Journal of the Association for Information Systems

Research Paper ISSN: 1536-9323

The Effects of Information Request Language and Template Usage on Query Formulation

Leo Vijayasarathy
Computer Information Systems, College of Business,
Colorado State University
leo.vijayasarathy@colostate.edu

Gretchen Irwin Casterella
Computer Information Systems, College of Business,
Colorado State University
Gretchen.Irwin@colostate.edu

Abstract:

The ability to retrieve accurate information from relational databases requires proficiency in structured query language (SQL). In spite of its declarative nature, teaching and learning SQL for complex data-retrieval tasks remains a challenge. Cognitive load theory (CLT) explains the interactions between working and long-term memories in acquiring complex knowledge and skills. We used CLT to identify two instructional interventions expected to improve query writing performance under conditions of high and low query (code) complexity. First, we presented information requests in pseudo-SQL language rather than manager English to clarify the relevant elements in the data model. Second, we provided a query template as an intermediate problem-solving step prior to coding. We conducted an experiment with 63 student participants in a 2 x 2 x 2 repeated measures design in which the request language was either pseudo SQL or manager English, the task included a query template or did not, and the task had either low or high query (code) complexity. Results show that both request language and template usage significantly affected query writing performance but in different situations. Pseudo SQL requests had a significant positive impact on query accuracy with less complex queries but had no impact with more complex queries. On the other hand, using a query template prior to writing SQL code improved task performance with more complex queries but not with less complex queries. We discuss the implications of these findings for instructional design and future research.

Keywords: SQL, Database, Query Formulation, Cognitive Load Theory.

Jeffrey Parsons was the accepting senior editor. This article was submitted on August 11, 2014 and went through two revisions.

1 Introduction

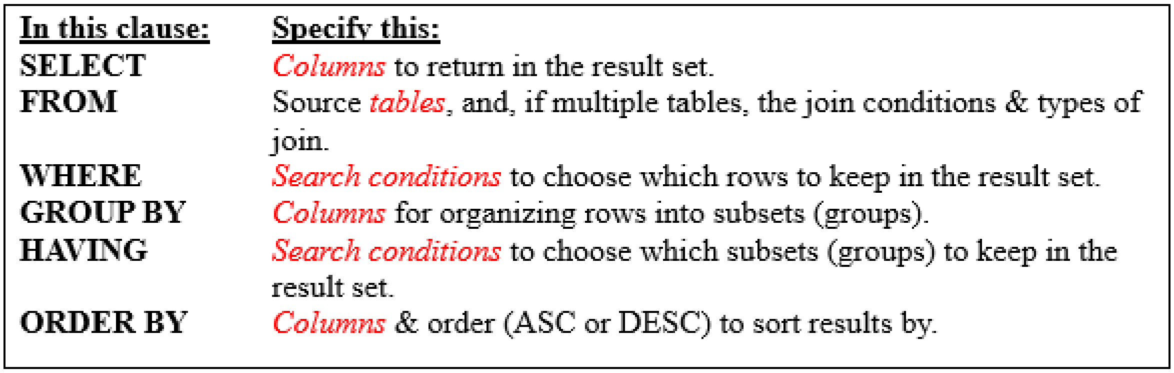
Organizations are finding innovative ways to leverage information in their databases for competitive advantage. For example, Rosenberger (2014) reports that the German national soccer team, winner of the 2014 World Cup, had a strategic edge over its rivals due in part to the enormous amount of performance data that it had gathered about opposing players and teams in its Match Insights database. Many advances in data storage and management have been made in recent years, including the various types of NoSQL ("Not Only" SQL) systems that handle vast amounts of unstructured and semi-structured data (Elmsari & Navathe, 2016). However, relational database management systems (DBMSs) still provide the foundation technology for operational data, and structured query language (SQL) remains the de facto standard for retrieving data from these systems (Allen & March, 2006). Some NoSQL systems also support SQL via extensible middleware that translates SQL queries into the language of the underlying NoSQL system (Rith, Lehmayr, & Meer-Wegener, 2014) and also some data analytic systems, such as R (Grothendieck, 2012). Yet, despite SQL's pervasiveness, learning to correctly formulate queries remains a complex and challenging task (Allen & Parsons, 2010; Bowen, O'Farrell, & Rohde, 2009; Allen & March, 2006; Speier & Morris, 2003; Chan, 1999; Rho & March, 1997; Chan, Wei, & Siau, 1993).

On the one hand, we expect query formulation to be complex since it is a programming task and since programming is an ill-structured problem whose psychological complexity researchers have studied extensively for decades (e.g., Davern, Shaft, & Te'eni, 2012; Curtis, Sheppard, Milliman, Borst, & Love, 1979). On the other hand, SQL differs from most programming languages in that it is a declarative language (i.e., the writer specifies the structure and contents of the query result rather than the sequence of steps required to produce it) (Chamberlin, 2012). Thus, one does not need in-depth knowledge of programming constructs such as variables and control structures to formulate queries, which suggests that writing queries should be less challenging than writing other software programs (Reisner, 1977). However, SQL programming presents its own challenges. To provide a result set that correctly answers the user's information request, query writers must understand a user's natural-language request for information, determine which tables and columns in the database are relevant to the request, mentally simulate the necessary manipulation of rows and columns of data across multiple tables, and map this mental model into SQL syntax constrained by the specific program structure of a SELECT statement.

In our experience, individuals learn to formulate fairly simple queries without much difficulty but struggle with more advanced queries, even after considerable training (Allen & March, 2006; Borthick, Bowen, Jones, & Tse, 2001; Bowen et al., 2009; Casterella & Vijayasarathy, 2013). As the demand for sophisticated information retrieval skills increases, so too does our desire to guide learners to a more advanced level of SQL skills in a limited timeframe. Thus, we are interested in training interventions that improve performance on query-formulation tasks particularly as task complexity increases. We use cognitive load theory to inform the design of the training interventions. The study contributes to the greater body of literature on cognition in the information systems field by examining interactions between cognition and context that influence performance on database retrieval tasks (Davern et al., 2012). We focus on understanding the challenges of formulating queries and on developing instructional methods to improve task performance in training environments.

2 Theoretical Background

Software programming is an ill-structured task and often features poorly defined problem statements, large sets of possible solutions with varying degrees of quality, and a lack of algorithms for deriving the solution from the problem statement (Goel & Pirolli, 1992; Simon, 1973). Query formulation with SQL shares these characteristics, although the solution space is more constrained because of SQL queries' specific structure (see Figure 1). Researchers have described query formulation as a template-generation and mapping problem (Reisner, 1977) in which the writer retrieves the correct SELECT statement template from memory based on the information-retrieval task and fills in the template with the appropriate operations and elements from the database. Other researchers describe query formulation as a multiple-transformation problem (Borthick et al., 2001; Ogden, 1986) in which the writer transforms the initial information request into a mental approximation of the query and then into the specific query language syntax. Figure 2 shows a model of the query-formulation process from Casterella and Vijayasarathy (2013), who adapt Borthick et al.'s (2001) model.



In this clause:	Specify this:
SELECT	*Columns* to return in the result set.
FROM	Source *tables*, and, if multiple tables, the join conditions & types of join.
WHERE	*Search conditions* to choose which rows to keep in the result set.
GROUP BY	*Columns* for organizing rows into subsets (groups).
HAVING	*Search conditions* to choose which subsets (groups) to keep in the result set.
ORDER BY	*Columns* & order (ASC or DESC) to sort results by.

Figure 1. Query Template: Structure of an SQL SELECT Statement

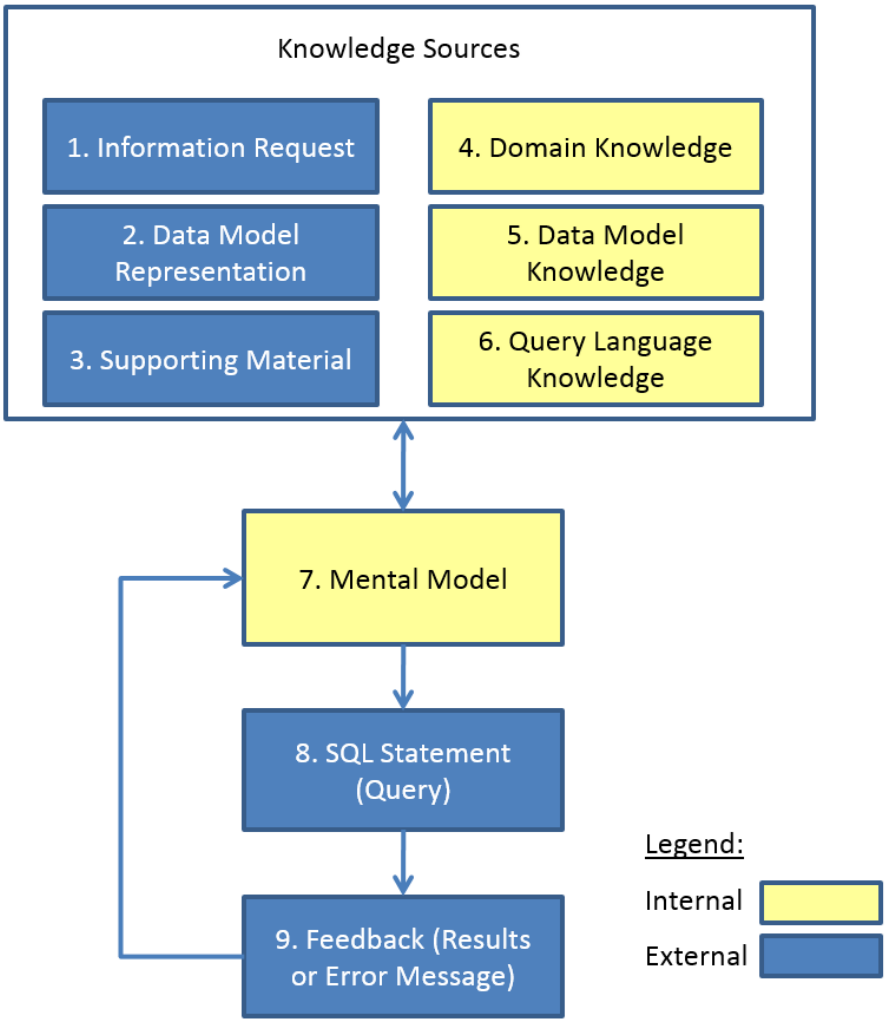


Figure 2. Query Formulation Process Model (Casterella & Vijayasarathy, 2013)

Several knowledge sources provide the input one needs to generate an accurate mental model of the required query. Consider a user's information request (item 1 in Figure 2), such as "Which students registered for more than 15 credit hours last fall?". The query writer needs to determine which elements from the data model to include and which database operations to apply to those elements (Siau & Tan, 2006; Borthick et al., 2001). To make these decisions, the query writer uses knowledge (i.e., internal representation) of the domain, the data model, and SQL (items 4-6 in Figure 2) and the externally represented data model and dictionary (items 2 and 3 in Figure 2). For example, the query writer needs domain and data model knowledge to recognize that finding last fall's registrations involves joining rows from multiple tables (e.g., student, registration, section, and course tables) and filtering that set of rows to retain only those where the column containing term has a value equal to "Fall 2013". The writer needs similar knowledge to transform the request for more than 15 credit hours into a sum of the credit hours for each student across all of the student's fall 2013 registrations. Further, the writer needs SQL knowledge to know that one needs a GROUP BY clause if the SELECT clause includes student names and the sum of the credit hours for which the students registered and a HAVING clause to filter students based on their aggregated (i.e., sum of) credit hours.

The query writer generates a mental model of the query (item 7 in Figure 2) based on this information and transforms that model into a SELECT statement with the syntax that the particular DBMS dictates (item 8 in Figure 2). Siau and Tan (2006) present a concept map (Figure 3) of the elements and transformations one needs generate an appropriate SELECT statement (i.e., program code) to answer a query similar to the example discussed above. This map illustrates the complexity of items 7 and 8 in Figure 2 by highlighting the number of concepts that one must relate and transform in specific ways to generate the correct code. Finally, when the query writer executes the code, the DBMS provides feedback via a result set or an error message (item 9 in Figure 2), and the query writer's mental model of the query may change, which, in turn, may lead to a revised query.

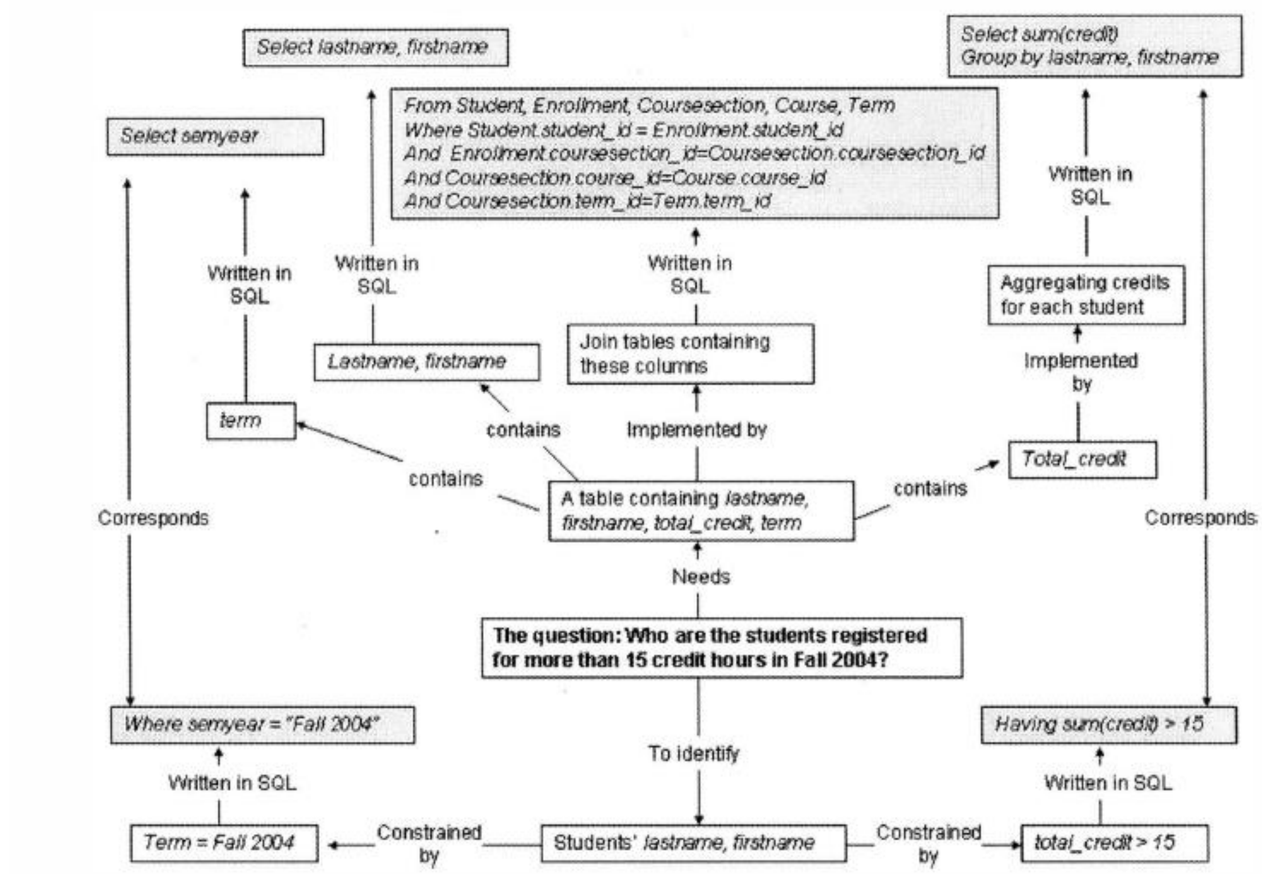

Select lastname, firstname
Select sum(credit)
Group by lastname, firstname
From Student, Enrollment, Coursesection, Course, Term
Where Student.student_id = Enrollment.student_id
And Enrollment.coursesection_id=Coursesection.coursesection_id
And Coursesection.course_id=Course.course_id
And Coursesection.term_id=Term.term_id
Select semyear
Written in SQL
Written in SQL
Written in SQL
Written in SQL
Aggregating credits for each student
Lastname, firstname
Join tables containing these columns
Implemented by
term
contains
Implemented by
Total_credit
contains
A table containing lastname, firstname, total_credit, term
contains
Corresponds
Corresponds
Needs
The question: Who are the students registered for more than 15 credit hours in Fall 2004?
Where semyear = "Fall 2004"
Having sum(credit) > 15
To identify
Written in SQL
Written in SQL
Term = Fall 2004
Constrained by
Students' lastname, firstname
Constrained by
total_credit > 15

Figure 3. Concept Map of Query Formulation (Siau & Tan, 2006)

2.1 Cognitive Load Theory

Cognitive load theory (CLT) (Sweller, van Merrienboer, & Paas, 1998; van Merrienboer & Sweller, 2005) provides instructional design recommendations to help one learn complex knowledge and skills based on a rich understanding of the interactions between long-term and working memory. Long-term memory has no known capacity limits and comprises domain-specific and highly structured schemas or mental models (Chi, Glaser, & Rees, 1982; Chandler & Sweller, 1996). Working memory, on the other hand, has capacity and duration limits. For instance, humans hold seven (plus or minus two) elements in working memory at a time (Miller, 1956) but focus only on two to four of those elements, and one loses information in working memory after about 20 seconds unless one uses or rehearses it (van Merrienboer & Sweller, 2005). Learning, then, involves developing schemas in long-term memory. These schemas extend the amount of information that one can process in working memory because they treat multiple interacting elements as a single chunk of information. Eventually, one can automate these schemas and bypass working memory altogether—a hallmark of expertise (Chase & Simon, 1973; Sweller, 1994; Chandler & Sweller, 1996). Experienced SQL programmers have well-developed schemas for writing SELECT statements and heuristics for when and how to translate the user's request for information into the various clauses of the statement. These individuals will chunk and automate many of the mappings in Figure 3.

CLT focuses on strategies that promote schema development in long-term memory while easing the demands on working memory to improve learning complex tasks (Cook, 2006). When problem solving, one faces three types of demand: intrinsic load (IL), extraneous load (EL), and germane load (GL) (Sweller et al., 1998). IL refers to a task's inherent complexity. It is a function of the interactivity among the elements one seeks to learn—the task's coordinative complexity (Wood, 1986)—but is offset by the problem solver's expertise (Sweller et al., 1998). EL refers to the aspects of the instructional materials or environment that are unnecessary or irrelevant for learning (i.e., the "noise" imposed on the problem solver). GL refers to the mental resources invested in activities that encourage schema acquisition and automation. Effective instructional design reduces EL and increases GL by redirecting learners' attention away from processes irrelevant to learning and toward processes that foster "the construction and mindful abstraction of schemas" (Sweller et al., 1998). In teaching SQL, we try ensure learners acquire SELECT statement schemas and the rules and heuristics for when and how to use the specific clauses in the statements. To the extent that we can reduce EL and simplify early learning tasks to reduce IL, learners have the opportunity to use more working memory resources to acquire relevant schemas.

2.2 Instructional Strategies

CLT research has identified instructional strategies that improve learning how to solve complex problems through a reduction in EL and/or an increase in germane load (Ayres & Paas, 2012; Chandler & Sweller, 1996; Pollock, Chandler, & Sweller, 2002; Rourke & Sweller, 2009; Wang, Yang, Liu, & Cao, 2014; Wickens, Hutchins, Carolan, & Cumming, 2013). We focus on three instructional strategies in particular because they apply well to query-writing tasks: worked examples, part-task sequencing, and simplified-whole task sequencing. These strategies reduce cognitive load, facilitate learning, and improve task performance in a variety of domains such as electronic circuit troubleshooting, programming, art appreciation, mathematics, ecommerce website design. They are similar in that they reduce the intrinsic load of the target task by artificially simplifying it in the early stages of learning so that one has working memory resources available for germane load and schema formation. Early schema formation then helps learners handle the intrinsic load of the true, target task at a later time.

The worked examples strategy has consistent empirical support in CLT studies. With this strategy, learners initially receive problems with solutions that illustrate the domain-specific tactics used to create the solution (Ayres & Paas, 2012; Rourke & Sweller, 2009; van Gog, Kester, & Paas, 2010). This strategy circumvents novices' reliance on weak problem-solving methods that overwhelm working memory by changing the initial task from problem solving to problem understanding. Learners in a problem-understanding task can then use germane load to acquire domain-specific schemas that will later assist in problem-solving tasks. After the initial worked examples, learners receive problems with partially worked solutions, and finally, problems to solve "from scratch". Rourke and Sweller (2009) show that participants performed better with worked-examples followed by problem solving than with problem solving only. Van Gog et al. (2010) studied three variations on the worked-examples strategy and found that participants had higher performance in all three conditions compared to the problem solving-only condition.

The part-task sequencing strategy addresses the challenges of learning tasks with high element interactivity, such as in electronic circuit troubleshooting (Pollock et al., 2002). The dependencies and relationships among the elements of the task increase the coordinative complexity (Wood, 1986) and present the learner with a paradox:

> *Material can be understood once a schema has been constructed allowing all the elements to be processed in working memory simultaneously, but until that point has been reached, the elements cannot be processed simultaneously…and so cannot be understood…. [In essence] the initial*

> *stages of schema construction must occur in an environment in which the material cannot be understood*. (Pollock et al., 2002, p. 64)

Part-task sequencing artificially and temporarily reduces intrinsic load by decomposing the task into segments and focusing on each segment in isolation and sequentially before training on the whole task. It is a progressive learning strategy that research has shown to be effective when one performs a task's subtasks sequentially rather than concurrently (Cook, 2006; Gerjets, Scheiter, & Catrambone, 2004; van Merrienboer & Sweller, 2005; Wickens et al., 2013).

The simplified-whole task or increasing difficulty strategy corresponds to part-task sequencing in that it also artificially reduces the complexity of the problem-solving task in the initial learning phases. However, this strategy presents the whole task with the parameters initially set to lower levels to reduce the intrinsic load stemming from the task's otherwise high coordinative complexity (Gerjets et al., 2004). As training progresses, the strategy sets the parameters to increasingly difficult levels (van Merrienboer & Sweller, 2005). This strategy is consistent with scaffolding training (Wood et al., 1976) in which learning begins with a framework of the task and the learner receives information to flesh out the framework over time with practice. The strategy illuminates certain aspects of the framework and hides others so that it reduces intrinsic load and promotes active engagement in learning (Belland, Kim, & Hannafin, 2013; Wood, Bruner, & Ross, 1976). Research has shown the simplified-whole task strategy to be effective when the task requires a high degree of element integration and coordination (van Merrienboer & Sweller, 2005).

The three strategies that we describe above do not mutually exclude each other and apply to learning SQL and to learning how to form queries in a university environment. Indeed, while database management textbooks, such as Hoffer, Venkataraman, and Topi (2012), Coronel and Morris (2014), Pratt and Adamski (2012), and Kroenke and Auer (2013) do not explicitly describe instructional strategies, one can see the strategies in these texts to varying degrees. These textbooks consistently present worked examples, often based on ongoing business cases, to illustrate and discuss solutions to data-modeling and query-formulation problems. End-of-chapter problems then allow students to solve similar problems from scratch. However, we could not find an instance in these textbooks that used partially worked examples to transition learners from completely worked examples to completely unworked ones.

One can notice the part-task sequencing strategy in database management textbooks through how they sequence their content. For example, most texts include chapters on relational database design prior to chapters on SQL and query formulation so that understanding how databases organize data precedes learning how to manipulate that data in the confines of an SQL SELECT statement. Further, most texts organize their content on SQL queries and present query examples in specific ways. Hoffer et al. (2012), Coronel and Morris (2014), and Kroenke and Auer (2013) first discuss queries with the SELECT, FROM, and WHERE clauses and later add other clauses such as GROUP BY and HAVING, which allows students to acquire skills for a core part of the SELECT statement before being exposed to the remaining clauses. However, not all authors use the same sequencing strategy. For example, some texts discuss the ORDER BY clause before (e.g., Rockoff, 2011) or after (e.g., Hoffer et al., 2012) the WHERE clause.

One can also see part-task sequencing in the textbooks in the way they phrase information requests. Authors often present the natural-language information request in a way that illuminates the parallels between the request and certain elements in the corresponding SELECT statement. For example, Hoffer et al. (2012) present an information request as follows: "List product name, finish, and unit price for all desks and tables in the PRODUCT table that cost more than $300" (p. 269). The request explicitly mentions the relevant table name (PRODUCT), and the sequence of the phrases and clauses in the request closely parallel the phrases that query solution needs (e.g., the first part of the request, "List product name, finish, and unit price," parallels the first part of the query solution, "SELECT ProductDescription, ProductFinish, ProductStandardPrice"). The request uses phrasing that clarifies which table to reference, which then narrows the set of possible columns to include in the SELECT clause. The reduced effort on the first part of the query allows the reader to focus on the query's WHERE clause. While, in the early query examples, authors tend to use more precise language in their information requests, the texts (individually and as a group) vary considerably both in the extent to which they phase out and replace this precise language with more typical end-user natural language.

Finally, one can see that the current database management textbooks adopt the increasing difficulty or simplified-whole task approach but to a lesser extent than the previous instructional strategies. Most textbooks begin discussing queries with a simplified query template that includes only the SELECT and FROM or the SELECT, FROM, and WHERE clauses (e.g., Hoffer et al., 2012; Coronel & Morris, 2014; Pratt

& Adamski, 2012; Kroenke & Auer, 2013) rather than the entire SELECT statement template (see in Figure 1). The authors do consistently increase the complexity in each clause of the query (e.g., simple conditions before compound or complex conditions in the WHERE clause, single table before multiple tables in the FROM clause) and increase the number of clauses included in queries over time.

The CLT instructional strategies—worked examples, part-task sequencing, and simplified-whole task sequencing—are interrelated and evident in popular database management textbooks used in university courses that teach SQL. While CLT explains how these approaches can help learners acquire complex skills, little empirical evidence on the effectiveness of these or other specific approaches in the domain of query formulation exists.

2.3 The Expertise-Reversal Effect

CLT research has also shown that instructional methods that work well for novices may not work well for more proficient problem solvers (Kalyuga, Ayres, Chandler, & Sweller, 2003; Pollock et al., 2002). Researchers have called this phenomenon the expertise-reversal effect. Specifically, the effect notes that instructional material that helps novices reduce cognitive load actually interferes with experienced learners' ability to apply long-term memory schemas because they are compelled to attend to or use the instructional material even when they don't need it—in other words, the materials adds extraneous load for the experienced learners but not for the novices. The expertise-reversal effect applies to learners at intermediate skill levels as well. Instructional material that helps learners with more complex tasks (with which they have little experience) may interfere with simpler tasks (with which they are proficient). An alternative explanation for the expertise-reversal effect is that, with less instructional material, the more experienced problem solvers engage more deeply with the task because more resources are available for germane load (Kalyuga et al., 2003). Thus, CLT research recommends fading out the instructional interventions as learners gain experience and acquire domain-specific schemas. However, with query formulation instruction, we do not yet know which interventions are effective for learners at a given level of training let alone when and how to phase those interventions out.

3 Hypothesis Development

For this study, we focus on two training interventions that CLT motivate and that we expect to improve query formulation task performance: 1) presenting information requests in pseudo-SQL to clarify mappings to the data model and 2) providing a query template (Figure 1) as an intermediate problem-solving step. We expect these interventions to have positive effects on task performance, particularly as task complexity increases. Figure 4 shows our research model.

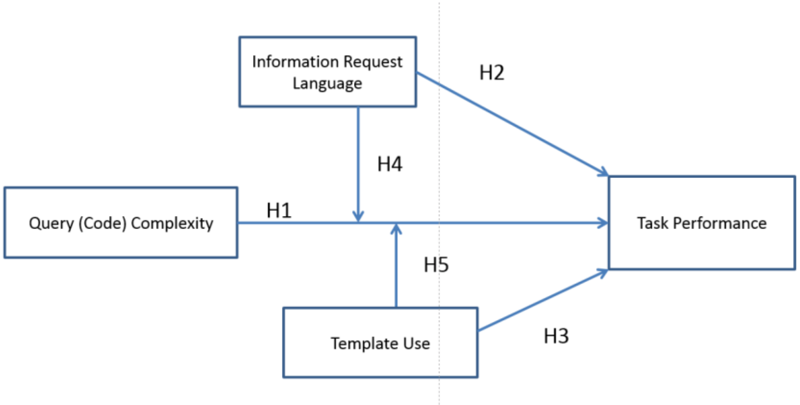


Figure 4. Research Model

3.1 Task Performance

Prior studies measure task performance in terms of query accuracy, the time taken to write the query, and the participants' confidence in the accuracy of their queries (Allen & Parsons, 2010; Allen & March, 2006; Borthick et al., 2001; Chan et al., 1993). While we focused predominantly on query accuracy, we included all three dependent variables and the number of attempts (defined as the number of times the participants executed a query before submitting their final answer) to be consistent with prior research (e.g., Borthick et al., 2001).

3.2 Query (Code) Complexity

Empirical studies of query formulation have consistently found a negative relationship between query complexity and task performance (Borthick et al. 2001; Bowen et al., 2009; Chan, 1999; Rho & March 1997). Query complexity refers to the properties of the SELECT statement solution that make it cognitively challenging to write (independent of the writer's expertise). The solution's complexity is a primary source of the intrinsic load one experiences. Researchers have described this complexity as comprising component complexity and coordinative complexity (Wood, 1986) or structural and lexical complexity (Allen & Parsons, 2010; Casterella & Vijayasarathy, 2013). It is a function of the number and type of elements in the solution and of the number and nature of the relationships among the elements (Pollack et al., 2002; Wickens et al., 2013; Wood, 1986).

For example, consider the following two queries:

Query 1

```
SELECT EmployeeID, HireDate
FROM Employee
SELECT EmployeeID, HireDate
```

Query 2

```
SELECT LastName, SalesOrderPK,COUNT(*)
  AS NumberOfOrderLines, SUM(OrderQty)
  AS TotalOrderQty
FROM SalesOrderHeader JOIN
  SalesOrderDetail ON SalesOrderPK =
  SalesOrderFK
  JOIN Person ON PersonPK = CustomerFK
GROUP BY LastName, SalesOrderPK
HAVING COUNT(*) > 5
```

Independent of the individual's and the data model's characteristics, the second query is more cognitively complex than the first one not only because the second query is longer and contains additional, new elements (e.g., COUNT, JOIN, HAVING, *) but also because of the relationships or interactivity among these elements. For example, the columns specified in the GROUP BY clause relate to the columns specified in the SELECT clause; the HAVING clause must come after the GROUP BY clause, and the number of tables and joins in the FROM clause are driven by which columns other clauses specify (Bowen, O'Farrell, & Rohde, 2006; Bowen et al., 2009; Reisner, 1977). The second query has higher structural complexity (requiring more clauses from the query template) than the first one and greater lexical complexity because it has more column names and table names. We expect, other things being equal, that greater cognitive complexity in the target solution will impose greater demands on the writer's intrinsic load and, thus, reduce the writer's task performance. As such, we hypothesize that:

H1: As query complexity increases:

H1a: Solution accuracy decreases

H1b: Query writing time increases

H1c: The number of query attempts increases, and

H1d: Query confidence decreases.

3.3 Request Language

Formulating queries is a multiple-transformation task, which makes it a good candidate for CLT's part-task sequencing training strategy (Cook, 2006; Gerjets et al., 2004; Pollock et al., 2002; Wickens et al., 2013). When beginning to formulate a query, the writer transforms the user's information request into a mental model of the query, which includes a subset of elements from the data model (the top portion of Figure 2). The intrinsic load for this part of the task depends, in part, on the cognitive gap between terms in the information request and elements in the data model that the writer needs to include and manipulate in the query solution (Borthick et al., 2001). One way to reduce this gap is to present the information request in a different form—one that partially translates the natural-language request and that one can more clearly and directly map into a query solution. In theory, this increased clarity in the information request frees up one's cognitive resources to spend on the details and syntax of each SELECT statement clause.

Consider the example request: "Which students are registered for more than 15 credits?". This request, phrased in manager English terms (Siau & Tan, 2006), places fairly high demands on the query writer's cognitive resources to determine which tables, columns, and data values the query needs. The same task worded in pseudo SQL (Siau & Tan, 2006) might be: "List the lastName and firstName of students (from the Student table) if the sum of the credits (from the Course table) for which they registered is greater than 15.".

The pseudo SQL request has greater clarity than the manager English request because it explicitly references the elements from the data model that the query needs.

Prior research indicates that pseudo SQL requests positively impact query formulation performance when compared to manager English requests (Axelson, Borthick, & Bowen, 2001; Borthick et al., 2001; Casterella & Vijayasarathy, 2013). While, in practice, information requests will not appear in pseudo SQL, the pseudo SQL may be an effective training intervention that allows novices to develop schemata to formulate other queries so that, when they later have to perform manager English tasks, the novices have more cognitive resources available for the task's request-data model-mapping portion. For these reasons, we expect that increasing information request clarity through pseudo SQL wording will improve task performance. Thus, we hypothesize that:

H2: When the information request language is pseudo SQL rather than manager English:

H2a: Solution accuracy increases

H2b: Query writing time decreases

H2c: The number of query attempts decreases, and

H2d: Query confidence increases.

3.4 Template Usage

We designed our second intervention—a SELECT statement template—to help learners transform their mental models into queries with the appropriate clauses and specifications (items 7 and 8 in Figure 2). CLT's simplified-whole task strategy (Gerjets et al., 2004; Wickens et al. 2013) is relevant for this part of the task because of (potentially) high element interactivity (Wood, 1986)—the clauses of a SELECT statement work together such that the elements in one clause relate to elements in another clause.

The template intervention may reduce the intrinsic load of generating the correct SELECT statement by providing an explicit intermediate solution. The template in Figure 1 specifies the possible clauses to include in the query in the correct sequence and with the correct starting key words, and it reminds learners what to specify in each clause. We expect template usage to improve task performance in terms of solution accuracy because it encourages a mental dialog about which parts of the SELECT statement the query needs and why, which requires germane load and fosters schema acquisition. The template also eliminates the need to remember the sequencing of clauses (artificially simplifying that part of the task) so that the writer spends resources on determining the content of the clauses. For similar reasons, we expect template usage to reduce the number of query attempts and increase the writer's confidence.

However, the impact of the template on the time to complete the task is unclear. We expect the time spent on the template to reduce the subsequent time spent writing, executing, and revising a query, but we do not know whether the total time spent on the task—which includes completing the template *and* writing and revising the query—will be greater than, the same as, or less than the time spent on the task with no template.

As such, we hypothesize that:

H3: When a writer uses a query template prior to coding:

H3a: Solution accuracy increases

H3b: The total time spent on the task is unchanged

H3c: The number of query attempts decreases, and

H3d: Query confidence increases.

3.5 Interaction of Complexity with Instructional Interventions

CLT research has shown that the effectiveness of instructional interventions varies with task complexity (Gerjets et al., 2004; Pollack et al., 2002; Wickens et al., 2013). Higher element interactivity produces greater intrinsic load, which leaves fewer cognitive resources available for extraneous and germane load. With low element interactivity, the intrinsic load is less and, thus, one has more working memory resources available and can tolerate greater extraneous load without a negative impact on task performance (e.g., Chandler & Sweller, 1996). Similarly, we expect the impact to vary based on the problem solver's expertise (Cook, 2006) because more experienced problem solvers will have schemata in long-term memory that reduce the task's intrinsic load. In our study, we focus on intermediate-level novices. Over several weeks of instruction in a college-level database course, we expected that they developed schemas for simpler query tasks but that they possibly would not have had sufficient practice to develop schemas for more complex tasks. Accordingly, we expected that our interventions would benefit them—particularly in undertaking more complex queries.

We expect that request language and query complexity will have an interaction effect on task performance. If a query task has low intrinsic load—because it is objectively a simpler query or because the writer has sufficient experience with queries of the same complexity level—then the writer does not need the extra clarity offered through pseudo SQL. However, if writers have had less practice with complex queries and so have not developed the corresponding schemata (e.g., to recognize when they need GROUP BY and HAVING clauses), then artificially reducing the intrinsic load will be beneficial and should improve performance. We expect that, for complex queries, the increased clarity of pseudo SQL requests will improve task performance. As such, we hypothesize that:

H4: Request language clarity has a stronger positive effect on performance as query complexity increases. Specifically, as query complexity increases, participants given pseudo SQL requests:

H4a: Write more accurate queries than participants given manager English requests

H4b: Spend less time writing queries than participants given manager English requests

H4c: Make fewer query-writing attempts than participants given manager English requests, and

H4d: Are more confident in their query's accuracy than participants given manager English requests.

We also expect to see an interaction effect between template usage and query complexity*.* As we mention earlier, the template is an instructional intervention to help reduce cognitive load by providing an intermediate solution plan. We expect the template to be particularly useful with complex queries because it can extend individuals' working memory. Specifically, we expect template usage to improve solution accuracy, reduce the number of query writing attempts, and increase confidence when dealing with more complex query tasks. However, the expected effect of template usage on total task time for complex tasks is unclear because the time one spends on the template may exceed any time saved on making fewer query attempts.

For queries with low complexity, where the cognitive load is manageable without any instructional intervention, asking participants to use the template may actually be detrimental because of the expertise-reversal effect (Kalyuga et al., 2003; Pollack et al., 2002). If the writer does not need the template, it may distract the writer to have to attend to it prior to writing the query. As such, having to attend to the template when not needed may increase extraneous load, subsequently leave fewer working memory resources for intrinsic and germane load, and, thus, decrease performance. In essence, for low-complexity queries, the template-filling task may overcomplicate an otherwise straightforward task. Thus, we hypothesize that:

H5: When comparing tasks with the template to those without the template:

H5a: For low-complexity queries, template usage decreases solution accuracy, but, for high-complexity queries, template usage increases solution accuracy.

H5b: For low-complexity queries, template usage increases the time spent, but, for high-complexity queries, template usage does not affect the time spent.

H5c: For low-complexity queries, template usage increases the number of attempts, but, for high-complexity queries, template usage decreases the number of attempts.

H5d: For low-complexity queries, template usage decreases confidence, but, for high-complexity queries, template usage increases confidence.

4 Research Method

4.1 Experimental Design

We used a 2 x 2 x 2 repeated measures factorial design for our study. Each of the three factors had two levels (i.e., high vs. low task complexity, high request clarity vs. low request clarity, and template vs. no template), which resulted in eight experimental conditions. We designed eight query-formulation tasks to meet the requirements of these conditions and asked participants to complete all eight tasks. To minimize carry-over effects in our repeated measures experiment, we used a balanced Latin-square design (Pezzullo, 2008; Williams, 1949). We randomized the order of tasks across participants.

4.2 Independent Variable: Query (Code) Complexity

Query complexity refers to the cognitive complexity of the SQL SELECT statement code that solves the given problem. Halstead's (1977) program difficulty metric is an approximate measure of a program's psychological complexity based on the number of unique operators (e.g., SELECT, AND, IN, LIKE, =) and operands (e.g., table names, column names, column values) and the frequencies of those operators and operands in the program (Curtis et al., 1979). We treated each query solution as a "program" and calculated the Halstead difficulty metric as in prior studies of query formulation (Borthick et al., 2001; Casterella & Vijayasarathy, 2013). All query tasks were for the same database whose data model Appendix A shows.

The low-complexity queries comprised three clauses (SELECT-FROM-WHERE) and required an inner join of two tables. The high-complexity queries comprised six clauses (SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY) and an inner join of three tables. All low-complexity tasks had a Halstead difficulty of 12.5; all high-complexity tasks had a Halstead difficulty of 33.3.

We note that two prior studies on query formulation (Borthick et al., 2001; Casterella & Vijayasarathy, 2013) also measured complexity via Halstead's difficulty metric and had participants similar in background to ours. In Borthick et al., queries ranged in complexity from a Halstead score of about 5 to about 30; in Casterella and Vijayasarathy (2013), they ranged from about 4 to 20. In both prior studies, the least complex query tasks were (nearly) free from errors and completely accurate, and we chose not to use our limited experimental time for tasks such as these. Thus, our low-complexity tasks were more akin to the mid-range tasks in these two other studies. Our high-complexity tasks were similar in terms of Halstead complexity to the high-complexity tasks in Borthick et al. (2001) and higher in complexity than the most complex queries in Casterella and Vijayasarathy (2013). Appendix B lists the query solutions for all eight experimental tasks.

4.3 Independent Variable: Request Language

For our study, we used either pseudo SQL or manager English as the request language (Borthick et al., 2001). Pseudo SQL requests had greater clarity than manager English requests because they explicitly referenced the table names and column names needed in the query and they included clues about the query solution contents, such as referring to "groups" when one needed a GROUP BY clause. Appendix B shows examples of the two wordings for each query task.

4.4 Independent Variable: Template Usage

We operationalized template usage by presenting the query template (Figure 1) for half of the tasks. In these template tasks, we asked participants to take notes and submit the template *before* they could proceed to query writing. In most cases, the participants used the template as we intended. In some cases, however, participants entered meaningless information, and, thus, we did not count these participants as having "used" the template.

4.5 Dependent Variables: Task Performance

We used several measures of performance, including the accuracy of a participant's final query solution, the time taken to complete the query writing task, the number of query attempts that the participants executed (including the final solution), and the participants' confidence in the accuracy of their final solution. We discuss each of these variables below.

4.5.1 Query Accuracy

We assessed accuracy by comparing a participant's query solution to an "ideal" solution (e.g., Allen & March 2006; Borthick et al., 2001; Casterella & Vijayasarathy, 2013). For each query task, we identified all elements that a correct query required, assessed whether each participant's final query contained the required elements, and computed query accuracy as the ratio of the number of elements present to the number of required elements. A trained research assistant assessed the queries. The second author audited the assessments, and that author and the research assistant discussed and reconciled all differences. Appendix C shows an example of the query accuracy scoring grid for one participant and one task.

4.5.2 Query Formulation Time

Time is a complimentary performance measure to accuracy that empirical studies of query performance have often used (e.g., Allen & March, 2006; Allen & Parsons, 2010; Borthick et al., 2001; Bowen et al., 2009). The application used for the experiment recorded the time each participant initiated a new query task and the time they submitted their final query try for that task. We computed query formulation time as the difference between these two times. For tasks that included the query template treatment, formulation time *included* the time spent making notes in the template prior to using the query editor window.

4.5.3 Confidence

Confidence measures how well the participants thinks they performed on a query task and is another measure of performance that query research has often used (e.g., Allen & March 2010; Ashkanasy, Bowen, Rohde, & Wu, 2007; Borthick et al., 2001). After submitting the final query for each task, the participants rated their confidence on a five-point scale (1 = not at all confident; 5 = very confident).

4.6 Participants

Eighty-seven students participated in the study. They were computer information systems and computer science students enrolled in one of two sections of a third-year database management course at a U.S. public university. The same instructor taught both sections. There were 22 female and 65 male participants, most of whom (83%) were between the ages 18 and 25. The number and background of these participants is comparable to the participants in other studies of query formulation (Borthick et al., 2001; Bowen et al., 2006, 2009; Chan et al., 1993). We controlled for individual differences through using a relatively homogenous group of students and the study's within-subject, repeated measures design (Allen & Parsons, 2010). Prior to the study, participants received several weeks of instruction on entity-relationship modeling and database design, followed by several weeks of training on manipulating relational databases with SQL. Thus, at the time of our study, participants were not "pure" novices with respect to relational databases; more accurately, they were intermediate-level novices.

As incentives, we gave participants extra course credit for participating in the study and the opportunity to earn additional extra credit based on their performance on the experimental tasks. We also told them the experimental tasks would provide practice for query tasks that would be on their final course examination. Participants registered for an experimental session of their choosing. Many session timeslots were available over a five-day period.

4.7 Experimental Procedure

To gather data for the study, one of the authors developed an application called CeeKwel using Microsoft development technologies that had functionality similar to that described in other query studies (Allen & March 2006; Allen & Parsons, 2010). CeeKwel has a tabbed-interface for writing, executing, and receiving feedback on queries (i.e., query result or error message). A few days prior to the experimental sessions, participants completed an in-class training session, which included training on CeeKwel and query-formulation tasks comparable to those used in the experimental sessions.

At the beginning of the experimental session, participants signed in to CeeKwel using an assigned user ID and session password. They then completed an online comprehension quiz to assess their understanding of the data model they would be using for the main part of the study. When they completed the comprehension quiz, they began the main query-formulation part of the study. Participants worked on a series of eight query tasks, one for each of the experimental conditions (see Appendix B). We randomized the order of tasks across participants. For the four template tasks, CeeKwel presented a query template and asked participants to use the template to take notes or otherwise plan their query. When participants submitted the template, the query editor window opened in the corresponding pane.

Participants could revise their queries as many times as they wanted but had to submit their final query before moving on to the next task. Participants could also refer to the data model at any time. After submitting their final query for a task, participants rated their confidence in the accuracy of their final query and the perceived difficulty of the query task. Then the system presented the next task. For each participant and task, CeeKwel recorded each executed query, the time of execution, error messages and template contents (where applicable), and the confidence and difficulty ratings. At the end of the session, participants completed an exit survey to gather background data (e.g., grade point average, age, gender). Experimental sessions were about two hours long, similar to other query formulation studies (Borthick et al., 2001; Bowen et al., 2006).

5 Results

We used a repeated measures design and, thus, included data from only those participants who submitted queries for all eight experimental tasks and used the template for the four tasks that presented the template. We dropped 24 of the original 87 participants from the analysis because they did not meet these requirements. Eight of these twenty-four participants did not follow the instructions for using the template and sixteen did not complete all eight tasks. Of the sixteen who did not finish, nine almost finished (completing seven of the eight tasks) and five were not close to finishing (completing five or fewer tasks). We compared the dropped participants to the remaining 63 participants on demographic variables and found one significant difference: the participants who completed five or fewer tasks had lower self-reported SQL proficiency than the other participants (3.00 versus 4.84, on a scale of 1-7). To investigate the impact of dropping the 24 participants, we reran the repeated measures analysis while dropping one of the three independent variables (query complexity, request language, and template use) at a time. The results from this analysis were consistent with the results from the main analysis described below, which are based on the remaining 63 participants. We note, however, that the statistical power of the analyses of the dropped subjects were low; thus, one should interpret them with caution.

The participants had an average self-reported GPA of 3.12 (out of 4.0). On the data model comprehension pre-test, the average and median scores out of 4.0 were 3.1 and 3.0, respectively. Their average (median) self-reported comfort levels (1 = not at all comfortable; 7 = highly comfortable) were 5.8 (6.0) for reading ER diagrams and 4.9 (5.0) for writing SQL queries.

5.1 Manipulation Check

We designed our instructional interventions to lessen the cognitive demands on participants, particularly when working on more complex queries. At the end of each task, we asked participants to report the perceived difficulty of the task using a Likert-scale measure (1 = lowest difficulty; 5 = highest difficulty). CLT studies have used subjective measures of perceived task difficulty as a proxy for overall cognitive load (e.g., Ayres, 2006), although the number of items and verbal labels on the scales varies across these studies (Ayres & Paas, 2012). We used this measure to check whether participants actually perceived the treatments designed to reduce cognitive load as less difficult. Specifically, we expected the following:

1. That participants would perceive tasks whose solutions had lower Halstead complexity scores as less difficult than tasks whose solutions had higher Halstead scores.
2. That participants would perceive tasks with pseudo SQL requests as less difficult than those with manager English requests.
3. That participants would perceive tasks that included the query template as less difficult that those without the template.
4. That query complexity and request language would interact such that participants would perceive pseudo SQL requests as less difficult than manager English requests only when query complexity is high.
5. That query complexity and template usage would interact such that participants would perceive template usage as increasing difficulty when complexity was low and decreasing perceived difficulty when complexity was high.

We tested the effect of the three independent variables on perceived difficulty using repeated measures multivariate analysis of variance (MANOVA). Table 1 shows descriptive statistics by treatment factors. Table 2 shows the results for within-subjects effects.

Table 1. Descriptive Statistics for Perceived Difficulty by Treatment

			Perceived difficulty	
			Mean	Mean
Low complexity	Low ambiguity	No template	2.10	1.12
		Template	2.30	1.23
	High ambiguity	No template	1.98	1.13

Table 1. Descriptive Statistics for Perceived Difficulty by Treatment

		Template	2.75	1.06
High complexity	Low ambiguity	No template	3.73	0.99
		Template	3.49	1.08
	High ambiguity	No template	3.46	1.11
		Template	3.60	0.91
Low complexity			2.28	0.82
High complexity			3.57	0.77
Low ambiguity			2.90	0.78
High ambiguity			2.95	0.69
No template			2.82	0.69
Template			3.04	0.74
Overall			2.93	0.66

Table 2. Within-subject Effects on Perceived Difficulty

Source	F	Sig.
Query complexity	135.71	**0.000**
Request language	0.31	0.582
Template usage	9.79	**0.003**
SQL proficiency	0.76	0.387
GPA	0.38	0.540
Query complexity*request language	2.85	0.097
Query complexity*template usage	16.26	**0.000**
Query complexity*SQL proficiency	0.01	0.939
Query complexity*GPA	1.28	0.263
Request language*template usage	8.61	**0.005**
Request language*SQL proficiency	0.54	0.467
Request language*GPA	1.00	0.322
Template usage*SQL proficiency	0.20	0.657
Template usage*GPA	2.26	0.138
SQL proficiency*GPA	0.73	0.398
Note: For SQL Proficiency, GPA, and SQL Proficiency * GPA, the effects are between subject.		

As Table 2 shows, participants perceived the tasks whose solutions had higher Halstead scores as more difficult than those with lower Halstead scores (F = 135.71, p < 0.000). Template usage had a significant main effect on perceived difficulty (F = 9.79, p < 0.003) and a significant interaction effect with query complexity (F = 16.26, p < 0.000), consistent with our expectations. Figure 5 illustrates this two-way interaction: the template increased the perceived difficulty of low-complexity queries but decreased the difficulty of high-complexity queries.

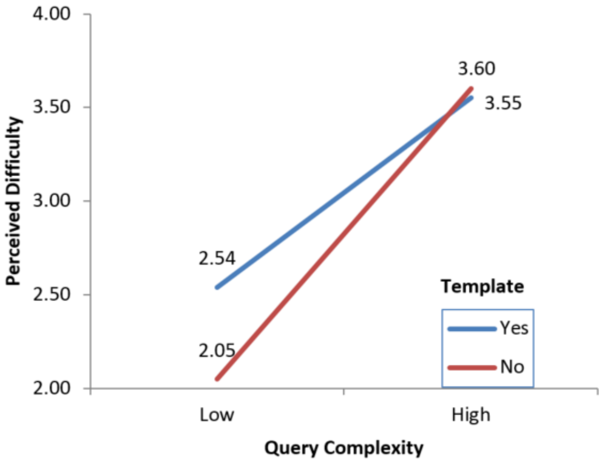


Figure 5. Two-way Interaction between Query Complexity and Template Usage on Perceived Difficulty

Information request language, however, did not have the expected direct effect on perceived difficulty nor the intended interaction effect with query complexity. Our participants did not perceive the pseudo SQL requests as less difficult than their manager English counterparts. In addition, there was a significant interaction between request language and template usage, although we had no a priori expectation about this effect. As Figure 6 shows, the participants perceived pseudo SQL tasks as having similar difficulty with or without the template but the manager English tasks as *more* difficult when the template was also part of the task. We discuss possible reasons for these findings in Section 5.4. As Table 2 shows, the participant characteristics of SQL proficiency and GPA had no direct or interaction effects on perceived task difficulty.

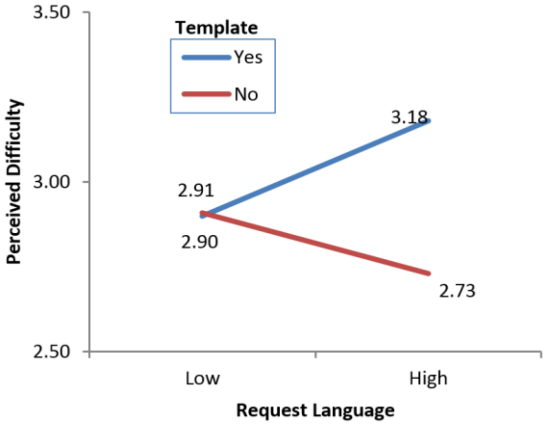


Figure 6. Two-way Interaction between Request Language and Template Usage on Perceived Difficulty

5.2 Main Effects (Hypotheses 1-3)

We tested the effect of our three independent variables on query performance using repeated measures multivariate analysis of variance (MANOVA). Table 3 shows descriptive statistics for each performance measure by the treatment factors. Tables 4 and 5 show the within-subjects main effects and interaction effects, respectively. In addition to the three independent variables of interest in our study, Table 4 also shows GPA and self-reported SQL proficiency. We expected these variables to be positively related to our dependent variables, although they were not the focus of our study. Both GPA and SQL proficiency had a significant main effect on query accuracy, time, and confidence. In addition, Table 5 shows that GPA and SQL proficiency had significant interaction effects with complexity on query accuracy. Further analysis showed that the higher GPAs and SQL proficiency levels had stronger positive effect on accuracy when query complexity was high versus when query complexity was low. Table 5 shows all of the two-way interaction effects completeness’ sake, although only the first two two-way interactions are of interest for this study. None of the three-way, four-way, or five-way interactions were significant or relevant to our hypotheses, so Table 5 does not show them.

Table 3. Descriptive Statistics for Task Performance Variables by Treatment

			Tries		**Time**		**Accuracy**		**Confidence**	
			Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
Low complexity	Pseudo-SQL	No template	4.27	3.60	303.54	182.24	0.97	0.08	4.37	0.99
		Template	5.27	4.63	464.78	256.96	0.96	0.09	4.38	0.94
	Manager English	No template	6.60	7.32	419.57	369.86	0.88	0.16	4.22	1.24
		Template	7.11	6.65	566.14	286.89	0.84	0.13	3.86	1.38
High complexity	Pseudo-SQL	No template	10.43	8.40	753.51	389.14	0.80	0.21	2.78	1.48
		Template	6.97	6.46	781.00	372.11	0.88	0.16	3.30	1.48
	Manager English	No template	8.35	6.05	774.65	346.97	0.81	0.18	2.92	1.43
		Template	8.02	6.39	811.86	387.54	0.83	0.19	3.05	1.36
Low complexity			5.81	3.46	438.51	160.07	0.91	0.09	4.21	0.87
High complexity			8.44	4.34	780.25	192.22	0.83	0.16	3.01	1.09
Pseudo SQL request language			6.73	3.82	575.71	187.26	0.90	0.12	3.71	0.88
Manager English request language			7.52	4.14	643.06	218.31	0.84	0.13	3.51	0.87
No template			7.41	4.05	562.82	151.64	0.87	0.13	3.57	0.84
Template			6.84	3.71	655.94	171.95	0.88	0.11	3.65	0.88
Overall			7.13	3.32	609.38	137.15	0.87	0.12	3.61	0.80

Attempts: The number of times the participant executed a query, including the final submission for a task.
Time: The elapsed time, in seconds, from the query editor or template activation until the participant submitted the final query.
Accuracy: Out of 100%.
Confidence: On a scale of 1 (not at all confident) to 5 (very confident)

Table 4. Within-Subjects Main Effects on Task Performance & Other Variables

Source	**Measure**	**F**	**Sig.**
Query complexity	Tries	25.81	**0.000**
	Time	145.39	**0.000**
	Accuracy	43.62	**0.000**
	Confidence	25.81	**0.000**
Request language	Tries	1.80	0.184
	Time	3.49	0.067
	Accuracy	33.43	**0.000**
	Confidence	4.25	**0.044**
Template usage	Tries	1.23	0.272
	Time	17.61	**0.000**
	Accuracy	2.12	0.151
	Confidence	1.02	0.318
SQL proficiency	Tries	3.15	0.081
	Time	4.05	**0.049**
	Accuracy	12.10	**0.001**
	Confidence	22.24	**0.000**
GPA	Tries	0.03	0.876
	Time	5.35	**0.024**
	Accuracy	20.03	**0.000**
	Confidence	5.58	**0.021**

Note: For SQL proficiency & GPA, the effects are between subject.

Table 5. Within-Subjects Interaction Effects on Task Performance Variables

Source	Measure	F	Sig.
Query complexity * request language	Tries	5.96	**0.018**
	Time	2.26	0.139
	Accuracy	27.12	**0.000**
	Confidence	2.46	0.122
Query complexity * template usage	Tries	9.39	**0.003**
	Time	7.07	**0.010**
	Accuracy	23.11	**0.000**
	Confidence	10.09	**0.002**
Query complexity * SQL proficiency	Tries	0.04	0.840
	Time	0.03	0.854
	Accuracy	25.49	**0.000**
	Confidence	1.87	0.177
Query complexity * GPA	Tries	0.07	0.800
	Time	1.19	0.280
	Accuracy	13.66	**0.000**
	Confidence	2.26	0.138
Request language * template usage	Tries	1.98	0.164
	Time	0.03	0.859
	Accuracy	11.17	**0.001**
	Confidence	5.24	**0.026**
Request language * SQL proficiency	Tries	0.02	0.899
	Time	1.19	0.280
	Accuracy	0.82	0.369
	Confidence	0.56	0.458
Request language * GPA	Tries	0.16	0.692
	Time	0.03	0.872
	Accuracy	1.40	0.241
	Confidence	0.00	0.963
Template usage * SQL proficiency	Tries	1.23	0.272
	Time	0.06	0.802
	Accuracy	0.04	0.849
	Confidence	0.01	0.917
Note: For SQL Proficiency * GPA, the effects are between subject.			

As expected, query complexity had a significant effect on accuracy (F = 43.62, p < 0.000), time (F = 145.39, p < 0.000), attempts (F = 25.81, p < 0.000), and confidence (F = 70.33, p < 0.000), which supports H1. The queries with the higher Halstead score were less accurate, took longer to write, and involved more query attempts than the queries with the lower Halstead score. Participants were also less confident in the quality of their final queries.

Request language had a significant effect on accuracy (F = 33.43, p < 0.000) and confidence (F = 4.25, p < 0.044) but not on the time to complete tasks (F = 3.49, p < 0.067) or on the number of attempts (F = 1.80, p < 0.184). Participants with pseudo SQL requests produced higher-quality queries and were more confident in their queries but were no more efficient completing the task in terms of time or the number of attempts than with manager English requests. These results demonstrate partial support for H2.

Template usage had a significant effect on time (F = 17.61, p < 0.000) such that it increased the total task time as compared to no-template tasks. Template usage had no significant effect on accuracy (F = 2.12, p < 0.151), attempts (F=1.23, p < 0.272), or confidence (F=1.02, p < 0.318). Thus, these results do not supportH3, and the benefits of template usage may only be relevant when combined with other factors (which we discuss below).

5.3 Interaction Effects (Hypotheses 4-5)

Table 5 shows the interaction effects between treatments on the dependent variables. H4 states that pseudo SQL requests have a more positive effect on performance as query complexity increases. The complexity-request language interaction was significant for two of our performance measures: query accuracy (F = 27.12, p < 0.000) and the number of attempts (F = 5.96, p < 0.018). However, the interaction effect was *opposite* of what we predicted. As Figure 7 shows, pseudo SQL requests improved query accuracy for low-complexity tasks (97% accuracy compared to 86% accuracy with managerial-worded requests) but provided little benefit for high-complexity tasks. In addition, participants had fewer query attempts with pseudo SQL when task complexity was low but had more attempts than participants with manager English requests when task complexity was high. Thus, the results do not support H4. Pseudo SQL requests in our study were helpful when the queries involved straightforward SELECT-FROM-WHERE clauses but not helpful when the queries also required aggregation, GROUP BY, and HAVING clauses.

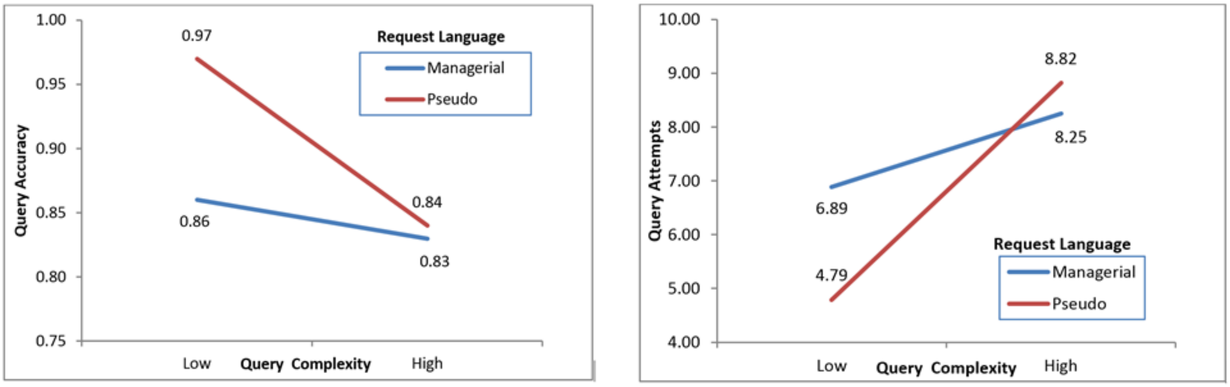


Figure 7. Two-way Interactions between Query Complexity and Request Language on Query Accuracy and on Attempts

H5 states that template usage is detrimental to task performance when complexity is low but is largely beneficial to performance when complexity is high. The complexity-template interaction was significant with respect to all four dependent variables: accuracy (F = 23.11, p < 0.000), time (F = 7.07, p < 0.010), attempts (F = 9.39, p < 0.003), and confidence (F = 10.09, p < 0.02). Figure 8 shows the interaction effects on each dependent variable.

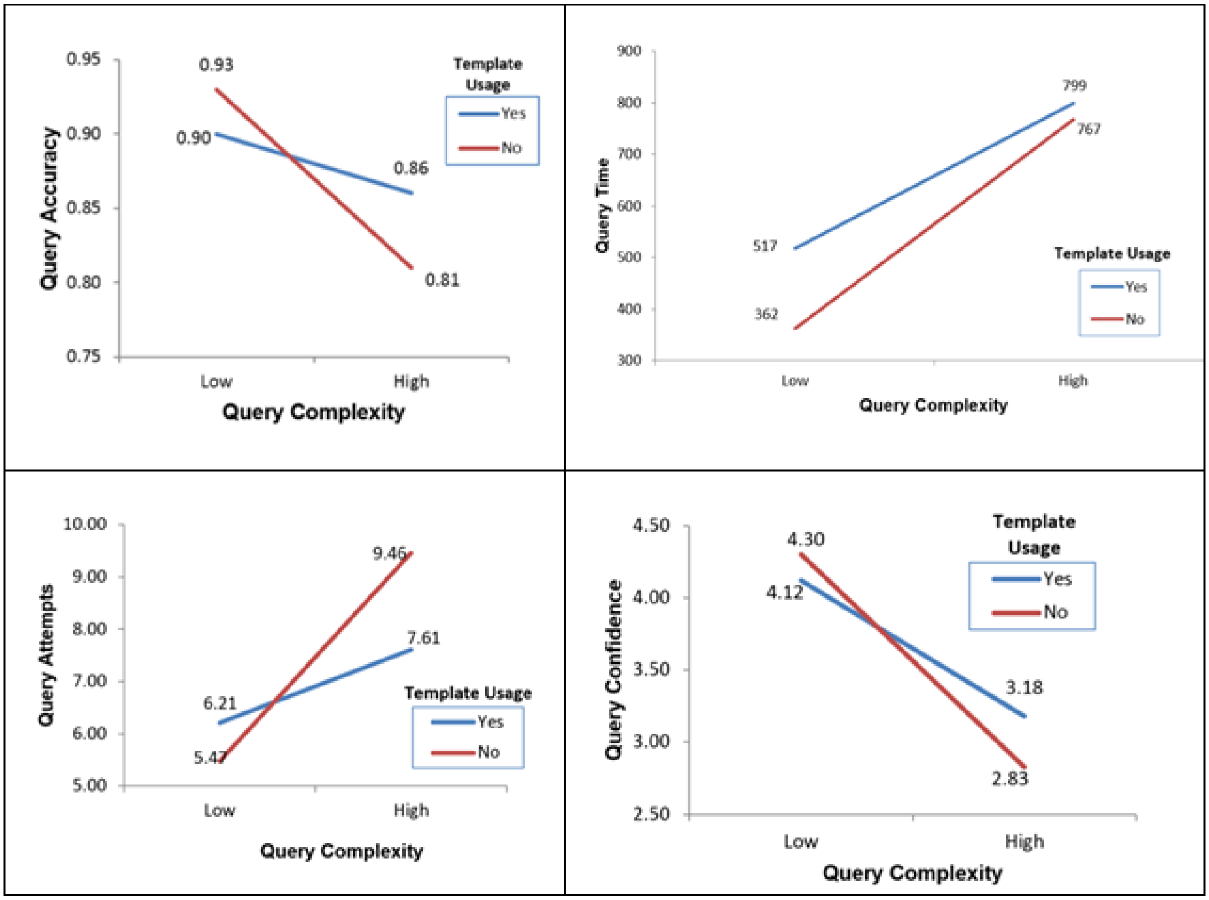


Figure 8. Two-way Interactions between Query Complexity and Template Usage on each of the Four Dependent Variables

Figure 8 shows that template usage significantly increased the total time spent on the task when complexity was low but did not impact time as much when complexity was high compared to no template usage. For the other dependent variables—accuracy, attempts, and confidence—using the template *hindered* performance when complexity was low but *improved* performance when complexity was high compared to having no template. That is, when the queries required only SELECT-FROM-WHERE clauses, participants made fewer attempts, produced more-accurate queries, and had greater confidence without the template than with it. But, when the queries required aggregation, GROUP BY, and HAVING elements, the participants made fewer attempts, produced more accurate queries, and had greater confidence with the template than without. These results support H5.

Table 6 summarizes the results of our hypothesis testing. Query complexity had a significant effect on task performance, consistent with prior research (Borthick et al., 2001; Bowen et al., 2009; Chan et al., 1999). Increased clarity through the pseudo SQL request language had a positive effect on query accuracy and confidence. More interesting, however, were the interaction effects of request language and template usage with task complexity. Pseudo SQL request language was beneficial—in terms of accuracy and attempts—when task complexity was low. Template usage was beneficial—in terms of accuracy, attempts, and confidence—when task complexity was high.

Table 6. Summary of Hypothesis Testing

Hypothesis	Expected effect on dependent variables	Findings
H1: (Main effect) Query complexity negatively impacts task performance.	Comparing higher and lower query task complexity, higher complexity → lower query accuracy (H1a) greater time (H1b) more attempts (H1c), and lower confidence (H1d).	Support for H1 on all dependent variables.
H2: (Main effect) Pseudo SQL requests improve task performance.	Comparing pseudo-SQL to manager English requests, pseudo-SQL requests → higher query accuracy (H2a) less time (H2b) fewer attempts (H2c), and higher confidence (H2d).	Support for H2 on accuracy (H2a) and confidence (H2d) only.
H3: (Main effect) Template usage improves task performance.	Comparing template usage to no-template tasks, template usage → higher query accuracy (H3a) no difference in time (H3b) fewer attempts (H3c), and higher confidence (H3d).	Not supported.
H4: (Interaction effect) Pseudo SQL requests have a stronger positive effect on performance when complexity is high versus when complexity is low.	Comparing pseudo-SQL to manager English request language as query complexity increases, pseudo SQL requests → higher accuracy (H4a) less time (H4b) fewer attempts (H4c), and higher confidence (H4d).	Contrary to expectations, pseudo SQL provided a benefit (on the number of tries and accuracy) only when task complexity was *low*. H4 not supported.
H5: (Interaction effect) Template usage has a negative effect on task performance when complexity is low, but a (mostly) positive effect on performance when complexity is high.	When complexity is low, template usage → lower accuracy, less confidence, more time, and more attempts than not having the template, but when complexity is high, template usage → greater accuracy (H5a) no difference in time (H5b) fewer attempts (H5c), and higher confidence (H5d).	Support for H5 on all dependent variables except time.

5.4 Post Hoc Analyses

We had no a priori expectations about a two-way interaction between request language and template usage or about a three-way interaction between language, template, and complexity. While we found no significant three-way interactions, Table 5 does show a significant interaction between request language and template usage on query accuracy and confidence. Specifically, template usage resulted in higher accuracy with pseudo SQL wording but had no differential effect with manager English wording. Similarly, template usage produced higher confidence when combined with pseudo SQL wording and appeared to lower confidence when combined with manager English wording (Figure 9). Increased clarity through pseudo SQL wording had a positive overall effect on accuracy and confidence, and this positive effect was most evident when the task included the template or when query complexity was low. The template had a neutral or slightly negative effect on accuracy and confidence with manager English requests. In addition, there was an interaction effect on perceived difficulty such that the participants perceived manager English requests combined with the template as more difficult than pseudo SQL requests with or without the template. These findings indicate a need for future investigations into the interaction of instructional techniques and the situations in which certain combinations are effective (or not).

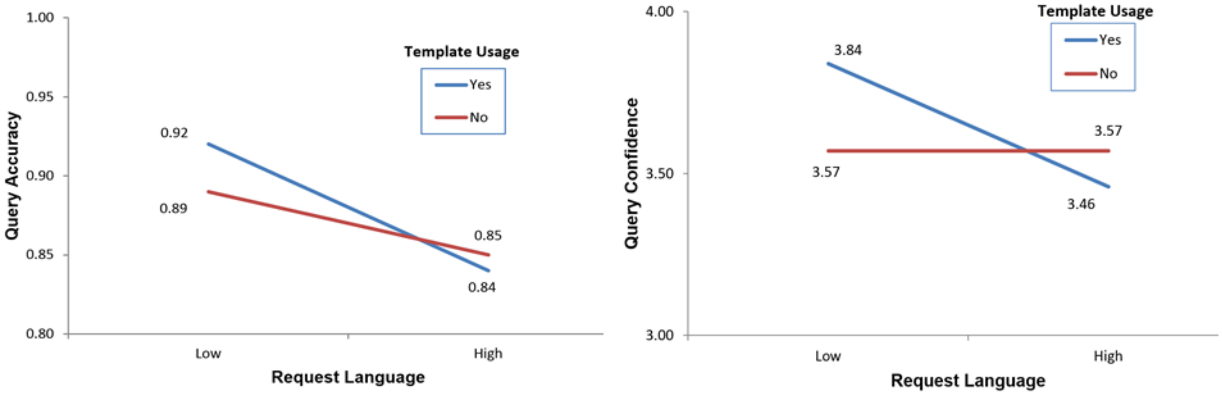
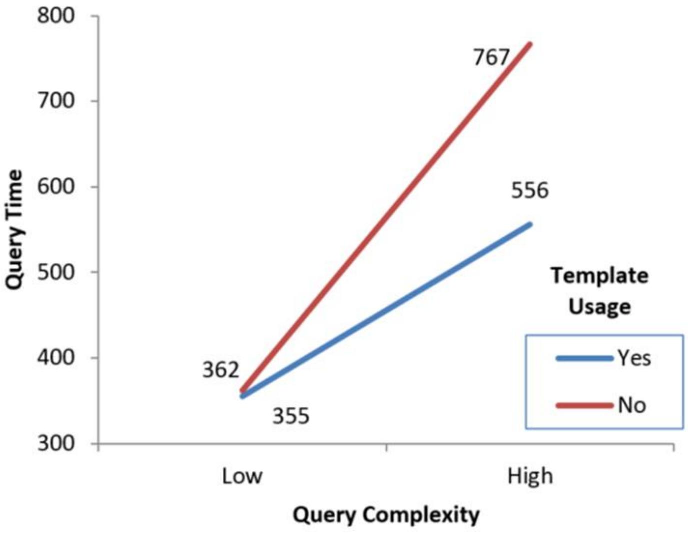


Figure 9. Two-way Interactions between Request Language and Template Usage on Accuracy and Confidence.

Since the template intervention is new in studies of query formulation, we explored its use in further detail, specifically with respect to the time spent on the overall task and to the quality of the template contents.

With respect to time, our hypothesis testing shows that the template increased the total time that participants spent on tasks when compared to not having a template. However, we were also interested in whether the time spent on the template—essentially planning what code to write—reduced the time spent writing, executing, and revising code in the query editor window. The template tasks required participants to enter notes into the template before they could use the query editor to write their SQL statements. We separated the template task time into two parts—time spent on the template and time spent in the query editor—and compared the time spent in the query editor to the time spent (in the query editor) in the no-template condition. When we re-ran the analysis with this alternative time measure, we found that using the template significantly reduced the time spent in the query editor (F = 27.81, p. < 0.000). The interaction effect of template usage with query complexity was also significant (F = 21.72, p < 0.000). As Figure 10 shows, the template condition did not differ much from the no-template condition when query complexity was low, but the time spent writing and revising code in the query editor was significantly lower in the template condition versus the no-template condition when query complexity was high. Time spent completing the template had some benefit in terms of reducing the time spent coding and debugging, but the benefit was not sufficient to reduce the overall time spent on task. We need future studies that improve the efficiency of the template portion of the task (e.g., by making the template contents executable or automatically transferring the template contents to the query editor) to determine whether the time invested in planning the query can pay off in reduced overall task time.



Note: Query time is measured as the time spent in the query editor window (i.e., excluding time spent in the template for the template-task condition).

Figure 10. Post Hoc Analysis of Interaction between Query Complexity and Template Usage on Time

With respect to template quality, we examined the four experimental conditions that included the template: the low complexity/pseudo SQL, low complexity/manager English, high complexity/pseudo SQL, and high complexity/manager English conditions. We analyzed and graded each participant's template using a scale of 1 to 6 to reflect the number of clauses in the template (see Figure 1) with accurate notes or comments. We used repeated measures multivariate analysis to test the effect of query complexity and request language on this template quality variable. As Table 7 shows, both complexity and request language had significant effects on template quality. Specifically, template quality was significantly higher for low-complexity queries versus high-complexity queries and with pseudo SQL requests versus manager English requests. Furthermore, as Table 8 shows, there was a significant positive correlation between template quality and the accuracy of the final query solutions across all treatments. The factors that improved query formulation accuracy also improved template accuracy, and better templates were associated with better final solutions.

Table 7. Within-Subject Effects on Template Quality

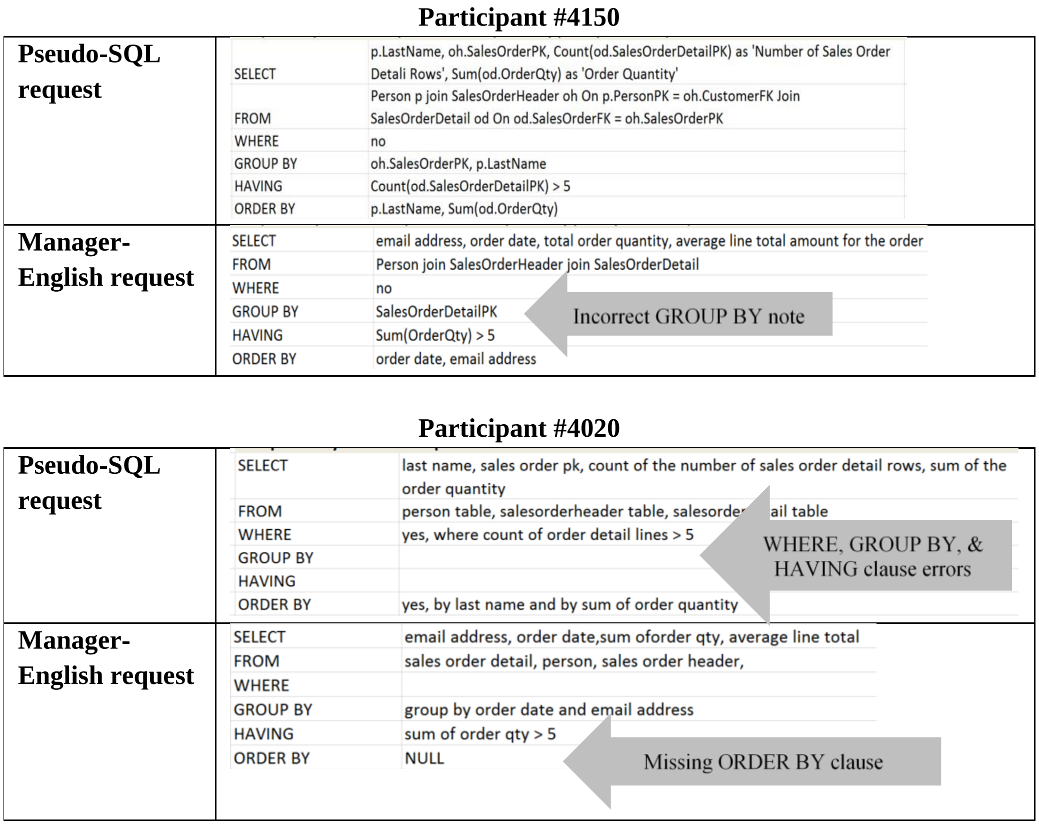
Source	F	Sig.
Query complexity	100.69	**0.000**
Request language	20.83	**0.000**
Query complexity * request language	0.61	0.439

Table 8. Correlations Between Template Quality and Query Accuracy

Source		Pearson r	Sig.
Low complexity	Pseudo SQL	0.32	**0.004**
	Manager English	0.28	**0.012**
High complexity	Pseudo SQL	0.53	**0.000**
	Manager English	0.43	**0.000**

To explore *how* the participants used the template in high-complexity conditions, we reviewed the template and query attempt data for several participants in the pseudo SQL/template and the manager English/template conditions. Figure 11 shows the template contents for two of these participants: #4150 and #4020. Participant #4150 performed well on all of the query tasks. His template for the high-complexity, pseudo SQL task was detailed and free from errors, and his first attempt at coding the query was also free from errors. This same participant, with the manager English request, wrote fewer details in the template and had one error. His first coding attempt included the same error as in the template, but he was able to correct this error by the fourth attempt. Other participants exhibited a similar pattern to #4150 in that the template contained more detail for the pseudo SQL request than the manager English request and the first coding attempt closely paralleled the template contents.

Other participants, such as #4020, made template notes at about the same level of detail regardless of which request language they used. Participant #4020's first coding attempt also closely paralleled his template notes. As Figure 11 shows, his template for the manager English task had fewer errors than for the pseudo SQL task. For the manager English task, he incorrectly noted that the query did not need ORDER BY but correctly commented on the other clauses. By his third attempt, which was his final attempt, he had corrected the ORDER BY oversight but had a misspecified GROUP BY clause, which the template correctly noted. His final query was 93 percent accurate. This same participant had several errors in his template for the pseudo SQL task, most of which were evident in his initial coding attempts. By the eleventh and final attempt, he had corrected some of the problems that began with the template but had several misspecified clauses and incorrect aggregate functions. His final query was 81 percent accurate.



Participant #4150

Pseudo-SQL request	SELECT	p.LastName, oh.SalesOrderPK, Count(od.SalesOrderDetailPK) as 'Number of Sales Order Detali Rows', Sum(od.OrderQty) as 'Order Quantity'
	FROM	Person p join SalesOrderHeader oh On p.PersonPK = oh.CustomerFK Join SalesOrderDetail od On od.SalesOrderFK = oh.SalesOrderPK
	WHERE	no
	GROUP BY	oh.SalesOrderPK, p.LastName
	HAVING	Count(od.SalesOrderDetailPK) > 5
	ORDER BY	p.LastName, Sum(od.OrderQty)
Manager-English request	SELECT	email address, order date, total order quantity, average line total amount for the order
	FROM	Person join SalesOrderHeader join SalesOrderDetail
	WHERE	no
	GROUP BY	SalesOrderDetailPK
	HAVING	Sum(OrderQty) > 5
	ORDER BY	order date, email address

Incorrect GROUP BY note

Participant #4020

Pseudo-SQL request	SELECT	last name, sales order pk, count of the number of sales order detail rows, sum of the order quantity
	FROM	person table, salesorderheader table, salesorder ail table
	WHERE	yes, where count of order detail lines > 5
	GROUP BY	
	HAVING	
	ORDER BY	yes, by last name and by sum of order quantity
Manager-English request	SELECT	email address, order date,sum oforder qty, average line total
	FROM	sales order detail, person, sales order header,
	WHERE	
	GROUP BY	group by order date and email address
	HAVING	sum of order qty > 5
	ORDER BY	NULL

WHERE, GROUP BY, & HAVING clause errors

Missing ORDER BY clause

Figure 11. Example Templates from Participants in High Complexity Conditions.

We need further research into the type and frequency of template errors under different experimental conditions and into the process by which individuals revise queries in response to error messages. From reviewing the template notes of our participants, we observe that many of them engaged with the template in a dialog style: that is, they noted “yes” or “no” for the optional clauses to indicate whether the query would need the clause or not. To the extent that engaging with the template promotes germane cognitive load, research into ways to further encourage this problem-solving dialog may also prove fruitful. For example, one could add additional prompts to help participants determine what to specify in a clause once they determine that that the query needs the clause.

6 Discussion

Formulating queries is a complex task that involves transforming natural-language information requests into the specific structure of SQL SELECT statements based on a database’s structure and contents and while constrained by the query language’s syntax. We investigated two instructional interventions to facilitate query-formulation task performance under two levels of query complexity. We focused on education and training environments with the long-term goal of increasing query-writing proficiency in a given time period, such as in the confines of a university database course.

Results of our experiment support the main effect of program complexity on task performance, which, while not surprising, is noteworthy given the wide range of complexity measures prior research has used. Prior studies have differentiated low- from high-complexity queries based on the number of tables involved (Rho & March, 1997), lines of code (Chan, 1999), or Halstead difficulty scores (Borthick et al., 2001; Casterella & Vijayasarathy, 2013). We used Halstead’s difficulty metric to create two clearly differentiated task categories, such that our high-complexity queries had more than double the Halstead score of the low-complexity queries. Future investigations with a wider range of program complexity would further illuminate the effectiveness of instructional interventions. For example, our tasks did not require nested queries (subqueries), which, in our experience, are at least as difficult for students as queries with GROUP BY clauses. Halstead’s (1977) difficulty metric does not consider the level of nesting in a program, although nesting may increase cognitive load (Curtis et al., 1979). We may need other measures of program complexity to broaden the range of query tasks investigated.

For our first instructional intervention, we increased information-request clarity through pseudo SQL wording. In practice, query writers will not have requests for information presented in pseudo SQL. In a training environment, however, pseudo SQL requests do show some promise. We based this intervention on CLT studies of part-task sequencing training (Gerjets et al., 2004; Pollack et al., 2002; Wickens et al., 2013) in which one artificially reduces the complexity of one part of a task until learners gain experience performing other parts. Pseudo SQL is also an intervention that is noticeable in some database management textbooks, particularly in early examples of query formulation, where information requests sometimes adopt phrasing to clearly indicate the table(s), conditions, and/or clauses needed in the solution (e.g., Hoffer et al., 2012, Coronel & Morris, 2014). We designed our pseudo SQL requests to simplify the process of mapping elements of the request to elements in the data model, which is a necessary precursor to writing the query (Reisner, 1977; Borthick et al., 2001; Siau & Tan, 2006). Our pseudo SQL requests also provided clues about which clauses of the SELECT statement a query needed (e.g., using the words “from”, “order by”, and “having” to indicate the need for clauses beginning with those words).

Consistent with our hypotheses and prior studies, participants with pseudo SQL requests wrote higher-quality queries and were more confident than those with manager English requests (Borthick et al., 2001; Casterella & Vijayasarathy 2013). However, the pseudo SQL requests did not reduce query-writing time or attempts, were not perceived as less difficult than manager English requests, and, contrary to our hypothesis, improved accuracy only when query complexity was *low*, not high. Borthick et al. (2001) did not investigate the interaction between query complexity and request language. Casterella and Vijayasarathy (2013) found that pseudo SQL requests had a more pronounced and positive impact on query accuracy when queries were *more* complex rather than less complex. However, their most complex queries had a Halstead difficulty score of 20, which is in between our low-complexity queries (Halstead of 12.5) and our high-complexity queries (Halstead score of 33). It may be that pseudo SQL language is most helpful when queries are within a certain mid-level range of difficulty. While it would be consistent with CLT to find that pseudo SQL language is an unnecessary instructional aid for simple SELECT-FROM queries, it is not as clear why pseudo SQL requests did not significantly improve performance with our high-complexity tasks.

One explanation for the negative impact of pseudo SQL requests on high-complexity queries is that the requests’ length may have confounded their clarity. Pseudo SQL requests clarified the mappings to data model elements, which, however, made them considerably longer than those written in manager English. A post hoc calculation showed that the pseudo SQL requests contained, on average, 63 percent more words than the manager English requests (57 and 35 words per request, respectively), and the more complex queries had longer requests than the simpler queries. We do not know about any prior research that has investigated the effect of information request length on query-formulation task performance. However, prior studies have investigated the effect of data model clarity on query formulation (e.g., Bowen et al. 2009) in which data model diagrams with greater ontological clarity were also larger (e.g., more entities and relationships) than less ontologically clear data models. Bowen et al. (2009) found that greater clarity in data models improved query task performance, but only to a point. After that point, the increased size of the clearer models had a detrimental effect on performance. It may also be that there is a clarity-size trade-off with information requests and a non-linear relationship between the clarity of the information request and task performance.

A related issue is that the pseudo SQL requests—with their increased length and specific table and column names—may have prompted participants to cross-reference between the request and the data model more during the task, which may have split their attention to a point that was detrimental. This situation would be exacerbated in the high-complexity conditions, which explicitly mentioned more tables and columns than compared to the low-complexity conditions, and would also be exacerbated in the template conditions, which placed an additional demand on participants’ attention. CLT studies have shown a negative relationship between split-attention and task performance (see van Merrienboer & Sweller, 2005, for a review), which may partially explain the unanticipated interaction effects between request clarity and template usage.

We need more research on interventions that change the initial presentation of the query task to effectively reduce the cognitive load in the initial part of query formulation (i.e., the information-request-to-data-model mapping). For example, instead of increasing the clarity of required data model elements through the (longer) wording of the information request, one could increase clarity through a more complete data dictionary or by using color and highlighting to draw attention to relevant portions of the data model diagram. One could superimpose requests on the data model diagram to reduce cross-referencing. We recommend that future studies include measures of the different types of cognitive load (intrinsic, extraneous, and germane) so that we may understand the impact of instructional materials on, in particular, extraneous and intrinsic load. While not published at the time of our data collection, CLT researchers recently developed a survey instrument to measure each type of cognitive load and successfully tested the instrument with statistics and language tasks (Leppink, Paas, Van der Vlueten, Van Gog, & Van Merrienboer, 2013; Leppink, Paas, Van Gog, Van der Vleuten, & Van Merrienboer, 2014). One could adapt and test this instrument in the context of query formulation.

As learners become more proficient writing queries with specific SQL syntax, they should free up their cognitive resources, and they should not need training interventions such as pseudo SQL as much. As training progresses, one should present information requests to learners with increasingly realistic levels and types of ambiguity. For example, our manager English requests were less clear than our pseudo SQL requests with respect to which columns and tables to include in the solution but were reasonably clear in terms of their English meaning. However, natural-language information requests have much ambiguity (see Ashkanasy et al., 2007, for a brief review). We need future research to understand the cognitive demands placed on learners as the gap between the information request and the data model increases and to identify strategies that incrementally and adaptively widen this gap as the learner's skill set expands. For example, one strategy might be to have query writers themselves translate ambiguous natural language requests into pseudo SQL requests as an intermediate problem-solving step (Borthick et al., 2001).

For our second intervention, we provided a query template for participants to use before they began writing SQL statements. Database management textbooks often use partial query templates to introduce the SELECT statement (e.g., Hoffer et al., 2012; Coronel & Morris, 2014), although they typically do not include the template as the queries become more complex. We created our query template, which included all of the SELECT statement clauses, due to CLT research findings that tasks with high coordinative complexity can benefit from a simplified-whole task strategy (Gerjets et al., 2004; van Merrienboer & Swell, 2005). With this strategy, learners see the big picture of the task but with certain aspects of the task artificially simplified until they acquire more experience. The query template outlines the structure of the SELECT statement with prompts the learner to help fill in the details and builds on Reisner's (1977) and Borthick et al.'s (2001) characterization of query formulation as a mapping-and-template-filling or multiple-transformation task. We designed the template to simplify the task because one does not need to recall the keywords, sequencing, and purpose of the clauses from memory, and the reminders indicate the inter-relatedness of the clauses. It provides "scaffolding" (Wood et al., 1976)—an instructional aid designed to help learners accurately formulate queries that might otherwise be too cognitively demanding or beyond their developing skill set.

Consistent with our hypotheses, we found that template usage improved performance for more complex tasks and had a minimal or negative effect for simpler tasks. We also found that participants perceived the template to increase the difficulty of simpler queries and to decrease the difficulty of complex queries. This finding agrees with CLT research on the expertise-reversal effect (Kalyuga et al., 2003): that an intervention may increase *extraneous* load for easier tasks but decrease *intrinsic* load for more complex tasks. These findings suggest that the template has some promise as a training tool for intermediate-level novices learning more complex queries that combine multiple joins, aggregate functions, and grouping columns with basic SELECT-FROM-WHERE queries.

However, we do not know whether the template would provide a useful scaffold for less (or more) complex queries at an earlier (or later) point in the SQL skill-development process. To understand the range of situations in which the template helps learners, future research should include the impact of the template on the different types of cognitive load (Leppink et al., 2013). If, in fact, the template impacts cognitive load in the manner that the expertise-reversal effect suggests, then instructional design research could focus on how improvements in the template's design/implementation or changes to the situations in which one provides the template affect the types of cognitive load.

One direction for instructional design is to improve the implementation of the template in the training environment. For example, one could integrate the template-use and query-writing activities by making the template contents executable or having the template contents transfer automatically to the query editor, which might lessen the perceived difficulty we observed when participants used the template for low-complexity queries by making the template a more natural part of the query-formulation task rather than a separate precursor to it. Another direction is to evaluate the design and contents of the template itself. The template we used is an annotated outline of a complete SELECT statement (Figure 1) and, as such, provides the framework for constructing a query solution. However, we do not know the extent to which this template approximates the mental schemas of expert query writers. Studies of expertise and mental schemas, such as those in the domains of chess (e.g., Chase & Simon, 1973) and conceptual data modeling (e.g., Gerard, 2005), can improve the intermediate solution plans or scaffolds that training environments include.

Another direction is to investigate *when* the template is likely to be an effective training aid (or not). Our findings with respect to the template with low- and high-complexity queries are relevant for intermediate-level novices at the end of a typical university database course, but we need further research to better understand the cognitive load for learners at different skill levels. For example, would the template be effective for simpler query tasks if the participants were earlier in their SQL skill-development process? Educational psychology recommends scaffolding to help learners bridge the distance between the problems that they can independently and those that require guidance or assistance (Wood et al., 1976). Researchers have referred to this distance, where the learner does not quite have the level of skill required to successfully accomplish the task independently, as the zone of proximal development (Vygotsky, 1978). It is in this zone that instructional aids such as the query template might be most effective, because, in it, the template will likely reduce the intrinsic load of the task. For one learner, a query task with a certain Halstead complexity score may be easy to successfully complete, while, for another learner, the same task lies in the zone of proximal development. Instructional tools and strategies can benefit if they recognize this difference and adaptively phase the template in or out based on each participant's performance on queries at a certain level of complexity (Kalyuga et al., 2003; Wickens et al., 2013).

Readers should consider this study's limitations when reviewing our findings. First, we used university students (mostly information systems majors) enrolled in an introductory database course. Thus, our findings may not generalize to other populations. However, given that we focused on education and data-retrieval skills, using college students who are likely to need and use these skills in the workplace is reasonable. Second, our sample, while larger than some studies of query formulation (e.g., Casterella & Vijayasarathy, 2013; Borthick et al., 2001), was small compared to other studies (e.g., Allen & March, 2006; Allen & Parsons, 2010). Finally, while the two-hour allotted time for our study was consistent with other studies (e.g., Bowen et al., 2009; Bowen et al., 2006; Borthick et al., 2001), our repeated measures design meant we had to drop many subjects who did not complete all eight of our experimental tasks. Our low-complexity tasks were more complex than the least complex tasks in other studies, which likely increased the time pressure on our participants.

7 Conclusion

Cognitive load theory provides useful insights into instructional techniques that are effective in various domains, but researchers have not widely applied this theory to studying query-formulation tasks. Our study indicates that CLT-informed instructional techniques have promise to help individuals more effectively and efficiently formulate queries with SQL and that CLT is also useful in explaining the situations in which different techniques are effective. Specifically, for the intermediate-level novices in our study, we found that the query template helped students solve *complex* queries with 3-table joins, aggregate functions, and GROUP BY and HAVING clauses. On the other hand, participants performed better with information requests in pseudo SQL rather than manager English for *simpler* queries with 2-table joins and multiple filtering conditions (but no aggregation or grouping). Data-retrieval skills are in high demand, and the expected breadth and depth of those skills are increasing as well. Educators must respond by finding more efficient and effective strategies to guide students through this skill-acquisition process.

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.

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

Axelson, M., Borthick, A. F., & Bowen, P. L. (2001). A model for and the effects of information request ambiguity on end-user query performance. In *Proceedings of the 22nd International Conference on Information Systems* (pp. 537-542).

Ayres, P. (2006). Using subjective measures to detect variations of intrinsic cognitive load within problems. *Learning and Instruction*, *16*, 389-400.

Ayres, P., & Paas, F. (2012). Cognitive load theory: New directions and challenges. *Applied Cognitive Psychology*, *26*, 827-832.

Belland, B. R., Kim, C., & Hannafin, M. J. (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 Association for Information Systems*, *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 formulation tasks. *Journal of Information Systems Education*, *24*(3), 211-221.

Chamberlin, D. C. (2012). Early history of SQL. *IEEE Annals of the History of Computing, 34*(4), 78-82.

Chan, H. C. (1999). The relationship between user query accuracy and lines of code. *International Journal of Human-Computer Studies*, *51*(5), 851-864.

Chan, H. C., Wei, K. K., & Siau, K. L. (1993). The user-database interface: The effect of abstraction levels on query performance. *Management Information Systems Quarterly*, *17*(4), 441-464.

Chandler, P., & Sweller, J. (1996). Cognitive load while learning to use a computer program. *Applied Cognitive Psychology*, *10*, 151-170.

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology*, *5*, 55-81.

Chi, M. T. H., Glaser, R., & Rees, E. (1982). Expertise in problem solving. In R. Sternberg (Ed.), *Advances in the psychology of human intelligence* (pp. 7-75). Hillsdale, NJ: Erlbaum.

Cook, M. P. (2006). Visual representations in science education: The influence of prior knowledge and cognitive load theory on instruction design principles. *Science Education*, *90*(6), 1073-1091.

Coronel, C., & Morris, S. (2014). *Database systems: Design, implementation, and management* (11th ed.). Boston, MA: Cengage Learning.

Curtis, B., Sheppard, S. B., Milliman, P., Borst, M. A., & Love, T. (1979). Measuring the psychological complexity of software maintenance tasks with Halstead and McCabe metrics. *IEEE Transactions on Software Engineering*, *5*(2), 96-104.

Davern, M., Shaft, T., & Te'eni, D. (2012). Cognition matters: Enduring questions in IS research. *Journal of the Association for Information Systems*, *13*(4), 277-314.

Elmsari, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). New York: Pearson.

Gerard, G. J. (2005). The REA pattern, knowledge structures, and conceptual modeling performance. *Journal of Information Systems*, *19*(2), 57-77.

Gerjets, P., Scheiter, K., & Catrambone, R. (2004). Designing instructional examples to reduce intrinsic cognitive load: Molar versus modular presentation of solution procedures. *Instructional Science*, *32*(1), 33-58.

Goel, V., & Pirolli, P. (1992). The structure of design problem spaces. *Cognitive Science*, *16*, 395-429.

Grothendieck, G. (2012). *SQLDF: SQL select on R data frames.* Retrieved from http://code.google.com/p/sqldf/

Halstead, M. H. (1977). *Elements of software science*. Amsterdam: Elsevier.

Hoffer, J. A., Venkataraman, R., & Topi, H. (2012). *Modern database management* (11th ed.). New York: Prentice Hall.

Kalyuga, S., Ayres, P., Chandler, P., & Sweller, J. (2003). The expertise reversal effect. *Educational Psychologist*, *38*(1), 23-31.

Kroenke, D. M., & Auer, D. J. (2013). *Database concepts* (6th ed.). New York: Pearson.

Leppink, J., Paas, F., Van der Vleuten, C. P. M., Van Gog, T., & Van Merrienboer, J. J. G. (2013). Development of an instrument for measuring different types of cognitive load. *Behavioral Research*, *45*(4), 1058-1072.

Leppink, J., Paas, F., Van Gog, T., Van der Vleuten, C. P. M., & Van Merrienboer, J. J. G. (2014). Effects of pairs of problems and examples on task performance and different types of cognitive load. *Learning and Instruction*, *30*, 32-42.

Miller, G. A. (1956). The magical number seven plus or minus two: Some limits in our capacity for processing information. *Psychological Review*, *63*(2), 81-92.

Ogden, W. C. (1986). Implications of a cognitive model of database query: Comparison of a natural language, formal language, and direct manipulation interface. *ACM SIGCHI Bulletin*, *18*(2), 51-54.

Pezzullo, J. C. (2008). *Latin squares for constructing "Williams Designs", balanced for first-order carry-over (residual) effects*. Retrieved from http://statpages.org/latinsq.html

Pollock, E., Chandler, P., & Sweller, J. (2002). Assimilating complex information. *Learning and Instruction*, *12*(1), 61-86.

Pratt, P. J., & Adamski, J. J. (2012). *Concepts of database management* (7th ed.). New York: Cengage Learning.

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.

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*(2), 3-14.

Rith, J., Lehmayr, P. S., & Meyer-Wegener, K. (2014). Speaking in tongues: SQL access to NoSQL systems. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing* (pp. 855-857).

Rockoff, L. (2011). *The language of SQL.* New York: Cengage Learning.

Rosenberger, J. (2014). Germany's secret world cup weapon: Big data. *CIO Insight.* Retrieved from http://www.cioinsight.com/it-news-trends/germanys-secret-world-cup-weapon-big-data.html

Rourke, A., & Sweller, J. (2009). The worked-example effect using ill-defined problems: Learning to recognize designers' styles. *Learning and Instruction*, *19*(2), 185-199.

Siau, K. L., & Tan, X. (2006). Cognitive mapping techniques for user-database interaction. *IEEE Transactions on Professional Communication*, *49*(2), 96-108.

Simon, H. A. (1973). The structure of ill structured problems. *Artificial Intelligence*, *4*, 181-201.

Speier, C., & Morris, M. G. (2003). The influence of query interface design on decision-making performance. *Management Information Systems Quarterly*, *27*(3), 397-423.

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. *Learning and Instruction*, *4*(4), 295-312.

Sweller, J., van Merrienboer, J. J. G., & Paas, F. (1998). Cognitive architecture and instructional design. *Educational Psychology Review*, *10*(3), 251-296.

Van Gog, T., Kester, L., & Paas, F. (2010). Effects of worked examples, example-problem, and problem-example pairs on novices' learning. *Contemporary Educational Psychology*, *36*, 212-218.

van Merriënboer, J. J. G., & Sweller, J. (2005). Cognitive load theory and complex learning: Recent developments and future directions. *Educational Psychology Review*, *17*(2), 147-177.

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes.* Cambridge, MA: Harvard University Press.

Wang, Q., Yang, S., Liu, M., Cao, Z., & Ma, Q. (2014). An eye-tracking study of website complexity from cognitive load perspective. *Decision Support Systems*, *62*, 1-10.

Wickens, C. D., Hutchins, S., Carolan, T., & Cumming, J. (2013). Effectiveness of part-task training and increasing-difficulty training strategies: A meta-analysis approach. *Human Factors*, *55*(2), 461-470.

Williams, E. J. (1949). Experimental designs balanced for the estimation of residual effects of treatments. *Australian Journal of Scientific Research, 2*(3), 149-168.

Wood, D., Bruner, J. S., & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry*, *17*(2), 89-100.

Wood, R. E. (1986). Task complexity: Definition of the construct. *Organizational Behavior and Human Decision Processes*, *37*, 60-82.

Appendix A: Database Model Used for Experiment Tasks

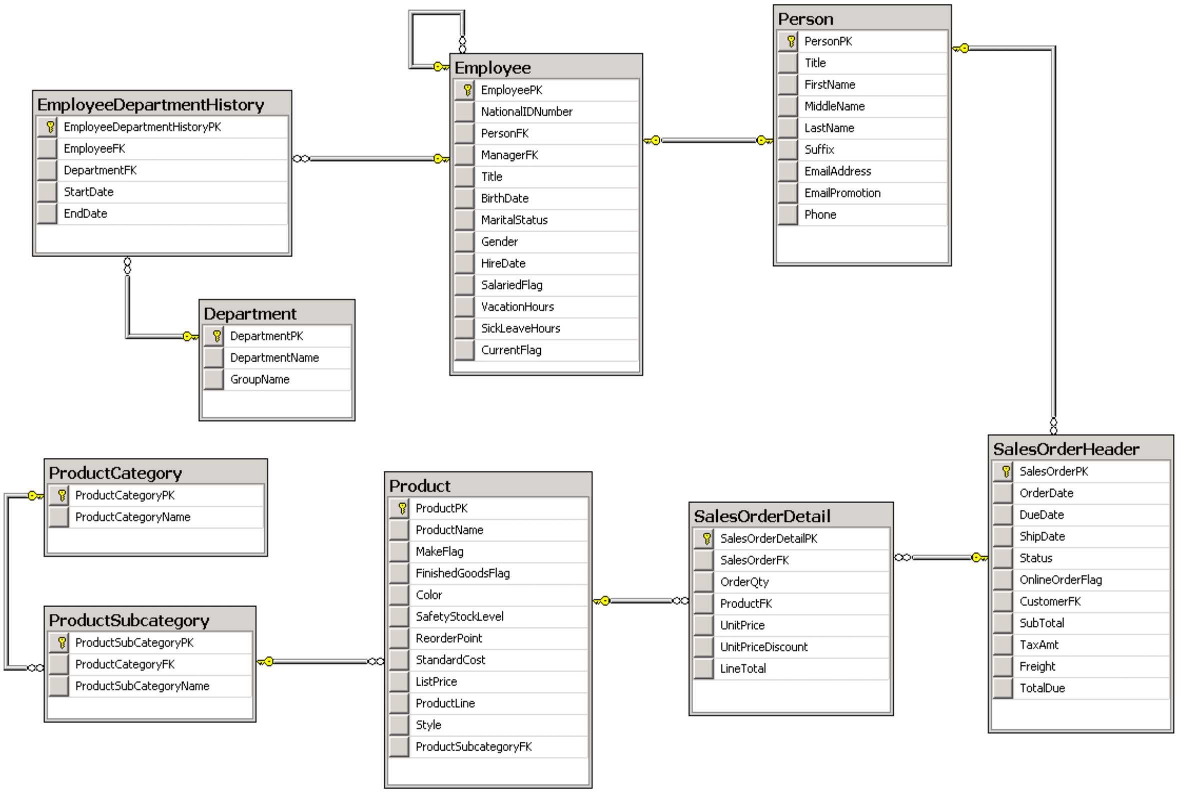
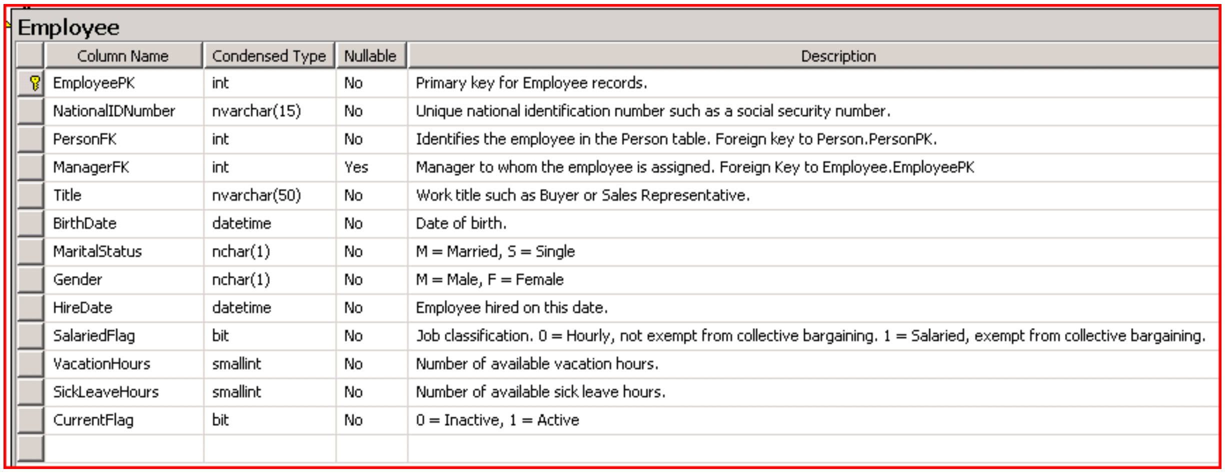


Figure A1. Model of Database Used for Experiment Tasks



Employee

Column Name	Condensed Type	Nullable	Description
EmployeePK	int	No	Primary key for Employee records.
NationalIDNumber	nvarchar(15)	No	Unique national identification number such as a social security number.
PersonFK	int	No	Identifies the employee in the Person table. Foreign key to Person.PersonPK.
ManagerFK	int	Yes	Manager to whom the employee is assigned. Foreign Key to Employee.EmployeePK
Title	nvarchar(50)	No	Work title such as Buyer or Sales Representative.
BirthDate	datetime	No	Date of birth.
MaritalStatus	nchar(1)	No	M = Married, S = Single
Gender	nchar(1)	No	M = Male, F = Female
HireDate	datetime	No	Employee hired on this date.
SalariedFlag	bit	No	Job classification. 0 = Hourly, not exempt from collective bargaining. 1 = Salaried, exempt from collective bargaining.
VacationHours	smallint	No	Number of available vacation hours.
SickLeaveHours	smallint	No	Number of available sick leave hours.
CurrentFlag	bit	No	0 = Inactive, 1 = Active

Figure A2. Data Dictionary Excerpt

Appendix B: Query Tasks and Solutions

Table B1. Low-complexity Query Conditions

Pseudo SQL request language	
Template	**No template**

List the department name (from the Department table), the employeeFK (from the EmployeeDepartmentHistory table), and the start date (from the EmployeeDepartmentHistory table). Show only those rows where the end date is null and the group name is Manufacturing.
Solution:
SELECT DepartmentName, EmployeeFK, StartDat
FROM Department
JOIN EmployeeDepartmentHistory ON DepartmentPK = DepartmentFK
WHERE EndDate IS NULL AND GroupName = 'Manufacturing'
	List the product name (from the Product table), the list price (from the Product table), and the unit price (from the SalesOrderDetail table) columns. Show only those rows where the product's style is not null and the unit price is less than 10.
	Solution:
	SELECT ProductName, ListPrice, UnitPrice
	FROM Product
	JOIN SalesOrderDetail ON ProductPK = ProductFK
	WHERE Style IS NOT NULL AND UnitPrice < 10

Manager English request language	
Template	**No template**

Who placed online orders after December 23, 2003? Show the customer's last name, the date of the order, and the total amount of the order.
Solution:
SELECT LastName, OrderDate, TotalDue
FROM Person
JOIN SalesOrderHeader ON PersonPK = CustomerFK
WHERE OnlineOrderFlag = 1 AND OrderDate > '12/23/2003'
	Which salaried employees were hired prior to January 1, 2000? Show the employee's first name, last name, and hire date.
	Solution:
	SELECT FirstName, LastName, HireDate
	FROM Person
	JOIN Employee ON PersonPK = PersonFK
	WHERE HireDate < '1/1/2000' AND SalariedFlag = 1

Table B2. High-complexity Query Conditions

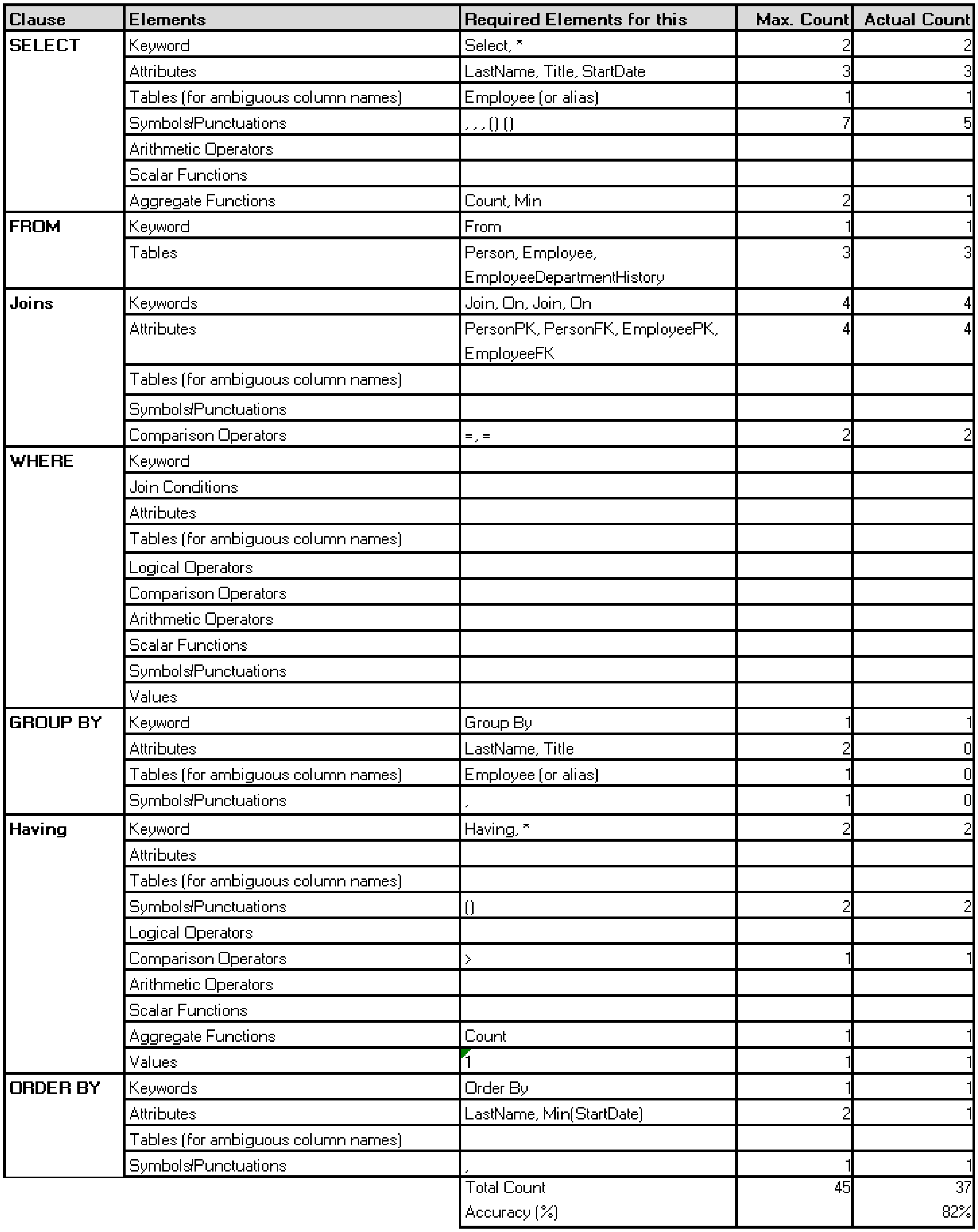
Pseudo SQL request language	
Template	**No template**

List the last name (from the Person table), the sales order PK (from the SalesOrderHeader table), the count of the number of SalesOrderDetail rows, and the sum of the order quantity (from the SalesOrderDetail table). Show only those groups having more than 5 order detail lines. Order the results by last name and sum of the order quantity.
 	
Solution:
SELECT LastName, SalesOrderPK, COUNT(*) AS NumberOfOrderLines, SUM(OrderQty) AS TotalOrderQty
FROM SalesOrderHeader JOIN SalesOrderDetail ON SalesOrderPK = SalesOrderFK JOIN Person ON PersonPK = CustomerFK
GROUP BY LastName, SalesOrderPK
HAVING COUNT(*) > 5
ORDER BY LastName, TotalOrderQty
	List the last name (from the Person table), the title (from the Employee table), the number of departments for which an employee has worked, and the earliest (minimum) start date among all the departments the employee has worked. (Note: The number of departments is the count of the number of rows in the EmployeeDepartmentHistory table for an employee, and the start date is in the EmployeeDepartmentHistory table.) Show only those groups having more than 1 department (row). Order the results by last name and earliest start date.
	
	Solution:
	SELECT LastName, Employee.Title, COUNT(*) AS NumberOfDepts, MIN(StartDate) AS EarliestStartDate
	FROM Person JOIN Employee ON PersonPK = PersonFK JOIN EmployeeDepartmentHistory ON EmployeePK = EmployeeFK
	GROUP BY LastName, Employee.Title
	HAVING COUNT(*) > 1
	ORDER BY LastName, EarliestStartDate

Manager English request language	
Template	**No template**

For each sales order with a total order quantity above 5, show the customer's email address, the order date, the total order quantity, and the average line total amount for the order. Sort the results by order date and email address.
 	
Solution:
SELECT EmailAddress, OrderDate, SUM(OrderQty) AS TotalQtyOrdered, AVG(LineTotal) AS AvgLineTotal
FROM Person JOIN SalesOrderHeader ON PersonPK = CustomerFK JOIN SalesOrderDetail ON SalesOrderPK = SalesOrderFK
GROUP BY EmailAddress, OrderDate
HAVING SUM(OrderQty) > 5
ORDER BY OrderDate, EmailAddress'
	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.
	
	Solution:
	SELECT ProductSubCategoryName, ProductCategoryName, AVG(ListPrice) AS AvgListPrice, COUNT(DISTINCT Color) AS NumDistinctColors
	FROM ProductCategory JOIN ProductSubcategory ON ProductCategoryPK = ProductCategoryFK JOIN Product ON ProductSubCategoryPK = ProductSubcategoryFK
	GROUP BY ProductCategoryName, ProductSubCategoryName
	HAVING AVG(ListPrice) > 100
	ORDER BY ProductCategoryName, ProductSubCategoryName

Appendix C: Sample Accuracy Scoring Sheet



Clause	Elements	Required Elements for this	Max. Count	Actual Count
SELECT	Keyword	Select, *	2	2
	Attributes	LastName, Title, StartDate	3	3
	Tables (for ambiguous column names)	Employee (or alias)	1	1
	Symbols/Punctuations	, , , () ()	7	5
	Arithmetic Operators			
	Scalar Functions			
	Aggregate Functions	Count, Min	2	1
FROM	Keyword	From	1	1
	Tables	Person, Employee, EmployeeDepartmentHistory	3	3
Joins	Keywords	Join, On, Join, On	4	4
	Attributes	PersonPK, PersonFK, EmployeePK, EmployeeFK	4	4
	Tables (for ambiguous column names)			
	Symbols/Punctuations			
	Comparison Operators	=, =	2	2
WHERE	Keyword			
	Join Conditions			
	Attributes			
	Tables (for ambiguous column names)			
	Logical Operators			
	Comparison Operators			
	Arithmetic Operators			
	Scalar Functions			
	Symbols/Punctuations			
	Values			
GROUP BY	Keyword	Group By	1	1
	Attributes	LastName, Title	2	0
	Tables (for ambiguous column names)	Employee (or alias)	1	0
	Symbols/Punctuations	,	1	0
Having	Keyword	Having, *	2	2
	Attributes			
	Tables (for ambiguous column names)			
	Symbols/Punctuations	()	2	2
	Logical Operators			
	Comparison Operators	>	1	1
	Arithmetic Operators			
	Scalar Functions			
	Aggregate Functions	Count	1	1
	Values	1	1	1
ORDER BY	Keywords	Order By	1	1
	Attributes	LastName, Min(StartDate)	2	1
	Tables (for ambiguous column names)			
	Symbols/Punctuations	,	1	1
		Total Count	45	37
		Accuracy (%)		82%

Figure C1. For Participant #4020's Final Query for the High Complexity, Pseudo-SQL Wording, No Template Task

About the Authors

Leo R. Vijayasarathy is Associate Professor of Computer Information Systems in the College of Business at Colorado State University. He earned an MBA from Marquette University and his PhD 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*, *Information and Software Technology*, *International Journal of Production Economics*, and *Journal of Management Information Systems*. He serves on the editorial advisory board of Internet Research.

Gretchen Irwin Casterella is Associate Professor of Computer Information Systems in the College of Business at Colorado State University. She earned a PhD in Information Systems from the University of Colorado, Boulder. Her research interests include human-computer interaction, database management, and systems analysis and design. She has published in the *Journal of the Association for Information Systems,* the *Journal of Management Information Systems*, and *IEEE Transactions on Professional Communication.*