Journal of Information Systems Education

Volume 30
Issue 3
Summer 2019

Query Structure and Data Model Mapping Errors in Information Retrieval Tasks

Gretchen I. Casterella and Leo Vijayasarathy

Recommended Citation: 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.

Article Link: http://jise.org/Volume30/n3/JISEv30n3p178.html

Initial Submission:	25 May 2018
Accepted:	20 February 2019
Abstract Posted Online:	5 June 2019
Published:	12 September 2019

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Query Structure and Data Model Mapping Errors in Information Retrieval Tasks

Gretchen I. Casterella
Department of Accountancy & Business Law
Cameron School of Business
University of North Carolina
Wilmington, NC 28403, USA
casterellag@uncw.edu

Leo Vijayasarathy
Computer Information Systems Department
College of Business
Colorado State University
Fort Collins, CO 80523, USA
Leo.Vijayasarathy@colostate.edu

ABSTRACT

SQL query writing is a challenging task for novices, even after considerable training. Query writing is a programming task and a translation task where the writer must translate a user's request for information into code that conforms to the structure, constraints, and syntax of an SQL SELECT statement and that references specific tables and columns from a database. This paper investigates the impact of two instructional interventions on query errors under conditions of low and high query complexity. Data was collected from an experimental study of 63 undergraduate students nearing completion of a 15-week database course. Our analysis reveals specific areas of query writing where each of the interventions helped, and hindered, task performance. We discuss the implications of these findings for improving SQL training and for future research on SQL training effectiveness.

Keywords: Information retrieval, Relational database, Structured query language (SQL), Human-computer interaction (HCI)

1. INTRODUCTION

Structured Query Language (SQL) has been the *de facto* programming standard for accessing data in relational databases for over three decades (Allen and March, 2006). A recent survey of data scientists found SQL to be the third most commonly used data analysis tool (Kaggle, 2017), and many of the languages for accessing NO-SQL databases also incorporate SQL (Soat, 2014). We expect the demand for SQL skills to persist and grow, which in turn reinforces its importance in university database courses (Topi et al., 2010; Bell, Mills, and Fadel, 2013).

A primary SQL skill is writing *ad hoc* queries that pull data from multiple tables in a database in response to an information request. However, novices struggle to acquire these query-writing skills (e.g., Bowen, O'Farrell, and Rohde, 2009; Allen and Parsons, 2010; Casterella and Vijayasarathy, 2013). One of the learning challenges relates to the gap between the data that the user wants to see (the query result) and the way the data is organized in the database. The query writer must be able to think in *sets* – not in *steps* as with other programming languages – to determine how to manipulate detailed data scattered across multiple tables to obtain the desired data set, which may be at a summarized level. Added to this is the fact that the initial request for information is typically expressed in the end user's natural language, rife with ambiguity, and the query writer must transform that request into a highly-structured SQL statement that references specific tables and columns in a data model, which itself is often represented in diagrammatic form (Borthick et al., 2001). Clearly, there are significant cognitive obstacles learners must overcome to be competent query writers.

We are interested in the impact of different training interventions on query writing performance, and how those interventions can be improved. Vijayasarathy and Casterella (2016), hereafter called VC2016, investigated the impact of two training interventions on novices' overall query task performance. The training interventions were designed to reduce specific sources of task complexity and thus promote successful query writing performance. One intervention changed the way the information retrieval task was presented to the novice (the request language treatment), and the other intervention added a planning task prior to the query coding task (the planning task treatment). Data was collected from 63 university undergraduate students who had completed several weeks of training on SQL queries. VC2016 found that the request language treatment improved task performance but only for less complex queries, while the planning task treatment improved task performance only for more complex queries. While the findings provide some support for the training interventions, the interaction effects were not entirely as expected. Further, VC2016's focus on overall query writing performance (e.g., overall query accuracy, time to complete task), while useful, does not provide the fine-grained information needed to improve the training interventions.

To gain a deeper understanding of how the experimental treatments helped or hindered novices, in this paper we analyze the VC2016 data set with a focus on *specific types of error* that directly relate to the treatments. The request language and planning task treatments were designed to reduce query-writing complexity by making it easier for novices to: 1) map words from the information request to tables and columns in the data model and 2) identify which clauses were needed in the query solution. We define two corresponding types of error – data model (DM) and query structure (QS) – and we examine the effect of the treatments on each error type. In addition, since prior research indicates that certain parts of a SELECT statement are particularly difficult (Reisner, 1981, Greene et al., 1990), we analyze errors on a clause-by-clause basis. Finally, to examine the impact of the planning task on final query errors, we compare errors in the planning task to those in the final queries.

We find that many of the errors in VC2016 stemmed from overlooking adjectives in the natural-language information request, a weak understanding of the data model, or both. The request language treatment – where information requests were presented in pseudo-SQL rather than the more ambiguous "Manager-English" – helped reduce these errors, particularly errors of omitted filtering conditions in the WHERE clause. The planning task treatment – where students made notes in a query template prior to coding – helped reduce errors in the SELECT, FROM, and ORDER BY clause, but not with the WHERE, GROUP BY, and HAVING clauses. We discuss the implications of these findings for future research on SQL training.

2. QUERY WRITING TASK COMPLEXITY

In practice, the user requesting information from a database is often not the same individual who writes the query to retrieve that information. The query writer must bridge the distance between the source language (natural language) and the target language (SQL) to produce a query that executes and returns the requested information from the database. The cognitive complexity of a query-writing task is determined by factors inherent in the task itself that increase the mental effort required (Batra, 2007). This is not the same as the perceived difficulty of a task, or the cognitive load the writer experiences, which will vary across individuals depending, for example, on their knowledge of the domain and their SQL expertise (Batra, 2007). The cognitive complexity of writing SQL queries has been discussed extensively in prior research (e.g., Borthick et al., 2001; Bowen, O'Farrell, and Rohde, 2009; Allen and Parsons, 2010). We use the following two examples to illustrate the complexities.

2.1 Example of a Less Complex Query Formulation Task

Consider the following information request and corresponding query solution.

Information request:	Query solution:
"*Who placed online orders after December 23, 2015, and what was the subtotal for each of those orders?*"	`SELECT FirstName, LastName, SubTotal FROM Person INNER JOIN SalesOrderHeader ON PersonPK = CustomerFK WHERE OnlineOrderFlag = 1 AND OrderDate > '12/23/2015'`

The query writer must transform the natural-language request into the SQL code based on a given data model, which in this case is for a sales order database whose schema is shown in Appendix 1. This transformation requires an understanding of the database design as well as of SQL syntax and the specific rules for how to structure SQL queries using at most six clauses (SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY), each with a specific purpose. Two main subtasks involved in this transformation are query structure generation and data model mapping (Reisner, 1977). Query structure generation is the process of generating the basic outline or template of the query solution, while data model mapping is identifying which columns and tables from the data model to include in the outline or template.

In this example, query structure generation would produce the following solution outline: **SELECT** ____ **FROM** ____ **INNER JOIN** ____ **ON** ____ **WHERE** ____ **AND** ____. The query writer must determine that the FROM clause needs to join two tables (hence the INNER JOIN and ON keywords), that a WHERE clause is needed with two conjunctive conditions (hence the AND keyword), and that the GROUP BY, HAVING, and ORDER BY clauses are not needed. Recognizing the need for a WHERE clause involves recognizing the need to restrict the rows (in this case, orders) in the result set and, in this case, we would expect that the phrase "after December 23, 2015," would be a clue that a filtering condition is needed. Recognizing the need for a compound condition is more difficult, as its clue is just one word, "online." Thus, even though the query solution is rather short, there are several sources of complexity that could impact the structure generation portion of the task.

Table 1 shows the data model mappings needed to fill in parts of the solution outline. These mappings are the connections the query writer must make between elements in the original information request and elements in the data model. The right-hand column in Table 1 highlights the mappings from the information request to the empty "slots" in the query structure outline that are needed. The bolded elements in the right column are the data model elements (i.e., tables and columns).

Query Structure	Data Model Mappings to Complete the Query Structure
SELECT ____	"who" → **LastName**, **FirstName** "subtotal" → **SubTotal**
FROM ____ **INNER JOIN** **____ ON ____**	"who placed" → **Person** "orders" → **SalesOrderHeader** Ø → **PersonPK** = **CustomerFK**
WHERE ____ **AND ____**	"online" → **OnlineOrderFlag** = 1 "after December 23, 2015" → **OrderDate** > '12/23/2015'

Table 1. Data Model Mappings for a Less Complex Query

For the SELECT clause, the writer must map terms in the request to the appropriate columns that the user wants to see in the query result. This mapping may be complex when there is ambiguity in the information request, such as if the user asked for the "total" of each order (e.g., including shipping, tax, etc.) rather than the "subtotal" (which has an exact column match). Mapping the data model elements to the FROM clause requires identifying which tables are needed (in this case, Person and SalesOrderHeader) and the columns that link the two tables together. There is no word in the information request that indicates what this linking or join column is (i.e., Ø → **PersonPK** = **CustomerFK**). In the WHERE clause, the phrase "after December 23, 2015" is mapped to a search condition based on the OrderDate column. For the "online orders" phrase, the writer needs to: (1) notice the single word "online," (2) know that the SalesOrderHeader table includes online and not-online orders, and (3) recognize that the OnlineOrderFlag column will indicate whether an order is online with a 1 or 0 value. Even with a relatively short query, there are places where data model mapping is a source of complexity.

2.2 Example of a More Complex Query Formulation Task

Consider another information request and corresponding query solution for the same sales order database:

Information request:	Query solution:
"For each product subcategory with an average list price above $100, show the name of the subcategory, the name of the parent category, the average price of products in the subcategory, and the number of different product colors in the subcategory. Sort the results alphabetically by the category name and the subcategory name."	SELECT ProductSubCategoryName, ProductCategoryName, AVG(ListPrice), COUNT(DISTINCT Color) FROM ProductCategory INNER JOIN ProductSubcategory ON ProductCategoryPK = ProductCategoryFK INNER JOIN Product ON ProductSubCategoryPK = ProductSubcategoryFK GROUP BY ProductCategoryName, ProductSubCategoryName HAVING AVG(ListPrice) > 100 ORDER BY ProductCategoryName, ProductSubCategoryName

Consider the query's structure: **SELECT ____ FROM ____ INNER JOIN ____ ON ____ INNER JOIN ____ ON ____ GROUP BY ____ HAVING ____ ORDER BY ____**. The query writer must recognize that two joins are needed in the FROM clause, that the WHERE clause is not needed (although "average list price above $100" sounds like a filtering condition that might be specified in a WHERE clause), and that three other clauses are needed (GROUP BY, HAVING, and ORDER BY). The information request does not have clear clues that tell the query writer to include a GROUP BY or HAVING clause. The request does include the phrase, "for each product subcategory," which implies a need for grouping if the writer recognizes that rows in the merged table after the join will be at the individual product level rather than at the subcategory level, but this is difficult for novices to recognize (Bowen, O'Farrell, and Rohde, 2006). The challenge with table joins and grouping is that there is a mismatch between data model representation and the requirements of the information retrieval task. The data model shows each table's structure (i.e., its columns) and lines represent the relationships between tables, but the task requires the writer to think about manipulating the specific rows in the tables, joining them, and rearranging them into subsets (Reisner, 1977). Thus, performance is expected to decline for queries that involve grouping.

There are also data-model mapping challenges in this example. Table 2 shows the transformations needed to complete the query structure outline. The data-model mapping elements are shown in bolded text in the right-hand column. Note that many transformations do not have a counterpart in the information request.

Query Structure	Transformations to Complete the Query Structure
SELECT ____	"the name of the subcategory" → **ProductSubCategoryName** "the name of the parent category" → **ProductCategoryName** "the average price of products in the subcategory" → AVG(**ListPrice**) "the number of different product colors in the subcategory" → COUNT(DISTINCT **Color**)
FROM____ **INNER JOIN** **____ ON ____** **INNER JOIN** **____ ON ____**	"For each product subcategory" → **ProductSubcategory** Ø → **ProductCategory** Ø →**ProductCategoryPK** = **ProductCategoryFK** Ø →**Product** Ø → **ProductSubCategoryPK** = **ProductSubcategoryFK**
GROUP BY ____	Ø → **ProductCategoryName** Ø → **ProductSubCategoryName**
HAVING ____	"an average list price above $100" → AVG(**ListPrice**) > 100
ORDER BY ____	"sort results alphabetically by the category name" → **ProductCategoryName** "and the subcategory name" → **ProductSubCategoryName**

Table 2. Data Model Mappings for a More Complex Query

These two examples illustrate some of the difficulties a novice query writer faces. While awareness of these challenges is important when teaching SQL, we are interested in teaching resources that target the early stages of SQL skill acquisition so that novices are not in a situation where they have to be competent at many things before they can accomplish anything. Toward that end, we next summarize the VC2016 study which included two training interventions to improve novices' query writing task performance.

3. SUMMARY OF VC2016

The VC2016 study examined the impact of two training interventions on query writing. One was the information request language, which was based on prior research showing improved query writing performance when the information request was written in "pseudo-SQL" rather than "manager-English" (Borthick et al., 2001; Casterella and Vijayasarathy, 2013). Pseudo-SQL requests used terms that more closely matched the SELECT statement solution than the manager-English requests. Pseudo-SQL requests were expected to improve task performance because they provided clearer clues about which data model elements were needed and which SELECT statement clauses were needed when compared to the less-precise manager-English requests.

The second intervention was to introduce a planning task prior to coding, where participants were asked to take notes in a query template. The template outlined the six possible clauses to include in the query solution, along with questions to prompt participants to consider which clauses were needed and what to specify in each. The template was expected to improve task performance by providing reminders about the order in which clauses must be written and the purpose of each clause. Participants had to use the template before they could begin writing queries in the editor window. The notes written in the template were visible throughout the query writing task so that it was, in essence, an intermediate solution that was expected to reduce the participants' cognitive load.

VC2016 conducted a 2 x 2 x 2 repeated measures experiment to assess the impact of three factors – query complexity, request language, and planning task – on overall query task performance. Query complexity was either low or high. Low complexity queries were about half the length and less than half the difficulty of the high complexity queries. The four low complexity query solutions were similar to the first example discussed in the prior section, with the same structural characteristics shown in Table 1. The four high complexity queries were similar to the second example discussed earlier, with the same structural characteristics shown in Table 2. Request language referred to the task presentation, where the information request was presented in either pseudo-SQL or manager-English. Planning task referred to whether the participant was given a query planning task prior to coding.

Sixty-three undergraduate students in an introductory database course completed the study, which consisted of eight query tasks, one for each condition. VC2016 found main effects for query complexity and request language, such that overall query accuracy was higher for less complex queries and for pseudo-SQL requests (Note: Two control variables, GPA and prior SQL experience, also had significant positive effects on overall query accuracy, as expected). VC also found an interaction effect for complexity and request language, but it was opposite of their hypothesis – the pseudo-SQL requests improved overall query accuracy for less complex queries but not for more complex queries. They found that the planning task improved overall query accuracy for more complex queries but not for less complex queries. While the VC2016 findings provide some support for their training interventions, it is not clear why the planning task had no main effect on query accuracy or why the pseudo-SQL requests were no more helpful than manager-English requests for complex queries. The rest of the paper describes several new analyses of the VC2016 data to uncover specific trouble spots in the participants' queries.

4. A CLOSER LOOK AT QUERY ERRORS

The main dependent variable of interest in VC2016 was overall query accuracy, calculated as the ratio of the number of actual correct elements to the number of required correct elements in each query solution. This accuracy score included all coding elements in a query, including punctuation, function names, mathematical operators, etc. However, not all of these elements are directly related to the experimental treatments. Thus, we conducted a finer-grained analysis of two specific types of error – query structure (QS) errors and data model (DM) errors. **Query structure** (QS) errors are missing or incorrectly specified SQL keywords that are needed to begin each clause, indicate table joins, and/or indicate compound search conditions (see the examples in Tables 1 & 2). **Data model** (DM) errors are missing or incorrectly specified table names or column names from the data model that are needed in the query (see the examples in Tables 1 & 2). We expect the treatments to directly impact these two error types, as described below.

4.1 Hypotheses

With respect to the request language treatment, the pseudo-SQL requests include explicit table and column names, and thus are expected to reduce DM errors. The pseudo-SQL requests also include terms that directly match the keywords for certain clauses in the SELECT statement and so should reduce the cognitive load needed for generating the query's structure, and reduce the QS errors. On the other hand, if a request is presented in manager-English, more cross-referencing between the request, the data model, and the query editor is needed, which may lead to more DM errors. Without the explicit clues indicating the clauses needed in the query, we expect more QS errors as well. In the VC2016 study, pseudo-SQL had a main effect on overall query accuracy. Here, we expect that it will have a main effect on both types of error.

H1: Query writers given pseudo-SQL requests will have lower DM and QS error rates than writers given manager-English requests.

The interaction between query complexity and request language on overall query accuracy was not as expected in the VC2016 study. Participants had several weeks of practice writing queries prior to the study, so VC2016 expected that the pseudo-SQL assistance would not be as useful for the less complex queries as for the more complex queries. However, the opposite was found. In our analysis, we investigate whether our more focused dependent variables detect the expected difference. We have no *a priori* reason to believe that the pseudo-SQL requests would impact one error type more/less than the other. Thus, our second hypothesis is as follows.

H2: The impact of pseudo-SQL requests on QS and DM error rates will be greater when query complexity is high versus low.

With respect to the planning task treatment, the VC2016 study found that completing the planning task prior to coding did not significantly improve overall query accuracy, except when query complexity was high. The planning template prompts the writer to consider which clauses are needed in the query solution (a QS issue) and indicates what should be specified within each clause (a DM issue). With our dependent variables narrowly focused on QS and DM errors, we expect the planning task will have a significant main effect on both error rates.

H3: Query writers who complete the planning task prior to coding will have lower DM and QS error rates than writers who have no planning task prior to coding.

As with the request language intervention, we expect that for less complex queries, the benefits of the planning task will be negligible but as complexity increases, the planning task will be more helpful. The prompts in the planning task were designed to engage the participants in task analysis and planning prior to coding. Taking notes in the template would also allow writers to "offload" some of their cognitive load to this external artifact and free up cognitive resources for the coding part of the task, which we expected to be particularly helpful for complex queries that involved aggregation and grouping. Thus, the hypothesis is:

H4: The impact of the planning task on QS and DM error rates will be greater when query complexity is high versus low.

4.2 Results of Hypothesis Testing

To measure QS and DM errors, we first re-examined each of the correct query solutions and counted the number of elements in the queries related to QS and DM. All other elements were considered "other" and not included in our hypothesis testing or subsequent analyses. Appendix 2 shows the breakdown of the eight query solutions into QS, DM, and other elements. The "other" category accounts for one-third to one-half of each solution, and is not included in our analysis. The next step was to re-evaluate the participants' queries for each task to identify the corresponding number of QS and DM errors that were present. Because the maximum possible number of errors differed depending on complexity level, we transformed each error count into a rate by dividing by the maximum number of errors possible for that task. Thus, the two dependent variables were defined as follows. The **DM error rate** is the proportion of required table and column references that were missing or incorrect in each participant's final query solution, and the **QS error rate** is the proportion of required query structure keywords that were missing or incorrect in each participant's final query solution.

We analyzed the final queries from the same 63 participants in the VC2016 study using these new DM and QS error rates. To test our hypotheses, we used repeated-measures MANOVA (Hair et al., 2010). One of the 63 participants had error rates in excess of the threshold for multivariate outliers, and was dropped for this analysis. Further, since one of the eight experimental groups exhibited multi-collinearity between the dependent variables ($r > 0.80$), only the univariate results from the MANOVA procedure are reported and discussed. Table 3 shows the mean error rate by error type and experimental treatment, and Table 4 shows the within-subjects effects of our treatments on the two types of error.

			Data-Model Mapping (DM) Error Rate	**Query Structure (QS) Error Rate**
Low Complexity	Pseudo-SQL	No Planning	2.87	2.69
		Planning	3.76	2.33
	Manager-English	No Planning	8.42	6.99
		Planning	11.65	11.47
High Complexity	Pseudo-SQL	No Planning	17.34	14.68
		Planning	11.52	11.13
	Manager-English	No Planning	16.43	16.77
		Planning	12.50	14.19
Low Complexity			6.68	5.87
High Complexity			14.45	14.19
Pseudo-SQL			8.87	7.71
Manager-English			12.25	12.36
No Template			11.27	10.28
Template			9.86	9.78
Overall			10.56	10.03

Table 3. Mean Error Rate (%) by Error Type and Experimental Treatment

Source	Error Type	F	Sig.
Query Complexity	Data-Model (DM)	58.81	0.000
	Query Structure (QS)	35.88	0.000
Request Language	Data-Model (DM)	12.50	0.001
	Query Structure (QS)	27.84	0.000
Planning task	Data-Model (DM)	4.29	0.043
	Query Structure (QS)	0.47	0.495
SQL Proficiency	Data-Model (DM)	4.97	0.030
	Query Structure (QS)	12.82	0.001
GPA	Data-Model (DM)	12.35	0.001
	Query Structure (QS)	10.75	0.002
Query Complexity * Request Language	Data-Model (DM)	15.21	0.000
	Query Structure (QS)	6.39	0.014
Query Complexity * Planning task	Data-Model (DM)	21.74	0.000
	Query Structure (QS)	8.86	0.004

Table 4. Within-Subject Effects on Error Rates (%)
Note: For SQL Proficiency and GPA, the effects are between subjects

Hypothesis H1 is supported. Request language had a significant main effect on both error rates. Participants had significantly lower DM error rates and lower QS error rates when given pseudo-SQL requests rather than manager-English requests.

With respect to H2, the interaction effect of complexity and request language on error rates was significant, such that pseudo-SQL requests reduced both error rates for less complex queries but not for more complex queries. This effect is in the opposite direction of our hypothesis. As shown in Figure 1, participants with pseudo-SQL requests had much lower error rates than those with manager-English requests when complexity was low, but the request language had little impact when complexity was high. This is consistent with VC2016's unexpected finding. Our findings, however, shed some light on this discrepancy. When query complexity was high, pseudo-SQL requests provided some benefit in lowering QS error rates, but no benefit with respect to DM error rates. We explore this finding further in the next section.

Hypothesis H3 is partially supported. The planning task significantly reduced DM errors, but not QS errors. This provides some insight into the non-significant main effect in the VC2016 study where overall accuracy was not improved with the planning task. In part, this was because the planning task did not significantly reduce QS errors. This is surprising because the planning task focuses heavily on which clauses might be needed and so might be expected to reduce QS errors more so than DM errors. We explore this finding further in the next section.

H4 is supported. The interaction effect of planning task and query complexity was significant for both error rates, and in the expected direction. As shown in Figure 2, the planning task clearly helped reduce both QS and DM error rates for more complex queries, but actually increased both error rates for less complex queries.

Query complexity had a significant main effect on both error types – we did not have a hypothesis for this as is it has been such a consistent finding that complexity reduces performance across many studies of query writing. We also note that individual differences have a significant impact on error rates, such that participants with higher GPAs and higher self-reported SQL proficiency had lower error rates.

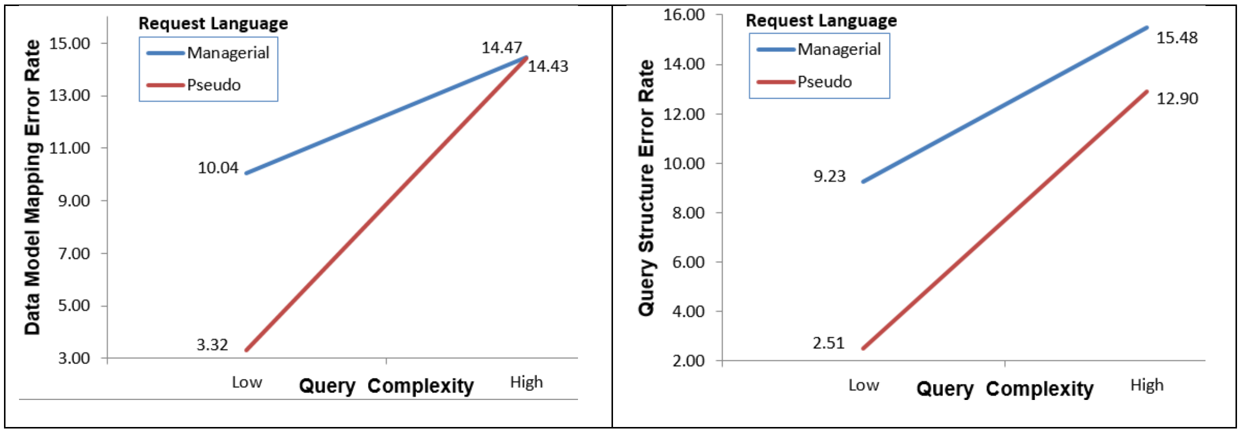


Figure 1. Interaction Effect between Query Complexity and Request Language on Data Model (DM) and Query Structure (QS) Error Rates

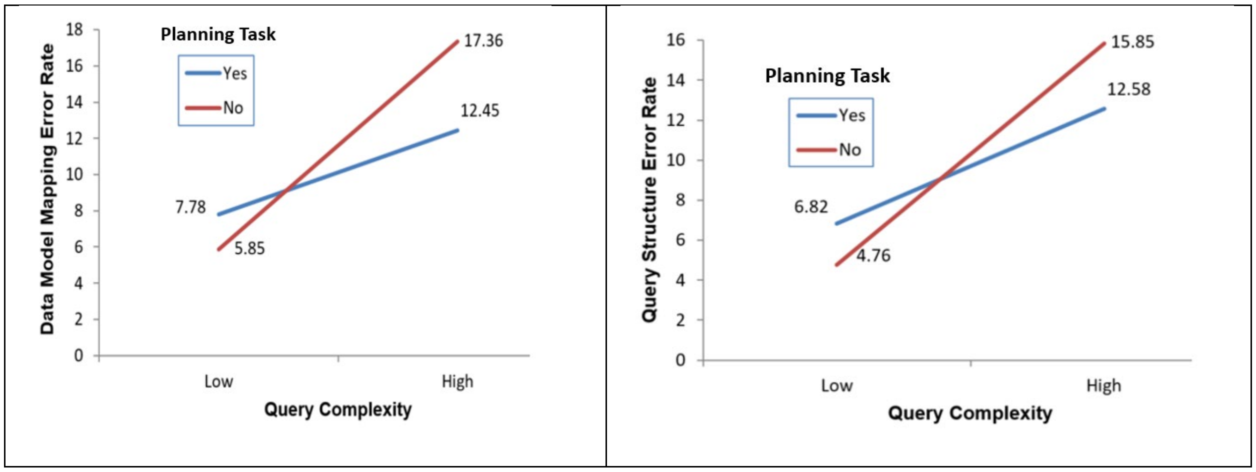


Figure 2. Interaction Effect between Query Complexity and Planning Task on Data Model (DM) and Query Structure (QS) Error Rates

Comparing the results of our analysis with that of VC2016, we find that, in general, our results using the specific error types are consistent with their overall query accuracy measure. However, our analysis uncovers two issues. First, when query complexity was high, the pseudo-SQL requests were no better than manager-English requests in terms of reducing DM error rates. Second, when query complexity was high, the planning task did not have as strong an impact on reducing QS errors as expected.

4.3 Analysis of Errors by Clause

We wanted to see how widespread errors were, so we calculated how many participants had one or more errors of each type in each clause. The vertical axes in the following figures show the percentage of students who had at least one occurrence of the error type.

Figure 3 shows the errors by clause for the four low-complexity queries. Recall that all low complexity tasks required a solution with a SELECT, FROM, and WHERE clause. For these three clauses, both DM and QS errors are shown. Participants may have incorrectly included GROUP BY, HAVING, and/or ORDER BY clauses. We coded a solution as having a QS error if it incorrectly included one of these clauses, but we did not analyze the clauses further for DM errors.

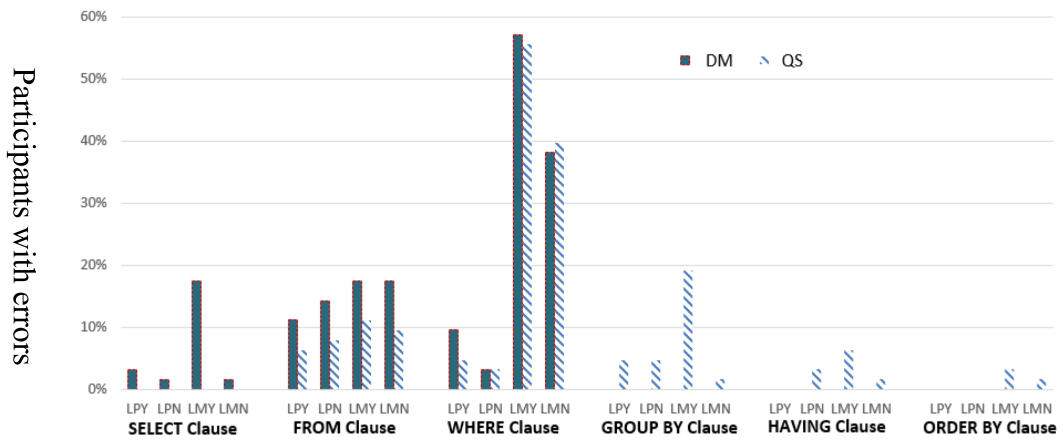


Figure 3. Errors in Low Complexity Tasks, by Clause

Task Abbreviations: L = Low complexity; P = Pseudo-SQL request; M = Manager-English request; Y = Yes planning template was used; N = No planning template was available.

Note: The GROUP BY, HAVING, and ORDER BY clauses were not applicable to the low-complexity tasks. We coded an unnecessary clause inclusion as a structural (QS) error, but did not examine the contents of the incorrectly-added clause, so data model (DM) errors are not applicable for these clauses in the low-complexity tasks.

In Figure 3, the first two tasks in each series are the pseudo-SQL tasks (LPY & LPN), and the latter two tasks in each series are the manager-English tasks (LMY & LMN). In general, fewer participants had errors with pseudo-SQL requests than with manager-English requests, and this is most evident in the WHERE clause, where roughly 40-60% of participants had DM and QS errors. The manager-English tasks included modifiers – "online" orders in the LMY task and "salaried" employees in the LMN task – that were overlooked by these participants when specifying the filtering conditions in the WHERE clause. They missed both the fact that a second condition was needed (omitting the keyword AND, a QS error) as well as the column needed in that condition (OnlineOrderFlag or SalariedFlag, respectively, a DM error). This was the most common problem in the low-complexity tasks. The pseudo-SQL requests explicitly outlined both filtering conditions needed (e.g., "end date is null" or "group name is manufacturing"), and 10% or fewer participants had WHERE clause errors.

While the WHERE clause errors stem from incompletely specifying the solution, the GROUP BY, HAVING, and ORDER BY clause errors stem from over-specifying the solution. These clauses were not needed for the low-complexity tasks and, while some of the queries executed without error, they produced the wrong results. However, other than the LMY task, 10% or fewer of the participants had these over-specification errors in the low complexity tasks. In the LMY task, however, about 20% of participants incorrectly included a GROUP BY clause (QS error) and also incorrectly specified the columns in the SELECT clause (DM error). The LMY manager-English request asked for the "total amount" for each order, which seems to have led this 20% of participants into thinking they needed to aggregate rows (GROUP BY clause) and calculate a sum, rather than simply listing the amount already stored in the TotalDue column. It is clear that the shorter, more natural phrasing used in the manager-English requests resulted in more errors than the longer but more programming-like phrasing used in the pseudo-SQL requests for the low complexity tasks.

Figure 4 shows a similar analysis, but for the high complexity queries. Our hypothesis testing and the interaction effect graphs in Figure 1 showed that, for high complexity queries, pseudo-SQL provided no benefit over manager-English requests in terms of reduced DM errors and only a slight benefit in terms of QS errors. All high complexity queries required a solution with a SELECT, FROM, GROUP BY, HAVING, and ORDER BY clause. For these clauses, both DM and QS errors are shown. Participants may have incorrectly included a WHERE clause, which we coded as a QS error.

In Figure 4, the first two tasks in each series are the pseudo-SQL tasks (HPY & HPN) and the latter two tasks in each series are the manager-English tasks (HMY & HMN). Comparing the pseudo-SQL to manager-English tasks on a clause-by-clause basis, we note the following. First, the HPN task had the highest frequency of DM errors for the SELECT, GROUP BY, and ORDER BY clauses (30%, 38%, and 57% of participants had DM errors in these clauses, respectively). A closer examination of the HPN solutions indicates that the dominant DM error in the SELECT and GROUP BY clauses was omitting a table name as a prefix to the "title" column to clarify which of two possible title columns was referenced (i.e., the title in a person's name, Person.Title, or the job title of an employee, Employee.Title). This ambiguous-column situation was only applicable to the HPN condition and is likely why DM error rates for pseudo-SQL high complexity tasks were comparable to their manager-English counterparts.

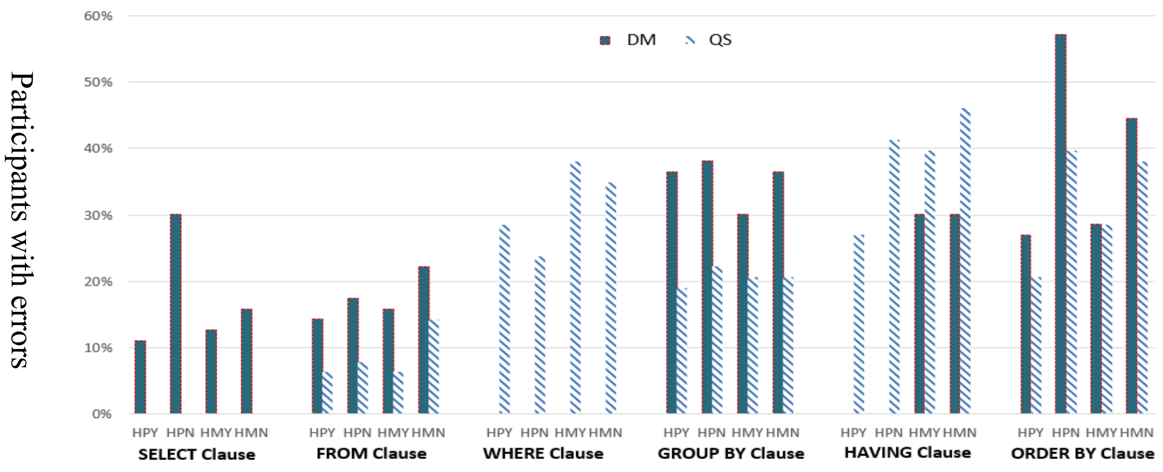


Figure 4. Errors in High Complexity Tasks, by Clause

Task Abbreviations: H = High complexity; P = Pseudo-SQL request; M = Manager-English request; Y = Yes planning template was used; N = No planning template was available.

Note: The WHERE clause was not applicable to the high-complexity tasks. We coded an unnecessary WHERE clause inclusion as a structural (QS) error, but did not examine the contents of the incorrectly-added clause, so data model (DM) errors are not applicable for the WHERE clause in the high-complexity tasks.

Second, for the high complexity queries, fewer participants had QS errors in the FROM, WHERE, and HAVING clauses when given pseudo-SQL requests versus manager-English requests. All participants included a FROM clause in their query solutions, but QS errors occurred if the solution did not recognize the need for two joins of three tables inside the FROM clause. The pseudo-SQL requests explicitly listed the needed tables, and very few participants had errors recognizing the needed structure in the FROM clause. The WHERE and HAVING clauses, on the other hand, were more problematic. The high complexity queries required a HAVING clause but not a WHERE clause, yet 24-38% of participants incorrectly included a WHERE clause and 27-46% incorrectly excluded the HAVING clause. Clearly, participants struggled with the distinction between WHERE and HAVING clauses when query complexity was high. However, it appears that the pseudo-SQL requests, which explicitly included the word "having" instead of "where" (e.g., "show only those groups having more than..."), helped more than the manager-English requests without this clue (e.g., "for each order with a total….above").

Third, we note that for high complexity tasks, the ORDER BY clause was error-prone with 21-57% of participants having a DM and/or QS error. The pattern of errors in Figure 4 is not driven by request language, however, but rather by the planning task, discussed below. The ORDER BY clause is one of the easier clauses to understand and learn, since it simply specifies the desired sort order for the rows in the query result. However, it is also the last clause of a query, and a review of the participant solutions indicates that most of the ORDER BY errors were due to omitting the clause and its columns entirely. It may be that participants were mentally fatigued working on other parts of the high-complexity queries and simply forgot about sorting the rows in the requested order.

4.4 Analysis of Planning Task Errors versus Final Query Errors

Our final analysis concerns the effect of the planning task on final query errors, particularly QS errors. We re-examined the notes that participants wrote for each prompt/clause in the four planning task conditions to record on a clause-by-clause basis whether the clause contained a QS error. The note-taking in the planning task varied considerably in terms of detail – for example, one participant's notes for the SELECT clause state, "last name and total quantity," while another writes, "LastName, SUM(OrderQty) AS TotalQuantity." Thus, we dropped the DM coding for the planning task and focus only on the QS coding.

Figure 5 compares QS errors in the planning task to the QS errors in the final query solution, by clause. Note that the comparison is for only the four query tasks that were preceded by the planning task (LPY, LMY, HPY, HMY).

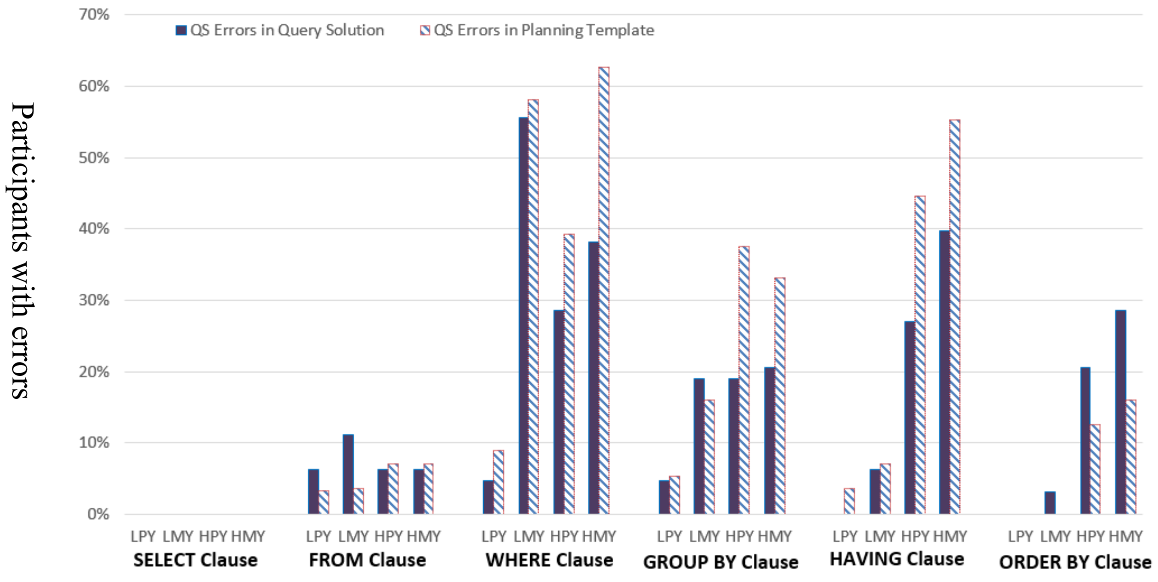


Figure 5. Errors in Planning Task Conditions, by Clause

Task Abbreviations: L = Low complexity; H = High complexity; P = Pseudo-SQL request; M = Manager-English request; Y = Yes planning template was used.

Note: Errors in WHERE clause for HPY and HMY tasks, and errors in the GROUP BY, HAVING, and ORDER BY clauses for the LPY and LMY tasks are for incorrectly including the clause(s).

We note the following observations from Figure 5. First, and unsurprisingly, there are no QS errors in the SELECT clause; this is the first keyword in every query and participants clearly understand this. Second, few participants have QS errors in the FROM clause, which for all of the query tasks required at least one join of two tables. This is interesting in that recognizing and specifying table joins has been problematic in prior studies (e.g., Welty, 1985; Borthick et al., 2001; Bowen, O'Farrell, and Rohde, 2006). We attribute this difference to the fact that in prior studies, join conditions were specified in the WHERE clause, where a missing join condition would not cause an error in the query's execution. In this study, participants were trained to specify joins in the FROM clause using the keywords JOIN and ON, which, if missing, will throw an error when the query is executed. The error message would provide immediate feedback that likely helped participants revise their queries accordingly.

The more interesting observations are about the WHERE, GROUP BY, and HAVING clause results shown in Figure 5. The number of participants who struggled with these clauses varied considerably from the low complexity queries (LPY & LMY) to the high complexity queries (HPY & HMY). For the high complexity queries, more people had WHERE, GROUP BY, and HAVING clause errors in the template than in their final queries. In other words, the prompts in the planning task, intended to help participants determine which clauses were or were not needed in the solution, seem to have hurt their initial solution's structure. For example, in the plans for the HMY condition, 63% of participants incorrectly included a WHERE clause and 55% incorrectly excluded a required HAVING clause. On the positive side, many of these planning task errors were corrected in the participants' final query solutions, although participants may have overcome the problems in spite of the template rather than because of it. A review of the planning notes for the HPY and HMY tasks indicates that most participants knew an aggregate function was needed in either the SELECT statement or a filtering condition (i.e., a WHERE or HAVING clause). If their first query attempts included aggregate functions in the SELECT clause with the related clauses mis-specified, error messages would have provided some feedback which may have helped them correct the problems.

On the other hand, the planning task did appear to help with the ORDER BY clause. All participants in the low-complexity planning tasks correctly recognized that an ORDER BY clause was not needed, and less than 20% of participants in the high-complexity planning tasks failed to correctly include this clause. For this clause, participants' final queries were more error-prone than their planning notes. This suggests that prior to coding, most participants knew that an ORDER BY clause was needed but possibly, by the time they worked through errors in the WHERE, GROUP BY, and/or HAVING clauses, they were not paying attention to their planning notes.

A final observation from Figure 5 is that the WHERE clause in the LMY condition was clearly problematic, both in terms of template notes and final query accuracy. This was discussed earlier – the problem with the "hidden" online order condition in the manager-English request that was missed in the final queries. Similarly, this condition was missed in the planning task, where almost 60% of the planning templates were missing any hint of a second filtering condition.

5. SUMMARY, IMPLICATIONS, AND FUTURE RESEARCH

Our analysis of query formulation errors from the VC2016 study furthers our understanding of the usefulness of two SQL training interventions. First, there is support for using pseudo-SQL requests in the early stages of skill acquisition. We found that pseudo-SQL requests help novices identify WHERE clause conditions that are easy to miss with manager-English requests because they may be "hidden" as adjectives (e.g., online orders, salaried employees). The fact that over half of the participants missed a filtering condition with manager-English requests suggests that novices struggle with analyzing natural language and mapping it to specific columns in the data model. The use of pseudo-SQL removes some of the natural language translation complexity and allows the novice to focus on SELECT statement coding.

The use of exact table and column names in the pseudo-SQL requests helped reduced data-model mapping errors in other clauses as well, with one notable exception – when the same-named column appeared in two tables and the tables were joined. In this case, the column name required a table name prefix, which novices often omitted. Pseudo-SQL requests could be modified to include table name prefixes. This explicit use of database table and column names reduces another source of complexity for novices – the complexity of repeatedly referencing the data model to determine the tables and columns to include. However, reducing this complexity may also encourage novices to ignore the data model and rely exclusively on the terms in the pseudo-SQL request.

These findings have implications for instructors who may find it useful to begin practice with each new SQL concept (e.g., the WHERE clause, the HAVING clause, or AND/OR/NOT conditions) using pseudo-SQL requests until students are competent with the structure and syntax. Instructors can increase the realism of the information request wording over time. However, with the increased realism, novices must learn to dissect the request (e.g., to look for adjectives), compare it to the data model, and ask questions to ensure they understand what tables and columns need to be included in their query solutions. There are many sources of ambiguity in a natural-language information request, and we need more research on how to promote novices' engagement with query formulation tasks. Research on cognitive load theory and scaffolding may be particularly useful here, as it focuses on instructional design that addresses both learner motivation and cognitive support (e.g., Belland, Kim, and Hannafin, 2013).

The usefulness of the second intervention, the planning template, is mixed. As instructors, we believe it is valuable for novices to think about (plan) their query solution before getting bogged down in SQL syntax and trouble-shooting error messages. Our analysis did find support for template usefulness in reducing FROM and ORDER BY clause errors. However, it did not help with more challenging parts of queries, such as recognizing when to use WHERE versus HAVING and recognizing when a GROUP BY clause is needed. We view this as primarily a query structure problem – understanding the purpose and function of these three clauses and their inter-relatedness (along with SELECT clause). There is unavoidable complexity here and those teaching SQL are likely keenly aware of these problem areas for novices.

The implication of these findings, along with other research on complex queries (e.g., Allen and Parsons, 2010), is that instructors still lack effective tools and strategies for teaching SQL grouping and aggregation. One avenue for future research is to improve the planning task so that participants more deeply engage with it. An interactive planning template could be designed, for example, to prompt for a GROUP BY clause only if the participant specifies an aggregate function in the SELECT clause in order to emphasize the relationship between these two clauses. The template contents could also be transferred to the query editor so that the plan, when complete, would essentially become the first draft of the query.

Another direction for future research is to abandon the planning task and instead focus on providing better feedback on query errors. We observed that for the WHERE, GROUP BY, and HAVING clauses, many participants overcame the errors in their initial planning notes by the time they submitted their final queries. This is likely due to errors in their initial queries that produced errors and led to iterative query refinement. A trial-and-error approach may be more natural for novices, and research on errorful learning followed by corrective feedback shows that it can be effective (e.g., Metcalfe, 2017). However, learning from erroneous examples may not be particularly efficient for novices (McLaren et al., 2016). Another approach would combine planning with coding, in smaller increments (a couple of clauses at a time) and iteratively, so that incomplete but working queries are successively refined to get to the correct result set. This could reduce the number of errors and provide positive feedback as novices more quickly get "some" results. It might also promote more engagement with the task, if novices examine the results in the context of what results are desired and use this "gap analysis" to drive their refinements to the query.

6. REFERENCES

Allen, G. N. & March, S. T. (2006). The Effects of State-Based and Event-Based Data Representation on User Performance in Query Formulation Tasks. *Management Information Systems Quarterly*, 30(2), 269-290.

Allen, G. & Parsons, J. (2010). Is Query Reuse Potentially Harmful? Anchoring and Adjustment in Adapting Existing Database Queries. *Information Systems Research*, 21(1), 56-77.

Batra, D. (2007). Cognitive Complexity in Data Modeling: Causes and Recommendations. *Requirements Engineering*, 12(4), 231-244.

Bell, C. C., Mills, R. J., & Fadel, K. J. (2013). An Analysis of Undergraduate Information Systems Curricula: Adoption of the IS 2010 Curriculum Guidelines. *Communications of the AIS*, 32(2), 73-94.

Belland, B., Kim, C., & Hannafin, M. (2013). A Framework for Designing Scaffolds that Improve Motivation and Cognition. *Educational Psychologist*, 48(4), 243-270.

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(1), 3-25.

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 AIS*, 7(8), 514-544.

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.

Casterella, G. I. & Vijayasarathy, L. (2013). An Experimental Investigation of Complexity in Database Query Formulations Tasks. *Journal of Information Systems Education*, 24(3), 211-221.

Greene, S. L., Devlin, S. J., Cannata, P. E., & Gomez, L. M. (1990). No Ifs, ANDs, or ORs: A Study of Database Querying. *International Journal of Man-Machine Studies*, 32(3), 303-326.

Hair, J. F., Black, W. C., Babin, B. J., & Anderson, R. E. (2010). *Multivariate Data Analysis* (7th ed.). Upper Saddle River, NJ: Prentice Hall.

Kaggle. (2017). The State of Data Science and Machine Learning. Retrieved April 16, 2018, from https://www.kaggle.com/surveys/2017.

McLaren, B. M., van Gog, T., Ganoe, C., Karabinos, M., & Yaron, D. (2016). The Efficiency of Worked Examples Compared to Erroneous Examples, Tutored Problem Solving, and Problem-Solving in Computer-Based Learning Environments. *Computers in Human Behavior*, 55(PA), 87-99.

Metcalfe, J. (2017). Learning from Errors. *Annual Review of Psychology*, 68(1), 465-489.

Reisner, P. (1977). Use of Psychological Experimentation as an Aid to Development of a Query Language. *IEEE Transactions on Software Engineering*, 3(3), 218-229.

Reisner, P. (1981). Human Factors Studies of Database Query Languages: A Survey and Assessment. *Computing Surveys*, 13(1), 13-31.

Soat, J. (2014). As Big Data Booms, SQL Makes a Comeback. Retrieved April 20, 2017, from https://www.forbes.com/sites/oracle/2014/09/08/as-big-data-booms-sql-makes-a-comeback/#6c3f1d3252aa.

Topi, H., Valacich, J. S., Wright, R. T., Kaiser, K., & Nunamker, J. F. (2010). IS 2010: Curriculum Guidelines for Undergraduate Degree Programs in Information Systems. *Communications of the AIS*, 26(1), 359-428.

Vijayasarathy, L. & Casterella, G. I. (2016). The Effects of Information Request Language and Planning Task on Query Formulation. *Journal of the Association for Information Systems*, 17(10), 677-710.

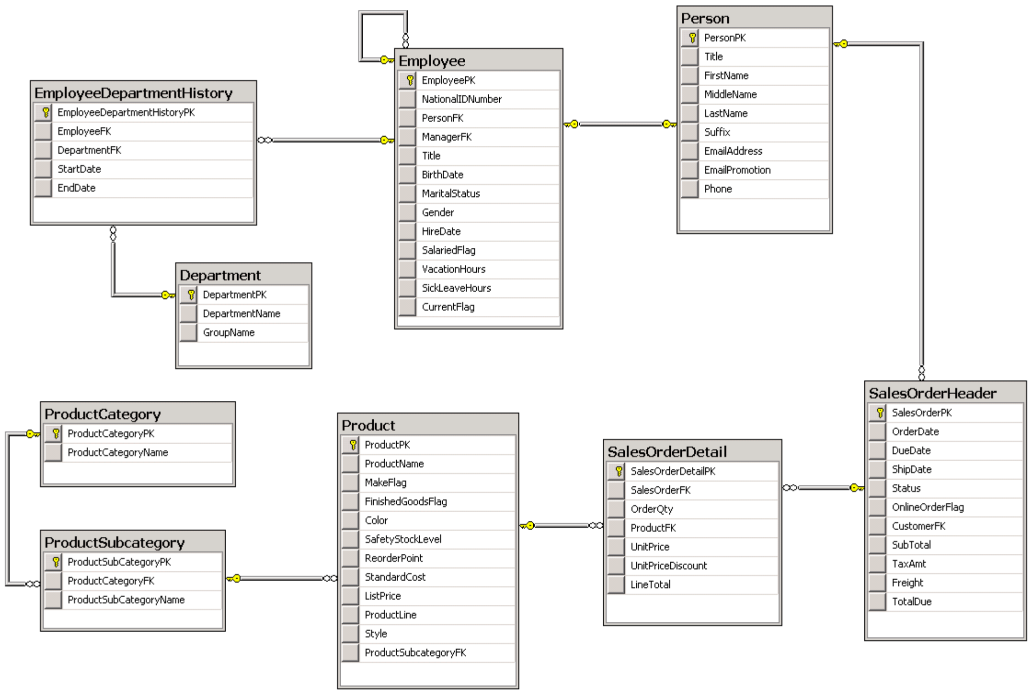
Welty, C. (1985). Correcting User Errors in SQL. *International Journal of Man-Machine Studies*, 22(4), 463-477.

AUTHOR BIOGRAPHIES

Gretchen Irwin Casterella is an associate professor in the Accounting & Business Law Department at the University of North Carolina Wilmington. Gretchen holds a Ph.D. and a Master of Science in Information Systems from the University of Colorado. Gretchen's research interests are in systems development, specifically in understanding how individuals learn and master tools, technologies, and approaches for systems analysis and design. Gretchen's research has appeared in the *Communications of the ACM,* the *Journal of Management Information Systems,* the *Journal of the Association for Information Systems*, and *IEEE Transactions on Professional Communication*.

Leo R. Vijayasarathy is professor of Computer Information Systems in the College of Business at Colorado State University. He earned an M.B.A. from Marquette University and his Ph.D. from Florida International University. His research on the development, use, and consequences of information systems has been published in *Electronic Markets*, *European Journal of Information Systems*, *IEEE Software*, *IEEE Transactions on Professional Communications*, *Information & Management*, *Internet Research*, *International Journal of Production Economics*, *Journal of the Association for Information Systems*, and the *Journal of Management Information Systems*. He serves on the editorial advisory board of *Internet Research*.

APPENDIX 1. Data Model



APPENDIX 2. Query Solution Analysis by Experimental Treatment

	Pseudo-SQL Wording (P)		Managerial Wording (M)	
	Planning Task (Y)	No Planning Task (N)	Planning Task (Y)	No Planning Task (N)
Low Query Complexity (L)	LPY	LPN	LMY	LMN
<u>Query solution breakdown:</u>				
Data-model mapping **(DM)** elements[1]	37.5%	39%	37.5%	37.5%
Query structure **(QS)** elements[2]	25.0%	26%	25.0%	25.0%
Other[3]	37.5%	35%	37.5%	37.5%
Halstead's program length	24	23	24	24
High Query Complexity (H)	HPY	HPN	HMY	HMN
<u>Query solution breakdown:</u>				
Data-model mapping **(DM)** elements[1]	30%	34%	34%	20%
Query structure **(QS)** elements[2]	20%	19%	19%	36%
Other[3]	50%	47%	47%	44%
Halstead's program length	46	47	47	44

[1] Data-model (DM) elements = proportion of the query solution made up of table names and column names.

[2] Query structure (QS) elements = proportion of the query solution made up of SQL keywords that outline the structure of the correct solution (e.g., SELECT, FROM, JOIN, ON, WHERE, AND, OR, GROUP BY, HAVING, ORDER BY).

[3] Other = proportion of the query solution made up of punctuation marks (e.g., commas, quotation marks, parentheses), comparison operators (e.g., <, >, =, IS NULL), function names (e.g., MIN, SUM, COUNT, YEAR), literal values (e.g., 1, manufacturing)

STATEMENT OF PEER REVIEW INTEGRITY

ISSN 2574-3872