DATA-DRIVEN DATABASE EDUCATION: A QUANTITATIVE STUDY OF SQL LEARNING IN AN INTRODUCTORY DATABASE COURSE

A Thesis

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Andrew Von Dollen

July 2019

COMMITTEE MEMBERSHIP

TITLE: Data-Driven Database Education: A Quantitative Study of SQL Learning in an Introductory Database Course

AUTHOR: Andrew Von Dollen

DATE SUBMITTED: July 2019

COMMITTEE CHAIR: Alex Dekhtyar, Ph.D.
Professor
Computer Science & Software Engineering

COMMITTEE MEMBER: Chris Lupo, Ph.D.
Professor
Computer Science & Software Engineering

COMMITTEE MEMBER: Lubomir Stanchev, Ph.D.
Professor
Computer Science & Software Engineering

ABSTRACT

Data-Driven Database Education: A Quantitative Study of
SQL Learning in an Introductory Database Course

Andrew Von Dollen

The Structured Query Language (SQL) is widely used and challenging to master. Within the context of lab exercises in an introductory database course, this thesis analyzes the student learning process and seeks to answer the question: "Which SQL concepts, or concept combinations, trouble students the most?" We provide comprehensive taxonomies of SQL concepts and errors, identify common areas of student misunderstanding, and investigate the student problem-solving process. We present an interactive web application used by students to complete SQL lab exercises. In addition, we analyze data collected by this application and we offer suggestions for improvement to database lab activities.

ACKNOWLEDGMENTS

The author would like to thank Dr. Alex Dekhtyar for invaluable guidance and an excellent set of SQL lab exercises; Dr. Lubomir Stanchev, for instilling in me an appreciation of database internals and serving on my committee; Dr. Chris Lupo, for leading a wonderful department and serving on my committee; Olga Dekhtyar, for lending statistical expertise; and Toshihiro Kuboi, for data-gathering assistance. Sincere thanks also to my mother-in-law, Dr. Regina Migler, for sparking my interest in academia; my parents and family, for steady encouragement along the way; and to my loving, kind, patient, and infinitely supportive wife, Theresa.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

Databases are widely used as core components of modern software systems. In any application that must maintain persistent information, a database management system often works behind the scenes to support reliable, efficient, and scalable data storage and retrieval. In particular, Relational Database Management Systems (RDBMS) and the Structured Query Language (SQL) have been dominant since the 1980s in the fields of finance, telecommunications, most enterprise software, as well as a wide variety of other application domains. With the advent of NoSQL and NewSQL database management systems [27], SQL and its companion relational model have been re-framed and implemented in new ways. However, the underlying relational foundations remain just as relevant and ubiquitous today as they have been for the last several decades.

Undergraduate database courses typically introduce students to the relational data model, Relational Algebra, and the Structured Query Language (SQL). SQL is a special-purpose declarative programming language used for data access and manipulation [7]. Most students readily learn the fundamentals of SQL due to its approachable syntax. As SQL query complexity grows, however, the appearance of simplicity often fades, revealing nuances and behaviors that can confuse the beginning student [38]. In some cases, this confusion may be traced to an incomplete grasp of the relational theory that underlies SQL. In other cases, SQL's unfamiliar declarative syntax is the source of confusion. Sadiq et al. [30] suggest that many student difficulties with SQL stem from the language's declarative nature, which forces learners to abandon the notion of *steps*, a common programming concept, and instead to think in *sets*. In an attempt to clearly understand the difficulties faced by students who are learning SQL, we focus this thesis on

common errors and student approaches to the process of translating English information requests into valid SQL queries that produce the desired result.

We further seek to identify, analyze, and ultimately work to correct cases where the approach taken by students devolves into simple trial and error mode, without careful problem analysis and evaluation of an attempted solution's correctness. When learning SQL, it is especially important that students ask questions such as: "Which relational concepts or patterns are most appropriate for this specific problem?" and "Does my solution solve the general problem clearly and efficiently?" Evaluating these questions requires that students develop a thorough understanding of the ways in which the features of SQL fit together, as well as a sound mental model of relational operators. Our study aims to highlight areas where these learning tasks may be particularly challenging.

Within the context of lab exercises in an introductory database course, we gather data and perform analysis. With this analysis, we seek catalog SQL query types and their varying levels of perceived difficulty. We also quantitatively study common errors in SQL, and identify patterns in the ways individual student solutions progress. Our ultimate goal is to better teach students to write valid queries in SQL.

We study individual student behavior across multiple lab exercises to construct a broad picture of the evolution over time of student approaches to problem solving. From this data, we extract trends related to query types and errors encountered, determining how individual students' approaches to solving SQL query problems may change over time. We also seek to identify problem-solving approaches and patterns shared by multiple students. Identification of these patterns offers insight into the habits students develop when faced with SQL programming tasks of varying levels of complexity. This analysis of challeng-

ing concepts and patterns will be used as an guide for the enhancement of lab exercises, while also identifying possible areas that deserve more emphasis or explanation during lectures of in other instructional material.

To conduct our study, we developed a custom tool to gather in real time and later analyze student interactions with a database management system, including all SQL statements issued by the student. We performed this data collection with the approval of Cal Poly's Institutional Research Board and with each student's explicit informed consent. The data we gathered provides a window into the problem solving approach followed by students as they complete course lab assignments. Our custom tool allowed passive collection of detailed data that would have been impossible to gather via surveys or similar instruments. We used these data to expand upon previous work devoted to the study of common semantic errors in SQL conducted by Brass and Goldberg [4], as well as studies of SQL learning by Taipalus et al. [36] [35] and Ahadi et al. [2]. We confirmed the completeness and accuracy of previously-compiled lists of common errors within the context of an undergraduate database course. We also compiled additional detailed statistics that provide insight into temporal relationships between errors. Put another way, we were able to determine whether students who encountered error X typically went on to later face error Y, perhaps representing a compounding aspect to initial confusion. Understanding these relationships between errors, along with student problem-solving patterns, allowed us to develop recommendations for course material and instructional tools that may help prevent student misunderstandings from accruing over time.

Our study also included a longitudinal investigation of student problem solving techniques. For each student, we collected timestamped data over the course of an academic quarter. The data consist of full SQL statements issued by the

student to the database for each exercise, along with the outcome: either success or, in the case of a syntax error, the specific error code and descriptive message returned by the RDBMS. With our detailed, longitudinal data, we identified misunderstandings that precede common errors, and developed a picture of the varying problem-solving approaches students followed after encountering a particular error. Using our study results to add depth to previously-compiled lists of common errors, we provide recommendations for additional teaching examples, enhanced practice problems, as well as improved lab exercises. Ultimately, we intend to validate these improved instructional tools and methods in future database courses.

The main contributions of this thesis are:

- Design and implementation of an online database lab tool, which we refer to as "Lab 365",
- Data collection in an introductory database course, spanning four sections taught by two instructors over two academic quarters,
- Statistical analysis of student effort, common errors, and student problem-solving processes, and
- Suggestions for improvement to lab activities.

With our analysis and recommendations, we aim to address areas where students face particular difficulty when learning relational concepts as they are implemented in the Structured Query Language.

The remainder of this document is organized as follows: In Chapter 2, we provide background information on the relational data model, relational algebra,

and SQL as well as a survey of related work. Chapter 3 describes the design and implementation of Lab 365, the web-based application we constructed to support this research. In Chapter 4, we discuss our research goals and methods used to conduct this study. Chapter 5 presents results of our study along with narrative analysis. Chapter 6 discusses threats to this study's validity. Finally, Chapter 7 offers summary conclusions, describes our recommendations for course improvement, and suggests ideas for future investigations that extend the study from this thesis.

2 Background and Related Work

2.1 The Relational Data Model

The relational data model plays a fundamental role of modern data management. Relational database management systems (RDBMS) such as PostgreSQL, Oracle, Microsoft SQL Server, and MySQL are in widespread use today. All share a common foundation in relational concepts introduced by EF Codd in 1969 [11]. The relational model serves as a logical representation of data, independent of the precise physical layout used for storage. In this logical model, the central data structure is the *relation*, or a set of tuples. A relational *schema* is defined based on a set of named attributes; each attribute has a defined domain (scalar data type). In less formal terms, a relation can be viewed as a two-dimensional table with named columns where values in a given column are all of the same simple, atomic type (atomic types include: integer, string, date, or decimal number). Tables 2.1 and 2.2 show example DEPARTMENT and STUDENT relations which might form a small part of a university's database.

Table 2.1: Sample DEPARTMENT Relation, with Primary Key DeptId

DeptId	College	DeptName
CSC	CENG	Computer Science
MATH	COSAM	Mathematics
BIO	COSAM	Biology

The relational data model further defines integrity rules used to ensure that the contents of a database adhere to certain constraints. Each relation must have a single *primary key*, consisting of an attribute or combination of attributes whose value uniquely identifies a single tuple in that relation. Key values may

Table 2.2: Sample STUDENT Relation, with Primary Key StudentId, and Foreign Key DeptId Referencing the DEPARTMENT Relation (Table 2.1)

StudentId	FirstName	LastName	DeptId	DateEnrolled
001	Samuele	Naldrett	CSC	2017-05-06
002	Dario	Stiger	BIO	2018-12-18
003	Minda	Hallick	CSC	2018-07-13

be used to create references from one tuple to another tuple, a concept known as referential integrity or foreign key constraints. The `DeptId` attribute (column) in the `STUDENT` relation depicted in Table 2.2 serves as a foreign key which references the primary key of the `DEPARTMENT` relation shown in Table 2.1.

2.2 Relational Algebra

In addition to the relational model's basic structure and integrity rules, a formal algebra has been defined for manipulation and queries over relational data. Relational algebra builds on first-order logic and set theory to define five primitive operators on relations: selection, projection, set union, set difference, and Cartesian product. Each operator takes one or two relations as its operands and yields a new relation.

The unary selection (σ) and projection (π) operators each take a single relation and return a filtered, possibly rearranged relation that contains a subset of the original relation's rows (in the case of selection) or a subset of columns (when projection is applied.) The example in Table 2.3 shows a simple example of selection combined with projection. Selection relies on a filter predicate over attribute values, which may be expressed using logical connectors: conjunction,

disjunction, or negation. Selection returns only those tuples that satisfy the filter expression. Projection generates a relation that includes only a subset of a relation's attributes. The projection operator may remove, perform calculations on, and/or change the order of attributes in the resulting relation. It is important to note that projection, like all relational algebra operators, yields a *set* of tuples from which any duplicates have been removed. For simplicity, the brief overview in this section deals only with relational algebra's set semantics. The algebraic operations described here are also defined (with different semantics) for bags, or multisets, which may contain duplicates.

Table 2.3: Example of Selection and Projection in Relational Algebra, Corresponding to the Relational Algebra Expression: $\pi_{DeptId,DeptName}(\sigma_{College='COSAM'}(DEPARTMENT))$

DeptId	DeptName
MATH	Mathematics
BIO	Biology

In relational algebra, set union and set difference borrow behavior from their counterparts in set theory, with the constraint that the two relations involved share the same schema, that is: the two relations must have the same degree (number of attributes) and each matching pair of attributes must be from the same domain (i.e. same data type.) This requirement of *union-compatibility* ensures that these operators can be applied to sets of tuples in a well-defined way.

In addition to the four relational algebra operators described above, the fifth primitive operator, Cartesian product($\times$) illustrated in Table 2.4, takes two relations as input and produces a new relation that combines tuples from the input

relations by concatenating the tuples pairwise in every possible way. Cartesian product forms the basis of the derived *join* operator ($\bowtie$) which pairs tuples from two relations based on a comparison of values within the tuples. Several join variations are defined for convenience, all based on the combination of Cartesian product with the primitive selection and/or projection operators. Join variants include *natural join*, which pairs tuples based on the equivalence of attributes with the same name and applies an implicit projection to remove duplicate columns; *theta join*, which pairs tuples based on some arbitrary condition; and *semi-join*, which pairs tuples, then projects attributes from just one of the input relations. Each of these joins operates on two relations and produces a subset of the full Cartesian product of the two input relations.

Table 2.4: Cartesian Product of STUDENT and DEPARTMENT: $STUDENT \times DEPARTMENT$

StudentId	FirstName	LastName	DeptId	DateEnrolled	DeptId	College	DeptName
001	Samuele	Naldrett	CSC	2017-05-06	CSC	CENG	Computer Science
002	Dario	Stiger	BIO	2018-12-18	CSC	CENG	Computer Science
003	Minda	Hallick	CSC	2018-07-13	CSC	CENG	Computer Science
001	Samuele	Naldrett	CSC	2017-05-06	MATH	COSAM	Mathematics
002	Dario	Stiger	BIO	2018-12-18	MATH	COSAM	Mathematics
003	Minda	Hallick	CSC	2018-07-13	MATH	COSAM	Mathematics
001	Samuele	Naldrett	CSC	2017-05-06	BIO	COSAM	Biology
002	Dario	Stiger	BIO	2018-12-18	BIO	COSAM	Biology
003	Minda	Hallick	CSC	2018-07-13	BIO	COSAM	Biology

A number of additional derived relational algebra operators are also defined. These include: set intersection, relation division, anti-join, and outer join. These represent common operations that, when expressed in terms of primitive relation algebra operators, are complex and difficult to readily manipulate. The definition

Table 2.5: Result of a Natural Join Between Relations STUDENT and DEPARTMENT with a Further Selection and Projection Applied, Corresponding to the Relational Algebra Expression: $\pi_{StudentId,LastName,DeptName}(\sigma_{College='CENG'}(STUDENT \bowtie DEPARTMENT))$

StudentId	LastName	DeptName
001	Naldrett	Computer Science
003	Hallick	Computer Science

of these derived operators allows for a more convenient and compact representation, and these operators (with the exception of division) correspond directly to language features available in SQL.

Finally, there are several operators that depart from the set-theoretic foundations of relational algebra. These operators correspond to features found in the SQL implementation, which offers support for grouping and aggregation, bag (or multiset) semantics, and the notion of result ordering. The "extended" relational algebra operators include group by (γ), duplicate elimination (δ), and sorting (τ).

Relational algebra is a powerful, formally defined query language used to manipulate data represented according to the structure and constraints defined by the relational model. The syntax of the SQL programming language closely follows the basic operators defined by relational algebra.

In practice, Relational Database Management Systems (RDBMS) parse SQL queries into trees of relational algebra operators. Algebraic laws and equivalence rules allow databases to find efficient logical query plans. Logical query plans, also typically represented as trees of relational algebra operators, form the basis for the

physical plan ultimately used to execute a given query. A physical execution plan specifies an implementation for each relational algebra operator in a logical plan. For example, a join operation might be performed using a simple nested loop. Or, the RDBMS might choose a hash or sort-merge join implementation. Given the tight link between relational theory and practical database implementations, relational algebra is an important topic in introductory database courses.

2.3 The Structured Query Language

The Structured Query Language (SQL) is a special-purpose declarative language that builds on concepts from the relational model and relational algebra. Introduced in the early 1970's, today SQL is widely used as a core component of many software applications and it endures as a fundamental and ubiquitous tool for data manipulation.

In contrast to typical imperative or procedural languages (such as C, Python or Java) SQL statements specify desired output without describing exactly *how* that output should be produced. This *declarative* aspect of SQL requires a problem-solving process that can feel unfamiliar to novice programmers. The basic syntax of SQL is approachable, but the language's declarative nature requires new thought processes for those who are accustomed to procedural or functional styles of problem solving. SQL's `SELECT` statement corresponds closely to the relational algebra concepts described in the previous section. A `SELECT` statement consists of a number of *clauses* (many of which are optional) that are assembled in a defined order to represent column projection, joins between tables (relations), and selection filters. As a simple illustration, consider the following SQL `SELECT` statement:

```
SELECT DISTINCT StudentId, LastName, DeptName
FROM Student NATURAL JOIN Department
WHERE College = 'CENG'
```

This query corresponds to the relational algebra expression from Table 2.5:

$$\pi_{StudentId,LastName,DeptName}(\sigma_{College='CENG'}(STUDENT \bowtie DEPARTMENT))$$

Generalizing from this simple example, the `SELECT` clause lists columns to be projected. The `FROM` clause identifies a query's subject table(s) and joins. Finally, the `WHERE` clause specifies conditions for row selection. Construction of a simple SQL query typically involves the following process:

1. Choose the table(s) from which data should be retrieved
2. If two or more tables are involved, determine the conditions to be used to pair tuples from multiple tables
3. Construct a boolean expression to filter for row(s) of interest
4. Choose column values to be included in the result table

After following these steps, it is entirely possible to arrive at a SQL command that is syntactically correct and produces correct results for certain input data, but which fails to be a generally valid solution to the problem at hand. As a simple example, given the sample student data in Table 2.2, the following query returns IDs, first and last name of all students enrolled in the CSC department:

```
SELECT StudentId, FirstName, LastName
FROM Student
WHERE StudentId = '001' OR StudentId = '003'
```

This query is syntactically correct and returns apparently correct data in this specific case. However, the above query is clearly not a generally valid solution to

the stated information request, since it fails to include CSC students not present in the sample table. A valid query would apply selection criteria on the DeptId column rather than the StudentId column. From a different perspective, it is often the case that, for a given SQL problem, a number of equally valid, yet syntactically dissimilar, solutions exist. The ability to correctly interpret the *semantics* of an SQL statement can be difficult for novice students to acquire, but it is an essential skill for any student who wishes to effectively apply SQL to all manner of database tasks.

To illustrate the steps required to construct advanced SQL queries and to highlight one of the possible pitfalls, consider the task of listing the IDs and names of all departments in which *no students* enrolled during calendar year 2017. Based on the sample `DEPARTMENT` and `STUDENT` relation instances in Tables 2.1 and 2.2, this task might be decomposed as follows:

1. Construct a query to list all departments in which at least one student *did enroll* during 2017, then
2. List all departments, *excluding* those returned by the above query.

A list of all departments in which at least one student *did enroll* during 2017 can be computed in SQL using the following straightforward query:

```
SELECT DISTINCT d.DeptId
FROM Department d JOIN Student s ON d.DeptId = s.DeptId
WHERE DateEnrolled >= '2017-01-01'
AND DateEnrolled <= '2017-12-31'
```

This query may then be used as a building block to produce the requested information, namely: the IDs and names of departments without student enrollments in 2017:

```
SELECT DeptId, DeptName
FROM Department
WHERE DeptId NOT IN (
  SELECT DISTINCT d.DeptId
  FROM Department d JOIN Student s ON d.DeptId = s.DeptId
  WHERE DateEnrolled >= '2017-01-01'
  AND DateEnrolled <= '2017-12-31'
)
```

There are several alternate approaches to this problem that are equally correct. Importantly, there are also approaches to this task that seem reasonable on the surface but which represent fundamental misunderstandings of SQL concepts. The following is an example of one such flawed attempt:

```
SELECT DISTINCT d.DeptId
FROM Department d JOIN Student s ON d.DeptId = s.DeptId
WHERE NOT(DateEnrolled >= '2017-01-01'
          AND DateEnrolled <= '2017-12-31')
```

In this (incorrect) example, the row filtering condition that was used in the previous query has simply been negated. While this may seem like a reasonable solution at a glance, the join between Department and Student will cause Departments without any students (such as Mathematics in our example data) to be incorrectly excluded. This behavior follows from the Cartesian product that underlies the join operation in relational algebra and SQL. This example highlights one case where a syntactically correct query may, in limited cases, return correct results. In this example, both queries above would return the same (correct) result if the following conditions hold: at least one student record exists

for *every* department, and all student enrollment dates within each department either fall entirely within, or entirely outside the date range. Subtleties such as this are present throughout the SQL language, and are the source of considerable student confusion.

Extensive research has been conducted in the areas of SQL semantics, validity, and quality. The breadth of this research underscores the complexity and intricacies of SQL and makes clear the need to carefully evaluate the ways in which SQL is taught and learned. Courses that cover SQL, including the one on which we base this thesis, typically introduce both the basic syntax of SQL as well as the language's more nuanced features and the ways in which these features combine to form complex queries.

2.4 Teaching Relational Algebra and SQL

The database course that is the subject of this thesis covers the relational model, relational algebra, SQL, transactions, and Java Database Connectivity (JDBC). This quarter-long (10 week) course does not extensively cover topics such as the Entity-Relationship (E-R) model or database normalization, which are typically included in a semester-length database course. After completing the course, students may elect to take a second quarter-long course that introduces E-R modeling, normalization, and other topics related to the design and implementation of database applications. Students also have the opportunity to take another course that covers advanced database system internals and implementation. The study described in this thesis is focused exclusively on the first, introductory course.

This introductory course is typically taken by third- or fourth-year majors in Computer Science and Software Engineering, or students from other STEM

fields pursuing a concentration in Data Science. All students who take the course are required to have previously taken courses in discrete math, fundamental data structures, and basic procedural programming.

The course's coverage spans SQL's main sublanguages, beginning with the Data Definition Language (DDL), followed by the Data Control Language (DCL), the Data Manipulation Language (DML), the Data Query Language (DQL), and finally the Transaction Control Language (TCL). A significant potion of the course's SQL coverage centers on the DQL `SELECT` statement, beginning with the `SELECT`, `FROM`, and `WHERE` clauses. The course then moves to more advanced concepts including grouping (`GROUP BY` clause), grouping with restriction (`HAVING` clause), and nested queries. The course focuses on ANSI-standard SQL syntax included in the SQL-92 specification. We also introduce several recently incorporated and widely supported language features, including window functions and common table expressions (using the `WITH` clause.) Vendor-specific extensions to standard SQL syntax are explicitly not introduced, except where they are useful for comparison with standard language features.

To support the learning process, assignments and lab exercises are designed to progressively build on one another. Initial assignments introduce core SQL skills, later assignments combine concepts in increasingly complex ways. This scaffolding process is first conducted with relational algebra concepts, then the SQL language is gradually introduced in a similar scaffolded manner. Following a student-centered learning style, the course presents various alternative methods and different language syntax that can be used to solve problems in equivalent ways. It is up to the students to choose the problem solving approaches that they find most natural. The course specifically avoids tool-driven as well as rigid, template-based, approaches to SQL query design.

During the course, all students complete the same sequence of lab assignments. An initial lab assignment requires students to work in pairs to develop an application to run queries over tabular data without the aid of a database or SQL. Students then work individually to complete a second lab focused on SQL DDL (`CREATE TABLE` and `ALTER TABLE`) and relational constraints. In the third lab, students again work individually on a lab that covers basic data manipulation using SQL DML. Following this, lab assignments shift to SQL query tasks, which are the focus of this thesis. Labs four, five, and six consist of 30-40 exercises, each of which is an English information request. Working individually, students are required to translate these information requests into valid SQL `SELECT` queries. Each of these three labs focuses on a different set of SQL concepts: table joins & selection criteria, grouping & aggregation, and finally nested SQL. The seventh, and final, lab asks students to work in small teams to build a Java application that uses the Java Database Connectivity (JDBC) API to interact with a database. Lab assignments are distributed to students in a sequential way. In other words, the lab two assignment is typically made available just after the submission deadline for the first lab. Students are allowed approximately one week to complete each lab assignment and are free to use any available resources, including class notes, textbooks, and online resources. The lab assignments we study in this thesis are strictly individual efforts; all solutions must be formulated without code sharing between students.

We perform course assessment via traditional midterm and final examinations. Both are in-class, paper-and-pencil exams designed to evaluate students' understanding of core course material. Each exam requires students to both write and interpret relational algebra expressions and SQL code.

This thesis concentrates on three lab assignments (labs four through six, mentioned above) that involve the translation of English information requests into SQL queries. In the sections that follow, these labs are referred to as labs A, B, and C, which correspond to numbered labs four, five, and six, respectively.

2.5 Related Work

The work described in this thesis builds on previous research in the areas of SQL concept categorization and error classification by Brass and Goldberg [4] [3] and Taipalus et al. [36], interactive environments designed as instructional aids, and specifically, interactive database instruction environments. When analyzing SQL query concepts and errors, we apply the categorization taxonomy described by Taipalus et al. in [36], and we compare our analysis to the analysis presented by Taipalus et al. in [35] and Ahadi et al. [2].

Our work adds to this research a detailed student-centric analysis of the relative difficulty of important SQL concepts, particularly the various combinations of concepts. We also study a much larger and more varied collection of SQL exercises: approximately 150 exercises in total, compared to 15 exercises studied by Taipalus et al. [35] and seven SQL exercises analyzed by Ahadi et al. [2]. This variety offers a richer dataset for analysis, giving us the opportunity to extract statistics related to the interactions of multiple concepts. As described in previous studies, relational concepts (and corresponding SQL language features) often combine ways that students find particularly difficult to understand, even when the concepts alone are readily understood.

2.5.1 SQL Concepts and Common Errors

Recent work by Taipalus et al. [36] identifies fundamental SQL query concepts and describes a unified error classification framework. Based on a large empirical study, the authors identify the most prevalent errors and misunderstandings encountered by students, and further determine the types of queries that invite certain errors. The authors describe three categories of SQL errors: syntax errors, semantic errors, and logical errors. Statements that are invalid according to the SQL specification are considered syntax errors, and the authors categorize these using a DBMS-independent classification. Valid statements that produce incorrect results are categorized as either semantic or logical errors. Queries that are incorrect *regardless* of the information request are considered to be semantic errors. Logical errors are present in those queries that are both syntactically valid and produce a correct result for *at least one* information request, but fail to produce the intended results for a given information request. With this classification of common errors and description of the the types of exercises that typically cause students to encounter certain errors, Taipalus et al. proceed to describe effective methods for the development of course exercises that encourage students to learn how to avoid common pitfalls.

In a follow-up study [35], Taipalus et al. consider types of errors that, once encountered, are most difficult for students to fix. Using the same concept and error classification frameworks proposed in their earlier work [36], Taipalus et al. identify classes of errors that occur across multiple types of exercises, and pinpoint certain errors with which many students struggle. Logical errors, in particular, are a significant source of student confusion. Specifically, simple but subtle errors related to missing join conditions or incorrect column choices in selection or

projection are highlighted as errors that are both commonly encountered and difficult for students to identify and fix. The authors point out that syntax errors, though common, are relatively easy for students to find and fix since they result in immediate DBMS error messages. A clear picture of less obvious types of errors serves as a useful guide for exercise design, and offers a valuable window into the student problem-solving process.

2.5.2 SQL Semantics and Correctness

In [3], Brass and Goldberg introduce a method to check consistency of an SQL query. With this approach, it becomes possible to detect certain runtime errors which can easily go unnoticed. The straightforward, declarative syntax of SQL often masks errors that are only detectable with the benefit of expert SQL knowledge, or when varying the underlying data referenced by a given query. In [4], the same authors build on their original work, enumerating types of semantic errors in SQL. In doing so, they provide a useful collection of common mistakes and misunderstandings faced by students learning SQL. The empirical analysis described in this work makes use of this baseline list of errors. In this work we experimentally confirm that the list of common errors described by Brass and Goldberg in [4] is, in fact, a representation of the types of errors encountered by students learning SQL. This class of errors represents areas of particular subtlety in the language. In addition to these subtle semantic errors, we expand our inquiry to include syntax errors, which are comparative simple and sometimes obvious, but can still vex students attempting to learn SQL. We seek to understand how best to prepare instructional material and practice exercises in a way that clearly illuminates common syntactic and semantic errors. Ultimately, we

hope to define techniques that demonstrably help students learn to avoid these errors.

In an attempt to bridge the gap between the formality of relational theory and its implementation in SQL, Guagliardo et al. [16] run experiments on a large collection of generated queries and database instances to arrive at a description of "real-life" SQL queries. In doing so, this work provides a useful tool for programmers who wish to understand the behavior of SQL queries. This work also provides a valuable experimental basis for questions about query correctness and the subtleties present in SQL implementations. Subtleties, such as the handling of *null* values and bag versus set semantics, can be a source of confusion for those attempting to learn SQL.

2.5.3 Query Similarity

In our observations of the ways in which students independently solve problems, we attempt to cluster students based on similarity of queries used among groups of students. This permits identification of certain problem-solving patterns shared by more than one student. To perform this clustering of queries, we analyze queries for their structural similarity.

We base our initial analysis of query similarity on simple rules that flow from the theoretical underpinnings of SQL as described by Sagiv et al. in [31]. These rules allow certain expressions to be readily matched with equivalent expressions that use different operators. We also leverage research by Kul et al. [23] focused on analysis of high-volume database access logs. In this work, the authors describe a method for clustering similar queries as a first step towards extracting broad patterns and thus developing a better understanding of large database logs. With

this analysis of query clustering, the authors provide a better basis for database performance tuning, security auditing, and benchmark design. Our study of the SQL learning process has at its core a similar collection of widely varying queries from multiple sources. Applying the clustering techniques from Hul et al. [23], in this thesis we seek to extract problem solving patterns that exist in students' lab work.

Directly addressing questions of query correctness and equivalence, Chu et al. describe COSETTE, an automated prover for determining whether two SQL queries are truly equivalent [9]. The authors demonstrate that COSETTE is able to decide equivalence for a large subset of SQL queries. In cases where the tool detects query *inequivalence*, COSETTE is further able to produce data that serves as a counterexample. This capability, integrated into an interactive tool, clearly highlights cases where a query returns correct results for a given database instance, but fails to correctly solve the general question. Cases such as this often represent subtle misunderstandings on the part of students learning SQL. The ability to automatically identify these cases would likely improve students' understanding of the complex features of SQL.

With a similar focus on the correctness of SQL statements, Hsu et al. are among many to have focused on the problem of rewriting queries while maintaining semantic correctness [21]. Adding a concrete implementation and empirical analysis to the question of query equivalence, Chu et al. describe a tool named HoTTSQL [10] that is able to determine, within a constrained class of SQL queries, whether or not two queries are equivalent to one another. The authors note that the general problem of determining query equivalence is undecidable. This highlights the difficulty of the task of determining whether a given query is a truly valid solution to a particular problem. It is therefore easy to understand

why novice students faced with the task of formulating a valid solution for a complex task can be overwhelmed by the intricacies and features of SQL.

2.5.4 SQL Testing and Quality Metrics

Traditional research into automated testing and test coverage focuses mainly on applications written in imperative or structured programming languages. Somewhat less emphasis has been paid to automated testing for declarative languages such as SQL, particularly in the typical case where SQL is embedded within an application that is written in a procedural language, such as Java or Python. In [34], Suarez-Cabal et al. describe coverage measurements specifically designed to measure SQL queries. Furthermore, the authors present testing techniques to ensure high test coverage of query code, especially SQL embedded within a larger application.

In a similar way, Veanes et al. describe methods for the generation of test data and SQL query parameters to fully exercise given test conditions [37]. To do so, the authors transform the problem of test data generation into one of satisfiability. The authors integrate their approach into the database testing framework exposed by the Visual Studio IDE.

When learning SQL, students benefit from a complete picture of eventual uses of SQL. In particular, it is important for students to gain an understanding of the mechanisms by which a given SQL query can be thoroughly tested for validity. We seek to find out how best to guide a student's problem-solving process, so that the student constructs SQL queries with the same care and eye toward testability as she would write any application code. In particular, our study aims to reveal how the building blocks of relational algebra operators and

corresponding SQL features can be most effectively taught, especially in cases where there is a need to layer concepts in complex ways. We hypothesize that students who firmly understand basic relational algebra building blocks will be better able to combine these concepts in a modular way to develop solutions to complex information requests.

2.5.5 Interactive Lab Environments

Significant research effort has been applied to the design and implementation of effective interactive environments to teach introductory programming concepts. Such environments have been used to teach a wide range of topics, including traditional procedural programming [18], test driven development [19], as well as SQL [25]. In this thesis, we adopt many proven ideas from this previous work, particularly the use of immediate, automatic feedback.

Higgins et al. describe a Computer Based Assessment (CBA) system, known as CourseMarker [18], which is used to teach introductory Java programming at the University of Nottingham. The design of CourseMarker emphasizes free-form programming exercises, in contrast to similar systems focused narrowly on simple text-based or multiple choice exercises. The CourseMarker system also provides immediate feedback to students, which is shown by the study to clearly benefit students. In this thesis, we adopt many of these same proven ideas to build an interactive tool to allow students to solve SQL exercises rather than procedural programming problems.

With an approach focused more intently on performance assessment and student perception, Hilton and Janzen [19] describe an environment to aid in introductory programming instruction. The authors focus their study on test-driven

development (TDD) and related learning processes. They design an interactive environment named WebIDE that encourages students to use TDD methods while solving programming problems related to array handling. The study finds that students who complete assignments using WebIDE, a custom "intelligent tutoring system," performed better on objective assessments and have a subjectively better learning experience.

2.5.6 Database Lab Environments

In the specific area of database courses, Mitrovic presents a SQL-specific intelligent tutoring system, known as SQL-Tutor [25], designed to teach the SQL programming language. The authors explore how well this interactive environment supports student learning by conducting three evaluation studies. First, the authors investigate a student modeling approach known as Constraint-Based Modeling (CBM) [26]. This approach is similar to the work described by Hilton and Janzen in [19]. In both cases, the authors evaluate the usefulness of their systems (SQL-Tutor and WebIDE) based on subjective student assessments. Additionally, the authors use SQL-Tutor to build long-term models of student knowledge. These models then drive adaptive problem selection, aimed to improve the student learning process. This last line of inquiry, in particular, forms part of the basis for the longitudinal studies conducted in the study of the SQL learning process described in this thesis.

In [6], Cagliero et al. describe the design and construction of a tool intended to assist teaching assistants tasked with supporting practical lab work in database courses. This application provides real-time data to instructors and teaching assistants, indicating which students or student teams are struggling to solve certain problems. The focus of the research is on the identification of students who could

benefit from immediate help during an in-person lab. The authors also describe the possibility of later data analysis to determine whether or not the interventions were successful, as well as to identify common misunderstandings. We do not, in this thesis, focus immediately on the real-time intervention possibilities. However, the lab tool described in this thesis does permit such investigations in the future.

2.5.7 Concept Inventory Construction and Evaluation

Drawing from prior research related to instructional effectiveness in the fields of physics, circuits, and engineering [14], recent work by Caceffo et al. describes the application of concept inventories to teaching and assessment in the field of computer science [5]. Traditionally, a concept inventory (CI) is a carefully constructed multiple-choice test that aims to thoroughly cover a given subject area. Each question that appears on a concept inventory is designed, based on careful research, to measure students' understanding of specific concepts.

Saarinen et. al. describe the difficult and time-consuming processes required to create an effective concept inventory in [29]. To address this, the authors present a tool that combines "classsourcing" with machine learning to produce sets of questions optimized to identify study misconceptions. The authors further describe how these question sets can be used to construct effective, comprehensive concept inventories. The resulting process, called Adaptive Tool-Driven Conception Generation, is validated in the context of problem sets related to array handling in Java. This study confirms that the generated problems are similar to those that would be created by a lengthy expert-driven process. In this thesis, we rely on expert-created SQL problem sets. Our study aims to confirm that these problem sets are complete, with no significant gaps between concepts or inade-

quate coverage of certain SQL concepts. In the future, we intend to explore the notion of class-sourcing SQL exercises and automatic construction and validation of a concept inventory for SQL queries.

3 Tool Implementation

To investigate the student learning process, we built a custom application, referred to throughout this thesis as "Lab 365". We used this application to gather data related to the student problem solving process within the context of normal course lab assignments. This tool allowed passive collection of data that would have been difficult to gather via surveys, interviews, or similar research instruments. The detailed data we collected provide a fine-grained, unfiltered view of the student problem-solving process. As noted by Fincher in [15], a rich data set, such as ours, allows analysis beyond what could be accomplished through the use of researcher-distant methods, such as questionnaires, surveys, or interviews. The data-gathering tool we constructed was specifically designed to support investigations that closely analyze the student problem solving process.

3.1 Application Overview

We constructed our data-gathering tool as a web application that can be accessed by students using any modern web browser. The user interface consists of standard HTML/CSS/JavaScript layout and controls. We incorporated standard visual components from the Bootstrap [24] and Webix [33] user interface libraries. Server side application logic was implemented in Java using the Spring Boot framework [28], specifically: the framework's Model-View-Controller (MVC) library, authentication and authorization API, and object-relational mapping capabilities, which are based on the Java Persistence Architecture (JPA) API. For data storage, we use the MySQL RDBMS. All server-side components are hosted via Amazon Web Services (AWS).

Lab 365 stores application configuration, user accounts, and student activity logs in a single MySQL database, depicted as an Entity-Relationship diagram in Figure 3.1. In addition, the tool connects to several separate lab databases against which student queries are executed. These MySQL lab databases are created and populated manually. All students have a read-only view of the same shared lab databases. Details about the content and structure of these lab databases can be found in Section 4.3.1 and Appendix B.

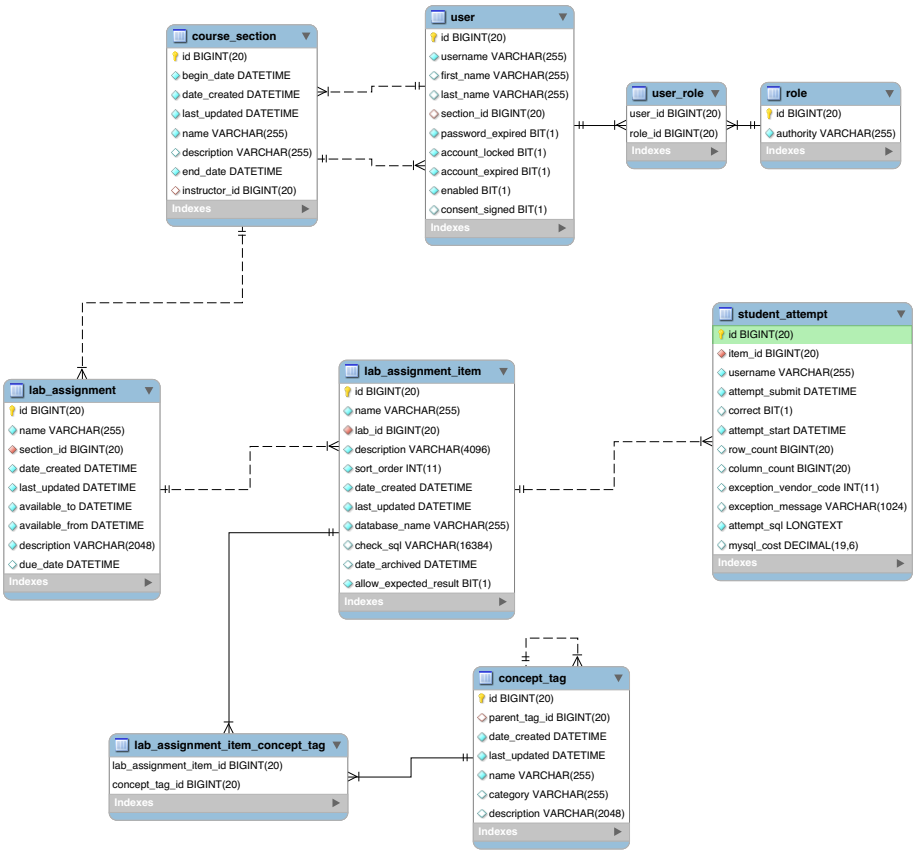

Figure 3.1: Lab 365 Application Database, Entity-Relationship Diagram

3.2 Instructor Features

Administrative screens, available only to course instructors, facilitate setup and review of lab assignments and the SQL exercises that comprise each lab. For each assignment, the instructor must specify a name, description, start/due dates, and a list of individual SQL exercises. Each SQL exercise consists of a label, an English-language information request, and a single "Check SQL" statement that is used to produce a result table for comparison with user submissions. The SQL concept(s) present in each exercise may optionally be provided to allow analysis on a concept-level basis, as we do in Chapter 5. Figure 3.2 depicts the screen an instructor uses to create or modify a lab assignment.

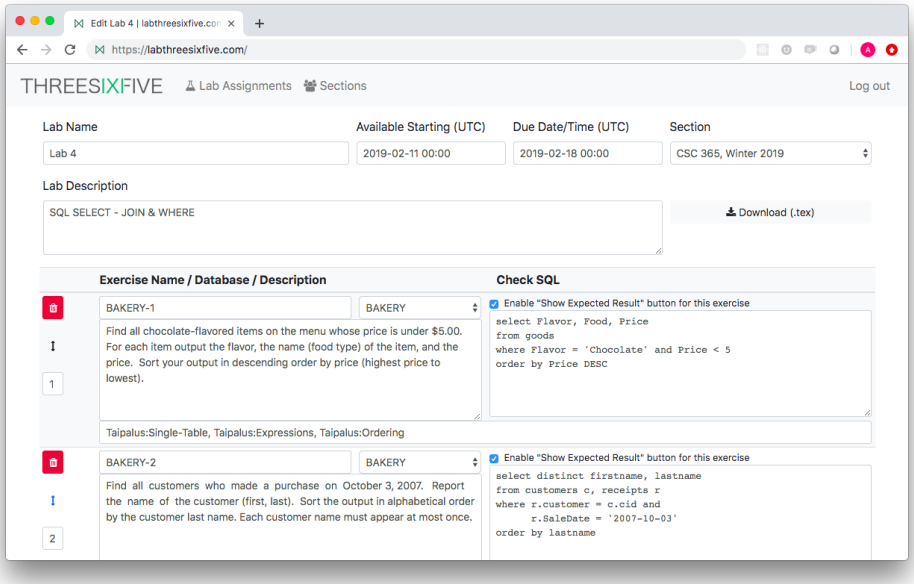

Figure 3.2: Lab 365: Lab Assignment Setup Screen; Available Only to Instructors

The Lab 365 application provides, for course instructors only, a simple dashboard that displays charts which offer a broad view of student progress for each lab assignment, as well as each SQL exercise. As shown in the example in Figure

3.3, a bar chart indicates the number of students who have attempted each exercise (light-grey bars) as well as a count of students who have submitted a correct response (green bars.)

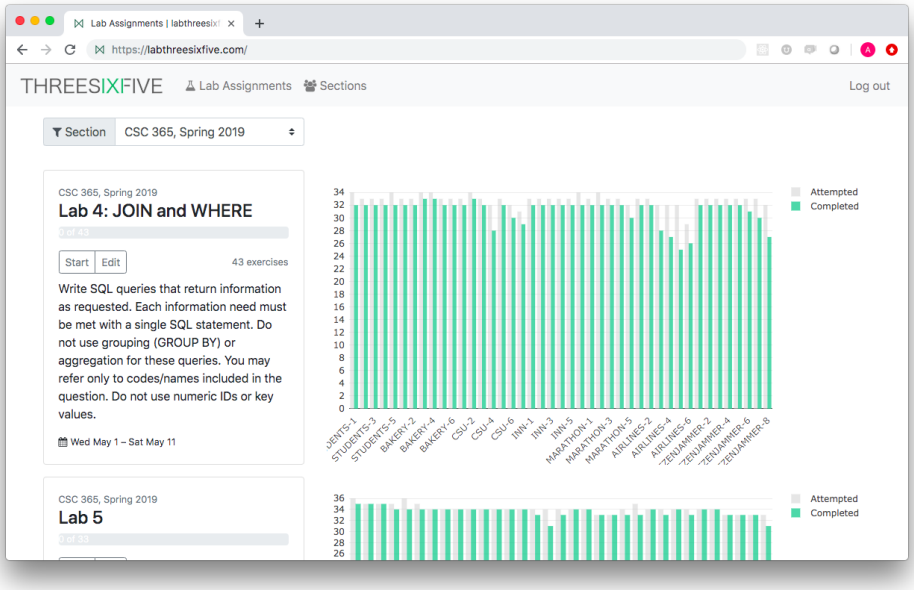

Figure 3.3: Lab 365: Instructor Home Screen; Displays High-Level Student Progress for Each Lab Exercise

3.3 User Interface for Students

After successful authentication, users of the Lab 365 application are first presented with a home screen that lists available lab assignments, shown in Figure 3.4. This listing includes a brief description of each assignment, the number of exercises in each assignment, and the number of exercises the student has successfully completed thus far within each lab assignment. After choosing a lab assignment, the system presents the student with a single screen (depicted in figure 3.5) that displays, on the left side, all exercises within the lab assignment. Clicking on an exercise displays an English information request, along with an

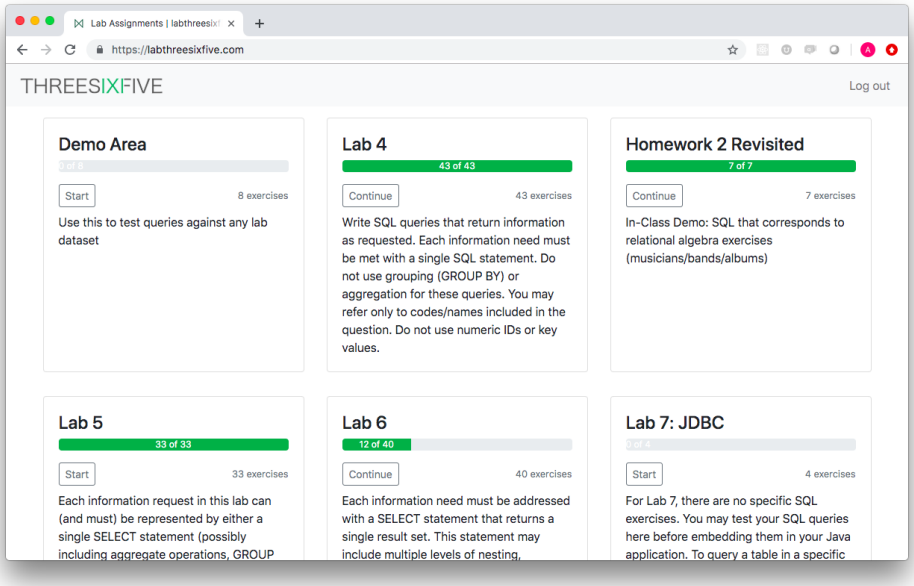

Figure 3.4: Lab 365: User Home Screen; Lists Available Lab Assignments

editor panel into which SQL commands may be typed. Three action buttons are available:

1. **Run SQL** - execute the SQL statement currently in the editor panel, returning either the result table or an error message from the RDBMS. This feature is intended to be used to run SQL statements that do not fully solve a problem. Example uses include: testing subqueries in isolation, or inspecting the contents of a specific table.

2. **Check** - execute a full SQL statement and compare the result of the current SQL statement to the expected result table. Displays the results from the student's SQL query along with a "Correct" or "Incorrect" indicator.

3. **Show Expected Result** - Display the expected result table

The results for each SQL statement are displayed in tabular form in a collapsible right-hand panel. Any errors from the underlying RDBMS, including syntax errors, are echoed directly to the user, as shown in Figure 3.6.

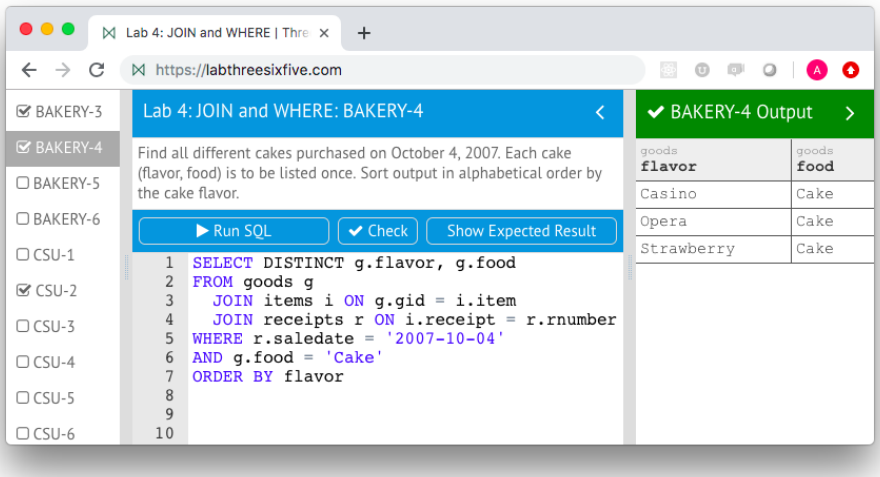

Figure 3.5: Lab 365: Main Lab Exercise Screen, User's View

Students are free to work on lab exercises in any order and at any time up to a pre-defined assignment due date. For most exercises, the "Show Expected Results" button is available, except as noted in Appendix D, and this feature may be accessed at any time). As mentioned by Taipalus in [36], the ability to view expected results does not accurately mimic real-world scenarios in which developers are often required to develop correct SQL queries without previous knowledge of expected results. We provide this capability to permit ready comparison with previous studies of SQL learning. We intend in the future to conduct experiments related to the availability of result tables.

The application determines "correctness" based on a simple comparison of result tables. This comparison does not in every case mean that a student's submitted query represents a generally *valid* response to an information request. The

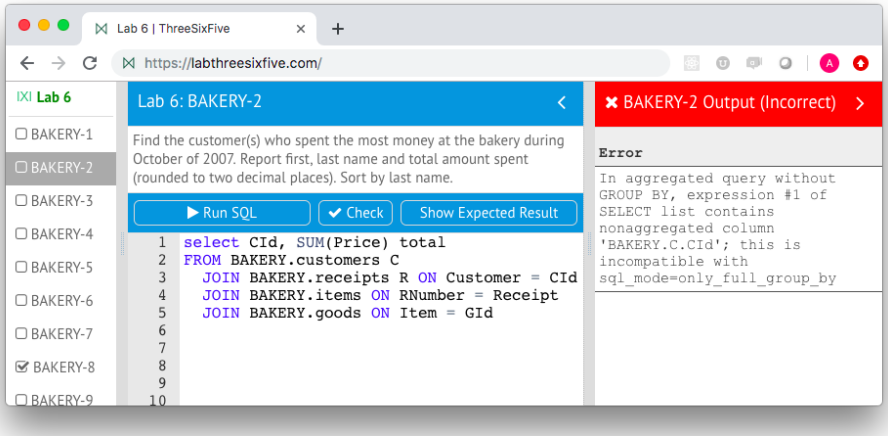

Figure 3.6: Lab 365: SQL Syntax Error Display, User's View

submitted query may correctly satisfy the request for a single database instance, but may fail to produce valid results if data is added, removed, or modified. Or, the query may incorrectly hard-code certain values that allow the query to yield correct results, but again would cause the query to fail in the presence of different data. In extreme cases, students may use static values to produce a correct-seeming result table without actually solving the problem at hand (for example, a `SELECT` with literal values and no `WHERE` clause). These, and other, error types are addressed through manual grading, and are discussed further in Section 4.5. The inability of the Lab 365 application to automatically detect these errors and provide immediate feedback represents a known limitation in the system, because it may offer false assurance to students that their solution is correct. Students are made aware of this before beginning to use the system and are cautioned that they should carefully review their final submissions for logical errors. We plan to address this limitation in a future release, using some combination of multi-instance or "hidden" tests and automatic error classification (which is also described as a future goal the the Taipalus study [36]).

The Lab 365 application records every student attempt (success or error) and stores student progress and SQL edit buffers across sessions. After logging out, a student may return at any time to continue work, picking up exactly where he/she left off. In addition to the valuable data recorded by the application for analysis by instructors, students appreciate the convenient, user-friendly interface that allows them to complete lab assignments on their own schedule, without the need to install or learn specialized software.

4 Methodology

In the introductory database course on which this thesis is based, students learn the principles of modern relational databases. Specifically, the course's learning objectives indicate that students should (after completing the course) be able to:

- Describe and apply the principles of the relational model and relational algebra.
- Create a relational database using SQL's Data Definition Language (DDL).
- Formulate SQL statements for data manipulation.
- Formulate information retrieval statements using relational algebra and SQL, including the primitive operations of selection, projection, Cartesian product, set union, and set difference; as well as derived operations such as joins.
- Design and implement a Java-based application that interacts with a relational database using the JDBC API.
- Describe, at a high level, the internal organization of a DBMS, including key subsystems and their purpose.

Much of the course's emphasis relates to the formulation of valid SQL statements to solve data manipulation and query tasks; first in isolation and then in the context of a complete application that includes embedded SQL statements. To this end, the course progressively and thoroughly introduces SQL syntax and semantics. Lab exercises are designed to give students practical experience applying SQL concepts to real-world information retrieval tasks, and to validate students' abilities to do so. Along the way, certain concepts invariably prove more difficult

than others. Although experience as an instructor provides some intuitive insight into the relative difficulty of SQL concepts, quantitative confirmation is required. Fundamentally, this study aims to answer the question: "What database query concepts, or combinations thereof, trouble students the most?"

We concentrate our study on the problem-solving steps taken by students who are faced with the task of transforming English-language information requests into syntactically and logically valid SQL queries. Specifically, we pose the following research questions:

1. What are the most difficult SQL concepts, or combinations of concepts, to master?
2. Among students who attempted a given query problem and were unsuccessful, what are the most common terminal errors?
3. Which SQL syntax errors require the most number of attempts for students to resolve?
4. Among those students who are *unable* to successfully solve a given problem, what errors are encountered that are not also encountered by those students who successfully solve the same problem?

4.1 Overview

We conducted our experiments over the course of two academic quarters across four sections of an introductory database course taught by two different instructors. The four cohorts are summarized in Table 4.1. Identical sets of lab exercises were assigned to all four cohorts, without any modification between academic quarters or instructors. All enrolled students were given an opportunity to review

an Informed Consent form (Appendix A.1). The "Study Participants" column in the summary table shows the number of students who chose to sign the form and participate in the study described in this thesis.

Table 4.1: Course Enrollment and Study Participants

Cohort	Quarter	Instructor	Enrollment	Study Participants
1	Winter 2019	A	31	26
2	Spring 2019	A	35	30
3	Spring 2019	B	37	32
4	Spring 2019	B	32	26

4.2 Experimental Design

The primary quantitative experiment described in this thesis uses repeated measures design [13] to compare student performance on multiple SQL lab exercises over time. In addition, we used simple mean and variance comparisons to compare lab assignments and, by extension, to compare student approaches to the SQL concepts that are the focus of each lab assignment. As discussed in Section 2.4, each lab assignment consists of 30-40 exercises centered around a particular set of SQL language features. Specifically, the first lab assignment focuses on joins and filtering, a second lab emphasizes grouping and aggregation, and the third lab covers nested queries. Lab assignments build on one another, combining concepts in progressively more complex ways.

Since our experiment uses repeated measures design, we introduce the possibility of effects related to the order in which students complete lab exercises. A student is free to complete exercises within a given lab in whatever order he or

she chooses, raising the possibility that fatigue or other factors may play a role in student performance. We can, however, observe the order in which students work on lab exercises. At a broad level, we control the order in which entire labs are assigned. In our study, students completed a full assignment related to joins and filtering expressions before beginning work on exercises related to grouping and nested SQL.

4.3 Lab Assignments and Query Types

We collected data to support this via a custom web-based application (described in Chapter 3). This lab tool is used by students to solve lab exercises assigned during normal course lab activities. This application captured student interactions with the course database, including all attempts to solve problems, as well as errors encountered. The study includes longitudinal data; we recorded student lab activity over time to allow an individual's learning progress to be analyzed and compared to that of other students. By collecting fine-grained detail consisting of all student attempts, rather than conducting traditional surveys or analyzing final responses only, we were able to perform data analysis that is much closer to the student, as described by Fincher et al. [15].

Our sample includes students enrolled in selected sections of an introductory database course. Student enrollment decisions were made with no prior knowledge of this study. Every student was given the opportunity to review an Informed Consent form (see Appendix A.1.) Students could opt out of the study altogether or at any point during the study. In the four sections that participated in this study, approximately 80% of students give their informed consent. No students chose to discontinue participation while the study was ongoing.

4.3.1 Query Types and SQL Concepts

Lab exercises are based on nine different databases containing hand-crafted data. Each database follows a common structural theme, as outlined in Table 4.2.

Table 4.2: Lab Databases Used in this Study

Database	**Type**	**Description**
AIRLINES	Graph	Information about airlines and flights between 100 different airports
BAKERY	OLTP	Sales detail for a small, fictitious bakery
CARS	Normalized	Statistics about 406 car models produced worldwide between 1970 and 1982
CSU	Normalized OLAP	Historical enrollment and fee data from the California State University 23-campus system
INN	OLTP	Reservation data for a fictional Bed & Breakfast
KATZENJAMMER	Normalized	Data related to the musical career of pop band Kaztenjammer from Norway
MARATHON	Universal Table	Results from a half-marathon in New England
STUDENTS	Normalized	Students, classrooms, and teachers at a small, fictitious elementary school
WINE	Partially Normalized	Ratings of a variety of wines produced in California

Appendix B contains full descriptions of the nine lab databases used in this study. These lab databases form the basis for all lab assignments. A single lab assignment consists of between five and eight exercises for each database. Within a lab assignment, exercises are arranged in increasing order of perceived difficulty. For example, an exercise labeled BAKERY-1 is intended to be easiest exercise within the BAKERY database for that lab assignment, while BAKERY-2 requires a more complex solution. Query concepts are repeated across multiple databases. For instance, the multi-table join concept might appear in BAKERY-3, CSU-4, and AIRLINES-2. In this way, students gain practice applying SQL query concepts within multiple database structures. Furthermore, some database structures invite or require certain approaches. For example, in the single-table MARATHON database, solutions must be expressed using self-joins or subqueries. Through exposure to multiple database structures, students are encouraged to practice and refine skills spanning all SQL concepts and multiple problem domains.

The use of standard databases avoids the need for students to familiarize themselves with a brand new database structure for each assignment. Instead, students solve information requests with the benefit of thorough knowledge of the tables, columns, and relationships present in a given database. All schema information is provided to students at the beginning of the course. In an initial lab, students write SQL Data Definition Language (DDL) and Data Manipulation Language (DML) statements to define and populate each of the lab databases. To complete the SQL query labs described in this thesis, students use standardized versions of the lab databases rather than their own versions (which may differ slightly, due to column naming conventions, etc.)

Table 4.3: Summary of Lab Assignments Used in this Study

Lab	**Exercises**	**Area of Focus**
A	43	JOIN, WHERE
B	33	GROUP BY, aggregate functions, HAVING
C	40	Nested SQL

Each lab assignment centers around certain features of SQL, as listed in Table 4.3. Below is an example introduction to a lab assignment:

> Write SQL queries that return information as requested. Each information need must be met with a single SQL statement. Do not use grouping (GROUP BY) or aggregation for these queries. You may refer only to codes/names included in the question. Do not use numeric IDs or key values.

In the first two labs, students are instructed to confine their solutions to specific SQL features. Students must formulate Lab A queries without the use of grouping, aggregation, or nesting. For Lab B, students are permitted to use joins, expressions, grouping, and aggregation but nesting remains disallowed. In the final lab assignment, Lab C, students are permitted to use any ANSI-standard SQL language features. These limitations on expressive power are intended to encourage students to develop a firm understanding of simple concepts. As labs progress, students are required to combine concepts in ever more advanced ways. As described by Chan et al. [8], expressive ease and task complexity are two important factors that affect user success in database query scenarios. Chan et al. describe a third factor, representational realism, which varies among different query languages. Since we have confined our study to a single query language (SQL) we set aside the question of representational realism. We do, however, rely

on measures of expressive ease and task complexity as two of our independent variables.

We adopt and extend SQL concept categorizations from previous studies of SQL learning by Ahadi et al. [2] and Taipalus et al. [36]. Both prior studies identify concepts such as single-table queries, multi-table joins, restriction expressions, grouping, nested SQL, and ordering. In this thesis, we study several SQL concepts that were not specifically included in these prior studies. These additional SQL concepts are: set operations, such as `UNION`, joins based on an inequality comparison between column values, which we term "non-equi-joins", relational division, full outer join, and pivot operations used to transpose rows and columns. Table 4.4 contains a complete list of SQL concepts that we address in this thesis.

Each exercise focuses on one or more fundamental SQL concepts, with key concepts repeated across multiple exercises. Appendix C lists the concepts present in each exercise. By repeating concepts, students gain practice applying each SQL concept in multiple ways against multiple database structures. Furthermore, the extensive collection of lab exercises we study in this thesis permits us to evaluate concept *combinations*. Many SQL concepts are easily understood in isolation but become challenging when combined with other concepts. For example, most students readily learn basic join concepts as well as single-table grouping and aggregation operations. Difficulties often arise, however, when students must combine joins and grouping within a single SQL query. Table 4.5 provides a summary view of the distribution of SQL concepts across the three lab assignments we study in this thesis. Figure 4.1 depicts a co-occurrence matrix showing the frequency with which pairs of concepts appear together in lab exercises. In the remainder of this thesis, we seek to identify concepts and combinations with

Table 4.4: SQL Query Concepts Studied in this Thesis

Query Concept	**Description**	
Single-table	Selection and projection using a single table	[2]
Multi-table	Multiple tables joined together using natural or equi- joins	[2]
Ordering	Sorting using the `ORDER BY` clause	[36]
Expressions	Row restriction expressions that appear in the `WHERE` clause	[36]
Expressions With Nesting	Row restriction expressions that involve nested conjunctions and disjunctions	[36]
Multiple Source Tables	Projected columns from multiple tables	[36]
Grouping	Row grouping using the `GROUP BY` clause	[2]
Grouping restrictions	Use of the `HAVING` clause	[36]
Aggregation Functions	Use of SQL aggregate functions *with or without* the `GROUP BY` clause	[2]
Computed Grouping	Grouping on computed values, rather than plain column values	
Parameter distinct	`DISTINCT` keyword required within an aggregate function (ie. `COUNT(DISTINCT ...)`	[36]
Does not exist	Negated existential quantification ($\neg\exists$), corresponds to SQL's `NOT IN`, `NOT EXISTS`, or `OUTER JOIN` syntax	[36]
Uncorrelated subquery	Single-level nested SQL in any of the `SELECT` clauses	[2]
Correlated subquery	Correlation required between inner and outer subquery	[2]
Equal subqueries	Two or more subqueries that appear at the same level	[36]
Self-join	A join that requires two instances of the same table	[2]
Scalar Functions	Use of SQL scalar functions, such as `ROUND`, or RDBMS-specific date functions	
Set Operations	Use of relational set operations, such as `UNION`	
Non-Equi-Join	Join based on an inequality between condition	
Relational Division	Identify values from a relation that are paired with *all* values from another relation	
Pivot	Transpose data from multiple rows into columns within a single row	
Full Outer Join	Include all values from both the left and right sides of a join	

which students have particular trouble. We also aim to identify cases where existing lab exercises fail to adequately cover important concept combinations.

Table 4.5: Summary of the Distribution of SQL Concepts Across Lab Assignments

Lab	Concept Count	Concepts per Exercise (Avg)
A	14	4.0
B	15	5.8
C	23	6.8
All Labs	**28**	**5.4**

4.4 Data Set

The data we analyze in this thesis consists of an established set of SQL query problems as described above, coupled with extensive student-database interaction data collected as part of this thesis. To allow comparison with previous similar studies, we classify query concepts and errors according to the taxonomies described by Taipalus et al. in [36]. In this section, we describe the raw data that we collected and we summarize the error taxonomy used in the next chapter.

4.4.1 Raw Data

The data-gathering application described in Section 3.1 records the following detail for every student interaction with the course database:

- Anonymized course/section identifier
- Anonymized student identifier
- Description of the task / objective
- Date and time (seconds precision) of student submission

Figure 4.1: SQL Concept Co-occurrence Matrix - Values in Each Cell Represent the Number of Times the Two Concepts Appear in the Same SQL Exercise. In this thesis, we studied 28 distinct SQL concepts and 130 different lab exercises. Concepts marked with an asterisk (*) were discussed by Ahadi et al. in [2]; concepts marked with a plus sign (+) were first discussed by Taipalus et al. [36]

- Full SQL statement submitted by student
- Database error code and message (if any)
- Number of rows and columns in the result set (if any)
- True/false flag indicating whether the query output matches the expected result

All data exported for analysis are anonymized. Course/section information and student detail, including all personally identifiable information (PII), is omitted from all files that are exported for analysis. Anonymized student identifiers are stable over time. This permits us to analyze an individual student's activities across multiple exercises, while still explicitly avoiding export of any personal information.

From this raw data we extract summary statistics, such as attempt counts per exercise, timing statistics, and success rates for each lab exercise (the percentage of students who successfully complete each exercise.) Certain SQL statements are excluded from attempt counts, including: commands used to inspect the RDBMS catalog (such as the MySQL `show tables` or `descr <table name>` commands), empty statements likely issued inadvertently, and identical statements run in succession. In addition to count, timing, and success rate metrics, we categorize each student-submitted query using the groups described in Tables 4.6, 4.7, 4.8, and 4.9. This categorization of queries allows us to analyze common syntax errors and error sequencing. In addition, we can begin to investigate misunderstandings that lead to semantic and logical errors.

4.4.2 Error Taxonomy

We further classify errors found in student responses using the error taxonomy described by Taipalus, et al. in [36] and summarized in Tables 4.7, 4.8, and 4.9. This taxonomy separates syntactic, semantic, and logical errors into detailed categories. Applying previously-proposed error taxonomies permits us to compare our results with conclusions from previous studies. It must be emphasized, however, that the study described in this thesis differs in several ways from the study described in [36]. Specifically, we excluded from our study an investigation into

Table 4.6: High-Level Groups Used to Categorize Student-Submitted SQL Queries

Category	**Description**
Syntax Error	One or more syntax errors
Semantic Error	Syntactically correct query that is incorrect, no matter the information request.
Logical Error	Syntactically correct query that satisfies an information request different from that stated.
Valid	Query fully satisfies the information request

query "complications," defined by Taipalus et al. as errors that do not affect the final validity of a query but do potentially impact readability, maintainability, or performance. Examples of complications include unnecessary joins, grouping using a single group, or unnecessary correlation names. In further contrast to the Taipalus study, we performed our study using the MySQL RDBMS rather than SQLLite. Lastly, we conducted our study during normal course activities, rather than as a separately-administered instrument. With these differences in mind, we still find value in adopting the well-thought-out concept categories and error taxonomies proposed by Taipalus et al.

Syntax Errors

Syntax errors represent the most frequently-encountered type of errors in our data. The presence of a syntax error prevents the database from evaluating a query and results in an immediate error message. We restrict our study to syntax errors as reported by the MySQL RDBMS. We explicitly do not consider cases where certain syntax violates the ANSI SQL standard but is accepted by a particular RDBMS. Examples of non-standard syntax include: non-standard quotes, non-standard operators (`&&` versus `AND`, `!=` versus <>, etc.), as well as implicit type coercion operations which are permitted by lenient default settings in certain RDBMSs (including MySQL.) As discussed in Section 7.1.4, future emphasis seems warranted in this area. However, the results discussed in this thesis considered a statement to be syntactically valid if it executed without error in the course MySQL lab environment, which does not exactly adhere to the ANSI standard.

When analyzing syntax errors in student data, we apply the RDBMS-independent classification scheme described by Taipalus and summarized in Table 4.7. The classes and identifiers we use for syntax errors (ie. **SYN-1**) are identical to those defined by Taipalus, et al. in [36].

Semantic Errors

The next class of errors, semantic errors, is defined by Brass and Goldberg [4] as those that cause a SQL query to be incorrect *no matter what the original information request may be.* Errors of this type include inconsistent logical expressions that always yield an empty result, as well as joins that fail to pair tuples in any

Table 4.7: Classification of Syntax Errors, as Proposed by Taipalus et al. [36]

ID	Description
SYN-1	Ambiguous database object
SYN-2	Undefined database object
SYN-3	Data type mismatch
SYN-4	Illegal aggregate function placement
SYN-5	Illegal or insufficient grouping
SYN-6	Other common syntax error

sensible way, such as comparing values from different domains or simple failure to include a necessary join condition.

While it would be possible in some cases for databases to detect these errors and warn users, commonly-used databases do not have this capability. Semantic errors can be difficult for SQL novices to detect and correct [32], and it is therefore useful to include them in this study of student errors.

Table 4.8: Classification of Semantic Errors, First Proposed by Brass & Goldberg [4], further Studied by Taipalus et al. [36]

ID	Description
SEM-1	Inconsistent expression
SEM-2	Inconsistent join
SEM-3	Missing join
SEM-4	Duplicate rows
SEM-5	Redundant column output

Logical Errors

The last category of errors, termed logical errors by Brass and Goldberg in [3] and further studied by Taipalus et al. in [36], includes many errors that trouble students considerably. Logical errors are those that cause a SQL query to be incorrect for a *particular* information request. Often, the presence of a logical error causes a query to be correct for a *different* information request than the stated request.

Examples of logical errors include confusion between `AND` and `OR`, missing parentheses in complex boolean expressions that combine `AND` with OR, joins on incorrect columns, or improper nesting of subqueries. As is the case with semantic errors, a database will (without error or warning) execute a query that contains logical errors. It is the user's responsibility to evaluate each query and its output to determine whether an error is present. Logical errors represent a particularly subtle and troublesome area for students who are learning SQL.

Table 4.9: Classification of Logical Errors, as Described by Taipalus et al. [36]

ID	**Description**
LOG-1	Operator error
LOG-2	Join error
LOG-3	Nesting error
LOG-4	Expression error
LOG-5	Projection error
LOG-6	Function error

4.5 Error Detection

The Lab 365 tool performs only a basic level of error detection and query validation. As described in Section 3.3, the tool determines query "correctness" by comparing results from a student's query against the output from a ground truth query within a single database instance. Students are instructed to ensure that their queries are generally valid; grades are assigned based on a manual review of SQL queries submitted by the student. Table 4.10 summarizes cases where queries were deemed correct by the Lab 365 tool, but were ultimately invalidated through manual review.

Table 4.10: Queries Deemed Correct by Lab 365 but Deemed Invalid by Manual Review (Percentage Average Across Exercises in Each Lab.)

Lab	**Valid**	**Violates Assignment**	**Invalid**
A	92%	5%	3%
B	96%	2%	2%
C	89%	0%	11%

In Table 4.10, the "Valid" column holds the average percentage of queries that were judged valid by Lab 365 and earned full credit after manual review. The "Violates Assignment" column represents cases where the submitted query was deemed valid by Lab 365 and would return correct results, but violated one or more assignment specifications. Examples include the use of `GROUP BY` or nested queries when these language features were expressly disallowed by the lab assignment. Finally, "Invalid" queries are those judged correct by Lab 365 which are, in fact, invalid due to logical or semantic errors. Common examples include:

- Hard-coded identifier(s) or key value(s) used instead of appropriate filter expressions;
- Comparisons performed using non-key values with incorrect assumptions about uniqueness, for example: an `IN` expression based on customer last name rather than unique customer identifier;
- Use of `LIMIT` to report a minimal or maximal value without accounting for the possibility of ties.

These common errors are described thoroughly to students in advance using examples, targeted lecture material, and detailed lab assignments, yet these errors clearly remain a point of confusion and misunderstanding. The manual validation process builds on the error taxonomy described in the preceding sections, allowing identification of specific SQL concepts that invite subtle logical errors that are difficult for novices to understand and avoid when writing SQL queries. In Sections 7.1.3 and 7.2.3, we suggest several ways to address these common areas of misunderstanding using new teaching material and extensions to the Lab 365 tool.

Using the comprehensive error taxonomy adapted from Taipalus, et al. [36] along with an extensive collection of lab exercises, we are well positioned to investigate our research questions. As outlined earlier in this chapter, we seek to identify the SQL concepts and query types that are most challenging for students to absorb and apply. The following chapter presents the results of our quantitative analysis based on a data set comprised of approximately 150,000 student-submitted queries spanning over 100 lab exercises which were designed to extensively cover commonly-used aspects of SQL's data retrieval capabilities.

5 Results

In this chapter, we present an investigation into the student problem-solving process spanning 116 SQL exercises and 28 distinct SQL concepts. We identify difficult SQL concepts and common errors, with a particular emphasis on SQL syntax errors. We analyzed 149,188 SQL statements representing the accumulated lab work in four sections of an introductory database course by 114 students, all of whom provided their informed consent (A.1).

5.1 Overview of Results

Of the 149,188 SQL statements we analyzed:

- 47,815 (32.1%) attempts consisted of syntax errors;
- 10,091 (6.8%) statements returned the correct result table which run against a single sample database instance;
- 4,126 (2.8%) ran without error but yielded zero rows, indicating a semantic error;
- 8,311 (5.6%) were executed without error using the "Check SQL" button but did not return correct results, implying a logical or semantic error;
- The remaining 78,845 (52.8%) attempts consisted of data exploration, partial attempts that may have been combined into complete solutions, semantic errors, or logical errors.

For each lab assignment, as described in Table 4.3, we first analyzed student data at a summary level. Figures 5.5, 5.6, and 5.7 offer an overview of student

success rates along with the average number of attempts submitted by each student. (Refer to Table 4.3 for a description of the lab assignments included in this study.) Guided by a high-level view of student behavior, we identified areas of particular difficulty. We confirmed the significance of these observations using the repeated measures ANOVA method described in Section 5.3.1. Finally, we performed detailed analysis on exercises that exhibit the combination of low success rate and a high average number of attempts.

5.2 Analysis by Instructor

We collected data in four sections of the same introductory class taught by two different instructors. As shown in Table 4.1, 56 study participants (49%) were enrolled in sections taught by instructor A. The remaining 58 students (51%) were enrolled in instructor B's sections. Both instructors taught based on a common set of learning objectives and general course content. Students in all sections completed an identical set of SQL lab exercises. Since course content and lab exercises are closely aligned between instructors, we expect the performance of students on lab exercises to be comparable between instructors. Figures 5.1, 5.2, 5.3, and 5.4 show the percent of student success and average attempts for each lab exercise, split by instructor.

To statistically determine differences due to instructor, we performed a two-sample, two-sided Kolmogorov-Smirnov (K-S) test [20], summarized in Table 5.1. We choose this statistical test because our samples indicate that we cannot assume a normal distribution of data. Figures 5.3 and 5.4 show histograms which bin exercises separately for each instructor based on percent success and average attempts. In Figure 5.3, for example, Instructor A saw all students succeed on

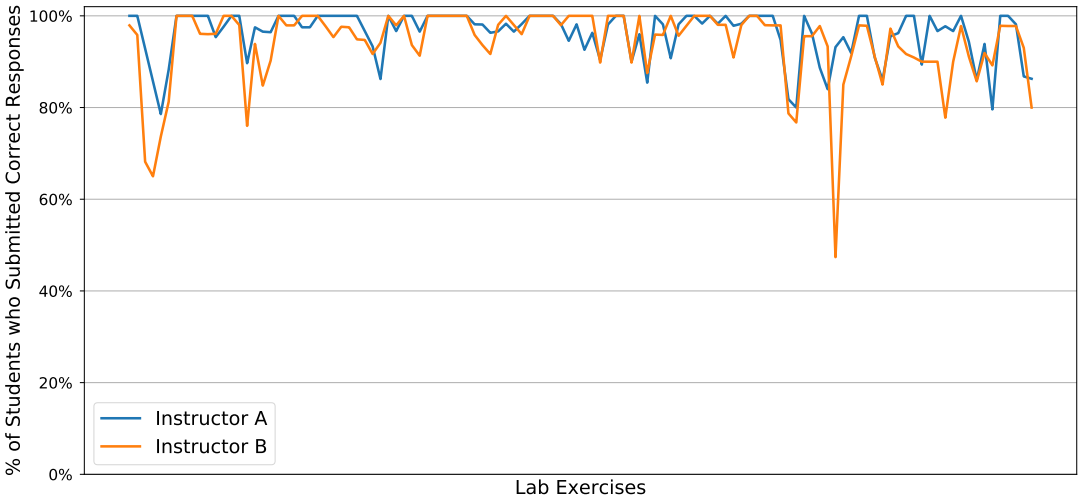

Figure 5.1: Percent Success by Instructor

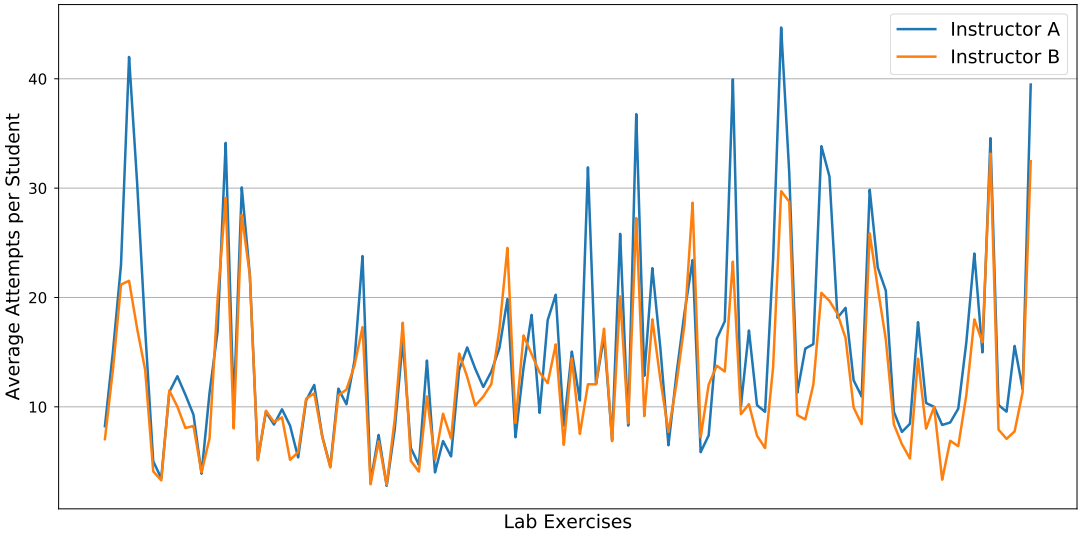

Figure 5.2: Average Attempts by Instructor

50 exercises, while Instructor B saw 100% student success on 35 exercises. Informally, these histograms visually indicate that the distributions are non-normal. We also statistically confirmed the non-normality of our samples using D'Agostino and Pearson's test [12].

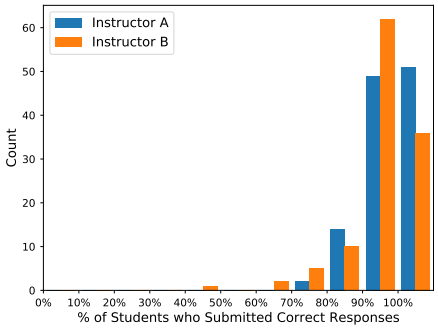

Figure 5.3: Histogram of Lab Exercises, Binned by Percent of Students who Successfully Solved the Exercise, Split by Instructor

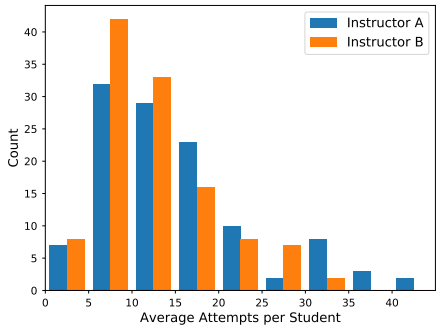

Figure 5.4: Histogram of Lab Exercises, Binned by Average Attempt Count per Student, Split by Instructor.

When applying the K-S test, the null hypothesis states that the two samples are taken from the same statistical distribution. Results of this K-S test are mixed for our instructor data. When comparing percent success between the two instructors, a low p-value (0.0022) causes us to reject the null hypothesis that the two samples are drawn from the same distribution. However, when comparing average student attempts between instructors, a relatively high p-value (0.1658) means that we cannot reject the null hypothesis. We conclude that student success percentages are significantly different between the two instructors in our study, but that average attempts per exercise are *not* significantly different between the two instructors. Further data collection and analysis is warranted, particularly for the lab exercises that exhibit large differences in percent success, as shown in Figure 5.1. In addition, analysis that controls for GPA or course exam performance would likely offer insight.

5.3 Lab A Analysis

For Lab A, we identified five exercises with the combination of high average number of attempts and low success rate. These exercises (CSU-4, CSU-6,

Table 5.1: Results from Two-sample K-S Test [20], Comparing Percent Success and Average Attempts for Each Exercise, by Instructor.

Sample	K-S Statistic	p-value
Percent Success	0.2414	0.0022
Average Attempts	0.1466	0.1658

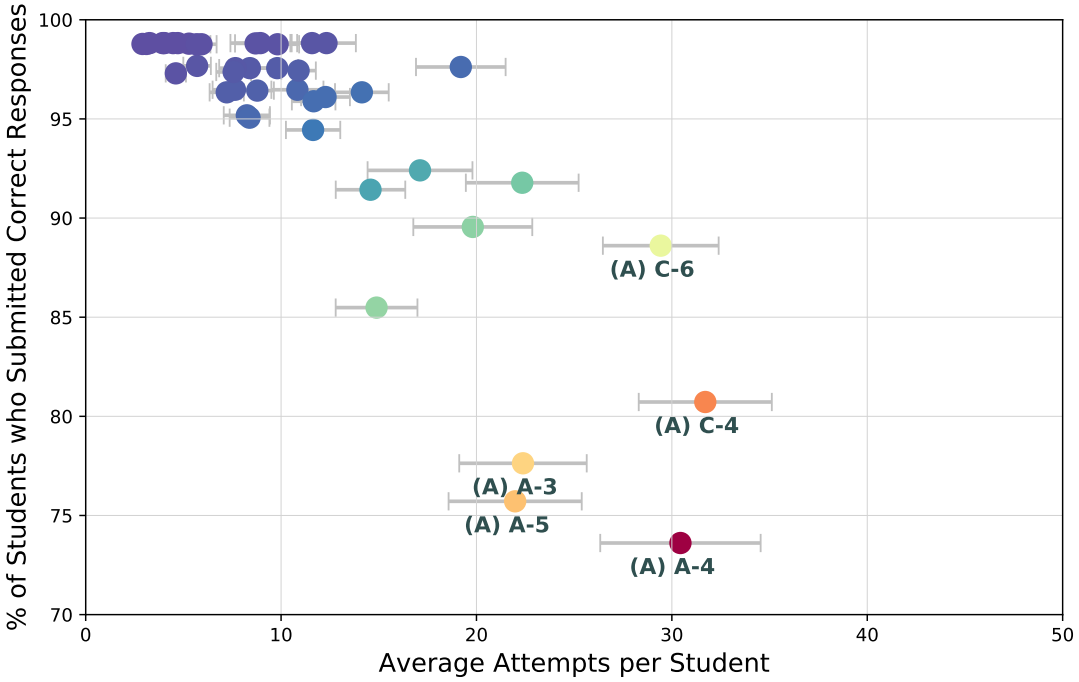

Figure 5.5: Summary of Lab A - Joins and WHERE. Each dot corresponds to a single lab exercise, plotted based on the percent of students who submitted a correct response (y-axis) versus the average number of attempts per student (x-axis). Average attempt counts reflect all students (incorrect and correct). Labels on data points are exercise identifiers, representing the first initial of the database name along with exercise number (eg. A-3 indicates the third exercise in the AIRLINES database)

AIRLINES-3, AIRLINES-4, and AIRLINES-5) are described in detail in Appendix D. Students' data for these three exercises, as represented in Figure 5.5, is summarized in Table 5.2.

Table 5.2: Exercises from Lab A on Which We Focus Our Detailed Analysis.

Exercise	**Query Concept(s)**	**Avg Attempts**	**Success Rate**
AIRLINES-3	Self-join, complex expression	23.3	77.4%
AIRLINES-4	Double self-join	24.1	59.6%
AIRLINES-5	Quintuple self-join	22.4	75.8%
CSU-4	Multi-table join, pivot	28.8	74.6%
CSU-6	Complex expressions	27.3	85%

5.3.1 Repeated Measures ANOVA

To confirm that certain exercises do, in fact, require a significantly higher number of attempts for students to complete, we analyzed student performance using one-way repeated measures ANOVA. In our analysis, the independent variable, often referred to as the *within-subjects* variable in a repeated measures experiment, is the query task itself. We choose as the dependent variable the number of attempts required for a student to produce a syntactically correct SQL query that is a valid response to the information request. Lab assignments are identical for all students. Each student functions as an experimental block, allowing control for factors that could cause variability between students.

With repeated measures ANOVA, we test for differences between mean number of attempts for two lab exercises: a control exercise and the exercise identified as problematic. In the case of Lab A, we compared CSU-4 and CSU-6 with CSU-1 and we compared AIRLINES-4 with AIRLINES-1. Both CSU-1 and AIRLINES-1 represent exercises for which 100% of students who attempted the exercise submitted a valid solution, so we treat these as the controls. The null hypothesis (H_0) states that the mean attempt counts are equal:

Hypothesis $\mathbf{H_0}$: $\mu_1 = \mu_2$,

where μ_n is the mean number of attempts by an individual student for a given lab exercise, n. The alternative hypothesis (H_1) states that the two means are different from one another:

Hypothesis $\mathbf{H_1}$: The mean number of attempts is significantly different for one of the lab exercises.

We calculate an F Statistic as follows:

$$F = \frac{\text{MS}_{\text{related groups}}}{\text{MS}_{\text{error}}} \tag{5.1}$$

Table 5.3 shows the computed F-values and p resulting from the ANOVA for each of the exercises. For all exercises, we reject the null hypothesis and we are able to say with confidence that the mean number of attempts is significantly different from the mean number for the control exercises: CSU-1 and AIRLINES-1.

Table 5.3: Lab A ANOVA Result for Exercises AIRLINES-3, AIRLINES-4, AIRLINES-5, CSU-4, and CSU-6

Exercise	**F Statistic**	**p-Value**
AIRLINES-3	12.042	0.001
AIRLINES-4	6.347	0.025
AIRLINES-5	5.681	0.031
CSU-4	32.109	0.000
CSU-6	38.122	0.000

5.3.2 Lab A Discussion

In Lab A, we identified self-joins (present in the AIRLINES exercises) as a particularly difficult concept, confirming similar observations from Ahadi et al. [2] and Taipalus et al. [36]. In addition, "pivot" operations, where rows and columns must be transposed, are a significant area of student confusion. Exercises, such as CSU-4, that require data to be pivoted stand out in the data charted in Figure 5.5 and, informally, were the source of a large number of student questions.

Lab A also revealed difficulty mapping complex English statements to appropriate SQL constructs (as in CSU-6). This last result was somewhat surprising, given that the "Expressions" SQL concept does not otherwise appear on our list of most difficult concepts. In this specific case, students appeared to face difficulty translating a highly complex English information request into a valid query. The difficulty may lie not in the underlying SQL concept, but in the challenging information request. The question of problem statement complexity was briefly discussed by Ahadi [2] in the context of grouping. A similar effect may be present in the case of CSU-6, and it would be interesting to explore further by designing additional exercises that are similar in nature.

5.4 Lab B Analysis

Lab B included exercises centered on SQL grouping, aggregation, and grouping restrictions. The repeated measures ANOVA results reported in Table 5.4 indicate that we cannot reject the null hypothesis for the first exercise listed (INN-5, p-value = 0.16). The INN-5 exercise relies on a combination of two relatively simple concepts (grouping and expressions) which may explain this statistical result. For the remaining two Lab B exercises that we highlight as particularly

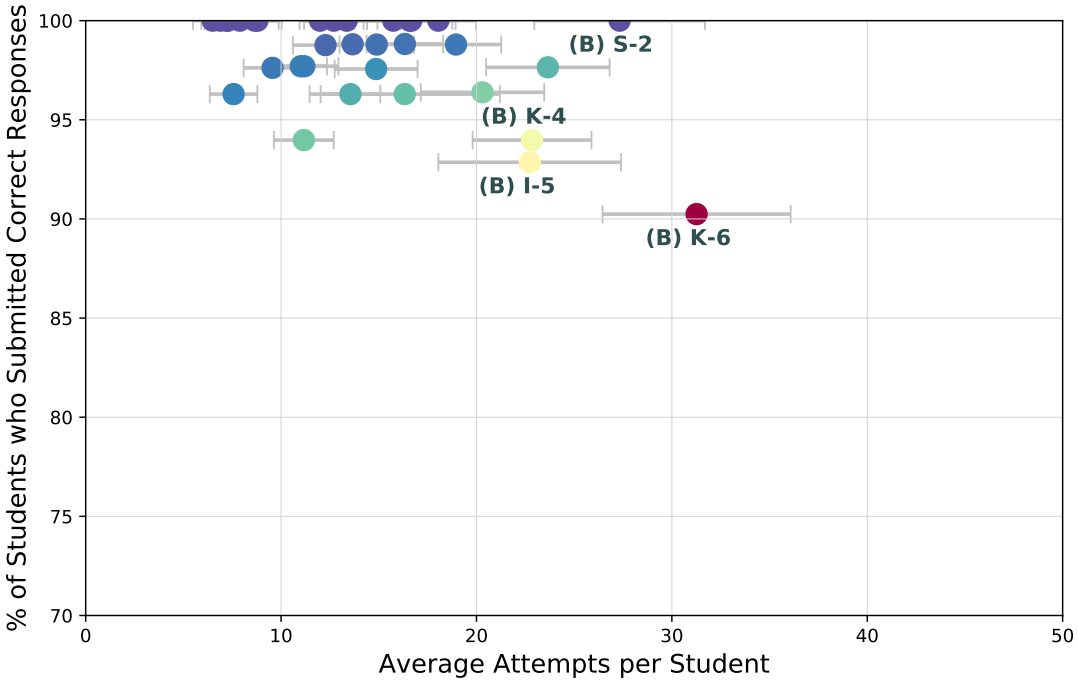

Figure 5.6: Summary of Lab B - GROUP BY and HAVING. Each dot corresponds to a single lab exercise, plotted based on the percent of students who submitted a correct response (y-axis) versus the average number of attempts per student (x-axis). Average attempt counts reflect all students (incorrect and correct). Labels on data points are exercise identifiers, representing the first initial of the database name along with exercise number (eg. I-5 indicates the fifth exercise in the INN database)

Table 5.4: Lab B ANOVA Result for Exercises INN-5, KATZENJAMMER-4, and KATZENJAMMER-6

Exercise	**F Statistic**	**p-Value**
INN-5	2.004	0.160
KATZENJAMMER-4	33.593	0.000
KATZENJAMMER-6	39.601	0.000

difficult (KATZENJAMMER-4, and KATZENJAMMER-6; listed in Table 5.5) we reject the null hypothesis and claim that these exercises are, in fact, more

difficult than control exercises: the mean number of attempts for each exercise is significantly different when compared to a control exercise.

Table 5.5: Exercises from Lab B on Which We Focus Detailed Analysis

Exercise	**Query Concept(s)**	**Avg Attempts**	**Success Rate**
INN-5	Grouping, Expressions	22.1	92.4%
KATZENJAMMER-4	Grouping restrictions, Self-join	22.3	93.8%
KATZENJAMMER-6	Grouping, Self-join	31.7	90.1%

5.5 Lab C Analysis

Table 5.6: Lab C ANOVA Result for Exercises BAKERY-8, BAKERY-9, and MARATHON-5

Exercise	**F Statistic**	**p-Value**
BAKERY-8	52.882	0.000
BAKERY-9	41.728	0.000
MARATHON-5	58.054	0.000

Lab C required students to apply the full power of SQL, including subqueries, grouping, and complex joins to respond to sophisticated information requests. We again performed repeated measures ANOVA tests (summarized in Table 5.6) to confirm that the mean number of attempts for the exercises listed in Table 5.7 are significantly different from the average number of attempts required to successfully solve a control exercise.

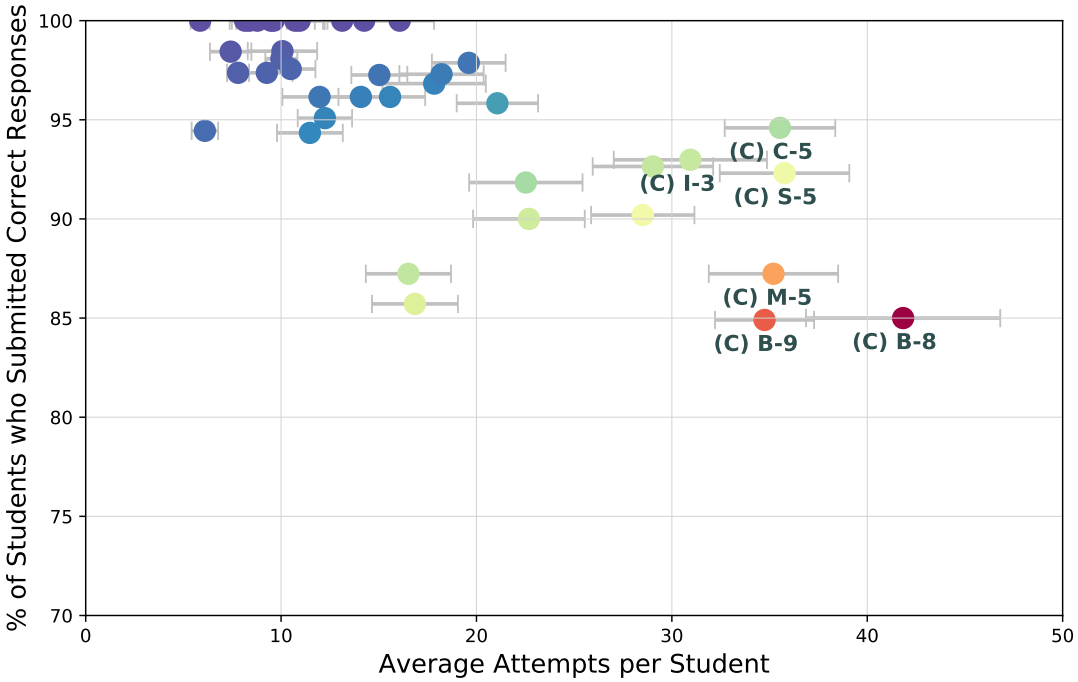

Figure 5.7: Summary of Lab C - Nested SQL. Each dot corresponds to a single lab exercise, plotted based on the percent of students who submitted a correct response (y-axis) versus the average number of attempts per student (x-axis). Average attempt counts reflect all students (incorrect and correct). Labels on data points are exercise identifiers, representing the first initial of the database name along with exercise number (eg. M-5 corresponds to exercise MARATHON-5 in Lab C)

Table 5.7: Exercises from Lab C on Which We Focus Detailed Analysis.

Exercise	**Query Concept(s)**	**Avg Attempts**	**Success Rate**
BAKERY-8	Subqueries: correlated and equal	42.2	85.0%
BAKERY-9	Grouping restrictions, Subqueries	34.3	84.9%
MARATHON-5	Full outer join	35.0	87.2%

5.5.1 Consolidated Discussion of Difficult Concepts

Across all labs, we identified several common concepts that seem to cause considerable difficulty for students. Specifically, the self-join, correlated subquery, and equal subquery concepts repeatedly appear in the exercises that are found to be most difficult. This observation likely aligns with the intuition of most database practitioners and instructors. We provided quantitative confirmation via our analysis. The grouping concept also appears across several of the most difficult exercises. Notably, however, in difficult exercises, grouping is always paired with another SQL concept. In exercises where grouping is the focal concept (often with its natural companion concept: aggregation) most students achieved a high level of success with relatively few attempts. This question of concept combinations is an interesting one, and we discuss future investigations in this area in Section 7.1.1.

5.6 Analysis of Errors

This section presents analysis and observations related to student problem solving behavior across all lab assignments. We begin with an investigation into common errors with a particular focus on syntax errors, specifically: (a) those errors that are unusually difficult for students to resolve, and (b) errors that occur as *terminal* errors. We define terminal errors as those that occur in final, unsuccessful attempts to solve a problem. In other words: when students abandon an exercise.

5.6.1 Common Syntax Errors

Several of our research questions relate to the frequency with which students encounter errors and to the identification of specific errors that cause considerable difficulty for students. In this section, we analyze common errors encountered by students. We pay particular attention to errors that require many attempts to resolve, as well as those errors that remain unfixed, causing a student to abandon an exercise without a correct response. We first investigate syntax error occurrence, based on calculations of the percentage of syntax errors encountered by each student for each exercise. We compute "percentage of syntax errors" by summing the total number of syntax errors encountered by a student for a given exercise, then dividing this sum by the total number of attempts submitted by the same student for that particular exercise.

Figures 5.8, 5.9, and 5.10 show syntax error percentages for all lab exercises, reported by syntax error classification (SYN-1 through SYN-6). Notably, we observe that the percentage of syntax errors remains relatively high throughout Lab C. Since labs A, B, and C are intended to build skills progressively, the elevated level of syntax errors in Lab C is unexpected. Early lab exercises are designed to introduce and reinforce basic SQL skills, with the objective of building to the more difficult exercises in Lab C. It is clear, however, that SQL syntax continues to trouble students even after significant practice.

Figure 5.11 plots the difference in error percentage between students who submitted a successful response and those who did not. This chart indicates that, for almost all lab exercises, unsuccessful students experienced a greater proportion of syntax errors than those students who submitted a successful solution. This confirms the observation by Ahadi et al. in [2] that syntax errors are a signifi-

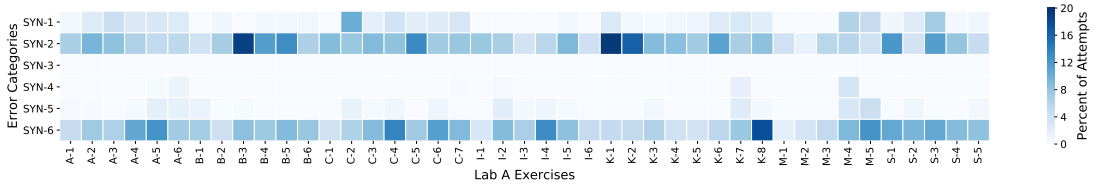

Figure 5.8: Syntax Error Percentages for All Exercises in Lab A (Per-Student Averages)

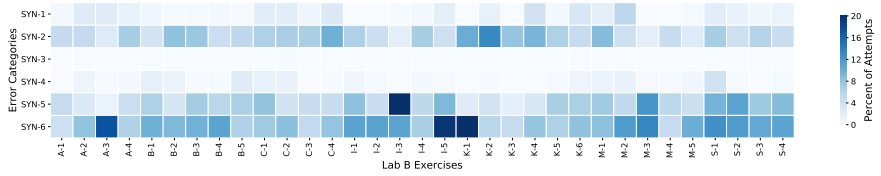

Figure 5.9: Syntax Error Percentages for All Exercises in Lab B (Per-Student Averages)

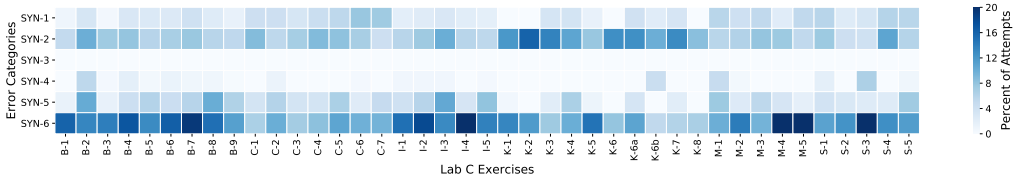

Figure 5.10: Syntax Error Percentages for All Exercises in Lab C (Per-Student Averages)

cant stumbling block for students learning SQL. We also observe that there is no difference in the set of syntax errors encountered by unsuccessful versus successful students. In other words, we failed to identify any syntax errors that were encountered solely by students who did not submit successful solutions. This provides a partial answer to our fourth research question: students, both successful and unsuccessful, encountered the full range of syntax errors. We do not yet know whether the same holds true for semantic and logical errors, since we do not in this thesis systematically perform fine-grained analysis of semantic and logical errors.

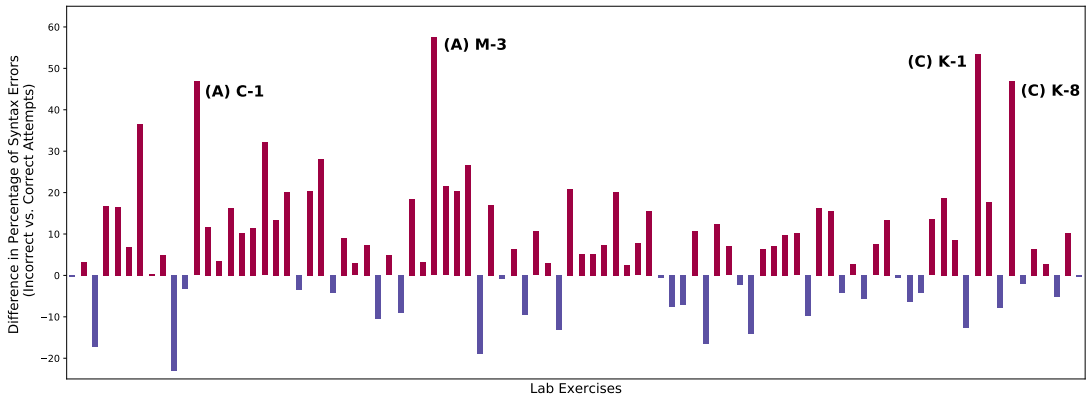

Figure 5.11: Syntax Errors for all Lab Assignments, Comparing Syntax Error Proportions Between Students who Correctly Solved the Exercise with Students who did not Submit a Correct Response. Bars above 0 represent exercises in which students who were unsuccessful encountered a greater percentage of syntax errors in relation to total attempts

5.6.2 Difficult-to-Resolve Syntax Errors

We now turn our attention to those syntax errors which are particularly difficult for students to resolve. To make this determination, we compute for each syntax error encountered the number of attempts required to produce a SQL query that executes without error. In this initial analysis, we do not scrutinize query structure. This simplifying omission raises the possibility that a student may have chosen to significantly revise, or entirely replace, her query during the process of error resolution. With this first analysis, we simply seek to identify syntax errors that significantly frustrate users. With this knowledge of particularly troublesome SQL syntax errors, an instructor may be able to provide to students additional exercises or instruction to aid in the learning process.

As Figure 5.12 shows, most syntax errors are fixed relatively quickly: 46% of errors are resolved in just one attempt, 83% require three or fewer attempts to resolve. This is expected and confirms our intuition that many syntax errors are

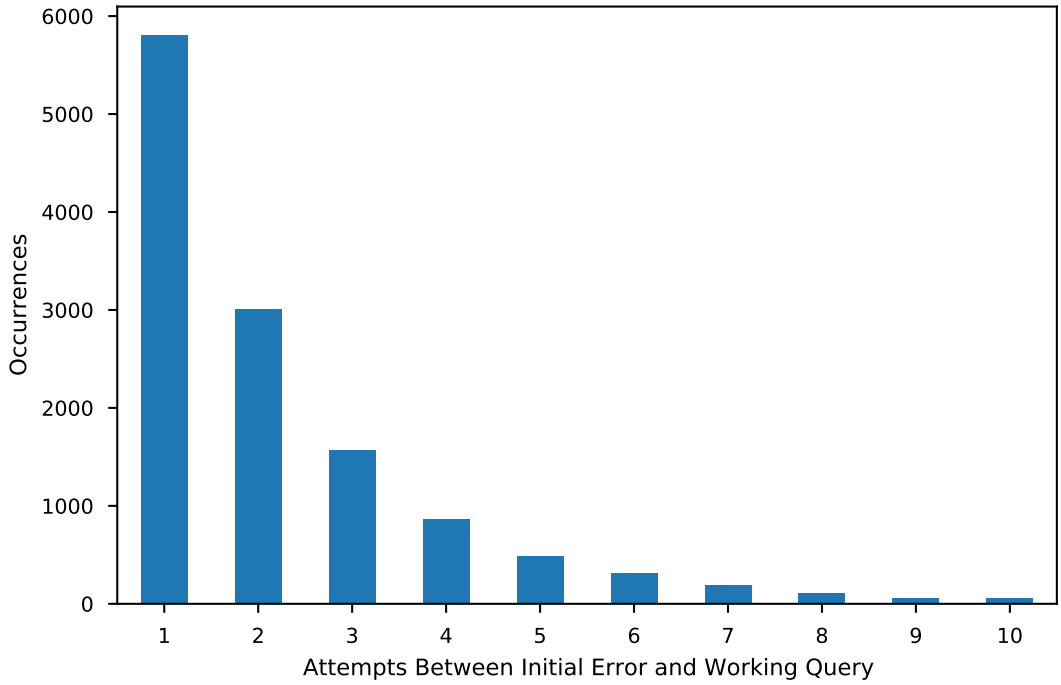

Figure 5.12: Syntax Errors, attempts Required to Fix. X-axis values represent the number of attempts required to resolve syntax errors; the y-axis indicates the number of distinct syntax errors encountered

simple misspellings, misplaced commas, or unbalanced delimiters, all of which are readily identified and fixed. However, this chart also reveals instances of syntax errors that are not so easily fixed. We focus on cases where a single syntax error required five or more attempts to resolve. Table 5.8 summarizes these difficult-to-resolve syntax errors using the categories defined by Taipalus et al. [36].

Among the most difficult-to-resolve syntax errors, the "Common syntax error" category (*SYN-6*) stands out as the most prevalent. This category is far less focused than the other categories, and includes a wide variety of SQL syntax errors, including comma omissions, mismatched delimiters, invalid clause ordering. In this thesis, we did not further break down the SYN-6 syntax error category.

Table 5.8: Cases Where More than Five Attempts Were Required to Resolve a Syntax Error. Error Categories (SYN-x) are as Defined by Taipalus et al. [36]. Some SYN-3 (data type mismatch) Errors may have been Mis-classified as SYN-6 (Common Syntax Error), an Issue we Discuss Further in Section 5.6.2

Category	**Occurrences**	**% of Students**
SYN-1	245	63.5%
SYN-2	409	87%
SYN-3	0	0% (*)
SYN-4	94	46.1%
SYN-5	328	84.3%
SYN-6	1143	99.1%

Such fine-grained categorized could be accomplished by extending our Lab 365 tool, and doing so may well yield valuable conclusions.

To detect data type mismatch errors (SYN-3), we relied on MySQL's default settings and parsing behavior, which we found to be both lenient (permitting implicit coercion in many cases) and lacking in detailed error reporting. We expect that some errors in the general SYN-6 category may be more accurately classified as SYN-3. Appropriate detection of SYN-3 errors would be best accomplished via extensions to the Lab 365 tool. This future work is discussed further in Section 7.2.

Setting aside the SYN-6 and SYN-3 categories, a large proportion of students experienced difficulty resolving almost all types of errors. Recall that Table 5.8 represents only those instances when a syntax error required more than five attempts to resolve. This finding offers further confirmation of the observations by

Ahadi et al. in [2]: students learning SQL struggle with its unfamiliar syntax, and this represents a hurdle in the learning process. In Section 7.1, we recommend several possible approaches to addressing these clear difficulties with SQL syntax through modifications to lab activities.

5.6.3 Terminal Attempts

When a student attempts a given problem but does not submit a correct response, we refer to the student's final attempt as a "terminal" attempt. In the following section, we investigate terminal attempts and we seek to identify patterns. In this analysis, we again apply the error taxonomy described in Section 4.4.2, seeking to identify errors that frequently occur as terminal attempts. Identification of these errors offers additional insight into the problem-solving process, highlighting key misunderstandings that can cause students to abandon problems.

In certain cases, a student's terminal attempt does not clearly represent an error, but instead appears to be an partially-formed, but incomplete, solution. We categorize these cases separately and remark that they warrant future analysis. It seems likely that detailed scrutiny of these non-error terminal attempts could offer valuable information. Non-error terminal attempts may simply indicate that a student ran out of time, or such attempts may reveal conceptual gaps that were difficult to bridge. Whatever the cause, it would be interesting to study the lead-up to these terminal attempts. We discuss this possible future work further in Section 7.2.4.

Comparing our results to a similar study of "persistent" errors by Taipalus et al. [35], in which the authors identified types of errors likely to remain unfixed throughout the problem-solving process, we find that our summary results gener-

Table 5.9: Categorization of Terminal Attempts. The category labeled "Other" represents final, incorrect, attempts that appear to be partially-formed or intermediate statements rather than true attempts. Examples include: isolated subqueries and simple exploratory queries such as: `SELECT * FROM Table`

Error Type	**Count**	**Percent**
Syntax Error	72	19.2%
Semantic Error	58	15.5%
Logical Error	78	20.8%
Other	167	44.5%

ally agree with the observations made by Taipalus et al. Specifically, in agreement with the Taipalus study, our data indicate that logical errors are most likely to appear in students' final responses. However, Taipalus et al. identify semantic errors are the second most likely, in contrast to our analysis which identified syntax errors as the second most likely. Our study considers a far smaller sample: we studied only 375 terminal queries compared to 8,773 final queries analyzed in the Taipalus study. Continued data gathering and study of persistent errors, and the steps leading up to these errors, will likely yield useful insight.

5.7 Quantifying Student Learning

At a high level, the labs and exercises we studied in this thesis were carefully designed to first introduce core SQL concepts. Subsequent exercises offer increasingly advanced practice. This sequencing is intended to allow students to learn, practice, then ultimately master SQL concepts. In an attempt to quantify this learn-practice-master progression, we choose several important SQL concepts

that appear repeatedly across exercises. We take advantage of the fact that the Lab 365 application records the order in which students complete exercises. Using this data, we chart student performance (and, we hope, improvement) over time on lab exercises that share the same primary concept.

We first analyze student improvement on exercises that center on the "Grouping Restrictions" concept. Although this concept appears in combination with many other concepts (as shown in Figures D.1, D.2, and D.3) we focus specifically on five exercises from Lab B where these Grouping Restriction is the focal concept, namely: AIRLINES-1, BAKERY-1, CSU-2, STUDENTS-2, and KATZENJAMMER-5. Figure 5.13 charts average student attempts (bars, left axis) and percent of attempts that yielded a syntax error (line, right axis) based on the order in which each student completed the five exercises. We interpret the decrease in both average attempts and proportion of syntax errors as an indicator of significant improvement through guided practice.

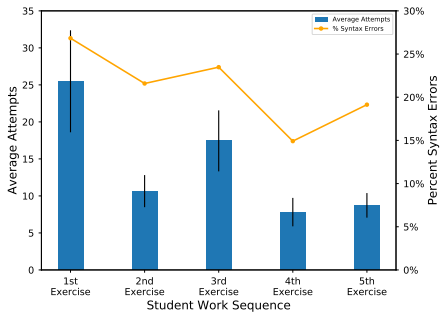

Figure 5.13: Student Work Sequence: Grouping Restrictions

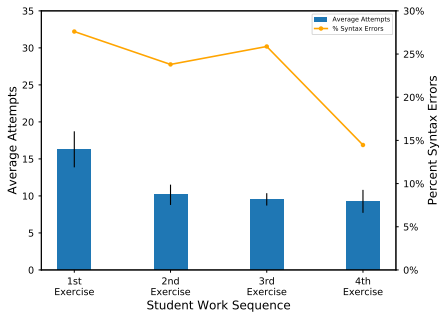

Figure 5.14: Student Work Sequence: Does Not Exist

We also investigated exercises that relied primarily on the "Does Not Exist" concept. This concept is the focus of four exercises from Lab C: BAKERY-1, BAKERY-3, KATZENJAMMER-1, KATZENJAMMER-5 (Figure 5.14.) Again, the downward trend in both attempts required and syntax errors encountered

offers confirmation that, through practice, students learn to efficiently apply advanced SQL concepts such as "Does not Exist."

We concentrated our analysis in this section on the "Grouping Restrictions" and "Does Not Exist" concepts. These two concepts are useful in many real-world query scenarios, and each individually often represents the focal concept within queries that also involve fundamental concepts such as joins and expressions. The SQL labs we studied in this thesis include a large number of similar, seemingly-repetitious, exercises that test students' understanding of a relatively small number of SQL concepts. The analysis in this section confirms the value of overlap in lab exercises. We did not, however, systematically determine the exact number of practice problems that would be appropriate for each SQL concept (which may expand or shrink the current list of lab exercises.) We discuss the possibility of adaptive problem generation as future work in Section 7.2.1.

5.8 Principal SQL Concepts

As mentioned in previous sections and in related work by Ahadi et al. [2], most students face little difficulty learning and applying basic SQL concepts, such as joins, expressions, and projection. However, students often struggle with more advanced concepts such as self-joins, grouping, grouping restrictions, and "does not exist." In this section, we rank SQL concepts based on subjective complexity (Table 5.10), then identify a single primary concept for each lab exercise. With this list, we analyze student success rates and average attempts by SQL concept.

Table 5.10 lists 18 core SQL concepts, along with a subjective complexity score (1-18) for each concept. Concepts are divided into two levels: fundamental and advanced. We omit in this section ten "extended" SQL concepts, represent-

Table 5.10: Core SQL Concepts Ranked by Subjective Complexity. The "Exercises" column lists the number of lab exercises for which each concept is the primary or focal concept.

Rank	Concept	Level	Exercises	Discussed In
1	Single-Table	Fundamental	4	[2] [36]
2	Expressions	Fundamental	3	[2] [36]
3	Expressions with Nesting	Fundamental	1	[2] [36]
4	Multi-Table	Fundamental	25	[2] [36]
5	Aggregate Functions	Fundamental	3	[2] [36]
6	Grouping	Fundamental	12	[2] [36]
7	Set Operations	Fundamental	1	
8	Uncorrelated Subquery	Fundamental	5	[2] [36]
9	Equal Subqueries	Fundamental	4	[36]
10	Correlated Subquery	Fundamental	3	[36]
11	Grouping Restrictions	Advanced	21	[2] [36]
12	Does Not Exist	Advanced	5	[36]
13	Parameter Distinct	Advanced	9	[36]
14	Non-Equi-Join	Advanced	3	
15	Pivot	Advanced	0	
16	Self-Join	Advanced	14	[2] [36]
17	Relational Division	Advanced	2	
18	Full Outer Join	Advanced	1	

ing complementary language features that are useful only in combination with one or more of the 18 core concepts listed in Table 5.10. Extended concepts include: Ordering, Scalar Functions, Computed Selection, Computed Projection,

and Computed Grouping. A further discussion of these extended SQL concepts can be found in Section 4.3.1.

Following the per-exercise analysis we performed in Sections 5.3, 5.4, and 5.5, Figure 5.15 charts each SQL concept based on average attempts and percent of students who successfully submitted correct responses. To compute per-concept statistics, we identified a single primary concept for each of the 116 lab exercises. In agreement with our previous analysis in Sections 5.3, 5.4, and 5.5, we observe that "Self-Join" and "Correlated Subquery" are troublesome concepts. Several other advanced SQL concepts also appear in the bottom right quadrant (low success, high average attempts) of the chart, namely: Set Operations, Full Outer Join, and Relational Division. Our data set for these three concepts is relatively small; they are poorly represented in the existing set of lab exercises. We also note that the Pivot concept does not appear in the lower right quadrant of Figure 5.15. This is due to the fact that Pivot does not serve as the primary, focal concept for any current lab exercises. Aside from these minor differences, the analysis in this section offers additional confirmation of observations described previously in this thesis.

5.9 Concept Associations

Building on the analysis of primary concepts in the previous section, we next investigate student success by SQL concept. For each student, we computed the set of SQL concepts (from Table 5.10) for which the student submitted at least one correct response. We then applied the Apriori algorithm described by Agrawal et al. in [1] to identify frequently-occurring subsets of SQL concepts. (Such subsets are often referred to as "frequent itemsets.") In this way, we identified areas where

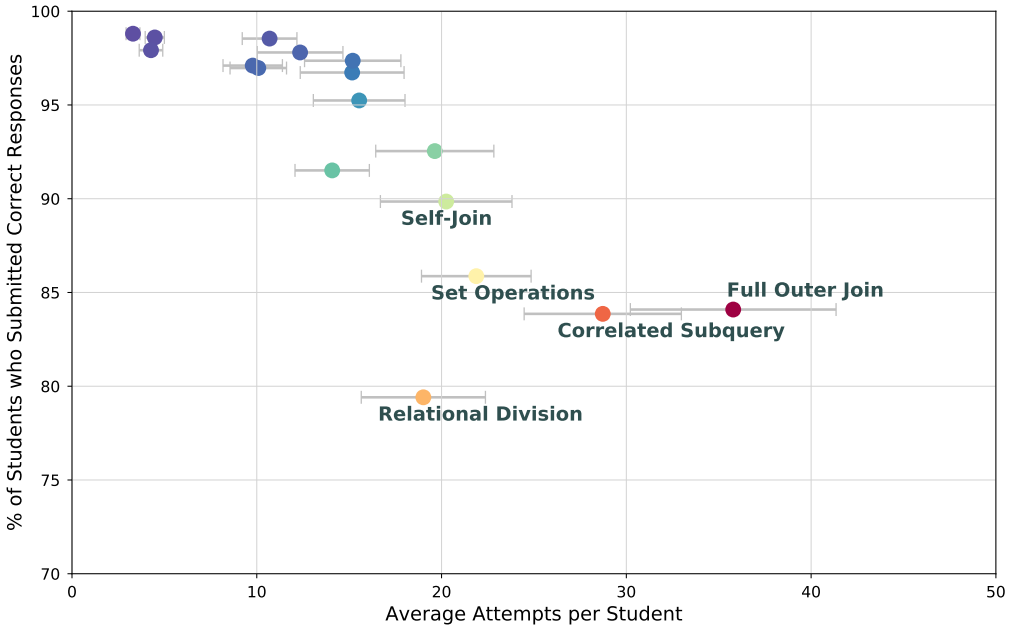

Figure 5.15: Average Attempts and Percent Success for the 18 Core SQL Concepts Listed in Table 5.10

student success in certain concepts is associated with success in other concepts. In particular, we identify several frequent subsets that include both fundamental and advanced concepts.

Table 5.11 lists SQL concept subsets that occur for 90% or more of students, when considering only concepts for which a student submitted at least one correct response. This list highlights associations between SQL concepts student success. Specifically, we note that the Grouping and Grouping Restrictions concepts appear in most of the frequent itemsets, Furthermore, these two concepts appear to be linked with several advanced concepts, including Self-Join. This seems counter-intuitive, since, on the surface, grouping has little to do with the self-join operation. We suspect that this association reveals a link from solid student understanding of unfamiliar relational concepts to mastery of SQL as a whole. Once a student has learned to successfully apply core concepts such

as joins and grouping, we believe that the student is better table to learn and apply advanced SQL concepts. This observation, and future analysis proposed in Section 7.2.2, offers key information related to the overall research questions we posed in this thesis.

Table 5.11: Frequently-Occurring ($\geq$ 90% support) Subsets of SQL Concepts in Successful Student Responses. The "Support" column represents the percentage of students who successfully solved at least one exercise containing each concept in the listed subset.

SQL Concept Subset	**Support**
Grouping, Grouping Restrictions	97.7%
Grouping, Parameter Distinct	97.7%
Parameter Distinct, Grouping Restrictions	97.7%
Grouping, Parameter Distinct, Grouping Restrictions	97.7%
Grouping, Multi-Table	93.1%
Multi-Table, Parameter Distinct, Grouping Restrictions	93.1%
Grouping, Multi-Table, Parameter Distinct, Grouping Restrictions	93.1%
Grouping, Self-Join, Grouping Restrictions	93.1%
Self-Join, Multi-Table, Grouping Restrictions	90.8%
Grouping, Multi-Table, Grouping Restrictions, Self-Join, Parameter Distinct	90.8%

Internal Validity. Each student is free to complete exercises within a given lab in whatever order he or she chooses, raising the possibility that fatigue or other factors may play a role in student performance. In addition, since our experiment uses repeated measures design, we explicitly do not address possible effects related to the order in which students work on lab exercises.

The Lab 365 tool we developed could affect participants' abilities to solve SQL exercises. Although the tool is designed as a minimal SQL interface, usability and user interaction decisions invariably have been made (either intentionally or not.) These decisions have not been rigorously tested for impact on study results.

For most exercises in this study, students were permitted to view expected output at any time during SQL query development. This matches previous similar studies ([2], [35]), which also allowed students to view expected query output. In most real-world situations, however, a SQL developer does not have the ability to preview results. The ability to preview results may artificially simplify certain query tasks.

External Validity. We conducted our study using a specific RDBMS (MySQL) within a quarter-long database course which did not extensively cover database design topics such as the Entity-Relationship model or normalization. Our results may not generalize to courses in which a different RDBMS is used, or to courses in which students are exposed to additional database topics.

At the beginning of our study, students are informed that their interactions with the Lab 365 application will be recorded for analysis. This knowledge may alter student problem-solving behavior, introducing Hawthorne effects. For example, a student might devote an atypical amount of effort to manually inspecting

SQL code before testing each query in an attempt to prevent the system from recording too many incorrect attempts.

7 Conclusions and Future Work

In this thesis, we originally set out to investigate the learning process in an introductory database course, and to quantitatively study troublesome SQL concepts and common errors. The database lab tool we developed proved effective. Lab 365 facilitated collection of a large volume of data related to the student problem-solving process, and promises to be a useful tool in the future.

Our results are largely consistent with similar previous studies. In concurrence with Ahadi et al. [2], we observe that SQL syntax errors are a significant source of student frustration. Analyzing the most difficult SQL concepts, we find that self-joins, correlated subqueries, and (to a lesser extent) grouping restrictions are most troublesome. These findings are similar to results reported by Taipalus et al. [36], and Ahadi et al. [2]. In comparison to these previous studies, we studied a larger collection of SQL exercises based on a wider variety of database structures and problem domains. We also investigated several SQL concepts that, to our knowledge, have not been previously studied in an educational context. With the benefit of these extensions to previous studies, we performed an initial study of SQL concept combinations. Drawing from our analysis, we are well-positioned to suggest improvements to lab exercises and to validate the effect of these changes on the student learning process.

7.1 Recommendations for Lab Improvement

The capabilities of the tool that we constructed in this thesis, combined with our initial analysis, offer a sound basis for future investigation and improvement. In this section we provide several recommendations for changes to lab exercises that are informed by the results that we presented in this thesis.

7.1.1 Concept Combinations

Certain concept combinations pose significant difficulty for students. We specifically identified the following challenging combinations:

1. Self-Join & Expressions
2. Muli-Table Joins & Pivot (recall that a "pivot" operation transposes rows and columns)
3. Grouping & Self-Join
4. Grouping & Subqueries (correlated and equal)

We suggest designing additional lab exercises and in-class examples to provide practice applying these difficult concept combinations. To rule out any effects related to specific database structures or application domains, we suggest designing these new exercises for all current (and future) lab databases. In this way, we would ensure broad student exposure to troublesome SQL concept combinations. One challenge is to ensure that additional exercises do not simply constitute repetitive, tedious work for students. To address this the online lab application could adapt to the individual student learning process, a possibility discussed further in Section 7.2.1.

7.1.2 Recently-Added SQL Features

The introductory database course described in this thesis focused mainly on SQL as it is specified in the SQL-92 standard. However, the course includes brief coverage of recent language features, such as window functions and common table expressions. Several existing lab exercises lend themselves to solutions based on

these extensions to the SQL standard. We find that some students successfully apply these SQL features. However, none of the lab exercises studied in this thesis were specifically designed to require the use of modern SQL features or standard language extensions. We suggest designing additional lab exercises to encourage students to master these powerful new language features.

7.1.3 SQL Interpretation Skill

As described in Section 2.4, the introductory database course on which this study is based first introduced basic relational algebra building blocks. Students then learned to assemble these building blocks into simple queries which, in turn, were combined together in a "scaffolded" manner to construct compound queries to solve complex tasks. Current lab exercises focus almost exclusively on this bottom-up approach to SQL learning.

Most beginning students quickly learn to build syntactically valid SQL queries. However, as revealed in this study and in previous similar studies ([2], [35], and [36]), students experience difficulty formulating semantically and logically correct solutions to complex query tasks. Given the prevalence of semantic and logical errors, it would be interesting to study the effect of devoting additional instructional attention to the skill of *deconstructing* complex SQL queries to identify the intended behavior and to identify errors. Without this important skill, the problem-solving process may too often devolve into simple trial-and-error instead of an intentional process based on careful design and evaluation on the part of students.

7.1.4 Emphasis on ANSI-Standard SQL

The analysis of common syntax errors presented in this study revealed many areas of confusion related to vendor-specific SQL features. Examples include: errors related to vendor-specific scalar functions, the use of non-standard keyword and string delimiters, and attempts to use the proprietary operators such as `TOP` or `PIVOT`. Such confusion may be difficult to avoid given the large number of SQL implementations, many of which support lengthy lists of legacy features. In addition, given the age of SQL as a language, large volumes of reference material of varying quality and currency exist both off- and online.

Although the use of non-standard syntax is in often benign, in some cases it can represent deeper areas of confusion or cases where a student's approach has defaulted to random trial and error. For these reasons, we recommend that future lab exercises include activities that encourage students to prefer ANSI-standard syntax, with specific emphasis on those areas of the standard which enjoy broad RDBMS support. This could be enforced through the addition of syntax validation or "linting" to the online lab tool.

7.2 Future Work

The Lab 365 tool constructed as part of this thesis makes possible many types of data collection and analysis. The tool, along with the analysis presented in this thesis, provides a common baseline and platform for future studies of student problem solving in a database environment. In addition, since the tool captures full SQL statements, future work could include an in-depth study of query structure. For example, the data would allow clustering of attempts based on the structural similarity or equivalence of queries, perhaps offering insight into

problem-solving patterns shared by multiple students. Several additional possible avenues for future work are described in the following sections.

7.2.1 Curriculum Mapping & Dynamic Exercise Assignment

The existing set of lab exercises is already quite large and significant overlap exists: the same concepts and concept combinations are often repeated, as shown in Tables D.1, D.2, and D.3. To accommodate further expansion in the exercise set, and to address the possibility of student fatigue due to repetition, the Lab 365 tool could dynamically assign exercises. In this way SQL concepts could be methodically introduced, practiced, and reinforced following the "curriculum mapping" approach advocated by Hausman [17]. If a student has successfully mastered a given concept, the system could skip (for that student) the remaining exercises designed around the already-mastered concept. Or, if a student is unable to solve exercises related to a certain concept combination, the system could reinforce each individual concept, then present additional exercises related to the concept combination with which the student struggled. Dynamic exercise assignment and curriculum mapping could be a powerful way to promote mastery of SQL without overwhelming students with a huge volume of practice exercises.

7.2.2 Formalizing Concept Associations

Section 5.9 identified frequently-occurring subsets of SQL concepts in students' correct responses. Following from this initial analysis, additional data analysis should be performed to determine how success on fundamental concepts translates into mastery of more advanced concepts. Such analysis would be particularly valuable for the concepts that appear in the difficult exercises identified in

Sections 5.3, 5.4, and 5.5. Identification of SQL concepts that predict success on on the most difficult exercises would permit further fine-tuning of labs and other instructional material.

7.2.3 Automatic Error Detection

To improve the granularity of our error analysis, particularly with regard to semantic and logical errors, it would be useful to include in our Lab 365 tool automatic classification of errors, based on the taxonomy proposed by Taiaplus [35] and adopted in this thesis. Automatic classification of errors would require some combination of SQL parsing and construction of multiple database instances (or "hidden tests") to expose errors. Doing so would have the added benefit of providing to students concrete examples of the subtle ways in which an apparently-correct SQL can, in fact, be invalid.

7.2.4 Further Investigation of Terminal Attempts

As described in Section 5.6.3, when a student does not submit a correct response for a given exercise, the nature of the final attempt(s) submitted can offer insight. In some cases, however, the final attempt does not clearly represent an error or misunderstanding. Instead, the final attempt may be an exploratory query of some sort that is not intended as a solution. In these situations, tracing back through the lead-up to final attempts could offer valuable information about the problem-solving process. Developing an improved understanding of the various reasons why students have difficulty formulating valid SQL queries was the stated goal of this thesis. We believe that a detailed understanding of student difficulties will allow the teaching and learning environment to be improved. The nature of

terminal or persistent errors may well offer key insight into factors that impede the learning process.

7.2.5 Measuring the Impact of Lecture

This thesis focused primarily on the learning process as demonstrated through practical lab exercises. However, significant learning also take place in lecture sessions and impromptu question-and-answer sessions during labs and instructor office hours. Future investigations could include analysis of lecturing styles, delivery methods, or even the use of specific examples. All of these factors (and more) undoubtedly impact student learning and could extend in many interesting ways the baseline analysis presented in this thesis.

BIBLIOGRAPHY

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[2] Alireza Ahadi, Vahid Behbood, Arto Vihavainen, Julia Prior, and Raymond Lister. Students' syntactic mistakes in writing seven different types of sql queries and its application to predicting students' success. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 401–406, New York, NY, USA, 2016. ACM.

[3] Stefan Brass and Christian Goldberg. Proving the safety of sql queries. In *Proceedings of the Fifth International Conference on Quality Software*, QSIC '05, pages 197–204, Washington, DC, USA, 2005. IEEE Computer Society.

[4] Stefan Brass and Christian Goldberg. Semantic errors in sql queries: A quite complete list. *J. Syst. Softw.*, 79(5):630–644, May 2006.

[5] Ricardo Caceffo, Steve Wolfman, Kellogg S. Booth, and Rodolfo Azevedo. Developing a computer science concept inventory for introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 364–369, New York, NY, USA, 2016. ACM.

[6] L. Cagliero, L. De Russis, L. Farinetti, and T. Montanaro. Improving the effectiveness of sql learning practice: A data-driven approach. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 01, pages 980–989, July 2018.

[7] Donald D. Chamberlin and Raymond F. Boyce. Sequel: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '74, pages 249–264, New York, NY, USA, 1974. ACM.

[8] HOCK C. CHAN, BERNARD C.Y. TAN, and KWOK-KEE WEI. Three important determinants of user performance for database retrieval. *Int. J. Hum.-Comput. Stud.*, 51(5):895–918, November 1999.

[9] Shumo Chu, Daniel Li, Chenglong Wang, Alvin Cheung, and Dan Suciu. Demonstration of the cosette automated sql prover. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1591–1594, New York, NY, USA, 2017. ACM.

[10] Shumo Chu, Konstantin Weitz, Alvin Cheung, and Dan Suciu. Hottsql: Proving query rewrites with univalent sql semantics. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, pages 510–524, New York, NY, USA, 2017. ACM.

[11] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[12] RALPH D'AGOSTINO and E. S. PEARSON. Tests for departure from normality. Empirical results for the distributions of b2 and b1. *Biometrika*, 60(3):613–622, 12 1973.

[13] Michael V. Ellis. Repeated measures designs. *The Counseling Psychologist*, 27(4):552–578, 1999.

[14] D. L. Evans, G. L. Gray, S. Krause, J. Martin, C. Midkiff, B. M. Notaros, M. Pavelich, D. Rancour, T. Reed-Rhoads, P. Steif, R. Streveler, and K. Wage. Progress on concept inventory assessment tools. In *33rd Annual Frontiers in Education, 2003. FIE 2003.*, volume 1, pages T4G–1, Nov 2003.

[15] Sally Fincher, Josh Tenenberg, and Anthony Robins. Research design: Necessary bricolage. In *Proceedings of the Seventh International Workshop on Computing Education Research*, ICER '11, pages 27–32, New York, NY, USA, 2011. ACM.

[16] Paolo Guagliardo and Leonid Libkin. A formal semantics of sql queries, its validation, and applications. *Proc. VLDB Endow.*, 11(1):27–39, September 2017.

[17] Jerome J. Hausman. Mapping as an approach to curriculum planning. *Curriculum Theory Network*, 4(2-3):192–198, 1974.

[18] Colin A. Higgins, Geoffrey Gray, Pavlos Symeonidis, and Athanasios Tsintsifas. Automated assessment and experiences of teaching programming. *J. Educ. Resour. Comput.*, 5(3), September 2005.

[19] Michael Hilton and David S. Janzen. On teaching arrays with test-driven learning in webide. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 93–98, New York, NY, USA, 2012. ACM.

[20] J. L. Hodges. The significance probability of the smirnov two-sample test. *Ark. Mat.*, 3(5):469–486, 01 1958.

[21] Chun-Nan Hsu and Craig A. Knoblock. Advances in knowledge discovery and data mining. chapter Using Inductive Learning to Generate Rules for

Semantic Query Optimization, pages 425–445. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[22] http://users.csc.calpoly.edu/ dekhtyar/. Alexander dekhtyar (professional website). `http://users.csc.calpoly.edu/~dekhtyar/`. Accessed: 2019-05-03.

[23] Gokhan Kul, Duc Thanh Luong, Ting Xie, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. Similarity measures for sql query clustering. *IEEE Transactions on Knowledge and Data Engineering*, Jul 2018.

[24] Jacob Thornton Mark Otto and Bootstrap contributors. Bootstrap: Html, css, and js library. `https://getbootstrap.com/`. Accessed: 2019-04-23.

[25] Antonija Mitrovic, Brent Martin, and Michael Mayo. Using evaluation to shape its design: Results and experiences with sql-tutor. *User Modeling and User-Adapted Interaction*, 12(2-3):243–279, March 2002.

[26] Stellan Ohlsson. Constraint-based student modeling. In Jim E. Greer and Gordon I. McCalla, editors, *Student Modelling: The Key to Individualized Knowledge-Based Instruction*, pages 167–189, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[27] Andrew Pavlo and Matthew Aslett. What's really new with newsql? *SIGMOD Rec.*, 45(2):45–55, September 2016.

[28] Inc. Pivotal. Spring boot: an open source java-based framework. `https://spring.io/projects/spring-boot`. Accessed: 2019-03-13.

[29] Sam Saarinen, Shriram Krishnamurthi, Kathi Fisler, and Preston Tunnell Wilson. Harnessing the wisdom of the classes: Classsourcing and machine learning for assessment instrument generation. In *Proceedings of the*

50th ACM Technical Symposium on Computer Science Education, SIGCSE '19, pages 606–612, New York, NY, USA, 2019. ACM.

[30] Shazia Sadiq, Maria Orlowska, Wasim Sadiq, and Joe Lin. Sqlator: An online sql learning workbench. *SIGCSE Bull.*, 36(3):223–227, June 2004.

[31] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, October 1980.

[32] John B. Smelcer. User errors in database query composition. *Int. J. Hum.-Comput. Stud.*, 42(4):353–381, April 1995.

[33] XB Software. Webix: Javascript ui library and framework for cross-platform web development. `https://webix.com/`. Accessed: 2019-04-19.

[34] María José Suárez-Cabal and Javier Tuya. Using an sql coverage measurement for testing database applications. In *Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering*, SIGSOFT '04/FSE-12, pages 253–262, New York, NY, USA, 2004. ACM.

[35] Toni Taipalus and Piia Perälä. What to expect and what to focus on in sql query teaching. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 198–203, New York, NY, USA, 2019. ACM.

[36] Toni Taipalus, Mikko Siponen, and Tero Vartiainen. Errors and complications in sql query formulation. *ACM Trans. Comput. Educ.*, 18(3):15:1–15:29, August 2018.

[37] Margus Veanes, Pavel Grigorenko, Peli Halleux, and Nikolai Tillmann. Symbolic query exploration. In *Proceedings of the 11th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering*, ICFEM '09, pages 49–68, Berlin, Heidelberg, 2009. Springer-Verlag.

[38] Charles Welty and David W. Stemple. Human factors comparison of a procedural and a nonprocedural query language. *ACM Trans. Database Syst.*, 6(4):626–649, December 1981.

A Experimental Materials

A.1 Informed Consent Form

INFORMED CONSENT TO PARTICIPATE IN A RESEARCH PROJECT:

Study of Student Learning of SQL

INTRODUCTION

This form asks for your agreement to participate in a research project studying student learning of the Structured Query Language (SQL). Your participation involves completion of laboratory exercises using a web-based application. It is expected that your participation will take no extra time commitment from you. There are no risks anticipated with your participation. Others may benefit from your participation. If you are interested in participating, please review the following information.

PURPOSE OF THE STUDY AND PROPOSED BENEFITS

The purpose of the study is to gather and analyze data related to the processes students follow when solving SQL query exercises in an introductory database course.

Potential benefits associated with the study include enhancements to lecture and lab materials for future students.

YOUR PARTICIPATION

If you agree to participate, you will allow your work on the lab exercises in CSC 365 to be used in analysis for this research. Study data will be gathered through normal use of the CSC 365 course lab environment.

Your participation will require no additional time commitment from you, beyond regular course lab activities.

Your course grade will not be affected by your decision whether or not to participate in this study.

PROTECTIONS AND POTENTIAL RISKS

Please be aware that you are not required to participate in this research, refusal to participate will not involve any penalty or loss of benefits to which you are otherwise entitled, and you may discontinue your participation at any time. The researcher may terminate your participation at any time for the following reasons: when sufficient data has been collected. There are no risks anticipated with your participation in this study.

Your confidentiality will be protected. All exported data used for analysis will be anonymized. Personally identifiable information (PII) will be excluded from all exports.

RESOURCES AND CONTACT INFORMATION

If you should experience any negative outcomes from this research, please be aware that you may contact Andrew Von Dollen (avondoll@calpoly.edu) or Alex Dekhtyar (dekhtyar@calpoly.edu)

This research is being conducted by Andrew Von Dollen (Graduate Teaching Associate) and Alex Dekhtyar (Professor) in the Department of Computer Science at Cal Poly, San Luis Obispo. If you have questions regarding this study or would like to be informed of the results when the study is completed, please contact the researcher(s) at avondoll@calpoly.edu or dekhtyar@calpoly.edu.

If you have concerns regarding the manner in which the study is conducted, you may contact Dr. Michael Black, Chair of the Cal Poly Institutional Review Board, at (805) 756-2894, mblack@calpoly.edu, or Ms. Debbie Hart, Compliance Officer, at (805) 756-1508, dahart@calpoly.edu.

AGREEMENT TO PARTICIPATE

If you agree to voluntarily participate in this research project as described, please indicate your agreement by signing below. Please retain a copy of this form for your reference, and thank you for your participation in this research.

Signature of Volunteer Printed Name Date

B Lab Databases

This section provides an Entity-Relationship (E-R) Diagram and brief description for each of the lab databases studied in this thesis. All databases contain hand-crafted or public domain data, and were originally created by Alex Dekhtyar [22].

AIRLINES is a graph-oriented database which contains information about airlines, airports, and flights between 100 different airports.

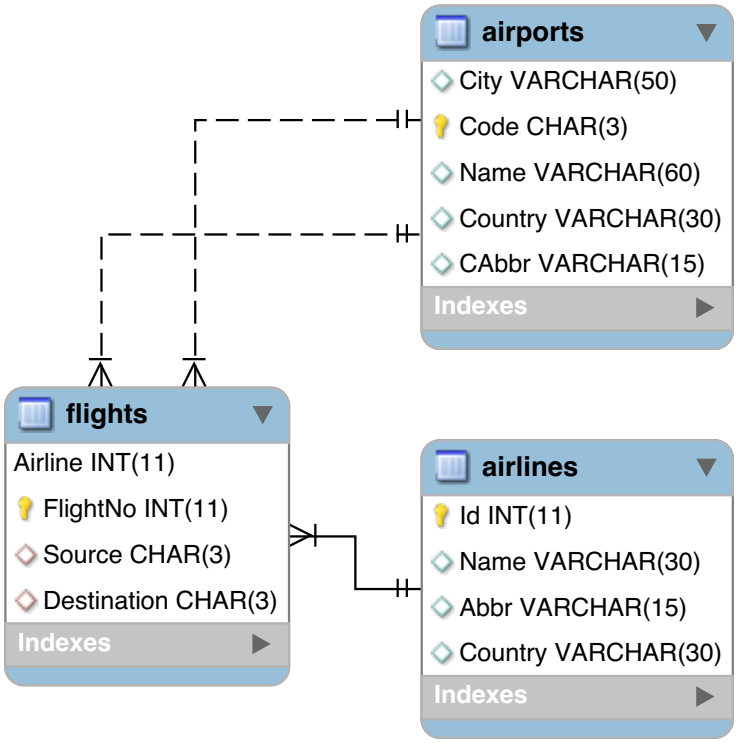

Figure .1: AIRLINES Database - ER Diagram

The BAKERY database contains sales detail for a small, fictitious bakery. It is organized using a simple Online Transaction Processing (OLTP) structure, in which customer receipts have a on-to-many relationship with receipt line items.

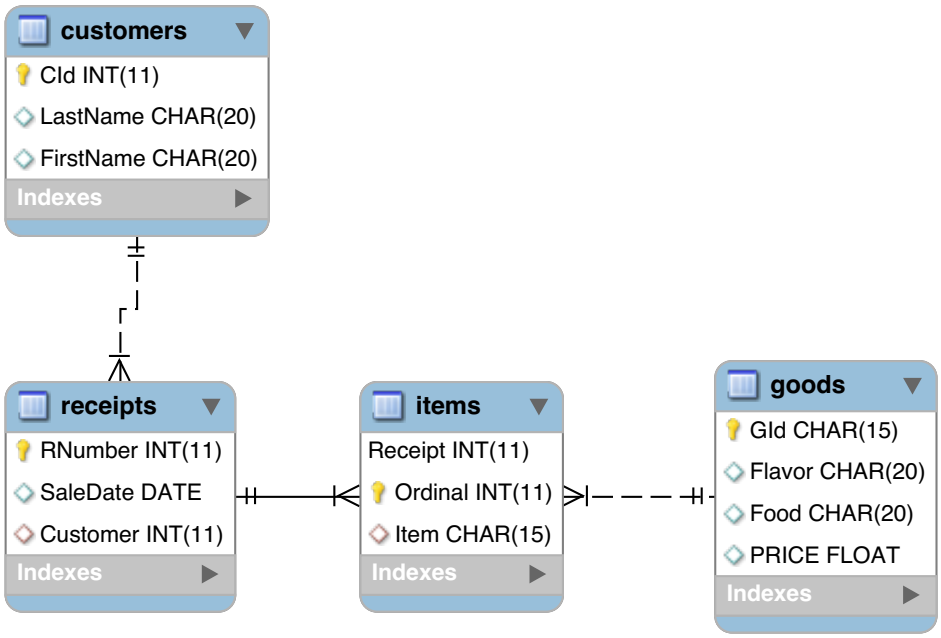

Figure .2: BAKERY Database - ER Diagram

CARS is a normalized database containing statistics about 406 car models produced worldwide between 1970 and 1982. The data were originally distributed by the American Statistical Association (ASA) Committee on Statistical Graphics in 1983 for a visualization competition.

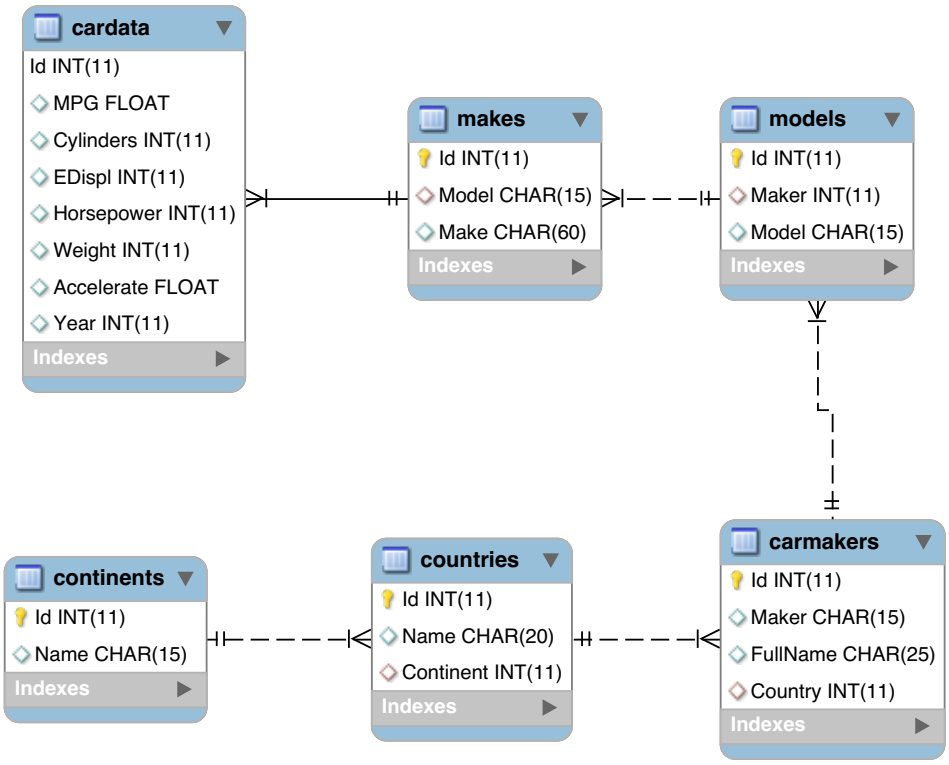

Figure .3: CARS Database - ER Diagram

The CSU database includes fee and enrollment data from the California State University's 23 campus system of higher education. This database uses a normalized Online Analytical Processing (OLAP) structure.

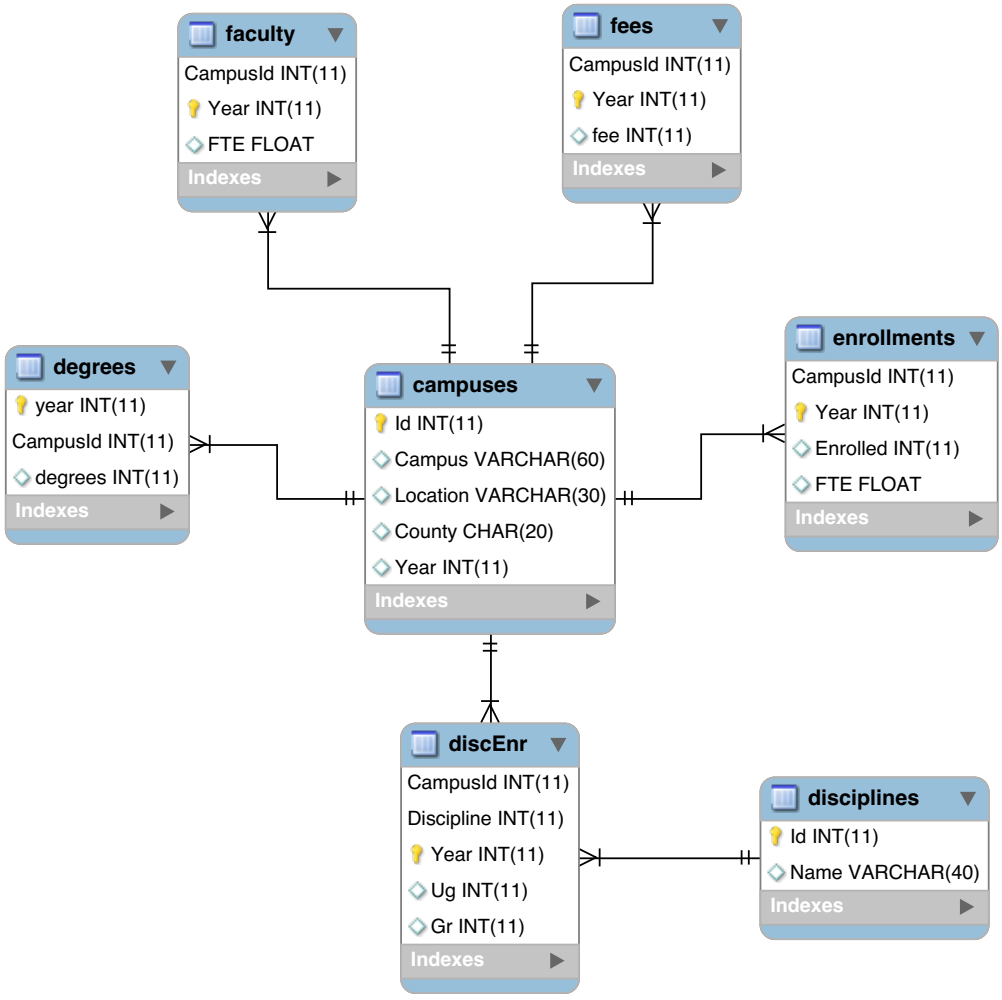

Figure .4: CSU Database - ER Diagram

INN contains simple OLTP-structured reservation data for a fictional Bed & Breakfast, including room information and reservation details.

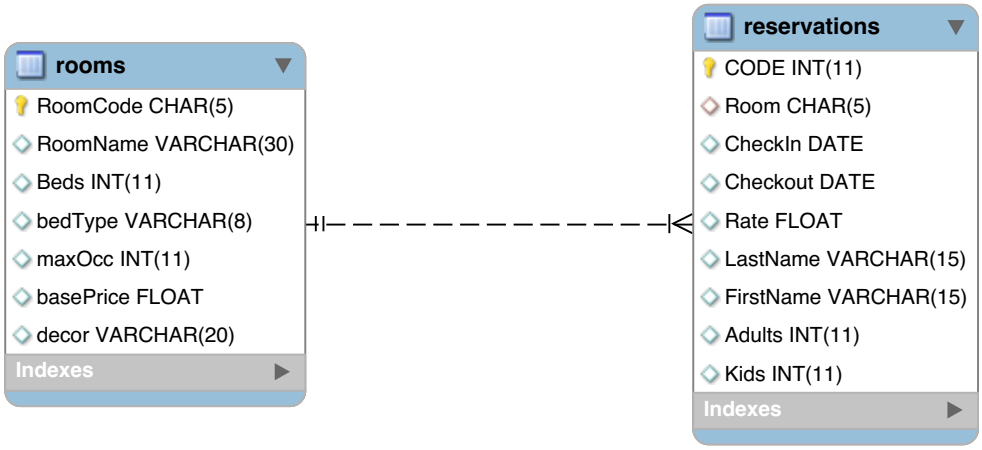

Figure .5: INN Database - ER Diagram

KATZENJAMMER is a fully normalized database that contains data related to the musical career of a Norwegian pop band named Kaztenjammer.

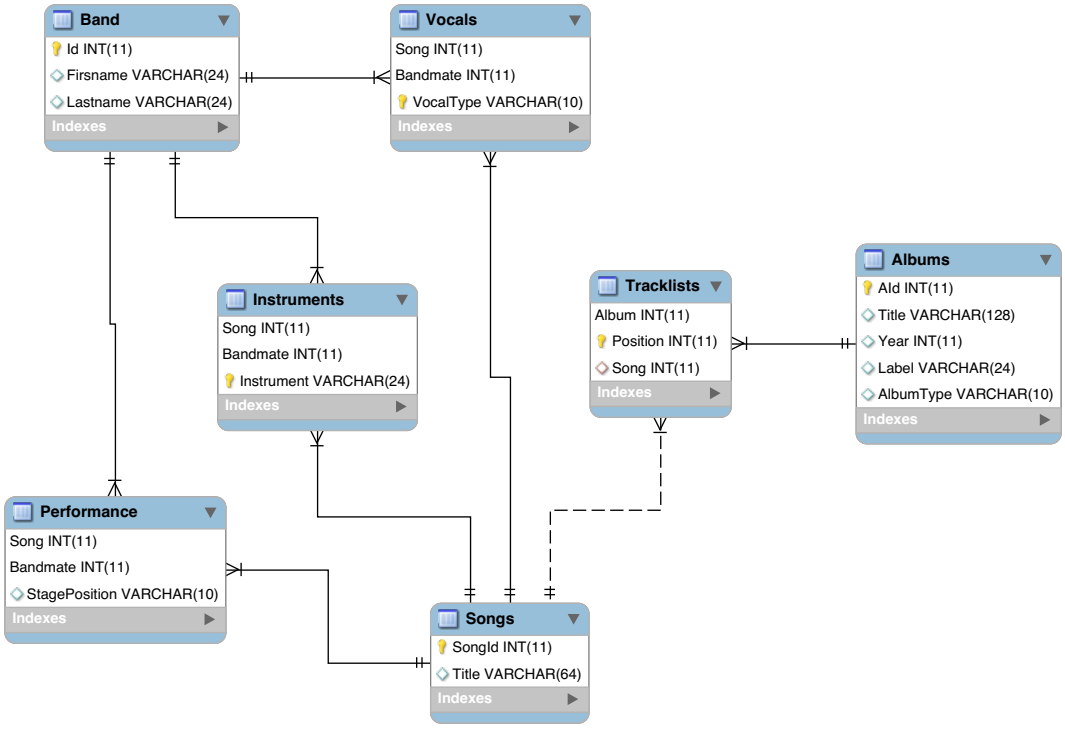

Figure .6: KATZENJAMMER Database - ER Diagram

The MARATHON database contains a single table that holds results (placing, pace, etc.) from a half marathon in New England. This database represents a "universal table" design, and is the focus of exercises that relate to the self-join SQL concept.

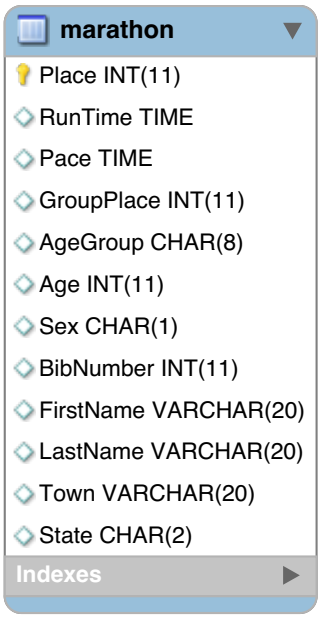

Figure .7: MARATHON Database - ER Diagram

STUDENTS is a simple normalized database that contains data about students, classrooms, and teachers at a small, fictitious elementary school.

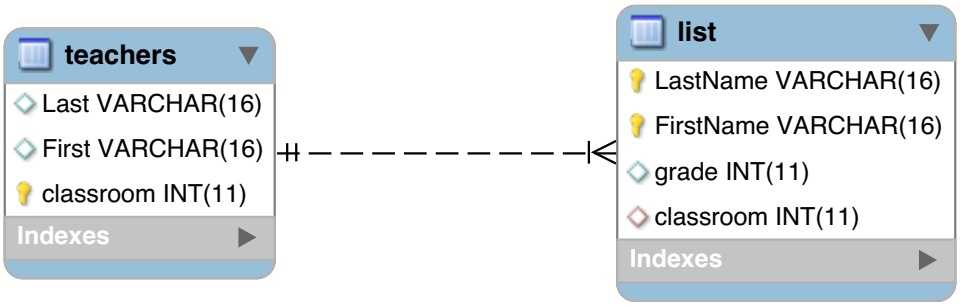

Figure .8: STUDENTS Database - ER Diagram

WINE is a *partially* normalized database that holds ratings for a variety of wines produced in California.

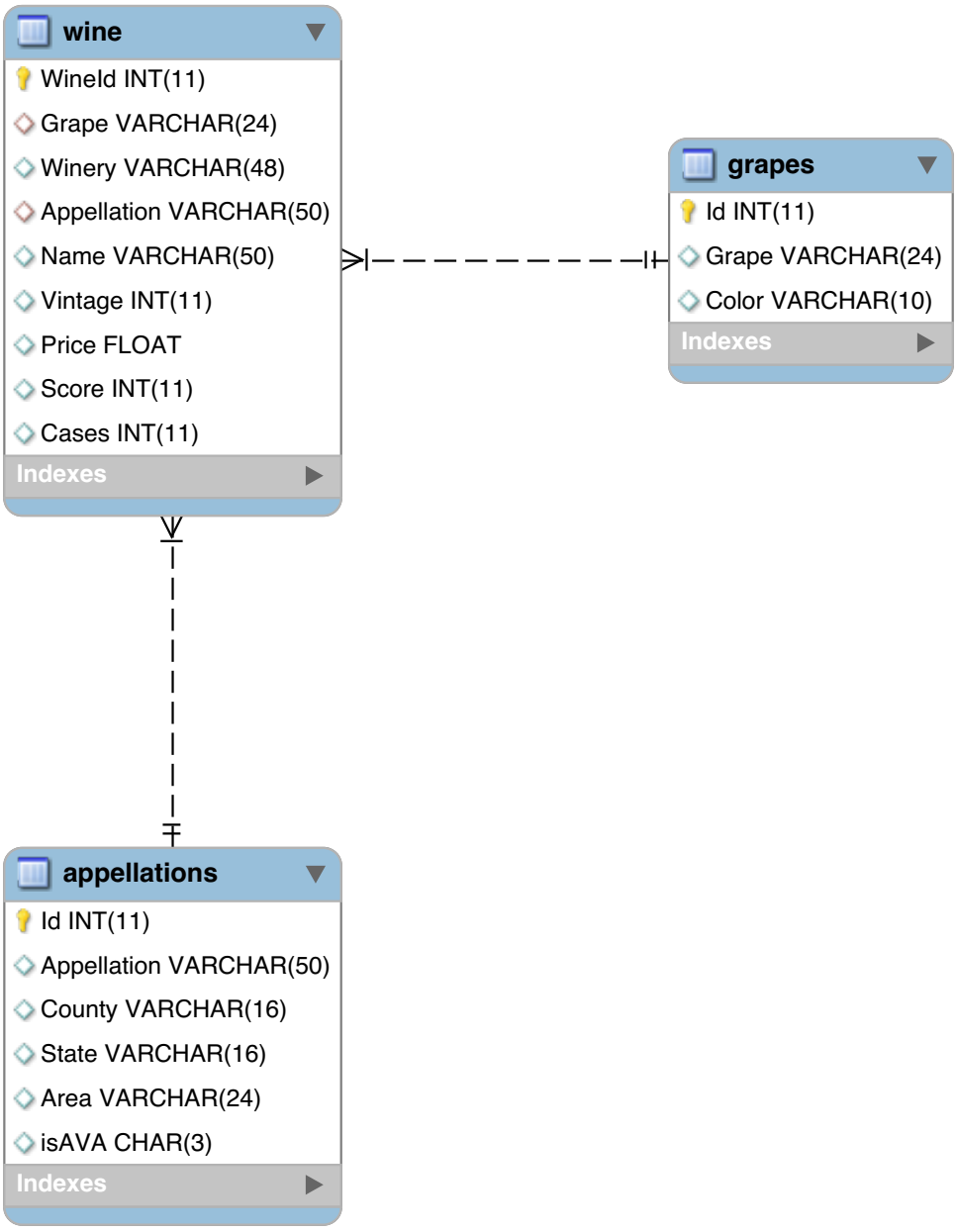

Figure .9: WINE Database - ER Diagram

C Classification of Lab Exercises

The tables in this section (Figures .10, .11, and .12) depict the distribution of concepts across lab exercises studied in this thesis. A gray square indicates that the exercise (horizontal rows) includes a particular SQL concept (vertical columns). The rightmost and bottom columns hold row and column totals. Row totals correspond to the number of concepts in a given exercise; column totals represent a count of the number of exercises which include each concept. Concepts marked with an asterisk (*) were discussed by Ahadi et al. in [2]; those marked with a plus sign (+) were described by Taipalus et al. in [36]. Concepts with neither an asterisk nor a plus sign represent SQL features that were not specifically discussed in prior work.

D Lab Exercises

The following section lists English information requests assigned to students as lab exercises. These lab exercises were originally prepared by Alex Dekhtyar [22]. They were used with permission during the study described in this thesis.

Each lab completed by students consists of 30-40 similar exercises. The list presented to students is grouped according to the subject database (BAKERY, CSU, LATZENAJMMER) and sorted based on expected difficulty within that database. As an illustration, BAKERY-1 is expected to be the easiest exercise within the BAKERY dataset, while BAKERY-4 requires more difficult concepts or an advanced combination of SQL features. Students are permitted to complete lab exercises *within* each lab in any order they choose. The brief assignment for Lab A reads:

> Write SQL queries that return information as requested. Each information need must be met with a single SQL statement. Do not use grouping (GROUP BY) or aggregation for these queries. You may refer only to codes/names included in the question. Do not use numeric IDs or key values.

In the exercises below, text that appears in `monospaced` type are values that appear in the database instances. Students are expected to use these values (rather than underlying primary key or ID values) when constructing filter expressions.

D.1 Selected Exercises from Lab A

AIRLINES-1 Find all airlines that have at least one flight out of `AXX` airport. Report the full name and the abbreviation of each airline. Report each name only once. Sort the airlines in alphabetical order.

AIRLINES-2 Find all destinations served from the `AXX` airport by `Northwest`. Report flight number, airport code and the full name of the airport. Sort in ascending order by flight number.

AIRLINES-3 Find all *other* destinations that are accessible from `AXX` on only `Northwest` flights with exactly one change-over. Report pairs of flight numbers, airport codes for the final destinations, and full names of the airports sorted in alphabetical order by the airport code.

AIRLINES-4 Report all pairs of airports served by both `Frontier` and `JetBlue`. Each airport pair must be reported exactly once (if a pair X,Y is reported, then a pair Y,X is redundant and should not be reported).

AIRLINES-5 Find all airports served by ALL five of the airlines listed below: `Delta`, `Frontier`, `USAir`, `UAL` and `Southwest`. Report just the airport codes, sorted in alphabetical order.

BAKERY-1 Find all `Chocolate` flavored items on the menu whose price is under $5.00. For each item output the flavor, the name (food type) of the item, and the price. Sort your output in descending order by price (highest price to lowest).

BAKERY-2 Report the prices of the following items (a) any `Cookie` priced above $1.10, (b) any `Lemon` flavored items, or (c) any `Apple` flavored item except for the `Pie`. Output the flavor, the name (food type) and the price of each pastry. Sort the output in alphabetical order by the flavor and then pastry name.

BAKERY-3 Find all types of `Cookie` purchased by `KIP ARNN` during the month of `October 2007`. Report each cookie type (flavor) exactly once in alphabetical order by flavor.

CSU-1 Report all campuses from `Los Angeles` county. Output the full name of campus in alphabetical order.

CSU-2 For each year between `1994` and `2000` (inclusive) report the number of students who graduated from `California Maritime Academy`. Output the year and the number of degrees granted. Sort output by year.

CSU-3 Report undergraduate and graduate enrollments (as two numbers) in `Mathematics`, `Engineering` and `Computer and Info. Sciences` disciplines for both `Polytechnic` universities of the CSU system in 2004. Output the name of the campus, the discipline and the number of graduate and the number of undergradu-

ate students enrolled. Sort output by campus name, and by discipline for each campus.

CSU-4 Report graduate enrollments in `2004` in `Agriculture` and `Biological Sciences` for any university that offers graduate studies in both disciplines. Report one line per university (with the two grad. enrollment numbers in separate columns), sort universities in descending order by the number of Agriculture graduate students.

CSU-5 Find all disciplines and campuses where graduate enrollment in `2004` was at least three times higher than undergraduate enrollment. Report campus names and discipline names. Sort output by campus name, then by discipline name in alphabetical order.

CSU-6 Report the total amount of money collected from student fees (use the full-time equivalent enrollment for computations) at `Fresno State University` for each year between `2002` and `2004` inclusively, and the amount of money (rounded to the nearest penny) collected from student fees per each full-time equivalent faculty. Output the year, the two computed numbers sorted chronologically by year.

D.2 Selected Exercises from Lab B

Lab B covers grouping, aggregation, and grouping restriction. Many exercises in this lab also build on concepts introduced in Lab A, including joins and expressions. The brief lab assignment reads:

> Each information request in this lab can (and must) be represented by either a single SELECT statement (possibly including aggregate operations, GROUP BY and HAVING clauses), or by a number of SELECT statements combined using the UNION operator.

INN-1 For each room, report the total revenue for all stays and the average revenue per stay generated by stays in the room that began in the months of `September`, `October` and `November`. Sort output in descending order by total revenue. Output full room names.

INN-2 Report the total number of reservations that began on `Friday`, and the total revenue they brought in.

INN-5 For each room report how many nights in calendar year `2010` the room was occupied. Report the room code, the full name of the room and the number of occupied nights. Sort in descending order by occupied nights. (Note: it has to be number of nights in 2010. The last reservation in each room can go beyond December 31, 2010, so the extra nights in 2011 need to be deducted).

KATZENJAMMER-1 For each performer (by first name) report how many times she sang lead vocals on a song. Sort output in descending order by the number of leads.

KATZENJAMMER-2 Report how many different instruments each performer plays on songs from the album `Le Pop`. Sort the output by the first name of the performers.

KATZENJAMMER-4 Report how many times each performer (other than `Anne-Marit`) played `bass balalaika` on the songs where `Anne-Marit` was positioned on the `left` side of the stage. Sort output alphabetically by the name of the performer.

KATZENJAMMER-6 Report all instruments (in alphabetical order) that were played by three or more people.

D.3 Selected Exercises from Lab C

Lab C covers all SQL language features, with an emphasis on nested SQL. Most exercises in Lab C are also designed to reinforce concepts introduced in Labs A and B. The brief lab assignment reads:

> Each information need must be addressed with a SELECT statement that returns a single result set. This statement may include multiple levels of nesting, grouping and aggregation, and/or UNION. You are permitted to use any ANSI-standard SQL feature, as well as MySQL-specific scalar functions.

BAKERY-1 Find all customers who did not make a purchase between `October 5` and `October 11` (inclusive) of `2007`. Output first and last name in alphabetical order by last name.

BAKERY-8 For every type of `Cake` report the customer(s) who purchased it the largest number of times during the month of `October 2007`. Report the name of the pastry (flavor, food type), the name of the customer (first, last), and the number of purchases made. Sort output in descending order on the number of purchases, then in alphabetical order by last name of the customer, then by flavor.

BAKERY-9 Output the names of all customers who made multiple purchases (more than one receipt) on the latest day in `October` on which they made a purchase. Report names (first, last) of the customers and the earliest day in October on which they made a purchase, sorted in chronological order.

MARATHON-1 Find the state(s) with the largest number of participants. List state code(s) sorted alphabetically.

MARATHON-2 Find all towns in Rhode Island (`RI`) which fielded more female runners than male runners for the race. Report the names of towns, sorted alphabetically.

MARATHON-5 For each town in `Connecticut` report the total number of male and the total number of female runners. Both numbers shall be reported on the same line. If no runners of a given gender from the town participated in the marathon, report 0. Sort by number of total runners from each town (in descending order) then by town.

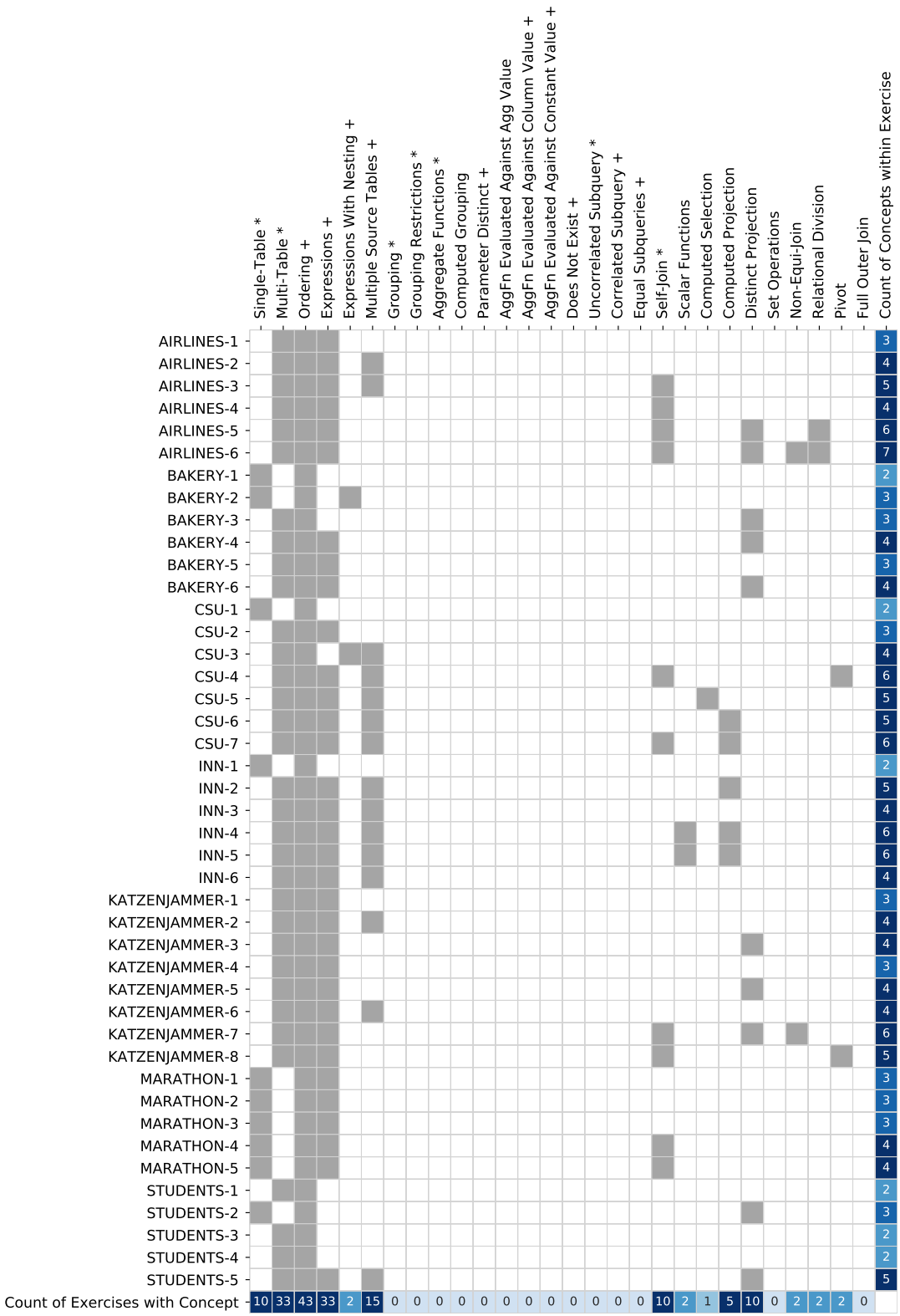

Figure .10: Lab A Exercises and Concepts

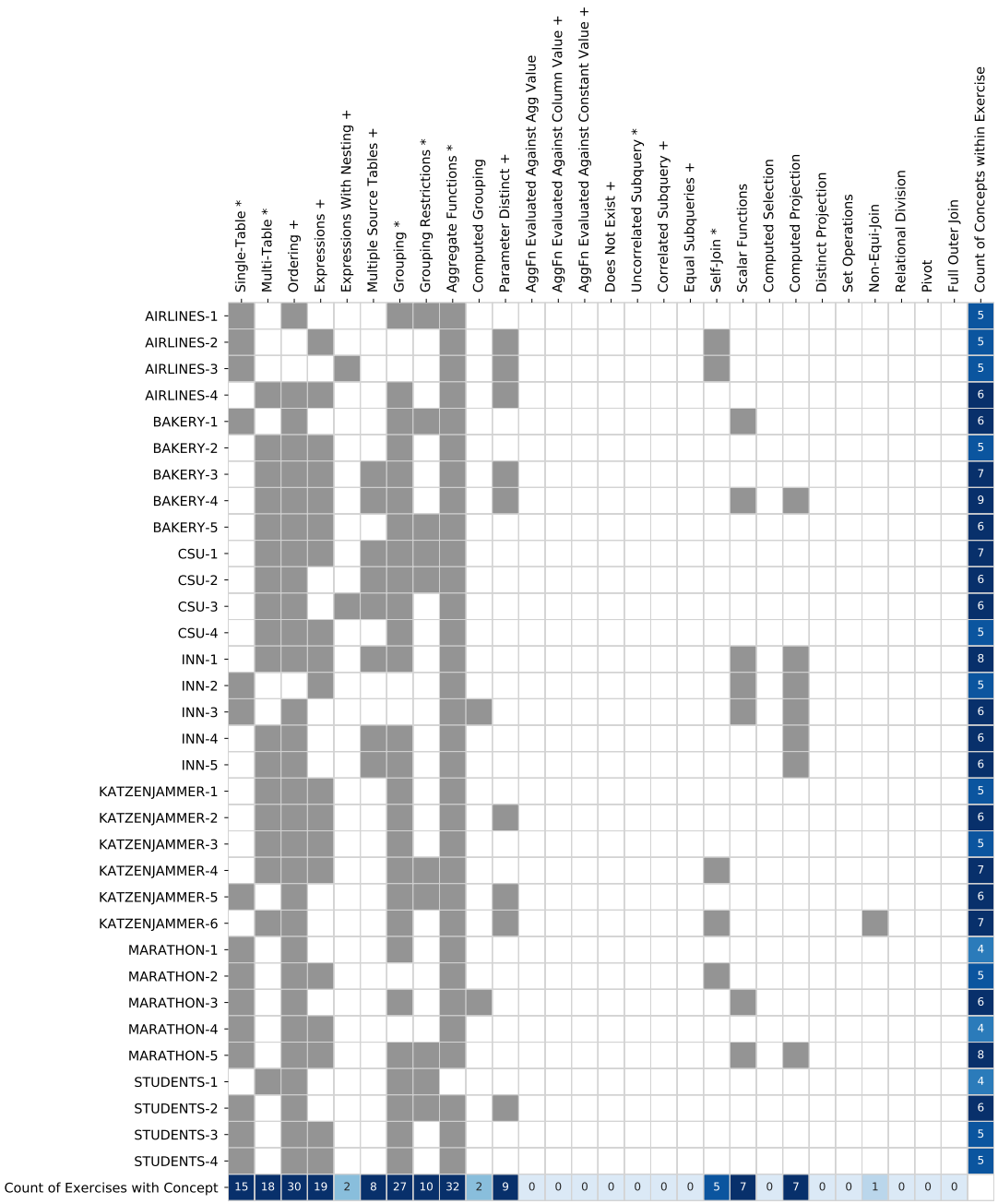

Figure .11: Lab B Exercises and Concepts

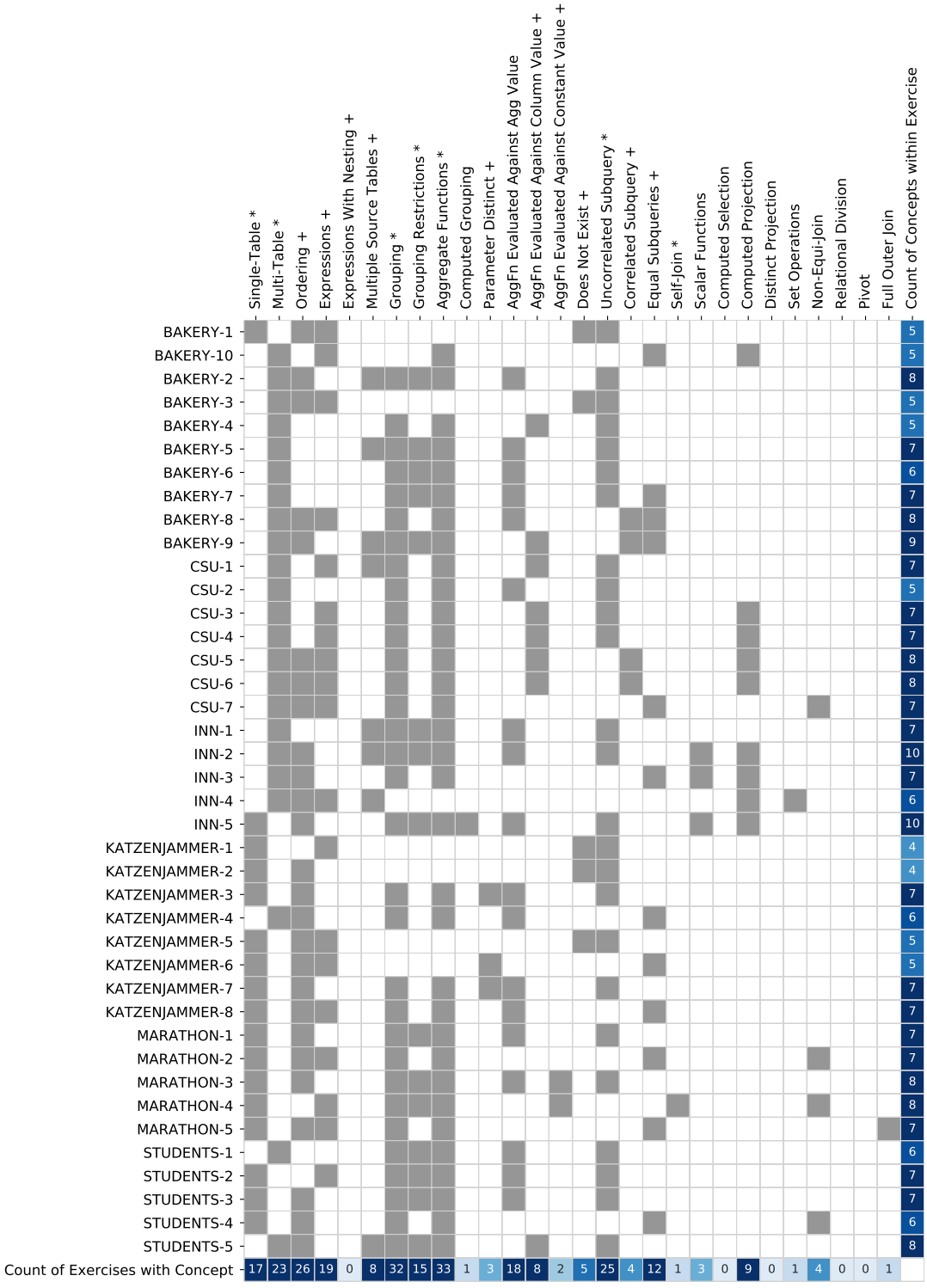

Figure .12: Lab C Exercises and Concepts