Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Do the errors produced by generative AI in formulating queries reflect students' misconceptions in learning SQL?

Abdolhamid Livani

Master Thesis

Università di Genova, DIBRIS Via Opera Pia, 13 16145 Genova, Italy
https://www.dibris.unige.it/

MSc Computer Science
Data Science and Engineering Curriculum

Do the errors produced by generative AI in formulating queries reflect students' misconceptions in learning SQL?

Abdolhamid Livani

Advisor: Giovanna Guerrini, Davide Ponzini

Examiner: Barbara Catania

December, 2024

Acknowledgements

I am profoundly grateful to my advisors, Professor Giovanna Guerrini and Davide Ponzini, for their invaluable guidance and support throughout this journey. I would also like to extend my sincere thanks to Professor Barbara Catania, whose thoughtful feedback helped elevate this work.

To my parents, whose steadfast support has made all of this possible and to my wife, whose patience and encouragement have been my anchor—thank you, from the bottom of my heart.

I am deeply appreciative of you all.

Abstract

The study compares the queries generated by ChatGPT and students to analyse the respective misconceptions in SQL query construction. An important contribution is the classification of common misconceptions as a basis for the study. Students' SQL queries as well as those generated by ChatGPT are analysed to find and classify misconceptions from both sources. The final step is a thorough investigation of these misconceptions to gain an understanding of the type and frequency of errors made by students compared to those generated by ChatGPT. The results will be used to guide educational plans aimed at improving SQL learning and to investigate the possible use of queries generated by artificial intelligence to support education.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The discipline of data education is thoroughly analysing the difficulties students have when learning to write SQL (Structured Query Language) queries [HH16] and [CDRFM18]. SQL is the standard language for manipulating relational databases, and its knowledge is required for students studying computer science and related subjects. Many students find SQL difficult, despite its significance, which leads to frequent misconceptions and many errors.

Recent studies [AFM24] and [PRH+24] have tackled these issues and identified specific concepts that students find challenging and frequent misconceptions. One of the key questions this thesis focuses on is whether these student issues match the flaws created by generative artificial intelligence (AI) systems when building SQL queries from natural language inputs. Generative AI, backed by advanced machine learning models, could automate processes such as SQL query formulation, yet these systems can also generate inaccurate queries.

This study seeks to test the premise that the errors caused by generative AI systems reflect the same misconceptions that students have when learning SQL. The research seeks connections and trends in errors made by students and artificial intelligence. Understanding these connections will help us to investigate the potential of generative AI as a tool for SQL training and get a deeper knowledge of the learning process.

1.1 Background

For students in computer science and related fields, knowledge of SQL is an indispensable ability since it is the main language used for relational database management and manipulation. Though it is important, many students find learning SQL to be somewhat difficult.

Among these difficulties we can list grasping the ideas of relational databases, understanding difficult query structures, and applying theoretical ideas to real-world problems. Many times, these challenges result in misconceptions and repeated mistakes in SQL queries by students.

As artificial intelligence and machine learning progress rapidly, generative AI systems have developed into potent tools capable of automating various activities, including generating SQL queries [Tia23]. These systems trained on big dataset algorithms might create SQL queries derived from natural language descriptions. Generative AI is not free from mistakes, even if it shows duty in supporting education and software development. In fact, generative AI models are taught on enormous volumes of web-based data, including queries written by and searches generated by programmers of all degrees of expertise. These models thereby mirror the prevalent habits, patterns, and even errors in actual SQL query authoring. This implies that, both right and wrong, the mistakes AI generates typically reflect the strategy and formulation of questions programmers use. Analysing the type and frequency of mistakes generated by generative AI systems can help one gain an important understanding of the underlying mechanisms and restrictions of these technologies.

Studies [APBL16] and [TSV18] have found both typical student SQL errors as well as the cognitive processes behind them. Studies show that pupils struggle with concepts such as joins, subqueries, and proper use of aggregate functions. Good instructional tools and tactics depend on an awareness of these mistakes.

Research on the comparability of errors committed by generative AI systems and those produced by students is very rare. Should the mistakes produced by generative AI mirror those of students, it would imply that, despite their increased capacity, generative AI systems, in some measure, reflect human learning processes[MWL24]. This similarity helps one to identify and manage student misunderstandings.

This work aims to carefully investigate the errors produced by generative AI in formulating SQL queries and compare them with student errors in order to identify patterns and relationships that could direct teaching efforts. The results could inspire the development of focused treatments and technologies meant to clear misconceptions, hence enhancing SQL learning results.

Located at the junction of computer science education, artificial intelligence, and educational technology, this study seeks to understand where students and generative AI misconceptions are related to each other.

1.2 Problem Statement

Students usually find it difficult to comprehend and appropriately apply SQL ideas in computer science coursework. These difficulties frequently lead to widespread misconceptions and recurring errors while drafting SQL queries. The emergence of generative AI systems that can construct SQL queries provides a new chance to check AI-generated queries for faults. It is still uncertain whether the mistakes made by generative AI reflect the same misconceptions students have when learning SQL.

This research aims to explore the similarities between AI-generated errors and student misconceptions, with the goal of using AI as a diagnostic tool to improve SQL education. By identifying and categorising the errors made by generative AI and comparing them to those made by students, this study seeks to gain insights about the areas that can be boosted to improve students faults.

1.3 Research Objectives

The research objectives of this thesis consist of three main goals:

- To contrast the usual misconceptions and blunders students run into while learning SQL with the mistakes made by generative AI in formulating queries. This aim is to investigate whether student misconceptions and mistakes produced by generative AI show any noteworthy correlation.
- to find and categorise the typical errors caused by generative AI systems creating SQL queries. This aim calls for a careful investigation to find a complete taxonomy of these mistakes.
- to evaluate the possibility of identifying student misconceptions in SQL by use of generative AI's errors as a tool. This objective seeks to assess how well AI-generated mistakes could draw attention to certain areas where students struggle, therefore offering more thorough understanding of their learning difficulties.

1.4 Research Questions

In the following, we will specify our research questions. These questions intend to study the specific types of mistakes generative AI systems occasionally generate in SQL query construction. Finding and categorising these mistakes will help us to ascertain whether they

represent typical SQL learning misconceptions. Examining mistakes made by generative AI alongside those often seen in student work in educational settings helps one to make this comparison.

1. **What kinds of faults may generative AI systems usually produce while creating SQL queries?**
2. **How do these errors connect to common student misconceptions in learning SQL?**

1.5 Scope

The study is centred on data education, with a particular emphasis on examining the errors generated by AI systems when crafting SQL queries and comparing these with common student misconceptions in learning SQL.

The errors made by one or more generative AI systems well-known for generating SQL queries will be examined in this work. Popular artificial intelligence models include Google Gemini, ChatGPT, Microsoft Copilot,Perplexity AI. We will use ChatGPT because it is popular among students, easy to use for everyone, always improving, and can do many things.

The study will gather and look at student common SQL query errors. One could find this material from academic institutions, online coding tools, past research papers, or even personal experience. In this thesis, we will focus on BSc students because they are at the beginning of the learning process. So we gathered queries from three resources that contain computer science database exams and labratories, computer engineering database labratories, and Miedema Thesis. For each resource, we will analyse queries and extract related queries for use in our study.

Errors from both generative AI systems and students will be categorised specifically into syntactic, semantic, logical, and complication ones. Subsequently, the thesis will analyse these groups by highlighting their differences in order to identify patterns and connections.

The data will inform the research in developing targeted educational interventions and tools to address identified errors and misconceptions. These treatments may require the use of technical tools, teaching strategies, and modifications to the course program.

The relevance of this research can thus be summarised as follows:

- This work provides a valuable evaluation of the challenges students face studying SQL, as well as the detection and analysis of typical blunders generated by AI.

- Analysing the similarities between artificial intelligence and human learning systems enables this research to support more general domains of AI and education.
- Teachers that recognise common student misconceptions and artificial intelligence-created error patterns will have a better awareness of their students' learning environments.
- The study's results can assist in developing customised learning routes for students.

1.6 Outline of the Document

This thesis is organized into six main chapters, each of them addressing a different aspect of this research. Chapter 2: Literature Review examines the existing body of knowledge relevant to the study, identifying gaps and establishing the theoretical foundation for the research. Chapter 3: Research Methodology briefly introduces the main steps of the process adopted to collect and analyze queries. Chapter 4: Our Study outlines the specific datasets (schemas, corresponding query specifications), how queries produced by students and generative AI are collected, and how issues and misconceptions are identified. Chapter 5: Results reports the findings derived from the study, including data analyses and other relevant outcomes. Chapter 6: Discussion interprets the results by discussing the research questions, highlighting implications and limitations of the study. Finally, Chapter 7: Conclusions synthesizes the key insights from the research, emphasizing its contributions to the field and proposing recommendations for further investigation.

Chapter 2

Literature Review

The incorporation of generative artificial intelligence (AI) into learning environments has brought new ideas for improving teaching and learning strategies. Teaching Structured Query Language (SQL), which is fundamental to database management and information retrieval, is a clear area where AI has shown promise. Using AI to teach SQL presents a unique set of difficulties, even if it seems promising. Examining past research can help us explore the nuances of learning SQL, the influence of AI in learning environments, and the types of errors and misconceptions that students typically encounter. In this chapter, we briefly review the related literature by first introducing the notion of misconception, then focusing on learning SQL, then on the use of generative AI in education, and finally on the proposals of using AI for assisting SQL learning.

2.1 Student Misconceptions

In the natural sciences, students that use overly simple explanations or everyday experiences to challenge scientific ideas create misconceptions. For example, human intuition guides their opinion even if scientific fact shows the opposite—that heavier objects fall faster than lighter ones[MM24]. Correcting these misconceptions calls for an active teaching strategy, one that uses inquiry-based learning—that is, where students investigate and challenge the content instead of just knowing it. Teachers should also be aware of these typical misconceptions and use focused techniques, including cognitive conflict (challenging a students current ideas), to lead them towards the proper knowledge. Marx et al. [MWL24] conducted an interview-based study to identify misconceptions among secondary school students about machine learning concepts. Ignoring these misconceptions might help pupils grow, which would challenge understanding of ever more complex scientific ideas [GRGDNT$^+$24].

A misconception in students learning to program refers to an incorrect understanding or belief about how programming concepts, syntax, or logic work. Misconceptions often arise when students attempt to form mental models about how programming operates but make errors in reasoning or understanding due to incomplete or incorrect knowledge [MAF22]. This concept ties closely to two ideas:

- *Incomplete mental models.* A mental model is an internal representation of how a system works, constructed by a learner based on their knowledge and experience. In programming, students build mental models to understand how code executes, how data flows, and how components of a program interact. When a mental model is incomplete, it lacks the necessary details to accurately represent how the system works. This can lead to incorrect reasoning, i.e., students make predictions about code behaviour that do not align with reality, and misinterpretations, i.e., assuming that a variable retains its value across different program runs without initialization. For instance, a student might believe that a for loop runs forever if there is no explicit condition, failing to understand that the loop iterates a fixed number of times or until a specified condition is met[Juh13].
- *The Notional Machine.* The notional machine refers to a simplified, conceptual understanding of how a programming language or environment executes code. It is a mental abstraction of the computers operation that helps students reason about their programs. Misconceptions arise when students either do not yet have a clear notion of the notional machine or have built a faulty notional machine due to misinterpretation or incomplete instruction. For instance, a student might think that when a function is called, the variables in the function scope are directly accessible outside of it, reflecting a misunderstanding of scope and how the notional machine manages memory [PM$^+$01].

Misconceptions often stem from gaps or inaccuracies in the students mental model of the notional machine. An incomplete mental model means the student is unable to predict how the notional machine will behave in various scenarios, leading to errors and misunderstandings. Addressing misconceptions involves helping students refine and expand their mental models to more closely align with the actual workings of the notional machine.

Misconceptions can create significant barriers to learning programming because they affect the students ability to debug, solve problems, and advance to more complex topics. Educators aim to identify and correct misconceptions early, using strategies like visualisation tools, clear explanations of the notional machine, and targeted exercises that challenge and refine students mental models.

2.2 SQL Learning and SQL Misconceptions

2.2.1 Overview of SQL Learning Difficulties

Learning SQL presents major difficulties for pupils because of its complexity and the twin need of knowing both syntax and semantics [GM15]. Ahadi et al. [APBL16] claim that students struggle with seven different kinds of SQL queries—from simple SELECT statements to more sophisticated JOIN procedures. In line with this, Miedema et al. [MAF22] conducted think-aloud research to highlight typical errors among beginners, including misreading the relational model and wrongly using aggregate functions.

Taipalus et al. [TSV18] examine SQL syntax and semantics; notice that even little errors may cause major misinterpretation. Their studies emphasise the important need for a strong fundamental knowledge basis in both avoiding and fixing these errors. In a same vein, Green and Petre [WS81] contrast procedural and non-procedural query languages. They underline that, being a non-procedural language, SQL requires a different cognitive approach, which might be especially difficult for students who know procedural programming languages [AS13].

2.2.2 Analysis of SQL Query Errors

Educators who want to build targeted educational tactics must effectively categorise mistakes. When it comes to SQL training, building successful lesson plans and improving learning outcomes is critically dependent on identifying and classifying student misconceptions. From syntactic errors to more advanced semantic and logical misinterpretations, misconceptions in SQL query development can take many forms. Establishing a strong framework for categorising these errors is essential to fully addressing these issues. Al-Shuaily and Renaud [ASR14] propose a foundational framework for identifying and categorising common student errors that is pattern based.

Ahadi et al. [APBL16] classify frequent SQL faults into seven distinct categories, providing a structured framework for identifying and correcting specific misconceptions. Building on this, Poulsen et al. [PBAH20] analyse student solutions to identify recurring error patterns that can be used to guide the development of educational materials. Additionally, [PMHS21] highlights approaches to identify SQL errors, focusing on providing repair strategies.

Taipalus investigates the causes of SQL query formulation problems in his work [Tai20], so illuminating the manner in which individuals believe to contribute to these blunders. Cagliero et al. [CDRFM18] underline how important it is to provide specific feedback based on a thorough assessment of errors in order to address these issues and support a

data-driven approach to learning SQL.

Understanding the thinking aspects of learning SQL is key to figuring out why students make mistakes so often [DLN+06]. Green and Petre [WS81] look at the human factors involved with procedural and non-procedural query languages. They say that because SQL is formal, it needs a different way of thinking. This difference is especially hard for students who are used to procedural computer languages.

Biswas et al. [BKS14] stress how important cognitive and metacognitive task models are for judging how people learn. These models give us a way to think about how students think about SQL queries and where they might go wrong.

Examining the causes of errors in SQL query writing is the focus of both [TSV18] and [Tai20]. Many of these errors, they discover, result from ignorance of the fundamentals and inadequate grasp of SQL syntax and semantics.

Mitrovic [Mit03] and Hussan [HH16] highlight the significance of metacognitive methods in SQL education, along with reflective learning practices in addressing misconceptions. Cagliero et al. [CDRFM18] propose using a data-driven approach to classify and fix SQL problems. They emphasise the importance of providing specific feedback that is based on a comprehensive examination of the mistakes. By spotting these mistakes, teachers may create more successful lesson plans and resources meant to target certain student misunderstandings.

[Tra24] article aims to find and examine typical SQL misconceptions among students. The research classifies SQL query formulation mistakes, including inappropriate usage of SQL functions, erroneous joins, and extra columns in SELECT queries. The study seeks to find trends in these errors to better grasp students difficulties with SQL and create focused instructional plans for improving SQL learning results.

2.3 Generative AI in Education

Generative AI is becoming a powerful tool in schools because it allows for personalised learning and quick feedback from students. Corbett and Anderson [CRA91] show that artificial intelligence teachers have a lot of potential. For example, the CMU LISP tutor has helped students learn a lot more by giving them feedback that is tailored to their specific needs. To examining student learning behaviours, [BKS14] and [KPÇ24] explore the function of cognitive and metacognitive task models, therefore demonstrating how AI may be tailored to suit a broad spectrum of educational needs [SY23].

Arroyo et al. [AOP+04] investigate how artificial intelligence may encourage deeper knowledge by forcing students to express their mental processes, hence augmenting the usefulness

of instructional discussion systems geared for self-explanation.

Ottenbreit-Leftwich et al. [OLGJ+23] looked at lessons learnt by adding artificial intelligence into education with main students and teachers. Their findings highlight the potential of AI tools to support both teaching practices and student engagement, emphasizing the importance of contextual adaptability in designing AI-driven educational interventions.

Researchers discovered [AB23] that Large Language Models (LLMs) have the ability to assist instructors by generating content, simulated data, and queries utilising aggregate functions. This feature not only saves a lot of time for exam preparation but also provides numerous fascinating and appealing visuals.

Nevertheless, some teachers are reluctant to use LLMs in the classroom, largely because of doubts over the accuracy and dependability of these instruments. [KAHM97] present proof of their efficacy in several educational environments, therefore supporting the success of AI tutoring systems in major applications.

2.4 Using AI in SQL Learning

Fletcher et al. [PRH+24] investigated if it would be feasible to introduce learning models (LLMs) for database systems. Their research focused on the tailored instruction and expected improvements in on-demand feedback these models may provide. According to their research, LLMs could be really useful in the teaching of difficult SQL ideas by providing careful explanations and customised assistance. Although this method needs to be improved progressively to meet the complex specifics of SQL query development, it can help students grasp and engage with the information.

[KRM+24] investigates the use of LLMs versus traditional web search methods in SQL learning among students; it shows that students using instructor-tuned LLMs required more interactions but achieved similar quality in SQL query output. Moreover, students in the LLM condition reported lower mental demand, which suggests that the cognitive load can be reduced by LLMs while learning SQL. The authors further state that LLMs could support traditional methods within programming courses with structured and contextual support.

[NGF+24] studies the performance of the LLM-based Text-to-SQL model on industrial-scale complex databases and thus challenges the current LLM Text-to-SQL state-of-the-art models, which tend not to perform well with large schemas and multiple relationships. However, the models performed significantly worse on tests conducted on Mondial and an industrial proprietary database compared to those conducted on benchmark settings. This indicates that perhaps current LLMs are limited in their operability regarding databases with large tables and foreign keys in the industrial world. This therefore presents a disparity

between controlled environments and real-life application in the capabilities of LLM, thus proposing an area that needs enhancement in AI SQL-driven tools

Mitrovic [Mit03] offers a sophisticated SQL trainer available online that helps pupils by mixing adaptive feedback with mistake detection. This web-based instructor has satisfactorily solved common SQL errors and misconceptions. Mitrovic's earlier work [Mit98] introduced the concept of using tools for SQL learning and steps of developing. Moreover, underscoring the advantages of reflective learning methods enabled by artificial intelligence, Hussaan and Hassan [HH16] investigate the use of metacognitive techniques in SQL query instruction with a teachable agent.

Ohlsson [Ohl94] further on this by talking about constraint-based student modelling, which is meant to fit learning requirements and solve particular misconceptions as they develop. This flexible approach works especially well in offering focused help where pupils most need it.

Analysing common SQL query difficulties, Fletcher et al. [PRH+24] underlined the possibility of LLMs to be taught in seeing and fixing these problems. Their studies showed that LLMs can identify patterns in student mistakes and provide insightful analysis—qualities needed to further understanding of SQL syntax and semantics. This method could satisfy the objectives of constraint-based student modelling and increase the efficiency of AI-powered teaching assistants [Ohl94].

Arroyo et al. [AOP+04] illustrate how encouraging students to explain themselves and consider what they are thinking can help to increase cognitive understanding by use of tutorial conversation systems. This clarifies any uncertainty and enables pupils to describe their mental processes.

While the questions produced were relevant and logical, [AFM24] analysis of the usage of LLMs to produce SQL workouts revealed that sometimes there were confusing or erroneous aspects. This underlines the need for teachers to carefully review and confirm AI-generated data to guarantee that typical student misconceptions are suitably handled and categorised.

[KAHM97] presents strong evidence supporting the usefulness of AI tutoring systems in large-scale deployments, thus affirming the important significance of AI in educational contexts.

Emphasising their possibilities to improve SQL learning [AOP+04] and [BKS14] investigate the benefits of tutorial conversation systems and cognitive/metacognitive task models. Moreover, Hussaan and Hassan [HH16] look at how metacognitive approaches might be used in SQL query instruction, therefore proving the value of reflective learning methods supported by artificial intelligence.

In [BBPP+24] the authors targeted remote or distant learning, such as during the COVID-19 pandemic. The authors provide a chatbot-based learning environment designed to assist

students in acquiring knowledge in SQL. The platform was used by computer engineering undergraduates to engage in SQL search exercises with the assistance of a chatbot for a relevant case study. [AFM24] performed a feasibility study on the utilisation of ChatGPT-3.5 for the creation of SQL exercises.

[DZG+23] presents the C3-methodology that, for the first time, leverages ChatGPT in a zero-shot setting for the Text-to-SQL task and shows that it is able to achieve an execution accuracy of 82.3 percent on the Spider dataset, significantly better than fine-tuning-based approaches. The C3 approach introduces clear prompting, calibration with hints, and consistent output to get over commonly observed limitations in the LLM-based text-to-SQL conversion task. These results prove that zero-shot methods like C3 can be efficient and cost-effective in generating SQL queries without requiring a tremendous deal of training data.

In [AFM24] examines the extent of the ability of ChatGPT-3.5 in generating SQL exercises along with respective database schemas and question sets that could potentially hasten curriculum development and goes on to state that indeed ChatGPT is able to generate relevant and organised SQL exercises, though some educators being contacted have reservations regarding its consistency in advanced SQL concepts. It means that with LLMs, educators could save a lot of preparation time, but they would need further refinement to fit the educational standards.

[CFFM24] investigates ChatGPT as an automatic grader for SQL exercises. It finds that it reaches the same level of grading accuracy as human instructors on benchmark data. ChatGPT was able to detect syntactic and semantic errors with fairly detailed feedback, thus giving scores that are comparable to human grading. This points to the potential use of LLMs as efficient, scalable grading assistants in large database courses, though the study mentions a decline in the models reliability when dealing with more complex SQL queries.

The results of the study clearly indicate that students receive contextualised and personalised guidance from ChatGPT in a manner that clarifies programming concepts and improves code accuracy. Indeed, the results show that ChatGPT helps students in basic learning and initial code correction, but structured prompts with close relevance to course objectives given by [MHLD24] work best for students.

2.5 Conclusions and Reference Categorisation

From the analysis of the literature, the lack of an analysis of the issues in AI-generated SQL queries emerged, as well as a comparison of the common problems in AI-generated queries with the typical student misconceptions. In this chapter, and in Section 2.2.2

specifically, we discussed different types of SQL misconceptions, methods for categorising them, and identified some common errors that students made in their queries. For developing our analysis, we need to rely on a suitable categorisation of misconceptions about SQL query formulation. We seek to find the proper framework covering all kinds of mistakes—syntactic, semantic, logical, and so on—by analysing the specific categorisations shown in various important studies.

We will go over the classification techniques from various research, each offering special analysis of the type of student mistakes. This thorough review is crucial to guarantee that the selected framework not only covers the wide spectrum of student mistakes but also offers pragmatic value for teachers and area of SQL education researchers.

The article [MAF22] authors mostly focus on categorising syntactic errors often generated by pupils. Among these errors are typographic ones, missing semicolons, incorrect keywords, and bad wording of sentences. In the study, [HW20] looks into student patterns in SQL queries and mistake rates coming from these searches. It gathers mistakes arising from incorrect SQL usage, including those brought about by JOIN operations or CASE expressions.

The paper [RB13] defines mistakes resulting from common misconceptions, including misinterpretation of JOIN operations and GROUP BY clauses. The main emphasis is on identifying logical and semantic problems that occur as a result of these misconceptions. The paper [BG06] provides a thorough classification covering logical inconsistencies, erroneous SQL function application, and misreading of SQL semantics.

Syntactic faults, including typos, wrong keywords, and clause ordering errors, are classified in the paper [ABV+16]. It also investigates how one may forecast student performance using these mistake trends.

The work of Taipalus et al. [TSV18] is chosen for its exact and comprehensive categorisation of SQL misconceptions 2.1.

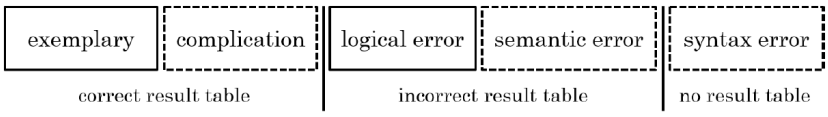

Figure 2.1: Semantic Errors and Error Categories [TSV18]

This framework not only provides insightful analysis for teachers and researchers to improve SQL education and appropriately manage student misconceptions but also fully covers syntactic, semantic, and logical errors. This makes it the most appropriate option for our investigation on whether students misconceptions in learning SQL influence the mistakes generated by generative artificial intelligence in query formulation.

Chapter 3

Research Methodology

This chapter describes the methodology adopted for our analysis. The steps in this chapter show a structured way to gather and sort data, focusing on misconceptions about how students learn and how generative AI can answer questions. The study technique consists of several essential stages, each of which is specifically designed to ensure a comprehensive and accurate examination. Our methodology figure 3.1 consists of the following steps:

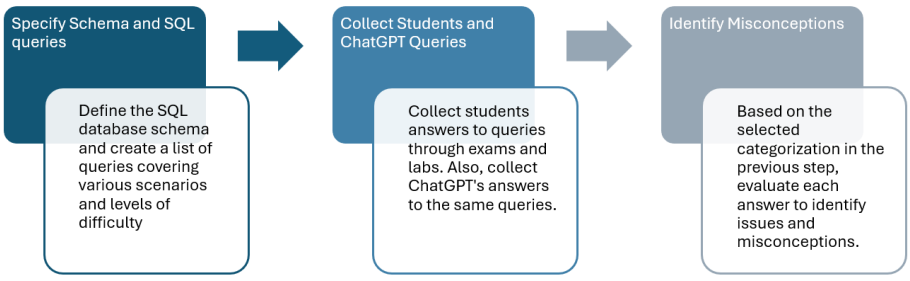

Figure 3.1: Our Study Methodology

1. Specify Schema and SQL Queries
    - Clearly state the schema. Tables, relationships, and constraints ought to form the structure of the SQL database. This is a stage of the process of creating the database we want to retrieve data from.
    - Make a list of questions you wish to probe. Organise SQL searches covering many different case scenarios and degrees of difficulty.
2. Collect Students and ChatGPT Queries

- Data Collection: Gather SQL queries from students during classroom exercises, assignments, or exams.
- Identify Misconceptions: Analyse these queries to find common errors and misconceptions.
- Use ChatGPT to Generate Questions: Assign the same tasks given to students to ChatGPT and record the SQL queries it produces.
- Identify Errors: Conduct a thorough review of these queries to identify any inaccuracies or ambiguous sections.

3. Identify Misconceptions

- Establish Criteria for Error Detection: Formulate a set of guidelines for identifying issues with SQL queries, such as semantic errors, logical errors, and syntax errors.
- Use these guidelines to systematically evaluate queries from both students and ChatGPT, identifying any issues.

Chapter 4

Our Study

This chapter discusses the specific sets of queries analysed in our study. Specifically, we first discuss the reference databases, corresponding schemas, and queries on such data, and then we discuss how we gather student queries on such data.

4.1 Datasets: SQL Schemas and Queries

This section will go over four distinct databases used to show different real-world situations. Each schema has been carefully selected for its usefulness, complexity, and the variety of SQL concepts it allows. The schemas provide a basis for performing searches that include a wide range of SQL features, from basic select queries to more complex operations such as joins, subqueries, and aggregate functions.

This section aims to provide a comprehensive explanation of each schema, including the key columns, table structure, and relationships. We will examine the rationale behind the selection of each schema, including its applicability in real-world contexts and its systematic facilitation of SQL learning.

4.1.1 Schema 1: Database Exam

In this section, we present the SQL schema used in the context of a database course exam. The schema consists of five main tables: `GIOCATORE`, `TORNEO`, `CATEGORIA`, `REGISTRAZIONE`, and `GIOCAIN`. These tables are designed to represent key entities in a sports tournament management system and facilitate complex queries related to player participation, tournament details, and performance analysis.

Table Overview

`GIOCATORE`: The `GIOCATORE` table stores information about players participating in tournaments. It contains the following fields:

- `IdG (SERIAL PRIMARY KEY)`: A unique identifier for each player.
- `Nome (VARCHAR(100))`: The first name of the player.
- `Cognome (VARCHAR(100))`: The last name of the player.
- `Genere (CHAR(1))`: The gender of the player, represented by a single character (`'M'` or `'F'`).
- `DataN (DATE)`: The date of birth of the player.
- `Nazione (VARCHAR(100))`: The country of origin of the player.

`TORNEO`: The `TORNEO` table holds the details of various tournaments. Its fields are:

- `IdT (SERIAL PRIMARY KEY)`: A unique identifier for each tournament.
- `NomeT (VARCHAR(100))`: The name of the tournament.
- `Luogo (VARCHAR(100))`: The location where the tournament takes place.
- `DataI (DATE)`: The start date of the tournament.
- `DataF (DATE)`: The end date of the tournament.
- `NumTurni (INT)`: The number of rounds or turns in the tournament.
- `Tipo (VARCHAR(50))`: The type of tournament (e.g., singles, doubles).
- `Terreno (VARCHAR(50))`: The type of playing surface (e.g., grass, clay).

`CATEGORIA`: The `CATEGORIA` table categorizes tournaments based on gender or other factors:

- `IdCat (SERIAL PRIMARY KEY)`: A unique identifier for each category.
- `NomeCategoria (VARCHAR(100))`: The name of the category (e.g., singles, doubles).
- `GenereCategoria (CHAR(1))` : The gender category (`'M'` or `'F'`).

`REGISTRAZIONE`: The `REGISTRAZIONE` table records tournament registrations. It includes:

- `IdT (INT)`: A reference to the tournament (foreign key to `TORNEO`).
- `IdCat (INT)`: A reference to the category (foreign key to `CATEGORIA`).
- `NumRegistrazione (SERIAL PRIMARY KEY)`: A unique registration number.
- `DataRegistrazione (DATE)`: The date of registration.
- `TestaDiSerie (BOOLEAN)`: Indicates whether the participant is a seeded player.

`GIOCAIN`: The `GIOCAIN` table tracks player participation in tournaments:

- `IdT (INT)`: A reference to the tournament (foreign key to `TORNEO`).
- `IdCat (INT)`: A reference to the category (foreign key to `CATEGORIA`).
- `NumRegistrazione (INT)`: A reference to the registration (foreign key to REGISTRAZIONE).
- `IdG (INT)`: A reference to the player (foreign key to `GIOCATORE`).

Queries

1. Exam 1a: Players who participated in both the singles and doubles categories in the same tournament
2. Exam 2a: The French players who have never been `TestaDiSerie` (Boolean attribute `TestaDiSerie`)
3. Exam 1b: Italian players who have participated in both the `US Open` and the `Australian Open` in the same year
4. Exam 2b: German players who have never participated in `Wimbledon`
5. Exam 1c: The tournament in which players from the greatest number of different nations participated
6. Exam 2c: The player who has played the most different tournaments
7. Exam 1d: The tournament with the highest number `teste di serie`

8. Exam 2d: The tournament with the highest average age of players (also considered correct current age)

Reasons for Schema Selection

With its five linked tables, `GIOCATORE` schema is perfect for analysing SQL learning and misconceptions because of its complexity, practical relevance, and many query kinds.

- Understanding data integrity and relational dynamics in SQL depends on many-to-many connections and foreign key restrictions; hence the schema includes both.
- The chosen searches cover many SQL ideas, from simple filtering to sophisticated aggregation and sub-queries. This range shows how students and ChatGPT approach several SQL functions and typical error-prone situations.
- The questions span basic to complicated, enabling study of SQL learning process and identification of places where misconceptions usually surface.

4.1.2 Schema 2: Computer Science Database Laboratories

In the context of a computer science database laboratory, this part offers the SQL schema intended for querying and analysis. Six tables total make up the model: `Professori`, `CorsiDiLaurea`, `Corsi`, `Studenti`, `Esami`, and `PianiDiStudio`. These tables enable sophisticated searches concerning student enrollment, course administration, and test results by capturing vital information for handling academic records, student information, and course-related facts.

Table Overview

`Professori`: The `Professori` table stores information about professors within the academic institution.

- `Id (DECIMAL(5,0) PRIMARY KEY)`: A unique identifier for each professor.
- `Cognome (VARCHAR(30))`: Last name of the professor.
- `Nome (VARCHAR(30))`: First name of the professor.
- `Stipendio (DECIMAL(8,2) DEFAULT 15000)`: Salary of the professor, with a minimum value of zero.

`CorsiDiLaurea`: The `CorsiDiLaurea` table holds details of the degree programs offered.

- `Id (DECIMAL(3,0) PRIMARY KEY)`: A unique identifier for each degree program.
- `Facolta (VARCHAR(50))`: The faculty offering the program.
- `Denominazione (VARCHAR(50))`: Name of the degree program.
- `Attivazione (CHAR(9))`: The year the program was activated.
- `Unique Constraint`: Ensures unique records by faculty and course name.

`Corsi`: The `Corsi` table represents individual courses offered under each degree program.

- `Id (CHAR(10) PRIMARY KEY)`: A unique identifier for each course.
- `CorsoDiLaurea (DECIMAL(3))`: A reference to the degree program (foreign key to `CorsiDiLaurea`).
- `Denominazione (VARCHAR(50))`: Name of the course.
- `Professore (DECIMAL(5,0))`: A reference to the course instructor (foreign key to `Professori`).
- `Attivato (BOOLEAN DEFAULT FALSE)`: Indicates whether the course is currently active.

`Studenti`: The `Studenti` table maintains records of students.

- `Matricola (VARCHAR(10) PRIMARY KEY)`: A unique identifier for each student.
- `Cognome (VARCHAR(30))`: Last name of the student.
- `Nome (VARCHAR(30))`: First name of the student.
- `Residenza (VARCHAR(30))`: Place of residence.
- `DataNascita (DATE)`: Birth date of the student.
- `LuogoNascita (VARCHAR(30))`: Birthplace of the student.
- `CorsoDiLaurea (DECIMAL(3,0))`: A reference to the degree program (foreign key to `CorsiDiLaurea`).
- `Iscrizione (INTEGER)`: Year of enrollment.
- `Relatore (DECIMAL(5,0))`: A reference to the advisor (foreign key to `Professori`).

- `Laurea (DATE)`: Date of graduation (if applicable).

`Esami`: The `Esami` table records exams taken by students.

- `Studente (VARCHAR(10))`: A reference to the student (foreign key to `Studenti`).
- `Corso (CHAR(10))`: A reference to the course (foreign key to `Corsi`).
- `Data (DATE)`: Date on which the exam was taken.
- `Voto (DECIMAL(2,0))`: The grade received by the student, constrained between `1` and `33`.

`PianiDiStudio`: The `PianiDiStudio` table represents the study plans submitted by students.

- `Studente (VARCHAR(10))`: A reference to the student (foreign key to `Studenti`).
- `AnnoAccademico (INTEGER)`: The academic year of the study plan.
- `Anno (DECIMAL(1,0))`: Year of study, constrained to values between `1` and `6`.

Queries

1. Lab CS 1: List the student ID and names of students enrolled before the academic year `2007/2008` who are not yet in their thesis phase (i.e., have not been assigned an advisor).
2. Lab CS 2: List degree courses activated before `2006/2007` and after `2009/2010`, in alphabetical order by faculty and course name.
3. Lab CS 3: List the student ID and names, in reverse ID order, of students whose last name is not `'Serra,''Melogno,'or 'Giunchi,'or who reside in Genova, La Spezia, or Savona`.
4. Lab CS 4: List the student ID of those who graduated in computer science before November `2009`.
5. Lab CS 5: Return an alphabetical list of student names with their advisor's last name.
6. Lab CS 6: List students in the thesis phase who submitted their fifth-year study plan for computer science in 2011/2012, without duplicates and in reverse alphabetical order.

7. Lab CS 7: List the last name, first name, and status (`'studente'or 'professore'`) of students and professors.

8. Lab CS 8: List computer science students who passed`Databases 1` but not `Graphic Interfaces` in June 2010.

9. Lab CS 9: List computer science students who passed both `Databases 1` and `Graphic Interfaces` in June 2010.

Reasons for Schema Selection

The Computer Science Database Laboratories schema was chosen to provide a structured and realistic learning environment ideal for SQL practice in an academic laboratory setting.

- Controlled Laboratory Environment: Under this controlled laboratory environment, students may investigate SQL in an environment where they have ample time to try many strategies, experiment, and get a better knowledge of challenging searches.
- Diversity of Query Types and Complexity: Covering fundamental SQL abilities like filtering, sorting, grouping, and time-based restrictions, the searches span basic to sophisticated. This kind is suitable for laboratory exercises as it lets one experience a slow learning process.
- Realistic Academic Relationships: Typical university database connections—one-to-many and many-to-many—offer students real-world academic data structures within the paradigm, therefore providing practical experience.
- Support for Error Exploration and Incremental Learning: The controlled environment enables students to learn from errors even if questions challenge them to think critically and improve their SQL thinking. This structure promotes slow assessment of SQL proficiency and skill growth.

4.1.3 Schema 3: Engineering Database Laboratories

This section presents the Engineering Database Laboratory schema, which is structured to simulate an e-commerce or supply chain scenario. The model consists of five tables: `Categoria`, `Prodotto`, `Cliente`, `Ordine`, and `DettaglioOrdine`. These tables represent core entities in an ordering and inventory system, allowing for complex queries related to product categories, customer orders, and order details.

Table Overview

`Categoria`: The `Categoria` table categorizes products.

- `idCat (SERIAL PRIMARY KEY)`: Unique identifier for each category.
- `nome (VARCHAR(255))`: Name of the product category.

`Prodotto`: The `Prodotto` table stores information about each product.

- `idProd (SERIAL PRIMARY KEY)`: Unique identifier for each product.
- `nome (VARCHAR(255))`: Name of the product.
- `fornitore (VARCHAR(255))`: Supplier of the product.
- `idCat (INT)`: Foreign key linking to `Categoria` to categorize the product.
- `prezzo (DECIMAL(10, 2))`: Price of the product.

`Cliente`: The `Cliente` table holds customer information.

- `idClient (SERIAL PRIMARY KEY)`: Unique identifier for each customer.
- `nome (VARCHAR(255))`: Customer's name.
- `indirizzo (VARCHAR(255))`: Customer's address.
- `città (VARCHAR(255))`: City of residence.
- `nazione (VARCHAR(255))`: Country of residence.

`Ordine`: The `Ordine` table records each order placed by a customer.

- `idOrd (SERIAL PRIMARY KEY)`: Unique identifier for each order.
- `idClient (INT)`: Foreign key linking to the `Cliente` table.
- `data (DATE)`: Date when the order was placed.

`DettaglioOrdine`: The `DettaglioOrdine` table records details of each product within an order.

- `idOrd (INT)`: Foreign key linking to `Ordine`.
- `idProd (INT)`: Foreign key linking to `Prodotto`.
- `quantità (INT)`: Quantity of the product in the order.

Queries

1. Lab CE 1: Select the names of the products in the beverages category (`bevande`) that were not ordered in 2023.
2. Lab CE 2: Select the total amount of the last order made by the customer having id `"1234"`.
3. Lab CE 3: Select the names of the products that have never been ordered by customers residing in `Venezia` or `Brescia`.

Reasons for Schema Selection

The Engineering Database Laboratories schema was chosen because it offers a practical, hands-on approach to learning SQL in a context familiar to engineering and business scenarios, such as inventory and order management.

- Computer engineering student: All our previous data was from computer science. Considering Computer Engineering students helps us to generalise student behaviour by focusing on a different set of students and on a different course and instructor.
- Perfect fit for laboratory learning: Without the pressure of a real work environment, students have time to experiment with several query approaches, troubleshoot errors, and test difficult SQL concepts in a lab.
- Examining the Many-to-Many relationships concept: Essential in database administration, the `Ordine` and `Prodotto` tables are connected by `DettaglioOrdine`. Important abilities in managing linked data, connecting related tables, and handling foreign keys, are let students practice under this framework.

4.1.4 Schema 4: Miedema Thesis Database

This part presents the Miedema Thesis Database, a created in a past PhD thesis with a research topic closely related to our study[MAF22]. Six tables total make up this schema: `customer`, `store`, `product`, `shopping list`, `transaction`, inventory. These tables provide a complete basis for running many SQL searches concerning consumer and shop interactions as they reflect basic entities for customer purchases, store management, product inventories, and transactional data.

Table Overview

`customer`: The `customer` table stores information about customers.

- `cID (DECIMAL(5,0) PRIMARY KEY)`: Unique identifier for each customer.
- `cName (VARCHAR(255))`: Name of the customer.
- `street (VARCHAR(255))`: Street address of the customer.
- `city (VARCHAR(255))`: City of residence.

`store`: The `store` table holds details of different stores.

- `sID (DECIMAL(5,0) PRIMARY KEY)`: Unique identifier for each store.
- `sName (VARCHAR(255))`: Name of the store.
- `street (VARCHAR(255))`: Street where the store is located.
- `city (VARCHAR(255))`: City where the store is located.

`product`: The `product` table lists each product available for purchase.

- `pID (DECIMAL(5,0) PRIMARY KEY)`: Unique identifier for each product.
- `pName (VARCHAR(255))`: Name of the product.
- `suffix (VARCHAR(255))`: Additional description or variation of the product.

`shoppinglist`: The `shoppinglist` table represents items in a customer's shopping list.

- `cID (DECIMAL(5,0))`: Foreign key referencing `customer`.
- `pID (DECIMAL(5,0))`: Foreign key referencing `product`.
- `quantity (DECIMAL(10,0))`: Quantity of the product on the shopping list.
- `date (DATE)`: Date the item was added to the list.

`transaction`: The `transaction` table records each purchase transaction.

- `tID (DECIMAL(5,0))`: Unique identifier for each transaction.
- `cID (DECIMAL(5,0))`: Foreign key referencing `customer`.
- `sID (DECIMAL(5,0))`: Foreign key referencing `store`.

- `pID (DECIMAL(5,0))`: Foreign key referencing `product`.
- `date (DATE)`: Date of the transaction.
- `quantity (DECIMAL(10,0))`: Quantity of the product purchased.
- `price (DECIMAL(10,2))`: Price of the product in the transaction.

`inventory`: The `inventory` table tracks each store's product inventory.

- `sID (DECIMAL(5,0))`: Foreign key referencing `store`.
- `pID (DECIMAL(5,0))`: Foreign key referencing `product`.
- `date (DATE)`: Date of inventory record.
- `quantity (DECIMAL(10))`: Quantity of the product in stock.
- `unit price (DECIMAL(10,2))`: Price per unit of the product.

Queries

1. Miedema Thesis 1: List all IDs and names of customers living in `Eindhoven`.
2. Miedema Thesis 2: List all pairs of customer IDs who live on a street with the same name but in a different city.
3. Miedema Thesis 3: List all customer IDs, dates, and quantities of transactions containing products named `Apples`.
4. Miedema Thesis 4: Find the names of all inventory items that have a higher unit price than `Bananas`.
5. Miedema Thesis 5: Return a list of the number of stores per city.
6. Miedema Thesis 6: Return the stores table ordered alphabetically by city.
7. Miedema Thesis 7: A store-chain consists of at least two stores with the same name but different IDs. Find the names of the store-chains that, on average, sell products in quantities of more than 4.

Reasons for Schema Selection

The Miedema Thesis schema was chosen as it strongly relates with the academic research objectives of investigating SQL query misconceptions. Based on a prior PhD thesis [MAF22], this model provides a reasonable basis for looking at SQL concepts within a customer and inventory control situation.

- Originally intended for an academic thesis similar to this one, this model offers a tested framework for analyzing SQL learning difficulties in a disciplined, research-oriented environment.
- The design provides just a range of SQL queries—simple retrieval to advanced filtering, conditional logic, and aggregation. This allows a comprehensive assessment of SQL ability, therefore highlighting areas where both ChatGPT and students might encounter difficult SQL operations.

In Table 4.1 we present an overview of the specifications of each schema and the corresponding dataset.

	Database Exam	Computer Science Database Laboratories	Engineering Database Laboratories	Miedema Thesis Database
Student Degree	Computer Science	Computer Science	Computer Engineering	Computer Science
Number of Queries	8	9	3	7
Number of Subjects (Students/Groups)	25	23	12	21
Language	Italian	Italian	Italian	English
Data Collection Method	Paper	online submission	Paper	online submission
Answer Time	Limited	Not Limited	Limited	Not Limited
Sample Data Provided	No	Yes	Yes	Yes
Group	No	Yes	Yes	No*
Possibility to Run the Queries	No	Yes	No	Yes
Expected Results Available	No	Yes	No	No

* The queries were formulated in an unsupervised context, so we cannot be sure that students work alone.

Table 4.1: Dataset Overview

4.1.5 Query Complexity

We classify queries into three levels: simple, medium, and hard, in order to enable later analysis depending on the complexity of the query. Simple queries are those that involve simple filtering and a few joins. For example, a query such as finding the French players who have never been in the *TestaDiSerie* is straightforward, as it consists mainly of filtering by nationality and checking a Boolean attribute without complex aggregations or joins. Similarly, identifying German players who have never played Wimbledon, even if slightly more complex, is still simple, involving only a nationality filter and an exclusion condition on a particular tournament. Finally, identifying players who have played singles and doubles in the same tournament mostly involves grouping and filtering across categories, with no complex logic other than simple grouping.

The medium-difficulty queries add a degree of complexity introduced by aggregation or conditional logic. For example, finding the tournament with the highest number of *TestaDiSerie* requires aggregating and counting records based on a Boolean attribute across tournaments, which is one level of complexity. Finding the Italian players who participated in both the US Open and the Australian Open in the same year requires the creation of players who participated in more than one tournament within a given time period, making the task even more difficult. Secondly, to find the player who has played the most different tournaments, you would have to count the different tournament participations for each player, which involves an aggregation function and is therefore moderately complex.

These hard queries use advanced features of SQL, including complicated joins, nested conditions, and multi-table aggregations. Identifying the tournament with the most participating nations involves multiple joins to count different nations for each tournament, plus the added complexity of identifying the tournament with the highest number of unique nationalities. This clearly requires extensive use of joins and grouping. Another challenging query is for the tournament with the highest average age of players, derived from each players date of birth, as the nature of the date calculation question is to provide current ages for players, hence aggregation per tournament. The combination of database calculation, aggregation, and grouping makes this one of the most complex queries in the set. It also allows the queries to be categorised into a particular structure, from which each query is analysed in terms of the associated challenges posed by the complexity of the different types of SQL. (See Appendix A for the full complexity categories of the queries).

4.2 Collecting Produced Queries

4.2.1 Collecting Queries Produced by Students

To gather queries produced by students, we collect SQL queries from two sources. First, we find and extract SQL queries recorded in [MAF22] related thesis. This involves a deep analysis of the document, finding out the relevant parts where specific student SQL queries are described, methodically extracting these queries, and structuring them (in either a database or spreadsheet). We then ask our current computer science database students to formulate the queries via an AulaWeb assignment (optional activity, with no incentive but feedback from instructors/tutors).

Based on the schemas and queries described in Section 4.1, we developed or adapted SQL queries that ranged in complexity from basic statements to more sophisticated actions. These queries will be given to students via tests, assignments, or classroom activities. Each answer is tagged with relevant information, including the student ID and task ID, so that the results are collected, ordered, and stored in a disciplined manner.

In Table 4.2 we present a sample of the answers collected for this study. The full dataset contains approximately 600 answers collected from students from a variety of sources, including exams, lab sessions, and previously published queries from Miedema's thesis.

Reference	Written by	Query Specification	Student Answer
Miedema thesis	Computer Science	List all IDs and names of customers living in Eindhoven.	`SELECT cId, cName FROM Customer WHERE city = "Eindhoven";`
CS Database Laboratory	Computer Science	List the student ID and names of students enrolled before the academic year 2007/2008 who are not yet in their thesis phase (have not been assigned any advisor).	`SELECT * FROM studenti WHERE iscrizione<2007 AND relatore ISNULL;`
CE Database Laboratory	Computer Engineering	Select the names of the products in the beverages category (bevande) that were not ordered in 2023.	`SELECT Prodotto.idProd FROM Prodotto, ORDINE, DETTAGLIO-ORDINE WHERE PRODOTTO.IDPROD = DETTAGLIO-ORDINE.idProd AND DETTAGLIO-ORDINE.QUANTIT = 0 AND ORDINE.DATA = '2023';`
Database Exam	Computer Science	Italian players who have participated in both the US Open and the Australian Open in the same year	`SELECT DISTINCT IDG FROM GIOCATORI WHERE IDG IS IN (SELECT DISTINCT IDG FROM GIOCAIN NATURAL JOIN ONC)`

Table 4.2: Students Sample Collected Queries

4.2.2 Collecting Queries Produced by ChatGPT

To gather queries created by ChatGPT, we ask the systems (ChatGPT 4o and ChatGPT 4o mini) to create an SQL query starting from a schema and the natural language specification of the query. We consider the identical set of SQL queries that the students have been assigned, with identical schema specification and natural language specification. We execute each request three times for ChatGPT 4o to get a range of SQL queries. These queries will be noted and marked with pertinent information like the iteration count and job identification. With ChatGPT 4o Mini, the same procedure is performed, running each of the requests three times and compiling the produced searches. For this study, we generated over 150 answers to queries using ChatGPT, covering scenarios from exams,

laboratory exercises, and the Miedema thesis database. See Figure 4.1 that indicates the sample process of preparing ChatGPT for asking queries.

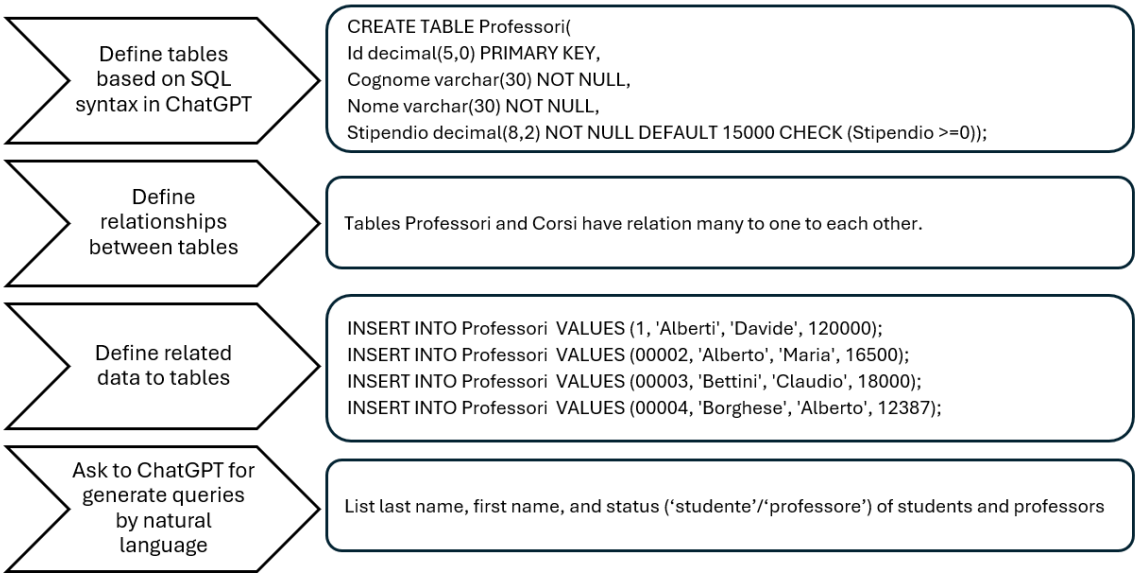

Figure 4.1: Sample Process of Preparing ChatGPT for Asking Queries

Every query gathered from both ChatGPT versions will be kept in an orderly manner so that every query can be easily identified and retrieved. We will guarantee integrated data management at every stage of the data collection process by using an Excel file with fixed columns. The columns include AnswerID, QueryID, the query definition (natural language request), the SQL code (response), categorisation, execution results, notes, checked by, and the misconception type, type of writer (student or chatGPT). This methodical methodology will enable us to methodically document, arrange, and evaluate the SQL inquiries so that all pertinent data is gathered and readily available for further research. This approach guarantees a complete comparison of SQL learning misconceptions between human students and generative AI models.

Table 4.3 is a sample of the information collected from various sources, illustrating the range of SQL queries categorised by complexity.

Reference	Written by	Query Specification	ChatGPT Answer
Miedema thesis	ChatGPT-4o	List all IDs and names of customers living in Eindhoven.	`SELECT cID, cName FROM customer WHERE city = 'Eindhoven';`
CS Database Laboratory	ChatGPT-4o	List the list of degree courses, in alphabetical order by faculty and course name, activated before the academic year 2006/2007 (excluded) and after the academic year 2009/2010 (excluded).	`SELECT Facolta, Denominazione FROM CorsiDiLaurea ..`
CE Database Laboratory	ChatGPT-4o-mini	List the computer science students who passed Databases 1 (basi di dati 1) but not Graphic Interfaces (interfacce grafiche) in June 2010.	`SELECT DISTINCT s.Matricola, s.Cognome, s.Nome FROM Studenti s JOIN Esami e1 ON s.Matricola = e1.Studente JOIN Corsi c1 ON e1.Corso = c1.Id ..`
Database Exam	ChatGPT-4o-mini	German players/hedgehogs who have never participated in Wimbledon.	`SELECT g.IdG, g.Nome, g.Cognome FROM GIOCATORE g ..`

Table 4.3: ChatGPT Sample Collected Queries

4.3 Identifying Misconceptions Produced by Students and ChatGPT

In this section, we discuss the categorisation and other relevant information for every query ChatGPT generates and for students in our file. Using the comprehensive Excel file, we will ensure that every query is correctly classified and has necessary information. The classification is based on found sorts of misconceptions and other pertinent criteria to provide a disciplined analysis of the mistakes.

Once the queries have been categorised, we will run every one using a tool we have developed. Designed to manage all SQL tables and data, this utility links to a PostgreSQL database to do the searches. After that, we analysed each result of the tool and added or removed some output and transferred it to Excel. Evaluating the execution outcome of

every query helps the tool to ascertain if the query executes successfully or comes across problems. Figure 4.2 contains a screenshot of our created tool. This tool has three main parts, which include query classification (1), query display (2), and query execution (3).

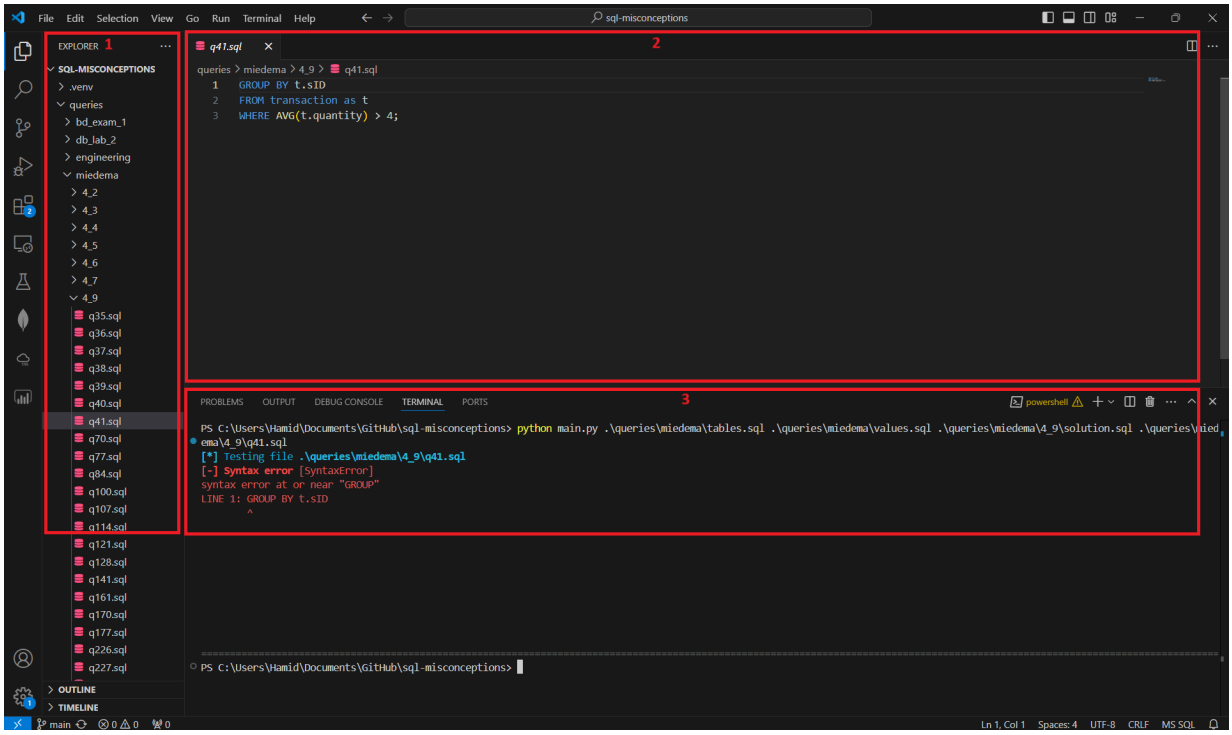

Figure 4.2: Tool Implemented for Identifying Misconceptions

We generated a dataset spanning 1,000 misconceptions using the information acquired and the categories of misconceptions identified in Chapter 2. This extensive collection provides a strong foundation for analysing the basic reasons for common errors in SQL query creation, therefore helping to identify patterns and a better understanding of the roots of certain misconceptions.

Process-wise, the searches—drawn from a variety of real-world academic and research settings—including tests, laboratory activities, and insights from the Miedema thesis—generated using ChatGPT were categorised based on difficulty. Specific criteria, including the number of joins, the requirement of grouping, and the existence of subqueries, influenced this classification of difficulty level (Simple, Medium, and Hard). Every degree of query complexity illustrates the range of logical structures and relational connections that students and artificial intelligence models have to negotiate, therefore offering important data for analysis of how mistakes show up in various query contexts.

Combining a broad range of settings and complexity, this structured dataset helps us to

explore further the difficulties students and AI models have with SQL, analysing frequency and types of misconceptions throughout query complexity. (See Appendix B for the sample Excel file to collect all data.)

4.4 Summary of Analysis Dimensions and Dataset Distribution

Dimensions of Analysis

Misconception Categories: The analysis focuses on four main misconception types:

- Logical
- Syntactic
- Complication
- Semantic

Complexity Levels: Queries are categorised as Simple, Medium, or Hard to understand the impact of difficulty on misconception rates.

Authors: Comparisons are drawn between students and ChatGPT to highlight differences and similarities in their SQL query approaches.

Context: Queries are analysed across different schemas, including:

- Miedema Thesis
- Computer Science Database Laboratories
- Engineering Database Laboratories
- Database Exam

Dataset Distribution

Query Complexity:

- Simple Queries: 20 percent of the total dataset
- Medium Queries: 35 percent of the total dataset

- Hard Queries: 45 percent of the total dataset

Student Demographics:

- Computer Science (CS): 70 percent of students
- Computer Engineering (CE): 30 percent of students

Misconception Categories:

- Logical: 35.2 percent
- Syntactic: 34 percent
- Complication: 19.2 percent
- Semantic: 11.6 percent

Chapter 5

Results

In this chapter, we analyse the misconceptions present in queries formulated by students and those resulting from ChatGPT in relation to corresponding SQL queries. The aim of the analysis is to see how closely the misconceptions generated by generative AI correspond with the misconceptions among SQL-learning students. This overall goal of the analysis is to explore the possibilities artificial intelligence presents as a diagnostic tool in learning environments.

We first provide an overview of the results based on the types of misconceptions observed in all queries in Section 5.1. Next, we investigate the relationship between each writer (i.e., students or ChatGPT) that is based on the kinds of misconceptions in Section 5.2. Next, in Section 5.3 we evaluate how the complexity of SQL queries affects misconceptions for both ChatGPT and students. Finally, we explore the comparison of ChatGPT and students on individual queries in Section 5.4.

After that, we show the comparison of the top 10 misconceptions between students and ChatGPT in Section 5.5. In Section 5.6, we highlight parallels and contrasts between the top 10 categories of misconceptions for ChatGPT-4o and ChatGPT-4o-mini. Next, we compare, in Section 5.7, queries by ChatGPT and students based on the occurrence of misconceptions in both. Finally, we provide the main chart, "top 10 Misconceptions by ChatGPT and Students" in Section 5.8.

5.1 Overview of Result Based on Misconceptions Type

Understanding the types of misconceptions that are common in SQL queries is critical to identifying where learners, whether human or AI, struggle the most. This categorisation helps trainers target specific areas for improvement in SQL training. Figure 5.1 shows the

percentage of misconceptions based on each misconception type.

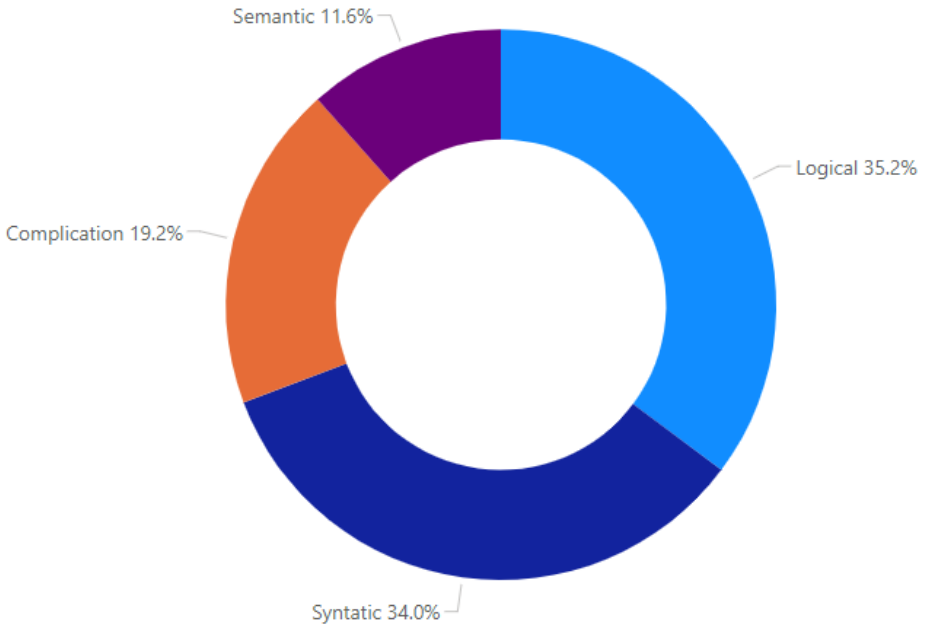

Figure 5.1: Percentage of Misconception Types

The chart offers a general picture of the categories of misconceptions. The most common logical misconception is 35.2 percent; syntactic misconceptions come second closely at 34 percent. Semantic misconceptions account for just 11.6 percent; complications-related misconceptions account for 19.2 percent. This points out that logical and syntactic misconceptions are the most typically occurring types of mistakes because complexity and semantic errors are least probable.

5.2 Correlation between Misconception Category and Author (Students vs ChatGPT)

Comparing various authors' misconceptions helps us to see trends and variations in how generative AI models and human pupils handle SQL queries. Figure 5.2 shows the correlation between misconception types and author types.

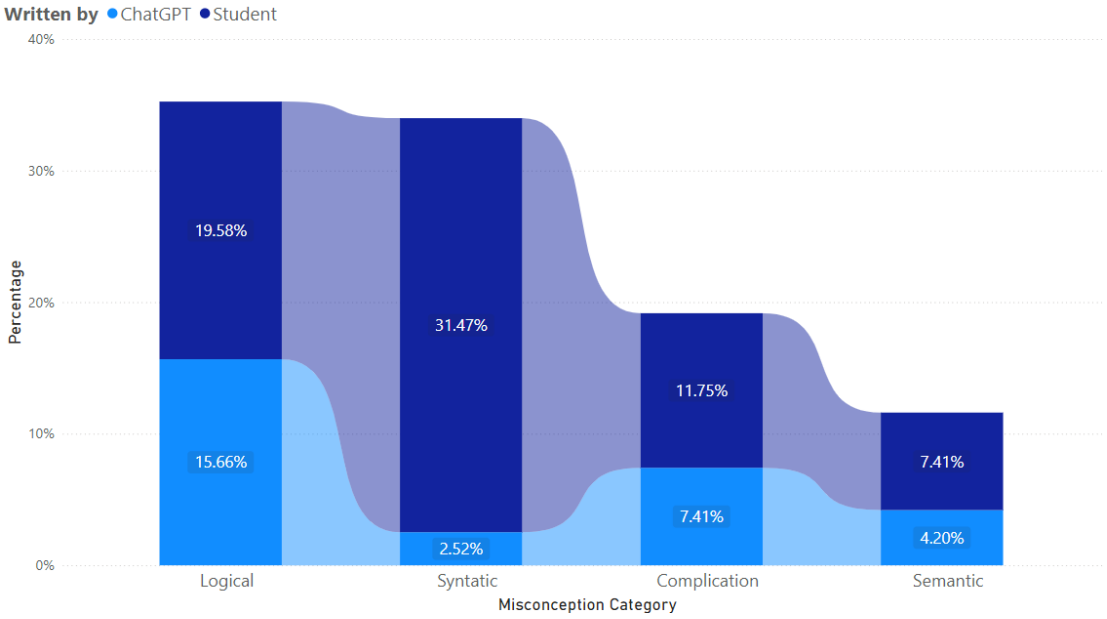

Figure 5.2: Correlation Between Misconception Types and Author Type (Students vs ChatGPT)

This chart compares SQL misconceptions between ChatGPT and students across the four main categories, namely logical, syntactic, complication, and semantic. Both are relatively high, with logical misconceptions for ChatGPT at 15.66 percent and for students at 19.58 percent. This indicates both have equal logical errors in the sense that either the reasoning is wrong or there are flaw conditions within the query. The outcome is that both of them need to be more focused on logical structuring when teaching SQL.

Syntactic misconceptions are sharply different: student queries contain a lot more syntactic misconceptions than ChatGPT, with 31.47 percent for the former and 2.52 percent for the latter. That reflects the fact that students are really struggling with the syntax of SQL, while ChatGPT is more capable of following syntax rules in a highly consistent manner.

Complications due to misplaced use of some SQL features are more present in the output generated by ChatGPT: 7.41 percent vs. 11.75 percent for students' queries. That would point to the possibility that ChatGPT may have a tendency to include redundant joins or non necessary columns or in general to formulate over-complex queries. This could be due to the fact that most of the queries on which ChatGPT has been trained are more complex that those used in training batabase students at Bachelor level and also suggests that perhaps with focused model adjustments, ChatGPT could become more efficient query-wise.

Finally, Semantic errors denote the failure to understand data or query context, and in

both groups, it stands a bit low; however, for students, it stands at 7.41 percent while for ChatGPT is 4.20 percent. That suggests that there was a rather good foundation in understanding the meaning of the data provided and the context of the query, even though the ability may still need more emphasis within the settings of an educational framework.

5.3 Impact of SQL Query Complexity on Misconceptions

Figure 5.3 is aimed at analysing the impact of query complexity on misconceptions, to understand whether the difficulty level of SQL queries influences the error rates and how these rates differ between generative AI and students.

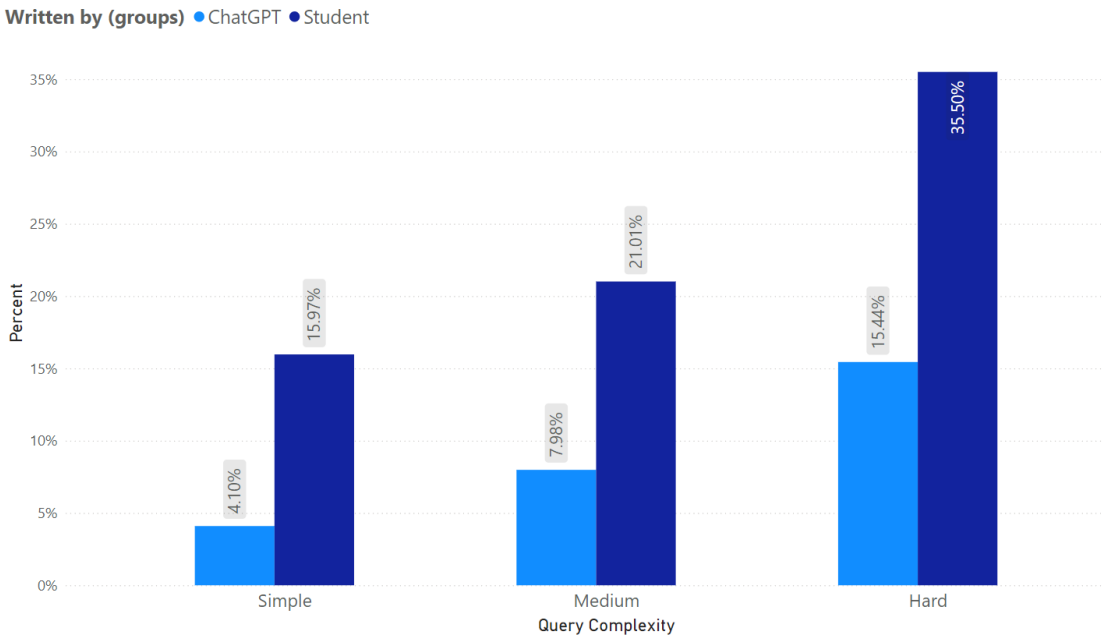

Figure 5.3: Percentage of Misconceptions in Each Complexity Category

The picture illustrates the distribution of misconceptions between students and ChatGPT across various degrees of query complexity: Simple, Medium, and Hard. Each bar illustrates the proportion of mistakes identified in query categorized by complexity, so offering insights into the performance of each group across varying levels of difficulty.

This chart clearly demonstrates that as the complexity of queries increases, the rate of misconceptions rises for both students and ChatGPT, albeit at different scales.

For students, the error rate grows from 15.97 percent for simple queries to 35.50 percent for hard queries. This suggests a direct relationship between query complexity and mistake probability. The significant rise in misunderstandings implies that students struggle more in controlling the logical structure, syntax, and general knowledge needed to build reliable SQL statements as searches become more complex.

For ChatGPT, too, the error rate rises with complexity from 4.10 percent for simple searches to 15.44 percent for hard searches. Although the AI has a lower general error rate than students, the increasing trend implies that tough questions demand more complicated thinking and accurate query phrasing, which may lead errors even for an experienced AI system.

The consistent increase in error rates for both groups underlines a fundamental trend: as the cognitive and technical demands of SQL queries grow, so does the chance of errors. This emphasises the need for better educational approaches to help students tackle complex queries and also suggests areas where AI like ChatGPT could be further refined to improve its handling of higher-complexity tasks.

5.4 Comparison of ChatGPT and Students on Individual Queries

By examining specific queries (Query Name), we can identify particular instances where both AI and students struggle, providing a granular view of the misconception patterns. Figure 5.4 shows for each query (Query Name) the comparison of the errors by ChatGPT and students.

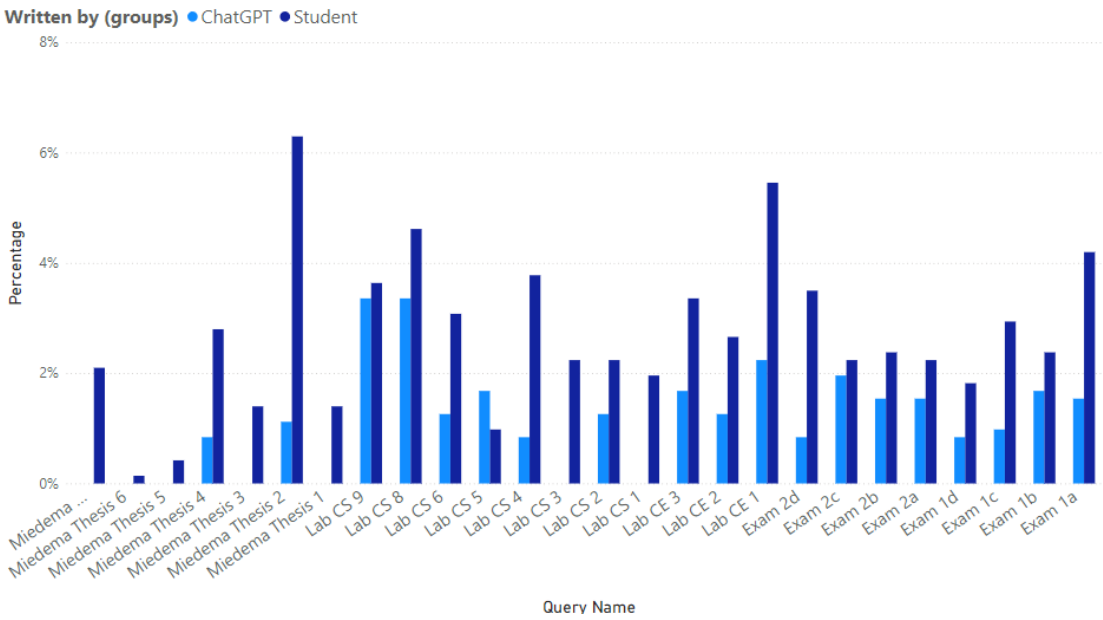

Figure 5.4: Percentage of Correlation of ChatGPT and Students for Each Query

By ChatGPT and students over different query areas—Miedema Thesis, Computer Science and Engineering labs, and Exam queries—this graph compares the percentage of SQL query misconception. The y-axis represents the percentage of error; the x-axis shows each query.

Some queries—especially those from CS labs—such as Lab CS 1 and queries like Exam 2d—seem to have more student misconceptions than others. That would imply that specific queries are more likely to bring complicated SQL procedures or create specific difficulties that cause pupils to generate error-ridden output. The final scores for ChatGPT distribute the error rates more evenly, though there were noticeable spikes in many of the same queries, hence showing points where both students and the AI model have shown to struggle.

Whereas queries from the Miedema Thesis set indicate a minor inclusion of misconceptions in both ChatGPT and students, this might refer to simpler SQL structures or SQL more conformed to standards. On the other hand, lab and exam appear to introduce more complex queries, as defined by a greater number of variations or restrictions in SQL assignments.

Thus, this analysis suggests that lab and exam questions have a greater complexity that could be served by more specific instructions. In terms of ChatGPT, this insight into which types of SQL tasks are more likely to be associated with higher misconception rates helps to inform training adjustments that may better help the model to match the specific challenges that students face in the educational environment.

5.5 Comparison Computer Science and Computer Engineering Top 10 misconceptions

Figure 5.5 shows the correlation between misconceptions in computer science and computer engineering students. We should consider that the number of misconceptions related to engineering was less than computer science because we just have access to 3 queries of the computer engineering labratory, so it affects the distribution of data, but for us it is important to just extract and analyse common misconceptions between both groups.

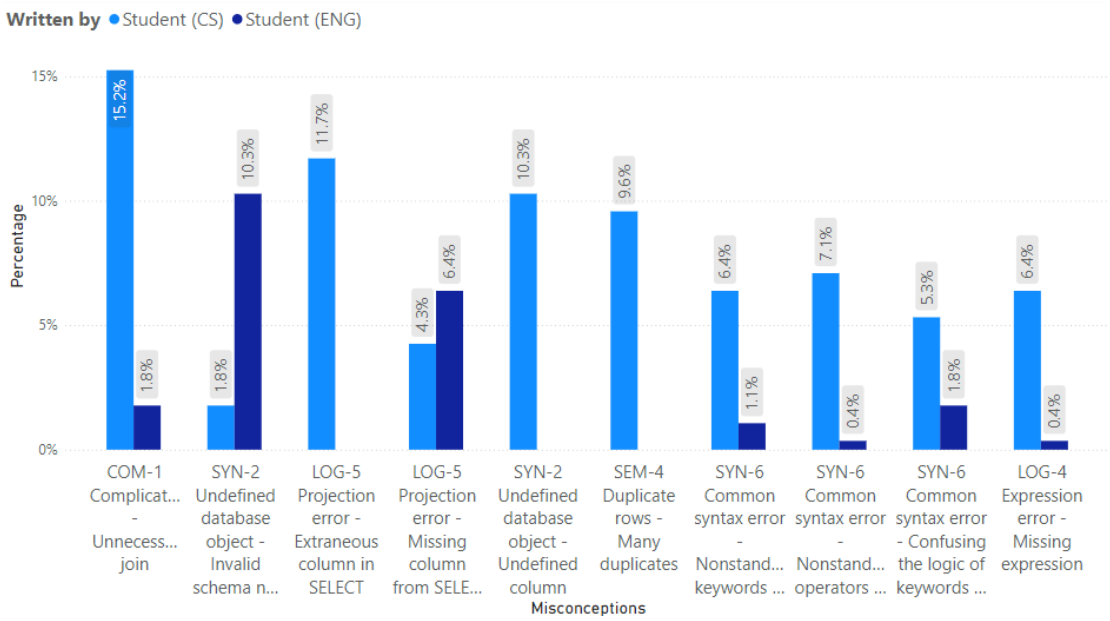

Figure 5.5: Percentage of Comparison CS and CE Top 10 Categorization

This chart shows SQL misconceptions for two sets of students, Computer Science and Engineering. The bars compare the frequency of each group's errors across given categories; the percentage of errors related to each type of misconception is reflected. Common misconceptions between Computer Science and Engineering include unnecessary joins, invalid schema names, missing columns from select, nonstandard keywords or standard keywords in the wrong context, nonstandard operators, confusing the logical of keywords, and missing expressions.

These misconceptions are common among both Computer Science and Computer Engineering students because they stem from fundamental challenges in learning SQL. Concepts like joins, schema names, column selection, and the correct use of keywords or operators require a solid understanding of relational databases and query logic, which both groups are still developing.

5.6 Comparison of ChatGPT-4o and ChatGPT-4o-mini Top 10 misconceptions

Comparing different versions of generative AI models in Figure 5.6 allows us to evaluate their relative performance and identify specific strengths and weaknesses, guiding future development and application.

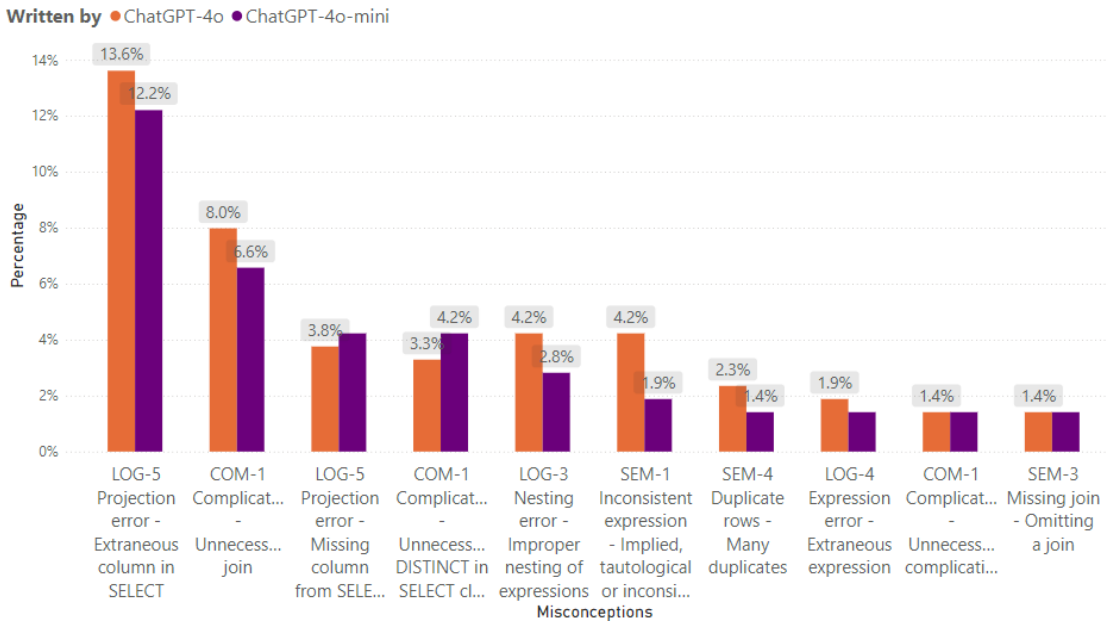

Figure 5.6: Percentage of Misconception Based on ChatGPT-4o and ChatGPT-4o-mini

Two gerative AI models from ChatGPT-4o and ChatGPT-4o-mini expose the most typically occurring types of SQL errors when compared. Every bar shows the mistake rate within a certain group. With regard to their handling of projection errors (LOG-5), this study indicates different behaviour across the models. ChatGPT-4o shows a higher error rate at 13.6 percent compared to ChatGPT-4o-mini's 12.2 percent, with both models tending to include unnecessary columns in SELECT operations.

In the Complication category (COM-1), which involves unnecessary joins, ChatGPT-4o records an 8.0 percent error rate, while ChatGPT-4o-mini has a slightly lower rate at 5.6 percent. This suggests difficulties in determining essential joins for both models, with ChatGPT-4o being somewhat more prone to these complications.

Projection Errors related to missing columns (LOG-5) yield similar results, with ChatGPT-4o at 3.8 percent and ChatGPT-4o-mini at 3.3 percent, reflecting comparable challenges in selecting the required columns for queries.

Furthermore, ChatGPT-4o has a somewhat higher rate of Nesting Errors (LOG-3) at 4.2 percent compared to ChatGPT-4o-mini's 2.8 percent, and Inconsistent Expressions (SEM-1), wherein ChatGPT-4o reports 4.2 percent against ChatGPT-4o-mini's 1.9 percent. These variations imply that while both models sometimes struggle with expression structure, ChatGPT-4o has somewhat more trouble managing these challenging situations.

5.7 Related Queries Between Students and ChatGPT

Figure 5.7 links ChatGPT frequency of misconceptions in SQL queries generated by students.

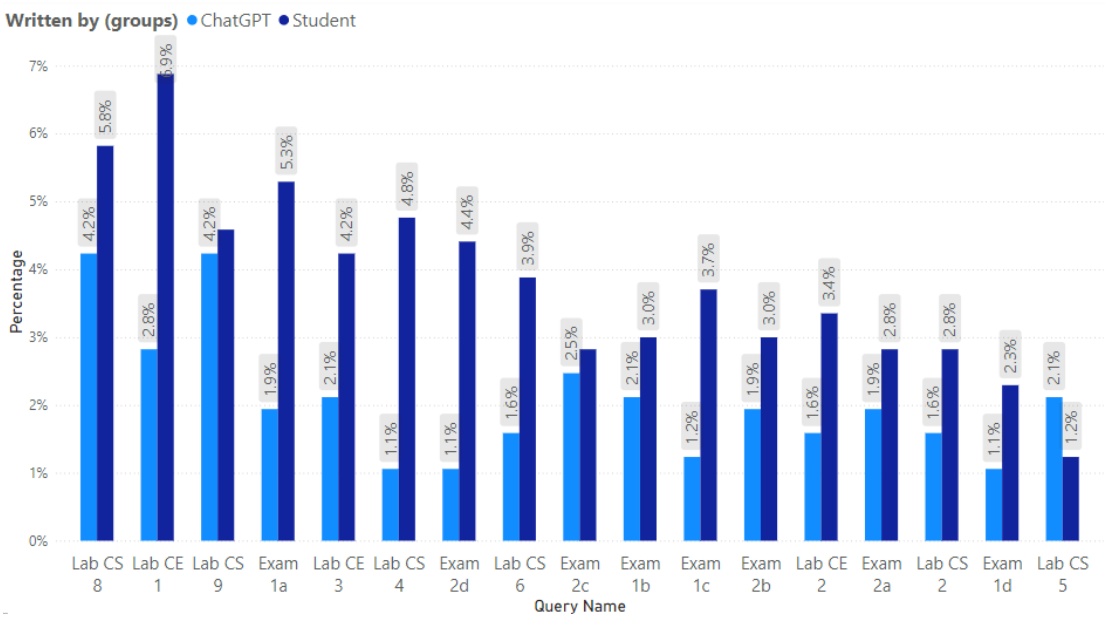

Figure 5.7: Percentage of Related Queries Between Students and ChatGPT

Each bar displays the percentage of misconceptions for a certain query; darker blue bars show student misconceptions and lighter blue bars indicate ChatGPT misconceptions.

5.8 Top 10 Misconceptions by ChatGPT and Students

Based on the previous chart in Figure 5.7, we provide the top 10 misconceptions occurring in ChatGPT and student queries in Figure 5.8. The top 10 SQL misconceptions found among

the more than 100 kinds of misconceptions gathered from different research publications are shown on this chart.

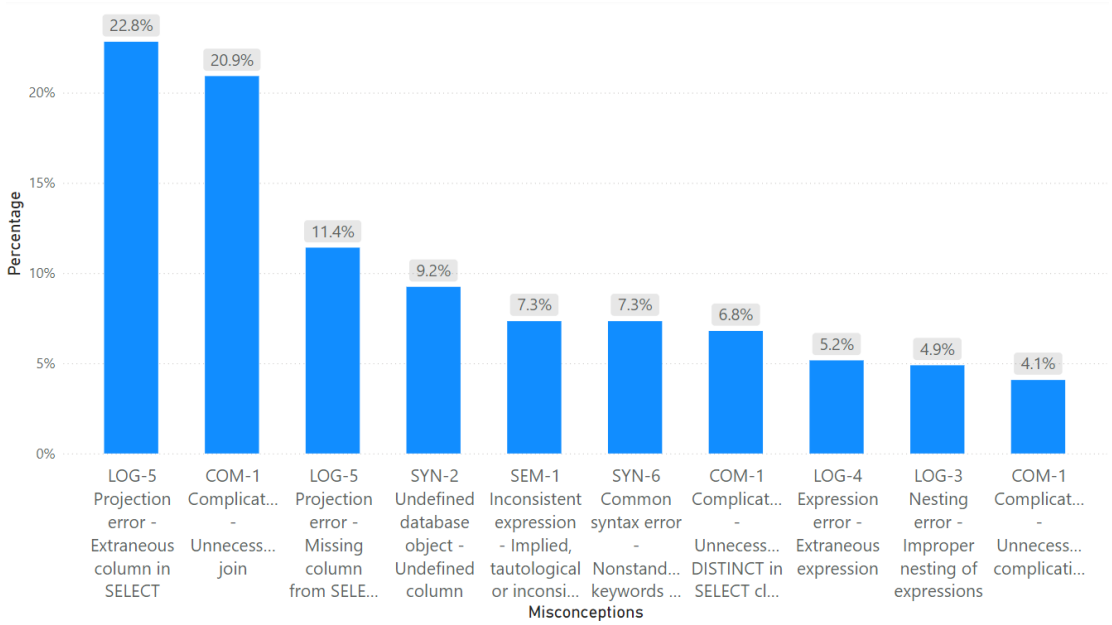

Figure 5.8: Top 10 Misconceptions by ChatGPT and Students

Providing a targeted view of the recurrent difficulties in SQL query design, the graphic shows the top 10 SQL misconceptions that often arise among both ChatGPT and student-generated queries. Projection misconceptions, including obtaining unnecessary columns in the SELECT clause, which implies that both students and ChatGPT frequently fail to limit their queries to just the important data. Examining closely issues with unnecessary joins exposes a general misconception about which tables and conditions are really required to satisfy query aims.

Other well-known misconceptions include missing columns in SELECT queries, undefined database objects, and erroneous expressions, each reflecting issues in accurate column selection, schema traversal, and logical consistency. More problems, including nonstandard syntax misconceptions and distinct use, highlight weaknesses in SQL's best practices and usefulness. These top 10 misconceptions expose common challenges between generative AI models and students and learners, therefore providing a road map for areas needing focused instruction and AI training. We will further examine these results in the upcoming chapter to expose the fundamental causes and ramifications of these widespread misconceptions.

5.9 Summary

In this chapter, the student-generated queries and SQL errors found in ChatGPT were examined in more detail. We carefully analysed and compared these errors to assess how well the misconceptions produced by generative AI matched those seen in students learning SQL.

In the next chapter, we will explore the branches of our conclusions and investigate ways to eliminate misconceptions and therefore further explore these findings, also discussing how generative AI can be a diagnostic tool to find and fix typical challenges in the SQL learning environment.

Chapter 6

Discussion

The data from the charts reported in Chapter 5 provide understanding of both generative artificial intelligence models and student performance with relation to SQL misconceptions. More specifically, the results expose several kinds of mistakes produced by the generative AI systems and show where student learning either fits or deviates from these mistakes. In this chapter, we analyse our results based on the top 10 misconceptions occurring among students and chatGPt and finally try to answer our research questions.

6.1 Interpretation of Top 10 Misconceptions

In this section, we analyse the top 10 misconceptions emerging from students and chatGPT. This provides insights into why it happens and what we should consider in the learning path for students by using chatGPT. In each subsection, we first report the information request specification in natural language, then include a table with a correct expected query, a student query, and a chatGPT query, that are then analyzed and discussed in the text.

6.1.1 Extraneous Column in SELECT

Question: Identify German players who have never participated in Wimbledon.

Query Type	SQL Query

Expected Query	`SELECT DISTINCT IdG`
	`FROM GIOCATORE`
	`WHERE Nazione = 'Germania'`
	`AND IdG NOT IN (SELECT DISTINCT IdG`
	`FROM GIOCAIN NATURAL JOIN TORNEO`
	`WHERE NomeT = 'Wimbledon');`

Student Query	`SELECT *`
	`FROM GIOCATORE`
	`WHERE NAZIONE = 'Germania'`
	`AND IdG NOT IN (`
	`SELECT IdG`
	`FROM GIOCATORE`
	`NATURAL JOIN GIOCAIN`
	`NATURAL JOIN REGISTRAZIONE`
	`NATURAL JOIN TORNEO`
	`WHERE TORNEO.LUOGO = 'Wimbledon'`
	`);`

ChatGPT Query	`SELECT g.Nome, g.Cognome`
	`FROM GIOCATORE g`
	`WHERE g.Nazione = 'Germany'`
	`AND g.IdG NOT IN (`
	`SELECT gi.IdG`
	`FROM GIOCAIN gi`
	`JOIN REGISTRAZIONE r ON gi.IdT = r.IdT AND`
	`gi.IdCat = r.IdCat AND gi.NumRegistrazione`
	`= r.NumRegistrazione`
	`JOIN TORNEO t ON r.IdT = t.IdT`
	`WHERE t.NomeT = 'Wimbledon'`
	`);`

Table 6.1: Comparison of SQL Queries Based on Extraneous Column in SELECT

Analysis of Extraneous Column in SELECT Misconception

Student Query: The students use of `SELECT *` pulls all columns from `GIOCATORE` instead of simply `IdG`, therefore adding pointless data and straying from the purpose of identifying players only by `IdG`.

ChatGPT Query: ChatGPT selects additional columns `(Nome, Cognome)`, which aren't needed for the task, indicating a projection error. While the structure is more accurate than the student's query, this extra data reduces query clarity and performance.

Both the student and ChatGPT do not use the exact required columns. Students use `SELECT *` thus returning all columns, and ChatGPT returns additional columns. Emphasising the need of choosing only necessary columns would help students as well as artificial intelligence models like ChatGPT to solve this. Together with training on exact projection, a more targeted strategy to grasp query aims will help to provide cleaner, more effective queries more in line with expected outcomes.

One must recognise the careful utilisation of `DISTINCT` is vital. Teaching students when to use `DISTINCT` helps to avoid duplicate results—particularly in queries requiring many joins. For ChatGPT, raising knowledge of output requirements—such as choosing only unique IDs when needed—would improve accuracy and clarity, thereby enabling returns that are summary and fit for the purpose of the inquiry.

6.1.2 Unnecessary Join

Question: Players who participated in both the 'single ' and 'double ' categories in the same tournament.

Query Type	SQL Query

Expected Query	`SELECT DISTINCT IdG`
	`FROM GIOCATORE NATURAL JOIN GIOCAIN NATURAL`
	`JOIN CATEGORIA WHERE Genere = 'f' AND`
	`NomeCategoria = 'singolo'`
	`AND (IdG, IdT) IN (SELECT DISTINCT IdG, IdT`
	`FROM GIOCAIN NATURAL JOIN CATEGORIA`
	`WHERE NomeCategoria = 'doppio')`

Student Query	`SELECT IDG`
	`FROM GIOCATORE G NATURAL JOIN GIOCAIN`
	`NATURAL JOIN REGISTRAZIONE R`
	`NATURAL JOIN CATEGORIA`
	`WHERE GENERE = F, NOMECATEGORIA = SINGOLI`
	`AND EXISTS`
	`(`
	`SELECT *`
	`FROM GIOCATORE N.JOIN GIOCAIN N.JOIN`
	`REGISTRAZIONE N.JOIN CATEGORIA`
	`WHERE IDG = G.IDG AND`
	`R.IDT = REGISTRAZIONE.IDT`
	`AND NOMECATEGORIA = 'DOPPI'`
	`);`

ChatGPT Query	`SELECT DISTINCT g.IdG, g.Nome, g.Cognome`
	`FROM GIOCATORE g`
	`JOIN GIOCAIN g1 ON g.IdG = g1.IdG`
	`JOIN GIOCAIN g2 ON g1.IdT = g2.IdT AND`
	`g1.IdCat <> g2.IdCat`
	`JOIN CATEGORIA c1 ON g1.IdCat = c1.IdCat`
	`JOIN CATEGORIA c2 ON g2.IdCat = c2.IdCat`
	`WHERE c1.NomeCategoria = 'singolo'`
	`AND c2.NomeCategoria = 'doppio';`

Table 6.2: Comparison of SQL Queries Based on Unnecessary Join

Analysis of Unnecessary Join

Student Query: The student's focus is on nonusful links, especially way the `REGISTRAZIONE` table is included in both the main and the subquestives. For the query—that is, to identify `players (IdG)` involved in both the singles `(IdG)` and doubles `(NomeCategoria = "doppio"`—this table provides no relevant filtering or data retrieval. The repeated occurrence of `REGISTRAZIONE` complicates the query structure excessively and makes interpretation and execution more difficult. This reflects a belief that, even in cases where such tables lack relevant information, more tables connected in a query increase its resilience.

ChatGPT Query: Similar needless joins by duplicating `GIOCAIN` and `CATEGORIA` into aliases `(g1, g2, c1, c2)` are introduced by the ChatGPT query. Although these joins seek to compare `categories (singolo and doppio)` within the same tournament, the structure entails needless duplication as fewer joins might get the same outcome. Furthermore, adding extra columns like `Nome` and `Cognome` to the `SELECT` clause goes beyond the query's need and increases even more processing complexity. This implies a misconception wherein exactly specifying all fields or replicating tables improves query accuracy but, in reality, it confuses the thinking and reduces efficiency.

Both queries reveal a common SQL misconception about using joins that including unnecessary tables not only complicates the queries but also increases the likelihood of logical errors and inefficiencies. This suggests that both students and generative AI models, like ChatGPT, would benefit from targeted instruction on join selection, focusing on including only essential tables for the query purpose.

A more targeted approach to query needs and table connections will help to optimise student and ChatGPT queries and prevent needless joins. Encouragement of students to specify the particular data required for a query helps them to restrict their joins to only those tables directly contributing to the outcome. Emphasising the goal of every join and challenging its requirements helps pupils learn to include only the tables, enhancing the output.

For ChatGPT, prompt engineering and fine-tuning should focus on enhancing context-specific assessments, thereby enabling the model to better understand task requirements and find when a particular table is worthless for the query. Moreover, demonstrating the performance impact of meaningless joins might motivate generative AI as well as students to adopt more straightforward, direct queries. Promoting purpose-driven joins, this better approach would enable SQL query creation to be more precise, clear, and effective.

6.1.3 Missing Column from SELECT

Question: Select the total amount of the last order made by the customer having id "1234".

Query Type	SQL Query
Expected Query	`SELECT O.idOrd, SUM (prezzo*d.quantita) AS TotaleOrdine FROM Ordine O JOIN DettaglioOrdine D ON O.idOrd=D.idOrd JOIN Prodotto P ON D.idProd = P.idProd WHERE O.idClient = '1234' AND O.data = ( SELECT MAX(data) FROM Ordine WHERE idClient='1234') GROUP BY O.idOrd`
Student Query	`SELECT SUM(PRODOTTO.PREZZO) FROM PRODOTTO INNER JOIN DETTAGLIO_ORDINE ON PRODOTTO.IDPROD=DETTAGLIO_ORDINE.IDPROD INNER JOIN ORDINE ON DETTAGLIO_ORDINE.IDORD=ORDINE.IDORD WHERE ORDINE.DATA= (SEELCT MAX (DATA) FROM ORDINE INNER JOIN CLIENTE ON ORDINE.IDCLIENT=CLIENTE.IDCLIENT WHERE CLIENTE.IDCLIENT='1234') AND CLIENTE.IDCLIENT='1234'`
ChatGPT Query	`SELECT SUM(p.prezzo*do.quantita) AS totalAmount FROM Ordine o JOIN DettaglioOrdine do ON o.idOrd = do.idOrd JOIN Prodotto p ON do.idProd = p.idProd WHERE o.idClient = 1234 AND o.data = ( SELECT MAX(data) FROM Ordine WHERE idClient = 1234);`

Table 6.3: Comparison of SQL Queries Based on Missing Column From SELECT

Analysis of Missing Column From SELECT

Student Query: Emphasising just `SUM(PRODOTTO.PREZZO)`, the student's query `"o"` misses the `O.idOrd` column. This helps the question to fall short in providing necessary background on the particular order ID connected with the overall determined value. Furthermore, the lack of quantity —`quantità`— in the computation reveals a partial awareness of the needs for the inquiry. This control produces an erroneous total and leaves uncertainty on which sequence the total amount pertains to. Such exclusions draw attention to misconceptions about the choice of only pertinent columns that completely meet the objective of the query.

ChatGPT Query: Though it computes the total amount properly using `SUM(p.prezzo * do.quantità)`, ChatGPT's query omits `O.idOrd` in the `SELECT` clause, same as the student query. Without the `order ID`, the response lacks the needed precision, especially when multiple orders are involved, increasing the risk of misinterpretation. The absence of this key detail suggests a limited understanding of the assignment's requirements, which reduces the clarity and completeness of the answer, even if ChatGPT's approach to calculating the total aligns more closely with the question than the student's.

Both queries define a frequent misconception in SQL queries where omitting significant data like the order ID lowers the usefulness of the query return. This emphasises how crucial it is for students to understand the need for providing results in their complete context. Training should highlight the need to include basic essential IDs in `SELECT` clauses to provide unambiguous, significant results.

By using a focused approach to SQL learning, raising knowledge of the purpose behind every chosen column would aid in enhancing both student and ChatGPT queries. Encouragement of students to validate that their queries meet the complete informational demand of the assignment will help them to avoid missing significant columns. By prompt-engineering or fine-tuning ChatGPT, its ability to grasp specific task demands might be enhanced, thus pointing out when necessary columns—such as order IDs—are required. Encouragement of the use of all pertinent data allows SQL query creation to be more precise, helpful, and efficient for generative AI models as well as human learners.

6.1.4 Undefined Column

Question: Italian players who have participated in both the US Open and the Australian Open in the same year.

Query Type	SQL Query
Expected Query	SELECT DISTINCT IdG FROM GIOCATORE NATURAL JOIN GIOCAIN NATURAL JOIN TORNEO WHERE Nazione = 'Italia' AND NomeT = 'US Open' AND (IdG,YEAR(DataNascita)) IN (SELECT DISTINCT IdG, YEAR(DataNasdcita) FROM GIOCAIN NATURAL JOIN TORNEO WHERE NomeT = 'Australian Open')
Student Query	SELECT IdG FROM GIOCAIN NATURAL JOIN REGISTRAZIONE NATURAL JOIN TORNEO T WHERE T.NomeT = 'US Open' AND EXISTS (SELECT IdG FROM GIOCAIN NATURAL JOIN REGISTRAZIONE NATURAL JOIN TORNEO WHERE NomeT = 'Australian Open' AND DataF = T.DataF);
ChatGPT Query	SELECT DISTINCT g.Nome, g.Cognome, g.Nazione, t1.NomeT AS Torneo1, t2.NomeT AS Torneo2, EXTRACT(YEAR FROM t1.DataI) AS Anno FROM GIOCATORE g JOIN GIOCAIN gi1 ON g.IdG = gi1.IdG JOIN REGISTRAZIONE r1 ON gi1.IdT = r1.IdT AND gi1.IdCat = r1.IdCat AND gi1.NumRegistrazione = r1.NumRegistrazione JOIN TORNEO t1 ON r1.IdT = t1.IdT JOIN GIOCAIN gi2 ON g.IdG = gi2.IdG JOIN REGISTRAZIONE r2 ON gi2.IdT = r2.IdT AND gi2.IdCat = r2.IdCat AND gi2.NumRegistrazione = r2.NumRegistrazione JOIN TORNEO t2 ON r2.IdT = t2.IdTWHERE g.Nazione = 'Italy'AND ((t1.NomeT = 'US Open' AND t2.NomeT = 'Australian Open') OR (t1.NomeT = 'Australian Open' AND t2.NomeT = 'US Open')) AND EXTRACT(YEAR FROM t1.DataI) = EXTRACT(YEAR FROM t2.DataI);

Table 6.4: Comparison of SQL Queries Based on Undefined Column

Analysis of Undefined Column

Student Query: The student's query encounters an issue with an undefined column, particularly in the sub-query's use of `DataF`, which incorrectly references `T.DataF`. Without a clear join or explicit specification of `DataF` in relation to the main `TORNEO` table, this creates an ambiguous reference. This mistake highlights a misconception of scope in sub-queries, where the student assumes `DataF` will automatically link to `T.DataF` in the main query. It is a misconception that can lead to runtime errors or unexpected behaviour in certain SQL environments.

ChatGPT's Query: ChatGPT's query is moving to undefined columns because it includes additional, unnecessary fields like `g.Nome, g.Cognome, and g.Nazione`, along with extra tournament fields.

Both queries reflect a need for greater clarity in context and column selection. The student's query makes a direct reference to an undefined column, while ChatGPT's approach includes ambiguous columns. This comparison highlights an opportunity for more precise instruction on query scoping and selecting only relevant columns. Emphasising the importance of defining all referenced columns within the specific query context—and aligning all fields with the query's purpose—could help prevent such issues.

6.1.5 Implied, Tautological, or Inconsistent Expression

Question: Players who participated in both the singles and doubles categories in the same tournament.

Query Type	SQL Query
Expected Query	SELECT DISTINCT IdG FROM GIOCATORE NATURAL JOIN GIOCAIN NATURAL JOIN CATEGORIA WHERE Genere = 'f' AND NomeCategoria = 'singolo' AND (IdG, IdT) IN (SELECT DISTINCT IdG, IdT FROM GIOCAIN NATURAL JOIN CATEGORIA WHERE NomeCategoria = 'doppio')
Student Query	SELECT IdG FROM FROM GIOCATORE NATURAL JOIN CATEGORIA NATURAL JOIN TORNEO NATURAL JOIN GIOCAIN WHERE genere = F AND AND IdG IN (SELECT IdG FROM GIOCATORE NATURAL JOIN CATEGORIA NATURAL JOIN TORNEONATURAL JOIN GIOCAIN WHERE genere = F AND genereCategoria = Doppi)
ChatGPT Query	SELECT DISTINCT g.IdG, g.Nome, g.Cognome FROM GIOCATORE g JOIN GIOCAIN gi1 ON g.IdG = gi1.IdG JOIN REGISTRAZIONE r1 ON gi1.IdT = r1.IdT AND gi1.IdCat = r1.IdCat AND gi1.NumRegistrazione = r1.NumRegistrazione JOIN CATEGORIA c1 ON r1.IdCat = c1.IdCat JOIN GIOCAIN gi2 ON g.IdG = gi2.IdG JOIN REGISTRAZIONE r2 ON gi2.IdT = r2.IdT AND gi2.IdCat = r2.IdCat AND gi2.NumRegistrazione = r2.NumRegistrazione JOIN CATEGORIA c2 ON r2.IdCat = c2.IdCat WHERE c1.NomeCategoria = 'singles' AND c2.NomeCategoria = 'doppio' AND gi1.IdT = gi2.IdT;

Table 6.5: Comparison of SQL Queries Based on Implied, Tautological or Inconsistent Expression

Analysis of Inconsistent Expression-Implied, Tautological or Inconsistent Expression

Student Query: The inconsistency with the `WHERE` clause was comprised of both duplicated and inconsistent conditions, such as `"genere = F AND AND IdG x IN..."`, that created a tautological versus implied expression. We suspect that such inconsistencies seem to indicate either a gap in understanding the formation of logical expressions or perhaps a trivial syntactic error not corrected by the student.

ChatGPT Query: Though still well-formed, the ChatGPT query is complex by the numerous joins and repeated table references to the same data (e.g., `JOIN GIOCAIN gi1 and JOIN GIOCAIN gi2`), thereby resulting in a duplication on criteria `NomeCategoria ("singles" and "doppio")`. This excessive complications implies all of these relationships that are not needed for the purpose of querying.

Both student and ChatGPT queries are good examples of a common SQL fallacy where predicates become inconsistent or redundant due to implicit conditions making an expression vacuous or tautological. This is a good reminder for students that greater precision in concise logical structuring, where expressions are arranged so that each directly supports the desired conclusion—without irrelevant digressions—is critical. Fine-tuning ChatGPT to better focus on relevant requirements and to avoid implied or additional relations would produce simpler queries with corresponding SQL expressions that are clearer and more efficient. For students, exercises focusing on removing implied conditions that do not contribute to the logical expressiveness of the query would be beneficial.

6.1.6 Nonstandard Keywords or Standard Keywords in Wrong Context

Question: Select the names of the products in the beverages category (bevande) that were not ordered in 2023.

Query Type	SQL Query

Expected Query	`SELECT idProd, P.nome`
	`FROM Prodotto P`
	`JOIN Categoria C ON P.idCat = C.idCat`
	`WHERE C.nome = 'Bevande'`
	`AND idProd NOT IN (`
	`SELECT idProd`
	`FROM Ordine O`
	`JOIN DettaglioOrdine D`
	`ON O.idOrd = D.idOrd`
	`WHERE EXTRACT (YEAR FROM data) = '2023'`
	`)`

Student Query	`SELECT p.Nome`
	`FROM Prodotto p JOIN`
	`Dettaglio_ORDINE ON p.IDProd= d.IDProd`
	`JOIN Ordine O ON O.IDORD = O.IDORDD`
	`WHERE p.IdCat ="bevanDe" AND`
	`EXTRACT(YEAR FROM date)! = 2023`
	`OR O.IDPROD IS NULL`

ChatGPT Query	`SELECT P.nome`
	`FROM Prodotto P`
	`JOIN Categoria C ON P.idCat = C.idCat`
	`LEFT JOIN DettaglioOrdine DO`
	`ON P.idProd = DO.idProd`
	`LEFT JOIN Ordine O ON DO.idOrd = O.idOrd AND`
	`EXTRACT(YEAR FROM O.data) = 2023`
	`WHERE C.nome = 'bevande' AND O.idOrd IS NULL;`

Table 6.6: Comparison of SQL Queries Based on Nonstandard Keywords or Standard Keywords in Wrong Context

Analysis of Nonstandard Keywords or Standard Keywords in Wrong Context

Student Query: The student's query contains many syntax errors due to non-standard use of terms and inappropriate use of SQL conventions. For example, for `IdCat`, `bevanDe` is the value; presumably it should match the `category Bevande`, but the irregular capitalisation and quotation marks around `bevanDe` make this non-standard. In addition, `Dettaglio ORDINE` and `date` are incorrectly capitalised and formatted, leading to ambiguity and potential execution errors. These linguistic errors illustrate the proper formatting of conditional statements and a misreading of SQL's keyword and case sensitivity rules.

ChatGPT Query: ChatGPT's query shows a closer adherence to standard SQL syntax but still contains some inconsistencies in keyword usage. Since "DO" is a reserved word used in SQL, some people may be confused when `DettaglioOrdine` chooses "DO" as a `codename`. It is also difficult to understand how `LEFT JOIN` with a condition on `EXTRACT(YEAR FROM O.data) = 2023` fits into the JOIN statement. ChatGPT's query has no major syntax errors but contains overly complex expressions that could be streamlined, especially in the use of non-standard aliases and keywords in unconventional contexts.

In both queries, syntax errors and non-standard keyword usage reveal common SQL misconceptions. Students often struggle with case sensitivity, poor operators, and inconsistent capitalisation, leading to errors and misconceptions of results. Although better organised, ChatGPT sometimes uses unusual aliases or contextually misleading terms, making the query difficult to read. Emphasising normal SQL syntax norms, case sensitivity, and the correct use of operators can help students reduce errors significantly. For ChatGPT, fine-tuning could improve alias selection and context awareness to minimise the use of confusing or non-standard keywords. Structured practice in the correct use of standard keywords and operators and the avoidance of unconventional aliases would greatly improve the clarity and accuracy of SQL queries for both students and AI models.

6.1.7 Unnecessary DISTINCT in SELECT Clause

Question: Select the names of products that have never been ordered by customers living in Venice or Brescia.

Query Type	SQL Query

Expected Query	`SELECT idProd, nome`
	`FROM Prodotto P`
	`WHERE NOT EXISTS(SELECT nome`
	`FROM Cliente C JOIN Ordine O`
	`ON C.idClient= O.idClient`
	`JOIN DettaglioOrdine D`
	`ON P.idProd=D.idProd`
	`WHERE C.citta IN ('Venezia', 'Brescia')`
	`)`

Student Query	`SELECT P.NOME`
	`FROM PRODOTTO P`
	`WHERE p.idProd NOT IN (`
	`SELECT DISTINCT d.IdProd`
	`FROM Dettaglio_Ordine`
	`JOIN Ordine O ON`
	`d.idOrd= O.idOrd`
	`JOIN Cliente C ON O.IdClient = c.idClient`
	`WHERE C.citta = "venezia" OR`
	`c.citta ="Brescia");`

ChatGPT Query	`SELECT nome`
	`FROM Prodotto`
	`WHERE idProd NOT IN (`
	`SELECT DISTINCT DettaglioOrdine.idProd`
	`FROM DettaglioOrdine`
	`JOIN Ordine ON Dettaglio`
	`Ordine.idOrd = Ordine.idOrd`
	`JOIN Cliente ON`
	`Ordine.idClient = Cliente.idClient`
	`WHERE Cliente.citta IN ('Venezia', 'Brescia')`
	`);`

Table 6.7: Comparison of SQL Queries Based on Unnecessary DISTINCT in SELECT Clause

Analysis of Unnecessary DISTINCT in SELECT Clause

Student Query: Not necessary in this case, the student query uses `DISTINCT` within the sub-query to ensure original `IdProd` values. Using `DISTINCT` here adds extra processing because `NOT IN` filters that depend on unique values do not affect the result. This is a common mistake made by students, who may use `DISTINCT` to deal with apparent duplicate values without fully understanding the purpose of `NOT IN`. Perhaps the student's use of `DISTINCT` shows a lack of confidence in SQL's inherent ability to filter within the `NOT IN` clause, resulting in a pointless and more complicated query.

ChatGPT Query: ChatGPT also uses `DISTINCT` in the subquery, although it is not necessary. By doing this, ChatGPT's query—along with the student's—introduces an additional level of complication that could affect performance. Although the form of the ChatGPT query is more sophisticated, the pointless `DISTINCT` implies that the model lacks sophisticated knowledge of the behaviour of SQL with `NOT IN`, where duplicates have no effect on the output. This duplicity suggests that the AI model errs on the side of caution, but at the cost of efficiency.

Both questions show misconceptions about the use of `DISTINCT` with `NOT IN`. For students, this may be due to a belief that `DISTINCT` is always required to avoid repetition, reflecting an over-reliance on this term without fully understanding its consequences. This may indicate an underlying model behaviour for ChatGPT where redundancy is added to ensure accuracy but without optimisation. Teaching students the specific scenarios where `DISTINCT` is needed and when it is not would help them to produce more efficient SQL queries. Similarly, improving ChatGPT to find situations where `Distinct` is unnecessary would increase its efficiency, thereby generating correct and effective questions.

6.1.8 Extraneous Expression

Question: List the computer science students who passed `Databases 1 (basi di dati 1)` but not `Graphic Interfaces (interfacce grafiche)` in June 2010.

Query Type	SQL Query
Expected Query	`SELECT studente FROM esami WHERE corso = 'bdd1n' AND voto >= 18 AND esami.data BETWEEN '2010-06-01' AND '2010-06-30' EXCEPT SELECT studente FROM esami WHERE corso = 'ig' AND voto >= 18 AND esami.data BETWEEN '2010-06-01' AND '2010-06-30'; )`
Student Query	`select matricola from studenti join esami on studenti.matricola=esami.studente where (corsodilaurea=9 and voto>=18 and esami.corso='bdd1n' and data>'2010-05-31' and data<'2010-07-01') except select matricola from studenti join esami on studenti.matricola=esami.studente where (corsodilaurea=9 and voto>=18 and esami.corso='graf' and data>'2010-05-31' and data<'2010-07-01');`
ChatGPT Query	`SELECT s.Matricola, s.Cognome, s.Nome FROM Studenti s JOIN Esami e1 ON s.Matricola = e1.Studente JOIN Corsi c1 ON e1.Corso = c1.Id LEFT JOIN Esami e2 ON s.Matricola = e2.Studente AND e2.Corso = 'ig' AND e2.Data BETWEEN '06/01/2010' AND '06/30/2010' WHERE s.CorsoDiLaurea = 9 AND c1.Denominazione = 'Basi Di Dati 1' AND e1.Voto >= 18 AND e1.Data BETWEEN '06/01/2010' AND '06/30/2010' AND (e2.Voto IS NULL OR e2.Voto < 18);`

Table 6.8: Comparison of SQL Queries Based on Extraneous Expression

Analysis of Extraneous Expression

Student Query: The student's query introduces unnecessary complexity by joining the `studenti` table with `esami` and including a condition on `corsodilaurea = 9`, which has no bearing on the goal of filtering students based on their exam performance in specific `courses (bdd1n and graf)`. Including this extraneous condition creates doubt and does not assist in defining the main objective of the issue. Moreover, the repetitive, useless duplicate filtering of the date range between both subqueries by the `WHERE` clause does not optimise the reasoning. This is a myth wherein the learner can believe that adding additional criteria or tables improves the accuracy of the query when in reality it just needlessly complicates the statement.

ChatGPT Query: Though syntactically true, ChatGPT's search adds extraneous words that unnecessarily complicate the logic. Joining `Corsi` with redundant conditions like `c1.Denominazione = Basi Di Dati 1` adds overhead without contributing to get the intended result. Additionally, the use of a `LEFT JOIN` with `Esami (e2)` and the condition `e2.Voto IS NULL OR e2.Voto < 18` offers an indirect method to accomplish what may have been achieved more easily with a smaller exclusion criteria. These extra expressions make the query more complex than required, reducing clarity and potentially affecting performance.

Both searches expose a common SQL fallacy: the idea that adding table references or more conditions would improve the correctness or thoroughness of the query. Actually, this method adds pointless complexity, which makes debugging, reading, and optimising searches more difficult.

For pupils, knowing succinct and targeted logical structures is crucial to make sure every condition directly supports the desired result. Emphasising the need to remove pointless joins and superfluous conditions from SQL queries, instructors should fine-tune ChatGPT's capacity to prioritise simplified, context-driven reasoning to assist in lowering unnecessary expressions, thus producing more exact and efficient searches. Exercises aimed at removing duplicates and preserving a clear, purpose-driven query structure will help students as well as artificial intelligence systems.

6.1.9 Improper Nesting of Expressions

Question: List the computer science students who passed Databases 1 (basi di dati 1) but not Graphic Interfaces (interface grafiche) in June 2010.

Query Type	SQL Query
Expected Query	`SELECT studente FROM esami WHERE corso = 'bdd1n' AND voto >= 18 AND esami.data BETWEEN '2010-06-01' AND '2010-06-30'EXCEPT SELECT studente FROM esami WHERE corso = 'ig' AND voto >= 18 AND esami.data BETWEEN '2010-06-01' AND '2010-06-30';`
Student Query	`SELECT studenti.matricola, studenti.nome, studenti.cognome FROM studenti JOIN esami on studenti.matricola = esami.studente JOIN corsi on esami.corso = corsi.id WHERE corsi.denominazione = 'Basi Di Dati 1' AND esami.data >= '2010-06-01' and esami.data <= '2010-06-30' AND studenti.corsodilaurea = (SELECT id from corsidilaurea where corsidilaurea.denominazione = 'Informatica')`
ChatGPT Query	`SELECT s.Matricola, s.Cognome, s.Nome FROM Studenti s JOIN Esami e1 ON s.Matricola = e1.Studente JOIN Corsi c1 ON e1.Corso = c1.Id WHERE c1.Denominazione = 'Basi Di Dati 1' AND e1.Data BETWEEN '06/01/2010' AND '06/30/2010' AND e1.Voto >= 18 AND s.CorsoDiLaurea = 9 AND NOT EXISTS (SELECT 1 FROM Esami e2 JOIN Corsi c2 ON e2.Corso = c2.Id WHERE e2.Studente = s.Matricola AND c2.Denominazione = 'Interfacce Grafiche' AND e2.Data BETWEEN '06/01/2010' AND '06/30/2010' AND e2.Voto >= 18);`

Table 6.9: Comparison of SQL Queries Based on Improper Nesting of Expressions

Analysis of Improper Nesting of Expressions

Student Query: The student's query chooses students who passed `Basi Di Dati 1` in a particular range while attempting to remove those who also passed `Interfacce Grafiche` during the same period. But its design uses `EXCEPT` with numerous joins and an integrated sub-query to filter by `corsidilaurea`. Instead of directly focusing on the criteria to include or exclude, this setup layers in extra joins that don't contribute to the main goal. This results in an overly nested structure that complicates understanding and makes the query harder to follow. The query could have been simpler if it had more directly expressed the inclusion and exclusion logic without these additional layers.

ChatGPT Query: ChatGPT's approach, using `NOT EXISTS` for the exclusion criteria within the main `WHERE` clause, is generally more streamlined. However, it still includes an unrelated condition, `s.CorsoDiLaurea = 9`, which isn't relevant to the main query goal and occurs by nesting in the query.

Both times, we see a comparable problem: increasing complexity for no apparent use related to the query. This implies that when creating queries, design calls for a more direct, simpler approach. Students and artificial intelligence models should concentrate on applying just necessary criteria and eliminating superfluous joins, thereby clarifying and optimising their queries. Better query structure and readability may follow from teaching students to identify and cut out these extraneous layers. This simplified technique not only increases the accuracy of queries but also helps them to more closely relate with the current work.

6.1.10 Unnecessary Complication

Question: List the student ID, who graduated in computer science (`informatica`) before November 2009.

Query Type	SQL Query

Expected Query	`SELECT matricola`
	`FROM studenti s JOIN corsidilaurea c ON`
	`s.corsodilaurea = c.id`
	`WHERE laurea < '2009-11-01'`
	`AND denominazione = 'Informatica';`

Student Query	`SELECT DISTINCT Studenti.Matricola`
	`FROM Studenti, Esami, Corsi`
	`WHERE Studenti.Matricola = Esami.Studente`
	`AND Esami.Corso = Corsi.Id`
	`AND Corsi.CorsoDiLaurea = (`
	`SELECT id`
	`FROM CorsiDiLaurea`
	`WHERE Denominazione = 'Informatica'`
	`)`
	`AND Studenti.Laurea < '2009-11-01';`

ChatGPT Query	`SELECT Matricola`
	`FROM Studenti`
	`WHERE CorsoDiLaurea IN (`
	`SELECT id`
	`FROM CorsiDiLaurea`
	`WHERE Denominazione = 'Informatica'`
	`)`
	`AND Laurea < '2009-11-01';`

Table 6.10: Comparison of SQL Queries Based on Unnecessary Complication

Analysis of Unnecessary Complication

Student Query: The student's query introduces an unnecessary level of complexity by including multiple joins and an additional sub-query. In particular, the student's use of tables such as `Esami` and `Corsi` in the query adds layers that do not contribute to the main purpose, which is simply to find students in a particular degree program `(Informatica)`

who graduated before a certain date. By joining the `Esami` and `Corsi` tables and using a sub-query to determine the `course ID`, the student has complicated the query beyond what's necessary. This overcomplication demonstrates the misconception that more tables or joins will produce a more accurate result, when in fact they only increase the potential for confusion and logical errors.

ChatGPT Query: Even if ChatGPT's query is simpler than the student's, it still has a useless sub-query. It uses `CorsoDiLaurea IN (SELECT id FROM CorsiDiLaurea WHERE Denominazione = "Informatica")`. This confuses things more than necessary. Simple joins using `CorsiDiLaurea` would have produced the same results more quickly. Although it still lacks the directness that would make it ideal, the structure of the question is better than that of the student.

Both queries show a tendency to overuse subqueries and joins when a simpler solution would suffice. This unnecessary complications could stem from the belief that SQL queries require nested structures for accuracy, which isn't always the case. Focusing on important joins and criteria helps to make the query more efficient, and simpler SQL statements follow. Teaching students when a simpler query structure is sufficient can help to avoid this type of complexity and improve query speed and readability. For ChatGPT, a better understanding of task-specific requirements could lead to more direct solutions free from unnecessary frameworks. As a possible support measure for students, we could think in explicitly stating whether a query requires subqueries or how many tables in total need to be accessed.

6.2 Interpretation Based on Research Questions

What types of errors can generative artificial intelligence systems typically produce when generating SQL queries?

Based on our results, generative AI systems often produce SQL problems that fall into a few main categories: projection errors, unnecessary joins, missing essential columns, overuse of `distinct`, and overly complicated query structures. The generative AI solutions often result in duplicate or pointless components in the query, causing these errors. The generative AI may therefore add complications or ambiguity, perhaps by offering non-standard syntax or contextually incorrect keywords. In addition, sometimes AI-generated queries lack the necessary structural alignment or filtering with the original query purpose, resulting in inadequate responses.

How do these errors relate to common student misconceptions about learning SQL?

Artificial intelligence errors might mirror typical SQL student errors like the motivation

to add pointless columns, joins, or duplicate subqueries driven on by the belief that more data or complexity guarantees improved query accuracy. Particularly with clauses like tt NOT IN, both artificial intelligence models and students replace tt DISTINCT for SQL's natural filtering capacity. These common mistakes draw attention to the belief that logical coherence is subordinated to grammatical accuracy, therefore resulting in too complicated and ineffective queries. Particularly in multiple-join or nested queries, generative AI models such as ChatGPT-4o struggle with more general, context-specific semantics and logical connections even when their training on error-free material usually helps them to avoid syntactic errors.

In two main respects, generative artificial intelligence may support SQL teaching. First of all, it offers precise examples to pupils and proper answers for syntactic problems. Second, it may be a diagnostic tool, quickly spotting typical problems that cross over with student errors, therefore lowering the requirement for hand-based query correction. The parallel challenges faced by both groups underline the shared difficulty in mastering SQL's logical foundations. These insights emphasise the importance of developing logical reasoning in SQL education and refining AI training to address complex, real-world query scenarios, ensuring both students and AI models can construct precise and effective SQL queries.

6.3 Limitations of the Study

This research has some limits that need to be acknowledged. First of all, especially in relation to student queries, the sample size was somewhat limited, therefore maybe not entirely reflecting the spectrum of SQL learning difficulties experienced by a larger community. Small sample sizes may limit the extendability of the results and complicate the process of getting solid conclusions on the larger population of students [Pil23].

Second, the generative AI models used in this work—such as ChatGPT-4o—are trained on language models that rely heavily on the quality of their training data and prompt design. While these models excel in natural language generation, their performance in creating accurate and complex SQL queries is limited by the extent of their training and fine-tuning for database-specific tasks. This limitation suggests that the AI's ability to generate SQL queries may not fully reflect student performance but rather highlights areas where targeted prompt engineering, retrieval-augmented generation, or additional fine-tuning could enhance the model's capacity for complex SQL reasoning.

Moreover, focusing on already-existing generative artificial intelligence models restricts the study by means of the knowledge base of the AI and training data. These models might have been trained on obsolete or poor SQL patterns, therefore compromising their accuracy against human learners.

Another restriction relates to the contextual knowledge of the artificial intelligence models. Although good at syntactic creation, generative AI systems can lack the deep contextual awareness required to manage challenging SQL quiries requiring a firm knowledge of data connections. The fact that many students also suffer with SQL's abstract logic adds to this problem and makes it difficult to make unambiguous comparisons between artificial intelligence and human performance [Sel24, ZK23].

Finally, the educational setting in which the generative AI models and students were evaluated could not be exactly like practical uses. The study does not fully include variations in curriculum contents, instructional styles, or student backgrounds—which can influence the nature and frequency of errors. This contextual gap could limit the application of the findings to various learning environments [Dav24].

Chapter 7

Conclusions

7.1 Summary of Research

This thesis examined how errors produced by artificial intelligence matched student misconceptions about SQL query creation. Analysing human learners in addition to artificial intelligence models like ChatGPT-4o and ChatGPT-4o-mini assisted the research to identify common themes in SQL mistakes, most crucially in the domains of syntax, semantics, complication, and logic. With an eye towards difficult situations requiring many joins and sub-queries, the research sought to show where both artificial intelligence and students struggle with SQL queries. The results highlight the limits of depending only on generative AI for teaching such a complex topic as well as providing ideas on how artificial intelligence may support SQL instruction.

7.2 Key Findings

The key findings of this thesis highlight significant parallels between the SQL misconceptions that students often encounter and those generated by ChatGPT. The most prevalent issues, including projection errors, unnecessary joins, and missing columns in `SELECT` clauses, suggest a shared difficulty in understanding when to include only essential elements for the query's intended outcome. Indicating a tendency to add components in an effort to cover all possibilities rather than simplify for speed and clarity, both students and AI may fall into the trap of overcomplicating queries with redundant joins or `DISTINCT` statements.

One very striking result is that while organising queries, ChatGPT and students both show comparable logical mistakes, suggesting that generative AI might act as a diagnostic

mirror to point out areas where students struggle conceptually. For instance, projection errors where extraneous columns are included demonstrate that both students and AI models benefit from explicit guidance on selecting only relevant data. The AI's use of nonstandard syntax in certain contexts mirrors student struggles with SQL's syntactical agreement, which points to a broader need for augmenting the importance of standardised syntax in SQL instruction.

These repeating trends mention that ChatGPT, with its constraints, has promise as a tool in instructional environments for spotting and fixing typical SQL mistakes. By using AI-generated errors as learning points, educators can anticipate and preemptively address specific SQL misconceptions, creating more effective and targeted instructional materials. Furthermore, the results of this research reveal that focused training may enable artificial intelligence models as well as students to produce SQL queries that are more precisely linked with the objectives of database querying.

7.3 Implications for SQL Education

Studying SQL misconceptions and the difficulties students encounter reveals several areas where SQL teaching needs to be improved [HW20] and [TS20]. Both syntactic knowledge and a deeper awareness of SQL logic, complications, and semantics need to be given top priority. Although many students struggle more with the logical and syntactic elements of SQL, such as joins and nested queries, traditional training approaches generally emphasise syntax, and this was also mentioned in [ASR10] and [FS15].

Studies have shown that combining theoretical knowledge with useful applications clarifies for students how SQL works in real-world situations [AGS20]. As emphasised in many educational methods, project-based learning and hands-on experience help students avoid common misconceptions by improving their ability to use SQL in real-world contexts [Pan24].

Virtual instructors could help improve learning by identifying and correcting errors in real time [MGJ+24]. Artificial intelligence-based feedback systems provide another interactive learning tool. These methods allow one to overcome logical problems in creating SQL queries and learn from mistakes more quickly. In addition, peer-based learning environments—where students engage in dialogue and quizzes with each other—have been shown to help clarify ideas between students, thereby reducing logical errors.

The evolution of SQL training will be greatly enhanced by the incorporation of AI-powered tools for real-time identification and correction of logical and semantic errors. Artificial intelligence technology has the potential to assist students in identifying and correcting common errors before they become mainstream. In addition, these systems can provide

targeted feedback on areas of weakness and adapt to the individual needs of the student to personalise the learning [AS12].

Finally, by incorporating AI-powered tools and collaborative environments and by adapting SQL instruction to emphasise not only syntax but also deep logical understanding, students would be better prepared to handle the intricacies of SQL in professional situations.

As a recommendation for educators, this study demonstrated that, at the current stage, students cannot rely on generative AI only to learn SQL. Teachers should focus on teaching the basic SQL query logic since both generative AI models and students struggle with logical consistency in SQL queries. The lesson should emphasise how joins, sub-queries, and conditions interact within a query. Since SQL is best acquired by experience, teachers should provide chances where students might put theory to practice by working together on practical database challenges. Through better communication of their mental processes, collaborative learning enables students to understand SQL logic more fully.

Generative AI can be exploited as a complementing tool, e.g., in supporting students learning SQL syntax and getting immediate feedback.

7.4 Future Research Directions

Future studies should widen the field by including more student SQL queries across many experience levels. This would provide a closer understanding of how SQL knowledge develops and the particular phases in which students have difficulty. Comparing many artificial intelligence models, outside of ChatGPT-4o and ChatGPT-4o-mini, would provide a more complete picture of how SQL jobs are handled and where they may either exceed or lag behind human learners.

Another study may look at how best to use artificial intelligence in SQL instruction to dispel misconceptions. Adaptive learning systems that can both detect student and AI-based mistakes, for example, can provide customised help to let students overcome particular SQL issues. Longitudinal studies measuring how student assumptions evolve with greater artificial intelligence integration over time will ultimately serve to define the long-term efficacy of AI-assisted SQL training.

In general, even though generative AI models provide perceptive study of common SQL errors, their limitations, along with those of human learners, highlight the need for continuous SQL research and improvements.

Bibliography

[AB23] Eman A Alasadi and Carlos R Baiz. Generative AI in education and research: Opportunities, concerns, and solutions. *Journal of Chemical Education*, 100(8):2965–2971, 2023.

[ABV+16] 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*, pages 401–406. IEEE, 2016.

[AFM24] Willem Aerts, George Fletcher, and Daphne Miedema. A feasibility study on automated sql exercise generation with chatgpt-3.5. In *Proceedings of the 3rd International Workshop on Data Systems Education: Bridging education practice with education research*, pages 13–19, 2024.

[AGS20] Kenneth Li-Minn Ang, Feng Lu Ge, and Kah Phooi Seng. Big educational data & analytics: Survey, architecture and challenges. *IEEE access*, 8:116392–116414, 2020.

[AOP+04] Vincent Aleven, Amy Ogan, Octav Popescu, Cristen Torrey, and Kenneth Koedinger. Evaluating the effectiveness of a tutorial dialogue system for self-explanation. In *Intelligent Tutoring Systems: 7th International Conference, ITS 2004, Maceió, Alagoas, Brazil, August 30-September 3, 2004. Proceedings 7*, pages 443–454. Springer, 2004.

[APBL16] Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. Students' semantic mistakes in writing seven different types of SQL queries. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 272–277, 2016.

[AS12] Huda Al-Shuaily. Analyzing the influence of SQL teaching and learning methods and approaches. In *10th International Workshop on the Teaching, Learning and Assessment of Databases*, volume 3, 2012.

[AS13] Huda Al-Shuaily. *SQL pattern design, development & evaluation of its efficacy*. PhD thesis, University of Glasgow, 2013.

[ASR10] Huda Al-Shuaily and Karen Renaud. SQL patterns-a new approach for teaching SQL. In *8th HEA Workshop on Teaching, Learning and Assessment of Databases, Abertay-Dundee*, pages 29–40, 2010.

[ASR14] Huda Al-Shuaily and Karen Renaud. SQL Pattern Design and Development. *submitted for review*, 2014.

[BBPP+24] Antonio Balderas, Rubén Baena-Pérez, Tatiana Person, José Miguel Mota, and Iván Ruiz-Rube. Chatbot-based learning platform for SQL training. *International Journal of Interactive Multimedia and Artificial Intelligence*, 2024.

[BG06] Stefan Brass and Christian Goldberg. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software*, 79(5):630–644, 2006.

[BKS14] Gautam Biswas, John S Kinnebrew, and James R Segedy. Using a cognitive/metacognitive task model to analyze students learning behaviors. In *Foundations of Augmented Cognition. Advancing Human Performance and Decision-Making through Adaptive Systems: 8th International Conference, AC 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings 8*, pages 190–201. Springer, 2014.

[CDRFM18] Luca Cagliero, Luigi De Russis, Laura Farinetti, and Teodoro Montanaro. Improving the effectiveness of SQL learning practice: a data-driven approach. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 980–989. IEEE, 2018.

[CFFM24] Luca Cagliero, Laura Farinetti, Jacopo Fior, and Andrea Ignazio Manenti. ChatGPT, be my teaching assistant! Automatic Correction of SQL Exercises. In *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 81–87. IEEE, 2024.

[CRA91] Albert T Corbett and John R Anderson. Feedback control and learning to program with the CMU LISP tutor. In *Feedback control and learning to program with the CMU LISP tutor*. Carnegie Mellon University, 1991.

[Dav24] Adrian John Davis. AI rising in higher education: opportunities, risks and limitations. *Asian Education and Development Studies*, 2024.

[DLN+06] Jason Dagit, Joseph Lawrance, Christoph Neumann, Margaret Burnett, Ronald Metoyer, and Sam Adams. Using cognitive dimensions: advice from the trenches. *Journal of Visual Languages & Computing*, 17(4):302–327, 2006.

[DZG+23] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. C3: Zero-shot text-to-SQL with chatGPT. *arXiv preprint arXiv:2307.07306*, 2023.

[FS15] Marin Fotache and Catalin Strimbei. SQL and data analysis. Some implications for data analysits and higher education. *Procedia Economics and Finance*, 20:243–251, 2015.

[GM15] Philip Garner and John Mariani. Learning SQL in steps. *Learning*, 12:23, 2015.

[GRGDNT+24] Frank Guerra-Reyes, Eric Guerra-Dávila, Miguel Naranjo-Toro, Andrea Basantes-Andrade, and Sandra Guevara-Betancourt. Misconceptions in the Learning of Natural Sciences: A Systematic Review. *Education Sciences*, 14(5):497, 2024.

[HH16] Aarij Mahmood Hussaan and Farhan Muhammad Hassan. Learning by teaching SQL Queries to Teachable Agent Using Meta-cognitive techniques. *ASIAN JOURNAL OF ENGINEERING SCIENCES and TECHNOLOGY*, 2016.

[HW20] Zahra Hatami and Peter Wolcott. Understanding Students' Identification and Use of Patterns While Writing SQL Queries. In *Proceedings of the 21st Annual Conference on Information Technology Education*, pages 20–25, 2020.

[Juh13] Sorva Juha. Notional machines and introductory programming education. *ACM Trans. Comput. Educ*, 13(2):1–31, 2013.

[KAHM97] Kenneth R Koedinger, John R Anderson, William H Hadley, and Mary A Mark. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8:30–43, 1997.

[KPÇ24] Yunus Kökver, Hüseyin Miraç Pektaş, and Harun Çelik. Artificial intelligence applications in education: Natural language processing in detecting misconceptions. *Education and Information Technologies*, pages 1–32, 2024.

[KRM+24] Harsh Kumar, Mohi Reza, Jeb Mitchell, Ilya Musabirov, Lisa Zhang, and Michael Liut. Understanding Help-Seeking Behavior of Students Using LLMs vs. Web Search for Writing SQL Queries. *arXiv preprint arXiv:2408.08401*, 2024.

[MAF22] Daphne Miedema, Efthimia Aivaloglou, and George Fletcher. Identifying SQL misconceptions of novices: Findings from a think-aloud study. *ACM Inroads*, 13(1):52–65, 2022.

[MGJ+24] Chandra Maddila, Negar Ghorbani, Kosay Jabre, Vijayaraghavan Murali, Edwin Kim, Parth Thakkar, Nikolay Pavlovich Laptev, Olivia Harman, Diana Hsu, Rui Abreu, et al. AI-Assisted SQL Authoring at Industry Scale. *arXiv preprint arXiv:2407.13280*, 2024.

[MHLD24] Matija Mikac, Miroslav Horvatić, Robert Logožar, and Emil Dumić. ChatGPT in Education-Use Cases in an Introductory Web Programming Course. In *INTED2024 Proceedings*, pages 3173–3182. IATED, 2024.

[Mit98] Antonija Mitrovic. Learning SQL with a computerized tutor. In *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 307–311, 1998.

[Mit03] Antonija Mitrovic. An intelligent SQL tutor on the web. *International Journal of Artificial Intelligence in Education*, 13(2-4):173–197, 2003.

[MM24] Khadija MAJHADI and Mustapha MACHKOUR. Chat-SQL: Natural Language text to SQL Queries based on Deep Learning Techniques. *Journal of Theoretical and Applied Information Technology*, 102(12), 2024.

[MWL24] Erik Marx, Clemens Witt, and Thiemo Leonhardt. Identifying Secondary School Students' Misconceptions about Machine Learning: An Interview Study. In *Proceedings of the 19th WiPSCE Conference on Primary and Secondary Computing Education Research*, pages 1–10, 2024.

[NGF+24] Eduardo R Nascimento, Grettel M Garcıa, Lucas Feijó, Wendy Z Victorio, Yenier T Izquierdo, Aiko R de Oliveira, Gustavo MC Coelho, Melissa Lemos, Robinson LS Garcia, Luiz AP Paes Leme, et al. Text-to-SQL Meets the Real-World. In *26th Int. Conf. on Enterprise Info. Sys*, 2024.

[Ohl94] Stellan Ohlsson. Constraint-based student modeling. In *Student modelling: the key to individualized knowledge-based instruction*, pages 167–189. Springer, 1994.

[OLGJ+23] Anne Ottenbreit-Leftwich, Krista Glazewski, Minji Jeon, Katie Jantaraweragul, Cindy E Hmelo-Silver, Adam Scribner, Seung Lee, Bradford Mott, and James Lester. Lessons learned for AI education with elementary students and teachers. *International Journal of Artificial Intelligence in Education*, 33(2):267–289, 2023.

[Pan24] Vijay Panwar. AI-Driven Query Optimization: Revolutionizing Database Performance and Efficiency. *arXiv preprint arXiv:2407.13280*, 2024.

[PBAH20] Seth Poulsen, Liia Butler, Abdussalam Alawini, and Geoffrey L Herman. Insights from student solutions to SQL homework problems. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pages 404–410, 2020.

[Pil23] Giuseppe Carmelo Pillera. In dialogue with ChatGPT on the potential and limitations of AI for evaluation in education. *PEDAGOGIA OGGI*, 21(1):301–315, 2023.

[PM+01] John F Pane, Brad A Myers, et al. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, 54(2):237–264, 2001.

[PMHS21] Kai Presler-Marshall, Sarah Heckman, and Kathryn Stolee. SQLRepair: Identifying and repairing mistakes in student-authored SQL queries. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 199–210. IEEE, 2021.

[PRH+24] Kishore Prakash, Shashwat Rao, Rayan Hamza, Jack Lukich, Vatsal Chaudhari, and Arnab Nandi. Integrating LLMs into Database Systems Education. In *Proceedings of the 3rd International Workshop on Data Systems Education: Bridging education practice with education research*, pages 33–39, 2024.

[RB13] Alexander Hoem Rosbach and Anya Helene Bagge. Classifying and measuring student problems and misconceptions. *Akademika forlag*, 2013.

[Sel24] Neil Selwyn. On the limits of artificial intelligence (AI) in education. *Nordisk tidsskrift for pedagogikk og kritikk*, 10(1):3–14, 2024.

[SY23] Jiahong Su and Weipeng Yang. Unlocking the power of ChatGPT: A framework for applying generative AI in education. *ECNU Review of Education*, 6(3):355–366, 2023.

[Tai20] Toni Taipalus. Explaining causes behind SQL query formulation errors. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. IEEE, 2020.

[Tia23] Yang Tian. Limitations and Advancements of AI Teachers in Classroom Instruction. *Education and Teaching Research*, 3:66–72, 2023.

[Tra24] Daniele Traversaro. *Insegnare SQL a Chi Non Ha Mai Programmato: Analisi delle Misconcenzioni*. ITADINFO, 2024.

[TS20] Toni Taipalus and Ville Seppänen. SQL education: A systematic mapping study and future research agenda. *ACM Transactions on Computing Education (TOCE)*, 20(3):1–33, 2020.

[TSV18] Toni Taipalus, Mikko Siponen, and Tero Vartiainen. Errors and complications in SQL query formulation. *ACM Transactions on Computing Education (TOCE)*, 18(3):1–29, 2018.

[WS81] Charles Welty and David W Stemple. Human factors comparison of a procedural and a nonprocedural query language. *ACM Transactions on Database Systems (TODS)*, 6(4):626–649, 1981.

[ZK23] Zuhair Dawood Zaghlool and Mohamad Ahmad Saleem Khasawneh. Incorporating the Impacts and Limitations of AI-Driven Feedback, Evaluation, and Real-Time Conversation Tools in Foreign Language Learning. *Migration Letters*, 20(7):1071–1083, 2023.

Appendix A

Query Complexity

Source	Query	SQL Query	Complexity
Database Exam Model (GIOCATORE)	The French players who have never been TestaDiSerie.	SELECT DISTINCT IdG FROM GIOCATORE WHERE Nazione = 'Francia' AND IdG NOT IN (SELECT DISTINCT IdG FROM GIOCAIN NATURAL JOIN REGISTRAZIONE WHERE TestaDiSerie)	Medium
Database Exam Model (GIOCATORE)	German players who have never participated in Wimbledon.	SELECT DISTINCT IdG FROM GIOCATORE WHERE Nazione = 'Germania' AND IdG NOT IN (SELECT DISTINCT IdG FROM GIOCAIN NATURAL JOIN TORNEO WHERE NomeT = 'Wimbledon')	Medium
Database Exam Model (GIOCATORE)	Players who participated in both the singles and doubles categories in the same tournament.	SELECT DISTINCT IdG FROM GIOCATORE NATURAL JOIN GIOCAIN NATURAL JOIN CATEGORIA WHERE Genere = 'f' AND NomeCategoria = 'singolo' AND (IdG, IdT) IN (SELECT DISTINCT IdG, IdT FROM GIOCAIN NATURAL JOIN CATEGORIA WHERE NomeCategoria = 'doppio')	Hard

Source	Query	SQL Query	Complexity
Computer Science Database Laboratories Model	List the student ID and names of students enrolled before 2007/2008 who are not yet in the thesis phase.	SELECT matricola, nome, cognome FROM studenti WHERE iscrizione <2007 AND relatore IS NULL	Simple
Computer Science Database Laboratories Model	List the student ID of students who graduated in computer science before November 2009.	SELECT matricola FROM studenti s JOIN corsidilaurea c ON s.corsodilaurea = c.id WHERE laurea <'2009-11-01' AND denominazione = 'Informatica';	Simple
Computer Science Database Laboratories Model	List degree courses, in alphabetical order by faculty and course name, activated before 2006/2007 and after 2009/2010.	SELECT facolta, denominazione FROM corsidilaurea WHERE attivazione >'2006/2007'OR attivazione >'2009/2010'ORDER BY facolta, denominazione;	Simple
Engineering Database Laboratories Model	Select the names of products in the beverages category that were not ordered in 2023.	SELECt idProd, P.nome FROM Prodotto P JOIN Categoria C ON P.idCat = C.idCat WHERE C.nome = 'Bevande' AND idProd NOT IN (SELECT idProd FROM Ordine O JOIN DettaglioOrdine D ON O.idOrd = D.idOrd WHERE EXTRACT (YEAR FROM data) = '2023')	Hard
Engineering Database Laboratories Model	Select the total amount of the last order made by the customer with ID "1234".	SELECT O.idOrd, SUM (prezzo*d.quantità) AS TotaleOrdine FROM Ordine O JOIN DettaglioOrdine D ON O.idOrd=D.idOrd JOIN Prodotto P ON D.idProd = P.idProd WHERE O.idClient = '1234' AND O.data = (SELECT MAX(data) FROM Ordine WHERE idClient='1234') GROUP BY O.idOrd	Hard

Source	Query	SQL Query	Complexity
Engineering Database Laboratories Model	Select the names of the products that have never been ordered by customers residing in Venezia or Brescia.	SELECT idProd, nome FROM Prodotto P WHERE NOT EXISTS(SELECT nome FROM Cliente C JOIN Ordine O ON C.idClient= O.idClient JOIN DettaglioOrdine D ON P.idProd=D.idProd WHERE C.città IN ('Venezia', 'Brescia'))	Hard
Miedema Thesis Database Model	List all IDs and names of customers living in Eindhoven.	SELECT cID, cName FROM customer WHERE city = 'Eindhoven';	Simple
Miedema Thesis Database Model	Return the stores table ordered alphabetically by city.	SELECT city, COUNT(sID) AS num-stores FROM store GROUP BY city;	Simple
Database Exam Model (GIOCATORE)	Italian players who participated in both the US Open and the Australian Open in the same year.	SELECT DISTINCT IdG FROM GIOCATORE NATURAL JOIN GIOCAIN NATURAL JOIN TORNEO WHERE Nazione = 'Italia' AND NomeT = 'US Open' AND (IdG,YEAR(DataNascita)) IN (SELECT DISTINCT IdG, YEAR(DataNasdcita) FROM GIOCAIN NATURAL JOIN TORNEO WHERE NomeT = 'Australian Open')	Hard
Database Exam Model (GIOCATORE)	The tournament with the highest number of TestaDiSerie.	SELECT DISTINCT IdT FROM REGISTRAZIONE WHERE TestaDiSerie GROUP BY IdT HAVING COUNT(*) >= ALL (SELECT COUNT(*) FROM REGISTRAZIONE WHERE TestaDiSerie GROUP BY IdT)	Medium

Source	Query	SQL Query	Complexity
Database Exam Model (GIOCATORE)	The player who has played the most different tournaments.	SELECT DISTINCT IdG FROM GIOCATORE NATURAL JOIN GIOCAIN NATURAL JOIN CATEGORIA WHERE Genere = 'f'The player who has AND NomeCategoria = 'singolo'The player who has AND (IdG, IdT) IN (SELECT DISTINCT IdG, IdT FROM GIOCAIN NATURAL JOIN CATEGORIA WHERE NomeCategoria = 'doppio')	Hard
Computer Science Database Laboratories Model	List the last name, first name, and status ('studente'/' professore') of students and professors.	SELECT nome, cognome, 'studente' AS qualifica FROM studenti UNION ALL SELECT nome, cognome, 'professore' AS qualifica FROM professori;	Medium
Computer Science Database Laboratories Model	List computer science students who passed Databases 1 but not Graphic Interfaces in June 2010.	SELECT studente FROM esami WHERE corso = 'bdd1n' AND voto >= 18 AND esami.data BETWEEN '2010-06-01' AND '2010-06-30' EXCEPT SELECT studente FROM esami WHERE corso = 'ig' AND voto >= 18 AND esami.data BETWEEN '2010-06-01 ' AND '2010-06-30';	Hard
Computer Science Database Laboratories Model	List computer science students who passed both Databases 1 and Graphic Interfaces in June 2010.	SELECT studente FROM esami WHERE corso = 'bdd1n' AND voto >= 18 AND esami.data BETWEEN '2010-06-01' AND '2010-06-30' INTERSECT SELECT studente FROM esami WHERE corso = 'ig' AND voto >= 18 AND esami.data BETWEEN '2010-06-01' AND '2010-06-30';	Hard
Miedema Thesis Database Model	Return a list of the number of stores per city.	SELECT city, COUNT(sID) AS num-stores FROM store GROUP BY city;	Simple

Source	Query	SQL Query	Complexity
Miedema Thesis Database Model	List all pairs of customer IDs who live on a street with the same name but in different cities.	SELECT c1.cid AS id1, c2.cid AS id2 FROM customer c1 JOIN customer c2 ON c1.street = c2.street AND c1.city != c2.city WHERE c1.cID <c2.cID;	Medium
Database Exam Model (GIOCATORE)	The tournament with the highest average age of players (also considered correct current age).	SELECT DISTINCT IdT FROM TORNEO NATURAL JOIN GIOCAIN NATURAL JOIN GIOCATORE GROUP BY IdT HAVING AVG(DataI-DataN) >= ALL (SELECT AVG(DataI-DataN) FROM TORNEO NATURAL JOIN GIOCAIN NATURAL JOIN GIOCATORE GROUP BY IdT)	Hard
Database Exam Model (GIOCATORE)	The tournament in which players from the greatest number of different nations participated.	SELECT DISTINCT IdT FROM GIOCAIN NATURAL JOIN GIOCATORE GROUP BY IdT HAVING COUNT(DISTINCT Nazione) >= ALL (SELECT COUNT(DISTINCT Nazione) FROM GIOCAIN NATURAL JOIN GIOCATORE GROUP BY IdT)	Hard
Computer Science Database Laboratories Model	Return an alphabetical list of student names, with the last name of their associated advisor for each one.	SELECT s.cognome, s.nome, p.cognome FROM studenti s JOIN professori p ON s.relatore = p.id ORDER BY s.cognome, s.nome;	Medium

Source	Query	SQL Query	Complexity
Computer Science Database Laboratories Model	Return the list, without duplicates and in reverse alphabetical order, of students who submitted the study plan for the fifth year of the computer science degree course in 2011/2012 and are in the thesis phase.	SELECT s.cognome, s.nome FROM studenti s JOIN pianidistudio p on s.matricola = p.studente WHERE p.anno = 5 AND p.annoaccademico = 2011 AND s.relatore IS NOT NULL ORDER BY s.cognome DESC, s.nome DESC;	Hard
Miedema Thesis Database Model	List all customer IDs, dates, and quantities of transactions containing products named Apples.	SELECT t.cID, t.date, t.quantity FROM transaction t JOIN product p ON t.pID = p.pID WHERE p.pName = 'Apples';	Hard
Miedema Thesis Database Model	Retrieve store-chains with multiple branches within the same city but different addresses.	SELECT s.sName FROM store s JOIN transaction t ON s.sID = t.sID GROUP BY s.sName HAVING COUNT(DISTINCT s.sID) >= 2 AND AVG(t.quantity) >4;	Medium

Table A.1: Query Complexity Categorization

Appendix B

Misconceptions Categories Excel File

AnswerID	QueryID	Written by	Query specification	SQL code (answer)	Execution re:	Categorization	Notes	Checked by	Miscon
1	Miedema_4-2	ChatGP...	List all IDs&names of customers living in Eindhoven.	SELECT c.cID, c.cName FROM customer c JOIN store s ON c.city = s.ci WHERE s.city = 'Eindhoven';	Can ...	COM-1 Complication...	returns wrong	Davide	COM
2	Miedema_4-2	ChatGP...	List all IDs&names of customers living in Eindhoven.	SELECT cID, cName FROM customer WHERE city = 'Eindhoven';	Can ...	EXEMPLARY - Quer...		Davide	EXE
3	Miedema_4-2	ChatGP...	List all IDs&names of customers living in Eindhoven.	SELECT c.cID, c.cName FROM customer c JOIN store s ON c.city = s.ci WHERE s.city = 'Eindhoven';	Can ...	COM-1 Complication...	returns wrong	Davide	COM
4	Miedema_4-2	ChatGP...	List all IDs&names of customers living in Eindhoven.	SELECT cID, cName FROM miedema.customer WHERE city = 'Eindhoven';	Can ...	EXEMPLARY - Quer...		Davide	EXE
5	Miedema_4-2	ChatGP...	List all IDs&names of customers living in Eindhoven.	SELECT cID, cName FROM miedema.customer WHERE city = 'Eindhoven';	Can ...	EXEMPLARY - Quer...		Davide	EXE
6	Miedema_4-2	ChatGP...	List all IDs&names of customers living in Eindhoven.	SELECT cID, cName FROM miedema.customer WHERE city = 'Eindhoven';	Can ...	EXEMPLARY - Quer...		Davide	EXE
7	Miedema_4-3	ChatGP...	List all pairs of customer IDs who live on a street with the same name but in a different city.	SELECT DISTINCT c1.cID, c2.cI FROM customer c1 JOIN customer c2 ON c1.street	Can ...	SEM-4 Duplicate row...		Hamid	SEM
7	Miedema_4-3	ChatGP...			Can ...	SYN-6 Common synt...		Hamid	SYN
7	Miedema_4-3	ChatGP...			Can ...	COM-1 Complication...		Hamid	COM
77	Miedema_4-9	Student ...	A store-chain consists of at least two stores with the same name but different IDs. Find the names of the store-chains that on average sell product in quantities of more than 4.	SELECT Store.sName, AVG(quant: INNER JOIN Store USING(sID) INNER JOIN (SELECT sName, COUI GROUP BY Store.sName HAVING A'	Can ...	LOG-5 Projection err...		Hamid	LOG
			List all pairs of customer IDs who live on a street	SELECT a.cID AS Customer1_ID, FROM miedema.customer AS a JOIN miedema.customer AS b ON WHERE a.cID < b.cID;					

Figure B.1: Misconceptions Categories Excel Sample 1

AnswerID	QueryID	Written by	Query specification	SQL code (answer)	Execution re	Categorization	Notes	Checked by	Miscon
			inventory items that have a higher unit price than	JOIN product p2 ON p2.pName = JOIN inventory i2 ON p2.pID =					
18	Miedema_4-5	ChatGP...	Bananas	WHERE i1.unit_price > i2.unit_	Can ...	COM-1 Complication...	Too many join	Hamid	COM
18	Miedema_4-5	ChatGP...			Can ...	COM-1 Complication...	The query is c	Hamid	COM
18	Miedema_4-5	ChatGP...			Can ...	LOG-4 Expression er...		Hamid	LOG
				SELECT p_other.pName AS highe FROM inventory i_banana JOIN product p_banana ON i_ba JOIN inventory i_other ON i_b JOIN product p_other ON i_oth WHERE p_banana.pName = 'Banan AND i_other.unit_price > i_ba					
19	Miedema_4-5	ChatGP...	Find the names of all inventory items that have a higher unit price than Bananas		Can ...	COM-1 Complication...		Hamid	COM
19	Miedema_4-5	ChatGP...			Can ...	COM-1 Complication...		Hamid	COM
19	Miedema_4-5	ChatGP...			Can ...	LOG-4 Expression er...		Hamid	LOG
				SELECT p.pName FROM miedema.product p JOIN miedema.inventory i ON p WHERE i.unit_price > (SELECT unit_price FROM miedema.inventory JOIN miedema.product ON m WHERE pName = 'Banana')					
20	Miedema_4-5	ChatGP...	Find the names of all inventory items that have a higher unit price than Bananas		Can ...	SEM-4 Duplicate row...	more than one	Hamid	SEM
				SELECT p.pName FROM miedema.product p JOIN miedema.inventory i ON p WHERE i.unit_price > (SELECT unit_price FROM miedema.inventory JOIN miedema.product ON m WHERE pName = 'Banana')					
21	Miedema_4-5	ChatGP...	Find the names of all inventory items that have a higher unit price than Bananas		Can ...	SEM-4 Duplicate row...	more than one	Hamid	SEM
				SELECT p.pName FROM miedema.product p JOIN miedema.inventory i ON p WHERE i.unit_price > (SELECT unit_price FROM miedema.inventory					

Figure B.2: Misconceptions Categories Excel Sample 2