A Study of SQL Programming Learning Assistant System for Novice Learners

March, 2026

Ni Wayan Wardani

Graduate School of Environmental, Life,
Natural Science and Technology

(Doctor's Course)
Okayama University

Dissertation submitted to
Graduate School of Environmental, Life, Natural Science and Technology
of
Okayama University
for
partial fulfillment of the requirements
for the degree of
Doctor of Philosophy in Engineering

Written under the supervision of

Professor Nobuo Funabiki

and co-supervised by
Professor Satoshi Denno
and
Professor Yasuyuki Nogami

Okayama University, March 2026.

To Whom It May Concern

We hereby certify that this is a typical copy of the original doctor thesis of
Ms. Ni Wayan Wardani

Signature of

the Supervisor

Seal of

Prof. Nobuo Funabiki

Graduate School of

Environmental, Life, Natural Science and Technology

Abstract

Programming education takes a crucial role in developing students' critical thinking, problem-solving, and creative abilities, which are essential skills in the modern digital era. Among various programming competencies, *database programming* has become a fundamental skill for software developers and data professionals due to its vital role in storage, retrieval, maintenance, integrity, and security of data.

Structured Query Language (SQL) is the most widely used domain-specific language for managing and querying relational databases, and its mastery is essential for extracting insights and supporting data-driven decision making across various industries. The integration of *SQL* with *Python* has strengthened data processing capabilities, as *Python* provides flexible libraries that enable efficient database access, transformation, and analysis. *Python* is now widely taught in universities and professional schools.

In this thesis, to support novice students in learning *database programming* effectively, I have developed the *SQL Database Programming Learning Assistant System (SQL-PLAS)* that takes a progressive learning strategy and emphasizes *code reading* before *code writing*. *SQL-PLAS* has implemented several structured exercise types such as *Grammar-concept Understanding Problems (GUP)*, *Comment Insertion Problems (CIP)*, and *SQL Query Description Problems (SDP)*. Through this step-by-step learning, students are guided from understanding basic grammar and program structures toward independently writing complete *SQL* queries. In this learning environment, similar code-reading-based exercises are adopted to help students understand keyword definitions and control flows before engaging in full query construction. The correctness of student answers is evaluated automatically using *string-matching* technique to ensure efficient and consistent assessment.

However, designing high-quality SQL exercises remains a time-consuming task for instructors, as it requires constructing database schemas before generating meaningful queries. This challenge becomes more significant when instructors frequently modify databases across different semesters. To address this issue, a *generative AI-assisted SQL query generator* has been developed to automatically produce diverse and practical SQL Query Description Problems (SDPs) based on real database schemas. By leveraging large language models (*LLMs*), the system generates question–answer pairs according to selected SQL topics, which are then automatically verified for syntax correctness and execution validity using a MySQL database. Teachers can further review and refine the generated content before integrating it into PLAS. This approach significantly reduces instructors' workload while maintaining pedagogical accuracy through human validation.

The first contribution of the thesis is the proposal of the *Grammar-concept Understanding Problem (GUP)* in both *SQL PLAS* and *SQL-Python PLAS*. A GUP instance consists of a source code and a set of questions focusing on the grammatical terms and concepts in the code, such as reserved words, commands, and common libraries. Students are tasked with answering questions about the meaning and usage of these keywords, with answers evaluated based on *string matching*,

providing precise feedback on their understanding of SQL grammar.

The second contribution introduces the *Comment Insertion Problem (CIP)* in *SQL-Python PLAS*. This problem type provides students with source code containing missing comments and a set of comments to be inserted in the appropriate places. The goal is to assess students' understanding of code structure and logic. Responses are evaluated using *string matching*, ensuring consistent feedback on the students' ability to interpret and understand the code.

However, designing high-quality *SQL* exercises remains a time-consuming task for instructors, as it requires constructing database schemas before generating meaningful queries. This challenge becomes more significant when instructors frequently change databases across different semesters.

The third contribution is the proposal of the *SQL Query Description Problem (SDP)* within the *SQL PLAS*. This problem type consists of a database table schema and a series of questions requiring students to write SQL queries for data retrieval or manipulation. Students' responses are evaluated automatically using *string matching* against the correct solutions, providing immediate feedback and supporting effective learning of SQL query construction.

To address this issue, a *generative AI-assisted SQL query generator* has been developed to automatically produce diverse and practical SQL Query Description Problems (SDPs) based on real database schemas. By leveraging large language models (*LLMs*), the system generates question–answer pairs according to selected SQL topics, which are then automatically verified for syntax correctness and execution validity using a MySQL database. Teachers can further review and refine the generated content before integrating it into PLAS. This approach significantly reduces instructors' workload while maintaining pedagogical accuracy through human validation.

The fourth contribution introduces the *Generative AI-assisted SQL Query Generator*, a tool that automates the generation of diverse and practical *SQL Query Description Problems (SDPs)*. This tool alleviates the burden on instructors by automating the creation of database schemas, practice questions, and correct answers, enabling more efficient and varied SQL training material creation.

In the future, we will expand the database programming topics into basic, intermediate, and advanced levels, and develop various types of problems tailored for novice students, which will be evaluated through practical applications. Additionally, instructions on how to use the system will be provided to facilitate better understanding. Another focus will be improving the *marking function* by using semantic similarity evaluation for logical assessment beyond *string matching*. Expanding the dataset and refining prompt templates based on teaching feedback will also be pursued. Integrating adaptive difficulty levels and automated error classification will enable a more personalized learning experience, with continuous quality improvement based on user feedback.

Acknowledgements

I would like to express my heartfelt gratitude to all who supported and guided me throughout the completion of this thesis at Okayama University, Japan. To all who have been part of this journey, I will just say that you are the greatest blessing in my life.

First and foremost, I owe my deepest gratitude to my honorable supervisor, Professor Nobuo Funabiki for his excellent supervision, meaningful suggestions, persistent encouragements, and other fruitful help at every stage of my Ph.D. study. His thoughtful comments and guidance helped me to complete my research papers and present them in productive ways. Besides, he was always patient and helpful whenever his guidance and assistance were needed in both of my academic and daily life in Japan. Indeed, his guidance has been nothing less than a gift. Needless to say, it would not be possible to complete this thesis without his guidance and active support.

I extend my heartfelt thanks to my Ph.D. co-supervisors, Professor Satoshi Denno and Professor Yasuyuki Nogami, for their continuous support, guidance, thoughtful suggestions, and proofreading of this thesis. I am also grateful to Assistant Professor Hto Hto Sandi Kyaw and all my course instructors for their enlightening knowledge and discussions.

I would like to acknowledge the Monbukagakusho Honors Scholarship (JASSO Scholarship) and Okayama University for financially supporting my Ph.D study.

I want to express my appreciation to the members of the Funabiki Lab at Okayama University. Ms. Keiko Kawabata, Ms. Safira Adine Kinari, Dr. Abdul Rahman Patta, Dr. Evianita Dewi Fajrianti, Ms. Irin Tri Anggraini, Dr. Radhiatul Husna, Mr. Amma Liesvarastranta Haz, Ms. Mustika Mentari, Mr. Putu Sugiartawan, Mr. I Nyoman Darma Kotama, Mr. Anak Agung Surya Pradhana and all members in general who have shared strength during our three years; Sharing time with these individuals in Okayama has provided me with great experiences and unforgettable moments. Thank you for your significant support and helpfulness in both my academic endeavours and daily life.

I also extend my gratitude to my colleagues at the Indonesian Institute of Business and Technology, Denpasar, Bali, Indonesia has been a continuous source of support throughout my study. Thank you for your invaluable assistance during this period.

My special thanks to my best friends (Agus Suarya, Dewa Ayu Giovany, Nirwana, Ayu Ningsih, Yessi, Surya Cipta, Surya Mahendra, Mila, Agus Krisna and Nurvin). Your support from before I started studying until the end of my studies means a lot to me.

Last but not least, I am eternally grateful to my beloved family, Mother, My late Father, Parents-in-law, Husband, Daughters, Brothers and Sisters and all my family members for their unconditional love, support, patience, and confidence in me, which have been my greatest motivation and reward. I am proud and blessed to have you all in my life.

Ni Wayan Wardani
Okayama University, Japan
March, 2026

List of Publications

Journal Paper

1. **Ni Wayan Wardani**, Nobuo Funabiki, Putu Sugiartawan, Anak Agung Surya Pradhana, I Nyoman Darma Kotama, and I Nyoman Agus Suarya Putra, "An Implementation of Self-study Exercise Problems for Entry-level SQL-Python Database Programming," IAENG Engineering Letters, vol. 33, no. 8, pp. 2939-2948, August 2025.

2. **Ni Wayan Wardani**, Nobuo Funabiki, Htoo Htoo Sandi Kyaw, Zhu Zhou, I Nyoman Darma Kotama, Putu Sugiartawan, and I Nyoman Agus Suarya Putra, "An SQL Query Description Problem with AI Assistance for SQL Programming Learning Assistant System," Information, vol.17, no.1, 65, January 2026.

International Conference Paper

3. **Ni Wayan Wardani**, Nobuo Funabiki, Putu Sugiartawan, and I Nyoman Agus Suarya Putra, "A Proposal of Grammar-Concept Understanding Problem for Self-Study of Introduction Basic to SQL Database Programming," 2024 Seventh International Conference on Vocational Education and Electrical Engineering (ICVEE), pp. 310-316, October 30, 2024. DOI: 10.1109/ICVEE63912.2024.10824028.

4. **Ni Wayan Wardani**, Nobuo Funabiki, Putu Sugiartawan, Soe Thandar Aung, and I Nyoman Agus Suarya Putra, "A Proposal of SQL Syntax Description Problem in SQL Programming Learning Assistant System," 2025 International Conference on Electrical Engineering and Information Systems (CEEIS), pp. 13-17, February 28 - March 2, 2025. DOI: 10.1109/CEEIS65979.2025.00011.

5. **Ni Wayan Wardani**, Nobuo Funabiki, Htoo Htoo Sandi Kyaw, Putu Sugiartawan, and I Nyoman Agus Suarya Putra, "An investigation of code modification problem for learning server-side JavaScript programming in web application system," 2025 Seventh International Conference on Vocational Education and Electrical Engineering (ICVEE), pp. 42-47, September 24, 2025. DOI: 10.1109/ICVEE66651.2025.11281402.

Oral and Poster Presentation

6. **Ni Wayan Wardani**, Nobuo Funabiki, Abdul Rahman Patta, Xiqin Lu, I Nyoman Agus Suarya Putra, "A Study of Grammar-Concept Understanding Problem for SQL Python Database

Programming Learning Assistant System," Technical Committee on Educational Technology, The Institute of Electronics, Information and Communication Engineers (IEICE), vol. 123, no. ET2023-37, pp. 7-12, December 2024.

7. **Ni Wayan Wardani**, Nobuo Funabiki, Htoo Htoo Sandi Kyaw, I Nyoman Darma Kotama, Putu Sugiartawan and I Nyoman Agus Suarya Putra, "A Comparative Study of Generative AI Models in Generating Exercises for SQL Programming Learning Assistant System," Proceedings of The Institute of Electronics, Information and Communication Engineers (IEICE), (Fukuoka,2026-3) (in printing).

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Background for SQL Study

Programming education plays a vital role in developing students' critical thinking, problem-solving abilities, and creativity. These skills empower students to explore diverse career opportunities after graduation. As a result, *computer programming* has become a core subject in universities and professional schools worldwide. Many academic institutions now offer programming courses to train the next generation of programming engineers.

Database programming languages are essential tools for managing and manipulating data within software applications. They play a crucial role in tasks such as data storage, retrieval, maintenance, migration, integrity, security, and the overall functionality of database-dependent applications. Proficiency in database programming languages is considered a fundamental skill for software developers and data professionals [1]. The growing demand for skilled database programmers has led many universities to offer dedicated database programming courses early in the curriculum.

SQL (Structured Query Language) is a powerful, domain-specific language used to manage and query relational databases in database programming [2]. As the volume and complexity of digital information continue to grow, *SQL* remains a foundational tool for ensuring data integrity, supporting analytical processes, and enabling evidence-based decision-making across industries. Its structured syntax and robust capabilities allow developers, analysts, and researchers to interact with large datasets in a consistent and reliable manner. In modern data-driven environments, mastering *SQL* is essential for extracting meaningful insights and building scalable data solutions.

To support novice students in learning database programming, the *SQL Database Programming Learning Assistant System (SQL PLAS)* has been developed. This system is designed to help students by providing a structured, step-by-step approach to mastering SQL. The approach emphasizes the importance of reading and understanding simple source code before attempting more complex tasks. Students are first encouraged to solve basic grammar concept problems to learn how to program efficiently. Once students grasp the fundamental principles and are comfortable with *code reading*, they can progress to *query writing* from scratch. To facilitate this progression, *SQL PLAS* offers a variety of exercise problems at different levels.

When learning programming, novice students should begin by solving simple, concise exercises that focus on *code reading*. This helps them understand the grammar and concepts of the programming language. After gaining a basic understanding, students can transition to *query writing* exercises. Developing strong skills in reading source code is essential, as it directly impacts their ability to write correct and effective queries.

To support the novice students' progressive programming study, *SQL PLAS* provides the following types of exercise problems. By solving these problems in this order, it is expected for the students to gradually advance their programming levels by themselves.

1. Grammar-concept Understanding Problem (GUP) gives questions about the concepts of important keywords, including reserved words and commonly used libraries in the programming language, in the provided source code. It focuses on keywords and libraries in the source code [3].

2. Comment Insertion Problem (CIP) gives CIP instance consists of source code with blank comments, a set of comments to fill in the blanks, and their correct answers. it focuses to understand the structures and logics of source codes [4].

3. SQL Query Description Problem (SDP) it is designed to aid novice students in comprehending the fundamental query to data manipulation and retrieve of SQL database programming for executing a range of queries. An SDP instance comprises a collection of questions and a database table.

1.2 Background for SQL-Python Study

As mentioned earlier, *SQL* is a powerful, domain-specific language designed for managing and querying relational databases in database programming [2]. On the other hand, Python is a versatile, high-level programming language renowned for its simplicity and extensive library ecosystem [5]. When combined, these two technologies provide a comprehensive solution for data handling. Python libraries, such as *sqlite3*, allow seamless integration with *SQL* databases, enabling developers to interact with data using familiar Python syntax. This synergy empowers developers to efficiently manipulate and retrieve data from databases, perform complex data transformations, and build data-driven applications with ease. Whether working as a data scientist, software engineer, or database administrator, the combination of *SQL* and Python offers a versatile and powerful toolkit for managing and extracting insights from data [6]. As a result, both *SQL* and Python programming are widely taught in universities and professional schools to equip students with essential skills for data management and analysis.

To assist novice students in progressing through their programming studies, *SQL-Python PLAS* provides *GUP* and *CIP* exercise problems that focus on *code reading*. These problems help students understand the definitions of keywords and control flows in source codes, laying the foundation for more advanced tasks.

For each exercise, the correctness of a student's answer is verified automatically. The system uses *string matching* to compare student answers with the correct solutions for *GUP*, *CIP*, *SSDP*, and *SDP*. In the database PLAS framework, students first tackle *code reading* problems to understand the definitions of keywords and the flow of control in the source code. Once students are comfortable with these concepts, they progress to solving *query writing* problems, allowing them to write full SQL queries independently.

1.3 Background for Generative AI-assisted SQL Query Generator

Generative AI has its roots in the early developments of artificial neural networks, including key

milestones such as the introduction of artificial neurons in the 1950s and the back-propagation algorithm in the 1980s. A significant breakthrough occurred in 2017 with the advent of the transformer architecture, which led to the rise of large language models (*LLMs*) like *ChatGPT*. These models excel in natural language generation and have revolutionized various industries, including education. The continuous evolution of *Generative AI* has opened up innovative opportunities for content creation and automation, with an increasing number of researchers and businesses applying *LLMs* in educational contexts.

The *Generative AI-assisted SQL query generator* is designed to assist teachers in generating diverse and practical *SQL Query Description Problems (SDPs)*. This tool utilizes *generative AI* models to automatically create question-answer pairs based on real database schemas. Teachers can easily interact with the generator via a simple graphical interface. By selecting a desired *SQL* topic, such as `SELECT`, `UPDATE`, or `JOIN`, and specifying the *generative AI* model to be used, the generator produces a set of corresponding *SQL* exercises that include both the question statements and the correct *SQL* queries.

Once teachers start the generator, it prepares prompts including the schema information and topic instructions. The generated results of *question-answer pairs* are automatically checked in syntax correctness and running ability on the target *MySQL* database system. Only the pairs that are executed successfully are saved for teacher validations. Then, teachers should review, edit, or discard generated contents before uploading them to *PLAS*. This workflow allows for efficient and semi-automated content creations while maintaining human oversights on accuracy and pedagogical consistency.

To assess whether students understand *SQL* queries syntactically, teachers create homework, assignments, and final exams. In the case of *SQL*, the process involves creating a database schema before the actual queries can be generated. This design is a time-consuming task, especially if teachers intend to vary their database significantly between course iterations. Teachers often spend more time creating stories and designing database schemas than creating the questions themselves [7]. This method takes a lot of effort, particularly if instructors want to significantly change their databases between course iterations [7].

1.4 Contributions

This thesis presents the four implementations for the improvement of Database Programming Learning Assistant Systems.

The first contribution of the thesis is the proposal of the *Grammar-concept Understanding Problem (GUP)* in both *SQL PLAS* and *SQL-Python PLAS*. A GUP instance consists of a source code and a set of questions focusing on the grammatical terms and concepts in the code, such as reserved words, commands, and common libraries. Students are tasked with answering questions about the meaning and usage of these keywords, with answers evaluated based on *string matching*, providing precise feedback on their understanding of SQL grammar.

The second contribution is introduces the *Comment Insertion Problem (CIP)* in *SQL-Python PLAS*. This problem type provides students with source code containing missing comments and a set of comments to be inserted in the appropriate places. The goal is to assess students' understanding of code structure and logic. Responses are evaluated using *string matching*, ensuring consistent feedback on the students' ability to interpret and understand the code.

However, designing high-quality *SQL* exercises remains a time-consuming task for instructors, as it requires constructing database schemas before generating meaningful queries. This challenge

becomes more significant when instructors frequently change databases across different semesters.

The third contribution is the proposal of the *SQL Query Description Problem (SDP)* within the *SQL PLAS*. This problem type consists of a database table schema and a series of questions requiring students to write SQL queries for data retrieval or manipulation. Students' responses are evaluated automatically using *string matching* against the correct solutions, providing immediate feedback and supporting effective learning of SQL query construction.

To address this issue, a *generative AI-assisted SQL query generator* has been developed to automatically produce diverse and practical SQL Query Description Problems (SDPs) based on real database schemas. By leveraging large language models (*LLMs*), the system generates question–answer pairs according to selected SQL topics, which are then automatically verified for syntax correctness and execution validity using a MySQL database. Teachers can further review and refine the generated content before integrating it into PLAS. This approach significantly reduces instructors' workload while maintaining pedagogical accuracy through human validation.

The fourth contribution introduces the *Generative AI-assisted SQL Query Generator*, a tool that automates the generation of diverse and practical *SQL Query Description Problems (SDPs)*. This tool alleviates the burden on instructors by automating the creation of database schemas, practice questions, and correct answers, enabling more efficient and varied SQL training material creation.

1.5 Contents of Dissertation

The remaining part of this thesis is organized as follows.

Chapter 2 reviews the overview the web-based *SQL PLAS*.

Chapter 3 presents the Grammar-concept Understanding Problem (GUP) in SQL database programming

Chapter 4 presents the Grammar-concept Understanding Problem (GUP) in SQL-Python database programming.

Chapter 5 presents the Comment Insertion Problem (CIP) in SQL database programming.

Chapter 6 presents the SQL Query Description Problem (SDP) in SQL database programming.

Chapter 7 proposes the Generative AI-assisted SQL Query Generator.

Chapter 8 presents previous works related to this thesis.

Finally, Chapter 9 concludes this thesis with some future works.

Chapter 2

Overview of SQL Database Programming Learning Assistant System (SQL PLAS)

This chapter provides an overview of the web-based *SQL Programming Learning Assistant System (SQL PLAS)*.

2.1 System Architecture

SQL PLAS is a self-learning web-based platform that supports well-known SQL database programming language. For teachers, it supports generating instances and analysing students' responses. For students, it supports practicing and using the system. An overview of software architecture in *PLAS* is shown in Figure 2.1.

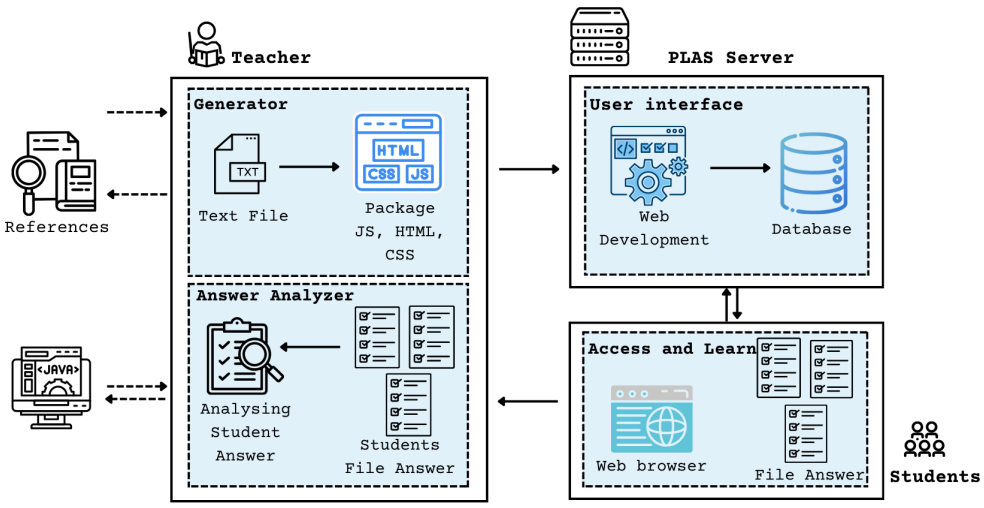

Figure 2.1: SQL PLAS architecture.

The *SQL PLAS* platform was designed as a web-based self-directed learning environment accessible via a web browser. The interface was implemented utilizing *HTML5*, *CSS*, and *JavaScript* to facilitate access to the exercises, allowing fast responses to inputs for immediate feedback. User interaction and data storage capabilities are executed on a browser with *JavaScript*. This design streamlines deployments and maintenance.

The exercises can be developed and managed by teachers through an administrative interface that enables them to establish question instances, categorize them, and provide correct answers along with validation criteria. Upon answer submission by a student, the system automatically compares them with the stored correct answers, identifies the incorrect ones, and documents them for future review.

Students engage with the *SQL PLAS* platform using a web browser to complete the exercises. The platform stores the related data, including the submission and answer histories, which will be utilized to assess student developments and address specific challenges.

2.1.1 Instance Generation Functions

The *instance generator* has been implemented for each exercise type and language within *PLAS*. It operates together with the settings for different types of exercise problems. Figure 2.2 shows how it processes and masks the answers for *SDP*.

In this process, teachers first input a source code according to the topic. Then, the generator masks the correct answers and generates the corresponding *HTML*, *CSS*, and *JavaScript* files before presenting them to students. During exercise executions, student inputs are evaluated through *string matching* against the correct answers stored in the program. The same mechanism is applied to generating instances for other exercise types. On the student side, the masked parts in *SDP* are displayed as blank input fields, allowing learners to fill in their answers directly through the web interface.

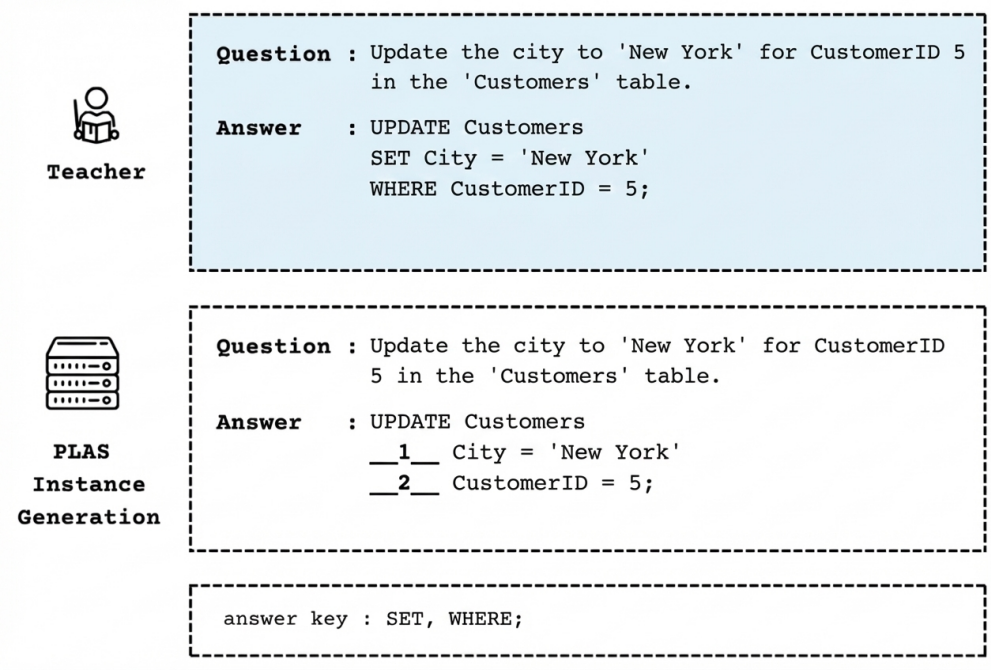

Figure 2.2: Example of *SDP* instance generation.

2.1.2 Operation Flow

The operation flow of the answering function in SQL PLAS involves the following steps:

1. Assignment generation: a teacher collects Java source codes for exercises, and generate the files for new assignments by running the corresponding generator or manually.

2. Assignment distribution: the teacher distributes the assignment files to the students by using a file server.

3. Assignment answering: a student answers the problem instances in the assignments using a web browser with the Internet connections.

4. Answering result submission: a student submits the final answers to the teacher using the file server or an email.

5. Answering result upload: the teacher stores the answers from the students at the corresponding folders in his/her PC for managements.

2.1.3 Distributed Files

Table 2.1 outlines the files to be distributed to the students for assignment answering. These files are necessary for the problem view, the answer marking, and the answer storage.

Table 2.1: Files for distribution in SQL PLAS.

File name	Outline
css	CSS file for Web browser
index.html	HTML file for Web browser
page.html	HTML file for correct answers
jplas2015.js	js file for reading the problem list
distinction.js	js file for checking the correctness of answer
jquery.js	js file for use of jQuery
sha256	js file for use of SHA256
storage.js	js file for Web storage

2.1.4 Cheating Prevention

To prevent disclosing the correct answers to the students, their hash values by SHA256 [8] are distributed. In addition, to avoid generating the same hash values for the same correct answers, the assignment ID and the problem ID are concatenated with each correct answer before hashing. This ensures that identical correct answers for different blanks are converted to different hash values, maintaining independence between the blanks.

2.2 Implemented Problem Types

SQL PLAS provides various types of practices to cover different learning stages of SQL database programming:

1. Grammar-concept Understanding Problem (GUP) gives questions about the concepts of important keywords, including reserved words and commonly used libraries in the programming language, in the provided source code. It focuses on keywords and libraries in the source code [3].

2. Comment Insertion Problem (CIP) gives CIP instance consists of source code with blank comments, a set of comments to fill in the blanks, and their correct answers. it focuses to understand the structures and logics of source codes [4].

3. SQL Query Description Problem (SDP) it is designed to aid novice students in comprehending the fundamental query to data manipulation and retrieve of SQL database programming for executing a range of queries. An SDP instance comprises a collection of questions and a database table.

For any exercise problem, the correctness of the answer from a student is verified automatically. The correctness of the student answer is checked through *string matching* with the correct one for *GUP*, *CIP*, and *SDP*.

2.3 SQL Query Description Problem (SDP)

An *SDP* instance comprises of a series of questions, materials on the *DML* in *SQL database programming*, and the correct answers to the questions. Students are required to answer each question with an *SQL* query statement based on the material that are taught in a class and/or are included in the system's module. Essentially, the question outlines a manipulation data concept of *SQL database programming* that is covered in the module. The aim of *SDP* is to assist students focus on skill developments by understanding the basics of *SQL* query for data manipulation and retrieval, rather than on studying factual or theoretical knowledge [9].

To generate a new assignment for *CWP*, the teacher needs to perform the following operations.

1. Pick up a material in *SQL database programming* from a university class, a textbook, or a website to create a module for students to study [10][11].

2. Extract the keywords from the module that correspond to the ones in Table 3.1.

3. Select the relevant questions in Table 3.1 for the extracted keywords.

4. Discard the redundant question-answer pairs that have already been selected for this material.

5. Generate the *SDP* instance text file with the associated image, questions, and correct answers.

6. Create the *HTML/CSS/JavaScript* files [12] for the answer interface system on the web browser by running the generation program [4] with the text file.

2.3.0.1 Instance List Page

The *instance list page* allows students to view and manage the assigned *SDP* instances. Figure 6.3 shows the layout of this page, where each *SDP* instance is displayed with its title and progress status. The instances highlighted in *green* indicate that all the questions in them have been answered correctly by a student, while those in *yellow* show partial completions. The instances with no highlighting are those that have not yet been attempted. Students are advised to read provided instructions before attempting exercises. This visual feedback mechanism helps learners track their progress and identify which topics require further review.

2.3.0.2 Answer Page

The *answer page* enables students to input and test *SQL* queries for questions interactively. Figure 6.4 illustrates the page providing a text input area where students can write the *SQL* query in response to each question. Upon submission, the *JavaScript* program on the page checks the answer using *string matching* and executes the query to confirm correctness.

Incorrect submissions are highlighted in *red*, while correct ones appear in *white*. Students may resubmit their answers until all questions are correctly solved. Each attempt is recorded, including the submission time, the answers, and results, allowing teachers to monitor learning progress and common errors.

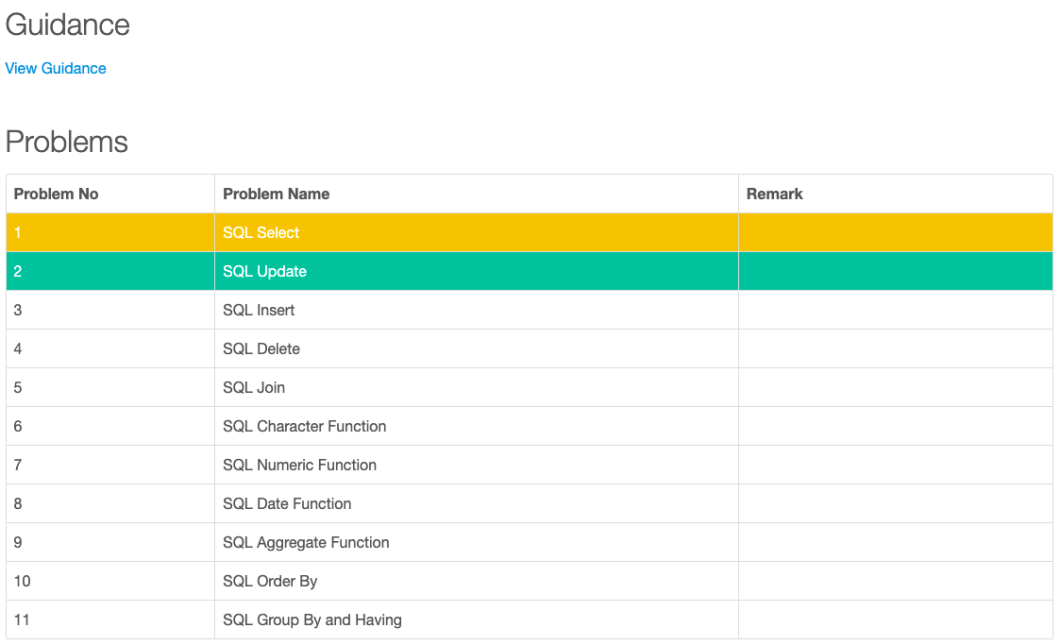

Figure 2.3: Instance list page.

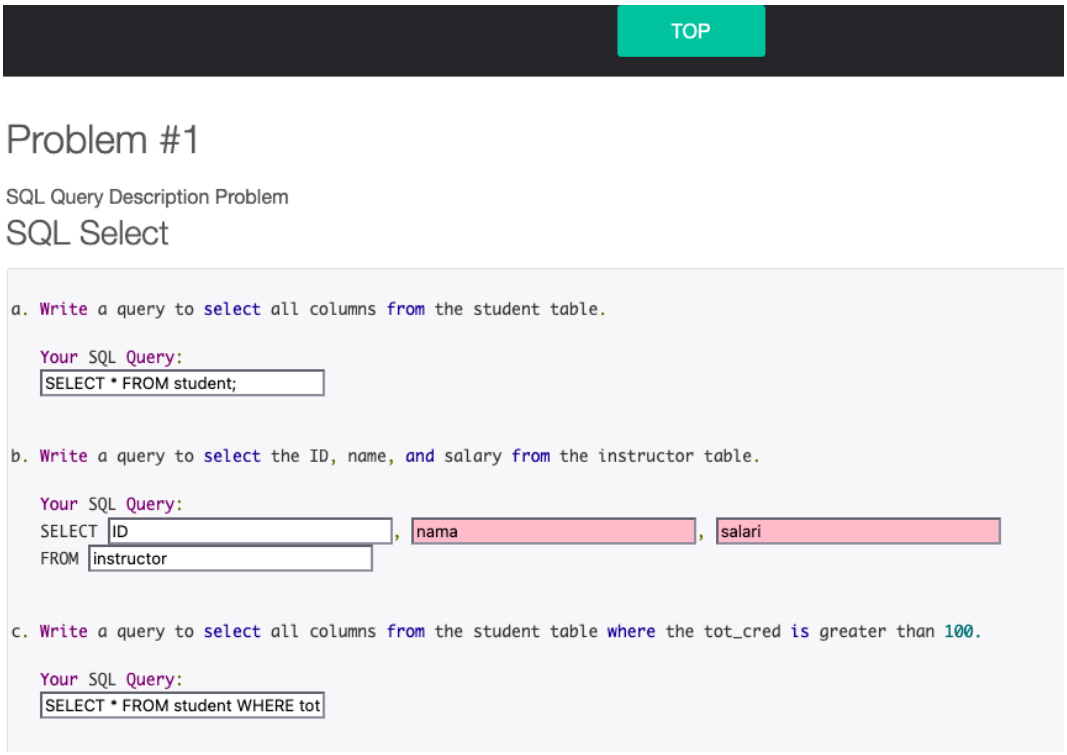

Figure 2.4: *SDP* answer page.

2.4 Summary

This chapter overviewed the *SQL database programming Learning Assistant System (SQL PLAS)*. It discussed the system architecture of the SQL PLAS implementation, the answering function in SQL PLAS, several types of exercise problems, and the details of the *SQL Query Description Problem (SDP)*.

Chapter 3

Implementation of Grammar-concept Understanding Problem for Self-Study of Introduction Basic to SQL Database Programming

This chapter presents the *grammar-concept understanding problem* for self-study of introduction basic to SQL database Programming [13].

3.1 Introduction

For the automatic generations of *GUP* instances, we defined 15 keywords related to it. Unlike *GUP* instances for other languages in previous studies, each question requests to answer either *T (true)* or *F (false)* to its statement. Any student answer is marked by *string matching* with the correct one.

3.1.1 Definition of GUP

A *GUP* instance is made up of a series of true/false questions, a material on the *introductory SQL database programming*, and the correct answers to the questions. Students are asked to answer each question with a correct or incorrect response based on the material that has been taught and is included in the system's module. Essentially, the question outlines the basic concepts of *SQL database programming*, which is covered in the module. The goal of this is to aid students in comprehending the basics of databases, database types, tables, data, and an overview of the *SQL*.

3.1.2 Input Files to Instance Generation

A teacher must collect materials for *introductory SQL database programming*. From this material, the teacher prepares some keywords and true/false questions to create a *GUP* instance that students are expected to solve. The associated *GUP* instance file will then be created by executing the *GUP* instance generation method in the section generation procedure.

3.1.3 Keywords and Questions

To generate *GUP* instances, a collection of keywords and the questions that match them must be prepared in advance. We chose 15 keywords and their corresponding questions for the Introduction to SQL Database Programming. Due to space constraints, Table 3.1 only displays a portion of them. In essence, the definition of each keyword is provided in the question that corresponds to it. The learner must comprehend the fundamental grammatical ideas in the Introduction to read the questions and react to the matching keywords in the answer forms, the simple *SQL* database administration.

Table 3.1: Keyword and question list

keyword	question
three level to view	A database stores data
database advantages	Storage is more secure due to encryption in the database
non-relational database	A non-relational database stores data without a schema
relational database	A relational database stores data according to a schema
structure of table	In s relational database, data is stored in tabular format
relationships	two tables are related through foreign key
read data	SQL SELECT statement is SQL command is used to select information
table manners	table names must be lowercase
assigned seat	Which command is used to group the result set of a SELECT statement based on one or more columns
SQL data types	fisrt_name attribute uses integer data type
schemas	Schemas are often referred to as ”blueprints” of databases
database storage	Laptop can be a server if it is set up to provide a service

Table 3.1 includes the basic terms in *database programming* that are represented by *CRUD*. *CRUD* stands for Create, Read, Update, and Delete of databases, tables, or records. In addition, it includes keywords to present the *SQL* functions for aggregate, date, character, and numeric. For the course application, the keywords for the *GUP* instances should be selected correctly, depending on the target database course according to the curriculum [10]. By solving the provided *GUP* instances, it is expected that a student understands the meaning or behavior of each crucial keyword in *SQL-Python database programming*.

The fundamental database programming keywords, represented by *CRUD*, are listed in Table 3.1. Create, Read, Update, and Delete databases, tables, or records is *CRUD*. It also contains keywords to display the date, character, numeric, and aggregate *SQL* functions. According to the curriculum, the keywords for the *GUP* instances in the course application should be appropriately

chosen based on the target database course [10]. A student is anticipated to comprehend the significance or operation of each key keyword in *SQL-Python database programming* by solving the given *GUP* examples.

3.1.4 GUP Generation Procedure

The answer interface for a new *GUP* instance can be generated through the following procedure:

1. Collect a material of introduction *SQL database programming* into a module to be studied by students from a university module, textbook, or website.
2. Extract the keywords into the list that correspond to the keywords in Table 3.1 from a module.
3. Select the relevant true/false question from Table 3.1 for each extracted keyword.
4. Discard the redundant pair of the question and the answer if they are already selected for this material.
5. Produce the *GUP* instance text file with the related image, true/false questions, and correct answers.
6. Generate the *HTML/CSS/JavaScript* files for the answer interface on the web browser by running the generator in [4] with this text.

3.1.5 GUP Answer Interface

Figure 3.1 shows the *answer interface* to solve the *GUP* instance at ID=4. When a student enters an incorrect answer in the interface, the corresponding input form is highlighted with the *red* background. Otherwise, the correct response is displayed on the *white* background. The student can repeat by submitting the answers until each question becomes correct. The *answer interface* stores the answers with their correctness and the submission date and time, each time the student submits the answers.

3.2 Evaluation

In this section, we evaluate the GUP for self-study of *introductory SQL database programming* by novice students.

3.2.1 Evaluation Setup

We generated five *GUP* instances with 50 questions in Table 3.1. Then, we assigned them to 30 first-year undergraduate students who were taking the database programming course at the *Indonesian Institute of Business and Technology* as a quiz with 30 minutes in the *introducing databases and SQL programming* course.

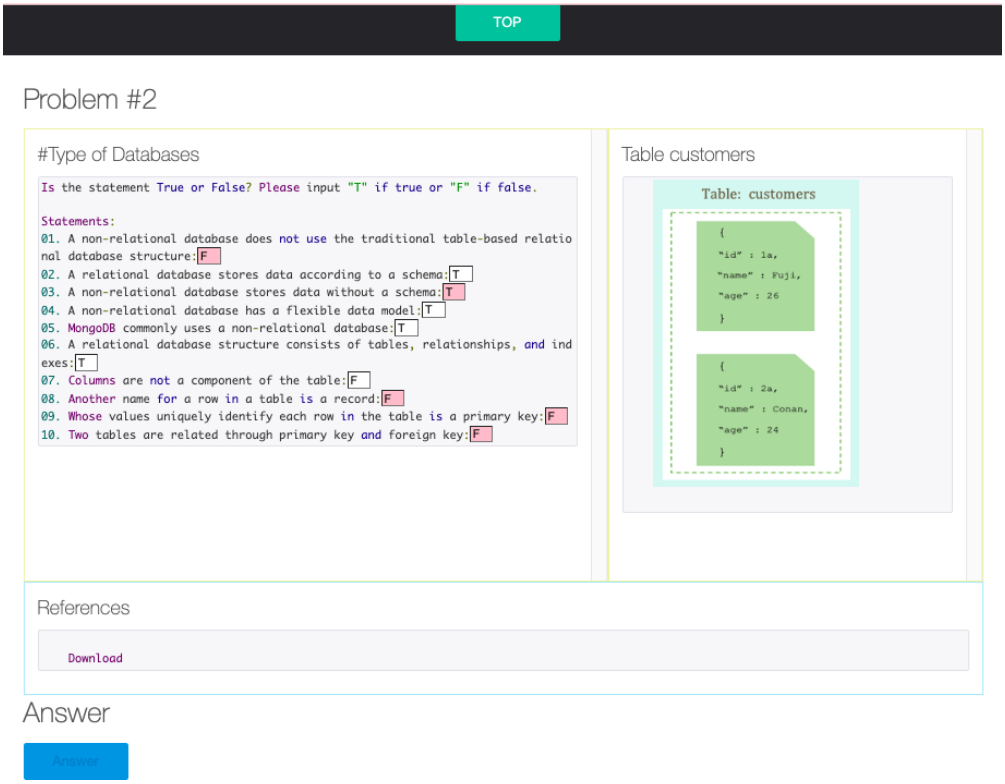

Figure 3.1: GUP answer interface.

3.2.2 Summary of Results

Table 6.17 displays the total number of students who completed the *GUP* task, the average correct rate and its standard deviation (SD), and the average number of submission times and its SD among the five *GUP* instances. With a high average right response percentage of 100%, we may infer from this table that the students are capable of answering them accurately. The students responded to each *GUP* instance just once or twice, as indicated by the average submission time of 1.78.

Table 3.2: Result for All GUP Instances

# of students	correct rate (%)		#of submission	
	ave	SD	ave	SD
30	100	0.00	1.78	0.30

3.2.3 Results for GUP Individual Instances

Figures 3.2 illustrates the average correct answer rate and the average number of answer submission times by the 30 students for each *GUP* instance, respectively. The graph indicates that instances at `ID = 2` and `ID = 3` show the highest submission times with average correct answer rate 100%.

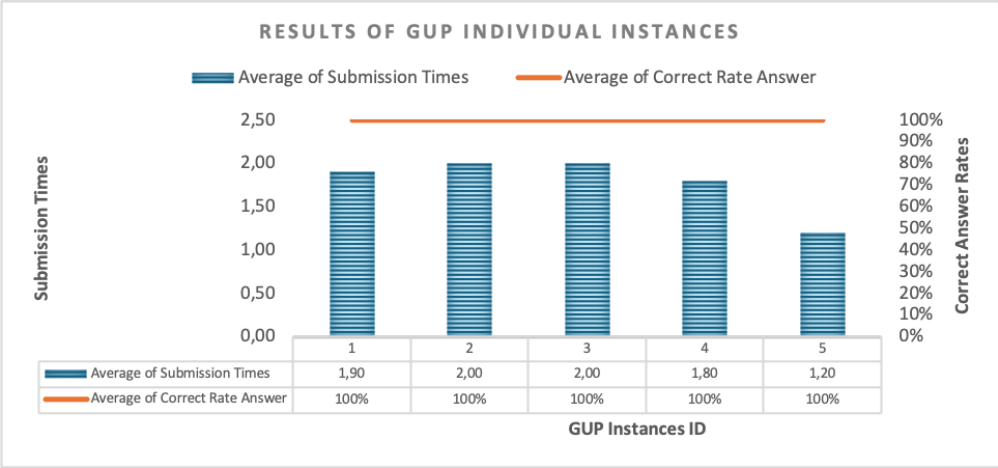

Figure 3.2: Result of *GUP* individual instances by students.

3.2.4 Results for Individual Students

Figure 3.3 illustrates the average correct rates and the average number of submission times of all *GUP* instances for each of the 30 first-year students in MDI major. The graph indicates that 30 students achieved the average correct rate 100%. The best student at `ID = 4`, `ID = 9`, and `ID = 14` could solve all the instance with 100% correctly with only 1.67 submissions for each instance. However, the worst student at `ID = 27` could solve 100% of the questions with 2.11 submission time for each instance. Five instances were solved at all, which means students studied very seriously.

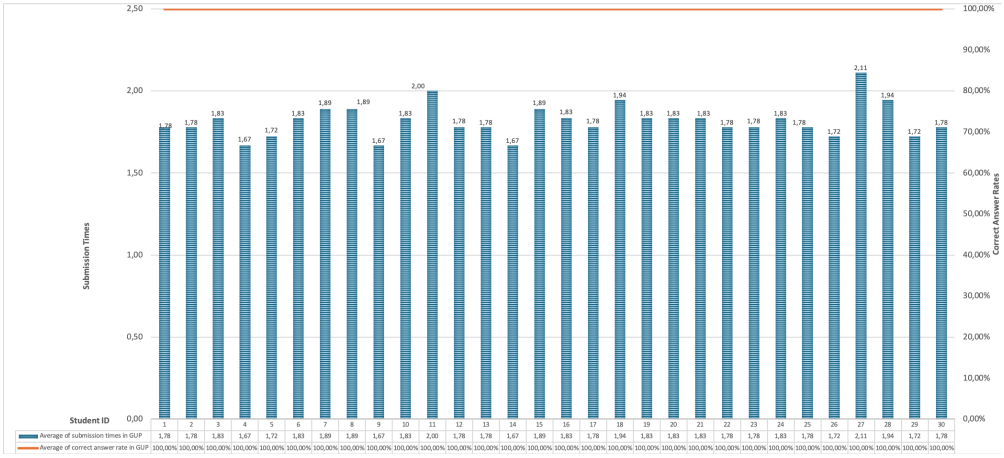

Figure 3.3: Result of *GUP* individual students.

3.2.5 Results of System Usability Scale (SUS)

Following the experiment, we delivered a *System Usability Scale (SUS)* questionnaire to the students. The questions are selected from [14]. The SUS score is calculated by the following procedure:

1. For each question with an odd number, deduct one point from your score.

2. Take each even-numbered question's value and deduct it from 5.

3. Increase the overall score by the newly found values. After that, multiply this by 2.5.

To determine the results of the assessment grade, there are two ways that can be used. First, looking at the level of user acceptance, scale grade, and adjective rating, which consists of the level of user acceptance, there are three categories, namely not acceptable, marginal, and acceptable. Meanwhile, in terms of scale grade level, there are six. The scale is A, B, C, D, E, and F. The second determination is seen from the percentile range (*SUS* score), which has an assessment grade consisting of A, B, C, D, and E. Determination of assessment results is based on the *SUS* score percentile rank carried out in general based on the results of user assessment calculations. These two determinations are contained in Tables 3.3 and 3.4.

Table 3.3: SUS Score Percentile Rank

Grade	Note
A	Score >= 80.3
B	Score >= 74 and <80.3
C	Score >= 68 and <74
D	Score >= 51 and <68
E	Score <51

Table 3.4: Acceptability Range

SUS Score	Note
0 - 50.9	Not acceptable
51 - 70.9	Marginal
71 - 100	Acceptable

Table 3.5 shows the score results from 30 respondents who are first-year students who takes database course. The results of the usability test in Table 3.5 were carried out step by step in accordance with the *SUS* calculation guidelines. The final *SUS* score from 30 respondents' responses was 77, in accordance with the *SUS* interpretation guidelines in Table 3.3, which can be interpreted as follows:

1. Interpretation with acceptability range. Referring to Table 3.4, the score of 77 is included in the Acceptable range.

2. Interpretation using the Grade scale as in Table 3.3. The score of 77 is included in grade B.

Table 3.6 presents the percentage of responses to all respondent statement items regarding the questionnaire that was distributed. There are minor problems that occur from the results of the tests that have been carried out, namely:

- The percentage in the even statements, namely Q2, Q4, Q6, Q8 and Q10 is 0%, which means that all respondents do not find it complicated to use the system, they do not need a technician to use the system, the system is quite consistent for respondents, not confusing and respondents quickly adapt to using the system.

Table 3.5: SUS Scores

Responden	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Total	SUS Score (Total * 2.5)
1	**4**	**4**	**4**	**4**	**4**	**3**	**4**	**4**	**3**	**4**	**38**	**95**
2	**4**	**2**	**4**	**2**	**4**	**2**	**4**	**2**	**2**	**2**	**28**	**70**
3	**3**	**3**	**3**	**3**	**2**	**3**	**2**	**3**	**3**	**3**	**28**	**70**
4	**4**	**2**	**4**	**2**	**4**	**3**	**4**	**2**	**4**	**2**	**31**	**78**
5	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**30**	**75**
6	**3**	**3**	**3**	**3**	**4**	**4**	**4**	**3**	**4**	**3**	**34**	**85**
7	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**2**	**3**	**29**	**73**
8	**4**	**3**	**4**	**3**	**4**	**2**	**4**	**3**	**2**	**3**	**32**	**80**
9	**3**	**3**	**3**	**3**	**4**	**2**	**4**	**3**	**3**	**3**	**31**	**78**
10	**2**	**2**	**2**	**2**	**2**	**3**	**2**	**2**	**2**	**2**	**21**	**53**
11	**3**	**3**	**3**	**3**	**3**	**2**	**3**	**3**	**3**	**3**	**29**	**73**
12	**3**	**3**	**3**	**3**	**3**	**2**	**3**	**3**	**3**	**3**	**29**	**73**
13	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**3**	**4**	**39**	**98**
14	**3**	**3**	**3**	**3**	**4**	**3**	**4**	**3**	**3**	**3**	**32**	**80**
15	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**30**	**75**
16	**4**	**4**	**4**	**4**	**3**	**4**	**3**	**4**	**3**	**4**	**37**	**93**
17	**3**	**3**	**3**	**3**	**4**	**3**	**4**	**3**	**3**	**3**	**32**	**80**
18	**4**	**3**	**4**	**3**	**4**	**4**	**4**	**3**	**3**	**3**	**35**	**88**
19	**2**	**2**	**2**	**2**	**3**	**3**	**3**	**2**	**3**	**2**	**24**	**60**
20	**4**	**3**	**4**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**32**	**80**
21	**4**	**3**	**2**	**3**	**2**	**3**	**2**	**3**	**3**	**2**	**27**	**68**
22	**3**	**3**	**3**	**3**	**3**	**2**	**3**	**3**	**2**	**3**	**28**	**70**
23	**2**	**3**	**3**	**4**	**3**	**2**	**3**	**4**	**3**	**2**	**29**	**73**
24	**3**	**2**	**4**	**3**	**4**	**4**	**4**	**3**	**4**	**3**	**34**	**85**
25	**3**	**3**	**3**	**3**	**4**	**3**	**4**	**3**	**3**	**3**	**32**	**80**
26	**4**	**3**	**3**	**4**	**3**	**3**	**3**	**4**	**4**	**3**	**34**	**85**
27	**3**	**4**	**4**	**3**	**3**	**4**	**3**	**3**	**2**	**3**	**32**	**80**
28	**3**	**3**	**3**	**3**	**4**	**3**	**4**	**3**	**2**	**3**	**31**	**78**
29	**4**	**3**	**4**	**2**	**4**	**4**	**4**	**2**	**3**	**2**	**32**	**80**
30	**3**	**3**	**2**	**3**	**3**	**3**	**3**	**3**	**2**	**3**	**28**	**70**
Average Score (Final Result)												77

Table 3.6: Presentation of Questionnaire Results

Score Scale	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
SD	0.0%	13.3%	0.0%	16.7%	0.0%	23.3%	0.0%	16.7%	0.0%	10.0%
D	0.0%	70.0%	0.0%	66.6%	0.0%	53.4%	0.0%	66.6%	0.0%	66.7%
N	10.0%	16.7%	13.3%	16.7%	10.0%	23.3%	10.0%	16.7%	26.7%	23.3%
A	53.3%	0.0%	50.0%	0.0%	43.3%	0.0%	43.3%	0.0%	60.0%	0.0%
SA	36.7%	0.0%	36.7%	0.0%	46.7%	0.0%	46.7%	0.0%	13.3%	0.0%

- In statement 1, there were 10% of respondents who were hesitant to use this system.
- In statement 5, there were 10% of respondents who were doubtful that the system would work properly.
- In statement 7, 10% of respondents were doubtful that other people would quickly understand how to use the system.
- In statement 9, there were 26.7% of respondents who were doubtful that there would be no obstacles in using the system.

3.3 Summary

This chapter presented the grammar-concept understanding problem (*GUP*) for the first-step self-study of *introductory SQL database programming*. For evaluations, five *GUP* instances were made on basic concepts and were assigned to 30 students in Indonesian Institute of Business and Technology (INSTIKI). The results confirmed the applicability to beginners in database programming. Besides, the usability of the proposal got a *SUS* score of 77 (grade B), which is acceptable and suitable for use by beginner students.

Chapter 4

An Implementation of Grammar-concept Understanding Problem for Entry-level SQL-Python Database Programming

This chapter presents the grammar-concept understanding problem (*GUP)* for SQL Python database Programming Learning Assistant System (*SQL-Python PLAS)* [15].

4.1 Introduction

Combining *SQL*, a domain-specific language for querying relational databases, with *Python*, a versatile high-level language supported by libraries *sqlite3*, provides an effective toolkit for integrating, manipulating, and analyzing data. This study addresses the *GUP* in *SQL–Python PLAS* for novice students by designing keywords on basic and advance *Python database programming*, generating definition-based questions, and automatically assessing answers via *string matching*, so that beginners can more accurately understand and internalize key programming concepts.

4.2 GUP Instance Generation

In this section, we present a *GUP* instance generation algorithm that a teacher can use to generate a new *GUP* instance from the specified source code [16].

4.2.1 Overview

A *GUP* instance is made up of a *SQL Python* source code, certain questions, and the right answers to those questions. Each query relates to a core grammar idea used in *SQL Python* database programming, which is found in the source code. The student is asked to choose the corresponding element or term from the code. String matching is used to compare the student's response to the right response in order to assess it.

4.2.2 Input Files

To use the technique, the teacher must first create a source code file that contains the grammatical ideas that students will learn about through the resolution of *GUP* instances. After reading this

source code file, the algorithm will use the steps described in Section 3.1.4 to create the associated *GUP* instance file. A list of keywords and the questions associated with them must be made before the algorithm is used [3].

4.2.3 Keywords and Questions

This algorithm utilizes a pre-defined set of keywords and correlated essay questions to facilitate learning. Table 4.1 lists the 105 selected keywords focusing on *SQL-Python* grammar, along with their corresponding definitions and problems. By analyzing the questions and identifying the correct answer keywords, students grasp the crucial grammatical concepts in database programming.

The *GUP* curriculum is derived from standard university modules [10]. It covers fundamental *CRUD* operations (Create, Read, Update, Delete) alongside essential SQL functions, including character, numeric, date, and aggregate functions. Based on these topics, we generated 14 distinct *GUP* instances using SQL Python source codes to test student understanding.

Figure 4.1 displays these 14 instances along with their status indicators. Student progress is visually tracked via color-coded highlights: a "Completed" status with a green highlight indicates the student has correctly answered all questions, while a "Tried" status with a yellow highlight signifies an incomplete attempt. Instances where no work has been done remain unhighlighted.

4.2.4 Example *GUP* Instance

TXT1 shows the sample *SQL-Python* source code of the *SQL* constraint. Students are expected to learn the meaning of primitives or keywords that appear in this code.

TXT1: SQL–Python Source Code

```
# Create customer table
cursor.execute("""
CREATE TABLE customer (
    customer_id INT NOT NULL,
    customer_code VARCHAR(20) UNIQUE,
    customer_name VARCHAR(50),
    customer_country VARCHAR(20) DEFAULT 'JAPAN',
    CONSTRAINT CustomerPK PRIMARY KEY (customer_id)
);
""")

# Create index
cursor.execute("""
CREATE INDEX customer_index
ON customer(customer_code);
""")

# Create product table
cursor.execute("""
CREATE TABLE product (
    customer_id INT,
    name VARCHAR(100),
    CONSTRAINT ProductFK
        FOREIGN KEY (customer_id)
        REFERENCES customer(customer_id)
);
""")
```

Table 4.1: Keywords and Question in GUP for Python Database Programming

Keywords	**Question**	Instance ID
sqlite3	What is the software library that provides a relational database management system (RDBMS) ?	1
import	Which keyword is used to import the module?	1
connect	Which function is used to create a connection object?	1
execute	Which keyword is used to execute SQL commands?	1
mydatabase	What is the name of the database in the above source code?	1
pragma	Which statement is used to retrieve a list of all databases accessible from the current connection?	1
fetchall	Which query is used to fetch all the rows returned?	1
drop	Which keyword is used to delete a database?	1
close	Which keyword is used to close the connection between MYSQL and Python?	1
createTable	Which keyword is used to create a table?	2
employees	What is the name of the table created in the above source code?	2
insertInto	Which keyword is used to insert values into a table?	2
3	How many rows are inserted into the 'employees' table?	2
IDNameAge	What information is printed when displaying the 'employees' table data?	2
id	What is an attribute as a primary key in the above source code?	2
integer	What is the data type of the 'id' attribute in the 'employees' table?	2
SELECT*	Which keyword is used to show all data in a table?	2
fetchall	Which keyword is used to fetch all the rows?	2
for	Which keyword is used to loop a statement that iterates over each row in the rows collection?	2
Where	Which keyword is used to filter specific tables to display?	2
dropTable	Which keyword is used to delete a table?	2
DROPTABLEIfExist	Which keyword is used to drop the table named "mytable" if it exists?	2
update	Which keyword is used to update existing records in a table?	3
alterTable	Which keyword is used to modify the structure of a table?	3
addColumn	Which keyword is used to modify a table with add a new attribute?	3
renameColumn	Which keyword is used to modify a table with rename an attribute?	3
dropColumn	Which keyword is used to modify the table with delete an attribute?	3
commit	Which method is called to save the changes made to the database?	3
innerJoin	What is a method of combining rows from two or more tables based on a related column between them?	4
crossJoin	What is a type of join operation in SQL that combines each row from one table with every row from another table?	4
leftJoin	What is a type of join operation in SQL that combines rows from two tables based on a related column between them?	4
rightJoin	What is a type of join operation in SQL that combines rows from two tables based on a related column between them?	4
insertInto	Which keyword is used to insert values into a table?	5
execute	Which function is used to insert values in some attributes?	5
executemany	Which keyword is used to insert multiple rows into a table?	5
select*from	Which keyword is used to show all records in a table?	5
fetchall	Which keyword fetches all rows of a query result set and returns a list of tuples?	5
fetchone	Which keyword is used to show only in one row?	5
where	Which keyword is used to filter records based on a condition in a SELECT statement?	5
orderBy	Which keyword is used to sort the result?	6
asc	Which keyword is used to sort the result in ascending?	6
desc	Which keyword is used to sort the result in descending?	6
offset	Which keyword is used to limit the number of records returned from the query starting from another position?	6
update	Which keyword is used to update records in a table?	6
set	Which keyword is used to update records with new values in a table?	6
where	Which keyword is used to update the selection record or selection values in a table?	6
delete	Which keyword is used to delete records from an existing table?	6
where	Which keyword is used to filter the selection record that wants to be deleted?	6
rowcount	Which keyword returns the number of rows affected by the last statement?	6
commit	Which keyword is used to permanently save all the changes made in the transaction of a database or a table?	6
Lower	Which function is used to convert a given string to lowercase letters?	7
upper	Which function is used for a string function that converts a given string to uppercase letters?	7
substring	Which function is used to extract a substring from a given string?	7
ltrim	Which function is used to remove any leading whitespace characters (space, tab, newline, etc.) from a given string?	7
rtrim	Which function is used to remove any trailing whitespace characters (space, tab, newline, etc.) from a given string?	7

right	Which function is used to return a specified number of characters from the right side of a given string?	7
left	Which function is used to return a specified number of characters from the left side of a given string?	7
char	Which function is used to return a character with a specified ASCII value?	7
length	Which function is used to return the length of a given string in bytes?	7
reverse	Which function is used to return a string with the characters in reverse order?	7
space	Which function is used to return a string consisting of a specified number of spaces?	7
abs	Which function is used to return the absolute value of a numeric value?	8
ceiling	Which function is used to return the smallest integer value greater than or equal to a numeric value?	8
floor	Which function is used to return the largest integer value less than or equal to a numeric value?	8
round	Which function is used to round a numeric value to a specified number of decimal places?	8
truncate	Which function is used to truncate a numeric value to a specified number of decimal places?	8
mod	Which function is used to return the remainder of a division operation between two numeric values?	8
power	Which function is used to return the remainder of a division operation between two numeric values?	8
sqrt	Which function is used to raise a numeric value to a specified power?	8
exp	Which function is used to return the square root of a numeric value?	8
log	Which function is used to return the exponential value of a numeric value?	8
result	Which function is used to return the natural logarithm of a numeric value?	8
datetime('now')	Which function is used to return the current date and time?	9
date('now')	Which function is used to return the current date?	9
time('now')	Which function is used to return the current time?	9
date()	Which function is used to extract the date part from a date or datetime value?	9
time()	Which function is used to extract the time part from a date or datetime value?	9
datediff()	Which function is used to calculate the number of days between two dates?	9
dateadd()	Which function is used to add a specified number of days, months, or years to date?	9
datesub()	Which function is used to subtract a specified number of days, months, or years from a date?	9
strftime()	Which function is used to format a date value using a specified format string?	9
strftime(y)	Which function is used to extract the year from a date or datetime value?	9
strftime(m)	Which function is used to extract the month from a date or datetime value?	9
strftime(d)	Which function is used to extract the day of the month from a date or datetime value?	9
sum	Which function is used to calculate the sum of a column?	10
avg	Which function is used to calculate the average of a column?	10
count	Which function is used to count the number of rows?	10
max	Which function is used to return the maximum value in a column?	10
min	Which function is used to return the minimum value in a column?	10
GroupBy	Which command is used to group the result set of a SELECT statement based on one or more columns?	11
having	Which command is used for filtering the results of grouped queries based on conditions involving aggregate functions?	11
union	Which operator is used to combine the result sets of two or more SELECT queries into a single result set?	12
intersect	Which operator is used to return only the rows that are common to the result sets of two or more SELECT statements?	12
except	Which operator is used to find differences or exceptions between two sets of data?	12
createView	Which keyword is used to make a virtual table that is based on the result of a SELECT statement?	13
alterView	Which keyword is used to modify the definition of an existing view in a database?	13
dropView	Which keyword statement is used to remove a view from the database?	13
selectIf	Which keyword is used to returns a value based on a condition?	13
IfNull	Which keyword is used returns the first non-null value?	13
NullIf	Which keyword is used to returns NULL if two values are equal, otherwise returns the first value?	13
Case	Which function can be useful when you want to replace specific values with NULL, based on a condition?	13
If	Which function is used to execute different statements based on a condition?	14
Case	Which function can be useful when you want to replace specific values with NULL, based on a condition?	14
While	Which function can be useful when you want to replace specific values with NULL, based on a condition?	14
For	Which function can be useful when you want to replace specific values with NULL, based on a condition?	14

Problems

Problem No	Problem Name	Remark
1	Connection and Create a Database	Tried
2	Creating a Table	Completed
3	Update a Table	Completed
4	Join Tables	Completed
5	Insert New Records and Show Records	Tried
6	Sorting, Update and Delete Record	Tried
7	Character Function	
8	Numeric Function	
9	Date Function	
10	Aggregate Function	
11	Group By and Having	
12	Set Operation	
13	View	
14	Control Flow Function	Completed

Figure 4.1: GUP Instances Interface

4.2.5 Answer Interface for GUP

The web browser's response interface for resolving a *GUP* instance is shown in Figure 4.2. When a student enters an incorrect response into the interface, the input form is highlighted in red; in contrast, a correct response is displayed with a white background. Students can examine and submit their responses multiple times through this interface until each one is accurate. The system automatically logs the submission times as well as each student's responses.

4.2.6 *GUP* Generation Procedure

The answer interface for a new *GUP* instance can be generated through the following procedure:

1. Collect an *SQL-Python* source code from a textbook or a website for students to study.
2. Extract the keywords into the list that correspond to the keywords in Table 3.1 from the source code.
3. Select the relevant question from Table 3.1 for each extracted keyword.
4. Discard the redundant pair of the question and the answer if they are already selected for this source code.
5. Produce the *GUP* instance text file that contains the source code, the related questions, and the correct answers.
6. Generate the *HTML/CSS/JavaScript* files for the answer interface on the web browser by running the generator in [16] with this text.

The Source Code

```
1     #import software library
2     import sqlite3
3
4     #Connect to the database
5     conn = sqlite3.connect('example.db')
6
7     #Create a cursor object
8     cursor = conn.cursor()
9
10    #Create a new database
11    cursor.execute("CREATE DATABASE mydatabase")
12
13    #Show all databases
14    cursor.execute("PRAGMA database_list")
15    databases = cursor.fetchall()
16    print("Databases:")
17    for database in databases:
18    print(database[1])
19
20    #Drop a database
21    cursor.execute("DROP DATABASE mydatabase")
22
23    #Close the connection
24    conn.close()

Question
Q1. What is the software library that provides a relational database management system (RDBMS) ? sqli
Q2. Which keyword is used to import the module? imp
Q3. Which function is used to create a connection object? cor
Q4. Which keyword is used to execute SQL commands? exe
Q5. What is the name of the database in the above source code? my
Q6. Which statement is used to retrieve a list of all databases accessible from the current connectio
n? pra
Q7. Which query is used to fetch all the rows returned? fet
Q8. Which keyword is used to delete a database? dro
Q9. Which keyword is used to close the connection between MYSQL and Python? clo
```

Answer

Answer

Figure 4.2: GUP Answer Interface

4.2.7 Implementation Steps

A teacher will implement *GUP* by the following steps:

1. *GUP instance generations:* The teacher collects appropriate source code for *SQL-Python* database programming, with a specific emphasis on the keywords to be studied, from modules, textbooks, or websites. Next, the teacher runs the response interface generating software and the *GUP* generation algorithm to build HTML files for the newly created *GUP* instances.

2. *Distributions to students and their answers:* The teacher uploads the files to a web server and provides the students with the corresponding URL. The students can then access the *GUP* instances through a web browser and respond to them. The automatic answer marking feature within the browser confirms the validity of their answers.

3. *Answer file submission and result analysis:* The student uses the browser to download the answer text files, which are then delivered to the teacher through email or an online learning platform. The teacher then processes the text files to retrieve the student-submitted solution results using an answer analysis tool.

4.3 Evaluation

In this section, we evaluate the generated 14 *GUP* instances with a total of 105 questions in *DB-PLAS* by applying them to undergraduate students who take the database programming course in the Technology and Information Faculty with three majors as follows: Data and Information Management (*MDI*), Computer Systems (*SK*), and Business Accounting Computer (*KAB*).

4.3.1 Overview

For evaluation, we generated 14 *GUP* instances with 105 questions from various source codes covering the basic database programming topics that have been adapted to the curriculum. It has been verified that the questions generated are appropriate for novice-level students. Then, we asked 90 undergraduate students tasked with resolving the given problems using the answering functionality.

4.3.2 Evaluation of All Instances by First-year students

1. *Summary of Results:*
   Table 4.2 shows the majors, the number of students who answered the *GUP* task, the average correct rate and its standard deviation (*SD*), the average number of submission times and its *SD* among the 14 *GUP* instances. From the table, we can know that the *MDI* students can solve the generated *GUP* instances correctly from the high average correct answer rate 94.93%. The average submission times is 1.44, which means the students answered any *GUP* instance just one or two times.

2. *Results for Individual GUP Instances:*
   Figure 3.2illustrates the average correct rate for the *GUP* and the average number of answer

Table 4.2: Results for all GUP instances by First-year students.

majors	# of students	correct rate (%)		# of submission	
		ave	SD	ave	SD
MDI	30	95.01	0.02	1.44	0.09
SK	30	94.90	0.02	1.46	0.11
KAB	30	93.90	0.02	1.76	0.18

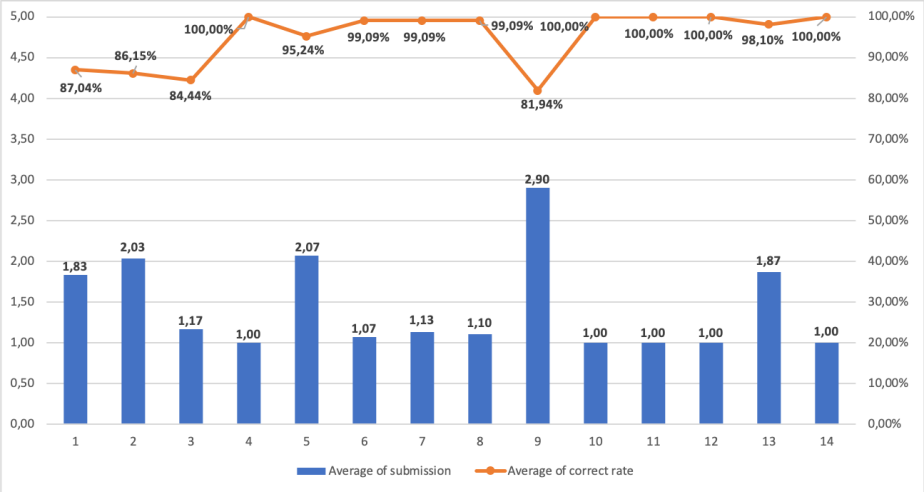

Figure 4.3: Solution performance for each of 14 GUP instances by first-year students with MDI major.

submission times by the 30 students in each *GUP* instance. The graph indicates that the instances at `ID=3` and `ID=9` exhibit the low correct rates, which include update a table and date function as the topic of instances. The instance at `ID=9`shows the highest submission times. These instances are hard for novice students.

Figure 4.4 and Figure 4.5 also illustrate the same results as Figure 4.3. However, the graph indicates that the instances at `ID=3` and `ID=9` exhibit the low correct rates and the instance at `ID=9` shows the highest submission times.

3. *Results for Individual Students:*
Figure 4.6 illustrates the average correct rates and the average number of submission times of all the *GUP* instances for each of the 30 first-year students. The graph indicates that 16 among 30 students achieved the average correct rate above 95%. The best student at `ID=11` could solve all the instance with 98.02% correctly with only 1.43 submissions for each instance. However, the worst student at `ID=2` could solve only 92.07% of the questions. Six instances were not solved at all, whereas the remaining instances were fully solved with 100%. This student needs more efforts in studying database programming seriously. The student at `ID=10` has the highest submission times but achieves more than the 95% correct rate, which means this student studied it very seriously.

   Figure 4.7 illustrates the average correct rates and the average number of submission times of all the *GUP* instances for each of the 30 first-year students. The graph indicates that 15 among 30 students achieved the average correct rate above 95%. The best student at `ID=6` could solve all the instance with 98.16% correctly with only 1.36 submissions for each

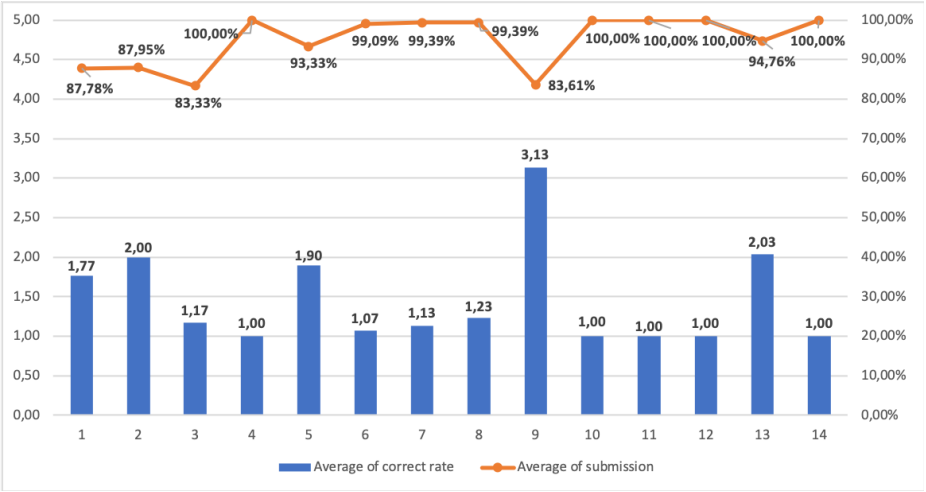

Figure 4.4: Solution performance for each of 14 GUP instances by first-year students with SK major.

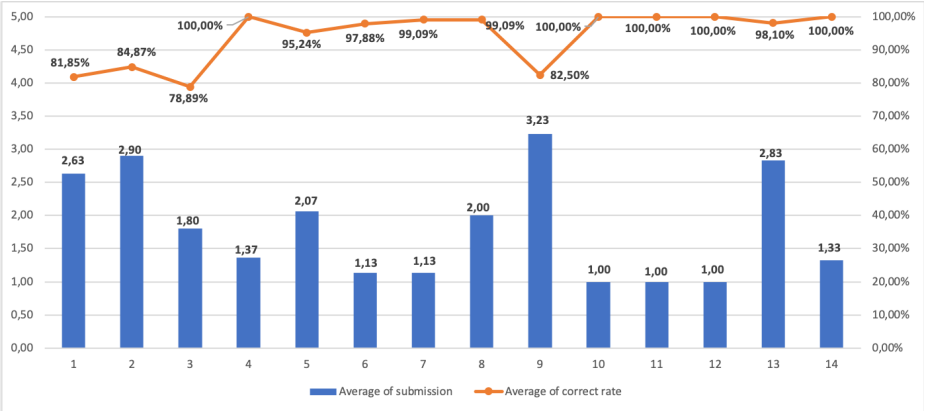

Figure 4.5: Solution performance for each of 14 GUP instances by first-year students with KAB major.

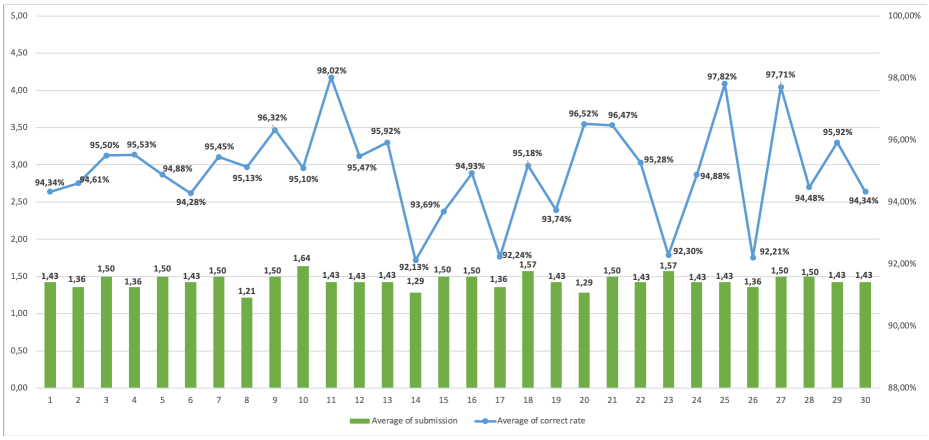

Figure 4.6: Solution performance for each of 30 students from MDI major

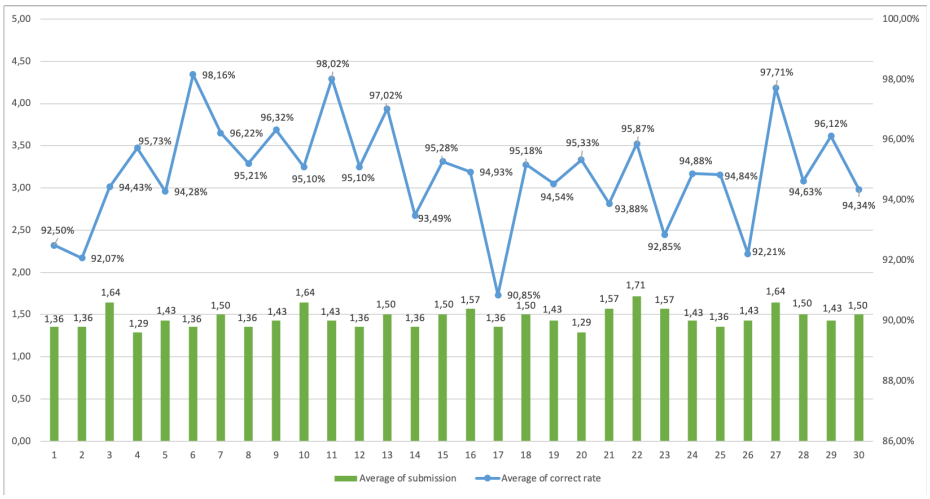

Figure 4.7: Solution performance for each of 30 students from SK major

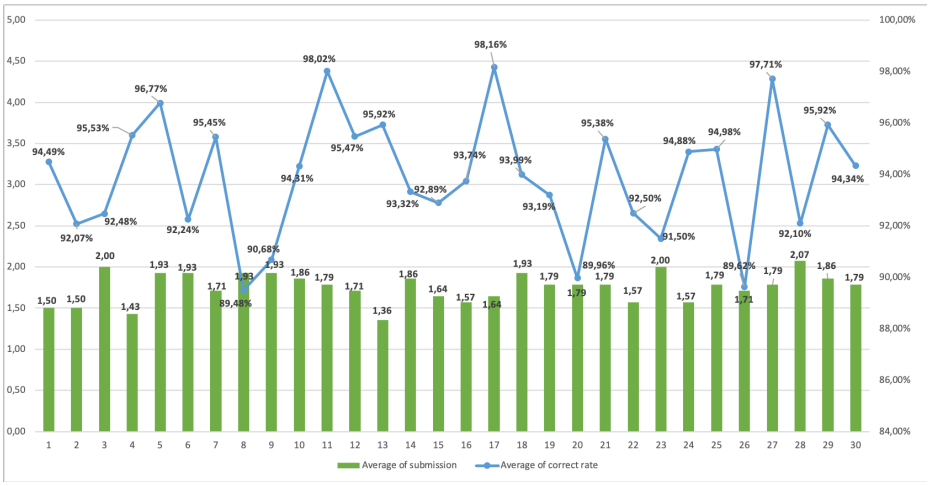

Figure 4.8: Solution performance for each of 30 students from KAB major

instance. However, the worst student at `ID=2` could solve only 92.07% of the questions. Six instances were not solved at all, whereas the remaining instances were fully solved with 100%. This student needs more efforts in studying database programming seriously. The student at `ID=22` has the highest submission times but achieves more than the 95% correct rate, which means this student studied it very seriously.

Figure 4.8illustrates the average correct rates and the average number of submission times of all the *GUP* instances for each of the 30 first-year students. The graph indicates that 16 among 30 students achieved the average correct rate above 95%. The best student at `ID=11` could solve all the instance with 98.02% correctly with only 1.43 submissions for each instance. However, the worst student at `ID=2` could solve only 92.07% of the questions. Six instances were not solved at all, whereas the remaining instances were fully solved with 100%. This student needs more efforts in studying database programming seriously. The student at `ID=10` has the highest submission times but achieves more than the 95% correct rate, which means this student studied it very seriously.

Table 4.3: Correct rate answer distribution of students

range of correct answer rate (%)	#of students			rate of students (%)		
	MDI	SK	KAB	MDI	SK	KAB
85 - 90	0	0	4	0	0	13.33
91 - 95	14	15	16	46.67	50	53.33
96 - 99	16	15	10	53.33	50	33.33
100	0	0	0	0	0	0

4.3.3 Result Distribution of Students

1. *Correct Answer Results:*
   Describes the distribution of the number of correct answers given by students. Based on the data presented in the table 4.3, it can be observed that out of a total of 90 students from 3 majors, none achieved a correct answer rate of 100% but as many as 86 students achieved a score above 90%. This shows that most students have sufficiently understood the *GUP* concepts needed for database programming even though they were given a time limit of only 90 minutes for the final semester exam. On the other hand, there were four students from the *KAB* major who answered below a score of 90%. Students in all three majors can benefit from refreshing their memories and reviewing database programming concepts and principles to enhance their understanding and improve their accuracy in the future.

   In summary, this table provides insight into the distribution of correct answers among students. These insights emphasize the need for further improvement and strengthening for all students to achieve 100% score.

2. *Submission Times Results:*
   The distribution of the number of submission times provides insights into how frequently students are submitting their answers. It helps identify patterns and trends in their engagement with the given tasks, highlighting the range of effort and dedication exhibited by the students in the learning process.

   Table 4.4 provides an illustration of the distribution of the number of times students submitted answers. 60 students in the *MDI* and *SK* majors (100%) submitted their answers an average of 1 to 2 times per example, while 28 students in the *KAB* major (98%) submitted their answers an average of 1 to 2 times per instance. They are confident in their initial responses and require minor adjustments or revisions to perfect their answers. Two students in the *KAB* major (6.67%) submitted their answers on average 2 to 3 times per instance but were able to achieve a correct answer rate score above 90%. This means that the documents undergo multiple cycles of review and modification to improve understanding, accuracy, and thoroughness. This shows that these students approached the *GUP* activity with serious commitment, submitting answers more than once even though they were given a time limit of only 90 minutes. This shows a strong dedication to continually improving and achieving the correct answer.

Table 4.4: Submission times distribution of students

range of submission times	#of students			rate of students (%)		
	MDI	SK	KAB	MDI	SK	KAB
0 - 1	0	0	0	0	0	0
1.1 - 1.99	30	30	28	100	100	93.33
2 - 3	0	0	2	0	0	6.67

4.4 Summary

This work addressed the Grammar-Concept Understanding Problem (GUP) in SQL-Python for novice learners. We developed 14 GUP instances covering 105 essential keywords and evaluated them with 90 undergraduate students at INSTIKI. The results confirm the strategy's effectiveness in precisely assessing student understanding.

Chapter 5

Implementation of Comment Insertion Problem for Database Programming Learning Assistant System

This chapter presents the *comment insertion problem (CIP)* in Database Programming Learning Assistant System [17].

5.1 Introduction

A *CIP* instance consists of source code with blank comments, a set of comments to fill in the blanks, and their correct answers. Again, a student's answer is marked by the *string matching*.

5.2 Preparation of Fill-in-the-Blank Questions

5.2.1 Course Outline

The experiment involved 60 first-year students at the Indonesian Institute of Business and Technology, majoring in *Data and Information Management (MDI)* or *Computerized Business Accounting (KAB)*. They studied database programming using SQL in the core database course, which consisted of 16 weekly 180-minute classes per semester. The database curriculum spanned two semesters: the first covered the basic theory of databases, while the second included a practicum using *SQL-Python*. The course was taught by one instructor, who delivered lectures and exercises, and supported by one teaching assistant for the practicum.

Table 5.1 represents the course outline in the second semester. We introduced *SQL-Python PLAS* to the students in the middle of July 2024 as a final examination.

5.3 Comment Insertion Problem for SQL-Python

In this section, we present the *comment insertion problem (CIP)* for learning *SQL-Python*.

Table 5.1: Course Outline

Week	Contents
1	Data definition language
2	Data definition language
3	Data manipulation language
4	Data manipulation language
5	Join tables (inner, left, and right join)
6	Join tables (cross, self, and full outer join)
7	Character and numeric functions
8	Mid examination
9	Date and aggregate functions
10	Order By, Group By, and Having
11	Set operation
12	Operator
13	Sql distinct
14	Sql As Alias
15	Sql view
16	Final examination

5.3.1 Source Code with Comments

The source code for a *CIP* instance must have multiple comments that adequately explain the essence or meaning of the corresponding steps of the code. If the selected source code has insufficient comments, the teacher should properly add comments to the code. Each code block should have one comment explaining the procedure so a novice student can easily understand it. Then, every comment is removed and blanked in the *CIP* instance. By requesting to fill in every blank with the relevant comment in the provided source code, a student is expected to comprehend all the blocks of the *SQL-Python* source code.

5.3.2 CIP Generation Procedure

The answer interface for a new *CIP* instance can be generated through the following procedure:

1. Collect a *SQL-Python* source code for students to study.
2. Add comments to the source code properly if current ones are insufficient.
3. Remove and blank every comment in the source code.
4. Save each comment as the correct answer to each blank.
5. Generate the *HTML/CSS/JavaScript* files for the answer interface on the web browser by running the generator with this text.
6. Add the problem statement and the answer options to the *HTML* file.

5.3.3 CIP Instance Interface

Figure 5.1 displays the list of the 18 *CIP* instances along with the comment. A green highlight indicates that if the learner can answer every question. If the remark state is still "tried" with a yellow highlight, the student has not answered all of the questions in the instance correctly; if all of the instances are answered successfully, the remark status is "completed." No highlights appear if the students has not attempted any instances.

Problems

No	Problem Name	Remark
1	Connection and Create a Database	Completed
2	Create a Table	Tried
3	ALTER Table	
4	Update, Delete, and Truncate a Table	
5	Join Tables	
6	Insert New Rows and Show Records	
7	ORDER BY	
8	Character Functions	
9	Numeric Functions	
10	Date Functions	
11	Aggregate Functions	
12	Group By and Having	
13	Set Operation	
14	Operator	
15	SQL Distinct	
16	SQL AS Alias	
17	SQL View	
18	SQL Constraints	

Figure 5.1: *CIP* instance interface.

5.3.4 CIP Answer Interface

Figure 5.2 displays the web-based *answer interface* for a sample *CIP* instance. The upper section presents the source code with placeholders for comments, while the lower section provides options for selecting appropriate comments. Students must insert the correct comment into each placeholder by choosing one option.

5.4 Usability

We conducted an *SUS* questionnaire for the students after the experiment. John Brooke invented the *SUS* in 1986, creating this 'quick and dirty' usability scale to evaluate practically any kind of system. Table III represents the *SUS* questions for students with the 5-point Likert scale.

In the *SUS*, a response point is the score a user assigns to each of the ten questionnaire items, rated on a scale from 1, meaning strongly disagree, to 5, meaning strongly agree, and these scores are adjusted and summed to calculate the final usability score.

To calculate the *SUS* score, each response point from the ten *SUS* questions, rated from 1 for strongly disagree to 5 for strongly agree, is adjusted by subtracting 1 for odd-numbered items and

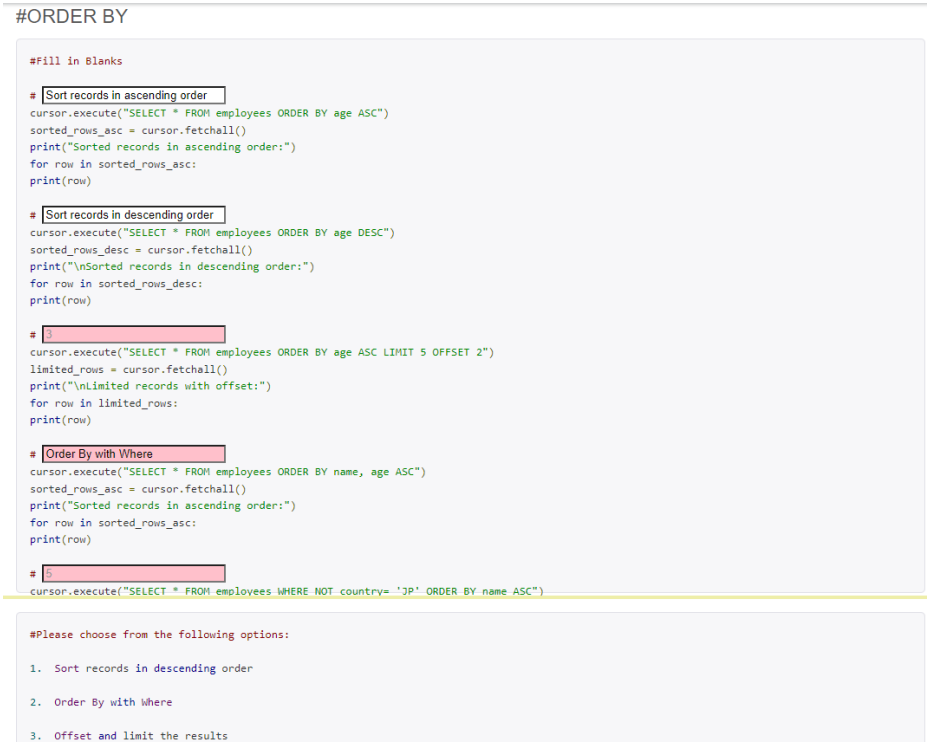

Figure 5.2: *CIP* answer interface.

subtracting the response from 5 for even-numbered items, then the adjusted scores are summed and multiplied by 2.5 to produce a final usability score ranging from 0 to 100.

For example, if a user gives a response point of 4 for item 1, it becomes 4 - 1 = 3. If they give a response point of 3 for item 2, it becomes 5 - 3 = 2. These adjusted points contribute to the total score.

To determine the System Usability Scale assessment grades, Table IV assigns grades based on percentile ranks, where a score of 80.3 or higher receives an A, a score from 74 to 80.2 receives a B, a score from 68 to 73.9 receives a C, a score from 51 to 67.9 receives a D, and a score below 51 receives an E, while Table V categorizes user acceptance, with scores from 0 to 50.9 deemed not acceptable, scores from 51 to 70.9 considered marginal, and scores from 71 to 100 rated as acceptable.

5.5 Evaluation

In this section, we evaluate the proposal through applications to novice students of *SQL-Python*.

5.5.1 Evaluation Setup

For evaluations, we made 18 *GUP* and 18 *CIP* instances using *SQL-Python* source codes for its basic concepts. These were assigned them to 60 first-year undergraduate students taking the database programming course in two majors, *Data and Information Management (MDI)* and *Computerized Business Accounting (KAB)*, at the *Indonesian Institute of Business and Technology* as the final examination in 180 minutes. Then, we analyze the correct answer rates and the number of submission times by the students for verifications. Table 5.5 shows the topic, and the number of comments

Table 5.2: The *SUS* Standard Scale

	The System Usability Scale Standard Version	Strongly Disagree 1 2	3	Strongly Agree 4 5
1.	I think that I would like to use this system frequently.			
2.	I found the system unnecessarily complex.			
3.	I thought the system was easy to use.			
4.	I think that I would need the support of a technical person to use this system.			
5.	I found the various functions in this system were well integrated.			
6.	I thought there was too much inconsistency in this system.			
7.	I would imagine that most people would learn to use this system very quickly.			
8.	I found the system very cumbersome to use.			
9.	I felt very confident using the system.			
10.	I needed to learn a lot of things before I could get going with this system.			

Table 5.3: *SUS* Score Percentile Rank for Assessment Grades

Grade	Note
A	Score >= 80.3
B	Score >= 74 and <80.3
C	Score >= 68 and <74
D	Score >= 51 and <68
E	Score <51

Table 5.4: Acceptability Ranges for User Acceptance

SUS Score	Note
0 - 50.9	Not acceptable
51 - 70.9	Marginal
71 - 100	Acceptable

for *CIP*.

5.5.2 Result Summary

Tables 5.6 show the number of students, the average correct answer rate and standard deviation (SD), the average number of submission times, and its SD in each major for*CIP*, respectively. It's suggests that they were similar for *CIP*. These students solved *CIP* very well with the 100% average correct answer rate and the 1.57 and 1.67 average submission times.

5.5.3 Results for CIP Individual Instances

The results of the *CIP* individual instances are shown in figure 5.3 which consist of results for submission times and correct answer rates.

5.5.3.1 Submission Times

Figure 5.3 depicts the average number of submissions time for each *CIP* instance by *MDI* and *KAB* students, respectively. The results indicate that *MDI* students solved six instances (ID=1, 3, 5, 11,

Table 5.5: Overview of *GUP* and *CIP* instances

ID	topic	# of CIP comments
1	connection and create database	7
2	create a table	7
3	alter table	6
4	update, delete, and truncate a table	8
5	join tables	6
6	insert new rows, and show records	7
7	order by	5
8	character function	11
9	numeric function	10
10	date function	12
11	aggregate function	5
12	group by and having	2
13	set operation	3
14	operator	5
15	sql distinct	5
16	sql as alias	5
17	sql view	7
18	sql constraint	7

Table 5.6: Result summary for *CIP*.

majors	#number of students	correct rate (%)		#of submission	
		ave	SD	ave	SD
MDI	30	100	0.00	1.57	0.14
KAB	30	100	0.00	1.67	0.08

12, 13) with a single submission, while *KAB* students solved five (ID=1, 3, 5, 11, 12). The *CIP* instances at ID=4 and ID=6 have the highest submission times, suggesting that some *MDI* students have difficulty solving it. For *KAB* students, *CIP* instances at ID=4 and ID=10 are suggested as the most difficult instances. *MDI* students achieved an average submission times score of 1.57, lower than *KAB* students at 1.79. This shows that *MDI* students can solve all *CIP* examples more quickly.

5.5.3.2 Correct Answer Rates

Figure 5.3 shows that both *MDI* and *KAB* students achieved a 100% correct answer rate across all *CIP* instances.

5.5.4 Results of Individual Students in MDI

Figure 5.4 illustrates the average correct answer rate and the average number of submission times of the *CIP* instances for each of the 30 first-year students in the *MDI* major.

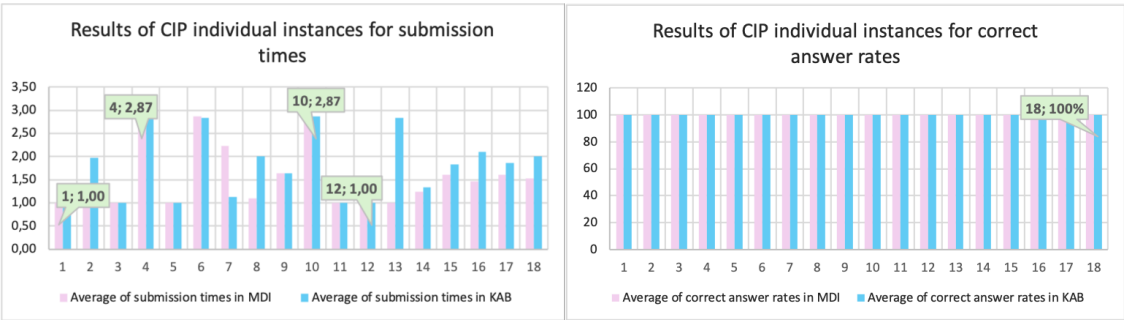

Figure 5.3: Results of *CIP* individual instances by students for submission times and correct answer rates.

For *CIP*, all students achieved the 100% average correct rate. All the students needed more than one submission time to complete the *CIP* instances, with an average of 1.78 submissions for each student.

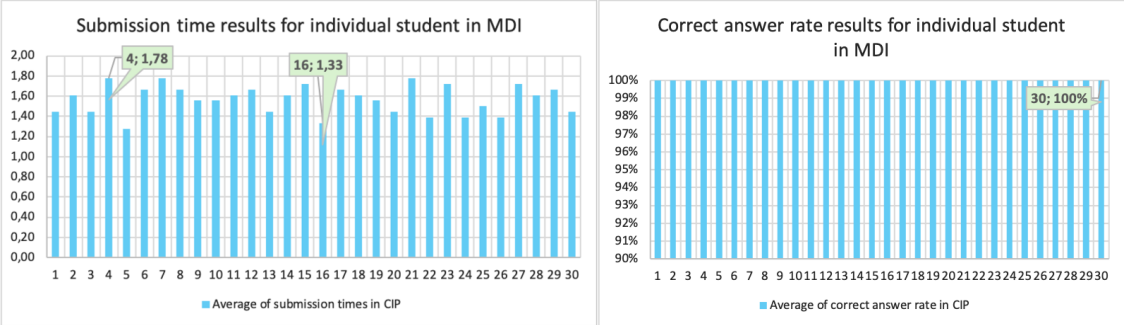

Figure 5.4: Results of individual students in MDI.

5.5.5 Results of Individual Students in KAB

Figure 5.5 illustrates the average correct answer rates and the average number of submission times of the *CIP* instances for each of the 30 first-year students in the *KAB* major.

For *CIP*, the 30 students achieved the 100% average correct rate. The best student at ID=13 and ID=16 could solve all instances correctly with one single submission for any instance. The worst students at ID=1 and ID=16 could solve them with 1.22 submissions on average.

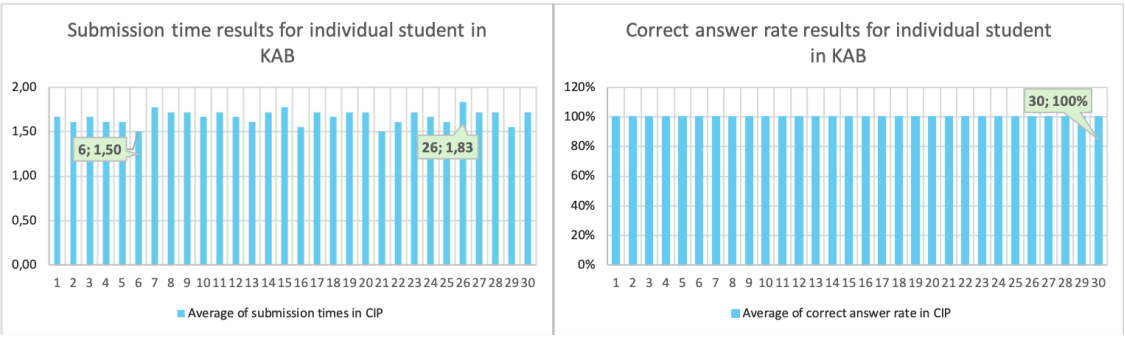

Figure 5.5: Results of individual students in KAB.

5.5.6 Comparison of Correct Answer Rates between Problems

Tables 5.7 compare the distribution of the number of correct answer rates given by the students. All students in both majors achieved a 100% score in 180 minutes on the final semester exam. The students found it easy to solve the *CIP* questions by checking the provided answer choices.

Table 5.7: Comparison of correct answer rates in *CIP*.

range of correct	# of students		rate (%)	
answer rate (%)	MDI	KAB	MDI	KAB
85-90	0	0	0	0
91-95	0	0	0	0
96-99	0	0	0	0
100	30	30	100	100

5.5.7 Comparison of Submission Times between Majors

Tables 5.8 compare the distribution of submission times between the two majors. For instance, all the students in *MDI* submitted correct answers more than once, whereas all the students in *KAB* did. These students were confident in their initial answers and might require small adjustments or revisions to perfect them.

Table 5.8: Comparison of submission times between majors in *CIP*

range of	# of students		rate of students (%)	
submission time	MDI	KAB	MDI	KAB
0-1	0	0	0	0
1.1-1.99	30	30	100	100
2-3	0	0	0	0

5.5.8 Student Opinion

A survey was conducted among students enrolled in the database programming module to assess their perceptions of self-study exercises. Over 85% of students valued the *CIP* exercises for introducing entry-level *SQL-Python*, appreciating the ability to progress at their own pace. These exercises allowed students to practice independently before the final exam, receive immediate feedback on their performance, and consult instructors during class if they encountered difficulties. Most experienced an increase in their grades on the final exam. When asked about their thoughts, students expressed no preference for traditional classes over self-study exercises. This suggests that both methods are acceptable for learning and exercise.

5.5.9 Analysis of SUS Questionnaire to the Students

The results of the usability test in Figure 5.6 were carried out step by step in accordance with the *SUS* calculation guidelines. The final *SUS* score from 30 respondents' responses was 84, in

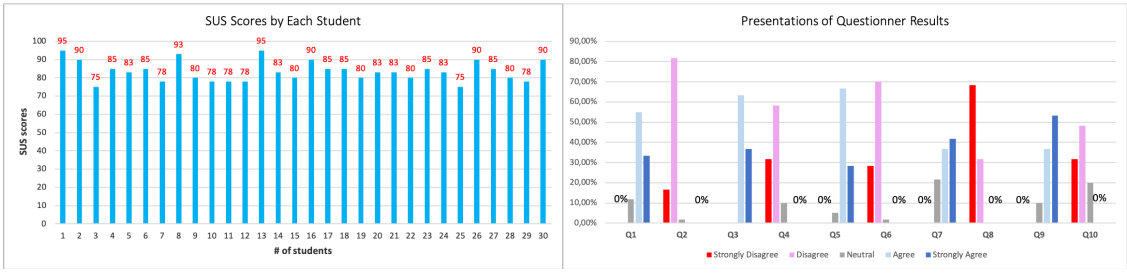

Figure 5.6: *SUS* Results.

accordance with the *SUS* interpretation guidelines in Table 5.4. The score of 84 was interpreted as follows:

1. Interpretation with acceptability range. Referring to Table 5.4, the score of 84 is included in the Acceptable range.

2. Interpretation using the Grade scale as in Table 5.3. The score of 84 is included in grade A.

Figure 5.6 presents the percentage of responses to all respondent statement items regarding the questionnaire that was distributed. There are minor problems that occur from the results of the tests that have been carried out, namely:

1. The percentage in the even statements, namely Q2, Q4, Q6, Q8 and Q10, is 0%, which means that all respondents do not find it complicated to use the system, they do not need a technician to use the system, the system is quite consistent for respondents, not confusing and respondents quickly adapt to using the system.

2. In statement 1, 11.7% of respondents were hesitant to use this system.

3. In statement 5, 5% of respondents doubted that the system would work properly.

4. In statement 7, 21.7% of respondents were doubtful that other people would quickly understand how to use the system.

5. In statement 9, 10% of respondents doubted that there would be no obstacles to using the system.

5.6 Summary

This chapter introduced the comment insertion problem (CIP) as a first-step self-study task for entry-level SQL–Python. Eighteen *CIP* instances based on basic source codes were given to 60 students at the Indonesian Institute of Business and Technology, and the results showed that *CIP* is suitable for beginners in database programming. Usability testing of the SQL–Python PLAS using the *SUS* yielded a score of 81 (grade A), and more than 85% of students appreciated *CIP* for entry-level self-study; many reported improved final exam scores and expressed no strong preference between traditional classes and self-study exercises, indicating that both approaches are acceptable for learning.

Chapter 6

An SQL Query Description Problem with AI Assistance for Database Programming Learning Assistant System

This chapter presents the SQL Query Description Problem with AI Assistance for Database Programming Learning Assistant System [56].

6.1 Introduction

Today, *relational database* takes central roles in information systems by managing large volumes of data efficiently. Then, *database programming* using *Structured Query Language (SQL)* has become a core skill for data scientists and software developers [1]. A lot of universities and professional schools offer courses related to *SQL* in their curricula, teaching data definitions, retrievals, and manipulations [18, 20, 19, 21, 22]. However, despite its importance, *database programming* is not offered in many universities, and novice students often struggle to write correct *SQL* queries because of their complexity, making both syntactic and semantic mistakes that hinder learning progress [23, 24, 25].

The difficulties faced by novice students can be effectively addressed through the lens of *Cognitive Load Theory (CLT)*. Novice learners often struggle with high *intrinsic cognitive load* stemming from the inherent complexity of *SQL* logic, which is frequently compounded by the *extraneous cognitive load* associated with configuring and managing complex database environments. To facilitate effective learning, it is crucial to minimize these extraneous demands and manage intrinsic difficulty through *scaffolding*, thereby freeing up working memory resources for *germane load*—the active construction of knowledge schemas. Guided by these principles, this study aims to optimize the learning process by providing a streamlined, web-based environment and structurally sequenced exercises.

To address these cognitive load challenges in *SQL* learning we have developed a web-based *Programming Learning Assistant System (PLAS)* to support self-study of popular programming languages such as *Java*, *Python*, and *JavaScript*. *PLAS* is designed and implemented to help novice students study programming by themselves through solving various types of exercises. Among them, we implemented the *grammar-concept understanding problem (GUP)* and the *comment insertion problem (CIP)* for learning keyword meaning and basic syntax of *SQL programming* as its initial studies. These exercises may help students in initial recognitions and grammar under-

standing. However, they do not provide realistic practices in composing *SQL* queries for database tables.

To provide realistic practice, instructors encounter challenges when designing varied *SQL* programming exercises. This task typically entails constructing diverse database schemas and crafting matching queries, which is time-consuming and often repetitive [7]. Recent studies show that automatic schema generation by generative AI such as *ChatGPT-3.5* can produce reasonable cores but often inconsistent relationships and redundancy, indicating the need for careful validation [7]. Thus, teachers need tools that can reduce manual works with ensuring schema correctness and pedagogical values.

In this paper, we propose an *SQL query description problem (SDP)* as a new exercise type inside *PLAS*. An *SDP* instance contains a database table schema and a set of questions that require students to write *SQL queries* for data retrieval or manipulation with their correct answers. Student answers are evaluated automatically using enhanced string matching against a bank of canonical solutions that includes multiple semantically equivalent variants. To reduce preparation efforts and increase variety, we also propose a *generative AI-assisted SQL query generator* that combines large human-labelled schema collections called *Spider dataset* [27] with popular *generative AI models* such as *ChatGPT 5.0* [28] and *Gemini 3.0 Pro* [29] to produce candidate schemas, questions, and reference *SQL* queries to the dataset.

For evaluations, we generated 11 *SDP* instances on fundamental *SQL* topics using the *generator* with the two generative AI models. The correctness of them was manually validated by teachers before use. We found that *ChatGPT-5.0* demonstrated superior technical efficiency with fewer execution errors and a lower mean error rate, making it highly effective for rapid query generation. However, *Gemini 3.0 Pro* emerged as the more consistent model for pedagogical purposes, achieving perfect scores in *Sensibleness*, *Topicality*, and *Readiness* with absolute inter-rater consensus (κ = `1.00`). While *Gemini 3.0 Pro* required slightly more manual syntax debugging than *ChatGPT-5.0*, its superior instructional alignment and logical stability make it more suitable for the standardized requirements of this study.

Then, we assigned the *SDP* instances to 32 undergraduate students at the *Indonesian Institute of Business and Technology (INSTIKI)* during a 120-minute class. Our analysis of student performances showed that the average correct answer rate was `95.2`% and the average number of submissions was `1.56` per question. The usability test using a questionnaire yielded `78` (Grade B) for the average *SUS* score. These results indicate that the proposed *SDP* with the *AI-assisted generator* using *Gemini 3.0 Pro* can be a reliable solution for *SQL* programming studies by novice students.

6.1.1 Contributions of This Study

To clarify the novelty of this work, the unique contributions of this study compared to prior research in the PLAS ecosystem and existing SQL learning tools are as follows:

1. Introduction of SDP: We propose the *SQL Query Description Problem* (SDP) as a new exercise type that focuses on logical-semantic mapping, bridging the gap between natural language requirements and *SQL* syntax.

2. LLM-based Automation: We develop a structured prompting framework that enables the automated generation of diverse *SQL* exercises, significantly reducing manual effort for instructors compared to previous manual methods used in *GUP* and *CIP*.

3. Enhanced Assessment Methodology: We introduce an enhanced string-matching logic for the automated evaluation of student answers. This methodology accounts for multiple semantically equivalent canonical solutions and normalizes syntactic variations, ensuring a robust and reliable assessment without the need for manual grading.

4. Comparative Generative AI Evaluation: We provide a detailed comparative evaluation of *Gemini 3.0 Pro* and *ChatGPT-5.0* specifically for generating educational *SQL* content, using metrics that ensure pedagogical suitability.

5. Educational Feasibility: We demonstrate the practical utility of the system through an empirical study involving 32 students, confirming system acceptance and task-solving performance.

6.2 Software and Dataset

In this section, we describe the main software tools and datasets used in the proposal.

6.2.1 Generative AI

Generative Artificial Intelligence (AI) models, particularly those based on the Transformer architecture introduced in 2017 [30], have demonstrated exceptional capabilities in natural language processing and content creation. This presents a significant opportunity for automation and *AI-powered* content creation in education, which is leveraged in this study to reduce the workload of teachers. The *LLM models* used in this study are:

OpenAI's GPT-5: Launched in August 2025, this model offers advanced language understanding, multi-level reasoning, and programming support [28]. A related model *ChatGPT-3.5* has shown an 87% accuracy rate in generating and correcting *SQL* syntax [31].

Google's Gemini 3.0 Pro: Released in November 2025, this multimodal generative *AI* model is known for its strong reasoning, comprehension, and coding capabilities, capable of handling complex problems and supporting contexts up to one million tokens [29]. A previous version *Gemini 1.0 Pro* has shown an 80% accuracy rate in *SQL* syntax generation [31].

6.2.2 Spider Dataset

Spider dataset is a major breakthrough in the fields of *natural language processing (NLP)* and *text-to-SQL* system developments, designed to handle complex, cross-domain semantic parsing tasks. A team in Department of Computer Science at Yale University created the *Spider dataset*, which includes 10,181 natural language questions and 5,693 unique *SQL* queries. They are carefully labelled by 11 undergraduate students, covering 200 databases with many tables from 138 different domains like education, aviation, and entertainment [27]. *Spider dataset* is unique because it uses different databases for training and testing, which allows a *generative AI* model to adapt to new structures and questions, unlike older datasets like *ATIS* or *WikiSQL* that usually focus on matching patterns within one area. With a state-of-the-art model accuracy of only 12.4% in the "split database" setting, *Spider* offers a new challenge that emphasizes the importance of true semantic understanding, rather than rote memorization, and is publicly available at https://yale-lily.github.io/spider to support [27].

From 200 databases with diverse domains, we selected *college_2* database schema. The database schema for *college_2* consists of 11 tables and approximately 30,000 data entries. The domain is familiar for teachers and students, which is relevant to daily life and educational environments [32]. Teachers can utilize this dataset to illustrate practical applications of *SQL* in educational data analysis, while students can learn to interpret realistic database schemas and complete challenging tasks, thereby enhancing their understanding of *SQL* programming and data processing.

6.3 Proposal of SQL Query Description Problem

In this section, we present the *SQL query description problem (SDP)* with a *generative AI-assisted SQL query generator*.

6.3.1 Overview

SDP with the generator is designed to help teachers generate practical *SQL* exercises efficiently and allow students to practice query writing interactively. Figure 6.1 illustrates the workflow of the proposed *SDP*.

First, teachers use the generator to automatically create new *SDP* instances. Teachers begin it by selecting a learning topic such as, `SELECT`, `UPDATE`, or `JOIN` and choosing a preferred *generative AI model* such as *ChatGPT 5.0* or *Gemini 3.0 Pro*. Based on these inputs, the generator generates pairs of questions and their corresponding *SQL* queries by applying prompt engineering techniques to the model.

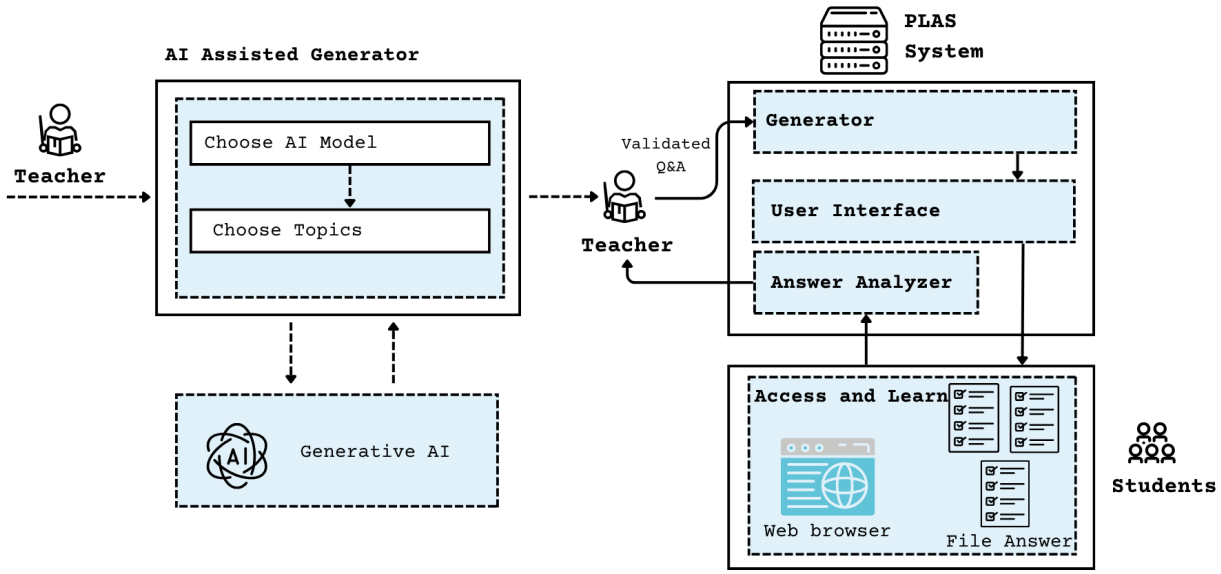

Figure 6.1: Overview of proposal.

The generated question–answer pairs should be checked by teachers for syntactic correctness and query running ability within a *MySQL* environment. Only questions that pass this verification can be included for further reviews. Teachers then perform manual validations to ensure that the generated contents align with the course materials and the intended difficulty levels. Once validated, these contents are exported into *SDP* instance files, ready to be uploaded into *PLAS*.

In this paper, we extended the existing instance generator of *PLAS* to support the *SQL*-specific structure in *SDP*, using the available *SQL* keywords as a reference. Each validated *SDP* instance consists of a database schema, multiple question statements, and their correct query answers. Students access to these *SDP* exercises through the *PLAS* user interface on a browser.

6.3.2 Generative AI-assisted SQL Query Generator

The *generative AI-assisted SQL query generator* is developed to support teachers in producing diverse and practical *SDP* instances. It leverages *generative AI* models to automatically generate question–answer pairs based on real database schemas. Teachers can use this generator through a simple graphical interface, as shown in Figure 6.2. They need to select a desired *SQL* topic such as `SELECT`, `UPDATE`, or `JOIN` and specifying the *generative AI* model to use. The generator then creates corresponding a set of *SQL* exercises that consist of question statements and correct *SQL* queries.

Once teachers start the generator, it prepares prompts as shown in Table 7.1 including the schema information and topic instructions listed in Table 6.7. The generated results of *question-answer pairs* are automatically checked in syntax correctness and running ability on the target *MySQL* database system. Only the pairs that are executed successfully are saved for teacher validations. Then, teachers should review, edit, or discard generated contents before uploading them to *PLAS*. This workflow allows for efficient and semi-automated content creations while maintaining human oversights on accuracy and pedagogical consistency.

6.3.2.1 Question-Answer Pair Generation

The *question-answer pair* generation process relies on *Large Language Models (LLMs)*, specifically *ChatGPT 5.0* and *Gemini 3.0 Pro*, chosen for their strong performances in text generations and structured query formulations. These models can interpret natural language prompts and produce consistent *SQL* query statements that follow the *MySQL* syntax rules.

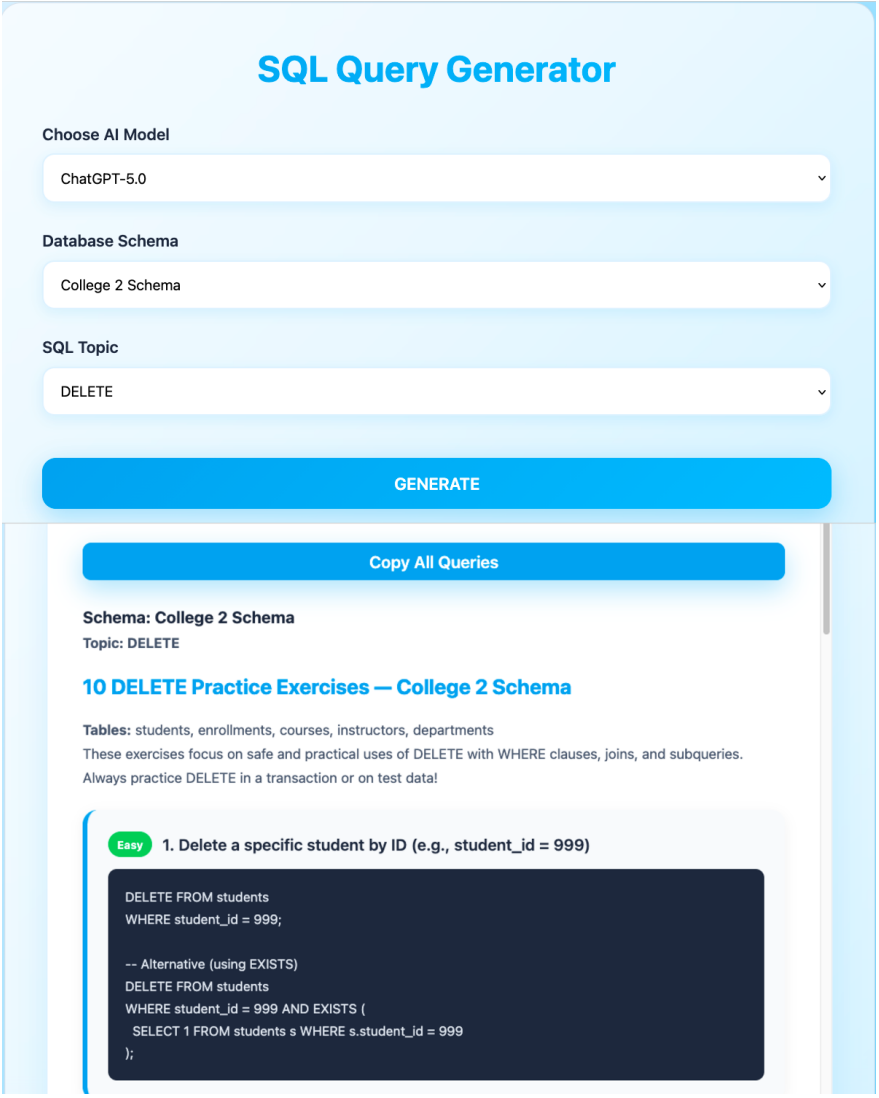

Figure 6.2: *Generative AI-assisted SQL query generator* interface.

For the input context, we use *Spider dataset* as a well-known human-labelled dataset for cross-domain *SQL* generations developed by Yale University [27]. This dataset provides database schemas and query examples across multiple domains, allowing *generative AI* models to generate realistic questions and accurate queries at various difficulty levels.

Table 7.1 illustrates the design of a standardized prompt structure used to ensure clarity and output consistency. The prompt instructs the selected generative AI model to produce enhanced question–answer pairs, where each question is accompanied by multiple semantically equivalent *SQL* queries and explanations. This standardized prompting ensures that each generated content includes a mix of basic and intermediate *SQL* operations, promoting balanced difficulty and comprehensive learning coverage.

6.3.2.2 Generated Results

Table 6.2 presents examples of generated content. In this work, we redefine the traditional concept of a "question-answer pair" to accommodate the flexibility of *SQL* syntax. Instead of a single reference answer, each generated instance functions as a *question-solution set*, consisting of one

Table 6.1: Prompt template to *generative AI* model to generate *SQL* question–answer pairs.

Prompt Template:
`"Create 10 SQL questions based on the attached database schema, focusing on [topic], with two levels of difficulty. For each question, provide an explanation and multiple alternative query formulations that are semantically equivalent and yield the same results."`

natural language question and multiple semantically equivalent valid queries.

These examples demonstrate the system's capability to produce clear, executable *SQL* queries aligned with educational topics.

Table 6.2: Examples of generated question-answer pairs with alternative queries.

Level	**Question**	**Answer (*SQL Query*)**
Easy	How can you retrieve the names of all departments from the `department` table?	Option 1: `SELECT dept_name FROM department;` Option 2: `SELECT d.dept_name FROM department AS d;`
Medium	Write a query to get the names of students (`student.name`) and the courses they are taking (`course.title`). Use the `student` and `takes` tables.	Option 1: `SELECT s.name, c.title FROM student s JOIN takes t ON s.ID = t.ID JOIN course c ON t.course_id = c.course_id;` Option 2: `SELECT s.name, c.title FROM student s, takes t, course c WHERE s.ID = t.ID AND t.course_id = c.course_id;`

Teachers then perform manual validations to confirm that the generated pairs aligned with course objectives and the corresponding schema design. Only validated pairs are subsequently registered as *SDP* instances in *PLAS*, where students can access to them for self-learning and practices.

6.3.3 SDP Instance Generation Program

The *answer instance generator* for *PLAS* was modified to support the structure and logic of *SQL Query Description Problem (SDP)*, to create new *SDP* instances that align with *SQL* query constructions using predefined *keywords* and *question templates*. Each keyword corresponds to a specific concept in *SQL database programming* and follows the rules of *Data Manipulation Language (DML)* and *data retrieval* operations.

The answer interface generation process ensures that the created instances reflect the real syntax and operations of *SQL*. Table 6.2 illustrates how a *DML* operation, such as `INSERT`, is used to build an *SDP* instance. A teacher selects the keyword, defines its question, and registers both in *PLAS* as a new instance.

In addition to *DML*, the generated *SDP* instances also include exercises for *Data Retrieval*. They focus on querying and analysing information using commands such as `JOIN`, `ORDER BY`, and `GROUP BY`. Table 6.3 shows some of them. These additional ones are made so that students will practice writing *SQL* queries that cover both data manipulation and complex retrieval logic, reinforcing their understanding of query syntax and data relationships.

Table 6.3: Examples of keyword and question pairs for *Data Retrieval*.

Category	**Keyword**	**Question Example**
Data Retrieval	`JOIN`	Write a query to display student names and their enrolled course titles by joining the `student` and `takes` tables using the student ID.
Data Retrieval	`ORDER BY`	Retrieve all student names and grades from the `student` table and sort the result in descending order by grade.

6.3.3.1 SDP Instance Generation Procedure

The generation of a new *SDP* instance running on a web browser involves several steps. The process begins with teachers selecting relevant learning materials from *SQL database programming* classes, textbooks, or online resources such as [10, 11]. These materials serve as the foundation for designing exercises that students can later study and practice.

From the materials, teachers need to identify important *SQL* keywords and choose the corresponding question templates. The relevant questions are then selected from the question–answer pairs generated by the *generative AI-assisted SQL query generator*. Each selected question–answer pair is manually reviewed by teachers to ensure the correctness, clarity, and alignment with the learning objectives.

At this step, any redundant or overlapping question–answer pairs are discarded to maintain variety within the exercise set. After validation, the finalized content is compiled into an *SDP instance text file* that contains all the related data, including schema information, images, questions, and correct answers.

Finally, the *answer interface generator* automatically produces the necessary *HTML*, *CSS*, and *JavaScript* files from the *SDP instance text file* [12, 4] that can run on a web browser. This integration allows students to access and practice the exercises interactively, with automatic grading and feedback mechanisms built in *PLAS*.

6.3.3.2 Instance List Page

The *instance list page* allows students to view and manage the assigned *SDP* instances. Figure 6.3 shows the layout of this page, where each *SDP* instance is displayed with its title and progress status. The instances highlighted in *green* indicate that all the questions in them have been answered correctly by a student, while those in *yellow* show partial completions. The instances with no highlighting are those that have not yet been attempted. Students are advised to read provided instructions before attempting exercises. This visual feedback mechanism helps learners track their progress and identify which topics require further review.

6.3.3.3 Answer Page

The *answer page* enables students to input and test *SQL* queries for questions interactively. Figure 6.4 illustrates the page providing a text input area where students can write the *SQL* query in response to each question. Upon submission, the *JavaScript* program on the page checks the answer using *string matching* and executes the query to confirm correctness.

Incorrect submissions are highlighted in *red*, while correct ones appear in *white*. Students may resubmit their answers until all questions are correctly solved. Each attempt is recorded, including the submission time, the answers, and results, allowing teachers to monitor learning progress and common errors.

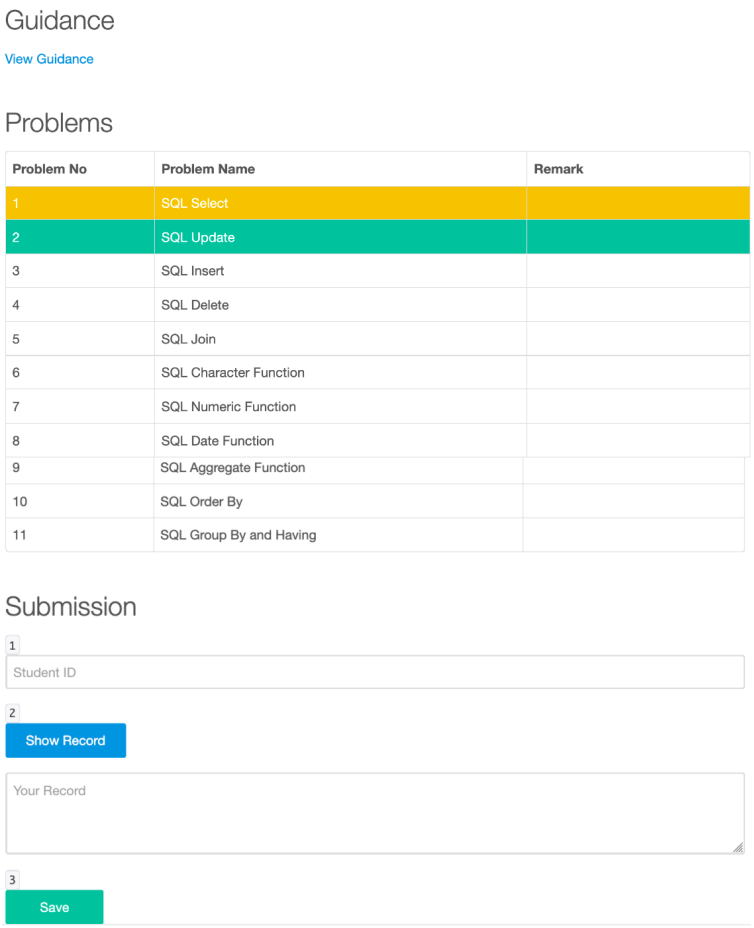

Figure 6.3: Instance list page.

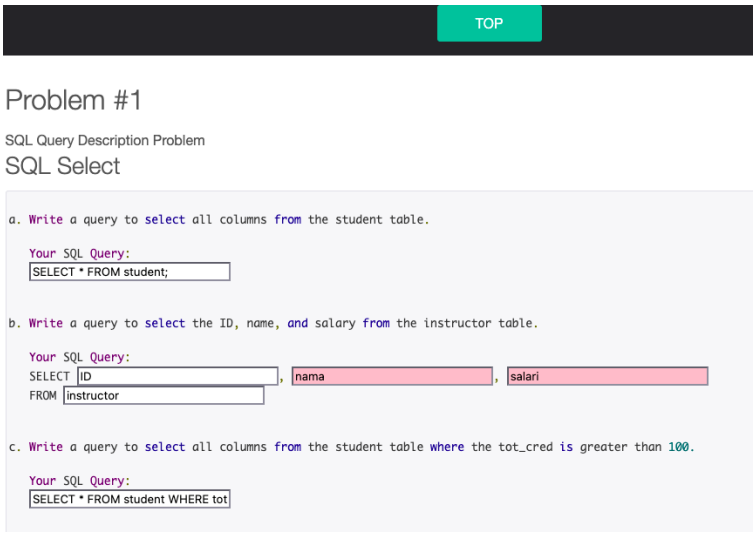

Figure 6.4: *SDP* answer page.

6.3.4 SDP Assessment Logic

Our *SDP* assessment system employs a *string matching* mechanism to validate student submissions. This process compares the student's query against a comprehensive set of alternative answers—representing semantically equivalent variants—stored within the *answer module*. This approach allows the system to accept various correct syntactic forms for a single problem.

6.4 Evaluation Setup

In this section, we present a methodology and experimental design.

6.4.1 Generative AI-assisted SQL Query Generator Evaluation Setup

To ensure the validity and reliability of the proposed *Generative AI-assisted SQL Query Generator*, an evaluation framework has been established, encompassing the system configuration and testing scenarios as detailed below.

6.4.1.1 Evaluation Metrics

To assess the quality of problems generated by a *generative AI*, we used metrics adopted from Sarsa et al. [33].

Table 6.4: Evaluation metrics for assessing the quality of SQL question–answer pairs generated by *generative AI*, adapted from Sarsa et al.

Aspect	Question	Description
A. Subjective Quality Metrics (Assessed by Experts using a 5-Point Likert Scale)		
Sensibleness	Does the problem description describe a sensible problem?	Assesses the clarity, grammatical accuracy, and factual correctness of the description. (1: Highly Illogical / 5: Highly Logical)
Novelty	Are we unable to find the SQL exercise via online search (Google) of the problem statement?	Assesses the degree of uniqueness of the SQL query problem/description. (1: Already Existing / 5: Highly Novel)
Topicality: concept/keyword	Does the answer to the problem statement require the primed concept/keyword?	Assesses the relevance of the description to the requested SQL concepts/keywords. (1: Not Relevant / 5: Highly Relevant)
Topicality: extras	Does the generated problem statement and solution use the extra primed words (e.g., chain-of-thought, etc.)?	Assesses the quality of integration of additional primed elements into the output. (1: Failure / 5: Perfect)
Readiness: answer	Is the answer to the generated problem provided?	Assesses the completeness and sufficiency of the description as a learning aid for students. (1: Not Ready / 5: Highly Ready)
B. Objective Metric (Numerical)		
Readiness: runnability	How many corrections do we need to make to be able to run the solution query correctly?	Integer

6.4.1.2 Assessment Procedure

The evaluation is conducted independently by two (2) lecturers of the Database course each having a minimum of three years of teaching experience at the university level. These assessors serve as expert validators. They are provided with a detailed list of output descriptions and assessment instructions (including the operational definitions above) without being informed whether the descriptions were generated by *AI* or manually created, to minimize bias.

6.4.1.3 Inter-Rater Reliability

Inter-Rater Reliability (IRR) is a crucial statistical measure designed to assess the degree of consistency and agreement among the ratings provided by two or more independent assessors [34]. In this study, the evaluation was conducted independently by two (2) expert validators—Database course lecturers with a minimum of three years of teaching experience. Employing *IRR* is essential to confirm that the evaluation of the output descriptions is not reliant on the subjective preferences of a single individual, but represents an objective and reliable consensus among experts.

6.4.1.4 Bias Minimization Procedure

To ensure the collected *IRR* data accurately reflects genuine agreement, strict measures were taken to minimize potential bias. The expert validators performed their evaluation tasks independently of one another. Crucially, they were blinded to the source of the output descriptions—whether they were *AI-generated* or manually created—thus eliminating the risk of rater expectancy bias. This procedure ensures that the assessment focuses purely on the quality of the description based on the defined operational metrics.

6.4.1.5 The Cohen's Kappa Coefficient

The *Cohen's Kappa* (κ) coefficient was selected as the appropriate statistical measure for assessing *IRR* in this study. Unlike *Fleiss' Kappa*, which is used for multiple raters, *Cohen's Kappa* is specifically designed to measure the agreement between two independent raters assigning categorical or ordinal ratings to a set of items [35].

Given that the evaluation utilized a 5-point *Likert scale*, a *Weighted Cohen's Kappa* was employed. This variation accounts for the magnitude of disagreement between raters (e.g., a difference between a score of 4 and 5 is penalized less than a difference between 1 and 5), which is essential for ordinal data. The coefficient is calculated using the following formula:

$$\kappa = \frac{P_o - P_e}{1 - P_e}$$

where P_o represents the observed proportion of agreement, and P_e denotes the expected proportion of agreement by chance. Based on the interpretation scale by Landis and Koch [36], the global *IRR* for this study achieved a linear weighted kappa of `0.631`, indicating a *substantial agreement* between the two human raters across all learning topics.

6.4.1.6 Interpretation of Kappa Value

The resulting *Cohen's Kappa* (κ) value is interpreted to determine the strength of agreement between the two human raters. This study adopts the widely accepted interpretation benchmarks proposed by Landis and Koch [36], as summarized in Table 6.6.

A satisfactory level of agreement is typically indicated by a κ value of at least 0.61, corresponding to a substantial level of agreement. In this evaluation, the global weighted kappa value was `0.631`. This result meets the threshold for substantial agreement, supporting the conclusion that the metrics assessments of the *AI models'* outputs demonstrate high reliability and reflect a credible and objective consensus between the raters.

Table 6.6: Interpretation of Kappa Values (Landis & Koch, 1977)

Kappa Value (κ)	**Strength of Agreement**
< 0.00	Poor
0.00 – 0.20	Slight
0.21 – 0.40	Fair
0.41 – 0.60	Moderate
0.61 – 0.80	**Substantial**
0.81 – 1.00	Almost Perfect

6.4.2 SDP PLAS Evaluation Setup

For evaluations, we generated 11 *SDP* instances with 67 questions. Table 6.7 lists the topic and the number of questions for each instance. Then, we assigned them to 32 first-year undergraduate students who were taking the *SQL database programming* course at *Indonesian Institute of Business and Technology (INSTIKI)*, Indonesia. Instances with IDs 1 through 5 were included in the midterm exam, whereas instances with IDs 6 through 11 are reserved for the final exam. The duration time of midterm exam was 60 minutes, which was sufficient for students to repeat answer submissions until they got the correct answers.

Table 6.7: Generated *SDP* instances with corresponding *SQL* topics.

No.	***SQL* Topic**	**Number of Questions**
1	SELECT Statement	5
2	UPDATE Statement	5
3	INSERT Statement	8
4	DELETE Statement	6
5	JOIN Operation	5
6	Character Functions	9
7	Numeric Functions	8
8	Date Functions	5
9	Aggregate Functions	6
10	ORDER BY Clause	8
11	GROUP BY and HAVING Clauses	4

6.4.2.1 Participants and Demographic Profile

The experiment involved 32 undergraduate students enrolled in the *'Introduction to Database'* course (typically a 1st or 2nd-semester course) at the *Indonesian Institute of Business and Technology (INSTIKI)* [37]. All participants were novice learners of *SQL*, as the experiment was conducted immediately following the introductory lectures on basic *SQL* commands such as `SELECT, FROM, WHERE, JOIN`.

Demographic information for the participants was as follows: 81.3% were male (26 participants) and 18.7% were female (6 participants). All participants had similar academic backgrounds (Information Technology major) and had no prior professional experience with *SQL* or relational databases. The homogeneity of the sample (novice level, same major) was intentional to control for prior knowledge variables and focus on the feasibility of the *SDP* for beginners.

The homogeneity was chosen to mitigate confounding variables related to prior *SQL* expertise, ensuring that the measured learning outcomes were attributable to the *SDP* method rather than varying skill levels. As discussed in Section 6.7.5, this intentional restriction limits the external validity, but strengthens the internal validity of the findings regarding the efficacy of *SDP* for introductory *SQL* students.

6.4.2.2 Experimental Procedure

The experiment was conducted during a scheduled 120-minute laboratory session. The procedure adhered to the following steps:

Step 1: **Preparation (10 min)**: Participants were briefed on the purpose of the study and the ethical consent form (including anonymity and voluntary participation).

Step 2: **Introduction to *SDP* (10 min)**: Participants were given a brief tutorial on the concept of the *SDP* and how to interact with the web-based *PLAS* interface.

Step 3: **Task Execution (60 min)**: Each participant was instructed to solve a predetermined set of 5 *SDP* instances (IDs 1–5), comprising a total of 29 questions, using the proposed learning assistant system. Participants were permitted to solve the questions in any order.

Step 4: **Data Collection (40 min)**: After completing the tasks, participants were asked to fill out two standardized instruments:

(a) a survey questionnaire to evaluate system feasibility (correctness rate and submission times), and

(b) the *SUS* questionnaire to assess the system's usefulness and usability.

All data collection was automated by the *PLAS*, and the experiment was supervised by one of the co-authors (affiliated with *INSTIKI*) and a dedicated teaching assistant to ensure standardized conditions across all participants.

6.4.2.3 System Usability Scale

Following the assignments, a *System Usability Scale (SUS)* questionnaire was distributed to the participating students. Originally developed by John Brooke in 1986 [38], the *SUS* has been known as a simple yet reliable method for measuring the usability of interactive systems. Table 6.9 presents the 10 *SUS* questions in this evaluation, each rated on a five-point *Likert* scale, where 1 represents *Strongly Disagree* and 5 represents *Strongly Agree*.

Each answer is scored by a point between 1 and 5. For odd-numbered questions, one point is subtracted from the participant's response, while for even-numbered ones, the response value is subtracted from five points. The adjusted values are then summed and multiplied by 2.5 to obtain a final score between 0 and 100.

For example, if a participant rates the question 1 with 4, the adjusted value becomes $(4-1) = 3$. If the question 2 is rated with 3, the adjusted value becomes $(5-3) = 2$. These adjusted points are added to the overall score.

According to Table 6.10, a score between 0 and 50.9 is considered *Not acceptable*, a score between 51 and 70.9 is *Marginal*, and a score from 71 to 100 is *Acceptable*. Table 6.12 shows the percentile-based grade interpretation, where a score of 80.3 or higher corresponds to *Grade A*, and a score below 51 corresponds to *Grade E*.

Table 6.9: *SUS* questions.

No.	**Question**
1	I think that I would like to use this system frequently.
2	I found the system unnecessarily complex.
3	I thought the system was easy to use.
4	I think that I would need the support of a technical person to be able to use this system.
5	I found the various functions in this system were well integrated.
6	I thought there was too much inconsistency in this system.
7	I would imagine that most people would learn to use this system very quickly.
8	I found the system very cumbersome to use.
9	I felt very confident using the system.
10	I needed to learn a lot of things before I could get going with this system.

Table 6.10: Acceptability *SUS* ranges for user satisfaction levels.

SUS Score Range	**Interpretation**
0 – 50.9	Not acceptable
51 – 70.9	Marginal
71 – 100	Acceptable

Table 6.12: *SUS* percentile rank interpretation.

Grade	**Score Range**
A	Score $\geq$ 80.3
B	74 $\leq$ Score $<$ 80.3
C	68 $\leq$ Score $<$ 74
D	51 $\leq$ Score $<$ 68
E	Score $<$ 51

6.5 Evaluation Results

In this section, we present a comprehensive evaluation of the proposal, focusing on the AI models performance and its feasibility for the self-study of *SQL* query by novice students.

6.5.1 Generative AI-assisted SQL Query Generator Evaluation Results

Our evaluation focuses on six key metrics. They include *Sensibleness*, *Novelty*, *Topicality: Concept*, *Topicality: Extras*, *Readiness: Answer*, and *Readiness: Runnability*. These indicators measure the logical accuracy, creativity, topical relevance, and technical correctness of the generated *SQL* exercises.

6.5.1.1 Inter-Rater Reliability Results

To ensure the objectivity and reliability of the evaluation process, an *IRR* test was conducted using *Cohen's Kappa* (κ). This analysis measures the level of agreement between two expert raters for each evaluation metric across a total sample of 220 assessments (110 problems generated by each *AI model*). The results of the *IRR* analysis are summarized in Table 7.5.

Table 6.14: Global Inter-Rater Reliability Results (n = 220)

Metric	**Cohen's Kappa (κ)**	**Interpretation**
Sensibleness	0.86	Almost Perfect Agreement
Novelty	0.91	Almost Perfect Agreement
Topicality: Concept	1.00	Almost Perfect Agreement
Topicality: Extras	0.85	Almost Perfect Agreement
Readiness: Answer	0.87	Almost Perfect Agreement
Runnability (Objective)	0.72	Substantial Agreement

The *IRR* results, as summarized in Table 7.5, confirm a robust level of consistency between the two expert raters. Based on the Table 6.6, five out of six metrics achieved an *Almost Perfect Agreement* with *Kappa* coefficients (κ) ranging from `0.85` to `1.00`.

The *Topicality: Concept* metric achieved a perfect *Kappa* score (κ = `1.00`), indicating complete agreement between the raters regarding the alignment of the generated problems with the intended *SQL* concepts. The *Novelty* metric also demonstrated exceptionally high consistency, with *Kappa* values of `0.91`, respectively. These result confirm a highly synchronized rater perception concerning the originality of the generated *SQL* queries. Metrics reflecting subjective stability, including *Sensibleness* (κ = `0.86`), *Topicality: Extras* (κ = `0.85`), and *Readiness: Answer* (κ = `0.87`), similarly maintained strong levels of agreement.

Interestingly, the *Runnability* metric yielded a *Kappa* score of `0.72`, which is categorized as *Substantial Agreement*. While still indicating a strong level of consensus, this lower coefficient compared to other metrics suggests that identifying technical *SQL* errors—such as subtle syntax mistakes or logical flaws—presented more room for interpretative differences between the raters. Nevertheless, the overall *IRR* results provide a highly reliable foundation, ensuring that the performance comparisons between *ChatGPT-5.0* and *Gemini 3.0 Pro* are substantiated by consistent expert judgment and minimal rater bias.

6.5.1.2 Overall Performance Summary

This section presents a comprehensive comparison between *ChatGPT-5.0* and *Gemini 3.0 Pro* based on subjective quality metrics and objective technical performance. The evaluation was conducted on 110 *SQL* problems spanning 11 core learning topics (Table 6.7), with the resulting mean scores (μ) and standard deviations (σ) summarized in Table 7.6 and Table 6.16. These data provide insights into the accuracy, consistency, and practical implementability of both models as automated problem generation assistants.

Table 6.15: Comparison of Evaluation Scores between ChatGPT and Gemini

Metric	**ChatGPT**		**Gemini**	
	μ	σ	μ	σ
Sensibleness	4.36	1.02	5.00	0.00
Novelty	1.27	0.47	1.86	0.32
Topicality: Concept	4.36	1.21	5.00	0.00
Topicality: Extras	2.95	0.15	4.68	0.46
Readiness: Answer	4.41	0.92	5.00	0.00

Table 6.16: Runnability Error Distribution Across 11 Learning Topics ($n = 110$)

Topic Index	**Gemini 3.0 Pro (Errors)**	**ChatGPT-5.0 (Errors)**
Topic 1–5	13.0	11.5
Topic 6–11	22.0	18.0
Total Errors	**35.0**	**29.5**
Mean Error per Topic	**3.18 ± 2.02**	**2.68 ± 2.14**

Based on the data presented in Table 7.6 and Table 6.16, the following key insights summarize the performance of *ChatGPT-5.0* and *Gemini 3.0 Pro*:

1. **Subjective Quality and Consistency:** *Gemini 3.0 Pro* demonstrates superior and highly consistent performance across nearly all subjective metrics. This is evidenced by a perfect mean score (μ = `5.00`) in *Sensibleness*, *Topicality: Concept*, and *Readiness: Answer*, with a standard deviation (σ) of `0.00`. These results indicate that *Gemini 3.0 Pro* consistently produces logical, relevant, and classroom-ready *SQL* problems without quality variance between iterations. In contrast, *ChatGPT-5.0* exhibits higher quality fluctuations, with σ values ranging from `0.15` to `1.21`, suggesting its output is less stable compared to *Gemini 3.0 Pro*.

2. **Novelty and Instructional Integration:** Both models achieved their lowest scores in *Novelty* (*Gemini 3.0 Pro*: `1.86`; *ChatGPT-5.0*: `1.27`), indicating that the generated problems tend to follow common *SQL* exercise patterns found in existing datasets. However, *Gemini 3.0 Pro* is significantly more effective at integrating additional instructional elements (*Topicality: Extras*) with a score of `4.68`, whereas *ChatGPT-5.0* only reached `2.95`, highlighting *ChatGPT-5.0's* limitations in following complex, specific instructions beyond the core concept.

3. **Technical Error Analysis (Runnability):** Despite *Gemini 3.0 Pro's* subjective superiority, *ChatGPT-5.0* demonstrates better technical performance in the objective *Runnability* metric. *ChatGPT-5.0* produced fewer total errors (29.5) compared to *Gemini 3.0 Pro* (35.0) across all 11 learning topics. On average, *ChatGPT-5.0* required 2.68 corrections per topic, while *Gemini 3.0 Pro* required 3.18.

4. **Error Distribution per Topic:** Both models showed an increase in technical errors as the topics progressed from basic (Topics 1–5) to advanced levels (Topics 6–11). This trend reflects the increased logic complexity in later *SQL* topics, such as advanced joins or subqueries, which remain challenging for generative AI models. While *ChatGPT-5.0* had a lower total error count, its higher standard deviation (σ = 2.14) compared to *Gemini 3.0 Pro* (σ = 2.02) indicates sharper error spikes in specific complex topics.

6.5.2 Evaluation Results for *SDP*

Table 6.17 summarizes the overall performance of the 32 participating students across the *SDP* instances 1-5. This table shows the number of students, the *average correct answer rate* along with its *standard deviation (SD)*, and the *average number of answer submissions* with its corresponding *SD*. The results indicate that the average correct answer rate was 95.20%, suggesting that students were generally able to answer the questions accurately. The average number of submissions was 1.56, implying that each student submitted their answers for a given *SDP* instance approximately one or two times before achieving full correctness.

Table 6.17: Summary of solution results across *SDP* instances.

Number of Students	Average Correct Answer Rate (%)	Standard Deviation	Average Submission Time	Standard Deviation
32	95.20	0.08	1.56	0.82

6.5.2.1 Results for Individual SDP Instances

Figure 6.5 illustrates the *average correct answer rate* and the *average number of answer submissions* by the 32 students for each *SDP* instance.

1. **Correct Answer Rate**: Figure 6.5 indicates that the three instances ID=1, ID=3, and ID=4 show a 100% correct answer rate. An instance ID=2 shows 96.8%, and the lowest correct answer rate is 79.20% for the instance ID=5. The interpretation of this statistic is that students were able to answer correctly and almost perfectly for the instances ID=1-ID=4 but experienced difficulty with the instance ID=5.

2. **Submission Times**: Four instances IDs 1--4 have relatively stable submission times between 1.12 and 1.20. However, the instance ID=5 reached 3.20. Students needed more attempts to answer correctly on the instance ID=5. This score indicates a higher difficulty level compared to the previous instance IDs.

This result suggests a possible need for review or adjustment of the instance `ID=5`, both in terms of question complexity and clarity of instructions.

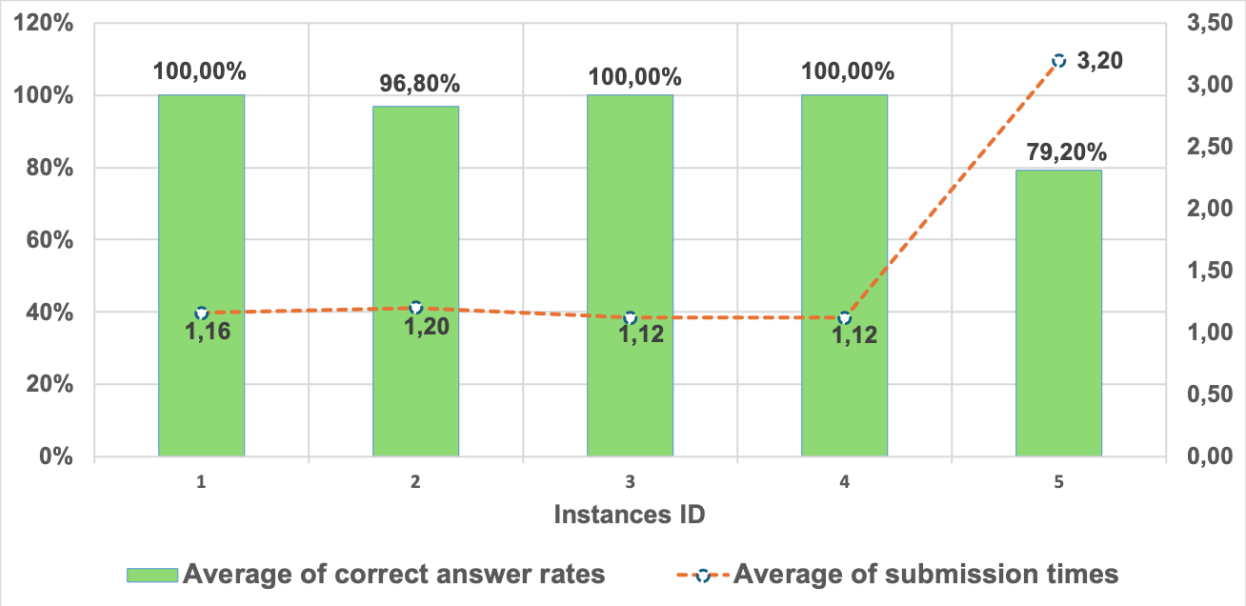

Figure 6.5: Average correct answer rates and submission times for each *SDP* instance.

6.5.2.2 Results for Individual Students

Figure 6.6 illustrates the *average correct rates* and the *average number of answer submissions* of all the *SDP* instances for each of the 32 first-year students.

1. **Correct Answer Rate**: Most students have a high correct answer rate, approaching 100%. The student with `ID = 2` has the lowest correct answer rate at 84%. Some students such as `ID = 12, 19, 21, 24, 27` are slightly below 100%. The remaining students achieved a 100% correct answer rate. The interpretation of this result is that most students are able to answer correctly on most trials, meaning they have a good understanding of the material or questions.

2. **Submission Times**: The average number of submissions was ranged from 1.4 to 2.0, meaning almost all students needed more than one attempt to obtain the correct answer. Four students `ID = 6, 25, 29, 32` showed the highest submission times of 2.0, indicating they needed more attempts to obtain the correct answer. Other students achieved 1.4 and 1.6 times. The student `ID = 32`, with submission times of 2.0, was able to answer all questions correctly.

Students with high submission times will need additional supports and/or guidance so as to complete the questions on their early attempts.

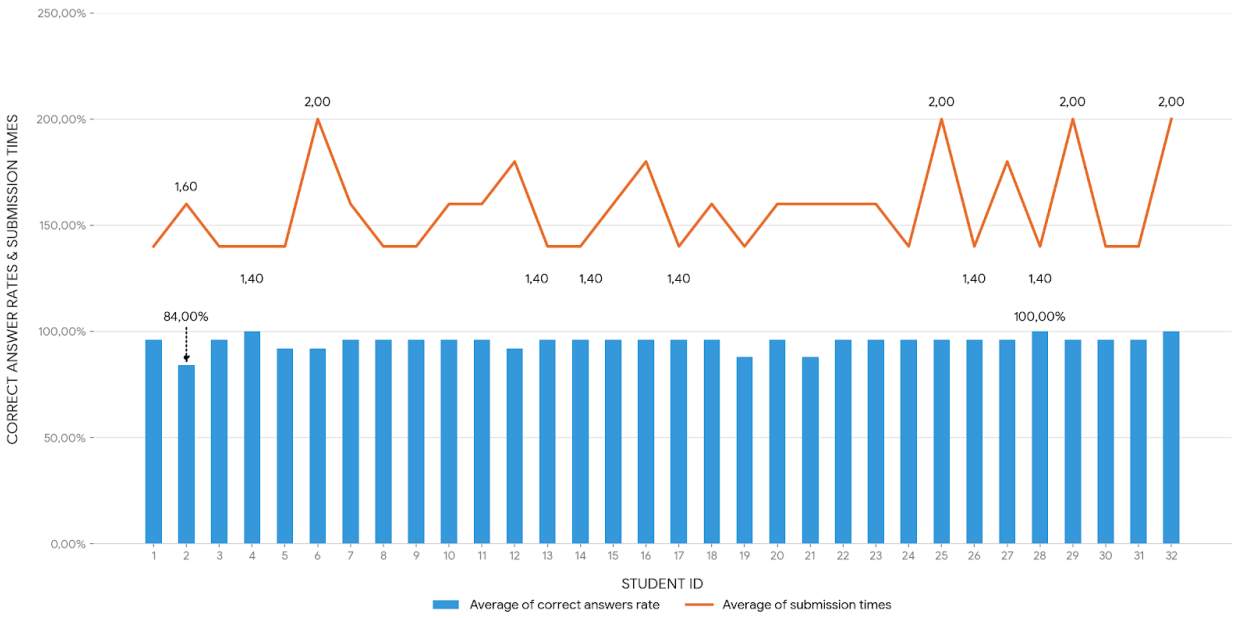

Figure 6.6: Average correct answer rates and submission times for each student.

6.5.2.3 SUS Evaluation Results

After solving the assignments, participants answered to the 10 *SUS* questions in Table 6.9. The results indicate that the proposal was generally perceived as easy to use and well-designed. Minor findings from the usability evaluation are summarized as follows:

1. For the even-numbered statements (Q2, Q4, Q6, Q8, and Q10), the percentage of disagreement responses was 0%, indicating that respondents did not perceive the system is complex or inconsistent, nor did they require technical assistance to use it.

2. For the odd-numbered statements (Q1, Q3, Q5, Q7, and Q9), the percentage of positive responses was 100%, suggesting that they found the answer interface intuitive, easy to learn, and felt confident using it.

Table 6.19: Distribution of responses for questions.
All values are expressed as percentages.

Response	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**
Strongly Disagree	0.00	12.50	0.00	15.63	0.00	21.88	0.00	15.63	0.00	9.38
Disagree	0.00	71.88	0.00	68.75	0.00	56.25	0.00	68.75	0.00	68.75
Neutral	9.38	15.63	18.75	15.63	9.38	21.88	9.38	15.63	31.25	21.88
Agree	56.25	0.00	46.88	0.00	46.88	0.00	46.88	0.00	56.25	0.00
Strongly Agree	34.38	0.00	34.38	0.00	43.75	0.00	43.75	0.00	12.50	0.00

Figure 6.7 illustrate the average *System Usability Scale (SUS)* score was 78, which corresponds to *Grade B* within the *Acceptable* range as in Tables 6.12 and 6.10. This finding indicates that students perceived the proposed *SDP* as usable and effective for learning *SQL* programming.

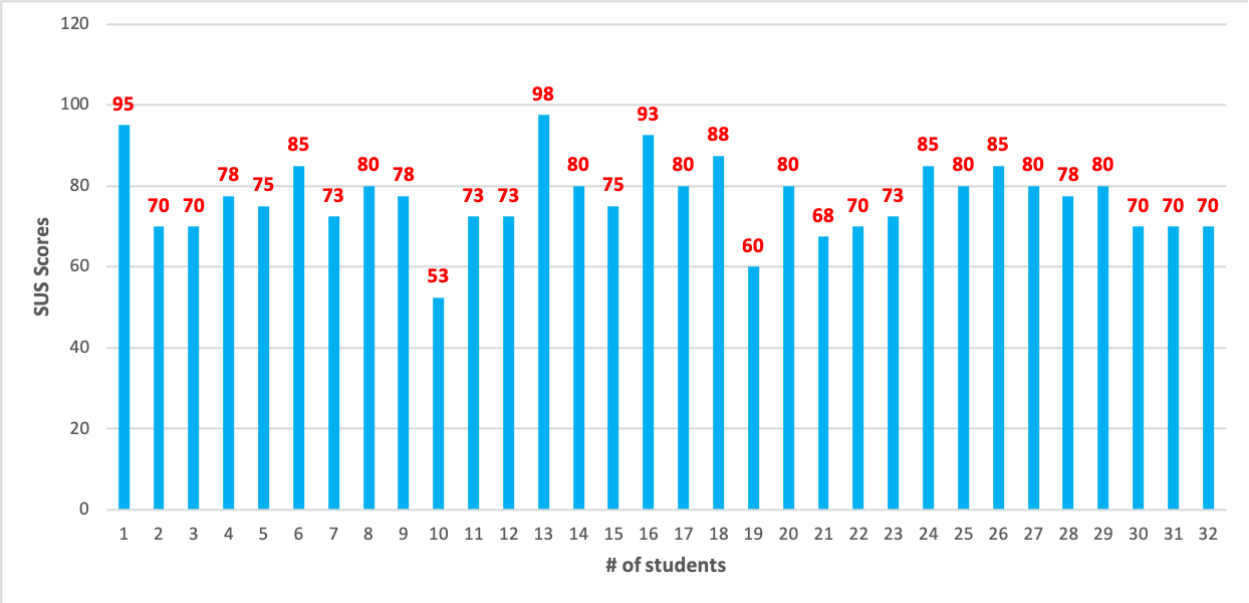

Figure 6.7: *SUS* scores.

6.6 Discussion

In this section we discuss findings, implications, and limitations of the proposal.

6.6.1 Performance of AI Models in Pedagogical Contexts

The comparison between *Gemini 3.0 Pro* and *ChatGPT-5.0* reveals significant insights into the suitability of *LLMs* for educational content generation. While both models were capable of generating *SQL* exercises, *Gemini 3.0 Pro* exhibited superior performance in terms of instructional alignment. The perfect scores in *Sensibleness*, *Topicality*, and *Readiness* suggest that *Gemini* is more reliable for teachers who require stable and ready-to-use exercises without extensive manual editing. *ChatGPT-5.0*, despite its technical efficiency, occasionally required more refinement in its pedagogical logic. This finding aligns with the growing consensus that different *LLMs* may excel in specific niches, with *Gemini* showing promise in structured educational task generation.

6.6.2 Student Performance and the Learning Curve

The high average correct answer rate of 95.2%, coupled with a low submission average of 1.56 per question, underscores the feasibility of the *SDP* in reducing *cognitive load* for novice learners. These metrics suggest that the *AI-generated* descriptions provided sufficient clarity, allowing students to navigate the logic of *SQL* without the typical frustrations of syntax errors. By bridging the gap between conceptual intent (the 'what') and technical execution (the 'how'), the *SDP* method facilitates better 'scaffolding' in programming education. This transition enables students to internalize relational logic more efficiently than traditional methods, which often lack immediate, actionable feedback.

6.6.3 Usability and System Acceptance

The system achieved a *System Usability Scale (SUS)* score of 78, which classifies it within the 'acceptable' and 'good' (Grade B) categories. This result is particularly significant for an educational platform, as it demonstrates that the interface successfully minimizes extraneous *cognitive load*. By reducing the complexity of platform navigation, the system allows students to focus their full attention on mastering *SQL* logic. The positive feedback from 32 students at *INSTIKI* proves that the system is practical and ready to be used more widely in university classes.

6.6.4 Reducing Pedagogical Overhead

One of the primary goals of this research was to alleviate the burden on teachers. Traditional manual creation of *SQL* exercises—including schema design, data insertion, and canonical query writing—is time-consuming. The *AI-assisted generator*, utilizing the *Spider dataset*, automates this entire pipeline. This automation allows teachers to generate a vast and diverse set of exercises in a short amount of time, enabling personalized learning paths where different students can receive different sets of problems of similar difficulty levels, thereby reducing the risk of academic dishonesty.

However, the higher submission rates in some instances (e.g., ID=5) show that question complexity and clarity still require fine-tuning. These findings confirm that *SDP* successfully supports independent learning while preserving teacher oversight and instructional control.

6.6.5 Limitations and Future Improvements

Despite the positive results, this study has several limitations. The system currently relies heavily on the quality of prompts provided to the LLMs, which can occasionally lead to logical inconsistencies or "hallucinations" when generating complex multi-table joins. Furthermore, the current assessment logic primarily uses an enhanced string-matching approach; while suitable for the current scope, it may not capture all forms of semantic equivalence in more advanced SQL queries. Additionally, this study is limited by the absence of a control group and a pre-test/post-test comparison. Furthermore, the sample size of 32 students from a single institution is relatively small, which limits the statistical power and generalizability of the findings. Consequently, the high correct answer rates and positive user feedback should be interpreted as a preliminary validation of the system usability and task completion rather than a direct measurement of learning gains or long-term knowledge retention across diverse educational environments.

To address these limitations and transition from this pilot study to full-scale educational implementation, we have outlined a comprehensive three-step roadmap.

First, regarding system refinement, we will focus on enhancing the robustness of our assessment by integrating semantic analysis techniques that can evaluate query logic beyond simple textual comparison. We also aim to refine the user interface and develop a more advanced *AI-driven* feedback mechanism to provide helpful hints for novice learners.

Second, to address generalizability, we plan to conduct cross-semester and multi-institutional longitudinal studies. This expansion will allow us to verify the system's robustness and evaluate how this method supports long-term knowledge retention among students from diverse learning backgrounds.

Finally, we aim to formally integrate the proposed method into the *SQL* programming curriculum as a primary teaching tool. This will enable a comparative analysis of learning outcomes

against traditional teaching methods, thereby solidifying the system's role in standard computer science education.

6.7 Discussion

In this section we discuss findings, implications, and limitations of the proposal.

6.7.1 Performance of AI Models in Pedagogical Contexts

The comparison between *Gemini 3.0 Pro* and *ChatGPT-5.0* reveals significant insights into the suitability of *LLMs* for educational content generation. While both models were capable of generating *SQL* exercises, *Gemini 3.0 Pro* exhibited superior performance in terms of instructional alignment. The perfect scores in *Sensibleness*, *Topicality*, and *Readiness* suggest that *Gemini* is more reliable for teachers who require stable and ready-to-use exercises without extensive manual editing. *ChatGPT-5.0*, despite its technical efficiency, occasionally required more refinement in its pedagogical logic. This finding aligns with the growing consensus that different *LLMs* may excel in specific niches, with *Gemini* showing promise in structured educational task generation.

6.7.2 Student Performance and the Learning Curve

The high average correct answer rate of 95.2%, coupled with a low submission average of 1.56 per question, underscores the feasibility of the *SDP* in reducing *cognitive load* for novice learners. These metrics suggest that the *AI-generated* descriptions provided sufficient clarity, allowing students to navigate the logic of *SQL* without the typical frustrations of syntax errors. By bridging the gap between conceptual intent (the 'what') and technical execution (the 'how'), the *SDP* method facilitates better 'scaffolding' in programming education. This transition enables students to internalize relational logic more efficiently than traditional methods, which often lack immediate, actionable feedback.

6.7.3 Usability and System Acceptance

The system achieved a *System Usability Scale (SUS)* score of `78`, which classifies it within the 'acceptable' and 'good' (Grade B) categories. This result is particularly significant for an educational platform, as it demonstrates that the interface successfully minimizes extraneous *cognitive load*. By reducing the complexity of platform navigation, the system allows students to focus their full attention on mastering *SQL* logic. The positive feedback from 32 students at *INSTIKI* proves that the system is practical and ready to be used more widely in university classes.

6.7.4 Reducing Pedagogical Overhead

One of the primary goals of this research was to alleviate the burden on teachers. Traditional manual creation of *SQL* exercises—including schema design, data insertion, and canonical query writing—is time-consuming. The *AI-assisted generator*, utilizing the *Spider dataset*, automates this entire pipeline. This automation allows teachers to generate a vast and diverse set of exercises

in a short amount of time, enabling personalized learning paths where different students can receive different sets of problems of similar difficulty levels, thereby reducing the risk of academic dishonesty.

However, the higher submission rates in some instances (e.g., ID=5) show that question complexity and clarity still require fine-tuning. These findings confirm that *SDP* successfully supports independent learning while preserving teacher oversight and instructional control.

6.7.5 Limitations and Future Improvements

Despite the positive results, this study has several limitations. The system currently relies heavily on the quality of prompts provided to the LLMs, which can occasionally lead to logical inconsistencies or "hallucinations" when generating complex multi-table joins. Furthermore, the current assessment logic primarily uses an enhanced string-matching approach; while suitable for the current scope, it may not capture all forms of semantic equivalence in more advanced SQL queries. Additionally, this study is limited by the absence of a control group and a pre-test/post-test comparison. Furthermore, the sample size of 32 students from a single institution is relatively small, which limits the statistical power and generalizability of the findings. Consequently, the high correct answer rates and positive user feedback should be interpreted as a preliminary validation of the system usability and task completion rather than a direct measurement of learning gains or long-term knowledge retention across diverse educational environments.

To address these limitations and transition from this pilot study to full-scale educational implementation, we have outlined a comprehensive three-step roadmap.

First, regarding system refinement, we will focus on enhancing the robustness of our assessment by integrating semantic analysis techniques that can evaluate query logic beyond simple textual comparison. We also aim to refine the user interface and develop a more advanced *AI-driven* feedback mechanism to provide helpful hints for novice learners.

Second, to address generalizability, we plan to conduct cross-semester and multi-institutional longitudinal studies. This expansion will allow us to verify the system's robustness and evaluate how this method supports long-term knowledge retention among students from diverse learning backgrounds.

Finally, we aim to formally integrate the proposed method into the *SQL* programming curriculum as a primary teaching tool. This will enable a comparative analysis of learning outcomes against traditional teaching methods, thereby solidifying the system's role in standard computer science education.

6.8 Summary

This chapter presented the SQL Query Description Problem (SDP) within the Programming Learning Assistant System (PLAS) to enhance self-regulated SQL learning, utilizing a generative AI-assisted generator to automate content creation. A comparative evaluation revealed that Gemini 3.0 Pro offered superior instructional alignment compared to ChatGPT-5.0, making it more suitable for generating ready-to-use educational materials. Field testing confirmed the system's feasibility and usability, evidenced by a high average correct answer rate of 95.2% and a System Usability Scale (SUS) score of 78. Future work will focus on refining AI assessment logic and conducting longitudinal studies to support formal curriculum integration.

Chapter 7

A Comparative Study of Generative AI Models in Generating Exercises for SQL Programming Learning Assistant System

This chapter presents the comparative study of generative AI models in generating exercises for SQL Programming Learning Assistant System for generate question–answer pairs pairs to reduce teacher workloads[39].

7.1 Introduction

Student competency in SQL is assessed through various evaluation instruments, including assignments and examinations. In this domain, assessment design is predicated on the initial development of a database schema and simulated data. This preparatory stage is an intensive effort, particularly when teachers aim for significant database variation across successive course iterations [7].

This study examines how different generative AI models generate SQL question–answer pairs for novice learners using a schema-aware framework. Several generative AI models are evaluated under *one-shot prompting*, with a predefined database schema from the *Spider dataset* provided as context.

Validated by high inter-rater reliability ($\kappa = 0.867$), results show *Gemini 3.0 Pro* excels in consistency and relevance. However, a universal lack of *Novelty* across all models necessitates human oversight to ensure assessment originality.

7.2 Software and Dataset

This study evaluates three leading generative AI models: *GPT-5* (Aug 2025), noted for its advanced reasoning [28]; *Gemini 3.0 Pro* (Nov 2025), which offers enhanced multimodal processing [29]; and *Claude Sonnet 4.5*, designed for efficiency. Their predecessors have shown robust performance in coding tasks, with Gemini 1.0 Pro and ChatGPT-3.5 achieving SQL accuracy rates of 80% and 87% respectively [31], and Claude Sonnet 4.0 reaching 77.2% on SWE-bench Verified [40].

For the experimental context, we utilized the *Spider dataset* [27], a benchmark for *Text-to-SQL* systems containing 10,181 questions across 200 diverse databases. To assess genuine semantic understanding, Spider separates training and testing databases. Specifically, the `college_2` schema

(11 tables, ≈ 30,000 entries) was selected as the test bed due to its relevance to the educational domain.

7.3 Proposal of Generative AI-Assisted SQL question–answer pairs Pairs Generator

The *generative AI–assisted SQL question–answer pairs pairs generator* supports teachers in creating diverse and practical question–answer pairs. The system prepares prompts containing schema details and topic instructions, then validates each generated pair for syntax correctness and executability on a *MySQL* database, ensuring that only successfully executed pairs are saved for teacher review. A standardized prompt template shown in Table 7.1, guides consistent outputs, ensuring balanced coverage of basic and intermediate SQL concepts.

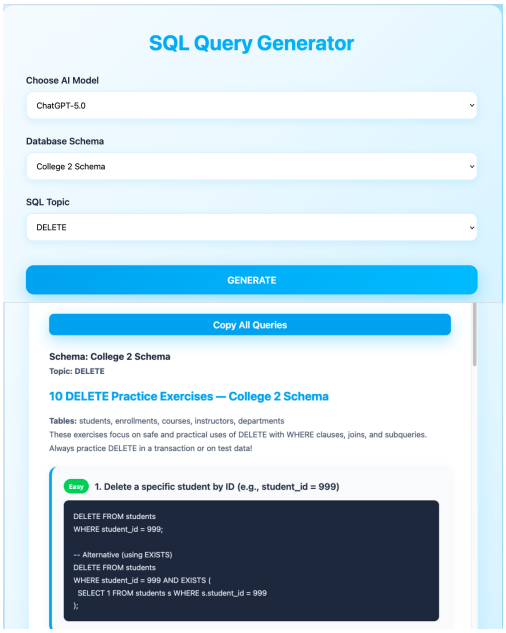

Figure 7.1: Generative AI-assisted SQL Query Generator.

Table 7.1: Prompt template for generative AI in producing SQL question–answer pairs.

Prompt Template:
`"Based on the schema, please create 10 questions with 2 levels of difficulty including answer and explanation about [topic]."`

7.4 Evaluation

7.4.1 Evaluation Setup

We evaluated three generative AI models on the `SQL JOIN` topic using the `college_2` schema, a topic known for its student difficulty and low *AI* novelty [7]. Two database instructors with a minimum of three years of teaching experience assessed the results using metrics adapted from Sarsa et al. (Table 7.3). These metrics measure the logical accuracy and creativity, of the exercises [33]. To ensure the objectivity of the evaluation, we calculated the *Inter-Rater Reliability (IRR)* [34] using *Cohen's Kappa coefficients* [35] for each pair. This statistical measure accounts for the agreement occurring by chance, providing a robust validation of the scoring for the metrics.

Table 7.3: Evaluation metrics for assessing the quality of SQL question–answer pairs generated by *generative AI*, adapted from Sarsa et al.

Aspect	Question	Description
Sensibleness	Does the problem description describe a sensible problem?	Assesses the clarity, grammatical accuracy, and factual correctness of the description. *(1: Highly Illogical / 5: Highly Logical)*
Novelty	Are we unable to find the SQL exercise through an online search (e.g., Google) using the problem statement?	Assesses the degree of uniqueness of the SQL exercise. *(1: Already Existing / 5: Highly Novel)*
Topicality (Concept)	Does answering the problem require the specified concept or keyword?	Assesses the relevance of the description to the requested SQL concepts or keywords. *(1: Not Relevant / 5: Highly Relevant)*

7.4.2 Evaluation Results

As shown in Table 7.5, the global *Quadratic Weighted Kappa* of `0.867` indicates 'almost perfect' agreement. While *Novelty* ($\kappa = 0.57$) reflected expected subjectivity, high consensus in *Topicality* and *Sensibleness* ($\kappa = 0.79$) confirms the framework's reliability for evaluating generated SQL exercises.

Table 7.6 shows *Gemini 3.0 Pro* excelled with perfect *Sensibleness* and *Topicality* scores ($\mu = 5.00, \sigma = 0.00$). However, *Novelty* remained low for all models (¡2.00). *Claude Sonnet 4.5* showed marginal improvement in *Novelty* but lacked the stability of *Gemini 3.0 Pro* and ChatGPT-5.0.

Table 7.5: Global Inter-Rater Reliability Results ($n = 30$)

Metric	**Cohen's Kappa (κ)**	**Interpretation**
Sensibleness	0.789	Substantial Agreement
Novelty	0.569	Moderate Agreement
Topicality	0.000	Perfect Agreement
Global IRR	**0.867**	**Almost Perfect Agreement**

Table 7.6: Comparison of Evaluation Scores

Metric	**ChatGPT**		**Gemini**		**Claude**	
	μ	σ	μ	σ	μ	σ
Sensibleness	4.60	0.46	5.00	0.00	4.62	0.67
Novelty	1.30	0.35	1.42	0.51	1.55	0.72
Topicality	5.00	0.00	5.00	0.00	4.87	0.32

7.5 Summary

This study evaluated *ChatGPT-5.0*, *Gemini 3.0 Pro*, and *Claude Sonnet 4.5* for SQL generation. Supported by high reliability ($\kappa = 0.867$), *Gemini 3.0 Pro* demonstrated superior stability. However, a universal lack of *Novelty* across all models necessitates manual validation to ensure the originality of educational assessments.

Chapter 8

Related Works in Literature

In this section, we introduce some related works to this thesis. First, we introduce works on *SQL* programming learning.

8.1 SQL Programming Learning

In [20], Garner et al. introduced *SQL in Steps (SiS)* as an online environment that combines a graphical user interface with a textual representation for improving *SQL* learning. *SiS* simplifies the transition from graphical to textual interfaces, making the transition seamless. Their study on the application to first-year undergraduates confirmed its potential.

In [2], Mitrovic et al. presented *Intelligent Teaching System (ITS)* research that develops electronic tutors that mimic one-on-one instruction with a human tutor. Their *SQL-Tutor* is designed as a practice environment, assuming students have completed database management courses. Although it only covers `SQL SELECT` statements, the system's significance can be extended to other *SQL* commands and relational database languages, as queries pose significant challenges for students.

In [41], Akhuseyinoglu et al. introduced *Database Query Analyzer (DBQA)* as a learning tool that uses interactive data visualizations to demonstrate effects of clauses and conditions on `SQL SELECT` statements. This tool provides result sets similar to those maintained by a database management system during query processing. *DBQA* modifies the result set according to each clause and condition, highlighting the clause currently being processed.

Overall, research on *SQL* learning show that a variety of tools and systems have been developed to improve students' comprehension and proficiency in learning *SQL*. They include interactive visualizations that show the effects of clauses in queries and graphical interfaces that make the transition easier.

8.1.1 Generative AI for Programming Education

Second, we introduce works on use of *Generative AI* models for programming education.

In [42], Luckin et al. showed that by building a proper *AI* model, educational contents can be made more engaging and effective to accomplish goals of individualized education.

In [43], Chen et al. analysed the impact of *AI* on education, verifying that *AI* has been widely used in education, helping teachers work more effectively, and improving the overall quality of teaching.

In [44], Baidoo-Anu explored potential benefits and limitations of *ChatGPT* in enhancing teaching and learning. They provided recommendations on how *ChatGPT* can be used to maximize its impacts on teaching and learning.

In [45], Choi et al. showed that *GPT-5* represents a significant advancement in educational language models, addressing shortcomings such as hallucinations and pedagogical alignment. This approach offers new opportunities across various subjects and learning profiles. Despite limited empirical studies, the evidence indicates that *GPT-5* aligns with learning objectives and adapts to diverse educational needs.

Based on the research mentioned above, it is clear that *generative AI* models have significant impacts on various educational needs, such as helping teachers work more effectively in making and presenting learning content, thereby improving the quality of teaching.

8.1.2 Generative AI for Query Generation

Third, we introduce works on use of *generative AI* model for query generations.

In [46], Pornphol et al. verified the relational completeness of *SQL* query codes generated by *ChatGPT* as one of the most widely used *generative AI* models. *ChatGPT* generated relational algebra, relational calculus, and *SQL* statements for five natural language test questions.

In [47], Li et al. presented a method for automatic test case development that initially creates a database and subsequently employs a *generative AI* model to ascertain the ground truth, defined as the anticipated execution outcomes of the corresponding *SQL* query on this database.

Together, these studies underscore the potential of *generative AI* models to streamline query generations and validations, improving efficiency and reliability in database management systems while addressing challenges in automation and accuracy.

8.1.3 Reducing Teacher Workloads

Fourth, we introduce works on use of *generative AI* model for reducing teacher workloads.

In [48], Gupta et al. investigated impacts of *generative AI* models on teacher productivity in higher education, highlighting their potential to save time and reduce workloads. They revealed that *generative AI* models streamline administrative tasks, enhance teaching efficiency, and optimize assessment processes. This research contributes to the discourse on their roles in sustainable productivity improvements.

In [49], Hashem et al. explored the use of *ChatGPT* in secondary schools to reduce teacher burnouts. They focused on the creation of materials and lesson plans, which are crucial factors causing burnouts. This study uses customized questions for science, math, and English classes, evaluating *ChatGPT* abilities in individualized planning and content creations. It emphasized benefits of task-specific suggestions and *AI-human* collaborations for personalized planning, aligning with *UAE's AI* integration goals.

Both studies highlighted significant roles of *generative AI* models in reducing teacher workloads and improving productivity. Together, these studies showed how they can effectively support teachers, enabling them to focus more on teaching and less on time-consuming administrative tasks.

8.1.4 Cognitive Load Theory in Programming Learning

In [50], Mason et al. redesigned an introductory database course using *Cognitive Load Theory (CLT)* to address high failure rates. By re-sequencing content, utilizing segmentation, and replacing visual tools with raw SQL to reduce extraneous load, they significantly improved student outcomes. Notably, final exam scores increased and the failure rate dropped from 42% to 8%.

In [51], Yousoof et al. proposed a framework to mitigate working memory limitations in programming education. By employing concept maps that mirror long-term memory schemas, alongside part-code methods and information filtering, the approach segments information effectively. This visualization strategy significantly reduces cognitive load and split attention for novice learners.

While both studies validate *CLT* in database and programming contexts, there remains a paucity of research regarding its application in *AI-assisted SQL programming* environments. Building upon these findings, this study extends the *CLT* framework to *SQL* query formulation, specifically aiming to reduce cognitive load through the proposed *SDP*, which utilizes *Generative AI* to provide scaffolded practice and immediate feedback.

8.1.5 Formative Assessment Theory

Ma et al. [55] utilize *correct answer rates* as a formative indicator to assess knowledge mastery. By visualizing these rates at the class level, they identify common misconceptions and provide targeted feedback to enable students to recognize weaknesses and improve performance.

Similarly, Sudo et al. [53] employ *correct answer rates* to monitor learner understanding and evaluate instructional interventions. Their findings confirm that quantitative performance indicators are effective in supporting learning diagnosis and driving instructional improvement.

Harvey and Aggarwal [54] investigate *submission timing* as a behavioral indicator in programming education. They found that early submissions significantly correlate with higher exam scores, suggesting that *submission time* is a meaningful predictor of student engagement and learning outcomes.

In the specific context of SQL learning, Ma et al. [55] further interpret *correct answer rates* through *CLT* . They posit that low correct rates indicate high cognitive burden, while improvements suggest efficient schema construction, thereby guiding the reduction of *extraneous cognitive load*.

In conclusion, these studies collectively demonstrate that quantitative metrics—ranging from *accuracy rates* to *submission timing*—serve as vital diagnostic tools. By leveraging these indicators, teachers can effectively monitor *cognitive load*, identify behavioral patterns, and provide timely interventions to support student mastery.

Chapter 9

Conclusion

In this thesis, I presented studies of the SQL Programming Learning Assistant System for Novice Learners

Firstly, I presented the grammar-concept understanding problem (*GUP*) for the first-step self-study of *introductory SQL database programming*. For evaluations, five *GUP* instances were made on basic concepts and were assigned to 30 students in Indonesian Institute of Business and Technology (INSTIKI). The results confirmed the applicability to beginners in database programming. Besides, the usability of the proposal got a *SUS* score of 77 (grade B), which is acceptable and suitable for use by beginner students.

Secondly, I presented the Grammar-Concept Understanding Problem (GUP) in SQL-Python for novice learners. We developed 14 GUP instances covering 105 essential keywords and evaluated them with 90 undergraduate students at INSTIKI. The results confirm the strategy's effectiveness in precisely assessing student understanding.

Thirdly, I presented the comment insertion problem (CIP) as a first-step self-study task for entry-level SQL–Python. Eighteen *CIP* instances based on basic source codes were given to 60 students at the Indonesian Institute of Business and Technology, and the results showed that *CIP* is suitable for beginners in database programming. Usability testing of the SQL–Python PLAS using the *SUS* yielded a score of 81 (grade A), and more than 85% of students appreciated *CIP* for entry-level self-study; many reported improved final exam scores and expressed no strong preference between traditional classes and self-study exercises, indicating that both approaches are acceptable for learning.

Fourthly, I presented the SQL Query Description Problem (SDP) within the Programming Learning Assistant System (PLAS) to enhance self-regulated SQL learning, utilizing a generative AI-assisted generator to automate content creation. A comparative evaluation revealed that Gemini 3.0 Pro offered superior instructional alignment compared to ChatGPT-5.0, making it more suitable for generating ready-to-use educational materials. Field testing confirmed the system's feasibility and usability, evidenced by a high average correct answer rate of 95.2% and a System Usability Scale (SUS) score of 78. Future work will focus on refining AI assessment logic and conducting longitudinal studies to support formal curriculum integration.

Finally, I presented comparison of *ChatGPT-5.0*, *Gemini 3.0 Pro*, and *Claude Sonnet 4.5* for SQL generation. Supported by high reliability ($\kappa = 0.867$), *Gemini 3.0 Pro* demonstrated superior stability. However, a universal lack of *Novelty* across all models necessitates manual validation to ensure the originality of educational assessments.

I evaluated, the application results from university students validated the effectiveness of the proposed contributions. By automating *SQL* exercise creation, the *AI-assisted generator* success-

fully reduced teacher workload and mitigated cheating, while *Gemini 3.0 Pro* specifically demonstrated superior stability during the evaluation.

In the future, we will expand the database programming topics into basic, intermediate, and advanced levels, and develop various types of problems tailored for novice students, which will be evaluated through practical applications. Additionally, instructions on how to use the system will be provided to facilitate better understanding. Another focus will be improving the *marking function* by using semantic similarity evaluation for logical assessment beyond *string matching*. Expanding the dataset and refining prompt templates based on teaching feedback will also be pursued. Integrating adaptive difficulty levels and automated error classification will enable a more personalized learning experience, with continuous quality improvement based on user feedback.

Bibliography

[1] Rahmalan, Hidayah, Sharifah Sakinah Syed Ahmad, and Lilly Suriani Affendey. "Investigation on designing a fun and interactive learning approach for Database Programming subject according to students' preferences." In Journal of Physics: Conference Series, vol. 1529, no. 2, p. 022076. IOP Publishing, 2020.

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

[3] S. T. Aung, N. Funabiki, Y. W. Syaifudin, H. H. S. Kyaw, S. L. Aung, N. K. Dim, and W.-C. Kao, "A proposal of grammar-concept understanding problem in Java programming learning assistant system," J. Adv. Inform. Tech., vol. 12, no. 4, pp. 342-350, 2021.

[4] X. Lu, N. Funabiki, I. Naing, H. H. S. Kyaw, and K. Ueda, "A proposal of two types of exercise problems for TCP/IP programming learning by C language," IEICE Tech. Report, NS2022-236, pp. 396-401, March 2023.

[5] Sharma, Akshansh, Firoj Khan, Deepak Sharma, Sunil Gupta, and F. Y. Student. "Python: the programming language of future." Int. J. Innovative Res. Technol 6, no. 2 (2020): 115-118.

[6] EDX: https://www.edx.org/microbachelors/snhux-data-management-with-python-and-sql

[7] W. Aerts et al., "A Feasibility Study on Automated SQL Exercise Generation with ChatGPT-3.5," in *Proc. Int. Workshop on Data Systems Education (DataEd)*, 2024, pp. 13–19.

[8] "SHA-256 Cryptographic Hash Algorithm," Internet: http://www.movable discretionary type.co.uk/scripts/sha256. html/, Access June 20, 2023

[9] M. I. E. Agha, A. M. Jarghon, and S. S. Abu-Naser, "SQL tutor for novice students," Int. J. Academic Inform. Syst. Res. (IJAISR), pp. 1-7, February 2018.

[10] https://instiki.ac.id/wp-content/uploads/2022/02/Modul-Praktikum-Basis-Data-Lanjut.pdf

[11] W3School: https://www.w3schools.com/sql/

[12] E. Ünal and H. Çakir, "Students' views about the problem based collaborative learning environment supported by dynamic web technologies," Malay. Online J. Edu. Tech., vol. 5, no. 2, pp. 1-19, 2017.

[13] N. W. Wardani, N. Funabiki, P. Sugiartawan, and I. N. A. S. Putra, "A proposal of grammar-concept understanding problem for self-study of introduction basic to SQL database programming," in *Proc. 2024 Seventh Int. Conf. on Vocational Education and Electrical Engineering (ICVEE)*, pp. 310–316, Oct. 2024.

[14] Klug, Brandy. “An overview of the system usability scale in library website and system usability testing,“ Weave Journal of Library User Experience 1, no. 6, 2017.

[15] N. W. Wardani, N. Funabiki, A. R. Patta, X. Lu, and I. N. A. S. Putra, “A study of grammar-concept understanding problem for SQL Python database programming learning assistant system,” *Technical Committee on Educational Technology*, vol. 123, no. ET2023-37, pp. 7–12, Dec. 2023.

[16] X. Lu, N. Funabiki, S. T. Aung, H. H. S. Kyaw, K. Ueda, W. C. Kao, ”A Study of Grammar-concept Understanding Problem in C Programming Learning Assistant System,” ITE Trans. on MTA, Vol. 10, no. 4, pp. 198-207, 2022.

[17] N. W. Wardani, N. Funabiki, P. Sugiartawan, A.A.S.Pradhana, I.N.D. Kotama, and I. N. A. S. Putra, “An Implementation of Self-study Exercise Problems for Entry-level SQL-Python Database Programming,” Engineering Letters, vol. 33, no. 8, pp. 2939-2948, Aug. 2025.

[18] S. H. Tung, T. Te Lin, and Y. H. Lin, “An exercise management system for teaching programming,“ J. Softw. vol. 8, no. 7, pp. 1718-1725, July 2013.

[19] A. Migler and A. Dekhtyar, “Mapping the SQL learning process in introductory database courses,“ in Proc. SIGCSE, pp. 619-625, 2020.

[20] P. Garner and J. Mariani, “Learning SQL in steps,“ Syst., Cybern. Inform., vol. 13, no. 4, pp. 19-24, 2015.

[21] A. Carbone, I. Mitchell, J. Hurst, and D. Gunstone, “An exploration of internal factors influencing student learning of programming,“ in Proc. Conf. Res. Pract. Inform. Tech. Ser., pp. 25-34, 2009.

[22] M. Piteira and C. Costa, “Learning computer programming: a study of difficulties in learning programming,“ in Proc. ISDOC, pp. 75-80, 2013.

[23] A. Alhadi, V. Behbood, A. Vihavainen, J. Prior and Raymond Lister. “Students’ syntactic mistakes in writing seven different types of SQL queries and its application to predicting students’ success ,“ in Proc. SIGCSE, pp. 401-406, 2016.

[24] T. Taipalus, “The Effects of Database Complexity on SQL Query Formulation,” *Journal of Systems and Software*, vol. 170, p. 110576, 2020.

[25] T. Taipalus and P. Perälä, “What to Expect and What to Focus on in SQL Query Teaching,” in *Proc. 50th ACM Tech. Symp. Computer Science Education (SIGCSE)*, 2019, pp. 198–203.

[26] T. Taipalus, “SQL Education: A Systematic Mapping Study and Future Research Agenda,” *ACM Trans. Computing Education*, vol. 20, no. 3, 2020.

[27] T. Yu et al., “Spider: A Large-Scale Human-Labeled Dataset for Text-to-SQL,” in *Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[28] OpenAI, “Introducing GPT-5,” OpenAI Blog, 2025.

[29] Google DeepMind, “Gemini 3.0 Pro,” DeepMind Blog, 2025.

[30] A. Vaswani *et al.*, "Attention Is All You Need," in *Proc. NeurIPS*, vol. 30, 2017.

[31] C. M. Rosca and A. Stancu, "Quality Assessment of GPT-3.5 and Gemini 1.0 Pro for SQL," *Computer Standards & Interfaces*, vol. 95, 2026.

[32] D. Miedema *et al.*, "Students' Perceptions on Engaging Database Domains and Structures," in *Proc. SIGCSE*, pp. 122–128, 2023.

[33] S. Sarsa et al., "Automatic Generation of Exercises Using Large Language Models," in *Proc. ACM Conf. Int. Computing Education Research (ICER)*, 2022.

[34] K. L. Gwet, *Handbook of Inter-Rater Reliability*. Gaithersburg, MD, USA: STATAXIS Publishing Company, 2001.

[35] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, vol. 20, 1960.

[36] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.

[37] T. S. Tullis and J. N. Stetson, "A Comparison of Questionnaires for Assessing Website Usability," in *Proc. Usability Professional Association Conf.*, 2004, pp. 1–12.

[38] J. Brooke, "SUS: A Quick and Dirty Usability Scale," in *Usability Evaluation in Industry*, London, UK: Taylor & Francis, 1996, pp. 189–194.

[39] N. W. Wardani, N. Funabiki, H. H. S. Kyaw, I. N. D. Kotama, P. Sugiartawan, and I. N. A. S. Putra, "A Comparative Study of Generative AI Models in Generating Exercises for SQL Programming Learning Assistant System," in *Proc. IEICE*, Fukuoka, Japan, Mar. 2026, in press.

[40] Anthropic, "Claude Sonnet 4.5," Anthropic News, 2024.

[41] K. Akhuseyinoglu *et al.*, "A Study of Worked Examples for SQL Programming," in *Proc. ITiCSE*, pp. 82–88, 2022.

[42] R. Luckin and W. Holmes, *Intelligence Unleashed: An Argument for AI in Education*, Pearson, London, UK, 2016.

[43] L. Chen, P. Chen, and Z. Lin, "Artificial Intelligence in Education: A Review," *IEEE Access*, vol. 8, pp. 75264–75278, 2020.

[44] D. Baidoo-Anu and L. O. Ansah, "Education in the Era of Generative Artificial Intelligence (AI): Understanding the Potential Benefits of ChatGPT in Promoting Teaching and Learning," *Journal of AI*, vol. 7, no. 1, pp. 52–62, 2023.

[45] W. C. Choi and C. I. Chang, "ChatGPT-5 in Education: New Capabilities and Opportunities for Teaching and Learning," *Preprints*, pp. 1–16, 2025, doi:10.20944/preprints202508.0684.v1.

[46] P. Pornphol and S. Chittayasothorn, "Verification of Relational Database Languages Codes Generated by ChatGPT," in *Proc. 4th Asia Service Sciences and Software Engineering Conf. (ASSE)*, pp. 17–22, 2023.

[47] Z. Li and T. Xie, "Using LLM to Select the Right SQL Query from Candidates," *arXiv preprint*, arXiv:2401.02115, 2024.

[48] M. S. Gupta, N. Kumar, and V. Rao, "AI and Teacher Productivity: A Quantitative Analysis of Time-Saving and Workload Reduction in Education," in *Proc. Conf. Advancing Synergies in Science, Engineering, and Management (ASEM)*, pp. 97–104, 2024.

[49] R. Hashem, N. Ali, F. El Zein, P. Fidalgo, and O. Abu Khurma, "AI to the Rescue: Exploring the Potential of ChatGPT as a Teacher Ally for Workload Relief and Burnout Prevention," *Research and Practice in Technology Enhanced Learning*, pp. 1–26, 2024.

[50] R. Mason, C. Seton, and G. Cooper, "Applying Cognitive Load Theory to the Redesign of a Conventional Database Systems Course," *Computer Science Education*, 2016, doi:10.1080/08993408.2016.1160597.

[51] M. Yousoof, M. Sapiyan, and K. Kamaluddin, "Reducing Cognitive Load in Learning Computer Programming," *World Academy of Science, Engineering and Technology, Int. J. Comput. Inf. Eng.*, vol. 1, no. 12, 2007.

[52] L. Ma, X. Zhang, Z. Wang, and H. Luo, "Designing Effective Instructional Feedback Using a Diagnostic and Visualization System: Evidence from a High School Biology Class," *Systems*, vol. 11, no. 7, p. 364, 2023, doi:10.3390/systems11070364.

[53] K. Sudo, S. Watanuki, H. Matsuoka, E. Otake, Y. Yatomi, N. Nagaoka, and K. Iino, "Effects of the Project on Enhancement of Teaching Skills in Gerontic Nursing Practice of Indonesian Nursing Lecturer and Clinical Nurse Preceptor," *Global Health & Medicine*, p. P1, 2023, doi:10.35772/ghm.2023.01046.

[54] L. Harvey III and A. Aggarwal, "Exploring the Effect of Quiz and Homework Submission Times on Students' Performance in an Introductory Programming Course in a Flipped Classroom Environment," in *Proc. ASEE*, 2021.

[55] L. Ma, X. Zhang, Z. Wang, and H. Luo, "Designing Effective Instructional Feedback Using a Diagnostic and Visualization System: Evidence from a High School Biology Class," *Systems*, vol. 11, no. 7, p. 364, 2023, doi:10.3390/systems11070364.

[56] **Ni Wayan Wardani**, Nobuo Funabiki, Htoo Htoo Sandi Kyaw, Zhu Zhou, I Nyoman Darma Kotama, Putu Sugiartawan, and I Nyoman Agus Suarya Putra, "An SQL Query Description Problem with AI Assistance for SQL Programming Learning Assistant System," *Information*, vol. 17, no. 1, p. 65, January 2026.