An Implementation of Self-study Exercise Problems for Entry-level SQL-Python Database Programming

Ni Wayan Wardani, Nobuo Funabiki, Putu Sugiartawan, Anak Agung Surya Pradhana, I Nyoman Darma Kotama and I Nyoman Agus Suarya Putra

***Abstract*—Currently, *SQL* and *Python* are popular in many universities and professional schools worldwide. *SQL* is an effective domain-specific language used for relational database management and querying. *Python* is a high-level, flexible programming language well known for its abundance of libraries and ease of use. When integrated, they provide a comprehensive approach to study database management by students. This study implements the *grammar-concept understanding problem (GUP)* and the *comment insertion problem (CIP)* for the first-step self-studies of entry-level *SQL-Python database programming*. These problems are designed to understand the keywords and behaviors of the given source code. In *GUP*, each question asks to answer the corresponding key element in the given code representing the defined grammatical idea. In *CIP*, each question asks for the proper comment to be inserted in the code's corresponding place. A string matching with the correct one automatically marks any answer. For evaluations of our *GUP/CIP* approach for the first-step *SQL-Python database programming*, we developed 18 instances each of *GUP* and *CIP* for basic concepts. The approach was tested with 60 students at the Indonesian Institute of Business and Technology, who completed GUP and CIP exercises and assessed system usability via the System Usability Scale (SUS), achieving a satisfactory score of 84. Furthermore, surveys showed self-study enabled flexible learning and practice, over 85% of students reporting improved final exam grades. The findings suggest GUP and CIP enhance self-paced database programming education.**

***Index Terms*—Database programming, SQL, Python, grammar-concept understanding problem, comment insertion problem, self-study, usability evaluation.**

I. Introduction

NOWADAYS, *database systems* have been used in various practical application systems to handle many data efficiently. For them, *database programming* is the essential tool for the maintenance, migration, retrieval, integrity, and security of data in the database. As a result, the competence of *database programming* is considered the inevitable ability of data scientists and software engineers [1]. Many universities and professional schools offer database programming courses worldwide in response to this growing need for qualified database programmers. *SQL (Structured Query Language)* is the potent domain-specific language used for managing and querying relational databases in *database programming* [2].

On the other hand, *Python* is a versatile, high-level programming language known for its simplicity and a huge ecosystem of libraries [3]. They can work together to provide a complete solution for managing database data. Their integration will offer a comprehensive approach to study database management by students.

Previously, the *grammar-concept understanding problem (GUP)* has been presented for self-study of fundamental grammar concepts of *Python programming* [4]. A *GUP* instance consists of a source code, a set of questions on grammatical terms that appear in the code, and their correct answers [5] [6]. Each question describes the grammatical concept or meaning of a keyword, such as a reserved word, a command, and a common library, that appears in the source code and asks to answer it. Any answer from a student is automatically marked by a *string matching* the correct one stored in the answer interface. A student can continue studying *GUP* instances until they reach the correct answers to the questions.

In addition, the *comment insertion problem (CIP)* has been presented for self-study of *code reading* in *TCP/IP network programming* [7]. 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*.

In this paper, we implement the *GUP* and the *CIP* for the first-step self-studies of entry-level *SQL-Python database programming* (*SQL-Python*). We selected 105 keywords for *SQL-Python* and created the corresponding questions based on their definitions. *GUP* and *CIP* are designed for novice learners to start studying programming by understanding the keywords and behaviors of the given source code. For evaluations of our *GUP/CIP* approach for the first step of the *SQL-Python* study, we created 18 *GUP* instances and 18 *CIP* instances on fundamental topics. We assigned the final examination within 180 minutes to 60 undergraduate students who took the database programming course in two majors at the *Indonesian Institute of Business and Technology*. Their solution results

Manuscript received April 17, 2024; revised June 13, 2025. This work was supported by Okayama University and Indonesian Institute of Business and Technology (INSTIKI).

Ni Wayan Wardani is Ph.D candidate of Information and Communication System Department, Okayama University, Okayama, 700-8530, Japan, (e-mail:pj5w1e4c@s.okayama-u.ac.jp).

Nobuo Funabiki is a Professor of Information and Communication System Department, Okayama University, Okayama, 700-8530, Japan (e-mail:funabiki@okayama-u.ac.jp).

Putu Sugiartawan is a Ph.D candidate of Information and Communication System Department, Okayama University, Okayama, 700-8530, Japan (e-mail:p18z9yov@s.okayama-u.ac.jp).

Anak Agung Surya Pradhana is a Ph.D candidate of Information and Communication System Department, Okayama University, Okayama, 700-8530, Japan (e-mail:p44c722y@s.okayama-u.ac.jp).

I Nyoman Darma Kotama is a Ph.D candidate of Information and Communication System Department, Okayama University, Okayama, 700-8530, Japan (e-mail:p9363bg2@s.okayama-u.ac.jp).

I Nyoman Agus Suarya Putra is an Assistant Professor of Business and Creative Design Department, Indonesian Institute of Business and Technology, Denpasar, 80225, Indonesia (e-mail:nyomansuarya@instiki.ac.id).

confirm the proposal's validity in the *SQL-Python* study.

The remaining sections of this paper are organized as follows: Section II overviews related works in literature. Section III overviews the preparation of fill-in-the-blank questions. Section IV presents *GUP* for database programming. Section V presents *CIP* for database programming. Section VI overviews the usability aspects of the proposed system. Section VII evaluates the proposal. Section VIII concludes this paper with future works.

II. Literature Review

In this section, we briefly introduce some related works in literature.

In [2], A. Mitrovic presented *SQL Tutor*. Developing electronic tutors that mimic the benefits of one-on-one instruction with a human tutor is the aim of the *Intelligent Teaching System (ITS)* research. *SQL Tutor* is designed as a practice environment, presupposing that students have completed courses on the fundamentals of database management. Thereby complementing rather than replacing traditional education. Although the system covers only the SQL SELECT statement, other *SQL* statements might be covered using a similar methodology. Because queries lead to the significant challenges among students, this emphasis on the SELECT statement does not lessen the system's significance. Additionally, SELECT concepts are directly applicable to other *SQL* commands and relational database languages.

In [8], Tung et al. introduced the *Programming Learning Web (PLWeb)* as a comprehensive system designed to manage programming exercises, assisting instructors in designing tasks, and fsupporting student programming learning. Specifically, *PLWeb* includes an integrated development environment (IDE) as a key feature to enhance the learning experience.

In [9], Alaoutinen et al. presented a *student self-assessment tool* that can be used to motivate learning and monitor the student's progress. This tool offers a survey with questions scaled based on Bloom's new taxonomy. In addition, it provides instructors a framework that is more objective than general scales to assess the student's level of knowledge acquisition.

In [10], Agha et al. introduced a *SQL tutor* designed for novice students to learn database programming. This tutoring system displays materials such as video, audio, and image files to enhance the learning process.

In [11], Lavbic et al. built a method to support students' *SQL* learning. The system utilizes attempts by previous students at completing *SQL-related* tasks. They used the *Markov Decision Process (MDP)* to explore the states in search of the best hint and to encode the knowledge derived from past data. The authors parsed the *SQL* language and constructed the solution steps to bridge the gap between raw queries and *MDP* states. Their findings show that even after multiple steps guidance, the hints are widely accepted and significantly reduce the time required to find the correct answer.

In [12], Garner et al. discussed existing technologies to improve students' *SQL* learning. They introduced *SQL in Steps (SiS)* as an online environment that combines a graphical user interface with a textual representation to facilitate learning. *SiS* allows users to easily identify and understand errors, easing the transition from graphical to textual interfaces. The interface is intuitive, and the close coupling between the user interface and textual equivalent makes the transition seamless. Their study involving first-year undergraduates confirmed the potential of *SiS* in *SQL* learning.

In [13], Migler et al. examined students' learning process in an introductory database course. They studied 114 students' efforts to solve 116 SQL lab exercises. They tracked how well students understood the *SQL* concepts and the effort they invested in completing database tasks.

In [14] and [15], Ala-Mutka et al. and Konecki highlighted common challenges novice programmers face and explored approaches to programming education. Various tools have been developed to support students in overcoming programming obstacles. To address these challenges, *ToolPetcha*, proposed by Queiros et al., is an automated tutoring tool designed to enhance programming education [16].

In [17], Carbone et al. investigated internal factors influencing students' struggles with introductory programming courses, focusing on motivation and problem-solving skills. Their study at Monash University revealed that intrinsic, extrinsic, and achieving motivation, alongside deficiencies in skills like code tracking and debugging, significantly impact student engagement and persistence in programming education.

In [18], Piteira et al. explored challenges students encounter in learning computer programming. This study identified specific concepts that students find difficult when acquiring programming skills. Based on this analysis, the authors provided insights and recommendations to improve computer programming education.

In [19], Nguyen et al. discussed the development of an intelligent chatbot designed for educational purposes, particularly in programming courses. The authors introduces the *Integ-Rela* model, which integrates multiple knowledge domains to form a comprehensive knowledge base. This model enables the chatbot to retrieve and provide relevant information across programming-related topics effectively. Consequently, the chatbot serves as a virtual tutor, helping students learn programming concepts. The effectiveness of this system is demonstrated through experiments, showing its potential as a practical tool in online programming education.

In [20], Okonkwo et al. developed *RevBot*, a chatbot to help students practice past exam questions in *Python programming*. Using the *Snatchbot Chatbot API*, *RevBot* is designed to interact with students, providing questions and answers for revision. In their evaluation, assessed its effectiveness, indicating that it can enhance students' performance in *Python programming*. The authors highlights the potential of *RevBot* as a effective tool in programming education, especially for introductory programming courses.

In [21], Ihantola et al. reviewed recent developments in automated assessment tools designed for programming exercises. The authors examined the key features and development approaches, such as programming languages, learning management systems, testing systems, resubmission restrictions, manual assessments, security measures, distribution mechanisms, and customized functionalities for introductory programming courses. The review highlights the strengths and limitations of these tools in supporting programming education.

In [22], Elgendy et al. presented a method using *Genetic Algorithms (GAs)* for automatically generating test data for *ASP.NET* web applications. The study introduces new genetic

operators designed for the specialized architecture of web applications, aiming to improve the execution efficiency and code coverage in test data generation. The system developed in this study uses static analysis to identify critical data-flow elements in applications and then applies *GAs* to generate test cases that effectively cover these elements, such as variables or dependencies. The study demonstrates the system's efficacy through case studies on real-world *ASP.NET* projects and empirical evaluation, highlighting its value in enhancing the reliability of *ASP.NET* web applications.

In [23], Unal et al. conducted a qualitative study to explore ¨ students' perceptions of a collaborative learning environment. They developed a educational platform utilizing technology for web-based interactive problems. To evaluate this environment, a semi-structured interview format was employed to gather students' opinions, and facilitating collaborative problem-solving using dynamic web technologies. The findings suggested that incorporating collaborative learning techniques focused on problem-solving and leveraging dynamic web technologies can enhance student engagement and problem-solving skills in a community college settings.

In [24], Akhuseyinoglu et al. introduce *Database Query Analyzer (DBQA)*, a learning tool that uses interactive data visualizations to demonstrate the effects of clauses and conditions on SQL SELECT statements. In this study, the tool was used to illustrate *SQL* query examples. Specifically, *DBQA* provides result sets comparable to those maintained by the database management system while the query is being processed. Clauses in an SQL SELECT statement are processed by *DBQA* in the following order (if applicable) upon query submission: FROM, WHERE, SELECT, GROUP BY, HAVING, and ORDER BY. *DBQA* modifies the result set shown to the learner according to each clause and condition, highlighting the clause currently being processed. The study assessed *DBQA's* impact on students' comprehension of *SQL* query processing.

In [25], Kenny et al. present an automated tutor for a database skills training environment to help students improve their *SQL* skills. Because *SQL* is a formal language, it is well-suited for automated tutoring. They employ a virtual teaching model for automated tutoring *SQL*, a query language, where students are involved in a learning-by-doing process. As with most structured languages, errors in SQL can be classified as syntactical, semantic, or pragmatic, enabling detailed analysis. The tutor system emulates this error classification by providing scaffolding in the form of direction and feedback. The student may view increasing levels of feedback, such as error alerts, tips, and partial solutions, in line with guided discovery. The study assessed the tutor's impact on students' SQL proficiency through user testing with pre- and post-assessments, showing improved performance.

In [26], Kakeshita et al. present *pgtracer*, a Moodle plugin that supports programming instruction. *Pgtracer* gives students fill-in-the-blank programming problems and gathers student logs to assess the students' comprehension level and learning process. In their study, they use *pgtracer* to assign homework to students in a real programming course. Every week, they create fill-in-the-blank questions based on the course material. Data analytics features offered by *pgtracer* are used to analyze learner behaviors on the platform. The analysis results are used to determine each student's level of understanding and to create the questions for the upcoming weeks. The system provides instructors with analysis of students' accomplishments and activities to enhance coordination between homework and lectures. The teacher interviews and student surveys provided insights confirming *pgtracer's* effectiveness.

In [27], Klug summarizes examples from library science and academic literature of how *SUS* has been incorporated into usability testing for websites, discovery tools, medical technologies, and print materials. Additionally, the advantages and difficulties of the *SUS* are discussed. The study also explores optimal methods for applying the scale to usability testing and interpreting usability complexity. Klug emphasizes the need for tailored *SUS* applications to address domain-specific usability challenges.

III. Preparation of Fill-in-the-Blank Questions

A. 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 I 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.

TABLE I
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

IV. Grammar-Concept Understanding Problem for SQL-Python

In this section, we present the *grammar-concept understanding problem (GUP)* for learning *SQL-Python*.

A. Definition of GUP

A *GUP* instance consists of questions, *SQL-Python* source code, and the correct answers. Each question asks students to respond to the matching element or keyword in the given source code. It describes a fundamental grammar or library method

in *database programming* that can be found in the code. It is intended to help the student understand the meaning of each significant element or keyword in the source code provided.

B. Keywords and Questions

A list of keywords and their corresponding questions must be produced to generate *GUP* instances. We selected 118 keywords for *SQL-Python* with their questions. Table II shows only a part of them due to space limitations. The definition of each keyword is explained in the corresponding question. A student needs to read the questions and answer the corresponding keywords in the answer forms by understanding the essential grammatical concepts in *SQL-Python*.

Table II includes the basic terms in *database programming* that *CRUD* represents. *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 *GUP* instances should be selected correctly, depending on the target database course by the curriculum [28]. By solving the provided *GUP* instances, a student is expected to understand the meaning or behavior of each crucial keyword in *SQL-Python*.

C. 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 a 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
CREATE  INDEX customer_index
ON customer(customer_code);

#Create a table
cursor.execute( CREATE  TABLE product (
customer_id INTEGER,
name VARCHAR(100),
FOREIGN KEY (customer_id),
REFERENCES customer(id)
    );
```

D. Input Files to Instance Generation

To create a new *GUP* instance, a teacher must select a source code file containing the programming concepts students are expected to learn. The system generates the associated *GUP* instance files using the instance generation method. Before this process, teachers must prepare a list of keywords paired with their corresponding questions.

E. 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 II from the source code.
3) Select the relevant question from Table II 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 [6] with this text.

F. GUP Instance Interface

Figure 1 shows the list of the 18 instances provided along with the remark. If students can complete all the questions, they are highlighted in green. If the remark status is still 'tried' with a yellow highlight, it means that students has not answered all the questions in the instance correctly, and if the instances are correct, then the remark status is 'completed'. If students have not done any work, then there are no highlights.


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	

Fig. 1. *GUP* instance interface.

G. GUP Answer Interface

Figure 2 shows the *answer interface* to solve the *GUP* instance at ID=2 in Table I. When a student enters an incorrect answer in the interface, the corresponding input form is highlighted with a *red* background. Otherwise, the correct response is displayed on a *white* background. The student can repeat the answer by submitting it until each question is correct. The *answer interface* stores the answers correctly, as well as the submission date and time each time the student submits the answers.

V. Comment Insertion Problem for SQL-Python

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

TABLE II
KEYWORD AND QUESTION LIST

keyword	question
connect	Which function is used to create a connection object?
CREATE TABLE	Which keyword is used to create a table?
INSERT INTO	Which keyword is used to insert values into a table?
SELECT*	Which keyword is used to show all data in a table?
WHERE	Which keyword is used to filter specific tables to display?
ALTER TABLE	Which keyword is used to modify the structure of a table?
UPDATE	Which keyword is used to update existing records in a table?
ADD COLUMN	Which keyword is used to modify a table with add a new attribute?
DROP TABLE	Which keyword is used to delete a table?
ORDER BY	Which keyword is used to sort the result?
DELETE	Which keyword is used to delete records from an existing table?
SUM	Which function is used to calculate the sum of a column?
IF	Which function is used to execute different statements based on a condition?

Fig. 2. *GUP* answer interface.

A. 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.

B. 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.

C. CIP Instance Interface

Figure 3 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.

Fig. 3. *CIP* instance interface.

D. CIP Answer Interface

Figure 4 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.

VI. 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,

TABLE III
THE *SUS* STANDARD SCALE

	The System Usability Service Standard Version	Strongly Disagree 1	2	3	4	Strongly Agree 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 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.					

Fig. 4. *CIP* answer interface.

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 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.

TABLE IV
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 V
ACCEPTABILITY RANGES FOR USER ACCEPTANCE

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

VII. EVALUATION

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

A. 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 VI shows the topic, the number of questions for *GUP*, and the number of comments for *CIP*.

B. Result Summary

Tables VII and VIII 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 *GUP* and *CIP*, respectively. Table VII suggests that for *GUP*, the students in *MDI* solved the *GUP* instances better than those in *KAB*, while Table VIII 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. Clearly, for them, *CIP* is easier than *GUP*.

C. Results for GUP individual instances

The results of the *GUP* individual instances are shown in Figure 5 which consists of results for submission times and correct answer rates.

TABLE VI
OVERVIEW OF *GUP* AND *CIP* INSTANCES

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

TABLE VII
RESULT SUMMARY FOR *GUP*

majors	#number of students	correct rate (%)		#of submission	
		ave	SD	ave	SD
MDI	30	95.75	0.01	1.82	0.10
KAB	30	92.90	0.02	1.49	0.17

TABLE VIII
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

1) Submission Times: Figure 5 illustrates the average number of answer submission times for each *GUP* instance by the students in *MDI* and *KAB* major students, respectively. They show that the three *GUP* instances at (ID=12, 13, 14) exhibit only one time to solve those instances for *MDI* students. While, for *KAB* students, they exhibit at (ID=10, 12, 13). The *GUP* instance at ID=2 has the highest submission times, which suggests that some students have difficulty solving it. *MDI* students achieved an average submission times score of 1.82, higher than *KAB* students' score of 1.50.

2) Correct Answer Rates: Figure 5 illustrates the average correct answer rates for each *GUP* instance by the *MDI* and *KAB* major students, respectively. It shows that the five *GUP* instances at (ID=5, 11, 12, 13, 14) exhibit 100% correct answer rates for *MDI* students. For *KAB* students, it shows four *GUP* instances at (ID=10, 11, 12, 14), which exhibit 100% correct answer rates. *MDI* students were able to complete more instances perfectly.

There were 1 *GUP* instance completed with the same score between the two majors, namely ID=16. *KAB* students had more difficulty working on instances compared to *MDI* students, who, on average, had higher correct answer rates. *MDI* students achieved an average correct answer rate score of 95.75%, higher than *KAB* students at 94.04%. From the correct answer rate scores, it can be observed that some MDI students have difficulty in instances ID=2 and ID=4. Other instances can be done well by achieving scores above 90%. Meanwhile, *KAB* students have difficulty with (ID=1, 2, 3), with rates below 90%.

D. Results for CIP Individual Instances

The results of the *CIP* individual instances are shown in figure 6 which consist of results for submission times and correct answer rates.

1) Submission Times: Figure 6 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, 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.

2) Correct Answer Rates: Figure 6 shows that both *MDI* and *KAB* students achieved a 100% correct answer rate across all *CIP* instances, indicating the relative ease of these tasks compared to *GUP* instances.

E. Results of Individual Students in MDI

Figure 7 illustrates the average correct answer rate and the average number of submission times of the *GUP* and *CIP* instances for each of the 30 first-year students in the *MDI* major.

For *GUP*, two students achieved the correct rate above 90.00%, and 24 of the 30 achieved the correct rate above 95.00%. The best student at ID=1 could solve all instances with a rate of 98.61% by 1.78 submissions for each instance. On the other hand, the worst student at ID=28 could solve only 91.79% of the questions where two instances were not solved at all, namely, ID=2 with 67.00% and ID=4 with 71.00%, although the remaining instances were fully solved with 100%. This student needs to put more effort into studying database programming. The student in ID= 27 has the highest submission times but achieves a 94.54% correct rate, which means that this student studied it very seriously. 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.

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 1.78 submissions for each student.

F. Results of Individual Students in KAB

Figure 8 illustrates the average correct answer rates and the average number of submission times of the *GUP* and *CIP* instances for each of the 30 first-year students in the *KAB* major.

For *GUP*, two of the 30 students achieved the average correct answer rate above 95%. The best student at ID=1 could solve all the instances with 98.15% correctly with only 1.67 submissions for each instance on average. On the other hand, the worst student at ID=22 achieved an 88.23% correct answer rate. Four instances were not solved at all, whereas the remaining instances were fully solved with 100%. This student needs more effort in studying *database programming*. The student at ID=8 has the highest submission times but

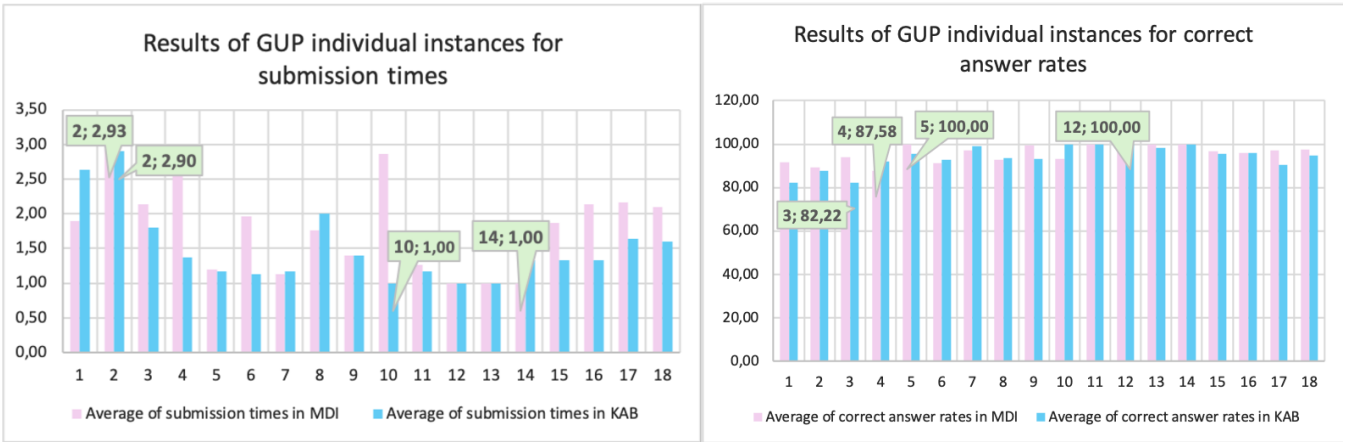

Fig. 5. Results of *GUP* individual instances by students for submission times and correct answer rates.

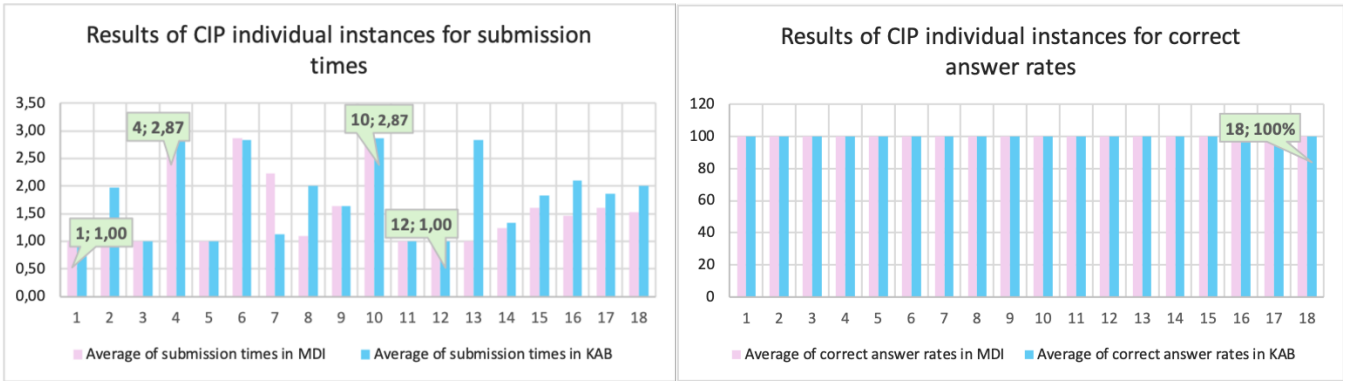

Fig. 6. Results of *CIP* individual instances by students for submission times and correct answer rates.

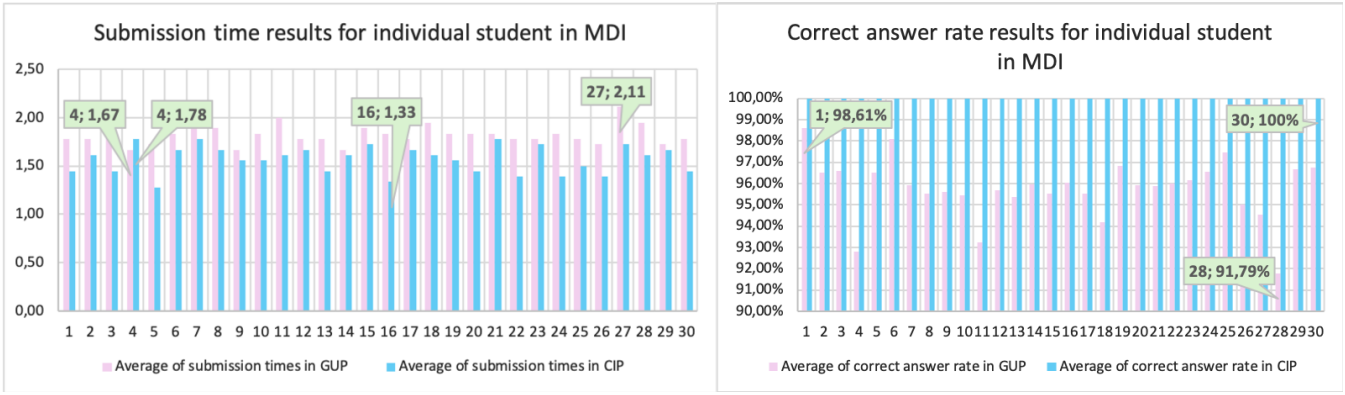

Fig. 7. Results of individual students in MDI.

achieves more than the 90% correct rate, which means this student studied it seriously enough.

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.

G. Comparison of Correct Answer Rates between Problems

Tables IX and X compare the distribution of the number of correct answer rates given by the students between *GUP* and *CIP*. Out of the 60 students in the two majors, no student achieved the correct answer rate of 100%, while 59 students scored above 90%. Two students in the *KAB* major scored less than 90%. The results suggest that the students need to put more effort into studying *SQL-Python*.

By solving *GUP* and *CIP* instances, students in the two majors could reinforce their understanding and review *SQL-Python* concepts and principles to enhance their understanding. It is emphasized that further improvements will be necessary for all students to achieve a 100% score. In *CIP*, 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.

H. Comparison of Submission Times between Majors

Tables XI and XII compare the distribution of submission times between the two majors. For instance, all the students in *MDI* submitted correct answers more than once, whereas

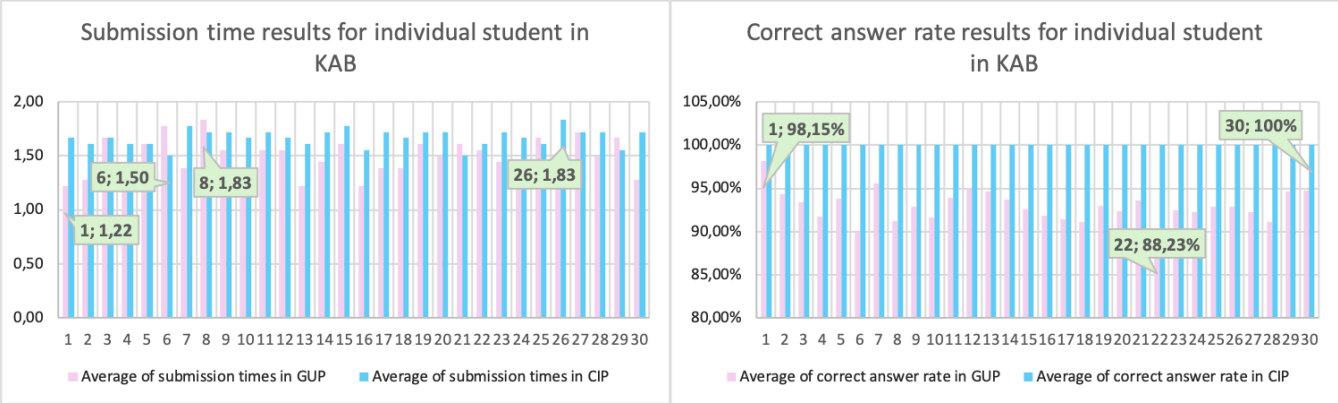

Fig. 8. Results of individual students in KAB.

TABLE IX
COMPARISON OF CORRECT ANSWER RATES IN *GUP*

range of correct answer rate (%)	# of students		rate (%)	
	MDI	KAB	MDI	KAB
85-90	0	2	0	6.67
91-95	18	27	60.00	90.00
96-99	12	1	40.00	3.33
100	0	0	0	0

TABLE X
COMPARISON OF CORRECT ANSWER RATES IN *CIP*.

range of correct answer rate (%)	# of students		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

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 XI
COMPARISON OF SUBMISSION TIMES BETWEEN MAJORS IN *GUP*

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

TABLE XII
COMPARISON OF SUBMISSION TIMES BETWEEN MAJORS IN *CIP*

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

I. 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 *GUP* and 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. Most students feel that answering questions on *GUP* is more challenging because there are no answer choices like *CIP*. 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.

J. Analysis of SUS Questionnaire to the Students

The results of the usability test in Figure 9 were carried out step by step in accordance with the *SUS* calculation guidelines. The final *SUS* score from 30 respondents' responses was 84, in accordance with the *SUS* interpretation guidelines in Table V. The score of 84 was interpreted as follows:

1) Interpretation with acceptability range. Referring to Table V, the score of 84 is included in the Acceptable range.
2) Interpretation using the Grade scale as in Table IV. The score of 84 is included in grade A.

Figure 9 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.

VIII. CONCLUSION

This paper presented the *grammar-concept understanding problem (GUP)* and the *comment insertion problem (CIP)* for the first-step self-study of entry-level *SQL-Python*. For evaluations, 18 *GUP* and 18 *CIP* instances were made using source

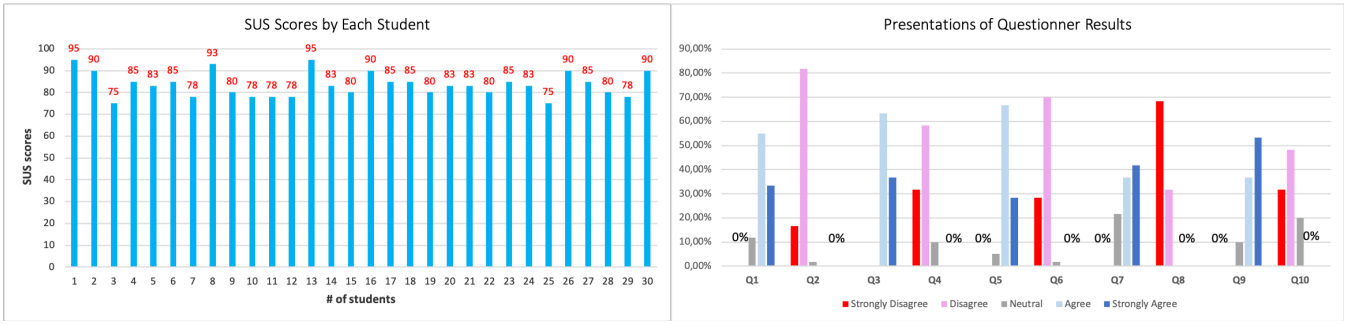

Fig. 9. *SUS* Results.

codes for basic concepts and were assigned to 60 students in the *Indonesian Institute of Business and Technology*. The results confirmed their applicability to beginners in database programming. Apart from that, testing the usability of *SQL Python PLAS* for students using *SUS* got a score of 81 (grade A), which is acceptable and suitable for beginner students to start learning database programming with the *SQL Python* language. A survey among database programming students revealed that more than 85% appreciated *GUP* and the *CIP* for the first-step self-studies of entry-level *SQL-Python*. With a self-study, the student may continue with coursework when they wish. 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. In future work, we will generate other *database programming* topics using several levels, namely basic, intermediate, and advanced, consider different types of problems suitable for novice students, and assess them through application to students. Additionally, in the future, teachers must prepare instructions for using the system appropriately so that users can more quickly understand how to use the system.

References

[1] H. Rahmalan, S. S. S. Ahmad, and L. S. Affendey, "Investigation on designing a fun and interactive learning approach for database programming subject according to students' preferences," J. Phys.: Conf. Ser., vol. 1529, no. 022076, 2020.

[2] A. Mitrovic, "Learning SQL with a computerized tutor," in Proc. SIGCSE Tech. Symp. Comput. Sci. Edu., pp. 307-311, 1998.

[3] A. Sharma, F. Khan, D. Sharma, S. Gupta, and F. Y. Student, "Python: the programming language of future," Int. J. Innov. Res. Tech., vol. 6, no. 2, pp. 115-118, 2020.

[4] E. E. Htet, S. H. M. Shwe, S. T. Aung, N. Funabiki, E. D. Fajrianti, and S. Sukaridhoto, "A study of grammar-concept understanding problem for Python programming learning," in Proc. LifeTech, pp. 245-246, 2022.

[5] S. T. Aung, N. Funabiki, Y. W. Syaifuddin, and H. H. S. Kyaw, "A proposal of grammar-concept understanding problem in Java programming learning assistant system," J. Adv. Inf. Tech., vol. 12, no. 4, pp. 342-350, 2021.

[6] X. Lu, N. Funabiki, S. T. Aung, H. H. S. Kyaw, K. Ueda, and W. C. Kao, "A study of grammar-concept understanding problem in C programming learning assistant system," ITE Trans. MTA, vol. 10, no. 4, pp. 198-207, 2022.

[7] 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, 2023.

[8] 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, 2013.

[9] S. Alaoutinen and K. Smolander, "Student self- assessment in a programming course using Bloom's revised taxonomy," in Proc. ITiCSE, pp. 155-159, 2010.

[10] 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, 2018.

[11] D. Lavbič, T. Matek, and A. Zrnec, "Recommender system for learning SQL using hints," arXiv:1807.02637v1 [cs.AI], pp. 1-18, 2018.

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

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

[14] K. Ala-Mutka, "Problems in Learning and Teaching Programming," A literature study for developing visualizations in the Codewitz-Minerva project, pp. 1-13, 2004.

[15] M. Konecki, "Problems in programming education and means of their improvement," DAAAM Int. Sci. Book, pp. 459-470, 2014.

[16] R. A. Queiros, L. Peixoto, and J. Paulo, "PETCHA - a programming exercises teaching assistant," in Proc. ITiCSE, pp. 192-197, 2012.

[17] 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.

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

[19] H. D. Ngyyen, T.-V. Tuan, X.-T. Pham, A. T. Huynh, V. T. Pham, D. Nguyen, "Design intelligent educational chatbot for information retrieval based on integrated knowledge bases," IAENG Int. J. Comput. Sci., vol. 49, no. 2, pp. 531-541, 2022.

[20] C. W. Okonkwo, and A. Ade-Ibijola, "Revision-Bot: A Chatbot for Studying Past Questions in Introductory Programming," IAENG Int. J. Comput. Sci., vol. 49, no.3, pp. 644-652, 2022.

[21] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in Proc. Koli Calling, pp. 86–93, 2010.

[22] I. T. Elgendy, M. R. Girgis, and A. A. Sewisy, "A GA-Based Approach to Automatic Test Data Generation for ASP.NET Web Applications," IAENG Int. J. Comput. Sci., vol. 47, no.3, pp. 557-564, 2020.

[23] E. Ünal and H. Çakir, "Students' views about the problem based collaborative learning environment supported by dynamic web technologies," *Malaysian Online Journal of Educational Technology*, vol. 5, no. 2, pp. 1-19, 2017.

[24] Akhuseyinoglu, Kamil, Ryan Hardt, Jordan Barria-Pineda, Peter Brusilovsky, Kerttu Pollari-Malmi, Teemu Sirkiä, and Lauri Malmi. "A Study of Worked Examples for SQL Programming," In Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 1, pp. 82-88, 2022.

[25] Kenny, Claire, and Claus Pahl. "Automated tutoring for a database skills training environment," In Proceedings of the 36th SIGCSE technical symposium on Computer science education, pp. 58-62, 2005.

[26] Kakeshita, Tetsuro, and Miyuki Murata. "Application of Programming Education Support Tool pgtracer for Homework Assignment," International Journal of Learning Technologies and Learning Environments, vol.1, no.1, pp. 41-60, 2018.

[27] 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.

[28] Modul (online), https://instiki.ac.id/wpcontent/uploads/2022/02/Modul, January 20, 2024.