vitro*), whereas evaluation research is effectively case study, field study, or survey (i.e., *in vivo*). In educational research, the dividing line between *in vitro* and *in vivo* is more difficult to determine. If *in vivo* studies are concerned with students in their natural learning environments, how much restriction (e.g., limiting the time to complete an exercise, forbidding communication or use of online materials) constitutes in making the research setting *in vitro*? Rather than trying to estimate how natural the research settings of the primary studies were, we adapted the research type facet classification to better fit educational research (Table 3).

As Wieringa et al. [112] point out, it is possible that one study covers more than one research type facet. For example, Taipalus et al. [101] present a query concept framework based on their teaching experience, an error categorization framework based on a qualitative study, and opinions on how SQL should be taught. By these three aspects, their study could be classified as a philosophical paper, evaluation research, or an opinion paper. We classified each primary study according to what we perceived as their primary contribution. As discussed in Wohlin et al. [115], these classifications merely represent an overview of the type of research in a given mapping.

After the final 89 primary studies were selected, the first author classified the studies into categories according to their topics. This was done based on full text reading, and according to directed content analysis [56] with the utilization of prior knowledge on SQL education research categories. Using preconceived topic categories, the first author classified each primary study into a category. If a study was concerned with a topic which did not fit to any category, a new category was considered. Topic categories are reported in Section 4.2, and this category scheme is used to structure Section 5, in which we report the teaching approaches in more detail.

3.6 Threats to Validity

3.6.1 Descriptive Validity. Descriptive validity concerns the objectivity and accuracy of the data gathering. We utilized a data collection form described in Section 3.4 to increase the objectivity and accuracy of the classification and study selection. Both authors used the same form when selecting the studies and classifying the research type.

3.6.2 Theoretical Validity. Theoretical validity concerns the selection and classification of the data. We tried to minimize the possibility of missing relevant studies by searching several databases, and by performing backward snowballing twice (Fig. 1). The first snowballing yielded 16 additional studies, and the second 3, yet after closer inspection, not all these studies were included in the final selection. As Petersen et al. [84] point out, researcher bias is a known threat to validity in the study selection phase. We tried to mitigate this by performing research type classification and applying

Table 3. Research type facet in educational research (adapted from Wieringa et al. [112] and Petersen et al. [83])

Category	Description
Evaluation research	Hypotheses are tested on (or phenomena studied among) their natural target populations, and preferably in as natural environments as possible. This means that if the hypotheses are concerned with evaluating a new method for teaching students, the natural target is a student or a novice in a given technique (e.g., a language), and the most natural testing environment should be the environment the students would be using regardless of the research setting. This is not always possible, and varying degrees of unnatural elements must often be included. Sufficient quantifiable evidence is presented.
Solution proposal	Paper presents a new or significantly improved solution for a common and recognized problem. The topic may be related to concepts that are difficult for students to learn, teacher's workload, or curriculum improvement. Solid arguments for (and preferably against) the proposal are presented.
Replication study	Paper replicates a previously reported research setting as accurately as possible, or with premeditated alterations (e.g., a different teacher, students who major in a different subject, or undergraduate instead of graduate students). The goal of the study is to check the validity of the previous study, or to study generalizability of the results. Sufficient quantifiable evidence is presented.
Philosophical paper	Paper presents a new conceptual framework, taxonomy, general teaching approach, new, improved or adapted research method, or simply summarizes existing work in a form of systematic literature review or systematic mapping. Depending on the type of philosophical paper, the paper may utilize existing literature, or be based on professional opinions or experiences.
Opinion paper	Paper expresses opinions of the author or a third party. These opinions may be concerned with, for example, whether something should be taught, how it should be taught, or to whom it should be taught. Typically no scientific evidence is presented.
Experience report	Paper describes how something was done, for example, a course or a curriculum implementation. The new setting should be described in sufficient detail, so that others may replicate it. Paper should report what worked and what did not.

exclusion criteria E3–E7 independently, and comparing results afterwards. Topic classification (Section 4.2) was done solely by the first author, and is the main threat to validity in this regard.

3.6.3 Interpretive Validity. Interpretive validity concerns researchers' biases in the interpretation of the data. The first author is an author of several selected primary studies, which may induce bias in interpretation. The second author, however, is not, and there were no disagreements on whether to include or exclude those studies. The first author's experience in educational research concerning SQL was also considered helpful in the study selection and classification processes, although this may have biased the interpretation of primary study results.

3.6.4 Repeatability and Generalizability. In order to increase the repeatability of our results, we followed systematic mapping guidelines proposed in Petersen et al. [83] and complemented later in Petersen et al. [84]. We also reported threats to validity, and how we tried to mitigate them. However, study selection and classification involve human judgment, and another group of researchers might select at least slightly different set of primary studies.

4 PUBLICATIONS

4.1 Publication Fora

Out of the 89 primary studies, 38 (43%) were journal articles published in 18 different journals. 50 (56%) studies were presented in 31 different conferences, and one primary study was a workshop paper. As can be observed in Table 4, the studies subject to this mapping study have been published in various journals and conference proceedings, and

searching teaching approach proposals should not be limited to merely educational fora. Appendix A lists the primary studies and their corresponding identifiers.

Publication fora and citations among primary studies are illustrated in Fig. 2. We have clustered the primary studies according to the journal (JISE, JCSC) or the organization (ACM, IEEE, Elsevier, and top IS) the forum is associated with. In addition to the AIS *senior scholars' basket of journals*[1] in IS, the top IS cluster contains primary studies from associated fora (CAIS and ICIS) with the exception of JISE, which formed a large enough cluster on its own. The color of an edge corresponds to the citing publication, for example, a blue edge between a red and a blue node indicates that the blue node is citing the red. Alternatively, a clockwise curving edge from node x to node y indicates that x cites y.

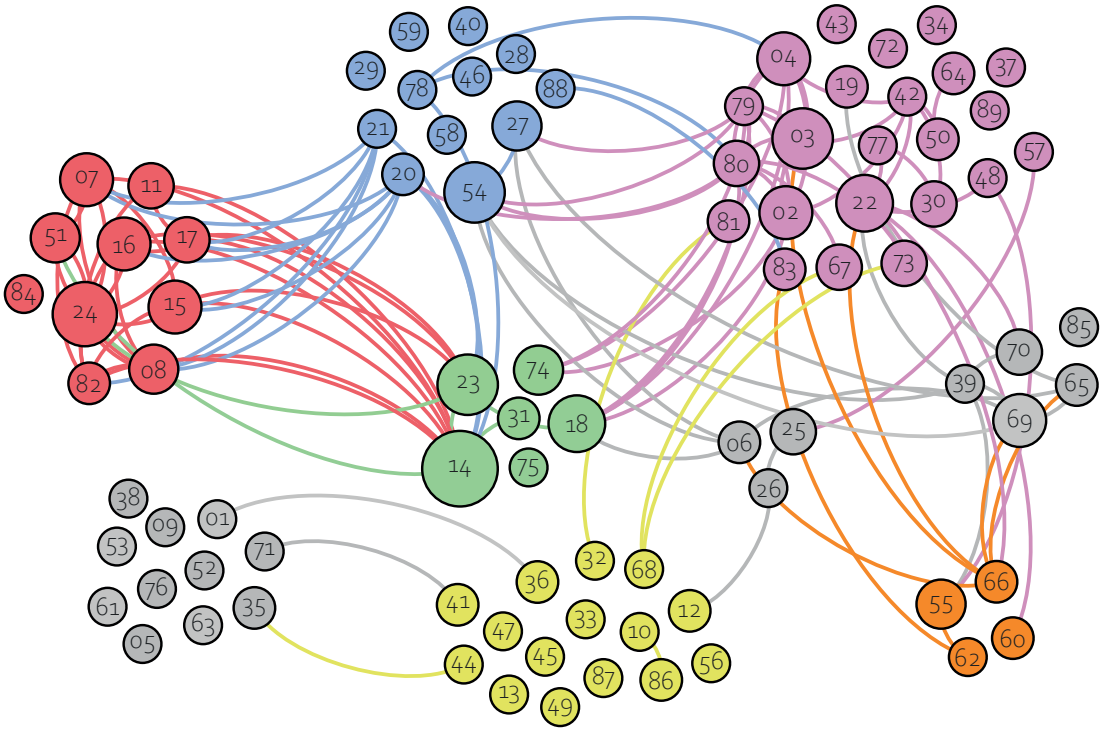


Fig. 2. Publication fora clusters and citations among primary studies - studies published in ACM journals or conference proceedings are clustered top right (purple), *Journal of Computing Sciences in Colleges* bottom right (orange), IEEE bottom center (yellow), top IS fora top left (red), *Journal of Information Systems Education* top center (blue), Elsevier center (green), and other fora (gray); size of a node represents in-degree, color of an edge corresponds to citing publication, edge curves clockwise from the citing to the cited publication, and numbers refer to primary study IDs

With a few exceptions, the top IS studies cite each other extensively, while citing among ACM studies varies. JISE, IEEE, Elsevier and JCSC studies cite each other relatively seldom. A small percentage of JISE studies cite top IS studies and some ACM and Elsevier studies. Top IS studies cite some Elsevier studies, but nothing else. ACM studies cite mostly Elsevier and JISE studies, but not IEEE or top IS studies. JISE studies cite mostly top IS, Elsevier, and ACM studies, but

[1] https://aisnet.org/page/SeniorScholarBasket

Table 4. Number of primary studies published in each forum

Forum name	Type	#
Journal of Information Systems Education (JISE)	Journal	12
ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)	Conference	6
ACM Technical Symposium on Computer Science Education (SIGCSE)	Conference	6
IEEE Frontiers in Education Conference (FIE)	Conference	4
Journal of Computing Sciences in Colleges (JCSC)	Journal	4
ACM Conference on Information Technology Education (SIGITE)	Conference	3
British National Conference on Databases (BNCOD)	Conference	3
Intl. Journal of Human-Computer Studies (IJHCS)*	Journal	3
ACM Conference on Management of Data (SIGMOD)	Conference	2
Information Systems Research (ISR)	Journal	2
Intl. Conference on Information Systems (ICIS)	Conference	2
Intl. Journal of Engineering Education	Journal	2
Journal of the Association for Information Systems (JAIS)	Journal	2
MIS Quarterly	Journal	2
ACM Annual Southeast Regional Conference (ACM-SE)	Conference	1
ACM Conference on Computer Personnel Research (SIGCPR)	Conference	1
ACM Conference on Extending Database Technology (EDBT)	Conference	1
ACM Conference on Information Technology Curriculum (CITC)	Conference	1
ACM Transactions on Computing Education (TOCE)	Journal	1
Annual Meeting of the Decision Sciences Institute	Conference	1
ASEE Annual Conference and Exposition	Conference	1
Communications of the Association for Information Systems (CAIS)	Journal	1
Computers & Education	Journal	1
Decision Support Systems	Journal	1
IEEE Annual Computer Software and Applications Conference (COMPSAC)	Conference	1
IEEE Conference on Industrial Electronics and Applications (ICIEA)	Conference	1
IEEE Global Engineering Education Conference (EDUCON)	Conference	1
IEEE Integrated STEM Education Conference (ISEC)	Conference	1
IEEE Intl. Conference on Data Engineering (ICDE)	Conference	1
IEEE Intl. Conference on Information Systems and Economic Intelligence (SIIE)	Conference	1
IEEE Intl. Conference on Intelligent Computer Communication and Processing (ICCP)	Conference	1
IEEE Intl. Conference on Networked Computing and Advanced Information Management (NCM)	Conference	1
IEEE Intl. Conference on Scalable Computing and Communications (ScalCom)	Conference	1
IEEE Transactions on Education (TOE)	Journal	1
IEEE Transactions on Software Engineering (TOSE)	Journal	1
Information Systems Education Conference (ISECON)	Conference	1
Intl. Conference on Cognition and Exploratory Learning in Digital Age (CELDA)	Conference	1
Intl. Conference on Computer Supported Education (CSEDU)	Conference	1
Intl. Conference on Interactive, Collaborative and Blended Learning (ICBL)	Conference	1
Intl. Conference on Web-Based Learning (ICWL)	Conference	1
Intl. Conference on Very Large Data Bases (VLDB Conference)	Conference	1
Intl. Convention on Information, Communication and Electronic Technology (MIPRO)	Conference	1
Intl. Journal of Learning, Teaching and Educational Research (IJLTER)	Journal	1
Intl. Journal on Very Large Data Bases (VLDB)	Journal	1
Intl. Scientific Conference Computer Science	Conference	1
Intl. Workshop on Teaching, Learning and Assessment of Databases (TLAD)	Workshop	1
Journal of Management Information Systems (JMIS)	Journal	1
Journal of Systems and Software	Journal	1
Journal on Systemics, Cybernetics and Informatics (JSCI)	Journal	1
UK & Ireland Computing Education Research Conference (UKICER)	Conference	1
Total		89

* Formerly Intl. Journal of Man-Machine Studies

not IEEE studies. IEEE studies cite some ACM studies, but nothing else. Two of the JCSC studies cite a total of three ACM studies. Some Elsevier studies cite some top IS studies. Publications per year are presented in Fig. 3.

The graph in Fig. 2 can be considered an indication of potentially untapped relevant primary research between clusters, yet it should be interpreted with caution. First, the edges only represent citations *among primary studies*, and are not an indication of how many citations a study has received (which, in itself, is not an indication of, for example, quality of a study). Second, the age of a study has a natural effect on the number of citations. The number of citations overall are presented in Table 5, to give an indication of the most commonly cited primary studies in SQL education. It is worth noting that even though a primary study is not cited among the selected primary studies, it may have received scientific attention outside these primary studies, as is the case with, for example, PS75. Finally, all SQL related research is not, intuitively, relevant to each other, but the graph propounds the view that researchers are studying similar aspects of SQL education without knowledge of each other. We give examples that support this argument in Section 7.3.

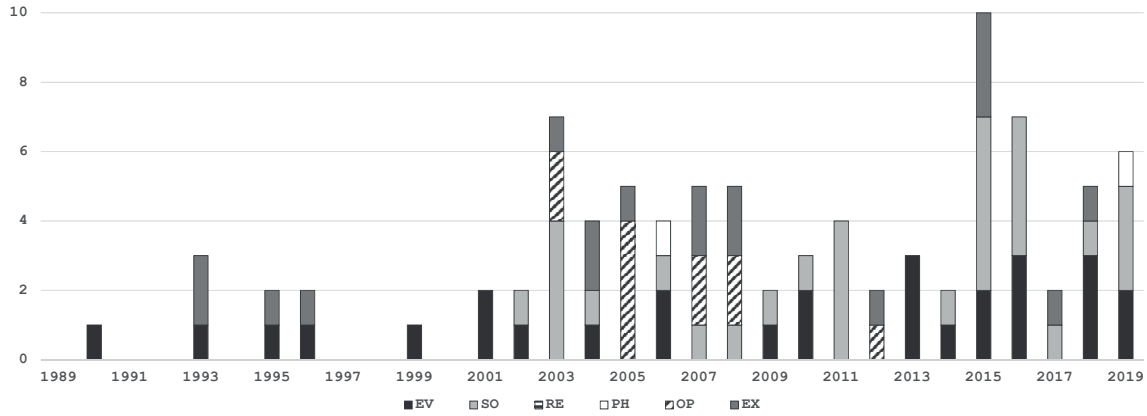


Fig. 3. Number of publications per year, and research type facets: evaluation research (EV), solution proposal (SO), replication study (RE, none present), philosophical paper (PH), opinion paper (OP), and experience report (EX)

4.2 Classification

We presented our adapted research type facet classifications in Table 3. Additionally, we classified the primary studies according to their topics, which we next describe briefly, and in detail in Section 5. It is worth noting that the categories overlap, and that a number of studies were candidates to more than one category. The names and descriptions of the categories are based on full-text reading of the primary studies, and constructed using directed content analysis [56]. The summary of primary study distribution between these two classifications is presented in Fig. 4, and detailed classification in Appendix B.

Studies concerning *student errors* (11 papers): these studies are concerned with presenting what kind of errors students commit during their query formulation processes, what types of errors students usually cannot fix, possible reasons why query formulation fails, and how to teach SQL in a DBMS or context independent viewpoint. Understanding what are the most common errors and what causes them is a crucial step in demonstrating and mitigating these errors in teaching. Most of these studies are evaluative in nature.

Table 5. Primary studies, number of citations from Google Scholar in Sep. 2019, and citations divided by publication age in full years - PS09 was not indexed by Google Scholar

ID	citations	citations/y	ID	citations	citations/y	ID	citations	citations/y
PS75	70	14	PS76	6	1.2	PS62	2	0.3
PS02	22	5.5	PS45	11	1.1	PS82	1	0.3
PS18	70	5	PS69	18	1.1	PS27	4	0.2
PS08	43	4.3	PS70	17	1.1	PS29	2	0.2
PS30	39	4.3	PS01	4	1	PS38	3	0.2
PS07	59	4.2	PS57	1	1	PS48	1	0.2
PS24	113	4.2	PS81	17	1	PS55	3	0.2
PS16	58	4.1	PS84	15	0.9	PS61	3	0.2
PS17	38	3.8	PS15	13	0.8	PS63	3	0.2
PS25	19	3.8	PS19	10	0.8	PS65	1	0.2
PS73	15	3.8	PS31	22	0.7	PS67	3	0.2
PS03	13	2.6	PS52	2	0.7	PS87	2	0.2
PS12	13	2.6	PS53	5	0.7	PS32	2	0.1
PS86	71	2.6	PS36	5	0.6	PS37	1	0.1
PS04	10	2.5	PS46	5	0.6	PS72	4	0.1
PS80	5	2.5	PS49	7	0.6	PS21	0	0
PS33	12	2.4	PS83	11	0.6	PS42	0	0
PS64	38	2.4	PS06	5	0.5	PS44	0	0
PS14	44	2.3	PS13	1	0.5	PS47	0	0
PS26	9	2.3	PS66	1	0.5	PS58	0	0
PS50	33	2.2	PS88	3	0.5	PS59	0	0
PS23	44	2.1	PS11	8	0.4	PS68	0	0
PS22	17	1.9	PS28	5	0.4	PS77	0	0
PS10	32	1.8	PS41	5	0.4	PS78	0	0
PS74	36	1.4	PS54	8	0.4	PS79	0	0
PS20	9	1.3	PS56	7	0.4	PS85	0	0
PS34	31	1.3	PS71	2	0.4	PS89	0	0
PS39	6	1.2	PS35	2	0.3	PS05	0	0
PS51	29	1.2	PS40	5	0.3	PS09	–	–
PS60	14	1.2	PS43	4	0.3			

Studies concerning the *exercise database* and elements closely related to it (20 papers): these studies evaluate, report experiences, and present opinions and solutions in regard to what kind of an exercise database is efficient in facilitating SQL learning. The studies discuss how to visually present the exercise database schema to students, how to express the data demands, what kind of database business domains should be used, how realistic the database should be in terms of data, and whether the students should be made aware if their SQL queries are logically correct. Most of these studies are evaluative in nature.

Studies presenting a *specific teaching approach* (9 papers): these studies present a teaching approach which concerns a specific subset of SQL, for example, how relational division, outer join, or existence negation should be taught. Most of these studies are solution proposals and opinion papers.

Studies presenting a *non-specific teaching approach* (22 papers): these studies discuss a more general teaching approach which should or could be used in teaching all SQL in a given course. The studies propose, for example, group learning and projects, how a course should be structured, and what kind of general techniques can facilitate SQL learning. Most of these studies are experience reports.

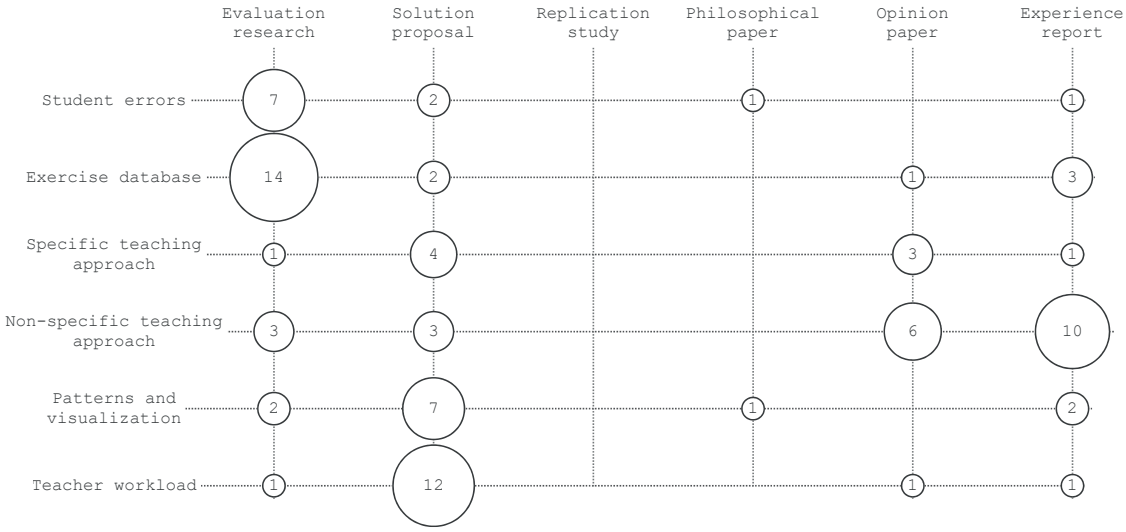


Fig. 4. Number of primary studies in each research type facet (x-axis) and topic facet (y-axis) intersection

Studies discussing *patterns and visualization* (12 papers): these studies mainly propose solutions on how to visualize the query execution process to students, whether to use visual query builders to facilitate SQL learning, planning queries before writing using a specialized notation, and utilizing steps and natural language patterns in query formulation. Most of these studies are solution proposals, and many overlap with the previously described category.

Finally, a number of studies proposed approaches to ease *teacher workload* (15 papers): these studies proposed solutions concerning, for example, automated exercise generation, automated grading and feedback, and pointed educators to materials available online. Arguably, as the teacher workload lightens, educators can focus more on difficult concepts regarding SQL.

5 SQL TEACHING PRACTICES

All the teaching considerations listed in this section are not actionable advice per se, but, for example, concerned with the most common errors students commit. These insights may be utilized by the teacher to focus on certain query concepts during lectures or in exercise design. Furthermore, these errors can be utilized in exercise database data generation, so that at least incorrect queries with the most common logical errors return data that is different from the correct result table. Finally, these errors may be used to guide digital learning environment development, so that feedback for the most common errors may be generated. It is worth noting that we have applied the nomenclature discussed in Section 2 to all the following teaching practice presentations.

Teaching considerations regarding student errors, the exercise database, specific and non-specific teaching approaches, patterns and visualization, and teacher workload are compiled into Tables 6, 7, 8, 9, 10, and 11, respectively. The teaching approaches are not in any particular order regarding arguments for and against. In other words, it is arbitrary whether an approach is presented as *Teach x [PS01; argued against in PS02]* or *Do not teach x [PS02; argued against in PS01]*. Appendix C lists the number of participants in each primary study.

Table 6. Teaching approaches or considerations regarding student errors

ID	Teaching approach or consideration
SE1	A list of semantic errors and complications can be used to support discussion with students on bad query writing practices [PS18]. This list is complemented with syntax and logical errors [PS80], and together give high level representation of what kinds of errors students can commit. Both of these lists are too long to discuss here.
SE2	Self-join is the most difficult query concept overall [PS02, PS03, PS04, PS79], and these queries fail due to logical errors [PS02], namely join errors: joins are formed with incorrect tables, columns, or comparison operators, a join is missing, or a join is extraneous and needs to be omitted [PS79].
SE3	After self-join, the most difficult query concepts are, not in order, correlated subquery [PS03, PS04, PS79], simple one-table query [PS04], simple subquery [PS03], grouping restrictions [PS04], uncorrelated subquery [PS79], and expressions with nesting [PS79].
SE4	The most common errors are, in order, incorrect ordering of columns in the SELECT clause, undefined column name used, joining incorrect columns from correct tables, unnecessary joins, extraneous tables, omitting tables, missing expression, aliases that are always identical, extraneous GROUP BY clause and COUNT function, and incorrect ordering of clauses [PS24].
SE5	Logical errors are the most common class of errors overall [PS79, PS80], and the most difficult class of errors to fix [PS04, PS79]. 40% of errors students commit are semantic or logical in nature, and occur in the SELECT and WHERE clauses [PS04].
SE6	The most frequent errors that student cannot fix are, in order, illegal or insufficient grouping, *common syntax errors*, inconsistent expression, inconsistent joins, missing joins, expression errors such as missing or extraneous expressions, or expressions in incorrect clause, and projection errors such as missing or extraneous columns in the main the SELECT clause [PS79].
SE7	54% of errors student commit are syntactical in nature, and 69% of syntax errors are caused by typing errors [PS04].
SE8	Most frequent syntax errors are *common syntax errors* [PS02, PS79, PS80] and the use of undefined database objects [PS02, PS24, PS79], although the latter type of errors are usually fixed [PS79].
SE9	The next most frequent syntax errors are, in order, grouping errors, use of aggregate functions in the GROUP BY clause, use of undefined operators, and problems with writing expressions [PS02].
SE10	Among queries requiring the use of aggregate functions, illegal or insufficient grouping is the the most frequent type of error [PS02, PS79], followed by the use incorrect functions, incorrect columns as function parameters, missing DISTINCT from the function parameter, and DISTINCT as a function parameter where not applicable [PS79].
SE11	Syntax errors are the cause of failure particularly in queries involving GROUP BY and HAVING clauses, as well as NATURAL JOIN [PS02].
SE12	In multi-table queries, the most frequent errors are, in order, inconsistent joins, missing joins, and join errors such as joins on incorrect table or using incorrect columns [PS79].
SE13	Unnecessary complications are frequent in all queries, regardless of the query concepts required [PS79, PS80].
SE14	Errors are usually caused by short-term memory limitations, absence of a clue in the data demand, procedural fixedness, or ignorance [PS74].
SE15	Most frequent omission errors are, in order, omitting a join clause, omitting a subquery, and omitting the HAVING clause [PS04].
SE16	When a student attempts an exercise more than 30 times, and there is at least one error regarding aggregate function usage in the GROUP BY or WHERE clause, it is statistically unlikely that the student can successfully formulate the correct query [PS02].
SE17	Teach standard SQL because using merely one DBMS will confuse students what is standard and what is DBMS specific. A practical approach to this is to choose two DBMSs to teach students [PS67].
SE18	Teach SQL as a general language that is used in modern tools (e.g., NewSQL DBMSs) as well to mitigate motivational concerns on the relevance of the language [PS73].

Table 7. Teaching approaches or considerations regarding the exercise databases

ID	Teaching approach or consideration
DB1	A list of 19 query concepts that introductory database course exercises may test, and corresponding 15 exercises with example answers are presented [PS80]. This list is too long to be presented here.
DB2	Data demands should be formulated with as little ambiguity as possible [PS11, PS65]. A less ambiguous data demand entails fewer attempts [PS11, PS14], more perceived confidence [argued against in PS14] and correctness [PS11], less time spent [PS11], and less errors [PS11, PS14, PS21]. There exist at least seven types of ambiguity [PS11].
DB3	Unambiguous data demands are more and more important as data demands' complexity increases [PS20].
DB4	With low complexity queries, less ambiguous data demands produce less query formulation errors, but with high complexity queries, data demand ambiguity has no effect on errors [PS21, PS82].
DB5	As training progresses, students should be introduced to more and more ambiguous data demands, which better reflect their future work environments [PS82].
DB6	When GROUP BY clause is needed, the natural language representations should (at least in early exercises) contain a clear indication to use it [PS03].
DB7	Presenting the database as an event-based ER or state-based ER does not affect query accuracy or student confidence [PS07], but in regard to query formulation success rates, it is better to represent a database schema rather than a list of database contents or an ER diagram [PS31]. Furthermore, database representation semantics [PS51], symbols [PS51; argued against in PS10], and foreign key constraint representation [PS51] all have influence on query formulation success.
DB8	The three most important factors in query formulation success rates and time needed are, in order, data model representation realism, high expressive ease, and query complexity. Data model representation realism refers to which level the data model is represented, and the levels are, in ascending order of realism, physical, logical, and conceptual. Expressive ease is concerned with the language used, were it SQL, natural language, or something else [PS23].
DB9	If incongruence (i.e., how well or poorly real world constructs match their equivalents in the database) increases, success rates fall, more time is needed, and students feel less confident [PS14]. However, best design practices (e.g., database normalization) should not be sacrificed in order to reach more ontological clarity, as the implications for benefits are conflicted [PS15, PS16, PS17; argued against in PS74].
DB10	Provide an interface (or a cheatsheet) that allows students to see SQL keywords and database object names to reduce typing errors [PS05]. Consider highlighting relevant parts of the data model for each data demand [PS82].
DB11	If data demand complexity increases, success rates fall, more time is needed, and students feel less confident [PS14].
DB12	Allowing students reuse similar queries in exercises leads to faster query formulation, but results in more errors, and a poorer relationship between confidence and query correctness [PS08].
DB13	Students should not execute queries in the same exercise database, because modifications affect others [PS44].
DB14	Use complex [PS41, PS62, PS83; argued against in PS57 because students cannot manually check problems with erroneous queries against complex data] and low quality [PS83] exercise data because students need to gain understanding of complex environments, and that real data contains errors and missing values. Furthermore, use databases with business domains which are novel to the students so that students learn the importance of domain knowledge and can recognize abstract patterns and utilize them in different domains [PS46]. More realistic databases are perceived more interesting an useful by students [PS88].
DB15	Provide students with the correct result table [PS68, PS80; argued against in PS03 as students may use brute force to write correct queries], or the number of rows in the correct result table [PS68]. If these are not provided, students should validate their results by manually writing tests [PS20]. Students should understand that query evaluation against a single dataset is not enough [PS39].

6 DISCUSSION

6.1 Patterns and Anti-patterns

Even though we listed numerous teaching approaches in the previous section, it remains unclear which approaches are patterns and which are anti-patterns, and in which contexts. As may be observed in the previous section, we do not differentiate between approaches based on objectively interpreted results and subjective discussion. Consequently, we

Table 8. A list of teaching approaches or considerations regarding a specific teaching approach

ID	Teaching approach or consideration
SA1	Teach relational division with GROUP BY and HAVING, rather than multiple existence negations. This is easier for students to learn [PS54, PS56], and the written queries are computationally faster [PS54; the latter point is argued against in PS56]. This work is extended from teaching relational division to teaching set comparison with a general approach [PS27].
SA2	Teach OUTER JOIN according to ANSI SQL-92, i.e., with OUTER JOIN rather than UNION or derived tables. This is perceived easiest and it is computationally faster than the alternatives [PS55].
SA3	Teach existence negation with an English-like query language before teaching the SQL equivalent [PS48].
SA4	Explain the differences in the logic of NOT EXISTS and NOT IN subqueries [PS18].
SA5	Teach strict grouping [PS21, PS80]. Effectively, this means that if the main SELECT clause contains at least one aggregate function, and at least one grouping column, all and only the grouping columns must be included in the GROUP BY clause.
SA6	Teach integrity constraints by dividing them into five classes: dynamic, domain, tuple, relation, and database integrity constraints [PS29].
SA7	If you use Microsoft Access to teach SQL, and want to teach recursive joins which are not supported, stored procedures can be used to complement SQL [PS28].
SA8	Teach transaction control using real SQL examples, and not simple READ(a) and WRITE(b) that are usually found in database textbooks [PS37].
SA9	Teaching SQL after QBE yields better results than teaching SQL first [PS86; the use of QBE is argued against in PS69 because mental models must be changed when switching to SQL].

Table 9. Teaching approaches or considerations regarding a non-specific teaching approach

ID	Teaching approach or consideration
NA1	Emphasize practical work [PS64, PS71].
NA2	Teach SQL with short online lectures [PS81].
NA3	Students should learn SQL in teams [PS01, PS09, PS34, PS41, PS53] and group projects [PS32, PS40, PS59, PS63, PS72], and the project should be based on realistic and reported specification [PS63]. These groups should be formed based on student skill, and the level of difficulty of the exercises set accordingly [PS38]. The online environment utilized in the course should facilitate team forming [PS01].
NA4	Teach students how to read SQL [PS61] before writing SQL [PS19]. Furthermore, demonstrate DBMS error messages [PS20] and erroneous queries [PS43], especially regarding difficult concepts such as ALL and NOT EXISTS [PS61]. Have students explain why they are erroneous, and why a certain solution works [PS43, PS61, PS89].
NA5	Have students come up with analogies for SQL query concepts and predicates. This helps students understand the concepts, and remember them longer [PS58].
NA6	Teach DDL first, then integrity constraints, and finally DML [PS32; argued against in PS85]. Teach SQL before relational algebra [PS61], and introduce relational algebra only in the context of implementation and optimization to avoid students confusing relational algebra with SQL [PS61]. Regarding concurrent courses, do not teach SQL at the same time with a procedural language [PS39].
NA7	Instead of a final exam, organize intermediary assessments which can be taken after a certain number of exercises have been passed [PS70]. This helps especially weaker students [PS70]. Giving the assessments in a digital learning environment positively affects grades [PS01]. SQL skills should not be assessed through SQL code alone, but also with multiple choise questions [PS13]. Brighter students' motivation suffers if a course is not challenging enough [PS70].
NA8	Demonstrate difficult SQL concepts with animations [PS33, PS60, PS89].
NA9	Encourage students against unnecessary SQL elements, even though such omittances affect readability [PS61].
NA10	Use SQLite to teach SQL, because it is lightweight and students do not need to configure anything [PS52; argued against in PS80].

Table 10. Teaching approaches or considerations regarding patterns and visualization

ID	Teaching approach or consideration
PV1	Utilize a template to help students write more complex SQL queries [PS06, PS20, PS82]. This increases success rates [PS82], and decreases errors in the FROM and ORDER BY clauses, but not in the GROUP BY clause [PS21].
PV2	Have students plan more complex queries to ease cognitive load [PS78]. A planning notation is introduced and described [PS78]. As data demand complexity increases, *a priori* planning decreases the number of errors more and more [PS21].
PV3	Teach students how to identify certain natural language patterns (e.g., *never*, *all*, *sum*) and their corresponding SQL clauses, constructs, and keywords [PS66, PS77].
PV4	Teach SQL query formulation in steps (i.e., procedurally) [PS04, PS21, PS66, PS77]. Alternatively, introduce both procedural and set-based query formulation approaches at the start of a course. Students can choose which to use [PS65]. Procedural approach to query formulation is more natural to students, but fails at complex queries [PS65].
PV5	Visualize query execution [PS22, PS30, PS42]. It is helpful if students can visualize the query step by step, and go forward and backward, similar to programming language debuggers [PS22, PS42]. If possible present the query simultaneously visually and textually [PS39].
PV6	If students are likely to never write complex SQL, alternatives such as QBE should be considered, as it is faster to utilize, and perceived more comfortable [PS45].

Table 11. Teaching approaches or considerations regarding teacher workload

ID	Teaching approach or consideration
WL1	A list of 14 small SQL course modules is presented [PS85]. The list is too long to be presented here. SQL concepts are divided into basic, advanced, and expert level modules [PS38] These modules may be used as, for example, a structure for short online lectures [PS81].
WL2	Learning environments that allow teachers to monitor student activity, and also allow students to give feedback to the teacher [PS01] are available. Furthermore, large online learning environments with exercises and exercise databases are available without fee [PS84].
WL3	Exercise database datasets can be generated automatically [PS12, PS25, PS26], and tested against expected erroneous queries automatically [PS12]. A query should be tested against multiple datasets [PS01], and discrepancies can be used to automatically provide feedback [PS01, PS49, PS87]. Alternatively, a query's correctness can be evaluated using string metrics [PS76] or XML transformations [PS39].
WL4	As an alternative to automatic exercise database generation, students may be required to create their own exercise databases and grant appropriate privileges [PS44].
WL5	Data demands can be automatically generated based on correct SQL queries [PS47].
WL6	Utilize examinations and exercises which can be automatically graded [PS81].
WL7	Students should be given the opportunity to select themselves how complex queries they want to practice writing (query concepts, number of tables etc.), and these exercises can be automatically generated [PS35]. Furthermore, students should be allowed to choose a level of hints which the system suggests [PS50].
WL8	SQL taught through game based learning significantly increases student performance when compared to textbooks [PS75].

advise a level of caution when interpreting the reported teaching approaches in the previous section, and the number of corresponding participants Appendix C.

As the nature of opinion papers and experience reports is as their names suggest, these approaches are seldom tested in a scientific setting. As an example, Matos and Grasser [70] suggested a teaching approach for teaching relational division which is easier for students to understand. The authors report no numbers concerning how many students found the approach easier. However, by comparing the proposed teaching approach and the commonly used alternative, the benefits are apparent; in addition to being computationally faster, the approach of using GROUP BY with HAVING is arguably easier to read than multiple existence negations, at least in our opinion. In contrast, Borthick et al. [15]

studied how the database normalization level affects errors committed in query writing, and found out that end-users commit fewer errors in queries against a database adhering to the first normal form than end-users against a database adhering to the third normal form. The hypotheses were tested with 80 undergraduate and masters level students.

Based on reported quantifiable evidence supporting the views presented in these two studies, it might be compelling to advise the use of lower normal form databases over higher ones, and to dismiss the one regarding relational division. Although fewer errors might be a desirable goal to strive for, lower normal forms in database education present significant downsides. Students learn bad design practices which later need to be unlearned, the database is subject to anomalies [38], and requires more disk space due to redundancy. Finally, it is not clear whether students should strive for fewer errors, (although other database end-users arguably should), as errors are arguably an efficient way through which students learn, as argued in SQL education research [54, 100] as well as broader educational contexts [74].

6.2 Natural and Unnatural Learning Environments

A recurring theme in the primary studies, regardless of the research topic, was argumentation for [5, 8, 101, 121] and against [75, 88, 101] natural learning environments. A natural learning environment better reflects industry, i.e., students' future work environments. Environmental traits differ between workplaces, job titles, and used technologies. For the sake of discussion, we state that in a workplace there is no known correct result table for a query [5], the data demand is ambiguous [24], the datasets are complex [51], and the business domain is unfamiliar [58]. In contrast, peers are often present to offer help, use of textbooks and the internet is naturally not forbidden, and the query may be formulated as many times as necessary in a feasible timeframe. In an unnatural learning environment, these characteristics are reversed. The underlying arguments for natural environments are that students need to be prepared for their future work, and the arguments against are usually that natural environments hinder the learning of SQL (e.g., perceived confidence and success rates decrease). In teaching, these two approaches are usually mixed to varying degrees, for example, Taipalus et al. [101] report giving students the correct result table but designing exercise database data to contain no anomalies, yet Wagner et al. [110] report utilizing low quality data.

If the goal of SQL education is to prepare students to effectively work in their future work environments, learning should take place in more natural environments, and there is no need to exclusively choose a natural, mixed, or unnatural environment. SQL should first be taught in an unnatural environment [12, 14], and when the syntax and semantics are mastered to a degree, natural elements such as data demand ambiguity may be introduced gradually [108], or a natural environment used in the final exam. Naturally, grading team performance is more difficult to the teachers, and students should be prepared to work independently in their future workplace, even though help is available. We discuss natural environments more in Section 7.1.

Although Lertnattee and Pamonsinlapatham [66] argue for using SQLite due to its relatively easy configuration, teachers should be aware that SQLite 3 contains features[2] which, in our experience, confuse students. For example, in SQLite 3, data types have little meaning (strings can be stored in INT columns), some arguably important SQL concepts are not implemented (ALL, RIGHT OUTER JOIN), PRIMARY KEY does not imply NOT NULL, and strict grouping is not enforced.

[2] https://www.sqlite.org/quirks.html

6.3 Decay

In Section 3.3, we wrote that we rather conveniently chose to include SQL education research from a timeframe of 30 years. However, we advise caution when interpreting the results from the older primary studies, as these teaching considerations *decay* over time. Both the SQL standard and its implementations develop over time, as do the technologies in IT field in general. For example, a learning environment from 1990s appears naïve in terms of features, and the general look of the user interface. Some, mostly older works study the effects of a conceptual database structure representation instead of logical representation [61], while others criticize the very purpose of such a research setting [90]. More importantly, examining some older studies raises questions whether the SQL language itself has changed too much for a teaching approach to hold true anymore. This point is further emphasized with the notion that the SQL standard has never been a simple source to interpret. Three examples follow.

First, a seminal study from 1995 [95] considered "omitting the FROM clause" a semantic rather than a syntax error, even though (at least current) SQL standard considers the FROM clause mandatory in a query. Furthermore, the study demonstrated all table joins with explicit WHERE clause conditions, without the use of JOIN predicate or subqueries. This might be an educated approach, a coincidence, or resulting from the fact that these concepts were introduced in the SQL-92 standard. At least one study [90] suggests that separating expressions and joins in their respective clauses reduces some types of query formulation errors. It is unclear why Smelcer [95] demonstrates table joins using only explicit join conditions in the WHERE clause, but this is a reason to infer that the students who participated in the study were taught table joins with explicit WHERE clause conditions.

Second, another study from 1993 [119] demonstrated erroneous queries with subqueries formulated with NOT EXISTS, in which the subqueries' SELECT clauses contains multiple column names, and stated "Both cases contain errors of form. The subqueries used with EXISTS (NOT EXISTS) should use the SELECT * ... format." Nowadays, it is more of a widely accepted practice to use simple (NOT) EXISTS subquery SELECT clauses such as SELECT * or SELECT 1, but effectively it does not matter what is selected, and even division by zero is accepted by DBMSs.

Third, a study from 1988 [21] demonstrates how the aggregate function SUM handled NULL at the time. The study demonstrated how SUM would return NULL if even one of the items was NULL. Nowadays, the standard has been revised, and in most implementations, SUM handles NULL similar to zero. Rather than criticism toward the aforementioned studies, we are trying to communicate that even though the language we are using today has the same name as decades ago, SQL has undergone notable changes, and for this reason alone older studies should be given closer scrutiny.

7 FUTURE RESEARCH AGENDA

7.1 Research Dearths

Concepts beyond SELECT have received little attention in educational research. The studied query concepts [4, 6, 101], and formulated error frameworks [17, 101] focus solely on data retrieval. Intuitively, the transition from SELECT to UPDATE and DELETE is relatively easy [49], as the query concepts in the WHERE clause are the same. In terms of SQL, DCL and TCL concepts are relatively simple, and the difficulty comes from the design of privileges and transactions rather than implementation. However, DDL statements and INSERT are both a fundamental and important part of SQL which have not been studied in detail.

Advanced SQL features have not been studied in educational contexts. If we consider the SQL concepts reported in the primary studies, most of them could be based on the SQL-92 standard, and in some cases, even on SQL-89. Since

then, numerous features have been added to the SQL standard, and they remain untapped from a research perspective. Such features are, for example, online analytical processing aggregate functions, the WINDOW clause, table functions, multisets, the MERGE statement, and generated columns added in SQL:1999 and SQL:2003 [47]. Additionally, SQL:2011 introduced both temporal [65] and non-temporal features [124] such as pipelined DML and enhancements to several older concepts. As we mentioned in Section 2.2, it is unclear whether knowledge about these features need to be propagated, or have they been omitted from course contents on purpose. Finally, in addition to software development, further research could also explore how SQL has extended to adjacent fields such as data science [18], broader contexts in general [44], and what types of SQL extensions have been introduced to better fit field specific needs [80].

Are unnatural environments beneficial remains an open question. Studies in which students or novices are aided by for example, automated feedback, simpler data, or unambiguous data demands achieve higher success rates in query formulation. This, however, does not necessarily reflect their future work environments. Furthermore, a recent study [99] discovered that as the logical complexity of the exercise database increases, students are less likely to succeed in query formulation. The same study, however, cautions the use of success rates alone in evaluating different teaching approaches; it is possible that although the students who fail in query formulation with a complex database, are more prepared for natural environments than students who succeed in query formulation with a simple database. Studies that test student skill in natural environments are needed, preferably so that one group of students learns SQL in an unnatural environment, and another in a natural environment, after which both groups are tested in a natural environment. Furthermore, as unnatural environments are intuitively targeted to help poor performing students, Russell and Cumming [91] raise an important concern that a certain level of simplification may impede both the learning, and the ardor towards the IT field of brighter students.

7.2 Replication

As presented in Fig. 4, there were no replication studies among the primary studies. While experience reports, opinion papers, solutions proposals, and philosophical papers are problematic to replicate due to their nature, even the most fundamental evaluation research studies [14, 34, 95] remain without replication. This is problematic, as central premises of subsequent studies are occasionally based on the results of the fundamental studies. The lack of replication studies in computing education in general has only recently received scientific attention [53]. Partly because of the lack of replication, we argue that educational SQL research is not mature enough to distinguish between patterns and anti-patterns. Moreover, a particularly insightful study by Rho and March [90] noted that some studies evaluated SQL on such a simple level, that the *ceiling effect* (i.e., variance in an independent variable is not measurable due to simplicity of the task) might explain the lack of differences in the results. Replication studies are not needed only because of reliability and the ceiling effect, but also because of obsolescence, as discussed in Section 6.3.

Beyond replication, and to uncover patterns and anti-patterns, it is crucial to evaluate proposed teaching approaches, especially those of solution proposals and opinion papers, in a scientific environment. Preferably, these evaluations should be done by researchers independent of the original authors, as it is common that reported solutions are considered helpful by the original authors. With propagation concerns [13] in mind, approaches supported by scientific evidence are likely to receive more attention among practitioners.

7.3 Building upon Existing Body of Knowledge

Based on the insights from the mapping process summarized in Fig. 2 and discussed in Section 4.1, we urge researchers to utilize and build upon existing body of knowledge in new approach proposals, and to critically evaluate all approaches.

Matos and Grasser [70] authored a study showcasing a new approach for teaching relational division. The study was published in the summer 2002 issue in JISE. Dadashzadeh [40] authored a study expanding and generalizing similar approach to other set comparison queries. This study shows relational division similarly to Matos and Grasser, and was published in the winter 2003 issue in JISE. Finally, McCann [72] authored a study presenting relational division similarly to Matos and Grasser, and this study was presented in the FIE conference in November 2003. Neither of the two latter studies cited Matos and Grasser, even though relational division is a specific concept. This might be a result of all the studies published within a short timeframe, but also due to potentially fragmented educational research fora. Regarding primary study citations in Fig. 2, is it that, for example, ACM studies in general considered top IS studies, but did not find them relevant, or is it that they did not find them? Did they not find them because of different nomenclature, or did they not utilize searches which included them? Would their research settings and conclusions have been different in this regard? As IEEE studies are seldom cited among the primary studies, we might have missed them if we did not know about IEEE beforehand. That being said, there might be relevant pockets of research that we have missed. Based on our results, we advise educational researchers and reviewers to utilize and search prior works widely, as educational considerations may be found in numerous fora.

Most of the 29 opinion papers and experience reports did not discuss potential downsides of their proposed or tested approaches, and only one [10] had a section dedicated to discussing disadvantages. We urge authors of studies of this nature to either critically evaluate their approaches, or discuss why the approach does not need critical evaluation. In comparison, even a course given as a textbook based exam (and nothing else) has positive implications, as students can study with a flexible schedule, and choose learning strategies based on their own preferences. With this in mind, one critical factor to discuss is time. Elements cannot be added to a course without expanding it or removing other elements. Expanding a course arguably has potential downsides, and, for example, sacrificing best database design practices to more efficiently teach SQL is not a desired goal in a database course. Alternatively, a teaching approach may be replaced altogether, as presented by for example, Matos and Grasser [70] and Matos et al. [71].

8 CONCLUSION

In this study, we set out to systematically map educational SQL research, and to list teaching approaches proposed in scientific literature. Our mapping shows that primary studies are published in numerous fora, not all of which are educational in nature. Recurring themes in educational SQL research are improved teaching approaches, students errors, the exercise database and related concepts, and easing teacher workload, and all types of research are represented, with the exception of replication studies. Furthermore, based on the 89 primary studies, we listed 66 teaching approaches to help educators teach SQL more efficiently. For researchers, and in addition to the systematic mapping, we proposed future research avenues, and general suggestions on how to conduct educational SQL research.

REFERENCES

[1] Alberto Abelló, Xavier Burgués, María José Casany, Carme Martín, Carme Quer, M. Elena Rodríguez, Óscar Romero, and Toni Urpí. 2016. A software tool for E-assessment of relational database skills. *International Journal of Engineering Education* 32 (2016), 1289—-1312. Issue 3. http://hdl.handle.net/2117/89668

[2] Alberto Abelló, M. Elena Rodríguez, Toni Urpí, Xavier Burgués, M. José Casany, Carme Martín, and Carme Quer. 2008. LEARN-SQL: Automatic Assessment of SQL Based on IMS QTI Specification. In *2008 Eighth IEEE International Conference on Advanced Learning Technologies*. IEEE. https://doi.org/10.1109/icalt.2008.27

[3] Elizabeth S. Adams, Mary Granger, Don Goelman, and Catherine Ricardo. 2004. Managing the introductory database course. In *Proceedings of the 35th Technical Symposium on Computer Science Education (SIGCSE)*. ACM Press. https://doi.org/10.1145/971300.971467

[4] 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

[5] Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. 2015. A Quantitative Study of the Relative Difficulty for Novices of Writing Seven Different Types of SQL Queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Press, New York, New York, USA, 201–206. https://doi.org/10.1145/2729094.2742620

[6] 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

[7] Judith D. Ahrens and Chetan S. Sankar. 1993. Tailoring Database Training for End Users. *MIS Quarterly* 17, 4 (1993), 419. https://doi.org/10.2307/249586

[8] A. AL-Salmi. 2018. A web-based semi-automatic assessment tool for formulating basic SQL statements: Point-and-click interaction method. In *Proceedings of the 10th International Conference on Computer Supported Education (CSEDU)*, Vol. 1. 191–198. https://doi.org/doi.org/10.5220/0006671501910198

[9] Brett Allenstein, Andrew Yost, Paul Wagner, and Joline Morrison. 2008. A Query Simulation System to Illustrate Database Query Execution. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '08)*. ACM, New York, NY, USA, 493–497. https://doi.org/10.1145/1352135.1352301

[10] W.J. Amadio. 2003. The dilemma of Team Learning: An assessment from the SQL programming classroom. In *Proceedings of the Annual Meeting of the Decision Sciences Institute*. 823–828. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0442278473&partnerID=40&md5=ffb48d535e80751aaf8b153af6a13c42

[11] Neal Ashkanasy, Paul L. Bowen, Fiona H. Rohde, and Chiu Yueh Alice Wu. 2007. The Effects of User Characteristics on Query Performance in the Presence of Information Request Ambiguity. *Journal of Information Systems* 21, 1 (2007), 53–82. https://doi.org/10.2308/jis.2007.21.1.53

[12] Micheal Axelsen, A Faye Borthick, and Paul L. Bowen. 2001. A model for and the effects of information request ambiguity on end-user query performance. *ICIS 2001 Proceedings* (2001), 68. http://aisel.aisnet.org/icis2001/68

[13] Heather Bort, David P. Bunde, Zack Butler, Christopher Lynnly Hovey, and Cynthia Taylor. 2019. Propagating Educational Innovations. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, NY, USA, 167–168. https://doi.org/10.1145/3287324.3287526

[14] A.Faye Borthick, Paul L. Bowen, Donald R. Jones, and Michael Hung Kam Tse. 2001. The effects of information request ambiguity and construct incongruence on query development. *Decision Support Systems* 32, 1 (nov 2001), 3–25. https://doi.org/10.1016/s0167-9236(01)00097-5

[15] 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

[16] Paul L. Bowen, Robert A. O'Farrell, and Fiona H. Rohde. 2009. An Empirical Investigation of End-User Query Development: The Effects of Improved Model Expressiveness vs. Complexity. *Information Systems Research* 20, 4 (dec 2009), 565–584. https://doi.org/10.1287/isre.1080.0181

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

[18] Jennifer E. Broatch, Suzanne Dietrich, and Don Goelman. 2019. Introducing Data Science Techniques by Connecting Database Concepts and dplyr. *Journal of Statistics Education* 27, 3 (2019), 147–153. https://doi.org/10.1080/10691898.2019.1647768 arXiv:https://doi.org/10.1080/10691898.2019.1647768

[19] Peter Brusilovsky, Sergey Sosnovsky, Danielle H. Lee, Michael Yudelson, Vladimir Zadorozhny, and Xin Zhou. 2008. An Open Integrated Exploratorium for Database Courses. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '08)*. ACM, New York, NY, USA, 22–26. https://doi.org/10.1145/1384271.1384280

[20] 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 (Jan. 2010), 15 pages. https://doi.org/10.1145/1656255.1656257

[21] R. B. Buitendijk. 1988. Logical errors in database SQL retrieval queries. *Computer Science in Economics and Management* 1, 2 (1988), 79–96. https://doi.org/10.1007/BF00427157

[22] Clifford G. Burgess. 1991. A graphical, database-querying interface for casual, naive computer users. *International Journal of Man-Machine Studies* 34, 1 (1991), 23–47. https://doi.org/10.1016/0020-7373(91)90049-d

[23] Luca Cagliero, Luigi De Russis, Laura Farinetti, and Teodoro Montanaro. 2018. Improving the Effectiveness of SQL Learning Practice: A Data-Driven Approach. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 01. 980–989. https://doi.org/10.1109/COMPSAC.2018.00174

[24] Gretchen Irwin Casterella and Leo Vijayasarathy. 2013. An Experimental Investigation of Complexity in Database Query Formulation Tasks. *Journal of Information Systems Education* 24, 3 (2013), 211–221. http://jise.org/Volume24/24-3/pdf/Vol24-3pg211.pdf

[25] Gretchen Irwin Casterella and Leo Vijayasarathy. 2019. Query Structure and Data Model Mapping Errors in Information Retrieval Tasks. *Journal of Information Systems Education* 30, 3 (2019), 178–190. http://jise.org/Volume30/n3/JISEv30n3p178.pdf

[26] Tiziana Catarci, Maria F. Costabile, Stefano Levialdi, and Carlo Batini. 1997. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages & Computing* 8, 2 (1997), 215–260. https://doi.org/10.1006/jvlc.1997.0037

[27] Claudio Cerullo and Marco Porta. 2007. A System for Database Visual Querying and Query Visualization: Complementing Text and Graphics to Increase Expressiveness. In *18th International Conference on Database and Expert Systems Applications (DEXA 2007)*. IEEE. https://doi.org/10.1109/dexa.2007.91

[28] Donald D. Chamberlin. 2012. Early History of SQL. *IEEE Annals of the History of Computing* 34, 4 (Oct 2012), 78–82. https://doi.org/10.1109/MAHC.2012.61

[29] Donald D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade. 1976. SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control. *IBM Journal of Research and Development* 20, 6 (Nov 1976), 560–575. https://doi.org/10.1147/rd.206.0560

[30] Donald D. Chamberlin and Raymond F. Boyce. 1974. SEQUEL: A Structured English Query Language. In *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control (SIGFIDET '74)*. Association for Computing Machinery, New York, NY, USA, 249–264. https://doi.org/10.1145/800296.811515

[31] Hock Chan, Keng Siau, and Kwok-Kee Wei. 1997. The effect of data model, system and task characteristics on user query performance: an empirical study. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems* 29, 1 (1997), 31–49. https://doi.org/10.1145/506812.506820

[32] Hock Chuan Chan. 2007. A two-stage evaluation of user query performance for the relational model and SQL. *PACIS 2007 Proceedings* (2007), 118. https://aisel.aisnet.org/pacis2007/118

[33] Hock Chuan Chan, Hock-Hai Teo, and XH Zeng. 2005. An evaluation of novice end-user computing performance: Data modeling, query writing, and comprehension. *Journal of the American Society for Information Science and Technology* 56, 8 (2005), 843–853. https://onlinelibrary.wiley.com/doi/full/10.1002/asi.20178

[34] Hock Chuan Chan, Kwok Kee Wei, and Keng Leng Siau. 1993. User-Database Interface: The Effect of Abstraction Levels on Query Performance. *MIS Quarterly* 17, 4 (Dec 1993), 441. https://doi.org/10.2307/249587

[35] Hock Chuan Chan, Kwok Kee Wei, and Keng Leng Siau. 1994. An empirical study on end-users' update performance for different abstraction levels. *International Journal of Human-Computer Studies* 41, 3 (1994), 309 – 328. https://doi.org/10.1006/ijhc.1994.1061

[36] H. C. Chan, K. K. Wei, and K. L. Siau. 1995. The effect of a database feedback system on user performance. *Behaviour & Information Technology* 14, 3 (May 1995), 152–162. https://doi.org/10.1080/01449299508914642

[37] Peter Pin-Shan Chen. 1976. The Entity-relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems* 1, 1 (March 1976), 9–36. https://doi.org/10.1145/320434.320440

[38] E. F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13, 6 (June 1970), 377–387. https://doi.org/10.1145/362384.362685

[39] Thomas Connolly and Carolyn Begg. 2015. *Database Systems (6th. ed.)*. Pearson.

[40] Mohammad Dadashzadeh. 2003. A Simpler Approach to Set Comparison Queries in SQL. *Journal of Information Systems Education* 14, 4 (2003), 345–348. http://jise.org/Volume14/n4/JISEv14n4p345.pdf

[41] Mohammad Dadashzadeh. 2007. Teaching Tip: Recursive Joins to Query Data Hierarchies in Microsoft Access. *Journal of Information Systems Education* 18, 1 (2007), 5–10. http://jise.org/Volume18/n1/JISEv18n1p5.pdf

[42] Mohammad Dadashzadeh. 2007. Teaching Tip: Specification and Enforcement of Semantic Integrity Constraints in Microsoft Access. *Journal of Information Systems Education* 18, 4 (2007), 393–398. http://jise.org/Volume18/n4/JISEv18n4p393.pdf

[43] J. Steve Davis. 1990. Experimental investigation of the utility of data structure and E-R diagrams in database query. *International Journal of Man-Machine Studies* 32, 4 (1990), 449 – 459. https://doi.org/10.1016/S0020-7373(05)80142-7

[44] S. W. Dietrich, D. Goelman, C. M. Borror, and S. M. Crook. 2015. An Animated Introduction to Relational Databases for Many Majors. *IEEE Transactions on Education* 58, 2 (May 2015), 81–89. https://doi.org/10.1109/TE.2014.2326834

[45] Doug Downey, Susan Dumais, Dan Liebling, and Eric Horvitz. 2008. Understanding the Relationship Between Searchers' Queries and Information Goals. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM '08)*. ACM, New York, NY, USA, 449–458. https://doi.org/10.1145/1458082.1458143

[46] A. Efendioglu and T. Yanpar Yelken. 2010. Programmed instruction versus meaningful learning theory in teaching basic structured query language (SQL) in computer lesson. *Computers and Education* 55, 3 (2010), 1287–1299. https://doi.org/10.1016/j.compedu.2010.05.026 cited By 9.

[47] Andrew Eisenberg, Jim Melton, Krishna Kulkarni, Jan-Eike Michels, and Fred Zemke. 2004. SQL:2003 has been published. *ACM SIGMOD Record* 33, 1 (mar 2004), 119. https://doi.org/10.1145/974121.974142

[48] Ramez Elmasri and Shamkant B. Navathe. 2016. *Fundamentals of Database Systems (7th. ed.)*. Pearson.

[49] Philip Garner and John Amedeo Mariani. 2015. Learning SQL in steps. *Journal of Systemics, Cybernetics and Informatics* 13, 4 (2015), 19–24.

[50] Don Goelman. 2008. Databases, non-majors and collaborative learning. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. ACM Press. https://doi.org/10.1145/1384271.1384281

[51] Venkat N. Gudivada, Jagadeesh Nandigam, and Yonglei Tao. 2007. Enhancing student learning in database courses with large data sets. In *2007 37th annual frontiers in education conference (FIE)*. IEEE. https://doi.org/10.1109/fie.2007.4418135

[52] Mario Guimaraes. 2005. Integrating the Kennesaw Database Courseware and Other Database Coursewares in Database Classes. In *Proceedings of the 43rd Annual Southeast Regional Conference - Volume 1 (ACM-SE 43)*. ACM, New York, NY, USA, 15–15. https://doi.org/10.1145/1167350.1167362

[53] Qiang Hao, David H. Smith IV, Naitra Iriumi, Michail Tsikerdekis, and Andrew J. Ko. 2019. A Systematic Investigation of Replications in Computing Education Research. *ACM Transactions on Computing Education* 19, 4, Article Article 42 (Aug. 2019), 18 pages. https://doi.org/10.1145/3345328

[54] Joseph E. Hollingsworth. 2008. Teaching Query Writing: An Informed Instruction Approach. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '08)*. ACM, New York, NY, USA, 351–351. https://doi.org/10.1145/1384271.1384393

[55] A. S. M. L. Hoque, G. M. M. Bashiry, and M. R. Uddin. 2014. Equivalence of Problems in Problem Based e-Learning of Database. In *2014 IEEE Sixth International Conference on Technology for Education*. 106–109. https://doi.org/10.1109/T4E.2014.23

[56] 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 arXiv:https://doi.org/10.1177/1049732305276687 PMID: 16204405.

[57] Ching-yu Huang. 2019. Integrative Curriculum for Teaching Databases. *Journal of Computing Sciences in Colleges* 34, 3 (Jan. 2019), 131–131. http://dl.acm.org/citation.cfm?id=3306465.3306487

[58] Gretchen Irwin, Wessel Lark, and Harvey Blackburn. 2012. Teaching Case: The Animal Genetic Resource Information Network (AnimalGRIN) Database: A Database Design & Implementation Case. *Journal of Information Systems Education* 23, 1 (2012), 19–28. http://jise.org/Volume23/n1/JISEv23n1p19.pdf

[59] ISO/IEC. 2016. ISO/IEC 9075-1:2016, "SQL - Part 1: Framework". https://www.iso.org/standard/63555.html

[60] ISO/IEC. 2016. ISO/IEC 9075-2:2016, "SQL - Part 2: Foundation". https://www.iso.org/standard/63556.html

[61] W.J. Kenny Jih, David A. Bradbard, Charles A. Snyder, and Nancy G.A. Thompson. 1989. The effects of relational and entity-relationship data models on query performance of end users. *International Journal of Man-Machine Studies* 31, 3 (Sep 1989), 257–267. https://doi.org/10.1016/0020-7373(89)90007-2

[62] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Technical Report. New York, NY, USA. https://doi.org/10.1145/2534860

[63] Nenad Jukic and Paul Gray. 2008. Using Real Data to Invigorate Student Learning. *SIGCSE Bulletin* 40, 2 (June 2008), 6–10. https://doi.org/10.1145/1383602.1383604

[64] Jai W. Kang, Qi Yu, Edward P. Holden, and Xumin Liu. 2019. Complementing Course Contents Between IT/CS: A Case Study on Database Courses. In *Proceedings of the 20th Annual SIG Conference on Information Technology Education (SIGITE '19)*. ACM, New York, NY, USA, 10–15. https://doi.org/10.1145/3349266.3351414

[65] Krishna Kulkarni and Jan-Eike Michels. 2012. Temporal features in SQL:2011. *ACM SIGMOD Record* 41, 3 (oct 2012), 34. https://doi.org/10.1145/2380776.2380786

[66] Verayuth Lertnattee and Perayot Pamonsinlapatham. 2017. Blended Learning for Improving Flexibility of Learning Structure Query Language (SQL). In *International Conference on Blended Learning (ICBL)*. Springer, 343–353. https://doi.org/10.1007/978-3-319-59360-9_30

[67] M. Luo. 2010. Study on design of graduation Practical Teaching management system based on SQL Server 2008 for Vocational College. In *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, Vol. 2. V2–292–V2–296. https://doi.org/10.1109/ICCASM.2010.5620476

[68] Maslin Masrom, Halimah Hasan, and Habsah Abdullah. 2007. Using a team-based approach in teaching database course. In *National Conference on Programming Science (ATUR 07)*, Vol. 5.

[69] Ramon A. Mata-Toledo and Carlos A. Reyes-Garcia. 2002. A Model Course for Teaching Database Administration with Personal Oracle 8i. *Journal of Computing Sciences in Colleges* 17, 3 (Feb. 2002), 125–130. http://dl.acm.org/citation.cfm?id=772636.772658

[70] Victor M. Matos and Rebecca Grasser. 2002. Teaching Tip A Simpler (and Better) SQL Approach to Relational Division. *Journal of Information Systems Education* 13, 2 (2002), 85–88. http://jise.org/Volume13/Pdf/085.pdf

[71] Victor M. Matos, Rebecca Grasser, and Paul Jalics. 2006. The Case of the Missing Tuple: Teaching the SQL Outer-join Operator to Undergraduate Information Systems Students. *Journal of Computing Sciences in Colleges* 22, 1 (Oct. 2006), 23–32. http://dl.acm.org/citation.cfm?id=1181811.1181814

[72] L. I. McCann. 2003. On making relational division comprehensible. In *Proceedings of the 2003 33rd Annual Frontiers in Education Conference (FIE)*, Vol. 2. F2C–6. https://doi.org/10.1109/FIE.2003.1264699

[73] Kirby McMaster, Samuel Sambasivam, Steven Hadfield, and Stuart Wolthuis. 2013. Relational Algebra and SQL: Better Together. *Information Systems Education Journal* 11, 1 (2013), 4–13. http://isedj.org/2013-11/N1/ISEDJv11n1p4.pdf

[74] Janet Metcalfe. 2017. Learning from Errors. *Annual Review of Psychology* 68, 1 (2017), 465–489. https://doi.org/10.1146/annurev-psych-010416-044022 PMID: 27648988.

[75] Zhengjie Miao, Sudeepa Roy, and Jun Yang. 2019. Explaining Wrong Queries Using Small Examples. In *Proceedings of the 2019 ACM Conference on Management of Data (SIGMOD) (SIGMOD '19)*. ACM, New York, NY, USA, 503–520. https://doi.org/10.1145/3299869.3319866

[76] Antonija Mitrovic. 1998. Learning SQL with a Computerized Tutor. In *Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '98)*. ACM, New York, NY, USA, 307–311. https://doi.org/10.1145/273133.274318

[77] Antonija Mitrovic. 2003. An intelligent SQL tutor on the web. *International Journal of Artificial Intelligence in Education* 13, 2-4 (2003), 173–197. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85013603633&partnerID=40&md5=7e8de8bf8e5659c6720ec83f64e2e388

[78] Antonija Mitrovic. 2006. Large-Scale Deployment of three intelligent web-based database tutors. *Journal of Computing and Information Technology* 14, 4 (2006), 275–281. https://doi.org/10.2498/cit.2006.04.02

[79] Keith Moss. 2010. DataBase teaching tools. In *IEEE EDUCON 2010 Conference*. IEEE. https://doi.org/10.1109/educon.2010.5492432

[80] Barzan Mozafari, Kai Zeng, and Carlo Zaniolo. 2010. K*SQL: A Unifying Engine for Sequence Patterns and XML. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*. Association for Computing Machinery, New York, NY, USA, 1143–1146. https://doi.org/10.1145/1807167.1807302

[81] David Olsen and Karina Hauser. 2007. Teaching Tip: Teaching Advanced SQL Skills: Text Bulk Loading. *Journal of Information Systems Education* 18, 4 (2007), 399–402. http://jise.org/Volume18/n4/JISEv18n4p399.pdf

[82] T. Permpool, S. Nalintippayawong, and K. Atchariyachanvanich. 2019. Interactive SQL Learning Tool with Automated Grading using MySQL Sandbox. In *2019 IEEE 6th International Conference on Industrial Engineering and Applications, ICIEA 2019*. 928–932. https://doi.org/10.1109/IEA.2019.8715175

[83] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic Mapping Studies in Software Engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08)*. BCS Learning & Development Ltd., Swindon, UK, 68–77. http://dl.acm.org/citation.cfm?id=2227115.2227123

[84] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1 – 18. https://doi.org/10.1016/j.infsof.2015.03.007

[85] Hongsiri Piyayodilokchai, Pintip Ruenwongsa, Watcharee Ketpichainarong, Parames Laosinchai, and Patcharin Panjaburee. 2011. Promoting Students' Understanding of SQL in a Database Management Course: A Learning Cycle Approach. *The International Journal of Learning: Annual Review* 17, 11 (2011), 325–338. https://doi.org/10.18848/1447-9494/cgp/v17i11/47345

[86] Julia Coleman Prior. 2003. Online Assessment of SQL Query Formulation Skills. In *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20 (ACE '03)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 247–256. http://dl.acm.org/citation.cfm?id=858403.858433

[87] 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

[88] Christine F. Reilly. 2018. Experience with Active Learning and Formative Feedback for a SQL Unit. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE. https://doi.org/10.1109/fie.2018.8659173

[89] Karen Renaud and Judy Van Biljon. 2004. Teaching SQL - Which Pedagogical Horse for This Course?. In *British National Conference on Databases (BNCOD)*. Springer, 244–256. https://doi.org/10.1007/978-3-540-27811-5_22

[90] Sangkyu Rho and Salvatore T March. 1997. An analysis of semantic overload in database access systems using multi-table query formulation. *Journal of Database Management* 8, 2 (1997), 3–15. https://www.igi-global.com/gateway/article/51176

[91] Gordon Russell and Andrew Cumming. 2004. Improving the Student Learning Experience for SQL Using Automatic Marking. In *Cognition and Exploratory Learning in Digital Age (CELDA)*. 281–288.

[92] Edward Sciore. 2007. SimpleDB: A Simple Java-based Multiuser System for Teaching Database Internals. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)*. ACM, New York, NY, USA, 561–565. https://doi.org/10.1145/1227310.1227498

[93] Bilal Shebaro. 2018. Using Active Learning Strategies in Teaching Introductory Database Courses. *Journal of Computing Sciences in Colleges* 33, 4 (April 2018), 28–36. http://dl.acm.org/citation.cfm?id=3199572.3199576

[94] Yasin N. Silva, Isadora Almeida, and Michell Queiroz. 2016. SQL: From Traditional Databases to Big Data. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE) (SIGCSE '16)*. ACM, New York, NY, USA, 413–418. https://doi.org/10.1145/2839509.2844560

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

[96] M. Soflano, T.M. Connolly, and T. Hainey. 2015. Learning style analysis in adaptive GBL application to teach SQL. *Computers and Education* 86 (2015), 105–119. https://doi.org/10.1016/j.compedu.2015.02.009

[97] Song Jian-gong. 2010. Design and application of collaborative learning system based on web to database experiment teaching. In *2010 International Conference on Educational and Information Technology*, Vol. 3. V3–140–V3–143. https://doi.org/10.1109/ICEIT.2010.5608405

[98] Toni Taipalus. 2019. Teaching Tip: A Notation for Planning SQL Queries. *Journal of Information Systems Education* 30, 3 (2019), 160–166. http://jise.org/Volume30/n3/JISEv30n3p160.pdf

[99] 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

[100] 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

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

[102] Josh Tenenberg and Robert McCartney. 2010. Why Discipline Matters in Computing Education Scholarship. *ACM Transactions on Computing Education* 9, 4 (2010), 1–7. https://doi.org/10.1145/1656255.1656256

[103] The Joint Task Force on Computing Curricula. 2015. *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. Technical Report. New York, NY, USA. https://dl.acm.org/citation.cfm?id=2965631

[104] Richard C. Thomas and Rebecca Mancy. 2004. Use of large databases for group projects at the nexus of teaching and research. In *Proceedings of the 9th annual conference on Innovation and technology in computer science education (ITiCSE)*. ACM Press. https://doi.org/10.1145/1007996.1008039

[105] 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://www.acm.org/binaries/content/assets/

education/curricula-recommendations/is-2010-acm-final.pdf

[106] Heikki Topi, Joseph S Valacich, and Jeffrey A Hoffer. 2005. The effects of task complexity and time availability limitations on human performance in database query tasks. *International Journal of Human-Computer Studies* 62, 3 (2005), 349–379. https://doi.org/10.1016/j.ijhcs.2004.10.003

[107] Susan D. Urban and Suzanne W. Dietrich. 2001. Advanced Database Concepts for Undergraduates: Experience with Teaching a Second Course. *SIGCSE Bull.* 33, 1 (Feb. 2001), 357–361. https://doi.org/10.1145/366413.364648

[108] Leo Vijayasarathy and Gretchen Casterella. 2016. The Effects of Information Request Language and Template Usage on Query Formulation. *Journal of the Association for Information Systems* 17, 10 (Oct 2016), 674–707. https://doi.org/10.17705/1jais.00440

[109] Adam H. Villa. 2016. Big Data: Motivating the Development of an Advanced Database Systems Course. *Journal of Computing Sciences in Colleges* 31, 3 (2016), 119–128. https://dl.acm.org/doi/abs/10.5555/2835377.2835395

[110] Paul J. Wagner, Elizabeth Shoop, and John V. Carlis. 2003. Using Scientific Data to Teach a Database Systems Course. In *Proceedings of the 34th ACM Technical Symposium on Computer Science Education (SIGCSE) (SIGCSE '03)*. ACM, New York, NY, USA, 224–228. https://doi.org/10.1145/611892.611975

[111] R. T. Watson. 2006. The Essential Skills of Data Modeling. *Journal of Information Systems Education* 17, 1 (2006), 39–42. http://jise.org/Volume17/n1/JISEv17n1p39.pdf

[112] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. 2005. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering* 11, 1 (nov 2005), 102–107. https://doi.org/10.1007/s00766-005-0021-6

[113] Coleen R. Wilder and Ceyhun O. Ozgur. 2015. Business Analytics Curriculum for Undergraduate Majors. *INFORMS Transactions on Education* 15, 2 (Jan. 2015), 180–187. https://doi.org/10.1287/ited.2014.0134

[114] Claes Wohlin. 2014. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14)*. ACM, New York, NY, USA, Article 38, 10 pages. https://doi.org/10.1145/2601248.2601268

[115] Claes Wohlin, Per Runeson, Paulo Anselmo da Mota Silveira Neto, Emelie Engström, Ivan do Carmo Machado, and Eduardo Santana de Almeida. 2013. On the reliability of mapping studies in software engineering. *Journal of Systems and Software* 86, 10 (2013), 2594 – 2610. https://doi.org/10.1016/j.jss.2013.04.076

[116] David A. Wolff. 2001. MySQL, PostgreSQL, and PHP: Open Source Technologies for a Database Management Course. *Journal of Computing Sciences in Colleges* 17, 2 (Dec. 2001), 91–92. http://dl.acm.org/citation.cfm?id=775339.775358

[117] Donna Wright. 2013. Database: CSI, Inc. *Journal of Computing Sciences in Colleges* 28, 5 (May 2013), 80–81. http://dl.acm.org/citation.cfm?id=2458569.2458586

[118] Wu Da-sheng and Wu Sheng-yu. 2010. Dynamically maintain the teaching examples of triggers and stored procedures about the course of database application. In *2010 2nd International Conference on Education Technology and Computers*, Vol. 1. V1–525–V1–527. https://doi.org/10.1109/ICETC.2010.5529193

[119] M. Y. . Yen and R. W. Scamell. 1993. A human factors experimental comparison of SQL and QBE. *IEEE Transactions on Software Engineering* 19, 4 (April 1993), 390–409. https://doi.org/10.1109/32.223806

[120] M.-H. Ying, N.-W. Chi, and Y. Hong. 2012. Enhancement of learning by using an Online SQL Learning System with Automatic Checking Mechanism. *International Journal of Digital Content Technology and its Applications* 6, 6 (Apr 2012), 239–248. https://doi.org/10.4156/jdcta.vol6.issue6.28

[121] Kwok-Bun Yue. 2013. Using a Semi-Realistic Database to Support a Database Course. *Journal of Information Systems Education* 24, 4 (2013), 327–336. http://jise.org/Volume24/n4/JISEv24n4p327.pdf

[122] Liu Yuelan, Liao Yiwei, Huang Yuyan, and Liu Yuefan. 2011. Study on Teaching Methods of Database Application Courses. *Procedia Engineering* 15 (2011), 5425–5428. https://doi.org/10.1016/j.proeng.2011.08.1006

[123] Abe Zeid and Sagar Kamarthi. 2008. Best teaching practices in database courses for engineering students. *International Journal of Engineering Education* 24, 5 (2008), 980–989.

[124] Fred Zemke. 2012. What's new in SQL:2011. *ACM SIGMOD Record* 41, 1 (2012), 67–73. https://doi.org/10.1145/2206869.2206883

A LIST OF PRIMARY STUDIES

Selected primary studies

ID	Publication
PS01	Abelló, A., Burgués, X., Casany, M.J., Martín, C., Quer, C., Rodríguez, M.E., Romero, Ó, & Urpí, T. (2016). A software tool for e-assessment of relational database skills. *International Journal of Engineering Education 32*(3), 1289-1323. http://hdl.handle.net/2117/89668
PS02	Ahadi, A., Behbood, V., Vihavainen, A., Prior, J., & Lister, R. (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 Computer Science Education (SIGCSE)* (pp. 401–406). ACM. doi.org/doi:10.1145/2839509.2844640
PS03	Ahadi, A., Prior, J., Behbood, V., & Lister, R. (2015). A Quantitative Study of the Relative Difficulty for Novices of Writing Seven Different Types of SQL Queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)* (pp. 201—206). ACM. doi.org/doi:10.1145/2729094.2742620
PS04	Ahadi, A., Prior, J., Behbood, V., & Lister, R. (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)* (pp. 272—277). ACM. doi.org/doi:10.1145/2899415.2899464
PS05	AL-Salmi, A. (2018). A web-based semi-automatic assessment tool for formulating basic SQL statements: Point-and-click interaction method. In *Proceedings of the 10th International Conference on Computer Supported Education (CSEDU)* (pp. 191–198). doi.org/10.5220/0006671501910198
PS06	Al-Shuaily, H., & Renaud, K. (2010). SQL patterns - a new approach for teaching SQL. In *8th HEA Workshop on Teaching, Learning and Assessment of Databases (TLAD)* (pp. 29–40). rke.abertay.ac.uk/ws/portalfiles/portal/9116027/TLAD2010Proceedings.pdf
PS07	Allen, G. N., & March, S. T. (2006). The effects of state-based and event-based data representation on user performance in query formulation tasks. *MIS Quarterly 30*(2), 269–290. doi.org/10.2307/25148731
PS08	Allen, G. N., & Parsons, J. (2010). Is query reuse potentially harmful? Anchoring and adjustment in adapting existing database queries. *Information Systems Research 21*(1), 56–77. doi.org/10.1287/isre.1080.0189
PS09	Amadio, W. (2003). The dilemma of team learning: An assessment from the SQL programming classroom. In *Proceedings of the Annual Meeting of the Decision Sciences Institute* (pp. 823—828).
PS10	Aversano, L., Canfora, G., De Lucia, A., & Stefanucci, S. (2002). Understanding SQL through iconic interfaces. In *Proceedings of the IEEE 26th Annual International Computer Software and Applications (COMPSAC)* (pp. 703-–708). IEEE. doi.org/10.1109/CMPSAC.2002.1045084
PS11	Axelsen, M., Borthick, A. F., & Bowen, P. L. (2001). A model for and the effects of information request ambiguity on end-user query performance. *ICIS 2001 Proceedings*, p.68. aisel.aisnet.org/icis2001/68
PS12	Bhangdiya, A., Chandra, B., Kar, B., Radhakrishnan, B., Reddy, K.V.M., Shah, S., & Sudarshan, S. (2015). The XDa-TA system for automated grading of SQL query assignments. In *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering (ICDE)* (pp. 1468—1471). IEEE. doi.org/10.1109/icde.2015.7113403
PS13	Boisvert, C., Domdouzis, K., & License, J. (2018). A comparative analysis of student SQL and relational database knowledge using automated grading tools. In *Proceedings of the 2018 IEEE International Symposium on Computers in Education (SIIE)*. IEEE. doi.org/10.1109/siie.2018.8586684
PS14	Borthick, A., Bowen, P. L., Jones, D. R., & Tse, M. H. K. (2001). The effects of information request ambiguity and construct incongruence on query development. *Decision Support Systems 32*(1), 3-25. doi.org/10.1016/s0167-9236(01)00097-5
PS15	Bowen, P., O'Farrell, R., & Rohde, F. (2004). How does your model grow? an empirical investigation of the effects of ontological clarity and application domain size on query performance. *ICIS 2004 Proceedings*, p.7. aisel.aisnet.org/icis2004/7/
PS16	Bowen, P. L., O'Farrell, R. A., & Rohde, F. H. (2006). Analysis of competing data structures: Does ontological clarity produce better end user query performance. *Journal of the Association for Information Systems 7*(22), 514-544. doi.org/10.17705/1jais.00098
PS17	Bowen, P. L., O'Farrell, R. A., & Rohde, F. H. (2009). An Empirical Investigation of End-User Query Development: The Effects of Improved Model Expressiveness vs. Complexity. *Information Systems Research 20*(4), 565–584. doi.org/10.1287/isre.1080.0181
PS18	Brass, S., & Goldberg, C. (2006). Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software 79*(5), 630-644. doi.org/10.1016/j.jss.2005.06.028
PS19	Caldeira, C. P. (2008). Teaching SQL: A case study. In *Proceedings of the 2008 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)* (pp. 340-–340). ACM. doi.org/10.1145/1384271.1384382
PS20	Casterella, G.I., & Vijayasarathy, L. (2013). An Experimental Investigation of Complexity in Database Query Formulation Tasks. *Journal of Information Systems Education 24*(3), 211-221. http://jise.org/Volume24/24-3/pdf/Vol24-3pg211.pdf
PS21	Casterella, G.I., & Vijayasarathy, L. (2019). Query Structure and Data Model Mapping Errors in Information Retrieval Tasks. *Journal of Information Systems Education 30*(3), 178-190. http://jise.org/Volume30/n3/JISEv30n3p178.pdf
PS22	Cembalo, M., De Santis, A., & Ferraro Petrillo, U. (2011). SAVI: A New System for Advanced SQL Visualization. In *Proceedings of the 2011 ACM Conference on Information Technology Education (SIGITE)* (pp. 165–170). ACM. doi.org/10.1145/2047594.2047641
PS23	Chan, H.C., Tan, B.C.Y., & Wei, K.K. (1999). Three important determinants of user performance for database retrieval. *International Journal of Human-Computer Studies 51*(5), 895-918. doi.org/10.1006/ijhc.1999.0272
PS24	Chan, H.C., Wei, K.K., & Siau, K.L. (1993). User-Database Interface: The Effect of Abstraction Levels on Query Performance. *MIS Quarterly 17*(4), 441. doi.org/10.2307/249587
PS25	Chandra, B., Chawda, B., Kar, B., Reddy, K.V., Shah, S., & Sudarshan, S. (2015). Data Generation for Testing and Grading SQL Queries. *The VLDB Journal 24*(6), 731-755. doi.org/10.1007/s00778-015-0395-0
PS26	Chandra, B., Mathew, J., Radhakrishnan, B., Acharya, S., & Sudarshan, S. (2016). Partial Marking for Automated Grading of SQL Queries. In *Proceedings of the VLDB Endowment 9*(13) (pp. 1541–1544). doi.org/10.14778/3007263.3007304
PS27	Dadashzadeh, M. (2003). Teaching Tip: A Simpler Approach to Set Comparison Queries in SQL. *Journal of Information Systems Education 14*(4), 345-348. http://jise.org/Volume14/n4/JISEv14n4p345.pdf
PS28	Dadashzadeh, M. (2007). Teaching Tip: Recursive Joins to Query Data Hierarchies in Microsoft Access. *Journal of Information Systems Education 18*(1), 5-10. http://jise.org/Volume18/n1/JISEv18n1p5.pdf

Selected primary studies (cont.)

ID	Publication
PS29	Dadashzadeh, M. (2007). Teaching Tip: Specification and Enforcement of Semantic Integrity Constraints in Microsoft Access. *Journal of Information Systems Education 18*(4), 393-398. http://jise.org/Volume18/n4/JISEv18n4p393.pdf
PS30	Danaparamita, J., & Gatterbauer, W. (2011). QueryViz: Helping Users Understand SQL Queries and Their Patterns. In *Proceedings of the 14th ACM International Conference on Extending Database Technology (EDBT)* (pp. 558–561). ACM. doi.org/10.1145/1951365.1951440
PS31	Davis, J.S. (1990). Experimental investigation of the utility of data structure and E-R diagrams in database query. *International Journal of Man-Machine Studies 32*(4), 449-459. doi.org/10.1016/S0020-7373(05)80142-7
PS32	Dean, T.J., & Milani, W.G. (1995). Transforming a database systems and design course for non computer science majors. In *Proceedings of the 1995 25th Annual Frontiers in Education Conference (FIE)*. IEEE. doi.org/10.1109/fie.1995.483191
PS33	Dietrich, S.W., Goelman, D., Borror, C.M., & Crook, S.M. (2015). An Animated Introduction to Relational Databases for Many Majors. *IEEE Transactions on Education 58*(2), 81-89. doi.org/10.1109/TE.2014.2326834
PS34	Dietrich, S.W., & Urban, S.D. (1996). Database Theory in Practice: Learning from Cooperative Group Projects. In *Proceedings of the 27th ACM Technical Symposium on Computer Science Education (SIGCSE)* (pp. 112–116). ACM. doi.org/10.1145/236452.236520
PS35	Do, Q., Agrawal, R.K., Rao, D., & Gudivada, V.N. (2014). Automatic generation of SQL queries. In *Proceedings of the 121st ASEE Annual Conference and Exposition*.
PS36	Dollinger, R., & Melville, N.A. (2011). Semantic evaluation of SQL queries. In *Proceedings of the 2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing (ICCP)* (pp. 57–64). IEEE. doi.org/10.1109/ICCP.2011.6047844
PS37	Fekete, A. (2005). Teaching Transaction Management with SQL Examples. In *Proceedings of the 2005 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)* (pp. 163–167). ACM. doi.org/10.1145/1067445.1067492
PS38	Fong, J., Lee, J., & Fong, A. (2005). Student Centered Knowledge Level Analysis for eLearning for SQL. In *Advances in Web-Based Learning (ICWL) 2005*, (pp. 174–185). Springer. doi.org/10.1007/11528043_17
PS39	Garner, P., & Mariani, J.A. (2015). Learning SQL in steps. *Journal of Systemics, Cybernetics and Informatics 13*(4), 19-24.
PS40	Green, G.G. (2005). Teaching Case: Greta's Gym: A Teaching Case for Term-Long Database Projects. *Journal of Information Systems Education 16*(4), 387-390. http://jise.org/Volume16/n4/JISEv16n4p387.pdf
PS41	Gudivada, V.N., Nandigam, J., & Tao, Y. (2007). Enhancing student learning in database courses with large data sets. In *Proceedings of the 2007 37th Annual Frontiers in Education Conference (FIE)*. IEEE. doi.org/10.1109/fie.2007.4418135
PS42	Hardt, R., & Gutzmer, E. (2017). Database Query Analyzer (DBQA) - A Data-Oriented SQL Clause Visualization Tool. In *Proceedings of the 18th ACM Conference on Information Technology Education (SIGITE)*. ACM. doi.org/10.1145/3125659.3125688
PS43	Hollingsworth, J.E. (2008). Teaching Query Writing: An Informed Instruction Approach. In *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)* (pp. 351–351). ACM. doi.org/0.1145/1384271.1384393
PS44	Huang, C., & Morreale, P.A. (2016). A web-based, self-controlled mechanism to support students learning SQL. In *Proceedings of the 2016 IEEE Integrated STEM Education Conference (ISEC)* (pp. 218–223). IEEE. doi.org/10.1109/ISECon.2016.7457536
PS45	Hvorecký, J., Drlík, M., & Munk, M. (2010). Enhancing database querying skills by choosing a more appropriate interface. In *Proceedings of the IEEE Global Engineering Education Conference (EDUCON)* (pp. 1897–1905). IEEE. doi.org/10.1109/EDUCON.2010.5492434
PS46	Irwin, G., Wessel, L., & Blackburn, H. (2012). Teaching Case: The Animal Genetic Resource Information Network (AnimalGRIN) Database: A Database Design & Implementation Case. *Journal of Information Systems Education 23*(1), 19-28. http://jise.org/Volume23/n1/JISEv23n1p19.pdf
PS47	Julavanich, T., Nalintippayawong, S., & Atchariyachanvanich, K. (2019). RSQLG: The Reverse SQL Question Generation Algorithm. In *Proceedings of the 2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA)* (pp. 908–912). IEEE. doi.org/10.1109/IEA.2019.8715233
PS48	Kawash, J. (2014). Formulating Second-order Logic Conditions in SQL. In *Proceedings of the 15th ACM Conference on Information Technology Education (SIGITE)* (pp. 115-120). ACM. doi.org/10.1145/2656450.2656452
PS49	Ke, H., Zhang, G., & Yan, H. (2009). Automatic Grading System on SQL Programming. In *Proceedings of the 2009 IEEE International Conference on Scalable Computing and Communications (ScalCom)* (pp. 537–540). IEEE. doi.org/10.1109/EmbeddedCom-ScalCom.2009.105
PS50	Kenny, C., & Pahl, C. (2005). Automated Tutoring for a Database Skills Training Environment. In *Proceedings of the 36th ACM Technical Symposium on Computer Science Education (SIGCSE)* (pp. 58–62). ACM. doi.org/10.1145/1047344.1047377
PS51	Leitheiser, R.L., & March, S.T. (1996). The Influence of Database Structure Representation on Database System Learning and Use. *Journal of Management Information Systems 12*(4), 187-213. doi.org/10.1080/07421222.1996.11518106
PS52	Lertnattee, V., & Pamonsinlapatham, P. (2017). Blended learning for improving flexibility of learning structure query language (SQL). In *International Conference on Blended Learning* (pp. 343–353). Springer. doi.org/10.1007/978-3-319-59360-9_30
PS53	Martín, C., Urpí, T., Casany, M.J., Burgués, X., Quer, C., Rodríguez, M.E., & Abelló, A. (2013). Improving learning in a database course using collaborative learning techniques. *International Journal of Engineering Education 29*(4), 986-997.
PS54	Matos, V.M., & Grasser, R. (2002). Teaching tip: A Simpler (and Better) SQL Approach to Relational Division. *Journal of Information Systems Education 13*(2), 19-28. http://jise.org/Volume13/n2/JISEv13n2p85.pdf
PS55	Matos, V.M., Grasser, R., & Jalics, P. (2006). The Case of the Missing Tuple: Teaching the SQL Outer-join Operator to Undergraduate Information Systems Students. *Journal of Computing Sciences in Colleges 22*(1), 23-32. http://dl.acm.org/citation.cfm?id=1181811.1181814
PS56	McCann, L.I. (2003). On making relational division comprehensible. In *Proceedings of the 2003 33rd Annual Frontiers in Education Conference (FIE)*. IEEE. doi.org/10.1109/FIE.2003.1264699
PS57	Miao, Z., Roy, S., & Yang, J. (2019). Explaining Wrong Queries Using Small Examples. In *Proceedings of the 2019 ACM Conference on Management of Data (SIGMOD)* (pp. 503–520). ACM. doi.org/10.1145/3299869.3319866
PS58	Mills, R.J., Dupin-Bryant, P.A., Johnson, J.D., & Beaulieu, T.Y. (2015). Examining learning styles and perceived benefits of analogical problem construction on SQL knowledge acquisition. *Journal of Information Systems Education 26*(3), 203-217. http://jise.org/Volume26/n3/JISEv26n3p203.pdf
PS59	Morris, S.A. (2008). Teaching Case: Remote Services, Inc. *Journal of Information Systems Education 19*(2), 147-156. http://jise.org/Volume19/n2/JISEv19n2p147.pdf
PS60	Murray, M., & Guimaraes, M. (2008). Animated Database Courseware: Using Animations to Extend Conceptual Understanding of Database Concepts. *Journal of Computing Sciences in Colleges 24*(2), 144-150. http://dl.acm.org/citation.cfm?id=1409823.1409855

Selected primary studies (cont.)

ID	Publication
PS61	Myers, C., & Douglas, P. (2007). The Un-Structured Student. In *Proceedings of the 24th British National Conference on Databases (BNCOD)* (pp. 3–9). doi.org/10.1109/BNCOD.2007.22
PS62	Ortiz, J., Dietrich, S.W., & Chaudhari, M.B. (2012). Learning from Database Performance Benchmarks. *Journal of Computing Sciences in Colleges 27*(4), 151-158. http://dl.acm.org/citation.cfm?id=2167431.2167457
PS63	Oussena, S., & Dunckley, L. (2007). Adopting Student-Centred Approach to Advanced Database Teaching. In *Proceedings of the 24th British National Conference on Databases (BNCOD)* (pp. 10–14). doi.org/10.1109/BNCOD.2007.5
PS64	Pahl, C., Barrett, R., & Kenny, C. (2004). Supporting Active Database Learning and Training Through Interactive Multimedia. In *Proceedings of the 2004 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)* (pp. 27–31). ACM. doi.org/10.1145/1007996.1008007
PS65	Petrov, P., & Djolev, D. (2015). Combining the procedural and the set-based approaches in the teaching of SQL select statements in the introductory databases course. In *Proceedings of the International Scientific Conference Computer Science* (pp. 249–254).
PS66	Qian, G. (2018). Teaching SQL: A Divide-and-conquer Method for Writing Queries. *Journal of Computing Sciences in Colleges 33*(4), 37-44. http://dl.acm.org/citation.cfm?id=3199572.3199577
PS67	Randolph, G.B. (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)* (pp. 234–236). ACM. doi.org/10.1145/947121.947174
PS68	Reilly, C.F. (2018). Experience with Active Learning and Formative Feedback for a SQL Unit. In *Proceedings of the 2018 48th Annual Frontiers in Education Conference (FIE)*. IEEE. doi.org/10.1109/FIE.2018.8659173
PS69	Renaud, K., & Van Biljon, J. (2004). Teaching SQL - Which Pedagogical Horse for This Course? In *Proceedings of the 21st British National Conference on Databases (BNCOD)* (pp. 244-256). doi.org/10.1007/978-3-540-27811-5_22
PS70	Russell, G., & Cumming, A. (2004). Improving the Student Learning Experience for SQL Using Automated Marking. In *Proceedings of the International Conference on Cognition and Exploratory Learning in Digital Age (CELDA)* (pp. 281–288).
PS71	Sastry, M.K.S. (2015). An Effective Approach for Teaching Database Course. *International Journal of Learning, Teaching and Educational Research 12*(1). https://www.ijlter.org/index.php/ijlter/article/download/357/162
PS72	Seyed-Abassi, B. (1993). A SQL Project As a Learning Method in a Database Course. In *Proceedings of the 1993 ACM Conference on Computer Personnel Research (SIGCPR)* (pp. 291–297). ACM. doi.org/10.1145/158011.158238
PS73	Silva, Y.N., Almeida, I., & Queiroz, M. (2016). SQL: From Traditional Databases to Big Data. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE)* (pp. 413–418). ACM. doi.org/http://doi.acm.org/10.1145/2839509.2844560
PS74	Smelcer, J.B. (1995). User errors in database query composition. *International Journal of Human-Computer Studies 42*(4), 353-381. doi.org/10.1006/ijhc.1995.1017
PS75	Soflano, M., Connolly, T.M., & Hainey, T. (2015). An application of adaptive games-based learning based on learning style to teach SQL. *Computers & Education 86*, 192-211. doi.org/10.1016/j.compedu.2015.03.015
PS76	Stajduhar, I., & Mausa, G. (2015). Using string similarity metrics for automated grading of SQL statements. In *Proceedings of the 2015 38th International Convention on Information, Communication and Electronic Technology (MIPRO)* (pp. 1250–1255). doi.org/10.1109/MIPRO.2015.7160467
PS77	Sundin, L. & Cutts, Q. (2019). Is it feasible to teach query programming in three different languages in a single session?: A study on a pattern-oriented tutorial and cheat sheets. In *Proceedings of the 1st UK & Ireland Computing Education Research Conference (UKICER)*, p. 7. ACM. doi.org/10.1145/3351287.3351293
PS78	Taipalus, T. (2019). Teaching Tip: A Notation for Planning SQL Queries. *Journal of Information Systems Education 30*(3), 160-166. http://jise.org/Volume30/n3/JISEv30n3p160.pdf
PS79	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)* (pp. 198–203). ACM. doi.org/10.1145/3287324.3287359
PS80	Taipalus, T., Siponen, M., & Vartiainen, T. (2018). Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education 18*(3), p. 15. doi.org/10.1145/3231712
PS81	Ullman, J.D. (2003). Improving the Efficiency of Database-system Teaching. In *Proceedings of the 2003 ACM International Conference on Management of Data (SIGMOD)* (pp. 1–3). ACM. doi.org/10.1145/872757.872759
PS82	Vijayasarathy, L., & Casterella, G.I. (2016). The Effects of Information Request Language and Template Usage on Query Formulation. *Journal of the Association for Information Systems 17*(10), 674-707. doi.org/10.17705/1jais.00440
PS83	Wagner, P.J., Shoop, E., & Carlis, J.V. (2003). Using scientific data to teach a database systems course. In *Proceedings of the 34th ACM Technical Symposium on Computer Science Education (SIGCSE)* (pp. 224–228). ACM. doi.org/10.1145/611892.611975
PS84	Watson, H.J., & Hoffer, J.A. (2003). Teradata university network: A new resource for teaching large data bases and their applications. *Communications of the Association for Information Systems 12*(1), 131-144. doi.org/10.17705/1cais.01209
PS85	Wu, P.Y., Baugh, J.M., & Harvey, V.J. (2005). Teaching SQL in database management for adult continuing education. In *Proceedings of the 2005 Information Systems Education Conference (ISECON)*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.561.522&rep=rep1&type=pdf
PS86	Yen, M.Y. & Scamell, R.W. (1993). A human factors experimental comparison of SQL and QBE. *IEEE Transactions on Software Engineering 19*(4), 390-409. doi.org/10.1109/32.223806
PS87	Ying, M. & Hong, Y. (2011). The development of an online SQL learning system with automatic checking mechanism. In *Proceedings of the 7th IEEE International Conference on Networked Computing and Advanced Information Management (NCM)* (pp. 346–351). IEEE.
PS88	Yue, K.B. (2013). Using a Semi-Realistic Database to Support a Database Course. *Journal of Information Systems Education 24*(4), 327-336. http://jise.org/Volume24/n4/JISEv24n4p327.pdf
PS89	Zilligen, R., & Hidayat, A. (2008). A Misconception Module to a Database Courseware. In *Proceedings of the 46th ACM Annual Southeast Regional Conference (ACM-SE)* (pp. 529–530). ACM. doi.org/10.1145/1593105.1593250

B PRIMARY STUDY CLASSIFICATION

Primary study classification by topic and research type facets, *PS*s from primary study identifiers are omitted for brevity

	Evaluation research	Solution proposal	Replication study	Philosophical paper	Opinion paper	Experience report
Student errors	02, 03, 04, 24, 74, 79, 80	61, 73		18		67
Exercise database	05, 07, 11, 14, 15, 16, 17, 20, 21, 23, 31, 51, 82, 88	57, 83			62	41, 46, 68
Specific teaching approach	48	27, 54, 55, 56			28, 29, 37	86
Non-specific teaching approach	08, 13, 53	43, 58, 64			09, 38, 40, 60, 81, 89	19, 32, 33, 34, 52, 59, 63, 70, 71, 72
Patterns and visualization	10, 45	06, 22, 30, 39, 42, 66, 78		77		65, 69
Teacher workload	75	01, 12, 25, 26, 35, 36, 44, 47, 49, 76, 84, 87			85	50

C NUMBER OF PARTICIPANTS IN EACH PRIMARY STUDY

Number and type of participants in each primary study; primary studies that are not listed involved no participants, or did not report participant numbers

Study	Evidence	Study	Evidence
PS01	1,584 students	PS31	116 subjects
PS02	approximately 161,000 queries from approximately 2,300 undergraduate students (possibly same data as PS04)	PS33	75 students and 32 students
PS03	986 students	PS44	21 students
PS04	approximately 161,000 queries from approximately 2,300 undergraduate students (possibly same data as PS02), out of which 551 queries from 321 students studied in more detail	PS45	116 students
PS05	60 undergraduate students	PS51	52 graduate business students
PS06	3 postgraduate students	PS52	4 graduated [sic] students
PS07	342 subjects	PS53	928 grades from 6 semesters
PS08	157 students	PS55	22 undergraduate information systems students
PS10	88 undergraduate telecommunication students	PS57	approximately 170 undergraduate students
PS11	95 advanced undergraduate and postgraduate students	PS58	80 students
PS13	103 students	PS66	120 students
PS14	23 graduate students	PS70	over 300 undergraduate students
PS15	81 advanced undergraduate and graduate commerce students (possibly same data as PS16 and PS17)	PS74	17 undergraduate business administration students
PS16	81 advanced undergraduate and graduate commerce students (possibly same data as PS15 and PS17)	PS75	120 higher education students
PS17	81 advanced undergraduate and graduate commerce students (possibly same data as PS15 and PS16)	PS76	393 student answers
PS19	48 students	PS77	21 students
PS20	33 undergraduate junior and senior students in computer information systems department	PS79	approximately 123,000 queries, out of which 8,773 queries from 744 undergraduate computer science and information systems students studied in detail
PS21	63 undergraduate students	PS80	approximately 33,000 queries from 237 students
PS23	112 subjects, but not everyone participated in all experiments	PS82	63 students
PS24	47 subjects, out of which 24 used SQL	PS86	65 students
		PS88	186 students

PII

ERRORS AND COMPLICATIONS IN SQL QUERY FORMULATION

by

Toni Taipalus, Mikko Siponen, and Tero Vartiainen 2018

ACM Transactions on Computing Education, 18(3), Article 15

Errors and Complications in SQL Query Formulation

TONI TAIPALUS and MIKKO SIPONEN, University of Jyvaskyla, Finland
TERO VARTIAINEN, University of Vaasa, Finland

SQL is taught in almost all university level database courses, yet SQL has received relatively little attention in educational research. In this study, we present a database management system independent categorization of SQL query errors that students make in an introductory database course. We base the categorization on previous literature, present a class of logical errors which has not been studied in detail and review and complement these findings by analyzing over 33,000 SQL queries submitted by students. Our analysis verifies error findings presented in previous literature and reveals new types of errors, namely logical errors recurring in similar manners among different students. We present a listing of fundamental SQL query concepts we have identified and based our exercises on, a categorization of different errors and complications and an operational model for designing SQL exercises.

CCS Concepts: • **Social and professional topics** → **Computing education**; **Computer science education**; *Model curricula*;

Additional Key Words and Phrases: Human factors, Languages, Standardization, Errors, Exercise Design, Query Languages, SQL

ACM Reference Format:
Toni Taipalus, Mikko Siponen, and Tero Vartiainen. 2010. Errors and Complications in SQL Query Formulation. *ACM Trans. Comput. Educ.* 9, 4, Article 39 (March 2010), 28 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Relational databases still dominate the field of enterprise applications [32], and Structured Query Language (SQL) continues to be the de facto database query language. In addition to SQL's current popularity as a topic in database courses [7], the role of SQL in both courses and practice in the future is strengthened by the recent emergence of NewSQL systems, which use SQL as their query language [8, 41].

Different errors that users make have been studied extensively in programming [24] and to some extent in other computer languages such as HTML and CSS [29]. Such studies contribute to increased understanding of the difficulties that users experience in the process of learning new languages [11]. Even though SQL was standardized by ANSI/ISO as early as 1986/1987 [14] and is used widely today, SQL has received less attention in educational research than programming [1].

SQL still lacks a unified error categorization, and error categorizations presented in previous studies [1, 34, 39] have focused on specific errors relevant only to each individual study or have allowed a specific database management system (DBMS) to perform the categorization. DBMS-specific SQL implementations differ from one another [31] and there exists no error categorization

Authors' addresses: Toni Taipalus, toni.taipalus@jyu.fi; Mikko Siponen, mikko.t.siponen@jyu.fi, University of Jyvaskyla, Faculty of Information Technology, P.O. Box 35 (Agora), FI-40014, Jyvaskyla, Finland; Tero Vartiainen, University of Vaasa, Department of Computer Science, Wolffintie 34, P.O. Box 700, 65101, Vaasa, Finland, tero.vartianen@uva.fi.

1946-6226/2010/3-ART39 $15.00
https://doi.org/0000001.0000001

that is both DBMS-independent and also addresses a wide variety of different errors. Additionally, no study is focused on categorizing student errors in SQL in detail in regard to data demand. As defined by Buitendijk [6], data demand is a task expressed in natural language, e.g., "give me all suppliers in London", to which a student is required to write an equivalent SQL query. In other words, all previous studies focused on errors that are apparent, even if the data demand was not known [e.g., 4] or the focus of those studies was not in error categorization [e.g., 6].

An SQL standard based, unified error categorization will benefit database query teaching and research by providing a framework by which study results are comparable, regardless of the DBMS used or error types studied. Furthermore, a DBMS-independent error categorization is not limited to pre-2010s relational implementations but extends to NewSQL systems as well. In terms of teaching, this means that by teaching SQL in a way that conforms to the SQL standard, we are teaching students the use of a query language that is usable in NewSQL systems [cf. 33] as well. With a categorization explaining what errors occur, we can move toward understanding why those errors occur and how we can adjust our teaching methods accordingly so that the occurrence of these errors can be minimized or avoided.

Our work makes two main contributions. First, we review the previous literature on SQL errors to form a basis for the unified error categorization in a DBMS-independent framework. We analyze student data to review and complement these findings and present a categorization of different errors. Second, we address an error class (namely logical errors, see Section 2.2) that has been identified in a previous study [4] but has not been studied in detail, and we categorize SQL errors within that class based on the analysis of student data.

In the next section, we review relevant studies on SQL errors. In Section 3, we describe the course from which the student data was collected, the exercises and the procedure for error categorization. In Section 4, we present the results of our study and compare them to previous research. In Section 5, we discuss the practical implications of our results and in Section 6 present conclusions and future work. Appendix A contains the database diagram presented to the students, and Appendix B contains the SQL exercises with example answers.

2 BACKGROUND

We explored previous research in both computer languages in general and SQL in particular. We conducted a keyword search in scientific databases, such as ACM, IEEE and Elsevier, with keywords and keyword pairs *sql, query, error, category* and *taxonomy*. The publications we found also provided us with references to related studies in other publications.

We have identified two error classes in previous studies: syntax and semantic errors. Before we review error categories identified in previous studies, we discuss which error classes are the focus of those studies and how these error classes are defined.

2.1 Terminology

The previous studies on SQL errors that we examined sometimes used conflicting definitions for key terms, e.g., a *query* could be interpreted either as a statement in SQL [6] or a statement in natural language [39]. Next, we define key terms for this work.

A *query* is an answer that is usually written by a student in SQL for a particular data demand. A query is submitted to the DBMS, and the DBMS outputs a result table or an error message. Although a query is commonly understood as any Data Manipulation Language (DML) statement, this work focuses on data retrieval queries. For convention and clarity, SQL keywords are written in all capital letters.

A *database object* is any object that exists in a database or can be defined with a Data Definition Language (DDL) statement. Database objects include, but are not limited to, tables, columns, schemas, aggregate and scalar functions, triggers and catalogs.

A *clause* is a part of a query initiated by one of the following keywords: SELECT, FROM, WHERE, GROUP BY, HAVING or ORDER BY, and is followed by one of the previous keywords or a semicolon terminating the whole statement. A *source table* is a table from which a query projects or calculates values into the result table. A *subject table* is a table that is utilized to restrict rows in the result table. From a single query's point of view, a table may be a source and a subject table at the same time. A query may contain zero to many source and subject tables. In a special case such as a self-join, the same table may be a subject table multiple times from a single query's point of view.

A WHERE clause is the main restriction clause of an SQL query, and we make a distinction between restrictions: table join conditions are called *joins* and other restrictions are called *expressions*.

2.2 Error Classes

Error categorization has been studied extensively in programming [e.g., 18, 23, 25], and these languages, regardless of whether they were studied in the 1970s or 2010s, are all imperative by nature, whereas SQL is a fundamentally different language in its declarative nature and purpose.

Errors in SQL query formulation have been studied in the past, and different categorizations for errors have been presented [e.g., 39, 40]. Although these studies are still largely relevant, the SQL standard has undergone several changes and received additional features, including the JOIN predicate, outer joins and new data types. Furthermore, more easily understood teaching methods have been proposed, e.g., teaching relational algebraic division with aggregate functions rather than multiple NOT EXISTS subqueries [26].

We aim to establish an error categorization that is independent of the DBMS and whether or not the DBMS is a traditional relational DBMS or a NewSQL system. Different DBMSs implement the SQL standard differently, and although these differences may be minor, we cannot base our error categorization on one DBMS. If we were to base, e.g., our syntax error categorization on SQL Server, that syntax error categorization would not be ours, and the errors would be categorized by the DBMS. Consequently, categorizing the same data with a different DBMS would result in a different categorization. Because of these differences and the aim of our study, we use the SQL standard as a reference when categorizing the errors.

Previous research [1, 2, 4, 6, 34, 39] categorizes SQL errors under two classes: syntax and semantic errors. The division is intuitive because DBMSs detect syntax but not semantic errors. All the aforementioned research agrees that a query containing a syntax error is not valid SQL, and the DBMS returns an error message. We add that, whether or not a query contains a syntax error can depend on the DBMS, e.g., query rewriters of some DBMSs like MySQL may automatically add type casts to expressions such as *order_number LIKE 100*, while others like PostgreSQL may return a syntax error. Because we consider syntax errors in accordance with the SQL standard, all syntax errors are not necessarily caught (or tolerated) by the DBMS.

In regard to semantic errors, previous research presents definitions for different levels of precision. Buitendijk [6, p. 79] states that a query that contains a semantic error is produced with "an erroneous train of thought". Buitendijk's [6] study contains the word pair *logical errors*, but the SQL errors that are not syntax errors are called semantic errors. However, Buitendijk [6] points out that it is possible for a user to commit a logical error, which results in the query being incorrect for the particular data demand. Smelcer [34] and Ahadi et al. [1] propose that the semantically incorrect query is syntactically correct but returns information not intended by the user. Ahadi et al. [3] state that a query with a semantic error produces either an empty result table or a result table that is not the same as desired.

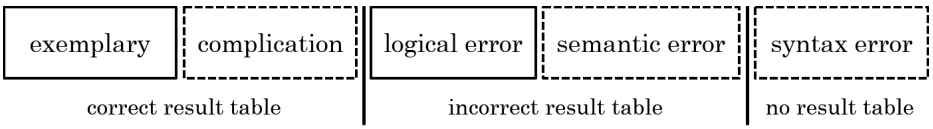


Fig. 1. Error classes and their relationship with the result table and data demand

Brass and Goldberg [4] detail the semantic error definition by stating that the SQL query is legal (i.e., syntactically correct) but does not always produce the intended results for a given data demand (they use the word *task*). Brass and Goldberg [4] make a further distinction between queries that are incorrect regardless of the data demand and queries that are incorrect for a particular data demand. These authors leave the latter class of semantic errors outside of their study and state that it is possible to implement functionality to DBMSs to detect the former class of semantic errors. We consider these former and latter classes of semantic errors fundamentally different by definition and from now on call the latter class *logical errors* and the former class *semantic errors*. To clarify the difference between semantic and logical errors, consider queries Q1 and Q2 in a database of a single table, EMP(empno, fname, sname, job):

Q1 [4]:
```
SELECT *
FROM emp
WHERE job = 'clerk' AND job = 'manager';
```

Q2:
```
SELECT *
FROM emp
WHERE job = 'clerk' OR job = 'manager';
```

Given that the database is in the first normal form, both of the expressions in Q1 can never be true, and, therefore, the WHERE clause could be reduced to *WHERE False*, which will always return an empty result table. It follows that, even without knowing what the data demand is, Q1 contains a semantic error. It could be argued that Q1 is correct if the data demand is to "list all the information about employees whose job is both clerk and manager", but this data demand is not valid because, with this database structure, it is not possible to store more than one job per employee. Q2, however, does not contain a semantic error. Whether or not Q2 contains a logical error depends on the data demand associated with the query. If the data demand for Q2 is to "list all the information about clerks and managers", Q2 is correct. If the data demand is something else, Q2 contains a logical error. Our work essentially builds upon Brass and Goldberg's study [4]; they recognize the existence of logical errors, but they leave logical errors outside of their study. Our study verifies their semantic errors, discovers new semantic errors and studies logical errors in detail, namely, what logical errors occur in query writing, and characteristics and frequencies of those errors.

Brass and Goldberg [4] further propose that queries that are unnecessarily complicated are also considered to contain a semantic error. While Brass and Goldberg focus on compiler warnings and optimization in their study, and we focus on students learning SQL, we make a further distinction between semantic errors and complications. Both semantic errors and complications share the characteristic of being evident by reading the query without knowledge of the data demand, but while semantic errors affect the data in the result table, complications do not (see Section 4.4).

Figure 1 summarizes the definitions we use in this study. The rectangles represent queries with a certain characteristic, the leftmost query being without errors or complications. The text below the rectangles represents the nature of the result table that the query returns. The rectangles with dotted lines represent error classes that can be recognized without knowing the data demand.

3 STUDY

In order to create an unified SQL error categorization, our study had the research goal of categorizing errors and complications in students' SQL retrieval statements in an introductory database course. As we categorize students' errors during their learning processes, our research approach is interpretive in nature. Interpretive studies may be divided into the types of "outside researcher" or "involved researcher" [38], and our study represents the involved type, as the first author of our study carries out the teaching activity and aims to develop database education based on the results of this study. This "close involvement" [38] means that, as the course teacher, the first author is close to the students who produce the SQL sentences and is therefore able to understand the context of the data production, the problems students face and the issues emerging. Although this kind of involvement is time and resource consuming, the first author's involvement makes it possible to create an in-depth understanding of how to develop education with respect to database teaching. The error categorization is based on analysis of over 33,000 SQL queries submitted by 237 students via an e-learning environment developed in the university of the first two authors. The queries were collected during a one-semester course taught by the first author.

Writing the correct SQL query for a course assignment can take several tries, and making an error or several errors does not necessarily mean that a student is unable to complete an assignment [7]. However, our e-learning environment logs all queries submitted to the DBMS, thus offering more fine-grained data than is usually available for analysis. The granularity of logging allows us to move closer to the student in our analysis as proposed by Fincher et al. [13], as opposed to only analyzing the final answers. If we only analyzed the final answers, it is possible that we would miss some errors. For this course, the e-learning environment provides a minimal interface for the students to communicate with the DBMS via a web browser. The user interface is an interactive SQL prompt similar to those provided by most relational DBMSs. The difference in the e-learning environment's SQL prompt is that it is embedded to a web page and that the students are not required to install anything. Even though different SQL learning environments have been studied in the past [e.g., 5], for this work, the e-learning environment simply provided a method of collecting data and was not the subject of the study.

3.1 Course

The course in question mostly follows the ACM/AIS IS 2010 curriculum [37] guidelines for the core course in data and information management. Topics of the course include conceptual modeling, relational model and algebra, SQL, normalization up to fourth normal form and data warehousing. The course is primarily taken by second-year CS and IT majors but also by students from other departments minoring in ICT with no previous experience in SQL.

Query languages are taught in five phases in the course. First, students are introduced to the concept of relationally complete relational algebra, which lays the foundation for different operations that queries utilize to retrieve data from a relational database. In the next four phases, SQL is introduced gradually by the four sublanguages DML, DDL, Data Control Language (DCL) and Transaction Control Language (TCL).

In order for the reader to understand the scope of our study, it is worth discussing how and what aspects of DML, namely data retrieval, are taught in the course. All the DML aspects in the course are taught according to the SQL standard and without a product-specific dialect. Our e-learning environment, however, utilizes the SQLite DBMS to which the student queries are sent. SQLite returns a result table or an error message depending on the query submitted. In addition to the basic concepts like SELECT, FROM and ORDER BY clauses, and basic operators like classic comparison, LIKE, IS and IN, table joins are taught in four different methods: subqueries with IN, subqueries

with EXISTS, explicit join without a subquery and the keyword JOIN and its features like OUTER and NATURAL. The differences in syntax and the logic behind each of the methods are explained. After that, a student is free to choose the methods to his or her liking and use those in practice.

Grouping concepts are also taught in the course to the extent of GROUP BY and HAVING clauses as well as aggregate functions. More advanced SQL concepts like CASE, WINDOW, recursion, non-primitive grouping [22, p. 345] or runtime SQL are not discussed (see Section 3.2 for more details on the query concepts). We expect error categorization to provide the disciplinary knowledge and the fine-grained student data to provide knowledge on how students learn, as proposed by Tenenberg and McCartney [36].

The students were given the opportunity to complete assignments to earn points toward a better grade, and among these assignments were 15 SQL retrieval statements. The students could choose whether to omit or include their queries from the study, and everyone chose to participate. Points were given regardless of participation. The students were given seven days and unlimited tries to complete each of the three sets of assignments (see sets A, B and C in Table 1) wherever and using whatever material available (course material, internet and any means of communication) to more accurately mimic today's work environments. The exercises within a set could be completed in whatever order, and the correct result table was presented for each of the exercises during the whole process so that the students could compare it with the result table produced by their query. Although existing literature on this approach is limited, Ahadi et al. [2] note that, compared to work environments, providing the correct result table constitutes in making the environment unnatural, as nobody knows the correct result table when writing a query. Prior [30], however, utilizes an approach similar to ours and suggests that providing the correct result table facilitates query formulation skills. The students were also presented a database schema diagram representing tables, their columns, data types and primary and foreign keys (see Appendix A). SQLite allowed the students to obtain more detailed information on the database objects, if necessary.

3.2 Exercises

The exercises were completed using a database with 11 tables (see Appendix A). The data were handcrafted, and each table contained 5-125 rows, with an average of 60 rows. Table 1 lists all the fundamental concepts associated with each of the exercises. The data demand and one example answer for each exercise are listed in Appendix B. While concepts like *single-table, multi-table, expressions, nesting, ordering* and *grouping* are self-explanatory and discussed in many course books [e.g., 12, 27], the more ambiguous concepts are explained below.

The concept named *facing foreign keys* describes a situation in which table A has a foreign key constraint linking to table B, and table B linking to table A (see foreign keys between tables *store* and *employee* in Appendix A). The concept named *does not exist* is expressed as a negated existential quantifier ($\neg\exists$) in tuple relational calculus. Syntactically, the concept is simple and is achieved by the logical operator *not* and in SQL with NOT EXISTS or NOT IN, followed by a subquery or with OUTER JOIN.

The concept named *equal subqueries* stands for two or more subqueries that are not nested, but on the same level (compare example answers B6 and B8 in Appendix B). *Aggregate function evaluated against a column value* (Q3) and *a constant* (Q4) are special cases in which the result returned by a aggregate function must be evaluated against a column value or a constant using a comparison operator. The column or constant side of the evaluation is in the upper level query, and the aggregate function is in the subquery. In the scope of our course, evaluation against a constant usually requires a correlated subquery, whereas evaluation against a column value requires an uncorrelated subquery. Note that such comparisons related to grouping restrictions accomplished by the HAVING clause are not included in this concept.

Table 1. Each exercise's fundamental concepts and the number of source and subject tables required to write a correct query. Tables total lists the number of tables in the correct query's FROM -clause or clauses. Concepts marked with an asterisk are also used in Ahadi et al. [2]

#	Fundamental concepts	Source tables	Subject tables	Tables total
A1	single-table*; expressions	1	1	1
A2	single-table*; expressions; ordering	1	1	1
A3	single-table*; wildcard; expressions with nesting	1	1	1
B4	multi-table*; expressions; facing foreign keys	1	1	2
B5	multi-table*; expressions with nesting; ordering	1	3	3
B6	multi-table*; expressions; does not exist	1	2	3
B7	multi-table*; expressions; does not exist	1	2	2
B8	multi-table*; expressions; does not exist; equal subqueries	1	2	3
B9	single-table*; expressions; aggregate functions*	1	1	1
B10	multi-table*; expressions; multiple source tables	2	3	4
B11	multi-table*; expressions; self-join*; aggregate function evaluated against a column value; correlated subquery*	1	2	2
B12	multi-table*; expressions; aggregate function evaluated against a constant; uncorrelated subquery*; parameter distinct	1	1	2
B13	multi-table*; expressions; self-join*;	1	5	5
C14	multi-table*; multiple source tables; aggregate functions*; grouping*	2	1	2
C15	multi-table*; multiple source tables; aggregate functions*; grouping*; grouping restrictions*; ordering	2	1	2

Q3:
```
[...]WHERE price =
  (SELECT MAX(price)
  FROM[...]
```

Q4:
```
[...]WHERE 8 <
  (SELECT COUNT(*)
  FROM[...]
```

Not all of the exercises test a novel fundamental concept but rather the student's ability to combine previously learned concepts in different contexts. For example, in multi-table queries, simple expressions require the necessary understanding of the query language as well as the database structure regarding an expression's placement in the correct clause. Based on previous teaching experience, we believe that a number of these concepts invite the possibility of different logical errors. For example, a data demand that requires expressions with nesting invites the possibility of missing or incorrect nesting, whereas a data demand with grouping restrictions invites the possibility of insufficient grouping, a missing HAVING clause or missing expressions.

3.3 Methodology

From the diversity of interpretive research methods, we selected directed and conventional content analysis [19] to study students' SQL queries. Directed content analysis is used when prior research exists about a phenomenon but the literature is perceived to be incomplete. In directed content analysis, the existing theory is used to direct the analysis of the data, for example. The strength of directed content analysis is that existing literature can be extended. Conventional content analysis

is used to study the phenomenon when research literature or existing theory on the phenomenon is limited. The emergence of categories is data-driven, meaning that preconceived categories are not used [19]. In this study, we first produced a synthesis of errors based on existing literature, and then the first author used directed content analysis to determine error categories in students' SQL queries. Most of the errors were categorized, but there were errors not found in pre-existing studies, especially in the class of logical errors. Therefore, conventional content analysis was then used to study those uncategorized logical errors. Next, we report our analysis step-by-step.

First, we gathered the errors, complications and error categories listed or discussed by Welty [39], Smelcer [34], Brass and Goldberg [4] and Ahadi et al. [1]. We did not discuss runtime SQL in our course and left runtime errors discussed by Brass and Goldberg [4] outside our categorization. Next, when it was possible in terms of the level of detail used in the previous studies, we considered to which of the four classes the error or complication belonged.

Second, we grouped similar errors and complications in previous studies together. This was done because some of the studies discussed similar concepts using slightly different names or descriptions, e.g., "misspellings" by Smelcer [34] and "using AVE instead of AVG" by Welty [39] were grouped together as "misspellings" in our listing.

Third, the first author used the lists of syntax errors, semantic errors and complications produced in the previous step to perform directed content analysis on the student data, as proposed by Shannon and Hsieh [19]. He grouped the SQL queries submitted by students by exercise number and then by student, and then he sorted the queries by the timestamp the query was submitted to the DBMS. We wanted the DBMS to influence the results as little as possible, and the first author analyzed the data without any computerized automation, e.g., a DBMS or scripts. It was possible for a single query to demonstrate several errors and complications. The first author coded queries with appropriate codes corresponding to errors and complications found in previous literature. Any query demonstrating a new error or complication was marked. The queries that were marked were analyzed again to determine the classes of errors the query demonstrated. New syntax errors, semantic errors and complications were given a new code, and if the query did not demonstrate a logical error, the mark was removed.

Fourth, the first author analyzed the queries that were left marked in the previous step, i.e., queries with logical errors, using conventional content analysis as proposed by Shannon and Hsieh [19]. Every time a new logical error was encountered, it was given a code and a brief description. After all the data had been analyzed, the first author considered whether or not the logical errors were sufficiently similar to be grouped together, e.g., he grouped "join with > operator, when equijoin is required" and "join with < operator, when equijoin is required" together as "join with incorrect comparison operator". We decided not to disregard errors just because they were infrequent because, intuitively, different exercises invite different errors, and just because an error is infrequent in our exercises does not necessarily mean that it would be infrequent in some other exercises.

Finally, the first author grouped errors and complications together into categories within each of the four classes. These eighteen categories each represent a distinct family of errors or complications that are similar in nature. These eighteen categories, and whether they are new or previously identified, are discussed in detail in Section 4.

4 RESULTS

All the SQL examples provided in this section contain an error demonstrating the related error category. The queries submitted by students were often complicated but the errors simple; because of this, the example queries in Section 4 are not actual student answers but modified by us for brevity and clarity to represent the error in question as clearly as possible. Unless stated otherwise, the database schema for the example queries is as presented in Appendix A.

We gathered syntax, semantic and logical errors found in previous literature and student data into Table 2, Table 4 and Table 5, respectively. Complications are presented in Table 6. The references in italics represent errors we also encountered in the student data. The errors without a reference are new errors we did not find in previous studies but encountered in the student data. The numbers inside brackets in the Discussed in -column correspond to errors numbered by Brass and Goldberg [4], and the IDs in the ID column are defined by us in the coding phase, as reported in Section 3.3. We have also briefly elaborated on some of the previously identified errors in the tables.

4.1 Syntax Errors

Syntax errors were numerous in the student data. We identified 23 errors in the previous literature, some of which had different levels of overlapping. Overlapping is a result of the level of detail errors are discussed in different previous studies, e.g., Ahadi et al. [1] list different cases where the query refers to nonexistent database objects (error IDs 4-8), whereas Smelcer [34] abstracts these cases to misspellings and synonyms (error IDs 9-10).

4.1.1 SYN-1 Ambiguous Database Object. Database objects, such as tables, schemas and catalogs, each form namespaces on different levels in the database, and the DBMS prevents naming conflicts within a namespace. In multi-table, multi-schema or multi-catalog queries, however, namespaces are merged and, e.g., without additional qualifiers, column names can become ambiguous. In SQL, these qualifiers are called correlation names (i.e., aliases), and omitting them leads to a syntax error if the DBMS cannot resolve the database object to which the query refers.

Cleve et al. [9] researched a schema evolution of a database in which the number of tables grew roughly from 100 to 450. A database of hundreds of tables creates a need for namespaces of different levels, and the queries to such databases need correlation names of corresponding levels. We see no need to divide this subcategory to more fine-grained categories, such as ambiguous columns, tables, schemas or functions, since future standards may introduce new database objects.

4.1.2 SYN-2 Undefined Database Object. Queries with references to nonexistent database objects cause a syntax error for the same reason as references to unambiguous database objects because the DBMS cannot resolve the database object based on the information schema.

This category combines syntax errors for undefined columns, tables and functions and extends the categorization to include all other database objects as well. Smelcer's [34] names "synonyms" and "misspellings" give some insight as to why these types of errors occur, such as the student making a typographical error or remembering the object name incorrectly, e.g., *customers* instead of *customer*. In these two cases, the error is relatively easy to fix, but the error can also be caused by more severe problems, such as the failure to understand the database structure.

4.1.3 SYN-3 Data Type Mismatch. As pointed out in Section 2.2, some DBMSs handle some type conversions automatically while others do not. We argue that the type conversion, i.e., using the correct operator, should always rest on the query writer, especially when teaching SQL in order to simulate a DBMS-independent environment. When appropriate operators are used and when necessary type casts are explicit and in accordance with the SQL standard, the queries are more portable from one DBMS to the other. Next, we elaborate on the common cases of this error based on the student data.

First, we observed using an unfit operator for a column data type. Examples of these errors include cases of using LIKE instead of a classic comparison operator, IS instead of LIKE and IN with something other than a list with atomic values. Second, we observed omitting quotes around character strings (error ID 11) or adding quotes around other data types such as numerical values or Booleans.

Table 2. Syntax errors and error categories

ID	Error	Discussed in
SYN-1 Ambiguous database object		
1	omitting correlation names	*[34, 39]*
2	ambiguous column	*[1]*
3	ambiguous function	[1]
SYN-2 Undefined database object		
4	undefined column	*[1]*
5	undefined function	*[1]*
6	undefined parameter	*[1]*
7	undefined object	*[1]*
8	invalid schema name	*[1]*
9	misspellings	*[34, 39]*
10	synonyms	*[34]*
11	omitting quotes around character data	*[34]*
SYN-3 Data type mismatch		
12	failure to specify column name twice	*[34]*
13	data type mismatch	*[1]*
SYN-4 Illegal aggregate function placement		
14	using aggregate function outside SELECT or HAVING	*[39]*
15	grouping error: aggregate functions cannot be nested	*[1]*
SYN-5 Illegal or insufficient grouping		
16	grouping error: extraneous or omitted grouping column	*[4] (21)*
17	strange HAVING: HAVING without GROUP BY	*[4] (32)*
SYN-6 Common syntax error		
18	confusing function with function parameter	*[39]*
19	using WHERE twice	*[34, 39]*
20	omitting the FROM clause	*[34]*
21	comparison with NULL	*[4] (9)*
22	omitting the semicolon	*[1]*
23	date time field overflow	[1]
24	duplicate clause	
25	using an undefined correlation name	
26	too many columns in subquery	
27	confusing table names with column names	
28	restriction in SELECT clause (e.g., SELECT fee >10)	
29	projection in WHERE clause (e.g., WHERE firstname, surname)	
30	confusing the order of keywords (e.g., FROM customer SELECT fee)	
31	confusing the logic of keywords (e.g. grouping instead of ordering)	
32	confusing the syntax of keywords (e.g., LIKE ('A', 'B'))	
33	omitting commas	
34	curly, square or unmatched brackets	
35	IS where not applicable	
36	nonstandard keywords or standard keywords in wrong context	
37	nonstandard operators: (e.g., &&, || or ==)	
38	additional semicolon	

Third, we observed failure to understand that both sides of logical operators AND and OR must be evaluated as Boolean values, i.e., arguments of WHERE and HAVING clauses must be Boolean type (error ID 13). This was a particularly common syntax error with the LIKE operator, and, in some rarer cases, a classic comparison operator was used to compare a value to a set of values. The former case was demonstrated by Smelcer [34]. Fourth, we observed using a column as a function parameter even though the column data type does not match the parameter's required data type (usually results in error ID 13).

4.1.4 SYN-4 Illegal Aggregate Function Placement. The only two clauses in which aggregate functions COUNT, SUM, AVG, MIN and MAX can be used in the scope of our course are the SELECT and HAVING clauses. Placing an aggregate function in any other clause or using an aggregate function as a parameter to another aggregate function causes a syntax error. This syntax error was common with exercises involving aggregate function evaluation against a column or a constant.

4.1.5 SYN-5 Illegal or Insufficient Grouping. According to the SQL standard, two approaches exist to implement grouping, and we call the non-additional implementation *strict* [22, p. 321-328]. The strict approach determines that a query with at least one aggregate function and at least one grouping column in the query's main SELECT clause must contain a GROUP BY clause that groups the result table according to all grouping columns and only the grouping columns. Execution of such a query should result in a syntax error if grouping is missing altogether, grouping is not applied to all grouping columns or grouping is applied to non-grouping columns.

The additional, less strict approach (optional feature T301) states that all the grouping columns in the SELECT clause must be functionally dependent on the columns listed in the GROUP BY clause [22, p. 341-349]. Although theories and implementations of automated functional dependency discovery to various levels of reliability have been proposed [e.g., 17, 20, 21], the query processors of relational DBMSs are seldom aware of the functional dependencies present in the database. Because of this, we consider the T301 approach problematic when learning SQL. To briefly demonstrate the matter, let us assume Table 3 and queries Q5 and Q6 for the remainder of this subsection only.

Table 3. Customer table

cno	fee
1	0
2	10
3	10

Q5:
```
SELECT cno, MAX(fee)
FROM customer;
```

Q6:
```
SELECT cno, MIN(fee), MAX(fee)
FROM customer;
```

We found this error particularly interesting because of different popular DBMS default settings. Because the grouping is missing, both Q5 and Q6 return a syntax error in PostgreSQL, Oracle Database, SQL Server and DB/2, all of which are among the most popular relational DBMSs. MySQL and SQLite, however, return a result table that contains one row based on the execution plan. It is worth noting that the result table returned by Q6 contains a spurious row, either (2, 0, 10) or (3, 0, 10). Furthermore, the row in the result table changes based on the order of the aggregate functions. The outcome of implementing the T301 approach and the fact that the query processor is not aware of functional dependencies is that grouping can be applied to whatever columns or omitted

completely. We extend this category to the HAVING clause as well because it is closely related to grouping, and the same approaches (strict and T301) apply to it. Our e-learning environment utilized SQLite, which implements the T301 approach for grouping by default, and because we overlooked this possibility when designing the exercise data, approximately 59% of the 158 students who solved exercise B11 used a similar erroneous query and, satisfied with the result table, moved on to the next exercise. Queries Q7 and Q8 explain the name of this error category.

Q7:
```
SELECT cno, AVG(fee)
FROM customer
GROUP BY city;
```

Q8:
```
SELECT cno, city, AVG(fee)
FROM customer
GROUP BY city;
```

Since customer number (cno) is not functionally dependent on city, Q7 displays illegal grouping and Q8 insufficient grouping. To sum up, insufficient or illegal grouping causes no syntax error in some popular DBMSs, but we cannot find any data demand by which having a result table with a spurious row or a row that is seemingly but not truly random could be useful. Based on the arguments above, we consider this a syntax error, and not a semantic or logical error unless functional dependencies can be reliably identified and utilized by the DBMS to prevent behavior not conforming to the SQL standard.

4.1.6 SYN-6 Common Syntax Error. Because syntax errors were numerous and diverse, we categorized all syntax errors not fitting in clear patterns as common syntax errors, similar to Ahadi et al. [1]. Examples of common syntax errors were misspellings of SQL keywords, missing semicolon, brackets or commas, projection in a wrong clause, incorrect clause ordering and missing clauses.

Without the FROM clause, no column values can be projected or calculated since no tables or views are declared. Although Smelcer [34] categorizes FROM clause omission as a semantic error, we consider it a syntax error since the SQL standard requires the FROM clause [22, p. 55-56]. Although some of the popular implementations conform to the SQL standard in this regard, others such as PostgreSQL allow the FROM clause to be omitted, e.g., to perform calculations or calling scalar functions that do not necessarily operate on database data. In such cases, omitting the FROM clause is not a semantic or logical error.

4.2 Semantic Errors

Brass and Goldberg [4] state that their semantic error listing has "a certain degree of completeness" and our findings support theirs; thus, we found few semantic errors not listed in their study. We did not encounter errors regarding UNION in the student data because none of our exercises required the use of UNION. Similarly, errors regarding OUTER JOIN were rare in the student data due to the fact that most of the students chose to solve the exercises dealing with the concept of *does not exist* with NOT IN or NOT EXISTS rather than using OUTER JOIN.

4.2.1 SEM-1 Inconsistent Expression. An inconsistent expression is an expression that causes the result table to be empty or to contain all rows. Smelcer [34] lists four sample cases of AND/OR difficulties, some of which are semantic errors and some clearly syntax errors.

A common example of an inconsistent expression in the student data was when the data demand implied an AND operator even though OR was required (e.g., exercise A1). We also grouped problems with wildcards to this subcategory. Examples of wildcard errors included using an asterisk instead of a percent sign, confusing the functionality of percent and underscore signs, using wildcards with classical comparison operators or with IN and errors with null comparison.

Table 4. Semantic errors and error categories

ID	Error	Discussed in
SEM-1 Inconsistent expression		
39	AND instead of OR (empty result table)	*[34]*
40	implied, tautological or inconsistent expression	*[4] (1, 8)*
41	DISTINCT in SUM or AVG	*[4] (33)*
42	DISTINCT that might remove important duplicates	*[4] (38)*
43	wildcards without LIKE	*[4] (34)*
44	incorrect wildcard: using _ instead of % or using, e.g., *	
45	mixing a >0 with IS NOT NULL or empty string with NULL	
SEM-2 Inconsistent join		
46	NULL in IN/ANY/ALL subquery	*[4] (10)*
47	join on incorrect column (matches impossible)	*[4] (29, 31)*
SEM-3 Missing join		
48	omitting a join	*[4, 34] (27, 28)*
SEM-4 Duplicate rows		
49	many duplicates	*[4] (37)*
SEM-5 Redundant column output		
50	constant column output	*[4] (3)*
51	duplicate column output	*[4] (4)*

4.2.2 SEM-2 Inconsistent Join. An inconsistent join is an error that causes the result table to be empty or, if there are no other restrictions, contain all the rows, and it is certainly not intended by the query writer. Examples of inconsistent joins in the student data were multi-table join conditions using columns with data types that did not match or that matched but had data ranges that never overlapped, resulting in the join condition returning no rows. Self-joins using a wrong correlation name were also observed.

```
Q9:
SELECT cust_id
FROM customer
WHERE cust_id IN
  (SELECT renno
  FROM rental);
```

```
Q10:
SELECT s1.stono, s1.street
FROM store s1, store s2
WHERE s1.city = s1.city
AND s2.stono = 100
AND s1.stono <>100;
```

Q9 demonstrates a join condition using columns that have different data types and no overlapping values. The join condition in Q10 by itself does not restrict any rows because one of the correlation names in the join is incorrectly s1 instead of s2.

4.2.3 SEM-3 Missing Join. Based on the student data, we noticed that table joins using IN or JOIN contained fewer missing joins than joins with EXISTS or explicit join conditions without subqueries. Even though a table join is usually required in multi-table queries, some special cases exist in which omitting the join is desired. This argument is also supported by the SQL standard that specifies an optional feature (F401-04) specifically for joinless (i.e., no column values are compared) multi-table queries; CROSS JOIN [22, p. 1197]. Practical implications for joinless multi-table queries, however, are scarce. We emphasize the meaning of the word *missing* here. A query contains this

semantic error if a join is clearly implied (but omitted) for the result table to contain meaningful data.

4.2.4 SEM-4 Duplicate Rows. Duplicate rows in the result table serve no purpose. We consider a query that by design invites the possibility of duplicate rows to contain a semantic error. Duplicate rows returned by the DBMS cause more data to be transferred between the disk and buffer, between software tiers and possibly in the network. Duplicate rows also make the result table more difficult to read for the end user. The error is remedied by using DISTINCT when needed. It is worth noting that DISTINCT should not be used unnecessarily because it can decrease performance [4].

4.2.5 SEM-5 Redundant Column Output. Redundant column output demonstrates a projection error that causes the result table to contain one or more columns that provide no useful data. Brass and Goldberg [4] demonstrated two semantic errors related to redundant column outputs: constant column output and duplicate column output. Even though it could be argued that SEM-4 and SEM-5 represent a similar concept, i.e., redundant data in the result table, we argue that they demonstrate a different kind of error. SEM-5 represents errors that are possible for the student to identify relatively easily even before running the query. SEM-4, however, can result from the student using different methods for joining tables, e.g., a join with IN is less likely to produce duplicate rows than an explicit join without a subquery.

4.3 Logical Errors

We discovered 30 logical errors, 28 of which were not discussed in previous studies. Contrary to syntax and semantic errors that have been demonstrated in previous studies, we have provided more examples of logical errors listed in Table 5.

4.3.1 LOG-1 Operator Error. For this category, we grouped together errors concerning comparison and logical operators. Operator errors cover errors with missing, extraneous (i.e., not required) or misplaced operators, such as NOT, confusions with classical comparison operators and BETWEEN, and using OR instead of AND. Note that using OR instead of AND can be a logical or semantic error depending on the columns compared and other expressions in the clause. Two particularly interesting and common examples of operator errors (IDs 55 and 56) were related to existence negation in exercises B7 and B8. For brevity, let us consider the data demand "list the surnames of actors who have never acted in a movie released in 2015":

```
Q11:
SELECT a.sname
FROM actor a
WHERE EXISTS
  (SELECT *
  FROM acts s);
  WHERE a.actno = s.actno
  AND EXISTS
    (SELECT *
    FROM movie m
    WHERE s.movno = m.movno
    AND m.year <>2015)
  );
```

```
Q12:
SELECT a.sname
FROM actor a
WHERE EXISTS
  (SELECT *
  FROM acts s);
  WHERE a.actno = s.actno
  AND NOT EXISTS
    (SELECT *
    FROM movie m
    WHERE s.movno = m.movno
    AND m.year = 2015)
  );
```

Both Q11 and Q12 contain a logical error and answer to different data demands: "list the names of actors who have acted in at least one movie not released in 2015" and "list the names of actors

Table 5. Logical errors and error categories

ID	Error	Discussed in
LOG-1 Operator error		
52	OR instead of AND	
53	extraneous NOT operator	
54	missing NOT operator	
55	substituting existence negation with <>	
56	putting NOT in front of incorrect IN/EXISTS	
57	incorrect comparison operator or incorrect value compared	
LOG-2 Join error		
58	join on incorrect table	
59	join when join needs to be omitted	
60	join on incorrect column (matches possible)	
61	join with incorrect comparison operator	
62	missing join	
LOG-3 Nesting error		
63	improper nesting of expressions	*[34]*
64	improper nesting of subqueries	
LOG-4 Expression error		
65	extraneous quotes	*[39]*
66	missing expression	
67	expression on incorrect column	
68	extraneous expression	
69	expression in incorrect clause	
LOG-5 Projection error		
70	extraneous column in SELECT	
71	missing column from SELECT	
72	missing DISTINCT from SELECT	
73	missing AS from SELECT	
74	missing column from ORDER BY clause	
75	incorrect column in ORDER BY clause	
76	exraneous ORDER BY clause	
77	incorrect ordering of rows	
LOG-6 Function error		
78	DISTINCT as function parameter where not applicable	
79	missing DISTINCT from function parameter	
80	incorrect function	
81	incorrect column as function parameter	

who have acted in at least one movie but not in a movie that was released in 2015", respectively. Although both of the examples above are syntactically and semantically correct, they display a lack of understanding of either the functionality of the language, the data demand or both. Moreover, both of these queries will likely return a seemingly correct result table in terms of the number of rows returned. Although we discuss these errors in the lectures prior to the exercises, the error demonstrated in Q11 was common in the student data.

4.3.2 LOG-2 Join Error. The student data showed a common join-related error in one of the exercises. Because the tables *employee* and *store* had facing foreign key constraints between them, the students were required to choose the appropriate columns for the table join. Consider Q13 with respect to the data demand "list the city and phone number of the store in which Jaakko Mattila works":

```
Q13:
SELECT s.city, s.phone
FROM store s, employee e
WHERE s.empno = e.empno
AND e.fname = 'Jaakko'
AND e.sname = 'Mattila';
```

```
Q14:
SELECT COUNT(*) AS total
FROM movie m, acts a
WHERE m.movno = a.movno
AND m.year BETWEEN 1970 AND 2000;
```

Because the join condition is erroneous (error ID 60), the result table shows the cities and phone numbers of the stores of which Jaakko Mattila is responsible for (but does not necessarily work in). Again, the result table can be seemingly correct depending on the number of stores an employee can be responsible for. Although this type of error might be mitigated by renaming the columns with more descriptive names, e.g., *manager_empno*, this exercise served as a demonstration of the importance of choosing the correct columns for a table join and that a NATURAL JOIN, although preferable in terms of enforcing uniform column names and code maintainability, is not always possible.

Q14 demonstrates an extraneous join condition for the data demand "list the number of movies released between the years 1970 and 2000." The example contains a join condition that checks whether or not the movie has any actors, thus possibly limiting the results. In this case, the join must be omitted for the query to be logically correct. A join on incorrect table (error ID 58) means that a join was required by the data demand, and the query contained a syntactically and semantically legal join but on an incorrect table. Missing join (error ID 62) as a logical error differs from missing join listed in semantic errors. Consider the example answer for exercise B8 in Appendix B; if we omit the last subquery, the query is still semantically correct but logically incorrect.

4.3.3 LOG-3 Nesting Error. Difficulties with Boolean logic are a recognized and studied phenomenon in query writing [16], and we categorized these errors as nesting errors, i.e., erroneous use of brackets. This subcategory is also listed by Smelcer [34] under AND/OR difficulties as improper nesting. Smelcer demonstrated improper nesting with an example of improper nesting of expressions. We extend this category with improper nesting of subqueries.

```
Q15:
SELECT fname, sname
FROM actor
WHERE sname LIKE 'F%'
OR sname LIKE 'S%'
AND dob IS NULL
OR dod IS NOT NULL;
```

```
Q16:
SELECT c.fname, c.sname, c.dob
FROM customer c
WHERE NOT EXISTS
  (SELECT *
  FROM rental rt
  WHERE c.cust_id = rt.cust_id
  AND EXISTS
    (SELECT *
    FROM review rv
    WHERE c.cust_id = rv.cust_id)
  );
```

Q15 (see exercise A3 in Appendix B) is similar to the one demonstrated by Smelcer [34]. A query that has both AND and OR operators in a single WHERE clause usually requires nesting the expressions. Errors with subquery nesting were expected in exercises with equal subqueries (B8). For Q16, consider the data demand "list the full names and dates of birth of customers who have never rented a movie but who have given at least one review." Because the subqueries are nested and not equal (i.e., on the same level) in Q16, the query functions like an exclusive OR operator even though that might not be outright evident by reading the query. In other words, the query returns a result table that contains the data of customers who have rented or reviewed at least one movie but have not done both.

4.3.4 LOG-4 Expression Error. We identified three straightforward logical errors related to expressions: missing expression, extraneous expression and expression on incorrect column. We observed missing expressions to be common in all queries and expressions on incorrect columns particularly common in queries involving self-joins. Consider the data demand "list the phone numbers of stores which are in the same city as store #100" for both Q17 and Q18.

Q17:
```
SELECT DISTINCT s1.phone
FROM store s1, store s2
WHERE s1.city = s2.city
AND s2.stono = 100;
```

Q18:
```
SELECT DISTINCT s1.phone
FROM store s1, store s2
WHERE s1.city = s2.city
AND s1.stono = 100
AND s2.stono <>100;
```

Q17 demonstrates a missing expression: the result table also contains the phone number of store #100, which is not desired. Q18 demonstrates an expression on the wrong column: the result table contains only one row representing store #100 if the store is in the same city as some other store. We also identified two more intricate expression errors. The first, extraneous quotes, is demonstrated by Welty [39]. The second error was placing the expression in an incorrect clause, e.g., placing the expression in WHERE clause instead of HAVING clause or vice versa, or placing the expression in an incorrect WHERE clause.

Q19:
```
SELECT sname, SUM(fee)
FROM customer
WHERE fee >= 100
GROUP BY sname;
```

Q20:
```
SELECT sname, SUM(fee)
FROM customer
GROUP BY sname
HAVING SUM(fee) >= 100;
```

Consider Q19 with the data demand "list the sums of fees of customers by surname. Omit surnames with sums below 100 euros" and Q20 with the data demand "list the sums of fees of customers by surname. Calculate only individual fees of 100 euros and above." Both Q19 and Q20 contain a logical error and answer to each other's data demands.

4.3.5 LOG-5 Projection Error. Projection errors are related to column listings in ORDER BY and the main SELECT clauses. We identified four projection errors in the main SELECT clause: missing column, extraneous but non-redundant column, missing DISTINCT and missing AS when it was required to rename a column in the result table.

Regarding ORDER BY clause, we identified four projection errors: fully or partially missing ORDER BY clause, extraneous column in ORDER clause, incorrect column in ORDER BY clause and incorrect ordering of rows. Although an extraneous ORDER BY clause may not affect the result

table, ordering will usually negatively affect performance. Therefore, ordering should not be used if it is not required.

Q21:
```
SELECT stono, city, zip
FROM store
ORDER BY city ASC, zip DESC;
```

Q22:
```
SELECT stono, city
FROM store
ORDER BY stono ASC, city ASC;
```

Q21 demonstrates incorrect ordering when the data demand is to "list the store numbers, cities and zip codes and sort the results in ascending order by city and then in ascending order by zip code." Q22 demonstrates incorrect ordering when the data demand requires that the results are to be sorted using city first and then, within a city, by store number. Q22 achieves the opposite.

4.3.6 LOG-6 Function Error. We identified four aggregate function errors in the student data: incorrect function, incorrect column as function parameter, using DISTINCT as function parameter where not applicable and missing DISTINCT from function parameter.

Q23:
```
SELECT SUM(fee)
FROM customer
WHERE fee >10;
```

Q24:
```
SELECT COUNT(dob)
FROM customer
WHERE fee >10;
```

The data demand for both Q23 and Q24 is to "list the number of customers with a fee over 10 euros", but Q23 instead lists the sum of all customers' fees and demonstrates the use of an incorrect function. Q24 demonstrates the use of an incorrect column as a function parameter. Using date of birth as a function parameter for counting customers with a fee may yield erroneous results since the value of the date of birth can be null.

Our exercise database contained data on movies, actors and actors' participation in different movies. As in real life, the exercise database contained movies in which the same actor acts several roles. Q25 demonstrates a missing DISTINCT when the data demand is to list the names of actors who have acted in at least eight different movies.

Q25:
```
SELECT ar.fname, ar.sname
FROM actor ar
WHERE 7 <
  (SELECT COUNT(ac.movno)
  FROM acts ac
  WHERE ar.actno = ac.actno);
```

Q26:
```
SELECT COUNT(DISTINCT zip)
FROM store;
```

Because the DISTINCT keyword is omitted from the function parameter, the function may count the same movie multiple times. Only one of the exercises required the use of a parameter DISTINCT, and approximately 45% of the 152 students who attempted exercise B12 could not formulate the correct query. Q26 demonstrates using DISTINCT as function parameter where it is not applicable. The data demand for Q26 is to "list the number of stores with a known zip code".

4.4 Complications

Similar to semantic errors, complications are evident without knowledge of the data demand, but unlike semantic errors, complications do not affect the result table. Queries with complications

return the correct result table but could be formulated in a simpler fashion. Such complications can affect the readability of the query and cause performance issues.

Table 6. Complications

ID	Complication	Discussed in
82	unnecessary complication	*[4] (unnumbered)*
83	unnecessary DISTINCT in SELECT clause	*[4] (2)*
84	unnecessary join	*[4] (6)*
85	unused correlation name	*[4] (5)*
86	correlation names are always identical	*[4] (7)*
87	unnecessarily general comparison operator	*[4] (11)*
88	LIKE without wildcards	*[4] (12)*
89	unnecessarily complicated SELECT in EXISTS subquery	*[4] (13)*
90	IN/EXISTS can be replaced by comparison	*[4] (14)*
91	unnecessary aggregate function	*[4] (15)*
92	unnecessary DISTINCT in aggregate function	*[4] (16)*
93	unnecessary argument of COUNT	*[4] (17)*
94	unnecessary GROUP BY in EXISTS subquery	*[4] (18)*
95	GROUP BY with singleton groups	*[4] (19)*
96	GROUP BY with only a single group	*[4] (20)*
97	GROUP BY can be replaced with DISTINCT	*[4] (22)*
98	UNION can be replaced by OR	[4] (23)
99	unnecessary column in ORDER BY clause	*[4] (24)*
100	ORDER BY in subquery	
101	inefficient HAVING	*[4] (25)*
102	inefficient UNION	[4] (26)
103	condition in the subquery can be moved up	*[4] (30)*
104	condition on left table in LEFT OUTER JOIN	[4] (35)
105	OUTER JOIN can be replaced by INNER JOIN	[4] (36)

Most of the complications were already recognized by Brass and Goldberg [4], who proposed four different reasons why a query writer (i.e., in our study, a student) would formulate a query that is unnecessarily complicated. One of the proposed reasons is that the student does not even consider an exemplary query. The other three reasons propose that the student considers an exemplary query but discards it as either erroneous, suboptimal in terms of performance or more difficult to read or maintain compared to his or her unnecessarily complicated query.

Common complications in the student data were unnecessary joins, using DISTINCT when it is not needed, and declaring correlation names that are never used. Another complication also common in the student data was an expression with no impact on the evaluation of the WHERE or the HAVING clause, e.g., *WHERE fee >= 0*, even though the expression is enforced by a CHECK constraint. One complication we did not observe in previous studies was using ORDER BY in a subquery. It is worth noting that, given the nature of complications, we should focus primarily on the errors in teaching, which affect the result table, and secondarily on complications, which affect performance.

Table 7 summarizes the frequencies of all eighteen categories for each of the exercises. As speculated in Section 3.2, certain query concepts appear to invite certain errors. Conversely, some

errors are absent in some exercises. For example, function errors (LOG-6) are not observed before exercise B9, possibly because the data demands in exercises A1-B8 do not imply the use of aggregate functions, or possibly because the students are not yet even aware of the concept of aggregate functions.

Table 7. Error and complication frequencies by category. A number represents the percentage of queries that contained an error or complication belonging to a category, among the queries for a specific exercise.

	A1	A2	A3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	C14	C15
SYN-1	0.0	0.0	0.0	0.9	0.5	0.0	0.6	0.0	0.0	0.9	0.0	0.6	0.0	0.3	4.4
SYN-2	25.9	20.4	1.9	19.5	15.5	4.0	10.1	10.7	8.0	11.1	8.7	4.7	2.0	4.9	6.5
SYN-3	22.3	7.8	4.2	2.6	1.0	30.3	4.9	3.3	2.3	9.8	6.7	3.4	0.5	1.0	1.5
SYN-4	0.0	0.0	0.0	0.0	0.0	0.0	1.2	0.9	0.0	0.0	9.3	1.7	0.0	0.0	2.5
SYN-5	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.4	47.3	15.1	0.0	62.2	67.3
SYN-6	28.9	8.7	30.7	26.0	31.6	17.4	17.5	15.3	23.9	37.8	11.3	20.4	22.9	12.8	26.5
SEM-1	6.6	0.0	19.2	0.4	1.5	3.0	6.5	10.7	19.3	0.4	10.0	4.2	1.5	9.0	12.4
SEM-2	0.0	0.0	0.0	3.9	4.9	3.1	0.6	0.5	0.0	1.8	0.0	6.1	0.0	2.8	0.0
SEM-3	0.0	0.0	0.0	1.7	0.0	3.5	8.1	1.9	0.0	19.6	2.0	8.9	13.9	14.9	11.3
SEM-4	0.6	0.0	0.0	0.8	0.8	0.2	0.4	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
SEM-5	2.4	3.9	3.5	0.0	0.0	0.5	0.0	0.0	0.0	5.8	0.0	0.0	0.0	0.3	0.0
LOG-1	0.0	0.0	3.2	0.4	16.1	0.0	62.2	43.7	2.3	12.4	9.3	17.6	1.1	0.0	0.4
LOG-2	0.0	0.0	0.6	58.0	27.2	7.0	7.1	45.6	0.0	24.4	47.3	23.2	23.9	0.7	6.2
LOG-3	0.0	0.0	47.3	0.0	0.0	2.2	0.0	38.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0
LOG-4	10.2	28.2	32.9	49.8	35.9	35.3	30.2	59.5	18.2	9.3	58.7	21.8	38.3	31.3	40.4
LOG-5	2.4	60.2	5.1	11.3	67.2	62.7	15.6	2.3	36.4	40.4	10.7	6.7	12.9	67.4	60.1
LOG-6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	18.2	0.0	5.3	56.1	0.0	21.5	13.8
COMP	41.6	76.7	0.6	50.6	10.7	64.2	75.3	28.4	4.5	48.4	89.3	49.4	66.2	24.0	9.5

5 DISCUSSION

In this study, we categorized student errors in SQL, a long-lived and popular database query language, and proposed a DBMS-independent categorization of syntax, semantic and logical errors and complications. In this section, we present considerations regarding our results, propose an operational model for designing SQL exercises for a database course and discuss the limitations of our study.

5.1 Regarding the Results

Based on the analyzed data, our findings showed similar syntax errors reported by Smelcer [34] and Ahadi et al. [1]. We encountered similar syntax errors as Ahadi et al. [1, 2], even though they used a different DBMS than us. In terms of syntax errors, we expected and encountered many errors regarding grouping. As criticized by Date [10] over thirty years ago, grouping rules in SQL can cause somewhat anticipated confusion, and this has not changed considering the analyzed data. While some syntax errors were clear misspellings and did not relate to a student's skill, other syntax errors, such as illegal aggregate function placement or illegal grouping, displayed inadequate knowledge about the query language.

Grandel et al. [15] report in their study a comparison of student errors in Java and Python and conclude that the syntax of the language can reduce both syntax and logical errors. Furthermore,

Stefik and Siebert [35] reported that the syntax of a programming language influences both the perceived and actual difficulty of the task. Given this information, the simple syntax of SQL might encourage students to try solving the given exercises. On the other hand, more complex tasks may be perceived simpler than they actually are, resulting in fallacies such as those demonstrated in Section 4.1.5.

Since no result table is returned, we considered syntax errors to be the easiest for a student to spot and fix because of the error message received. We considered other kinds of errors and complications to be more problematic, e.g., complications can cause additional workload on the system, network or both, and may never be fixed as the system, apart from possible performance issues, works as expected. Semantic errors identified in the student data support Brass and Goldberg's [4] listing, where applicable.

It is worth noting that a recent study by Ahadi et al. [3] investigated errors made by students when learning SQL. Ahadi et al. [3] make no distinction between semantic and logical errors but list similar logical errors we encountered in the student data, namely error IDs 62, 66-68, 70-72 and 74-76 in Table 5. In fact, even though their study does not focus on what errors students make, but more on why certain errors occur, errors listed in their study show clear similarities to our findings. These similarities support our findings and show that the errors students make share common patterns, no matter the teacher, teaching methods or the database domain. The error types listed by Ahadi et al. [3] did not affect our categorization because their study was published after our data analysis was completed.

The previously little-studied class of logical errors displayed patterns that we have encountered in our teaching in the past, such as errors with nesting, misplaced NOT operators, missing expressions and confusions with *does not exist*. Our error findings benefit future SQL research by providing an a unified and DBMS-independent categorization of various errors: future studies may use this categorization to identify and analyze errors without the need to establish their own. Our study also adds to the understanding of what difficulties students face when learning SQL, specifically the difficulties with understanding how the language logically operates.

5.2 Designing SQL Exercises

Our categorization of errors provides teachers with a framework of what kind of errors to expect when teaching SQL in an introductory database course. Specifically, considering the results of the study, logical errors are among the key points that cause difficulties with SQL logic among students. In this section, we propose an operational model for designing SQL exercises based on the fundamental concepts used in our exercises, our error categorization framework and positive experiences in the course.

After designing the exercise database structure, we design each of the query exercises around selected fundamental concepts, e.g., similar to those presented in Table 1, and recognize different errors those exercises may invite. When fundamental concepts for each exercise are recognized and acknowledged, we know what errors we as teachers might expect. As we know what errors to expect, we can mitigate the problems students face with purposeful data. Next, we populate the exercise database tables with appropriate amounts of data. We design the exercise data with the expected errors for all exercises in mind. When we are aware of erroneous result tables and the expected errors they are associated with, we can provide students with appropriate feedback. Concerning Figure 2, a query q_1 will have a correct result table RTc and expected logical errors e_1, e_2 and e_3 with their associated, erroneous result tables RTe_n. For example, in exercise B7, the teacher could identify three likely errors. First, there is the potential for confusion with *does not exist*. If a student erroneously formulates *does not exist* similarly to Q11, the result table will contain movies released in 2000-2009, of which there exists at least one copy that is not a BluRay

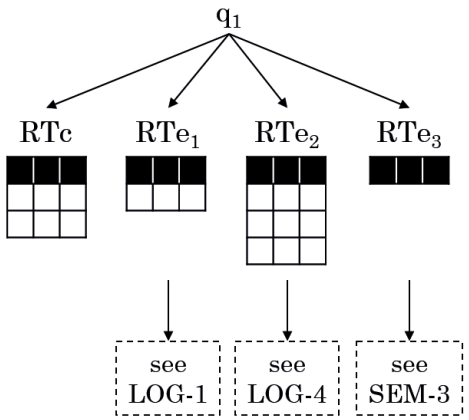


Fig. 2. Query q_1, expected result tables and errors associated with each result table.

(LOG-1). Second, if a student places the year expression into the subquery's WHERE clause instead of the main query's, the result table will contain movies released in 2000-2009, of which there does not exist a copy in BluRay format, and movies which are not released in 2000-2009 (LOG-4). Third, if the student formulates the query as in the example answer for B7 but omits the join, the result table will be empty (SEM-3). Each of these erroneous result tables can be identified, and feedback can be provided accordingly to the student. This feedback can be integrated into e-learning environments by comparing the actual result table with result tables produced by queries with expected errors. Such e-learning environments already exist and could benefit from more accurate feedback concerning errors. For example, SQL-Tutor [28] provided feedback on possible semantic errors in the student's query, and strove to provide more understandable syntax error messages than the DBMS. AsseSQL [30] was used to test the student's query formulation skill, but similar to our e-learning environment, AsseSQL provided no feedback about the correctness of the query, besides presenting the correct result table. Brusilovsky et al. [5] discuss several additional SQL learning environments and tools.

We provide the students with the correct result table for each exercise. If no result table is provided or the correct result table is the same as returned by the queries that contain an expected error, the students might continue to the next exercise even though the query contains an error. Providing the students with the correct result table, however, can be problematic if the expected errors are not recognized, as mentioned in Section 4.1.5.

5.3 Limitations

The first group of limitations of our study is related to the exercises. As stated earlier, the student data provided the means to complement previous error categorization and to create new knowledge on previously unstudied errors, and because the errors encountered are closely related to the exercise design, it may be possible that there exists a key concept that would fit the scope of an introductory-level database course. Even though we taught division operation in the course, it was not included in the exercises. Additionally, even though the fundamental concepts listed in Table 1 were recognized over many years from many different sources and teaching experiences involving diverse target domains, it is possible that some other domains introduce novel, domain-specific fundamental concepts. When analyzing the student data, we made several remarks regarding different errors. In order to identify semantic and logical errors, we needed to inspect more than just the result table and recognize whether or not the query was unnecessarily complicated. Furthermore, a single

query may contain a number of errors, both syntax and semantic [4], in addition to logical errors and complications. Finally, in order to categorize all errors in a query according to our proposed framework, we should not just focus on data demand and the query, but we also need knowledge on the database structure, data types and constraints and the business logic of the target domain.

On the other hand, creating a similar research environment to this study is relatively simple: the database structure and queries are reported in the appendices if the same database domain and queries are warranted. If not, the fundamental concepts and query complexities are listed in Table 1. The environment in which the students complete the exercises is minimally controlled. Students are given the correct result table for each exercise. The error messages from the DBMS will likely have a role in how the student will try to correct a syntactically incorrect query, as discussed by Hristova et al. [18] in the context of programming language compiler errors. Therefore, we suggest using SQLite with default settings in order to more accurately replicate our research environment.

The second group that affects our findings is related to the SQL standard. First, the standard is not an implementation and leaves room for interpretation. Second, the standard is divided into different levels, such as Entry, Intermediate and Full, which serve as instructions on how many features should be implemented. Different products implement the SQL standard for different levels, which may hinder the use of our framework with some DBMSs. Third, the standard contains optional features, some of which overlap with the core features. As demonstrated in Section 4.1.5, some DBMSs have adopted an optional feature over a core feature as their default behavior. All of these points considered, no obvious, single position exists to what a standard-conforming implementation is. As stated by Randolph [31], there is no conformance testing of SQL implementations anymore, and different implementations allow conflicting features. Although the SQL standard should be the reference point for different implementations, the companies behind DBMSs are known to add nonstandard proprietary features to their products' SQL dialects while omitting standard features.

The third group of limitations is related to the research methods and data. For directed content analysis, Shannon and Hsieh [19] discuss that the researchers approach the data with strong bias. Therefore, had we analyzed the data for syntax and semantic errors without examining previous studies beforehand, it is possible that our error categorization might have been different. For conventional content analysis, Shannon and Hsieh [19, p. 1281] argue that it might be challenging for the researchers to develop a "complete understanding of the context, thus failing to identify key categories". Additionally, only the first author coded the errors. However, we have tried to minimize the error margin for logical error categorization by prolonged exposure to the context, as discussed in Section 3.3. Even though our data was collected over one teaching period, the number of queries is relatively high for manual qualitative analysis.

6 CONCLUSIONS AND FUTURE WORK

This study presents a DBMS-independent categorization of SQL errors made by students in an introductory database course. The discovered errors add to the knowledge on what problems students face when learning SQL, specifically problems with the logic of the language.

We suggest applying our categorization framework in practice to proceed from what errors occur to why errors occur, namely, what types of errors have a tendency to stay unresolved, what the reasons are behind these errors and how much the error frequencies affect course performance. When and if we answer these questions in future research, we can begin to identify the students who have problems with SQL queries based on the exercise data, provide additional support for those students and adjust our teaching methods accordingly. Additionally, current research does not discuss the effects of providing versus not providing students with the correct result table.

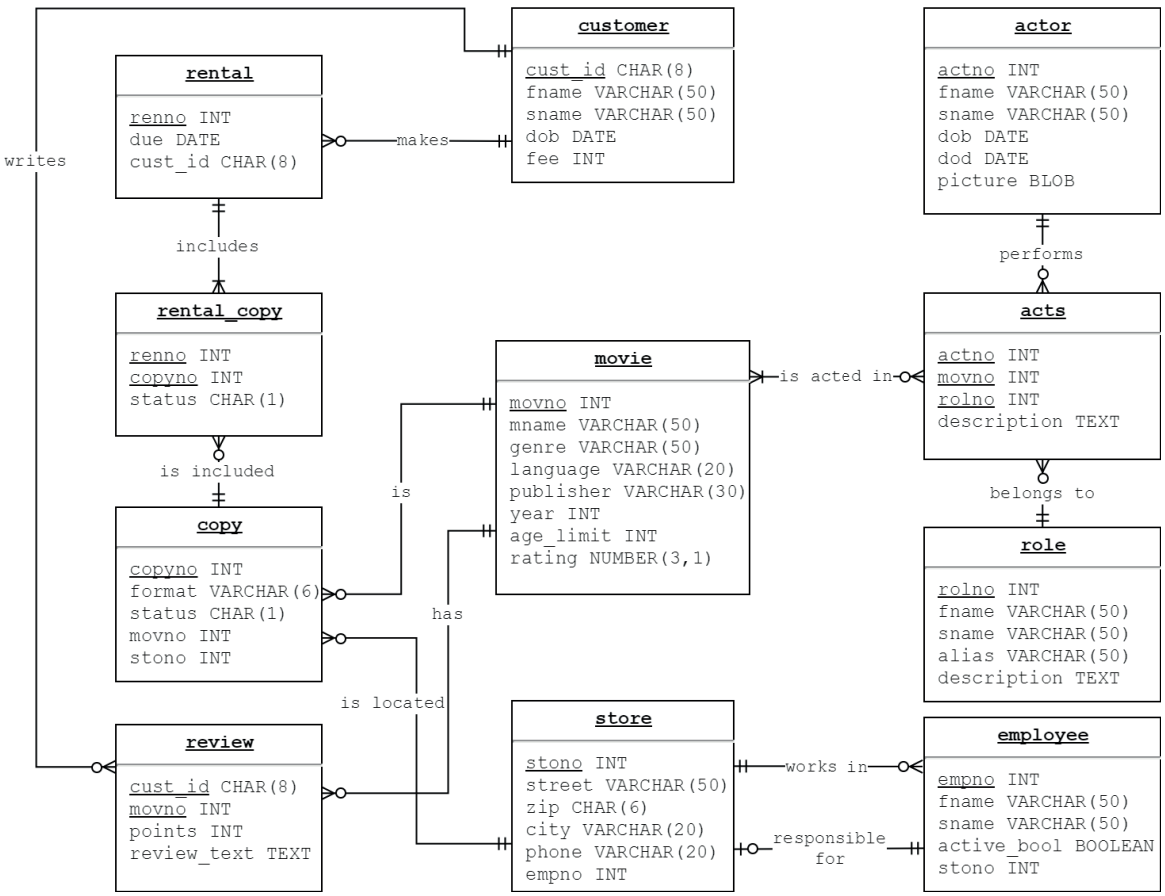
Teaching SQL with a DBMS-independent approach benefits the students in understanding what standard SQL is and what it is not [31]. When identifying different errors, the implementation

used should be secondary, and the SQL standard the primary reference. As more NewSQL systems emerge, it would be interesting to see an experimental research on how vigorously the query languages of those systems conform to the SQL standard.

A DATABASE SCHEMA AND DESCRIPTION

The following database schema diagram and domain description were given to the students. The students could also use the DBMS to acquire more information about the database objects, such as CREATE TABLE statements.

The domain in the SQL exercises is a movie rental service that has stores all over the world. The customers can use a website to search for movies, their favorite actors and the formats (e.g., BluRay) in which the physical copies of the movies can be rented. The customers can write reviews for the movies so that other customers can see whether or not a certain movie is popular. The database also contains information on the company employees that is not visible to customers.



B EXERCISES

The following table lists the 15 exercises used in this study, namely the data demands presented to the students, and example answers formulated by us. The example answers were also provided to the students after the weekly exercise deadlines had passed.

#	Data demand	Example answer
A1	List all information regarding stores in Helsinki and Tampere.	SELECT * FROM store WHERE city IN ('Helsinki', 'Tampere');
A2	List the names, age limits and years of movies that are in English but are not published by Goldeneye BC. Sort the results according to the name of the movie in ascending order.	SELECT mname, age_limit, year FROM movie WHERE language = 'English' AND publisher <>'Goldeneye BC' ORDER BY mname ASC;
A3	List the names, dates of birth and death of actors whose surname starts with an F or an S, and whose date of birth is unknown, or who have a date of death.	SELECT fname, sname, dob, dod FROM actor WHERE (sname LIKE 'F%' OR sname LIKE 'S%') AND (dob IS NULL OR dod IS NOT NULL);
B4	List the city and phone number of the store in which Jaakko Mattila works.	SELECT s.city, s.phone FROM store s, employee e WHERE s.stono = e.stono AND e.fname = 'Jaakko' AND e.sname = 'Mattila';
B5	List the names of actors whose date of death is known and who have acted in at least one movie released after 2010. Sort the results according to surname in descending order.	SELECT a.fname, a.sname FROM actor a INNER JOIN acts ac ON (a.actno = ac.actno) INNER JOIN movie m ON (ac.movno = m.movno) WHERE a.dod IS NOT NULL AND m.year >2010 ORDER BY a.sname DESC;
B6	List the names of actors who have acted a role as himself or herself. Sort the results according to surname, and then according to first name, both in ascending order.	SELECT a.fname, a.sname FROM actor a WHERE EXISTS (SELECT * FROM acts ac WHERE a.actno = ac.actno AND EXISTS (SELECT * FROM role r WHERE ac.rolno = r.rolno AND (r.alias = 'Himself' OR r.alias = 'Herself'))) ORDER BY a.sname ASC, a.fname ASC;
B7	List the movie numbers, names and years of movies that have been released in the first decade of the 2000s, but of which there exists no copy in BluRay format.	SELECT m.movno, m.mname, m.year FROM movie m WHERE m.year BETWEEN 2000 AND 2009 AND NOT EXISTS (SELECT * FROM copy c WHERE m.movno = c.movno AND c.format = 'BluRay');

B8	List the names and dates of birth of customers who have never rented a movie but who have given at least one review.	SELECT c.fname, c.sname, c.dob FROM customer c WHERE NOT EXISTS (SELECT * FROM rental rt WHERE c.cust_id = rt.cust_id) AND EXISTS (SELECT * FROM review rv WHERE c.cust_id = rv.cust_id);
B9	List the number of movies released between the years 1970-2000. Rename the column in the result table descriptively.	SELECT COUNT(*) AS "movies released in 1970-2000" FROM movie WHERE year BETWEEN 1970 AND 2000;
B10	List the names of actors who have acted in the movie Physics 101 and list the names of the roles they have played in that movie. Rename the columns in the result table descriptively.	SELECT a.fname AS "actor's first name", a.sname AS "actor's surname", r.fname AS "character's first name", r.sname AS "character's surname", r.alias AS "character's alias" FROM movie m, actor a, acts ac, role r WHERE m.movno = ac.movno AND ac.rolno = r.rolno AND a.actno = ac.actno AND m.mname = 'Physics 101';
B11	List the name, year and genre of the oldest movie published by Goldeneye BC.	SELECT mname, year, genre FROM movie WHERE publisher = 'Goldeneye BC' AND year = (SELECT MIN(year) FROM movie WHERE publisher = 'Goldeneye BC');
B12	List the actor numbers and full names of actors who have acted in at least five different movies.	SELECT a.actno, a.fname, a.sname FROM actor a WHERE 4 < (SELECT COUNT(DISTINCT ac.movno) FROM acts ac WHERE a.actno = ac.actno);
B13	List the names of customers who have rented exactly the same movie copy that Robert Butler (rbutler1) has rented, whenever.	SELECT c.fname, c.sname FROM customer c, rental r1, rental_copy rc1, rental_copy rc2, rental r2 WHERE c.cust_id = r1.cust_id AND r1.renno = rc1.renno AND rc1.copyno = rc2.copyno AND rc2.renno = r2.renno AND r2.cust_id = 'rbutler1' AND c.cust_id <>'rbutler1';

C14	List the numbers of movie copies located in stores by city and status of the copy. Sort the results by city in ascending order. Make sure that the structure of the result table is as below [example given].	SELECT s.city, c.status, COUNT(c.copyno) AS total FROM store s, copy c WHERE c.stono = s.stono GROUP BY s.city, c.status ORDER BY s.city ASC;
C15	List the numbers of movie copies by movie number and movie name. Disregard movies of which there are less than six copies, regardless of the status of the copy. Sort the results according to the number of the copies in descending order.	SELECT m.movno, m.mname, COUNT(c.movno) AS total FROM movie m, copy c WHERE m.movno = c.movno GROUP BY m.movno, m.mname HAVING COUNT(c.movno) >5 ORDER BY total DESC;

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 '16)*. 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. 2015. A Quantitative Study of the Relative Difficulty for Novices of Writing Seven Different Types of SQL Queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '15)*. ACM Press, New York, New York, USA, 201–206. https://doi.org/10.1145/2729094.2742620

[3] 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 (SIGCSE '16)*. 272–277. https://doi.org/10.1145/2899415.2899464

[4] Stefan Brass and Christian Goldberg. 2005. Semantic errors in SQL queries: A quite complete list. *J. Syst. Softw.* 79, 5 (2005), 630–644. https://doi.org/10.1016/j.jss.2005.06.028

[5] Peter Brusilovsky, Sergey Sosnovsky, Michael V. Uydelson, Danielle H. Lee, Vladimir Zadorozhny, and Xin Zhou. 2010. Learning SQL Programming with Interactive Tools: From Integration to Personalization. *ACM Trans. Comput. Educ.* 9, 4 (2010), 367–376. https://doi.org/10.1145.1656255.1656257

[6] R. B. Buitendijk. 1988. Logical errors in database SQL retrieval queries. *Comput. Sci. Econ. Manag.* 1, 2 (1988), 79–96. https://doi.org/10.1007/BF00427157

[7] Gretchen Irwin Casterella and Leo Vijayasarathy. 2013. An Experimental Investigation of Complexity in Database Query Formulation Tasks. *J. Inf. Syst. Educ.* 24, 3 (2013), 211–221. http://jise.org/Volume24/24-3/pdf/Vol24-3pg211.pdf

[8] Ugur Cetintemel, Nesime Tatbul, Kristin Tufte, Hao Wang, Stanley Zdonik, Jiang Du, Tim Kraska, Samuel Madden, David Maier, John Meehan, Andrew Pavlo, Michael Stonebraker, and Erik Sutherland. 2014. S-Store: a streaming NewSQL system for big velocity applications. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1633–1636. https://doi.org/10.14778/2733004.2733048

[9] Anthony Cleve, Maxime Gobert, Loup Meurice, Jerome Maes, and Jens Weber. 2015. Understanding database schema evolution: A case study. *Sci. Comput. Program.* 97, P1 (2015), 113–121. https://doi.org/10.1016/j.scico.2013.11.025

[10] Christopher J. Date. 1983. Critique of the SQL database language. *SIGMOD Rec.* 14, 3 (Nov 1983). https://doi.org/10.1145/984549.984551

[11] Alireza Ebrahimi. 1994. Novice programmer errors: language constructs and plan composition. *Int. J. Hum. Comput. Stud.* 41 (1994), 457–480. https://doi.org/10.1006/ijhc.1994.1069

[12] Ramez Elmasri and Shamkant B. Navathe. 2016. *Fundamentals of Database Systems (7th. ed.)*. Pearson.

[13] Sally Fincher, Josh Tenenberg, and Anthony Robins. 2011. Research Design : Necessary Bricolage. *Comput. Sci. Educ.* (2011), 27–32. https://doi.org/10.1145/2016911.2016919

[14] Michael M. Gorman. 2002. Is SQL A Real Standard Anymore. *The Data Administration Newsletter* (2002), 2–4.

[15] Linda Grandell, Mia Peltomäki, Ralph Johan Back, and Tapio Salakoski. 2006. Why complicate things? Introducing programming in high school using Python. *Conf. Res. Pract. Inf. Technol. Ser.* 52 (2006), 71–80.

[16] Sharon L. Greene, Susan J. Devlin, Philip E. Cannata, and Louis M. Gomez. 1990. No IFs, ANDs, or ORs: A study of database querying. *Int. J. Man. Mach. Stud.* 32, 3 (Mar 1990), 303–326. https://doi.org/10.1016/S0020-7373(08)80005-3

[17] Eric Gregoire, Richard Ostrowski, Bertrand Mazure, and Lahkdar Sais. 2005. Automatic extraction of functional dependencies. *Theory Appl. Satisf. Test.* 3542 (2005), 122–132.

[18] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. 2003. Identifying and correcting Java programming errors for introductory computer science students. *ACM SIGCSE Bulletin* 35, 1 (Jan 2003), 153. https://doi.org/10.1145/792548.611956

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

[20] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. Tane: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput. J.* 42, 2 (Feb 1999), 100–111. https://doi.org/10.1093/comjnl/42.2.100

[21] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. 2004. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. In *Proceedings of the 2004 ACM International Conference on Management of Data (SIGMOD '04)*. ACM Press, New York, New York, USA, 647. https://doi.org/10.1145/1007568.1007641

[22] ISO/IEC. 2003. ISO/IEC 9075-2:2003, "SQL - Part 2: Foundation". (2003).

[23] Eranki L.N. Kiran and Kannan M. Moudgalya. 2015. Evaluation of Programming Competency Using Student Error Patterns. In *2015 International Conference on Learning and Teaching in Computing and Engineering*. IEEE, 34–41. https://doi.org/10.1109/LaTiCE.2015.16

[24] A.J. Ko and B.A. Myers. 2003. Development and evaluation of a model of programming errors. In *Proceedings of the IEEE Symposium on Human Centric Computing Languages and Environments*. IEEE, 7–14. https://doi.org/10.1109/HCC.2003.1260196

[25] Charles R. Litecky and Gordon B. Davis. 1976. A study of errors, error-proneness, and error diagnosis in Cobol. *Commun. ACM* 19, 1 (Jan 1976), 33–38. https://doi.org/10.1145/359970.359991

[26] Victor M. Matos and Rebecca Grasser. 2002. Teaching Tip A Simpler (and Better) SQL Approach to Relational Division. *J. Inf. Syst. Educ.* 13, 2 (2002), 85–88. http://jise.org/Volume13/Pdf/085.pdf

[27] Jim Melton. 2002. *SQL:1999: Understanding Relational Language Components*. Morgan Kaufman.

[28] Antonija Mitrovic. 1998. Learning SQL with a computerized tutor. *ACM SIGCSE Bulletin* 30, 1 (1998), 307–311. https://doi.org/10.1145/274790.274318

[29] Thomas H. Park, Brian Dorn, and Andrea Forte. 2015. An Analysis of HTML and CSS Syntax Errors in a Web Development Course. *ACM Trans. Comput. Educ.* 15, 1 (2015), 4:1–4:21. https://doi.org/10.1145/2700514

[30] Julia Prior. 2003. Online Assessment of SQL Query Formulation Skills. *Proceedings of the Fifth Australasian Computing Education Conference* 20 (2003), 247–256. http://dl.acm.org/citation.cfm?id=858403.858433

[31] Gary B Randolph. 2003. The Forest and the Trees: Using Oracle and SQL Server Together to Teach ANSI-Standard SQL. *Design* (2003), 234–236.

[32] Julian Rith, Philipp S. Lehmayr, and Klaus Meyer-Wegener. 2014. Speaking in tongues: SQL access to NoSQL systems. *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14)*, 855–857. https://doi.org/10.1145/2554850.2555099

[33] Yasin N Silva, Isadora Almeida, and Michell Queiroz. 2016. SQL: From Traditional Databases to Big Data. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*. ACM Press, New York, New York, USA, 413–418. https://doi.org/10.1145/2839509.2844560

[34] John B Smelcer. 1995. User errors in database query composition. *Int. J. Hum. Comput. Stud.* 42, 4 (Apr 1995), 353–381. https://doi.org/10.1006/ijhc.1995.1017

[35] Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. *ACM Trans. Comput. Educ.* 13, 4 (2013), 1–40. https://doi.org/10.1145/2534973

[36] Josh Tenenberg and Robert McCartney. 2010. Why Discipline Matters in Computing Education Scholarship. *ACM Trans. Comput. Educ.* 9, 4 (2010), 1–7. https://doi.org/10.1145/1656255.1656256

[37] 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.

[38] Geoff Walsham. 2006. Doing interpretive research. *Eur. J. Inf. Syst.* 15, 3 (2006), 320–330. https://doi.org/10.1057/palgrave.ejis.3000589

[39] Charles Welty. 1985. Correcting user errors in SQL. *Int. J. Man. Mach. Stud.* 22, 4 (1985), 463–477. https://doi.org/10.1016/S0020-7373(85)80051-1

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

[41] Li-Yan Yuan, Lengdong Wu, Jia-Huai You, and Yan Chi. [n. d.]. A Demonstration of Rubato DB: A Highly Scalable NewSQL Database System for OLTP and Big Data Applications. *Proceedings of the 2015 ACM International Conference on Management of Data (SIGMOD '15)* ([n. d.]), 907–912. https://doi.org/10.1145/2723372.2735380

Received September 2016; revised October 2017; accepted December 2017

PIII

WHAT TO EXPECT AND WHAT TO FOCUS ON IN SQL QUERY TEACHING

by

Toni Taipalus and Piia Perälä 2019

Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)., ACM, New York, 198-203

What to Expect and What to Focus on in SQL Query Teaching

Toni Taipalus
University of Jyvaskyla
Jyvaskyla, Finland
toni.taipalus@jyu.fi

Piia Perälä
University of Jyvaskyla
Jyvaskyla, Finland
piia.m.h.perala@jyu.fi

ABSTRACT

In the process of learning a new computer language, writing erroneous statements is part of the learning experience. However, some errors persist throughout the query writing process and are never corrected. Structured Query Language (SQL) consists of a number of different concepts such as expressions, joins, grouping and ordering, all of which by nature invite different possible errors in the query writing process. Furthermore, some of these errors are relatively easy for a student to fix when compared to others. Using a data set from three student cohorts with the total of 744 students, we set out to explore which types of errors are persistent, i.e., more likely to be left uncorrected by the students. Additionally, based on the results, we contemplate which types of errors different query concepts seem to invite. The results show that syntax and semantic errors are less likely to persist than logical errors and complications. We expect that the results will help us understand which kind of errors students struggle with, and e.g., help teachers generate or choose more appropriate data for students to use when learning SQL.

CCS CONCEPTS

• **Social and professional topics → Computing education**; *Computational thinking*; *Student assessment*;

KEYWORDS

SQL, error, query language, database education, relational database

ACM Reference Format:
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 '19), February 27-March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3287324.3287359

1 INTRODUCTION

SQL has been taught at university level courses for decades, yet compared to programming languages, SQL has received relatively little attention in educational research. A number of new teaching methods have been presented to facilitate learning [5, 7–9], but scientific evidence on which parts of SQL students struggle with leaves room for interpretation.

This study is an attempt to address the issue of difficult, i.e., persistent errors in SQL. We consider an error persistent if it is present in a student's final answer to an exercise, i.e., the student did not fix the error during the query writing process. The aim of our study is to find out which errors are persistent, and then consider which query concepts invite which persistent errors. To that end, we analyzed the final answers of three student cohorts with the total of 744 students and over 8,700 SQL queries.

In Section 2, we discuss previous studies on SQL error categorization, and briefly explain the frameworks we chose for this study. In Section 3, we describe how we collected and analyzed the data, and in Section 4 we report our findings. In Section 5 we compare our findings to previous studies, and consider the practical implications and limitations of our research, as well as further research opportunities. Finally, in Section 6, we present conclusions.

2 BACKGROUND

2.1 Related Work

Previous SQL research in educational contexts mainly focuses on one of two perspectives. First, on the development and analysis of a particular tool for facilitating SQL learning, and second, on the study of student errors in SQL. The former class of SQL research is out of the scope of our study, but the latter class presents a number of error categorizations, which are needed to quantifiably measure and analyze student errors.

Brass and Goldberg [4] presented an extensive list of semantic errors to be used in database management system (DBMS) compilers, and a set of studies [1–3], which inspired us to this research, explored the frequencies of syntax and semantic errors students made when learning SQL. Ahadi et al. [1] used PostgreSQL to categorize syntax errors from over 160,000 SQL queries. Ahadi et al. [2] studied student errors in exercises with seven different query concepts, and, as the authors point out, theirs is the first published quantitative study of the relative student difficulties in regard to different SQL query concepts. Additionally, Ahadi et al. [3] closely investigated 551 queries that contained a semantic error which students were unable to correct.

Taipalus et al. [11] composed an unified error categorization and a framework of query concepts using earlier research [1, 2, 4, 10, 12]. These findings were validated and complemented by an analysis of over 33,000 SQL queries, and the categorization was based on the SQL standard, and is DBMS independent. The categorization mapped 105 errors and complications into four classes; 1) complications, which do not affect the result table; 2) logical errors, which affect the result table, and for which there exists a valid data demand (i.e., a natural language representation of what the query

SIGCSE '19, February 27-March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/02…$15.00
https://doi.org/10.1145/3287324.3287359

needs to return from the database); 3) semantic errors, which affect the result table, and for which there exists no valid data demand; and 4) syntax errors, which result in an error message instead of a result table.

2.2 Errors and Complications

We use the error categorization, and the query concept framework composed by Taipalus et al. [11], because both of them are wider than any of those presented in the previous studies. According to the categorization, there are three classes of errors in addition to complications. The 38 syntax errors are categorized into six categories; ambiguous database object (SYN-1), undefined database object (SYN-2), data type mismatch (SYN-3), illegal aggregate function placement (SYN-4), illegal or insufficient grouping (SYN-5), and common syntax error (SYN-6).

The 13 semantic errors are categorized into five categories; inconsistent expression (SEM-1), inconsistent join (SEM-2), missing join (SEM-3), duplicate rows (SEM-4), and redundant column output (SEM-5).

The 30 logical errors are categorized into six categories; operator error (LOG-1), join error (LOG-2), nesting error (LOG-3), expression error (LOG-4), projection error (LOG-5), and function error (LOG-6). The categorization points out that, while a semantic error is evident just by reading the query without knowing the data demand, a logical error can be identified only if the data demand is known.

Finally, in addition to errors, the framework contains 24 complications, which are, e.g., unnecessary joins and other structural problems which could be formulated in a simpler fashion. For brevity, we refer to all of these four classes simply as *errors*, when it is not necessary to differentiate errors from complications.

2.3 Query Concepts

As the base for our exercises, we use the the framework [11] which lists 18 query concepts, all of which are present at least once in the 15 exercises. Notably, all of the exercises test skill with more than one query concept, and, what's more, some query concepts invite others by design. All the query concepts per exercise are presented in Table 1.

While some of the concepts are basic, e.g., single-table queries, expressions, aggregate functions, and ordering, others, e.g., multiple source tables (i.e., data in the result table is projected or calculated from more than one source table), and parameter distinct (i.e., DISTINCT is required as an aggregate function parameter) are relatively difficult for an introductory database course. See to Taipalus et al. [11] for more detailed information about the query concepts, exercises, and error categories, as well as example data demands and queries for each of the 15 exercises.

3 METHOD

For this study, we had over 123,000 SQL queries which we collected from three student cohorts in a mandatory database course. The course consisted of lectures, voluntary exercises, exercise discussion sessions, and an exam. The students majored in computer science or information systems with no prior knowledge on using SQL. Each cohort answered to 15 SQL retrieval exercises. We designed the exercises using the query concept framework presented by Taipalus et al. [11], with the same query concepts, as well as the number of source and subject tables. In order to mitigate the polarization effect of the database business domain on the number of errors, we designed different database structures and business domains for each cohort.

Table 1: Query Concepts per Exercise [11]

Exercise	Concepts
A1	single-table; expressions
A2	single-table; expressions; ordering
A3	single-table; wildcard; expressions with nesting
B4	multi-table; expressions; facing foreign keys
B5	multi-table; expressions with nesting; ordering
B6	multi-table; expressions; does not exist
B7	multi-table; expressions; does not exist
B8	multi-table; expressions; does not exist; equal subqueries
B9	single-table; expressions; aggregate functions
B10	multi-table; expressions; multiple source tables
B11	multi-table; expressions; self-join; aggregate function evaluated against a column value; correlated subquery
B12	multi-table; expressions; aggregate function evaluated against a constant; uncorrelated subquery; parameter distinct
B13	multi-table; expressions; self-join
C14	multi-table; multiple source tables; aggregate functions; grouping
C15	multi-table; multiple source tables; aggregate functions; grouping; grouping restrictions; ordering

We also replicated the learning environment as presented by Taipalus et al. [11]. The students answered the 15 exercises during the course. The exercises were divided into three sets (A, B, and C in Table 1). For each set, the students had approximately one week to complete the exercises in the given set, in whatever order, and with unlimited tries. The learning environment was minimally controlled, and the students could use whatever material or forms of communication at their disposal, which more accurately mimics their future work environments, as argued for by Taipalus et al. [11]. The exercise discussion sessions were held after the weekly deadline had passed, and the sessions had no impact on the previous week's queries. The learning environment was effectively an interactive SQL prompt (SQLite) embedded into a web page. The correct result table was visible during the whole query writing process, which, in turn, constitutes as making the environment unnatural when compared to work environments.

In order to pinpoint persistent errors, we analyzed only final answers from each student, which left us with over 8,700 queries for further analysis. Out of these final queries, 2,765 were incorrect. We manually marked these errors according to the error categorization by Taipalus et al. [11]. Based on this, we approached error persistence from two perspectives; relative frequencies for each error category per exercise, and estimated means for each error class per exercise.

Table 2: Relative Error Frequencies by Error Category in Incorrect Final Answers, and Complications in All Final Answers

	A1	A2	A3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	C14	C15
Correct	0.67	0.77	0.60	0.79	0.77	0.78	0.76	0.82	0.91	0.63	0.41	0.43	0.62	0.75	0.56
SYN-1 ambiguous database object	0.00	0.00	0.04	0.02	0.09	0.01	0.03	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00
SYN-2 undefined database object	0.00	0.05	0.00	0.05	0.01	0.03	0.01	0.00	0.00	0.07	0.03	0.03	0.05	0.02	0.02
SYN-3 data type mismatch	0.07	0.00	0.32	0.09	0.23	0.11	0.03	0.03	0.00	0.02	0.01	0.01	0.09	0.06	0.00
SYN-4 ill. aggr. function placement	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.01	0.00	0.02
SYN-5 ill. or insufficient grouping	0.03	0.00	0.00	0.01	0.00	0.04	0.00	0.03	0.02	0.06	0.38	0.43	0.06	0.37	0.67
SYN-6 common syntax error	0.11	0.27	0.09	0.25	0.15	0.09	0.16	0.22	0.05	0.14	0.04	0.09	0.10	0.11	0.06
SEM-1 inconsistent expression	0.08	0.00	0.04	0.00	0.01	0.01	0.01	0.14	0.05	0.02	0.01	0.13	0.15	0.04	0.03
SEM-2 inconsistent join	0.00	0.00	0.00	0.04	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.01	0.02	0.01	0.00
SEM-3 missing join	0.00	0.00	0.00	0.01	0.05	0.06	0.05	0.04	0.00	0.34	0.09	0.03	0.08	0.13	0.03
SEM-4 duplicate rows	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.20	0.05	0.00	0.00	0.00	0.00
SEM-5 redundant column output	0.00	0.01	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
LOG-1 operator error	0.00	0.06	0.12	0.03	0.44	0.11	0.65	0.47	0.22	0.05	0.04	0.06	0.07	0.00	0.02
LOG-2 join error	0.00	0.00	0.00	0.34	0.15	0.23	0.19	0.35	0.00	0.40	0.07	0.20	0.39	0.06	0.02
LOG-3 nesting error	0.00	0.00	0.47	0.00	0.01	0.01	0.00	0.09	0.00	0.00	0.00	0.00	0.08	0.00	0.00
LOG-4 expression error	0.13	0.27	0.11	0.47	0.09	0.14	0.20	0.13	0.09	0.25	0.47	0.36	0.63	0.06	0.14
LOG-5 projection error	0.75	0.46	0.11	0.10	0.25	0.62	0.07	0.05	0.20	0.45	0.14	0.08	0.12	0.59	0.30
LOG-6 function error	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.75	0.00	0.01	0.60	0.00	0.25	0.25
Complication	0.05	0.07	0.14	0.37	0.08	0.33	0.37	0.12	0.06	0.46	0.41	0.21	0.40	0.31	0.17

We calculated the relative frequencies of different errors among incorrect final answers (N=2,765) in the 15 exercises, as we wanted to identify which query concepts invite which classes of errors. An answer was considered incorrect if it contained at least one syntax, semantic or logical error. Importantly, a complication by itself did not constitute in making an answer incorrect. In addition to errors, we calculated relative frequencies of complications among all final answers (N=8,773).

In order to identify persistent errors among exercises, we analyzed the data by counting the number of different classes of errors for each exercise. The homogeneity of variance was tested before analysis, and it was found that the data was overdispersed. Therefore, we modeled the data with negative binomial regression that is also common method for count data. Because our interest was in comparing the error rates under the different exercises, we added fixed effects of task to the model. The non-independence of the observations due to the fact that each student completed multiple exercises was modeled by including random effect of student in the model. We estimated the model using the SPSS (version 24) Mixed command, and interpreted the results by calculating the predicted number of errors and their confidence intervals (CI) for each exercise. We estimated the statistical significance using an overlap rule of 95% for the CI bars, as proposed by Cumming et al. [6].

4 RESULTS

4.1 Error Persistence for Error Categories

We collected success rates, relative error frequencies by error category in incorrect final answers, and complications in all final answers to Table 2. We ignored non-attempts, as not all students attempted to solve all exercises. Some errors were absent in some exercises altogether. Again, we observed high numbers of relative frequencies among logical errors when compared to other categories. In fact, for each of the logical error categories, at least one exercise yielded a relative frequency of at least 0.40. By contrast, e.g., illegal aggregate function placement (SYN-4), inconsistent join (SEM-2), and redundant column output (SEM-5) error categories had a relatively low highest relative frequency of at most 0.02. Complications were present in all exercises.

4.2 Error Persistence for Error Classes

We collected estimated means and CIs of syntax, semantic, and logical errors as well as complications in final answers per exercise to Table 3. Collectively, logical errors were more prominent than other error classes in all exercises. The results show that on average, students are more likely to commit persistent syntax errors in exercise C15 (estimate 0.828 with 95% CI [0.721, 0.951), whereas persistent semantic errors were most prominent in exercise B10 (estimate 0.855 with 95% CI [0.735, 0.995]).

Investigation concerning the class of logical errors showed that students committed fewer persistent logical errors in exercises A1, A2, A3, B11 and C15 when compared to exercises B6, B7, B9, B10, B12, and B13. The overall trend showed that, with a few exceptions, the means of logical errors increase over time, but in the final exercises C14 and C15, the means decreased.

Inspection of complications showed that students wrote more complications in exercises B10, B13, and C14 when compared to exercises A1, A3 and B5. Additionally, exercise B9 showed the widest CIs among all error classes.

Table 3: Estimated Means and CIs for Each Errors Class per Exercise

Exercise	Syntax errors			Semantic errors			Logical errors			Complications		
	Mean	95% CI		Mean	95% CI		Mean	95% CI		Mean	95% CI	
		LL	UL		LL	UL		LL	UL		LL	UL
A1	0.556	0.464	0.666	0.402	0.325	0.498	0.939	0.817	1.079	0.422	0.343	0.519
A2	0.533	0.426	0.666	0.370	0.283	0.485	0.845	0.708	1.009	0.491	0.390	0.620
A3	0.578	0.491	0.682	0.385	0.314	0.471	0.885	0.775	1.011	0.431	0.356	0.521
B4	0.566	0.455	0.704	0.392	0.301	0.510	1.098	0.939	1.285	0.539	0.431	0.675
B5	0.637	0.521	0.777	0.390	0.302	0.505	1.026	0.877	1.202	0.396	0.307	0.510
B6	0.555	0.444	0.693	0.395	0.303	0.515	1.266	1.092	1.467	0.532	0.424	0.667
B7	0.522	0.419	0.651	0.442	0.348	0.563	1.213	1.049	1.402	0.547	0.441	0.679
B8	0.505	0.386	0.660	0.602	0.470	0.770	1.171	0.982	1.397	0.555	0.430	0.716
B9	0.396	0.260	0.602	0.389	0.254	0.595	1.282	1.015	1.619	0.382	0.249	0.585
B10	0.639	0.537	0.759	0.855	0.735	0.995	1.325	1.175	1.494	0.652	0.550	0.774
B11	0.585	0.514	0.667	0.372	0.315	0.438	0.819	0.734	0.915	0.528	0.460	0.606
B12	0.687	0.598	0.791	0.467	0.394	0.554	1.409	1.278	1.553	0.540	0.461	0.632
B13	0.571	0.464	0.703	0.526	0.423	0.655	1.420	1.244	1.621	0.625	0.512	0.763
C14	0.692	0.569	0.841	0.440	0.344	0.562	1.062	0.907	1.243	0.630	0.514	0.773
C15	0.828	0.721	0.951	0.425	0.351	0.516	0.949	0.834	1.080	0.482	0.402	0.578

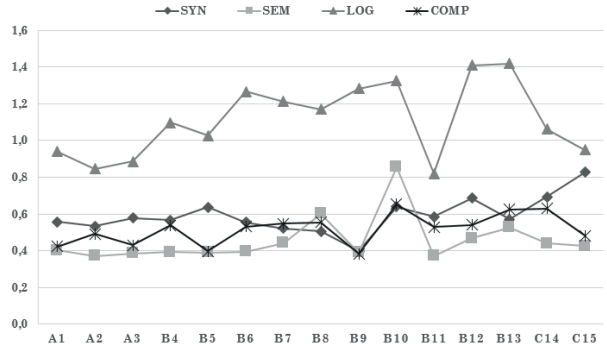


Figure 1: Estimated Means for Each Error Class

5 DISCUSSION

5.1 Persistent Errors in Previous Studies

Although Ahadi et al. [2] used slightly different query concepts, and did not quantitatively study which types of errors occur, we can compare the success rates of our results to theirs to gain understanding on which concepts students struggle with. Their study listed seven query concepts, five of which map to the framework [11] we used. *Simple, one table* [2] corresponds to *single-table* (exercises A1, A2 and A3). Their study reports success rate of 90%, while our rates vary from 60% to 77%. The success rate of a query with *group by* [2], which corresponds to *grouping* (exercise C14), is 75%, which is the same as in our results.

Their query for *group by with having* [2], which corresponds to our *grouping restrictions* (exercise C15), has a success rate of 61%, while ours is 56%. *Self-join* [2] corresponds to *self-join* (exercises B11 and B13), but the success rate of their study (24%) differs from ours (41% and 62%). Finally, *correlated subquery* [2] corresponds to *correlated subquery* (exercise B11), and the success rates are fairly similar with their 46% and our 41%. It is worth noting that most of our exercises tested more than one concept instead of just one. What differs between these two studies, is that the most difficult query concept in their results was self-join, but in our results the most difficult concepts were correlated and uncorrelated subqueries, with success rates of 41% and 43%, respectively.

An earlier study on syntax errors in the whole query writing process [1] (as opposed to our study, in which we only analyzed final answers) used PostgreSQL to categorize syntax errors, which makes the results uncomparable to ours. However, some similarities can be found, as the study points out *syntax error* (corresponds to SYN-6 common syntax error), *undefined column* and *undefined table* (SYN-2), and *grouping error* (SYN-5 illegal or insufficient grouping) among the most frequent syntax errors. Additionally, by examining the final incorrect answers, they discovered that 51% of students abandoned the exercise when they were not able to fix a syntactic error. Ahadi et al. [3] explored persistent semantic errors related to query concepts. The study lists several common errors, most of which correspond to the error categorization we used; *missing/wrong condition* (corresponds to LOG-4, also possibly SEM-1), *self-join not used* (SEM-3 missing join, also possibly SEM-2 inconsistent join), *missing group by or having clause* (SYN-5 illegal or insufficient grouping, and LOG-4 expression error), *use of wrong column* (SYN-5 illegal or insufficient grouping, and LOG-4 expression error), *missing order by clause* (LOG-5 projection error), *incorrect/incomplete column* (LOG-5 projection error), and *missing/extra column in select* (LOG-5 projection error). When compared to our findings in Table 2, these error categories are among the most prominent ones, and our findings seem to support theirs. Additionally, based on our results, data type mismatch (SYN-3), common syntax (SYN-6), operator (LOG-1), join (LOG-2), nesting (LOG-3) and function errors (LOG-6) were relatively frequent in the exercises.

In summary, the available evidence seems to suggest that students struggle with similar query concepts, and some errors are more persistent than others. The consensus view among our results and the results by Ahadi et al. [3] seems to be that the most persistent errors are illegal or insufficient grouping (SYN-5), common syntax error (SYN-6), inconsistent expression (SEM-1), inconsistent join (SEM-2), missing join (SEM-3), expression error (LOG-4), and projection error (LOG-5). Additionally, the results by Ahadi et al. [1] and Taipalus et al. [11] seem to agree that, among all queries, the most frequent syntax errors are common syntax errors (SYN-6), undefined database object errors (SYN-2), and, in queries involving aggregate functions, illegal or insufficient grouping errors (SYN-5).

5.2 Persistent Errors versus All Errors

Taipalus et al. [11] counted relative error frequencies in not just final answers, but during the whole query writing process, i.e., in all queries submitted to the DBMS. By comparing their results to those presented in Table 2, we can acquire some insight on which errors are common, but usually corrected by the students. First, rough comparison reveals that ambiguous database object (SYN-1) and illegal aggregate function placement errors (SYN-4) were uncommon in all and in final answers, while undefined database object errors (SYN-2) were common in all answers, but uncommon in final answers, i.e., usually corrected. The occurrence of data type mismatch (SYN-3) and illegal or insufficient grouping (SYN-5) appears to be closely related to the query concepts, as these errors were frequent in all and in final answers, but only in certain exercises. Common syntax errors (SYN-6) were common in all and in final answers.

The most frequent semantic errors among all and final answers were inconsistent expressions (SEM-1), and missing joins (SEM-3). In general, semantic errors were less frequent in both all and final answers than syntax errors, and the least frequent among all four error classes.

In general, logical errors were most frequent in all and in final answers. Interestingly, in 9 of the 15 exercises, operator errors (LOG-1) were more frequent in the final answers than in all answers. This suggests that operator errors are both common, and they have a high tendency to stay uncorrected. Join errors (LOG-2) were common in almost all multi-table queries, both all and final. Nesting errors (LOG-3) were common in queries which required nesting expressions (A3), or designing the subqueries by using previously uncommon nesting (B8). These errors, however, were closely related to the query concepts, and most of the exercises invited no such errors. Expression (LOG-4) and projection (LOG-5) errors were most common in all and in final answers. These types of errors seem relatively common in the query writing process, and are relatively difficult for students to fix. Smelcer [10] studied SQL query writing process in regard to short term memory, and the occurrence of these types of errors might be related to the thought process of a student translating the natural language data demand into SQL. Intuitively, function errors (LOG-6) occurred only in exercises involving aggregate functions (B9, B11, B12, C14, C15). With the exception of exercise B11, function errors were more common in final than in all answers. This suggests that function errors are difficult for students to fix.

Finally, while complications were relatively common in both all and in final answers, the number of final answers with complications was usually considerably lower than the number of all answers with complications. This suggests that, even though complications are still common, many of them are corrected during the query writing process.

In summary, different query concepts invite different errors by design, e.g., a query with aggregate functions invites the possibility of function errors. By examining the maximum error frequencies for each error category in all answers, we can determine the commonness of each error category, and, consequently, by examining the maximum error frequencies for each error category in all answers, we can determine the persistence of each error category. These things considered, it seems reasonable to assume that we as teachers and researchers should focus on errors which are both common and persistent. Our data indicates the same results as Taipalus et al. [11], i.e., such error categories with a maximum (and arbitrary) relative frequency of at least 0.40 are logical (LOG-1 through LOG-6), and illegal or insufficient grouping errors (SYN-5), as well as complications.

5.3 Persistent Errors and Query Concepts

Among syntax errors, the most persistent were data type mismatch (SYN-3), illegal or insufficient grouping (SYN-5), and common syntax errors (SYN-6). Common syntax errors were relatively frequent in all exercises. With the exception of exercise B13, data type mismatch errors seemed to decrease while the students completed more and more exercises, which might suggest that students learned to formulate queries in which the expressions were Boolean type. Illegal or insufficient grouping errors were common, but only in exercises involving aggregate functions (B11, B12, C14, C15), with the exception of exercise B9, which might be explained by the fact that B9 is the only single-table query with aggregate functions, and thus easier to solve. In terms of persistent syntax errors, answers to exercise C15 showed the most syntax errors. This might be explained by the query requiring the use of all six SQL clauses, as the second highest mean among syntax errors is in exercise C14, which, in turn, required the use of five SQL clauses, while all other exercises required the use of three to four clauses.

As seen in Fig. 1, the trend was that semantic errors were the least persistent among the four error classes. Among semantic errors, the most persistent were inconsistent expressions (SEM-1), inconsistent joins (SEM-2), and missing joins (SEM-3). Inconsistent expressions were relatively prominent compared to other semantic errors in almost all exercises. This is rather unsurprising, because expressions are required in almost all exercises (A1 through B13). Also, intuitively, missing and inconsistent joins were relatively common in all multi-table exercises (B4 through B8, and B10 through C15). Persistent semantic errors were most prominent in exercise B10. One explanation might be that this exercise is the first in which the result table must contain data from multiple tables, which in turn invites joins without subqueries. This might be the first time a student needs to use this approach, but it might not be evident that this approach is preferred.

Finally, and most importantly, logical errors were most persistent among the four error classes. Expression (LOG-4) and projection

Table 4: Which Persistent Errors to Expect

Concept	Expect
multi-table	SEM-2 inconsistent join, SEM-3 missing join, LOG-2 join error
equal subqueries	LOG-2 join error
self-join	LOG-2 join error
multiple source tables	SEM-3 missing join
aggregate functions	SYN-5 illegal or insufficient grouping, LOG-6 function error
(all)	SYN-6 common syntax error, LOG-4 expression error, LOG-5 projection error, complication

errors (LOG-5) were common across all exercises, and operator (LOG-1) and join errors (LOG-2) across almost all exercises. Among logical errors, nesting errors (LOG-3) were relatively uncommon, and the persistence of function errors (LOG-6) was closely related to query concepts involving the use of aggregate functions. Based on our results, we collected the different errors that some of the concepts invite to Table 4. For most of the concepts, however, the data showed no distinguishable patterns.

In conclusion, our results provide needed insight on which SQL errors students struggle with, and which aspects we as teachers and researchers should focus on when utilizing SQL. Although Taipalus et al. [11] proposed an operational model for designing SQL exercises and exercise database data, taking into account their framework's 105 different errors for each exercise is arguably an onerous task. We propose that teachers should start the exercise design by focusing on the most expected persistent errors first, and then work down to less common errors depending on their personal time and resource constraints. What's more, our study propounds the view that students are able to correct some types of errors by themselves, and therefore teaching should focus on errors which students struggle with.

5.4 Limitations and Further Research

We compared our results with certain previous studies [1–3], which did not use the same query concepts or error categorization. With this in mind, there is a chance of misinterpretation of the listed concepts and error categories. Furthermore, these concepts and categories might include or exclude aspects of the framework [11] we used. Finally, the framework lists multiple query concepts per exercise, while Ahadi et al. [2, 3] only list one. In some cases, this might be the result of different levels of specificity between the concept listings, but nonetheless makes the cause and effect analysis in our results more difficult, as it not clear whether frequent errors in any one exercise are caused by a particular query concept, or by a combination of two specific query concepts.

For further research, we propose a tool for automatically categorizing errors in student answers according to the error categorization by Taipalus et al. [11]. The categorization is extensive, and complications, semantic errors, and syntax errors must be analyzed by hand. This is further emphasized by the fact that the categorization is DBMS independent, which makes reliable syntax error discovery by a single DBMS unreliable. By automation, the categorization is open to larger datasets, and by making the automation of error categorization real-time, learning environments may provide students feedback as the query is being written.

6 CONCLUSION

In this study, we set out to investigate which errors are persistent, i.e., more difficult for students to fix, and which types of persistent errors different query concepts, such as joins or aggregate functions invite. The results show that logical errors and complications are more persistent than syntax or semantic errors. While function errors were common in exercises with certain query concepts, expression and projection errors were persistent in all exercises, regardless of the query concepts. We propose that while designing SQL exercises, teachers and researchers design the exercise data to take account most persistent errors, starting from the most common ones.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Mikko Rönkkö for his invaluable advice and support regarding the methodology, and the anonymous reviewers for their comments and suggestions.

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 '16)*. 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. 2015. A Quantitative Study of the Relative Difficulty for Novices of Writing Seven Different Types of SQL Queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '15)*. ACM Press, New York, New York, USA, 201–206. https://doi.org/10.1145/2729094.2742620

[3] 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 (SIGCSE '16)*. 272–277. https://doi.org/10.1145/2899415.2899464

[4] Stefan Brass and Christian Goldberg. 2005. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software* 79, 5 (2005), 630–644. https://doi.org/10.1016/j.jss.2005.06.028

[5] Luca Cagliero, Luigi De Russis, Laura Farinetti, and Teodoro Montanaro. 2018. Improving the Effectiveness of SQL Learning Practice: A Data-Driven Approach. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 01. 980–989. https://doi.org/10.1109/COMPSAC.2018.00174

[6] Geoff Cumming, Fiona Fidler, and David L. Vaux. 2007. Error bars in experimental biology. *The Journal of Cell Biology* 177, 1 (2007), 7–11. https://doi.org/10.1083/jcb.200611141 arXiv:http://jcb.rupress.org/content/177/1/7.full.pdf

[7] Mohammad Dadashzadeh. 2003. A Simpler Approach to Set Comparison Queries in SQL. *Journal of Information Systems Education* 14, 4 (2003), 345–348.

[8] Victor M. Matos and Rebecca Grasser. 2002. Teaching Tip A Simpler (and Better) SQL Approach to Relational Division. *Journal of Information Systems Education* 13, 2 (2002), 85–88. http://jise.org/Volume13/Pdf/085.pdf

[9] Gang Qian. 2018. Teaching SQL: A Divide-and-conquer Method for Writing Queries. *Journal of Computing Sciences in Colleges* 33, 4 (April 2018), 37–44. http://dl.acm.org/citation.cfm?id=3199572.3199577

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

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

[12] Charles 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

PIV

THE EFFECTS OF DATABASE COMPLEXITY ON SQL QUERY FORMULATION

by

Toni Taipalus 2020

Journal of Systems and Software, 165, Article 110576

The Effects of Database Complexity on SQL Query Formulation

Toni Taipalus

Faculty of Information Technology, University of Jyvaskyla, Jyvaskyla, Finland

Abstract

In Structured Query Language (SQL) education, students often execute queries against a simple exercise database. Recently, databases that are more realistic have been utilized to the effect that students find exercises more interesting and useful, as these databases more accurately mimic databases students are likely to encounter in their future work environments. However, using even the most engaging database can be counterproductive to learning, if a student is not able to formulate correct queries due to the complexity of the database schema. Scientific evidence on the effects of database complexity on student's query formulation is limited, and with queries from 744 students against three databases of varying logical complexity, we set out to study how database complexity affects the success rates in query formulation. The success rates against a simple database were significantly higher than against a semi-complex and a complex database, which indicates that it is easier for students to write SQL queries against simpler databases. This suggests, at least in the scale of our exercise databases, that educators should also consider the negative effects of more realistic databases, even though they have been shown to increase student engagement.

Keywords: Structured Query Language (SQL), database, database complexity, education, student learning

1. Introduction

Computer languages have been a major topic in ICT education curricula for decades. Even though most of these languages change over time, Structured Query Language (SQL) has proved especially resilient. Given the importance, long life, and pervasive nature of databases and query languages in the field of information technology, it is rather surprising that educational research on the topic is relatively scarce when compared to, e.g., programming languages. Furthermore, studies related to skills of professionals working with databases have pointed out the difficulties arising from the differences of database management system implementations of the SQL standard (McMinn et al., 2019), faults in database schema integrity constraint definition and enforcement (McMinn et al., 2015), and ill-designed database transactions (Warszawski and Bailis, 2017), all of which further emphasize the importance of effective SQL education.

In addition to teaching theoretical foundations, many university level database courses facilitate SQL learning by providing the students an environment in which they can execute SQL queries against an exercise database (e.g., Mitrovic, 1998). Similarly to programming education, teaching SQL in practice is justified, as many students are expected to perform similar tasks in their future work environments. The used exercise databases are usually constructed by the teacher, or provided by a third party. One of such third party database is the Sakila[1] database of MySQL database management system (Sakila, 2019), which contains both structure and data for a movie rental business domain. Traditionally, these exercise databases have been relatively simple, possibly to shift the focus of the

Email address: toni.taipalus@jyu.fi (Toni Taipalus)

[1] https://dev.mysql.com/doc/sakila/en/

learning process from the structure of the database to the logic and semantics of SQL (Wagner et al., 2003). Recently, though, more realistic databases such as Sakila, and some of the databases of Teradata University Network (Jukic and Gray, 2008a; Watson and Hoffer, 2003) have been utilized, and research shows that students find more realistic databases more interesting and useful (Yue, 2013). In effect, educational research has provided support for the assertion that more complex databases have positive effects on database education. However, little research touches the potential negative effects of the structural complexity of a database on SQL learning, e.g., a student's failure to formulate SQL queries. This inability is a likely indication that a student has not acquired the necessary practical knowledge to write valid SQL, which is arguably one of the goals of SQL education. This study provides the field with a perspective on the potential negative side effects (i.e., lower success rates in query formulation) of increasing exercise database complexity.

Given the consideration that, however interesting a more realistic exercise database might seem to a student, utilizing such a database may be counterproductive to learning, if the student cannot formulate correct SQL queries due to the structural complexity. This problem of database complexity potentially manifests in either as a failure to start the query formulation process due to perceived overwhelming complexity, or as a failure to successfully formulate the query despite one or several attempts. Although writing erroneous queries is part of any student's learning process, a student is able to correct some errors, but not necessarily all. An error left uncorrected is a common indication of some problem in knowledge, skill, or learning. In the vein of Taipalus and Perälä (2019), we call errors which are never corrected *persistent*. In this study, we set out to analyze differences in query writing performance (i.e., success rates) of three student cohorts with a total of 744 students. One cohort wrote SQL queries against a simple, one against a semi-complex, and one against a complex database. While the complexity of a database schema is both subjective and relative, we measured the complexity of a database schema according to previously established metrics (Calero et al., 2001), which effectively measure complexity by both the number of certain database objects, and the number of potential predictable joins (cf. Section 2.2 for a more detailed description, and the Appendices for the database schemas).

Our results show statistically significant differences in success rates between the student cohorts. Based on our results, we recommend that researchers and educators also consider the negative implications of more complex exercise databases, rather than using the more complex databases available.

The rest of the study is structured as follows. In Section 2 we discuss prior SQL education research, database complexity metrics, and the frameworks used in this study. In Section 3, we present our hypotheses. In Section 4, we describe the course and exercises from which the data were collected, the exercise databases, and our research method. In Section 5 we present our results, and in Section 6 discuss practical implications and limitations of our study, and future research avenues. Finally, in Section 7 we present conclusions.

2. Theoretical background

2.1. Database complexity in education

A number of studies discuss database complexity and SQL learning (Jukic and Gray, 2008b; Wagner et al., 2003; Yue, 2013), but it is worth noting that none of these studies explore the effects of database complexity on query writing performance, but on student interest (Yue, 2013), or how to better prepare students for their future work (Jukic and Gray, 2008b; Wagner et al., 2003). Additionally, the effects of task complexity (Topi et al., 2005) and data model representation (Chan et al., 1997, 2005; Rho and March, 1997) on query formulation have been studied. However, given that there exists no scientific evidence regarding the effects of logical complexity of a relational database on SQL query learning, we address here studies that consider database complexity in education in general.

Intuitively, it may seem obvious that it is easier to write SQL queries against a simple rather than a complex database. We traced the argument for more complex exercise databases to 2003, when Wagner et al. (2003) concluded that "[...] using large scientific datasets in a database systems course

has a number of benefits for students, and no discernible losses." The authors claim that increased complexity better prepares students for their future employment, students learn that real-world data have problems, and students learn interdisciplinary work and communications skills. Even though the authors present little numerical evidence to support their argument, we can certainly agree with the part concerning the benefits for students. Similar argument for the benefits for students has also been presented later by Jukic and Gray (2008b). However, the latter part of the quotation concerning no discernible losses seems somewhat contradictory, as the same article reports students perceiving increased complexity more difficult. It is worth noting that Wagner et al. (2003) focus their discussion on the complexity of data (i.e., extension), not the complexity of the database structure (i.e., intension), and in this study, by complexity of a *database* we refer to the complexity of the logical structure, rather than complexity of data.

A more recent study by Yue (2013) argued for more complex exercise databases from the point of view of student interest. The study measured the perceived interestingness and usefulness when Sakila-based assignments were gradually integrated into a database course. The students perceived Sakila more interesting and useful when compared to instructor and textbook assignments. The study also commended Sakila for having the right balance of complexity, meaning that the database is structurally complex, but does not contain unnecessary domain intricacies. This is a noteworthy observation, and in line with Wagner et al. (2003).

For studying how database complexity influences query writing performance, we identified three crucial aspects. First, a set of metrics is needed to measure database complexity, second, a unified set of SQL exercises for the cohorts despite the fact that the three databases are different, and third, a framework to measure whether or not a student's query is correct or incorrect. Next, we discuss these three aspects in prior studies, and argue for the choices we made concerning this study.

2.2. Database complexity metrics

Although normal forms can be considered a method of determining the complexity of a relational database, a higher normal form does not implicitly result in a simpler or more complex database structure. Even though a higher normal form implies more tables, and thus a more complex database, a lower normal form presents different complexities for the query writer. Regarding relational database structure complexity metrics, we found two scientific proposals which complement normalization. First, a four part metric was proposed by Calero et al. (2001) which consists of the number of attributes in the schema (NA), depth referential tree (DRT), number of foreign keys in the schema (NFK), and cohesion of the schema (COS). When a database is presented as a graph G of tables (nodes) and foreign keys (directed edges), DRT is the number of edges on the longest path (not counting loops), and COS is the sum of the square of the number of nodes in each component of G. Second, Pavlic et al. (2008) proposed a database complexity measuring method which decrees that the complexity of a database is the sum of the number of all attributes, keys (i.e., primary and secondary keys), indices, and foreign keys in the database. We wanted to limit this study to the logical complexity of a database, and chose to use the former metrics (Calero et al., 2001), as indices are not a part of the relational model but a part of physical database design.

We would like to add that while any of the metrics proposed by Calero et al. (2001) is insufficient to measure complexity by itself, together they consolidate into an adequate, high-level presentation of the logical complexity of a relational database. Two issues with database complexity metrics, Calero et al. (2001) included, is that there is no objective way to measure whether one database is more complex than the other, if one of the attributes (e.g., DRT) is higher, but another (e.g., COS) is lower. In contrast, database complexities measured by the metrics proposed by Pavlic et al. (2008) can be objectively compared by numbers, but these numbers do not represent objective complexity, as these numbers may be the same for different databases (compare a database with 9 attributes, 3 primary keys, and 5 foreign keys to a database of 12 attributes, 3 primary keys, and 2 foreign keys). The second issue is that these database complexity metrics measure only quantitative aspects of databases, but not the complexity of the business domain. Arguably, a genome database may be

more complex for a layperson than a movie rental database, even if these two databases are of equal complexity by both of the discussed metrics.

2.3. Exercises and query evaluation

A number of studies exploring SQL exercises have been published (Ahadi et al., 2016a,b; Prior and Lister, 2004; Smelcer, 1995; Taipalus et al., 2018), and many of these studies utilized exercises designed for each particular study. The query concepts in the SQL exercises reported in these five studies show similarities, e.g., exercises testing joins, expressions, ordering, and grouping with their respective clauses and predicates. In addition to reporting query concepts by name, Taipalus et al. (2018) provide example SQL queries of the exercises, and the number of tables needed for the formulation of each query. For its relative specificity, we designed our exercises for each database using the query concept framework presented by Taipalus et al. (2018). Although a query can be interpreted as any SQL statement, the scope of our chosen framework limits our study solely on data retrieval. This limitation would have also been the case, had we based our study on any other of the aforementioned studies' query concepts.

In order to measure success rates in student queries, we needed a framework to determine whether or not a student's query was correct or incorrect. Some studies discuss SQL error categorizations (Ahadi et al., 2016b,a), which are, however, not the results of the studies, but rather a vehicle for answering their respective research questions. Other studies, however, present SQL error categorizations as results of their respective studies. Brass and Goldberg (2006) present an extensive list of semantic errors and complications based on their teaching experience, and Taipalus et al. (2018) complement Brass and Goldberg's listing with syntax and logical errors, which are rooted in the SQL standard (ISO/IEC, 2016) rather than a single database management system's implementation. Taipalus et al. (2018) categorize 105 different errors into four error classes: 1) complications, which do not affect the result table, but hinder queries with readability or performance issues, 2) logical errors, which affect the result table, and make the query appear as if it was written to answer a different but valid data demand (i.e., natural language representation of the task), 3) semantic errors, which affect the result table, and make the query unsuitable for any valid data demand, and 4) syntax errors, which result in an error message instead of a result table. For this study, we chose this error categorization framework for its relative extensiveness, and for database management system independence. This error categorization framework (Taipalus et al., 2018) also fits our chosen database complexity metrics (Calero et al., 2001), as both of them disregard physical structure in its entirety.

3. Hypotheses

The discussion in the previous section has both highlighted the positive outcomes for using more complex exercise databases in teaching SQL, as well as the undercurrent of the possible negative effect of difficulty. We propose that as database complexity increases, so does the difficulty of successfully writing SQL queries that satisfy given data demands. The basic proposition to be tested is

> H_1: The success rates for formulating correct SQL queries decrease as logical complexity of the database increases.

The error categorization framework (Taipalus et al., 2018) divides errors into four classes (syntax errors, semantic errors, logical errors, and complications), and an incorrect query may, in theory, exhibit as many as 105 different errors. A recent study (Taipalus and Perälä, 2019), in turn, indicated that logical errors and complications are more likely to persist than syntax errors and semantic errors, meaning that although syntax and semantic errors are committed, they are more likely corrected by students. Given the framework to measure these four error classes, and rather than only studying whether there exists an effect between success rates and database complexity, we wanted to explore if different database complexities invite different kinds of errors. Therefore, as auxiliary hypotheses, we propose that the number of persistent errors committed for each of these four error classes increase as database complexity increases.

H_2: The number of syntax errors committed in incorrect final SQL queries increase as logical complexity of the database increases.

H_3: The number of semantic errors committed in incorrect final SQL queries increase as logical complexity of the database increases.

H_4: The number of logical errors committed in incorrect final SQL queries increase as logical complexity of the database increases.

H_5: The number of complications committed in final SQL queries increase as logical complexity of the database increases.

Concerning the auxiliary hypotheses, it is worth noting that lower success rates do not necessarily imply higher numbers of persistent errors. Even if success rates increase, it is possible that students who are unable to write correct queries commit more persistent errors.

4. Research setting

4.1. Course and data collection

We collected the queries from an introductory database course targeted for second year students majoring in computer science or information systems, with no prior knowledge on SQL. The course was mandatory, but completing the exercises was not, and by completing the exercises the students could earn points toward a better grade. We collected the queries from three student cohorts (237, 280, and 227 students), and each cohort completed SQL exercises against a simple, semi-complex, or complex database (cf. Appendices), respectively. This study took place over a period of three years, and the first author taught the course for each student cohort. The course was given in Finnish.

We constructed exercises for each of the cohorts using the query concept framework presented by Taipalus et al. (2018). The framework contains 15 exercises, all of which test a student's skill in several query concepts. These query concepts per exercise are presented in Table 1. This framework allowed us to construct similar exercises for each cohort in terms of query concepts tested, and the number of tables required to formulate the correct query. According to the framework, a source table is a table which is used to project or calculate values into the result table, and a subject table is a table which is used to restrict the values that are accepted into the result table. For the complex database, we utilized the database structure and exercises presented by Taipalus et al. (2018), and, from there constructed the simple and semi-complex databases with respective business domains and exercises. Refer to Taipalus et al. (2018) and Appendix D for detailed descriptions and examples of the query concepts. It is worth noting that the data demands and database schemas presented in the Appendices are translations from Finnish to English, and the translations introduce some natural language considerations about the similarity of the data demands between the student cohorts.

Students completed the exercises using an interactive database management system (SQLite) prompt embedded on a web page, which, depending on the query submitted by the student, output either a result table or an error message from SQLite. The correct result table was visible during the whole query writing process, and the students could compare their result tables with the correct one. The exercises were completed over three weeks during the course, in three sets (cf. A, B, and C in Table 1), each with their weekly deadlines. The exercises in a set could be completed in any order. The students were given unlimited tries within the weekly deadlines, and were allowed to use whatever materials or ways of communication. The database schema as well as a short description of the business domain was also visible during the whole process, and students could obtain more information on the database objects using built-in SQLite commands. After a deadline had passed, the students were given example answers for the respective set of exercises. Course contents prior to and during data collection are presented in Table 2, and the structure is common for a database course (Topi et al., 2010), containing enhanced/extended entity-relationship model (EER), transformation

Table 1: Query concepts for each exercise, the numbers of source tables and subject tables, and the total number of tables needed in the formulation of a correct query, based on Taipalus et al. (2018)

Exercise	Concepts	Source	Subject	Total
A1	single-table; expressions	1	1	1
A2	single-table; expressions; ordering	1	1	1
A3	single-table; wildcard; expressions with nesting	1	1	1
B4	multi-table; expressions; facing foreign keys	1	1	2
B5	multi-table; expressions; ordering	1	3	3
B6	multi-table; expressions with nesting; ordering	1	2	3
B7	multi-table; expressions; does not exist	1	2	2
B8	multi-table; does not exist; equal subqueries	1	2	3
B9	single-table; expressions; aggregate functions	1	1	1
B10	multi-table; expressions; multiple source tables	2	3	4
B11	multi-table; expressions; self-join; aggregate function evaluated against a column value; correlated subquery	1	2	2
B12	multi-table; expressions; aggregate function evaluated against a constant; uncorrelated subquery; parameter distinct	1	1	2
B13	multi-table; expressions; self-join	1	5	5
C14	multi-table; multiple source tables; aggregate functions; grouping	2	1	2
C15	multi-table; multiple source tables; aggregate functions; grouping; grouping restrictions; ordering	2	1	2

from EER to relational schema, relational calculus, and SQL sublanguages data manipulation language (DML), data definition language (DDL), data control language (DCL), and transaction control language (TxCL). The course continues with database normalization, data warehousing, database distribution, and NoSQL. The course structure was the same for each of the three cohorts.

4.2. Databases

We implemented the exercise databases with hand-crafted data. For clearer distinguishability, we call these databases "simple", "semi-complex" and "complex", although, compared to real life databases, they are all relatively simple. The logical complexities of the databases, some summarizing information, and, for comparison, additional databases from literature are presented in Table 3. Our three databases were normalized to Boyce/Codd normal form. In terms of data, the tables in the simple database contained 17–73 rows, with the average of approximately 46 rows, the tables in the semi-complex database 5–105 rows, with the average of approximately 50 rows, and the tables in the complex database 5–125 rows, with the average of approximately 61 rows. We designed the data to contain no anomalies or errors, and null values were only present in obvious columns, e.g., in a customer's email or an actor's date of death, as opposed to null values in foreign key columns. For these reasons, it was more feasible to hand-craft the data, rather than using automatic data generation tools such as DBMonster[2] or Mockaroo[3] and evaluating the schema with tools such as SchemaAnalyst (Kapfhammer et al., 2013; McMinn et al., 2016).

The textbook databases described in Table 3 are presented in their respective sections concerning SQL. These textbooks also present other databases in, e.g., sections addressing conceptual modeling or data warehousing. Note that in Elmasri and Navathe (2016), NFK, and hence DRT, are counted

[2]http://dbmonster.sourceforge.net/
[3]https://mockaroo.com/

Table 2: Course activities prior to and during data collection

Week	Course activity (chronologically ordered for each week)
n	Lectures: general concepts in database systems, conceptual modeling with EER
n+1	Lectures: relational model, transformation from EER to relational schema
n+2	Lectures: relational calculus, DML Exercises #1 presented: conceptual modeling with EER
n+3	Lectures: DML, DDL Answers for exercises #1 presented Exercises #2 presented: transformation from EER to relational schema, SQL exercise set A
n+4	Lectures: DCL, TxCL Answers for exercises #2 presented Exercises #3 presented: SQL exercise set B
n+5	Lectures: database normalization Answers for exercises #3 presented Exercises #4 presented: SQL exercise set C, additional SQL exercises (DML, DDL, DCL)
n+6	Lectures: data warehousing Answers for exercises #4 presented Exercises #5 presented: database normalization
	Course continues

using the table creation statements (p. 211) presented in the textbook. If the schema complexity is evaluated based on the database schema (p. 194), NFK = 8 and DRT = 5.

4.3. Protocol and method

After the last deadline for the last student cohort had passed, we collected all the submitted queries for the 15 exercises from our learning environment, a total of over 123,000 SQL queries. Some students had attempted to complete the course and exercises in a previous year or years. To achieve independence of observations, we removed all but first attempts from the data, i.e., if a student tried to complete the exercises in year *n*, *n+1* and *n+2*, we omitted their answers from years other than *n*. In order to study success rates and the numbers of persistent errors, we were only interested in the final queries from each student for each exercise. After omitting all non-final queries (i.e., queries submitted chronologically before the last query), we were left with 8,771 queries. Next, using the error categorization framework (Taipalus et al., 2018), we coded each final query with errors it exhibited, if any. We considered a query incorrect if it contained at least one syntax, semantic, or logical error. A query which contained only a complication or complications was considered correct.

We first conducted a chi-square test of homogeneity using count data with weighted cases to examine the relation between database complexity (simple, semi-complex, complex) and success rate in respective final queries. Not all students attempted all exercises, and we considered non-attempts as failures. We argue for our decision with a minimal counterexample of two students cohorts, ten students each: in cohort A, only one student tries to complete an exercise and succeeds (success rate = 100%), and in cohort B, all ten students try to complete an exercise but only three succeed (success rate = 33%). We consider our protocol to better reflect the equivalence of the research setting between the cohorts, as opposed to ignoring non-attempts. The chi-square test of homogeneity fit our data and research design, as we had a sufficiently large sample size, and, by design, independence of observations.

To test the auxiliary hypotheses, we compared the numbers of errors committed for each error class against the databases of different complexity. The data were not normally distributed between groups

Table 3: Database business domains and complexities (NT = number of tables, NA = number of attributes, NFK = number of foreign keys, DRT = depth referential tree, COS = cohesion of the schema) - databases marked with an asterisk can also be found in the Teradata University Network

	Business domain	NT	NA	NFK	DRT	COS
Simple	social media	5	22	8	3	25
Semi-complex	rally timing	7	32	8	3	49
Complex	movie rental	11	54	12	4	121
Hoffer et al. (2014)	order catalog	4	17	3	2	16
Kroenke and Auer (2016)	order catalog	5	27	2	1	11
Elmasri and Navathe (2016)*	company employees	6	28	6	3	36
Connolly and Begg (2015)	property rental	6	39	6	3	36
Hoffer et al. (2011)*	product lines	15	59	13	2	153
Sakila (2019)	movie rental	16	88	23	7	256

(simple, semi-complex, and complex), but the distributions of the number of errors committed were similar for all groups. Additionally, group sizes were not equal (745, 1,116, and 791 incorrect final queries). For these reasons, we ran a Kruskal-Wallis H test (one for syntax errors, one for semantic errors, and one for logical errors) to determine if there were differences in the number of errors committed between the database groups of different complexity. Finally, we ran a Kruskal-Wallis H test to determine if there were differences in the number of complications committed between the database groups. As complications by themselves do not constitute in making a query incorrect, we ran the test on final queries regardless of their correctness (2,870, 3,425, and 2,476 final queries).

5. Results

The null hypothesis for a chi-square test of homogeneity is that in all groups of the independent variable, the proportions are equal in the population, while the alternative hypothesis is that not all group population proportions are equal. There was a statistically significant difference between the three independent binomial proportions ($p < .001$). Therefore, we can reject the null hypothesis and accept the alternative hypothesis.

We analyzed 11,160 cases (8,771 queries and 2,389 cases of non-attempts) from a total of 744 students, each assigned to a group writing queries against either a simple, semi-complex or complex exercise database. The group of 227 students writing queries against the simple database had a higher success rate (62.4%) compared to the group of 280 students with the semi-complex database (55.0%), and to the group of 237 students with the complex database (47.4%). Post hoc analysis involved pairwise comparisons using the z-test of two proportions with a Bonferroni correction. All pairwise comparisons were statistically significant. The success rates for each database are visualized in Fig. 1, and the success rates for each exercise for each database in Fig. 2.

The null hypothesis for a Kruskal-Wallis H test is that the distribution of the number of errors (syntax, semantic, or logical) for the groups are equal, while the alternative hypothesis is that the distribution of the number of errors (syntax, semantic, or logical) are not equal. The Kruskal-Wallis H test is a common nonparametric alternative to one-way ANOVA (Ruxton and Beauchamp, 2008).

A Kruskal-Wallis H test was run to determine if there were differences in the number of syntax errors committed between three groups of students writing queries against four databases of different complexity: "simple" ($N = 745$), "semi-complex" ($N = 1{,}116$) and "complex" ($N = 791$), where N represents the number of incorrect final queries submitted by each group in total. Distributions of the number of syntax errors committed were similar for all groups, as assessed by visual inspection of a boxplot. Median number of syntax errors committed were statistically significantly different between groups, $H(2) = 23.481$, $p < .001$. Subsequently, pairwise comparisons were performed using

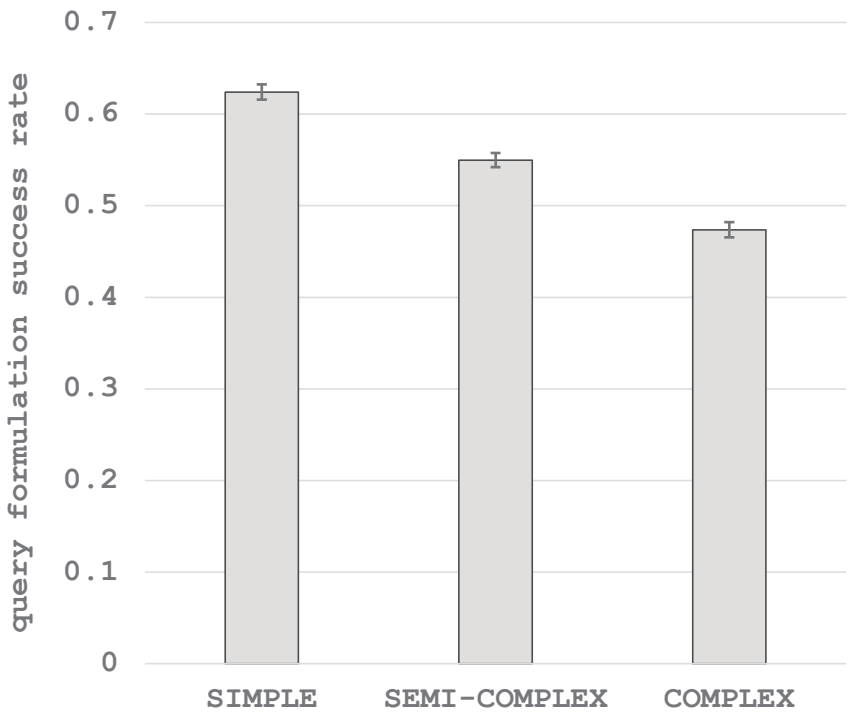


Figure 1: Success rates (mean ± SEM) for each database

Dunn's (1964) procedure with a Bonferroni correction for multiple comparisons. Adjusted p-values are presented. This post hoc analysis revealed statistically significant differences in the number of syntax errors committed between the simple (mean rank = 1,279.10) and complex (mean rank = 1,420.85)($p < .001$), and semi-complex (mean rank = 1,291.26) and complex ($p < .001$) database complexity groups, but not between the simple and semi-complex database complexity group.

A Kruskal-Wallis H test was run to determine if there were differences in the number of semantic errors committed between four groups of students writing queries against four databases of different complexity: "simple" ($N = 745$), "semi-complex" ($N = 1{,}116$) and "complex" ($N = 791$). Distributions of the number of semantic errors committed were similar for all groups, as assessed by visual inspection of a boxplot. Median numbers of semantic errors committed were not statistically significantly different between groups, $H(2) = 5.314$, $p = .070$.

A Kruskal-Wallis H test was run to determine if there were differences in the number of logical errors committed between three groups of students writing queries against three databases of different complexity: "simple" ($N = 745$), "semi-complex" ($N = 1{,}116$) and "complex" ($N = 791$). Distributions of the number of semantic errors committed were similar for all groups, as assessed by visual inspection of a boxplot. The numbers of logical errors committed were statistically significantly different between groups, $H(2) = 14.280$, $p = .001$. Subsequently, pairwise comparisons were performed performed using Dunn's (1964) procedure with a Bonferroni correction for multiple comparisons. Adjusted p-values are presented. This post hoc analysis revealed statistically significant differences in the number of logical errors committed between the complex (mean rank = 1,250.20) and simple (mean rank = 1,341.41)($p = .031$), and complex and semi-complex (mean rank = 1,370.62)($p < .001$) database complexity groups, but not between simple and semi-complex.

A Kruskal-Wallis H test was run to determine if there were differences in the number of complications committed between three groups of students writing queries against three databases of different

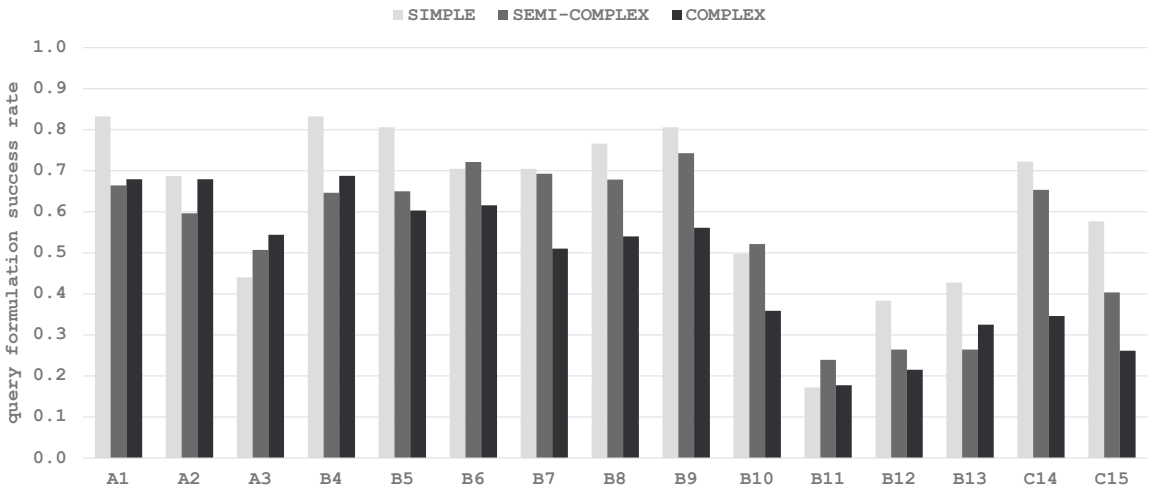


Figure 2: Success rates for each exercise for each database

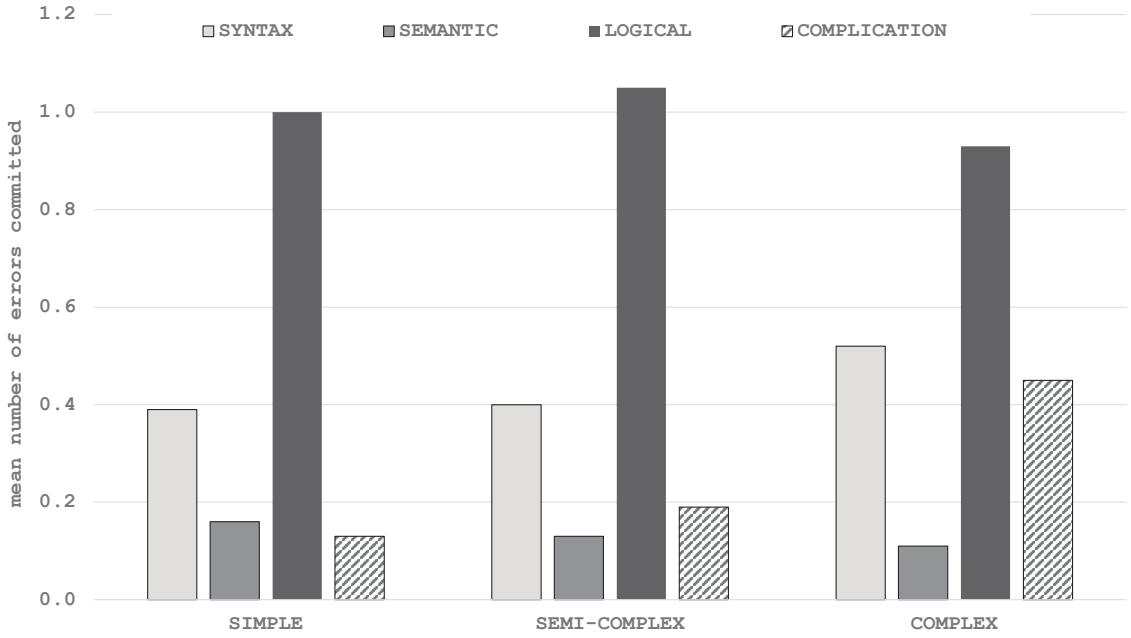


Figure 3: Means for each error class for each database

complexity: "simple" (N = 2,870), "semi-complex" (N = 3,425) and "complex" (N = 2,476). Distributions of the number of complications committed were similar for all groups, as assessed by visual inspection of a boxplot. The numbers of complications committed were statistically significantly different between groups, $H(2) = 717.363$, $p < .001$. Subsequently, pairwise comparisons were performed performed using Dunn's (1964) procedure with a Bonferroni correction for multiple comparisons. Adjusted p-values are presented. This post hoc analysis revealed statistically significant differences in the number of complications committed between the simple (mean rank = 3,932.88) and semi-complex (mean rank = 4,173.22)($p < .001$), and simple and complex (mean rank = 5,205.56)($p < .001$), and semi-complex and complex ($p < .001$) database complexity groups. These numbers for four error

classes for each of the three databases are visualized in Fig. 3 as means (rather than mean ranks) for readability.

Based on the aforementioned results, we can conclude that the basic proposition H_1, and auxiliary hypothesis H_5 were supported. Auxiliary hypothesis H_2 was supported, but the increase from simple database to semi-complex was not statistically significant. Auxiliary hypothesis H_3 was not supported, and had a negative, but statistically non-significant effect. Auxiliary hypothesis H_4 was not supported.

6. Discussion

6.1. Why the success rates differ

The chi-square test of homogeneity indicates that there is an association between database complexity and success rate, and on the basis of the evidence currently available, it seems fair to suggest that more often than not, a logically more complex relational database yields lower success rates than a simpler relational database when students try to write correct SQL queries. Even though this relationship does not appear uniform among all the exercises (Fig. 2), the results overall (Fig. 1) support the position that a more complex database results in lower success rates, and the question under scrutiny is not *if* but rather *why*.

Based on a set of studies by Reisner (1977, 1981, 1988), a seminal study on student errors in SQL query writing by Smelcer (1995) provided the field with six (later abstracted to four in the same study) cognitive explanations on why errors occur. First two, *absence of retrieval cue* and *imprecise retrieval cue* are closely related to the data demand. For example, the data demand "list the names of customers who have rented the same movie as John Doe has rented" lacks the cue to leave John Doe out of the results. However, in addition to the query concept framework, we designed the exercises for all cohorts to follow similar natural language expressions. Next three explanations, *misperception*, *procedural fixedness*, and *inaccurate procedural knowledge*, are closely related to human error, and lack of knowledge concerning the relational model, the business domain, or SQL. We believe that although these three explanations matter in the comparison of success rates between the cohorts, the differences of their effects between the cohorts are minor due to our research design, as explicated in Section 4.1. Finally, and in our opinion, most importantly, Smelcer (1995) explains SQL errors with *exceeding working memory's capacity* (Miller, 1956): when the number of query concepts, expressions, or database objects in a task increases, a student's working memory capacity exceeds, and errors (omission errors in particular) occur. Our results seem to support the observations presented by Smelcer (1995), although the connection is not straightforward. What is worth noting is that between our three cohorts, the query concepts, number of required tables, and database objects in a task are the same by design (cf. Table 1), and it is the complexity of the database which increases. The view that more database objects (were they merely present in the database, or also part of a query being written) cause more strain on working memory is in line with common sense. Based on the results by Smelcer (1995) and our research, we suggest that it is not only the complexity of the task that affects the success rate, but also the logical complexity of the exercise database. For future research, mapping errors to their cognitive explanations via e.g., student interviews would be a valuable addition to understanding *why* errors occur with databases of different complexities.

In this study, we did not consider student engagement, but intuitively, more interesting exercises should result in both more students trying to complete the voluntary exercises, and students engaging more in the exercises, e.g., a less interested student attempting 5 times, and a more interested student attempting 10 times to solve an exercise before giving up. In the analyses, we considered that a student had *attempted* to solve an exercise if they had written at least one SQL query. A post hoc inspection of attempt rates revealed that the cohort with the simple database had the highest attempt rates for 11 of the 15 exercises, while the cohort with the complex database had the lowest attempt rates for all exercises. This might suggest that the students in the cohort with the simple database (social media) were more interested in completing the exercises than the students in the cohort with semi-complex (rally timing) and the complex database (movie rental). This might be due to database complexity, but also due to the database business domain.

This leads to another point we feel compelled to make. Making mistakes is part of any learning process, and it is rare that a student is able to write the correct query on the first attempt. Moreover, even if a student is not able to formulate the correct query at all, the errors committed during the writing process constitute to learning, but non-attempts do not. This propounds the view that measuring success rates while ignoring non-attempts leaves out the factor of how many of the students in a cohort even attempted, thus possibly biasing the results towards higher success rates. In contrast, measuring the perceived interest and usefulness of the exercises, as studied by Yue (2013), leaves out the factor of success rates, as successfully formulating a query implies that a student has achieved the required level of knowledge in SQL, whereas failure to do so implies the opposite.

6.2. Considerations on lower success rates

Our results show statistically significant differences in success rates between the databases of different complexities. However, we do not wish to infer that a high success rate in query writing is a metric that educators should necessarily strive for, or that a high success rate conflates with learning. Arguably, the more a student commits errors, the more misconceptions are uncovered and uncertainties remedied. That being said, we did not consider the number of errors a student committed, only the number of persistent errors. As stated earlier, a persistent error arguably represents a misconception or uncertainty that is not remedied, at least not during the query writing process.

Prior studies have provided evidence on the positive effects of more complex databases, and while our results are not in conflict, they shed light on the possible negative effects. The data yielded by this study provides considerations for future research, as many questions regarding the matter of exercise database complexity remain open. If students learn SQL using more complex exercise databases, does that imply that the students are more familiar with complex databases, but do not have the skills to formulate correct SQL queries? In contrast, if students learn SQL using simpler exercises databases, does that imply that the students have the skills to formulate correct SQL queries, but not in complex database environments? If a more realistic database is demonstrated to cause positive feelings (i.e., it is interesting and useful, Yue, 2013) in students, does a low success rate in query formulation cause negative feelings in students towards SQL, query languages, or databases in general? These considerations also propound the future research question of how simple exercise database is too simple, and how complex is too complex.

Finally, as discussed by, e.g., Denny et al. (2012) in the context of programming languages, students have different levels of capability, and by making the task more difficult, performance decreases (Topi et al., 2005). With these considerations in mind, it is intuitive that students with high capability have a tendency to perform better than students with low capability, regardless of the task complexity. As our results have provided evidence that an increase in database complexity (as opposed to task complexity) also results in decrease in performance, it seems justified to foster debate whether more complex databases emphasize the capability differences between students.

6.3. Implications for research

Only one of the auxiliary hypotheses, H_5, was supported with a statistically significant effect. Consequently, while we cannot infer from our results that database complexity affects the number of syntax, semantic, or logical errors, complications seem to increase with a statistically significant effect as the the complexity of the database increases. According to the error categorization (Taipalus et al., 2018), complications can be, e.g., unnecessary joins, ordering in a subquery, or unused correlations names (i.e., aliases). As complications do not affect the result table, but query readability or computational performance, their severity is below that of other errors. Furthermore, it is theoretically possible to reliably identify complications in queries with computerized automation (Brass and Goldberg, 2006), as opposed to, e.g., identifying logical errors. Persistent and non-persistent SQL errors have been identified earlier (Taipalus and Perälä, 2019), but based on the evidence currently available, it seems reasonable to suggest that error persistence in regards to error class is not affected by database complexity.

An interesting set of studies by Bowen et al. (2004, 2009) investigated whether *ontological clarity* affects query writing performance. The authors effectively designed two relational databases with the same business domain. One database was designed following widely accepted design guidelines at the cost of ontological clarity, resulting in a simpler database structure. The other database was designed with the prioritization of ontological clarity, resulting in a more complex database structure. Their results indicated that the participants writing queries against the ontologically clearer database committed more semantic errors, took longer to write their queries, and were less confident in the accuracy of their queries than the participants writing queries against the ontologically less clear database. With the omittance of the factor of ontological clarity, our results provide an indication that increased structural complexity negatively affects query formulation performance.

6.4. Implications for teaching

Intuitively, there were three possible outcomes of this study; a more complex database either causes a decrease or an increase in success rates, or the success rates remain the same despite the change in database complexity. Depending on the results, and with Yue's study (2013) in mind, we encourage teachers to utilize simpler exercise databases now that the results suggest a decrease in success rates. We would like to point out that the two other possible outcomes would have been equally interesting, and in those cases we would have argued for the use of more complex databases. However, as discussed earlier, our results leave room for interpretation, and, given that a teacher has time, more than one exercise database can be utilized.

Although it is not apparent in the study by Yue (2013) whether the students found a more complex database more interesting and useful due to complexity or something else, for the sake of discussion, we would like to argue that structural complexity increases perceived usefulness and student interest. Furthermore, if an increase in structural complexity indeed implies decrease in success rates, we as researchers and teachers should either 1) consider other ways besides increasing structural complexity to convey interesting and useful exercise databases to students, or 2) support learning SQL in complex databases with a different or an auxiliary method. That said, if an interesting and useful database is inevitably also complex, we suggest utilizing both of the above. Finally, if the differences in success rates between simple and complex databases are indeed caused by increased load on working memory, we propose that the earlier, rather ambiguously phrased *auxiliary method* could be considered a way to simplify the SQL syntax, semantics, and the database structure into a form that puts less strain on a student's working memory. As a possible solution, we are currently investigating how a notation for planning more complex SQL queries (Taipalus, 2019) affects SQL query formulation in more complex exercise databases. In addition to the environment, concerns about the relationship between language syntax and cognitive load have been raised in the context of programming languages (Kelleher and Pausch, 2005; Lister, 2011a,b). Ahadi et al. (2016a) conclude their study on SQL syntax errors by noting that while semantic errors require more creative problem solving, solving them is not feasible until possible syntax errors are fixed. With this in mind, the relative difference in the means of syntax and logical errors (Fig. 3) should not be considered an indicator that syntax errors are somehow less important.

As shown in Table 2.2, our databases are somewhat similar in complexity to those presented in learning environments and textbooks. When a teachers chooses an exercise database for a course, the appropriate structural complexity depends on the difficulty of the planned exercises, student backgrounds (e.g., majoring in business analytics versus software engineering), as well as teacher skill and experience. Furthermore, a single database course is not necessarily limited to a single exercise database. A teacher may utilize a simple database to teach query concepts in theory and through examples, yet utilize a complex database against which the students can practice query formulation.

Finally, although in the vein of Yue (2013), we have effectively treated a more realistic database as a synonym for a more complex database, this connection does not necessarily hold true. The growing trend of, e.g., microservice architectures (Alshuqayran et al., 2016) and mobile applications are often concerned with subsets of business domains, and do not necessarily address structurally complex

databases. This puts forward the topical view that more realistic databases are not necessarily more complex, and educators should consider using databases which are both realistic (and thus engaging), and relatively simple (and thus query formulation is likely more successful). That said, what is an engaging business domain among students remains an open question. While the answer is changing and subjective, student engagement to different database domains is an interesting future research topic. In conclusion, the database feature of being more or less realistic is simply a student's *perception* of realistic. If educators can demonstrate that a simple exercise database indeed reflects the structure of a realistic database, that might positively affect student engagement without negatively affecting query formulation.

6.5. Limitations

There are two main limitations that affect the generalizability of the results of this study. First, the data were collected from one university, and a single course which took place three times over three years. This presents the question whether similar results could be obtained from students taking other database courses in other universities or under other teachers. As this study was to our knowledge the first to explore the effects of database complexity on query writing performance, it is not possible to compare the our results to other studies. Second, only a subset of SQL concepts, even in the scope of data retrieval, were studied. Then again, a narrower study scope does not necessarily imply weaker research, as argued for by Siponen and Klaavuniemi (2019).

6.6. Threats to validity

We wanted to study the effects of database complexity on SQL query formulation. Prior to the study, we identified seven control variables that could affect our results (cf. Fig. 4), and designed our research setting to mitigate the effects of these variables. Next, we discuss how these variables might have affected the results of this study, describe the measures (labels *a-g* in Fig. 4) we took to mitigate these effects, and argue for the choices we made concerning the research setting.

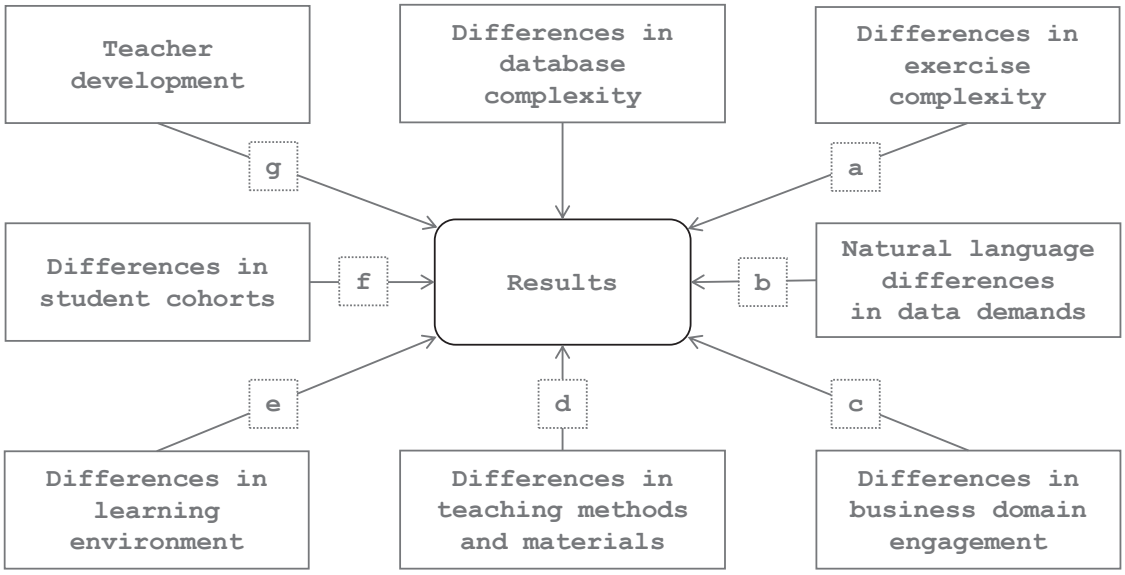


Figure 4: Variables potentially affecting the results

As we studied three students cohorts, each with their respective database and exercises, the effects of differences in exercise complexity (Fig. 4a) needed to be mitigated. We designed the exercises for

each cohort according to the query concept framework, which lists query concepts and the number of source and subject tables needed for each of the 15 exercises. Designing the exercises with this framework allowed us to control the complexity of the exercises, thus mitigating the risk that one cohort had easier or more difficult exercises than another.

Each of the three cohorts wrote queries against a database with a different business domain. Consequently, the data demands for each cohort were different from each other (Fig. 4b), e.g., one cohort had to list the names of social media users, one of rally drivers, and one of customers. Although these natural language considerations are relatively minor due to the fact that the exercises were designed using the same framework, natural language entails ambiguity (Borthick et al., 2001; Casterella and Vijayasarathy, 2013; Reisner, 1981). We tried to minimize the effects of natural language on query writing by providing the students with the correct result table, and we did not consider the number of tries a student needed to formulate the correct query. We hoped that if students saw that their result table differed from the correct result table, it would effectively steer students toward the correct interpretation of the data demand. Similarly, the effects of different database business domains (Fig. 4c) may have affected the number of students who decided to attempt the exercises. It has also been shown that understanding the business domain affects query writing performance (Siau et al., 2004). We considered using a single business domain and a single database, and modularly adding (or subtracting) tables and attributes for each cohort. However, we found it increasingly difficult to utilize the query concept framework and come up with at least somewhat realistic data demands. We also considered using the modular approach with same data demands for each cohort. This was not deemed feasible for two reasons. First, it would have meant that that some of the tables would not have been utilized in any query, for any cohort. In our study, the simple database contained five tables and the complex database eleven, and with same data demands, the remaining six tables of the complex database would not have been used in any of the queries. Second, based on our previous teaching experiences, some students have shared the example answers from previous years in different forums. Even the most diligent student may be tempted to look up an example answer to an exercise they could not solve, thus achieving more course points. For these reasons, we designed new exercises and databases for each student cohort, and strived to utilize business domains that are at least some way familiar to students.

We mitigated the effects of differences in teaching methods and materials (Fig. 4d), and in the learning environment (Fig. 4e) by using the same teaching materials (slides, handouts), not making adjustments to the teaching methods, and retaining the course outline (cf. Fig. 2) for all cohorts. All cohorts used the same e-learning environment and database management system (SQLite) even though the pedagogical shortcomings of SQLite became increasingly apparent during the study. The first author taught the course for each cohort, and also coded the queries according to the error categorization framework. It is possible that there were misinterpretations of the framework, but possible misinterpretations were at least consistent between the cohorts.

As we elaborated in Section 4.1, the students formulated the queries in a minimally controlled environment, and there is a possibility that the student cohorts studied were, in some unforeseeable way, different from each other (Fig. 4f). Perhaps there was a growing trend that students communicate with each other more and more, perhaps students are more and more skilled in utilizing internet search engines, or perhaps students have more and more certain personal characteristics - a factor which has been studied to affect query writing performance (Ashkanasy et al., 2007). Additionally, and although we used the same teaching materials for each cohort, it is possible, even likely, that the teacher's skills develop over time (Fig. 4g), thus possibly positively affecting the development of success rates over time. If such trends or development exist, we tried to mitigate the effects by following a schedule. Instead of chronologically gradually increasing or decreasing the database complexity for the cohorts, we utilized the complex database for the first cohort, the simple for the second, and the semi-complex for the third. Furthermore, the teacher had taught the same course for years before the first cohort subject of this study, so major developments in teaching skills were not likely.

In summary, we made deliberate choices to favor a more natural environment for the students to

write their queries. Although, as opposed to a more controlled environment, this presented several threats to validity, but in turn allowed us to study query writing that more accurately reflects the students' future work environments. Additionally, our data collection method allowed for a relatively large number of students and queries to be studied, whereas a more controlled experiment, at least in our experience, would possibly have yielded significantly less participants. We believe that the relatively large sample sizes compensate for the margin of error presented possibly by the less controlled environment.

7. Conclusion

In this study, we set out to investigate whether the logical structural complexity of a relational database affects the success rates of students writing SQL queries against three databases of varying complexity. Overall, the results show statistically significant differences between the different databases, which indicates that students are less likely to formulate correct SQL queries if the exercise database is complex. Rather than suggesting the usage of simpler databases when teaching SQL, we encourage educators to consider the potential negative effects of more complex databases on SQL learning, as it has been demonstrated that more complex databases also bring beneficial effects to teaching.

Acknowledgements

The authors would like to thank Hilkka Grahn for her invaluable advice regarding the data analysis and grammar, and the associate editor and anonymous reviewers for their helpful insights and comments on how to improve the paper.

Declarations of interest

None.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

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 '16), ACM Press, New York, New York, USA. pp. 401–406. doi:10.1145/2839509.2844640.

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 '16), ACM Press, New York, New York, USA. pp. 272–277. doi:10.1145/2899415.2899464.

Alshuqayran, N., Ali, N., Evans, R., 2016. A systematic mapping study in microservice architecture, in: 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), IEEE. pp. 44–51. doi:10.1109/SOCA.2016.15.

Ashkanasy, N., Bowen, P.L., Rohde, F.H., Wu, C.Y.A., 2007. The effects of user characteristics on query performance in the presence of information request ambiguity. Journal of Information Systems 21, 53–82. doi:10.2308/jis.2007.21.1.53.

Borthick, A., Bowen, P.L., Jones, D.R., Tse, M.H.K., 2001. The effects of information request ambiguity and construct incongruence on query development. Decision Support Systems 32, 3 – 25. doi:10.1016/S0167-9236(01)00097-5.

Bowen, P., O'Farrell, R., Rohde, F., 2004. How does your model grow? An empirical investigation of the effects of ontological clarity and application domain size on query performance, in: Proceedings of the International Conference on Information Systems (ICIS), p. 7. URL: https://aisel.aisnet.org/icis2004/7.

Bowen, P.L., O'Farrell, R.A., Rohde, F.H., 2009. An empirical investigation of end-user query development: The effects of improved model expressiveness vs. complexity. Information Systems Research 20, 565–584. doi:10.1287/isre.1080.0181.

Brass, S., Goldberg, C., 2006. Semantic errors in SQL queries: A quite complete list. Journal of Systems and Software 79, 630–644. doi:10.1016/j.jss.2005.06.028.

Calero, C., Piattini, M., Genero, M., 2001. Metrics for controlling database complexity, in: Developing Quality Complex Database Systems: Practices, Techniques and Technologies. IGI Global, pp. 48–68. doi:10.4018/9781878289889.ch003.

Casterella, G.I., Vijayasarathy, L., 2013. An Experimental Investigation of Complexity in Database Query Formulation Tasks. Journal of Information Systems Education 24, 211–221. URL: http://jise.org/Volume24/24-3/pdf/Vol24-3pg211.pdf.

Chan, H., Siau, K., Wei, K.K., 1997. The effect of data model, system and task characteristics on user query performance: an empirical study. SIGMIS Database 29, 31–49. doi:10.1145/506812.506820.

Chan, H.C., Teo, H.H., Zeng, X., 2005. An evaluation of novice end-user computing performance: Data modeling, query writing, and comprehension. Journal of the American Society for Information Science and Technology 56, 843–853. doi:10.1002/asi.20178.

Connolly, T., Begg, C., 2015. Database Systems (6th. ed.). Pearson.

Denny, P., Luxton-Reilly, A., Tempero, E., 2012. All syntax errors are not equal, in: Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE), ACM, New York, NY, USA. pp. 75–80. doi:10.1145/2325296.2325318.

Dunn, O.J., 1964. Multiple comparisons using rank sums. Technometrics 6, 241–252. doi:10.2307/1266041.

Elmasri, R., Navathe, S.B., 2016. Fundamentals of Database Systems (7th. ed.). Pearson.

Hoffer, J.A., Ramesh, V., Topi, H., 2011. Modern database management. Upper Saddle River, NJ: Prentice Hall.

Hoffer, J.A., Topi, H., Ramesh, V., 2014. Essentials of Database Management. Pearson Education.

ISO/IEC, 2016. ISO/IEC 9075-2:2016, "SQL - Part 2: Foundation". URL: https://www.iso.org/standard/63556.html.

Jukic, N., Gray, P., 2008a. Teradata university network: A no cost web-portal for teaching database, data warehousing, and data-related subjects. Journal of Information Systems Education 19, 395–402. URL: http://jise.org/Volume19/n4/JISEv19n4p395.html.

Jukic, N., Gray, P., 2008b. Using real data to invigorate student learning. SIGCSE Bulletin 40, 6–10. doi:10.1145/1383602.1383604.

Kapfhammer, G.M., McMinn, P., Wright, C.J., 2013. Search-based testing of relational schema integrity constraints across multiple database management systems, in: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, IEEE. doi:10.1109/icst.2013.47.

Kelleher, C., Pausch, R., 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys 37, 83–137. doi:10.1145/1089733.1089734.

Kroenke, D., Auer, D.J., 2016. Database Processing: Fundamentals, Design, and Implementation (14th. ed.). Pearson Education.

Lister, R., 2011a. Programming, syntax and cognitive load (part 1). ACM Inroads 2, 21–22. doi:10.1145/1963533.1963539.

Lister, R., 2011b. Programming, syntax and cognitive load (part 2). ACM Inroads 2, 16–17. doi:10.1145/2003616.2003622.

McMinn, P., Wright, C.J., Kapfhammer, G.M., 2015. The effectiveness of test coverage criteria for relational database schema integrity constraints. ACM Transactions on Software Engineering and Methodology 25, 8:1–8:49. doi:10.1145/2818639.

McMinn, P., Wright, C.J., Kinneer, C., McCurdy, C.J., Camara, M., Kapfhammer, G.M., 2016. SchemaAnalyst: Search-based test data generation for relational database schemas, in: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE. doi:10.1109/icsme.2016.93.

McMinn, P., Wright, C.J., McCurdy, C.J., Kapfhammer, G.M., 2019. Automatic detection and removal of ineffective mutants for the mutation analysis of relational database schemas. IEEE Transactions on Software Engineering 45, 427–463. doi:10.1109/TSE.2017.2786286.

Miller, G.A., 1956. The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review 63, 81. doi:10.1037/0033-295x.101.2.343.

Mitrovic, A., 1998. Learning SQL with a computerized tutor, in: Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education, ACM, New York, NY, USA. pp. 307–311. doi:10.1145/273133.274318.

Pavlic, M., Kaluza, M., Vrcek, N., 2008. Database complexity measuring method, in: Central European Conference on Information and Intelligent Systems, Faculty of Organization and Informatics Varazdin. URL: `http://archive.ceciis.foi.hr/app/index.php/ceciis/2008/paper/view/84/84`.

Prior, J.C., Lister, R., 2004. The backwash effect on SQL skills grading. SIGCSE Bulletin 36, 32–36. doi:10.1145/1026487.1008008.

Reisner, P., 1977. Use of psychological experimentation as an aid to development of a query language. IEEE Transactions on Software Engineering SE-3, 218–229. doi:10.1109/tse.1977.231131.

Reisner, P., 1981. Human factors studies of database query languages: A survey and assessment. ACM Computing Surveys 13, 13–31. doi:10.1145/356835.356837.

Reisner, P., 1988. Query languages, in: Helander, M. (Ed.), Handbook of Human-Computer Interaction. Elsevier, New York, pp. 257–280.

Rho, S., March, S.T., 1997. An analysis of semantic overload in database access systems using multi-table query formulation. Journal of Database Management 8, 3–15. URL: https://www.igi-global.com/gateway/article/51176.

Ruxton, G.D., Beauchamp, G., 2008. Time for some a priori thinking about post hoc testing. Behavioral Ecology 19, 690–693. doi:10.1093/beheco/arn020.

Sakila, 2019. Sakila sample database (accessed February 2019). URL: https://dev.mysql.com/doc/sakila/en/sakila-structure.html.

Siau, K.L., Chan, H.C., Wei, K.K., 2004. Effects of query complexity and learning on novice user query performance with conceptual and logical database interfaces. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 34, 276–281. doi:10.1109/TSMCA.2003.820581.

Siponen, M., Klaavuniemi, T., 2019. Narrowing the theory's or study's scope may increase practical relevance, in: Proceedings of the Annual Hawaii International Conference on System Sciences, University of Hawai'i at Manoa. pp. 6260–6269. URL: https://scholarspace.manoa.hawaii.edu/handle/10125/60060.

Smelcer, J.B., 1995. User errors in database query composition. International Journal of Human-Computer Studies 42, 353–381. doi:10.1006/ijhc.1995.1017.

Taipalus, T., 2019. Teaching Tip: A Notation for Planning SQL Queries. Journal of Information Systems Education 30, 160–166. URL: http://jise.org/Volume30/n3/JISEv30n3p160.pdf.

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, ACM, New York, NY, USA. pp. 198–203. doi:10.1145/3287324.3287359.

Taipalus, T., Siponen, M., Vartiainen, T., 2018. Errors and complications in SQL query formulation. ACM Transactions on Computing Education 18, 15:1–15:29. doi:10.1145/3231712.

Topi, H., Kaiser, K.M., Sipior, J.C., Valacich, J.S., Nunamaker, Jr., J.F., de Vreede, G.J., Wright, R., 2010. Curriculum Guidelines for Undergraduate Degree Programs in Information Systems. Technical Report. ACM/AIS. New York, NY, USA. URL: https://dl.acm.org/citation.cfm?id=2593310.

Topi, H., Valacich, J.S., Hoffer, J.A., 2005. The effects of task complexity and time availability limitations on human performance in database query tasks. International Journal of Human-Computer Studies 62, 349–379. doi:10.1016/j.ijhcs.2004.10.003.

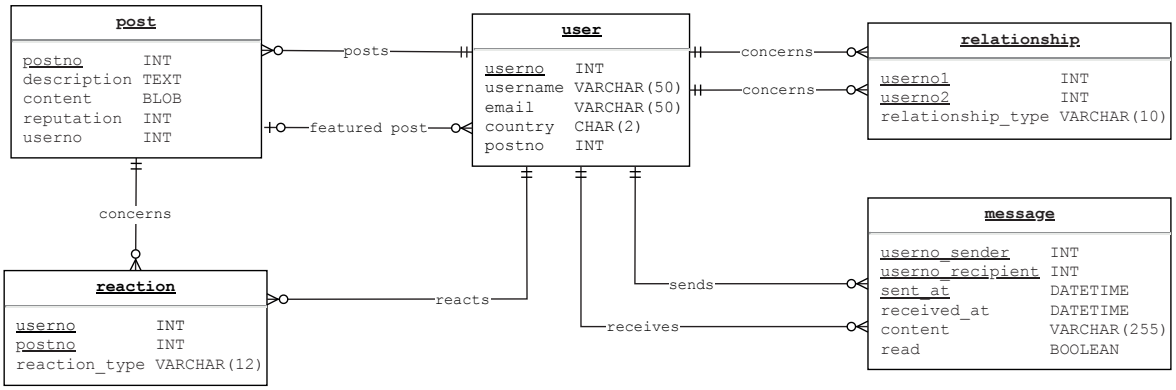
Wagner, P.J., Shoop, E., Carlis, J.V., 2003. Using scientific data to teach a database systems course, in: Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, ACM, New York, NY, USA. pp. 224–228. doi:10.1145/611892.611975.

Warszawski, T., Bailis, P., 2017. ACIDRain: Concurrency-related attacks on database-backed web applications, in: Proceedings of the 2017 ACM International Conference on Management of Data, ACM, New York, NY, USA. pp. 5–20. doi:10.1145/3035918.3064037.

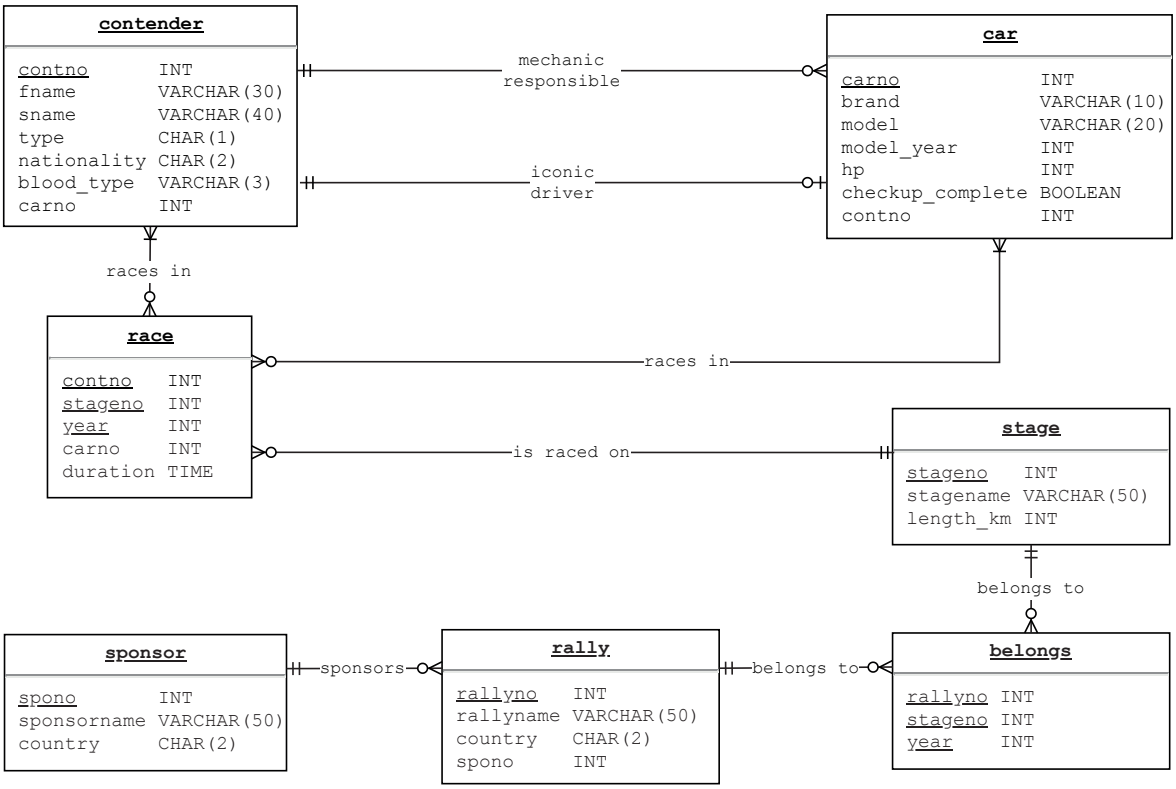
Watson, H.J., Hoffer, J.A., 2003. Teradata university network: A new resource for teaching large data bases and their applications. Communications of the Association for Information Systems 12, 9. URL: https://aisel.aisnet.org/cais/vol12/iss1/9/.

Yue, K.B., 2013. Using a semi-realistic database to support a database course. Journal of Information Systems Education 24, 327–336. URL: http://jise.org/Volume24/n4/JISEv24n4p327.html.

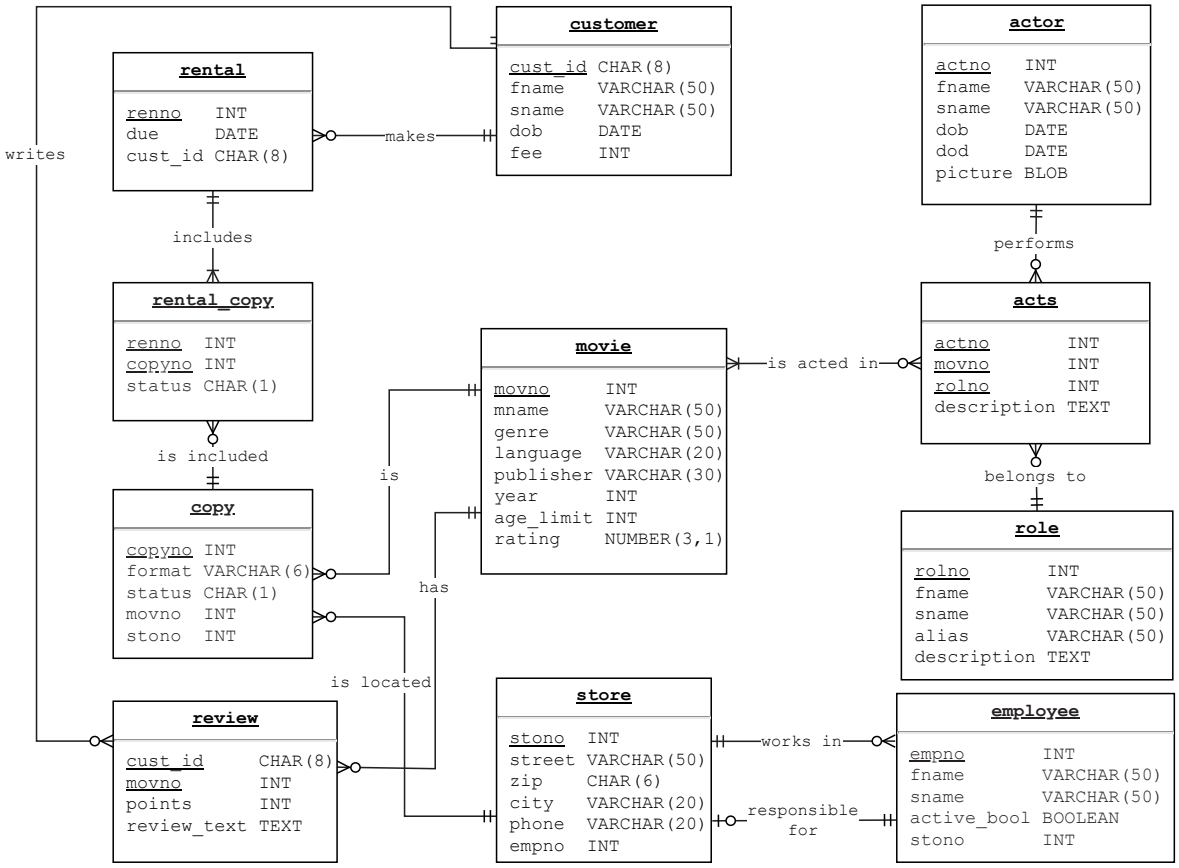
Appendix A. Simple database schema



Appendix B. Semi-complex database schema



Appendix C. Complex database schema (Taipalus et al., 2018)



Appendix D. Data demands and queries

	Simple	Semi-complex	Complex (Taipalus et al., 2018)
A1	List all information regarding users from Finland (FI) and Sweden (SE). SELECT * FROM user WHERE country IN ('FI', 'SE');	List all information regarding cars from Opel and Toyota. SELECT * FROM car WHERE brand IN ('Opel', 'Toyota');	List all information regarding stores in Helsinki and Tampere. SELECT * FROM store WHERE city IN ('Helsinki', 'Tampere');
A2	List the user numbers, user names, countries and emails of users who are Australian (AU) but have no featured post. Sort the results according to user name in ascending order. SELECT userno, username, country, email FROM user WHERE country = 'AU' AND postno IS NULL ORDER BY username ASC;	List the names, nationalities and blood types of co-drivers who are not Finnish (FI). Sort the results according to surname in ascending order. SELECT fname, sname, nationality, bloodtype FROM contender WHERE type = 'c' AND nationality <> 'FI' ORDER BY surname ASC;	List the names, age limits and years of movies that are in English but are not published by Goldeneye BC. Sort the results according to the name of the movie in ascending order. SELECT mname, age_limit, year FROM movie WHERE language = 'English' AND publisher <> 'Goldeneye BC' ORDER BY mname ASC;
A3	List the post numbers, descriptions and user numbers who made the post, of posts which description starts with an S or an R, and that have been posted by users whose user number is 1001 or 1003. SELECT postno, description, userno FROM post WHERE (description LIKE 'S%' OR description LIKE 'R%') AND (userno = 1001 OR userno = 1003);	List the names and nationalities of contenders whose surname starts with an A or a C, and who are from the United Kingdom (UK) or the United States (US). SELECT fname, sname, nationality FROM contender WHERE (surname LIKE 'A%' OR surname LIKE 'C%') AND (nationality = 'UK' or nationality = 'US);	List the names, dates of birth and death of actors whose surname starts with an F or an S, and whose date of birth is unknown, or who have a date of death. SELECT fname, sname, dob, dod FROM actor WHERE (sname LIKE 'F%' OR sname LIKE 'S%') AND (dob IS NULL OR dod IS NOT NULL);
B4	List user names and emails of users who have a featured post with no description. SELECT u.username, u.email FROM user u, post p WHERE u.postno = p.postno AND p.description IS NULL;	List the car brand and model of the car of which Ari Vatanen is the iconic driver. SELECT c.brand, c.model FROM car c, contender d WHERE c.contno = d.contno AND d.fname = 'Ari' AND d.sname = 'Vatanen';	List the city and phone number of the store in which Jaakko Mattila works. SELECT s.city, s.phone FROM store s, employee e WHERE s.stono = e.stono AND e.fname = 'Jaakko' AND e.sname = 'Mattila';

	Simple	Semi-complex	Complex (Taipalus et al., 2018)
B5	List the post numbers and descriptions of posts which at least one Finnish (FI) user has considered funny. Sort the results according to post number in descending order. SELECT p.postno, p.description FROM post p INNER JOIN reaction r ON (p.postno = r.postno) INNER JOIN user u ON (r.userno = u.userno) WHERE u.country = 'FI' AND r.reaction_type = 'funny' ORDER BY p.postno DESC;	List the names and countries of rallies which included at least one stage of over 20 kilometers in 1998. Sort the results according to rally name in descending order. SELECT r.rallyname, r.country FROM rally r INNER JOIN belongs b ON (r.rallyno = b.rallyno) INNER JOIN stage s ON (b.stageno = s.stageno) WHERE b.year = 1998 AND s.length_km > 20 ORDER BY r.rallyname DESC;	List the names of actors whose date of death is known and who have acted in at least one movie released after 2010. Sort the results according to surname in descending order. SELECT a.fname, a.sname FROM actor a INNER JOIN acts ac ON (a.actno = ac.actno) INNER JOIN movie m ON (ac.movno = m.movno) WHERE a.dod IS NOT NULL AND m.year > 2010 ORDER BY a.sname DESC;
B6	List the contents and user numbers of the receivers of messages which were sent by an user with user number 1001 or 1003, and who has reacted to some post at least once. Sort the results according to the user number, and then according to content, both in ascending order. SELECT m.content, m.userno_receiver FROM message m WHERE EXISTS (SELECT * FROM user u WHERE m.userno_sender = u.userno AND (u.userno = 1001 OR u.userno = 1003) AND EXISTS (SELECT * FROM reaction r WHERE u.userno = r.userno)) ORDER BY m.userno_receiver ASC, m.content ASC;	List the brands and models of cars which have been driven at least once on stage called Sweet Lamb 1 or Sweet Lamb 2. Sort the results according to brand, and then according to model, both in ascending order. SELECT c.brand, c.model FROM car c WHERE EXISTS (SELECT * FROM race r WHERE c.carno = r.carno AND EXISTS (SELECT * FROM stage s WHERE r.stageno = s.stageno AND (s.stagename = 'Sweet Lamb 1' OR s.stagename = 'Sweet Lamb 2'))) ORDER BY c.brand ASC, c.model ASC;	List the names of actors who have acted a role as himself or herself. Sort the results according to surname, and then according to first name, both in ascending order. SELECT a.fname, a.sname FROM actor a WHERE EXISTS (SELECT * FROM acts ac WHERE a.actno = ac.actno AND EXISTS (SELECT * FROM role r WHERE ac.rolno = r.rolno AND (r.alias = 'Himself' OR r.alias = 'Herself'))) ORDER BY a.sname ASC, a.fname ASC;

	Simple	Semi-complex	Complex (Taipalus et al., 2018)
B7	List the post numbers, contents and reputations of posts which have a reputation greater than 0, but which no one has ever considered funny. SELECT p.postno, p.content, p.reputation FROM post p WHERE p.reputation > 0 AND NOT EXISTS (SELECT * FROM reaction r WHERE p.postno = r.postno AND r.reaction_type = 'funny');	List the names and nationalities of co-drivers who have never raced in 1986-2010. SELECT c.fname, c.sname, c.nationality FROM contender c WHERE c.type = 'c' AND NOT EXISTS (SELECT * FROM race r WHERE c.contno = r.contno AND r.year BETWEEN 1986 AND 2010);	List the movie numbers, names and years of movies that have been released in the first decade of the 2000s, but of which there exists no copy in BluRay format. SELECT m.movno, m.mname, m.year FROM movie m WHERE m.year BETWEEN 2000 AND 2009 AND NOT EXISTS (SELECT * FROM copy c WHERE m.movno = c.movno AND c.format = 'BluRay');
B8	List the user names, emails and countries of users who have never posted anything but who have at least once reacted to a post. SELECT u.username, u.email, u.country FROM user u WHERE NOT EXISTS (SELECT * FROM post p WHERE u.userno = p.userno) AND EXISTS (SELECT * FROM reaction r WHERE u.userno = r.userno);	List the stage numbers, names and lengths of stages which have never been a part of any rally but on which someone has raced at least once. SELECT s.stageno, s.stagename, s.length_km FROM stage s WHERE NOT EXISTS (SELECT * FROM belongs b WHERE s.stageno = b.stageno) AND EXISTS (SELECT * FROM race r WHERE s.stageno = r.stageno);	List the names and dates of birth of customers who have never rented a movie but who have given at least one review. SELECT c.fname, c.sname, c.dob FROM customer c WHERE NOT EXISTS (SELECT * FROM rental rt WHERE c.cust_id = rt.cust_id) AND EXISTS (SELECT * FROM review rv WHERE c.cust_id = rv.cust_id);
B9	List the average of post reputations with a reputation greater than 0. Rename the column in the result table descriptively. SELECT AVG(repuration) AS "average positive reputation" FROM post WHERE reputation > 0;	List the number of stages with the length between 4 and 10 kilometers. Rename the column in the result table descriptively. SELECT COUNT(*) AS "number of 4-10 km stages" FROM stage WHERE length_km BETWEEN 4 AND 10;	List the number of movies released between the years 1970-2000. Rename the column in the result table descriptively. SELECT COUNT(*) AS "movies released in 1970-2000" FROM movie WHERE year BETWEEN 1970 AND 2000;

	Simple	Semi-complex	Complex (Taipalus et al., 2018)
B10	List the user names, emails, countries, and reaction types and post numbers which the reactions concern, but only from users who are married. Rename the columns in the result table descriptively. SELECT u.username AS "user name", u.email AS "email", u.country AS "country", r.reaction_type AS "reaction", p.postno AS "post number" FROM user u, reaction r, post p, relationship s WHERE u.userno = r.userno AND r.postno = p.postno AND u.userno = s.userno1 AND s.relationship_type = 'marriage';	List the names and types of contenders, and the car brands and models with which they have raced on a stage called Ouninpohja. Rename the columns in the result table descriptively. SELECT c.fname AS "first name", c.sname AS "surname", c.type AS "type", a.brand AS "brand", a.model AS "model" FROM contender c, car a, race r, stage s WHERE c.contno = r.contno AND r.stageno = s.stageno AND r.carno = a.carno AND s.stagename = 'Ouninpohja';	List the names of actors who have acted in the movie Physics 101 and list the names of the roles they have played in that movie. Rename the columns in the result table descriptively. SELECT a.fname AS "actor's first name", a.sname AS "actor's surname", r.fname AS "character's first name", r.sname AS "character's surname", r.alias AS "character's alias" FROM movie m, actor a, acts ac, role r WHERE m.movno = ac.movno AND ac.rolno = r.rolno AND a.actno = ac.actno AND m.mname = 'Physics 101';
B11	List the contents, sender user number, and the time the message was sent of the oldest unread message. SELECT content, userno_sender, sent_at FROM message WHERE read = False AND sent_at = (SELECT MIN(sent_at) FROM message WHERE read = False);	List the car number, model year and horse powers of the oldest Toyota. SELECT carno, model_year, hp FROM car WHERE brand = 'Toyota' AND model_year = (SELECT MIN(model_year) FROM car WHERE brand = 'Toyota');	List the name, year and genre of the oldest movie published by Goldeneye BC. SELECT mname, year, genre FROM movie WHERE publisher = 'Goldeneye BC' AND year = (SELECT MIN(year) FROM movie WHERE publisher = 'Goldeneye BC');
B12	List the user names and emails of users who have sent messages to exactly six different users. SELECT u.username, u.email FROM user u WHERE 6 = (SELECT COUNT(DISTINCT m.userno_recipient) FROM message m WHERE u.userno = m.userno_sender);	List the names of contenders who have raced with at least three different cars. SELECT c.fname, c.sname FROM contender c WHERE 2 < (SELECT COUNT(DISTINCT r.carno) FROM race r WHERE r.carno = c.carno);	List the actor numbers and full names of actors who have acted in at least five different movies. SELECT a.actno, a.fname, a.sname FROM actor a WHERE 4 < (SELECT COUNT(DISTINCT ac.movno) FROM acts ac WHERE a.actno = ac.actno);

	Simple	Semi-complex	Complex (Taipalus et al., 2018)
B13	List the user names and countries of users who have posted a post which has received the at least one similar type of reaction as any post made by user 1004. SELECT u.userno, u.country FROM user u, post p1, post p2, reaction r1, reaction r2 WHERE u.userno = p1.userno AND p1.postno = r1.postno AND r1.reaction_type = r2.reaction_type AND r2.postno = p2.postno AND p2.userno = 1004 AND u.userno <> 1004;	List the numbers and names of rallies which have at least one stage which is of the same length as some stage that has been part of the Rally of Wales (rallyno = 201), whenever. SELECT r.rallyno, r.rallyname FROM rally r, belongs b1, belongs b2, stage s1, stage s2 WHERE r.rallyno = b1.rallyno AND b1.stageno = s1.stageno AND s1.length_km = s2.length_km AND s2.stageno = b2.stageno AND b2.rallyno = 201 AND r.rallyno <> 201;	List the names of customers who have rented exactly the same movie copy that Robert Butler (rbutler1) has rented, whenever. SELECT c.fname, c.sname FROM customer c, rental r1, rental_copy rc1, rental_copy rc2, rental r2 WHERE c.cust_id = r1.cust_id AND r1.renno = rc1.renno AND rc1.copyno = rc2.copyno AND rc2.renno = r2.renno AND r2.cust_id = 'rbutler1' AND c.cust_id <> 'rbutler1';
C14	List the numbers of users by country and relationship type. Sort the results by country in ascending order. Make sure that the structure of the result table is as below [example given]. SELECT u.country, s.relationship_type, COUNT(u.userno) AS total FROM user u, relationship s WHERE u.userno = s.userno1 GROUP BY u.country, s.relationship_type ORDER BY u.country ASC;	List the numbers of stages by rally name and year. Sort the results by year in ascending order. Make sure that the structure of the result table is as below [example given]. SELECT r.rallyname, b.year, COUNT(b.stageno) AS total FROM rally r, belongs b WHERE r.rallyno = b.rallyno GROUP BY r.rallyname, b.year ORDER BY b.year ASC;	List the numbers of movie copies located in stores by city and status of the copy. Sort the results by city in ascending order. Make sure that the structure of the result table is as below [example given]. SELECT s.city, c.status, COUNT(c.copyno) AS total FROM store s, copy c WHERE c.stono = s.stono GROUP BY s.city, c.status ORDER BY s.city ASC;

	Simple	Semi-complex	Complex (Taipalus et al., 2018)
C15	List the numbers of sent messages by the sender's country and message read status. Disregard senders with less than four sent messages, regardless of the message read status. Sort the results according to the number of messages sent in descending order. SELECT u.country, m.read, COUNT(m.userno_sender) AS total FROM user u, message m WHERE u.userno = m.userno_sender GROUP BY u.country, m.read HAVING COUNT(m.userno_sender) > 3 ORDER BY total DESC;	List the numbers of raced stages by contender number and nationality. Disregard contenders with less than seven raced stages. Sort the results according to the number of stages raced in descending order. SELECT c.contno, c.nationality, COUNT(r.stageno) AS total FROM contender c, race r WHERE c.contno = r.contno GROUP BY c.contno, c.country HAVING COUNT(r.stageno) > 6 ORDER BY total DESC;	List the numbers of movie copies by movie number and movie name. Disregard movies of which there are less than six copies, regardless of the status of the copy. Sort the results according to the number of the copies in descending order. SELECT m.movno, m.mname, COUNT(c.movno) AS total FROM movie m, copy c WHERE m.movno = c.movno GROUP BY m.movno, m.mname HAVING COUNT(c.movno) > 5 ORDER BY total DESC;

PV

TEACHING TIP: A NOTATION FOR PLANNING SQL QUERIES

by

Toni Taipalus 2019

Journal of Information Systems Education, 30(3), 160-166

Journal of Information Systems Education

Volume 30
Issue 3
Summer 2019

Teaching Tip
A Notation for Planning SQL Queries

Toni Taipalus

Recommended Citation: Taipalus, T. (2019). Teaching Tip: A Notation for Planning SQL Queries. *Journal of Information Systems Education,* 30(3), 160-166.

Article Link: http://jise.org/Volume30/n3/JISEv30n3p160.html

Initial Submission:	April 4 2018
Accepted:	13 February 2019
Abstract Posted Online:	5 June 2019
Published:	12 September 2019

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Teaching Tip
A Notation for Planning SQL Queries

Toni Taipalus
University of Jyvaskyla
Faculty of Information Technology
Jyvaskyla, Finland, 40014
toni.taipalus@jyu.fi

ABSTRACT

Structured Query Language (SQL) is still the de facto database query language widely used in industry and taught in almost all university level database courses. The role of SQL is further strengthened by the emergence of NewSQL systems which use SQL as their query language as well as some NoSQL systems, e.g., Cassandra and DynamoDB, which base their query languages on SQL. Even though the syntax of SQL is relatively simple when compared to programming languages, studies suggest that students struggle with simple concepts due to working memory constraints when learning SQL. This teaching tip presents a novel, simple, and intuitive notation for planning more complex SQL queries, which 1) facilitates the learning of SQL by providing students with a big picture of a particular data demand in regard to the database structure and 2) separates the logic of a data demand from the syntax and semantics of SQL, thus alleviating the strain on the student's short-term memory. The notation can also be applied when discussing SQL semantics during the teaching process without focusing on the syntactical nuances of the language.

Keywords: Structured query language (SQL), Query language, Data management, Data visualization, Teaching tip

1. INTRODUCTION

When teaching programming, teachers often emphasize planning before writing, and encourage the use of various techniques, e.g., flowcharts, to plan how the software works. As the software becomes increasingly complex, planning can be supported by design, e.g., by using class diagrams. Various planning techniques that support learning have been proposed for programming (e.g., Hu, Winikoff, and Cranefield, 2012), but SQL has received less attention despite its popularity in both education and industry. The techniques intended for supporting the learning of programming cannot be utilized as is with SQL because of the declarative (i.e., a query is a description of what) and set focused (i.e., a query is difficult or impossible to divide into working subsets) nature of SQL as opposed to the imperative (i.e., a function is a description of how) and step focused (i.e., software operates line-by-line and function-by-function) nature of programming languages such as Java, C#, or Python. These differences make the use of flowcharts unsuitable for planning SQL queries.

The more complex the query is, the more strain it puts on the query writer's short-term memory (e.g., de Jong, 2010, for working memory in general; Smelcel, 1995, for working memory in SQL in particular). Additionally, Ahadi et al. (2016) found that omission errors are among the most common errors when students are learning SQL and proposed that following a systematic procedure and segmenting the question could be the solution for avoiding omission errors. Additionally, even though the syntax of SQL is relatively simple, during the query writing process, the writer must recall SQL keywords with their syntax and semantics, in addition to the database object names, namespaces, and required expressions which, according to Smelcer (1995), often causes strain on the student's short-term memory. Furthermore, Buitendijk (1988) discussed that one of the four major reasons for writing incorrect SQL queries was the complexity of the task. Our work introduces a simple and intuitive notation for planning SQL queries (NPSQ) which is not based on any existing notation. The purpose of the notation is two-fold. First, to assist the student in acquiring the big picture of more complex queries, and second, to separate logic and semantics from syntax, thus alleviating the strain on the student's short-term memory.

The notation can be utilized in any database course that involves SQL. We have used the notation in an introductory database course with approximately 250 to 350 students (depending on the year), mandatory for undergraduate students who major in information systems or computer science, who typically have no previous experience in SQL. We have taught SQL from the SQL standard's perspective as proposed by Randolph (2003). In addition to positive student feedback, several industry professionals have indicated that the notation has proven increasingly useful when planning more and more complex queries.

2. BACKGROUND

In this section, we first define key terms for this work. We then describe our perceptions on how a query writing process takes

place in order to give background on what conceptions have driven the evolution of the notation.

2.1 Terminology

A *data demand* is a natural language representation of what data is needed to which a query writer, e.g., a student, is required to write an equivalent *query* in SQL. When a query is run, the database management system outputs an error message, or a *result table* which contains the rows that satisfy the query. A *query plan* is a picture drawn by a query writer using NPSQ. A query plan is drawn after reading the data demand but before writing the query.

Rows that satisfy a query can be limited in two ways: *joins* and *expressions*. To the extent of our teaching, a student can write a join in one of four methods: using the JOIN predicate, an uncorrelated subquery with IN, a correlated subquery with EXISTS, or with an explicit join condition without a subquery. Not all the methods can be applied for all data demands, and some methods fit more naturally to some data demands. Expressions concern either a column, or groups, which means that the expression is placed either in a WHERE clause, or a HAVING clause, respectively. Concrete examples of some of these methods can be found in Appendix 1, and examples of all methods in Taipalus, Siponen, and Vartiainen (2018).

2.2 The Query Writing Process

Over the last eight years of teaching SQL, we have identified six steps in the query writing process which, in turn, have guided the formulation and usage of this notation. Similar steps or aspects have also been recognized by others (e.g., Casterella and Vijayasarathy, 2013). These steps are, in order: i) which tables are needed to answer the data demand; ii) which columns are needed in the result table; iii) which tables need to be joined; iv) which columns are the joining columns, and is there a need for an outer join; v) which columns are subject to expressions; and vi) is there a need for ordering, grouping, or expressions on groups. These steps can be interpreted as one of the lower level presentations of the model of the query formulation process suggested by Borthick et al. (2001).

3. THE NOTATION

In this section we first discuss the elements of NPSQ from a more theoretical viewpoint and then present practical step-by-step instructions on how to utilize the notation. More examples can be found in Appendix 1. The notation can also be utilized for complex UPDATE and DELETE statements with little or no modifications. Furthermore, the notation may be used with other relationally complete query languages which, however, appear to be scarce.

3.1 The Elements of the Notation

NPSQ does not decree the syntactical elements of the query, e.g., which method should be used when writing joins or how expressions should be written, but only the logic of the query. We have designed the notation for SELECT, FROM, WHERE, ORDER BY, GROUP BY, and HAVING clauses, because these are the most commonly taught data retrieval elements of SQL. Although we present the elements of NPSQ drawn with a computer program, we emphasize that the planning should take place with pen and paper for quickness and convenience. Figure 1 summarizes the notation, and Table 1 presents examples of the SQL equivalents. The elements on the left side correspond to relational algebraic operations (Codd, 1970, 1972):

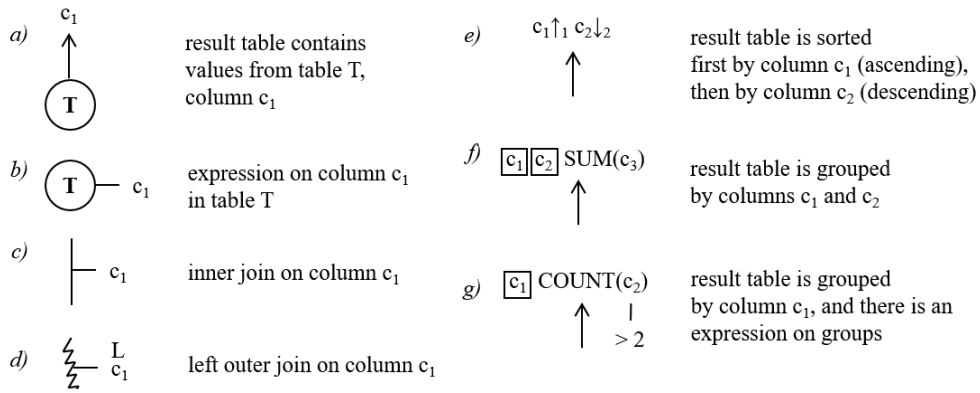


Figure 1. The Elements of the Notation

a) SELECT c_1 FROM <some table> T	*e)* ORDER BY c_1 ASC, c_2 DESC
b) FROM <some table> T WHERE c_1 = <some value>	*f)* SELECT c_1, c_2, SUM(c_3) […] GROUP BY c_1, c_2
c) FROM <some table> T INNER JOIN <some table> S ON (T.c_1 = S.c_1)	*g)* SELECT c_1, COUNT(c_2) […] GROUP BY c_1 HAVING COUNT(c_2) > 2
d) FROM <table> T LEFT OUTER JOIN <some table> S ON (T.c_1 = S.c_1)	

Table 1. The Corresponding SQL Concepts for Each Element of the Notation

projection (SELECT), restriction (WHERE), join (INNER JOIN), and intersection (OUTER JOIN). The elements on the right side correspond to SQL clauses: sorting (ORDER BY), grouping (GROUP BY), and expressions on groups (HAVING).

While we designed the elements of the notation around the six steps of the query writing process discussed in Section 2.2, the structure of a query plan is inspired by the query trees used as input and output by the query processing components of different database management systems. A query plan can also be understood as a graph with nodes (tables), edges (joins), and properties (joining columns and expressions) of both. Furthermore, a query plan is a kind of tree in which the root node is the table from which columns are projected into the result table. However, a tree can have multiple root nodes, if the result table contains columns from more than one table.

Tables should be represented not by table names but by short aliases for brevity and convenience. In a case such as a self-join when the same table must be presented more than once, different aliases should be considered, e.g., T_1 and T_2 for table *T*. If an expression is complex, or the expression repeated with different values for different tables, more precise notation can be used, e.g., c_1 = *'New York'* instead of c_1. If a join is complex, e.g., based on an aggregate function, or if a quantified comparison operator such as ALL is used, it can be presented as a property of the corresponding edge.

If the query is written with subqueries, the distance from the root node(s) represents the depth of a query; the root nodes represent the main SELECT clause, the nodes on the next level of the tree represent first level subqueries, the nodes on the level below that represent second level subqueries etc. A case of negated existential quantifier (¬∃) can be formulated with either left or right outer join, with a subquery using NOT IN or NOT EXISTS, or with ALL. In the former case, letters L or R can be used to illustrate the type of the outer join, as demonstrated in Figure 1 (d). If NATURAL JOIN or CROSS JOIN is used, the property of the edge can be omitted.

In the scope of our course, we teach only strict grouping. In practice, this means that if an aggregate function is used in the main SELECT clause with a grouping column, the result table must be grouped by all grouping columns, and only the grouping columns for the query to be syntactically correct, as opposed to the optional feature T301 (ISO/IEC, 2016). This grouping convention can be observed in Figure 1 and Table 1 (f, g).

3.2 Practical Examples

In order to demonstrate the usage of the notation in practice, and to demonstrate corresponding SQL clauses with complete examples, we utilize two data demands presented by Taipalus, Siponen, and Vartiainen (2018). We present the query plan formulation in six steps, which correspond to the steps presented in Section 2.2. Additionally, we present the corresponding SQL queries formulated in six steps. It is worth noting that we do not necessarily write the queries in the order presented in Tables 2 and 3, and the tables are presented merely for illustrative purposes. Refer to Appendix 2 for the database schema and business domain.

For Figure 2 and Table 2, consider the data demand "List the names of actors who have acted a role as himself or herself. Sort the results according to surname and then according to first name, both in ascending order."

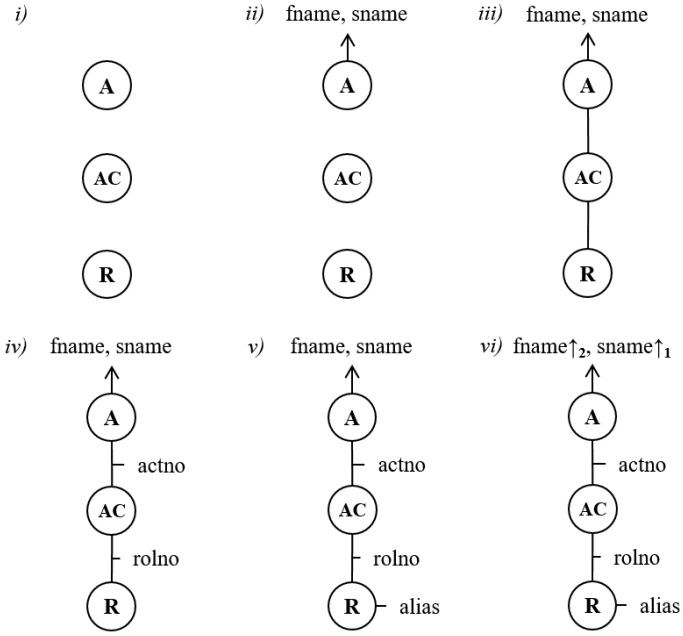


Figure 2. The Iterative Process of a Basic Query Plan Formulation - Table Abbreviations A, AC, and R Stand for Actor, Acts, and Role, Respectively

i)	*ii)*	*iii)*
SELECT FROM actor a, acts ac, role r	SELECT a.fname, a.sname FROM actor a, acts ac, role r	SELECT a.fname, a.sname FROM actor a, acts ac, role r WHERE a. = ac. AND ac. = r.
iv)	*v)*	*vi)*
SELECT a.fname, a.sname FROM actor a, acts ac, role r WHERE a.actno = ac.actno AND ac.rolno = r.rolno	SELECT a.fname, a.sname FROM actor a, acts ac, role r WHERE a.actno = ac.actno AND ac.rolno = r.rolno AND r.alias IN ('Himself', 'Herself')	SELECT a.fname, a.sname FROM actor a, acts ac, role r WHERE a.actno = ac.actno AND ac.rolno = r.rolno AND r.alias IN ('Himself', 'Herself') ORDER BY a.sname ASC, a.fname ASC;

Table 2. The Corresponding SQL Statements for Each Step Presented in Figure 2

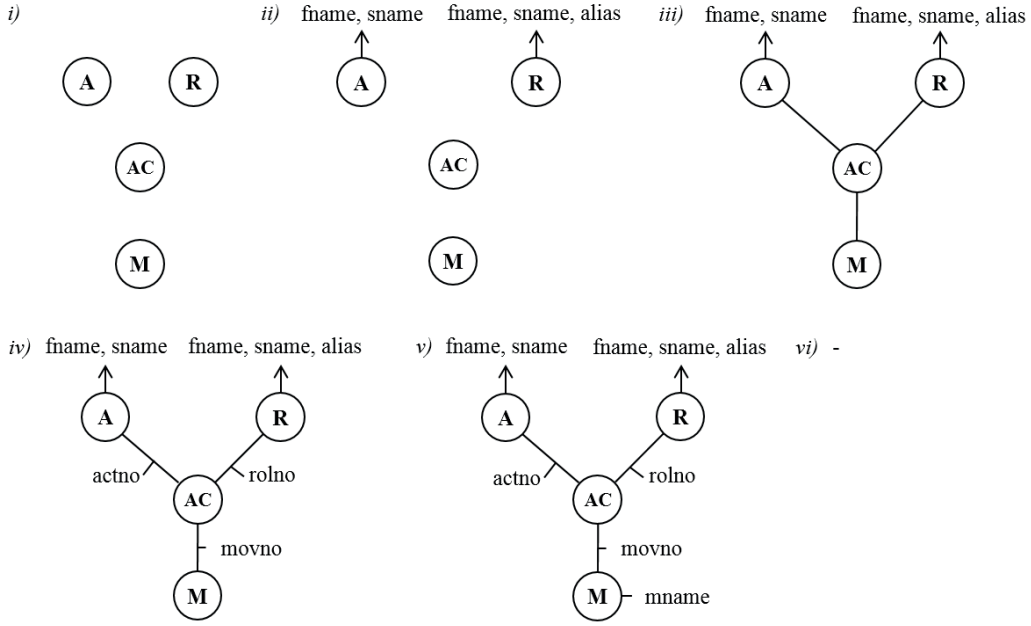


Figure 3. The Iterative Process of a More Complex Query Plan Formulation – Table Abbreviations A, R, AC, and M Stand for Actor, Role, Acts, and Movie, Respectively

i)	*ii)*	*iii)*
SELECT FROM actor a, role r, movie m, acts ac	SELECT a.fname, a.sname, r.fname, r.sname, r.alias FROM actor a, role r, movie m, acts ac	SELECT a.fname, a.sname, r.fname, r.sname, r.alias FROM actor a, role r, movie m, acts ac WHERE a. = ac. AND r. = ac. AND ac. = m.
iv)	*v)*	*vi)*
SELECT a.fname, a.sname, r.fname, r.sname, r.alias FROM actor a, role r, movie m, acts ac WHERE a.actno = ac.actno AND r.rolno = ac.rolno AND ac.movno = m.movno	SELECT a.fname, a.sname, r.fname, r.sname, r.alias FROM actor a, role r, movie m, acts ac WHERE a.actno = ac.actno AND r.rolno = ac.rolno AND ac.movno = m.movno AND m.mname = 'Physics 101';	*(nothing to add)*

Table 3. The Corresponding SQL Statements for Each Step Presented in Figure 3

For Figure 3 and Table 3, consider the data demand "List the names of actors who have acted in the movie *Physics 101*, and list the names of the roles they have played in that movie." For the query plans in Figures 2 and 3, notice how the distances of the nodes from the root node would represent the level of the subqueries.

4. IMPLICATIONS FOR TEACHING

We have identified fours ways of using the notation in teaching. First, when SQL is first taught in the course lectures, query plans can be utilized to explain the logic behind each data demand before writing the query. In our experience, the notation is so simple and intuitive that it can be explained simultaneously to drawing the first query plan. During the drawing process, the teacher can ask students the questions listed in Section 2.2 and draw the plan gradually. The students can be encouraged to plan all queries before writing them for lab assignments or in the final examination.

Second, as the notation separates logic and semantics from syntax, the students can ask the teachers whether their query is planned correctly without focusing on the syntactical aspects of the query. Subsequently, the teachers can point out possible logical errors in the plan, asking questions such as "this plan answers to a different data demand, can you tell me what it is?" This in turn informs the students whether they have understood the data demand and can then focus on the syntax. For example, if we join STORE with EMPLOYEE (see Appendix 2) using *stono*, the result table contains stores with at least one employee working in them. However, the teacher can draw a query plan in which the tables are joined using *empno* and ask the students to explain what the data demand is.

Third, in addition to writing queries in the final examination, query plans may be required. Although this requirement means that the students need to learn an additional notation for the final examination, it might eliminate some errors caused by carelessness, such as missing expressions or ordering from the queries, in addition to forcing the student to reflect on the logic behind the data demand before starting the query writing process.

Fourth, the logic behind joining different tables by different columns in a specific database domain can be practiced in pairs: one student draws the query plan and another student writes the query based on that plan. The exercise can be made more difficult if only the student drawing the query plan is aware of the data demand. We are eager to construct a research setting to see if scientific evidence supports our positive experiences with the notation.

5. CONCLUSION

In this paper, we presented a simple notation for planning complex SQL queries to separate the logic of a data retrieval task from the syntax of SQL and to alleviate the strain a task puts on the query writer's short-term memory. We hope that the paper will encourage other educators to use the notation in their database courses to facilitate the teaching of SQL and to help formulate, understand, and teach more complex queries to mimic the students' future work environments, whether those environments are in the domain of business analytics or software engineering.

6. REFERENCES

Ahadi, A., Prior, J., Behbood, V., & Lister, R. (2016). Students' Semantic Mistakes in Writing Seven Different Types of SQL Queries. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*, 272–277.

Borthick, A. F., Bowen, P. L., Jones, D. R., & Tse, M. H. K. (2001). The Effects of Information Request Ambiguity and Construct Incongruence on Query Development. *Decision Support Systems*, 32, 3-25.

Buitendijk, R. B. (1988). Logical Errors in Database SQL Retrieval Queries. *Computer Science in Economics and Management*, 1(2), 79-96.

Casterella, G. I. & Vijayasarathy, L. (2013). An Experimental Investigation of Complexity in Database Query Formulation Tasks. *Journal of Information Systems Education*, 24(3), 211-222.

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6), 377–87.

Codd, E. F. (1972). Relational Completeness of Data Base Sublanguages. *Data Base Systems (Courant Computer Science Symposium 6)*, Prentice-Hall.

de Jong, T. (2010). Cognitive Load Theory, Educational Research, and Instructional Design: Some Food for Thought. *Instructional Science*, 38, 105–134.

Hu, M., Winikoff, M., & Cranefield, S. (2012). Teaching Novice Programming using Goals and Plans in a Visual Notation. *Proceedings of the Fourteenth Australasian Computing Education Conference - Volume 123 (ACE '12)*, Darlinghurst, Australia, 43-52.

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

Randolph, G. B. (2003). The Forest and the Trees: Using Oracle and SQL Server Together to Teach ANSI-Standard SQL. *Proceedings of the 4th Conference on Information Technology Curriculum (CITC4)*, 234–236.

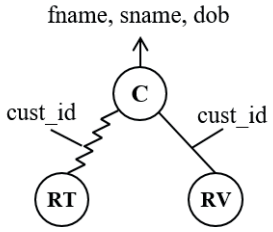
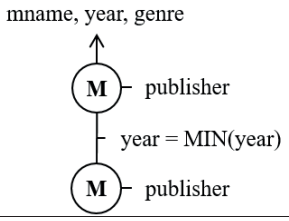
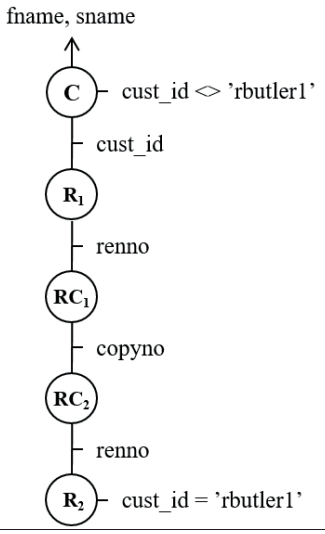
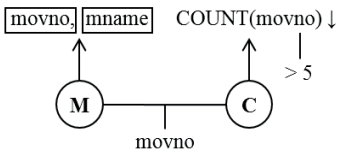
Smelcer, J. B. (1995). User Errors in Database Query Composition. *International Journal of Human-Computer Studies*, 42(4), 353–381.

Taipalus, T., Siponen, M., & Vartiainen, T. (2018). Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education,* 18(3), Article 15.

AUTHOR BIOGRAPHY

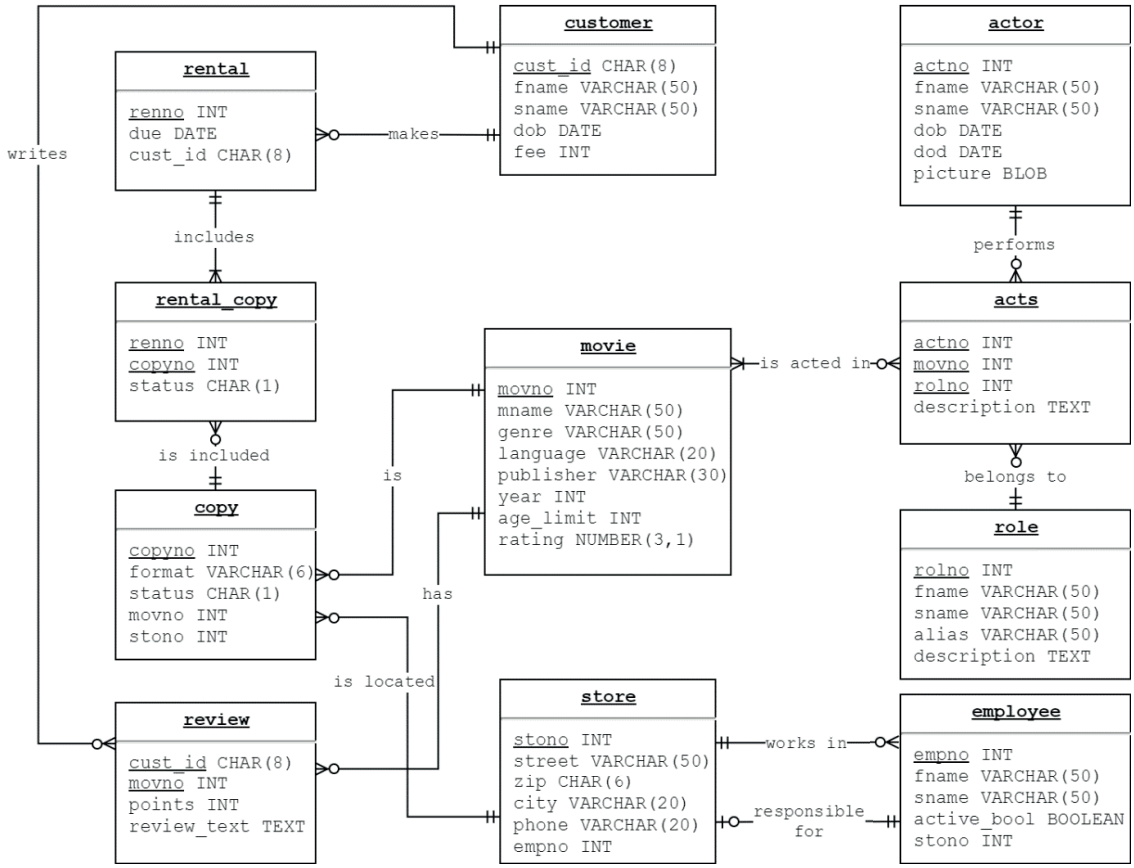
Toni Taipalus is a teacher at the University of Jyvaskyla. He teaches databases, data management, application programming, and system development. His research interests are in the pedagogical aspects of query languages, data models, and agile software development.

APPENDIX 1: EXAMPLE QUERY PLANS

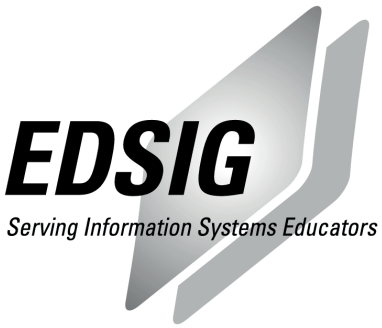
Query (Taipalus, Siponen, and Vartiainen, 2018)	Query plan
SELECT c.fname, c.sname, c.dob FROM customer c WHERE NOT EXISTS (SELECT * FROM rental rt WHERE c.cust_id = rt.cust_id) AND EXISTS (SELECT * FROM review rv WHERE c.cust_id = rv.cust_id);	
SELECT mname, year, genre FROM movie WHERE publisher ='Goldeneye BC' AND year = (SELECT MIN(year) FROM movie WHERE publisher ='Goldeneye BC');	
SELECT c.fname, c.sname FROM customer c, rental r1, rental_copy rc1, rental_copy rc2, rental r2 WHERE c.cust_id = r1.cust_id AND r1.renno = rc1.renno AND rc1.copyno = rc2.copyno AND rc2.renno = r2.renno AND r2.cust_id = 'rbutler1' AND c.cust_id <> 'rbutler1';	
SELECT m.movno, m.mname, COUNT(c.movno) AS total FROM movie m, copy c WHERE m.movno = c.movno GROUP BY m.movno, m.mname HAVING COUNT (c.movno) > 5 ORDER BY total DESC;	

APPENDIX 2: THE DATABASE SCHEMA

This appendix contains a database schema (Taipalus, Siponen, and Vartiainen, 2018) to be used in conjunction with the examples in Sections 3.2 and 4 and Appendix 1.



Information Systems & Computing
Academic Professionals



STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 2574-3872