UTM UNIVERSITI TEKNOLOGI MALAYSIA | Faculty of Computing

Semester 01 2025/2026

Subject : Database Programming (SECP3623)
Section : Section 1-2
Task : Project I
Due : 25th November 2025 (10%)

Name	Matric No.
ANIS SAFIYYA BINTI JANAI	A23CS0049
NURUL ASYIKIN BINTI KHAIRUL ANUAR	A23CS0162
NABIL AFLAH BOO BINTI MOHD YOSUF BOO YONG CHONG	A23CS0252
AFIF SHAQIR IRFAN BIN ARQAM	A23CS0204

Video Presentation Link: https://youtu.be/4ZRoW49ZYPc

Contents

1.0 Introduction

1.1 Background

Effective attendance tracking is essential in the current educational system for maintaining academic standards and ensure student engagement. The traditional attendance system is not efficient as it takes plenty of time to maintain and generate insights. By providing a reliable database-based attendance tracking system, the student attendance problem can be effectively solved and also increase accuracy.

The system aims to manage complex attendance relationships in the educational environment, such as multiple students attending courses taught by different teachers, including multiple classes and different class times. The database architecture supports the structure of most universities, including multiple departments, a large number of teachers and students, courses divided into multiple classes, and each class contains multiple class hours that need to be tracked separately.

1.2 System and Database Backend Objectives

The main goal of the attendance management system is to create a reliable and efficient database back-end to achieve accurate and convenient attendance monitoring. In addition to facilitating the evaluation of attendance patterns for different courses and periods of time, the system also aims to effectively manage student attendance data and completely record student, teacher and course registration information, so as to ensure the consistency of the data. The solution also includes the audit tracking function of the attendance log to improve accountability.

Then, the database backend needs to support a structured relationship between classes to maintain the data consistency and reduce redundancy. Scalability is also important in the database backend to adapt to the growing amount of data as educational organization continuously accepts new students and change the academic structures such as the courses and credit hour. Other than that, optimized queries are required to achieve fast data retrieval.

1.3 Team Members and Roles

Member Name	Role	Responsibilities
Anis	DDL Developer	DDL Design, Coordination
Afif	DML Developer	Data insertion, Updates
Asyikin	Query Developer	SELECT queries, Analysis
Aflah	Optimization Developer	Indexing, Performance

2.0 SQL Implementation Summary

The SQL implementation for the Attendance Management System consists of two key components: the Data Definition Language (DDL) script and the Data Manipulation Language (DML) script. Both scripts form the core of the backend and support accurate data storage, relationship management, and attendance operations within the academic environment. Together, they demonstrate the complete development of the lifecycle of a relational database system.

2.1 DDL of SQL Files

The DDL script establishes the structural foundation of the database. It defines the attendance_system database along with the major entities involved, such as faculties, lecturers, students, courses, sections, classes, enrollments, attendance records, and audit logs. Each table includes appropriate data types and constraints to ensure consistency, reduce redundancy, and reflect real-world academic relationships.

The DDL operations include:

- Creating all required tables using CREATE TABLE

```
CREATE TABLE faculty (
  fac_id INT AUTO_INCREMENT PRIMARY KEY,
  fac_code VARCHAR(10) NOT NULL UNIQUE,
  fac_name VARCHAR(100) NOT NULL
);
```

Figure 2.1.1

- Defining relationships among tables using FOREIGN KEY constraints

```
CREATE TABLE lecturer (
  lect_id INT AUTO_INCREMENT PRIMARY KEY,
  fac_id INT NOT NULL,
  fullname VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  phone VARCHAR(20),
  hire_date DATE,
  FOREIGN KEY (fac_id)
    REFERENCES faculty(fac_id)
);
```

Figure 2.1.2

- Enforcing data integrity with PRIMARY KEY, UNIQUE, CHECK and NOT NULL rules

```sql
CREATE TABLE attendance (
  attendance_id INT AUTO_INCREMENT PRIMARY KEY,
  class_id INT NOT NULL,
  stud_id INT NOT NULL,
  status ENUM('present','absent','late','excused') DEFAULT 'present' NOT NULL,
  mark_time DATETIME,
  note VARCHAR(255),
  UNIQUE(class_id, stud_id),
  FOREIGN KEY (class_id)
    REFERENCES class(class_id),
  FOREIGN KEY (stud_id)
    REFERENCES student(stud_id)
);
```

Figure 2.2.3

- Applying default values, such as default attendance status

```sql
status ENUM('present','absent','late','excused') DEFAULT 'present' NOT NULL,
mark_time DATETIME,
```

Figure 2.2.4

2.2 DML of SQL Files

The DML script focuses on populating and manipulating data within the defined schema. It introduces realistic datasets that simulate a university environment involving multiple faculties, lecturers, students, courses, sections, and classes.

2.2.1 Data Insertion

A minimum of **ten records per table** were inserted to simulate a functional university attendance environment. Insertions were carried out in a hierarchical order beginning with independent tables such as *faculty*, followed by dependent tables including *lecturer, student, course, section, class,* and *attendance*. This sequencing prevents foreign key violations and ensures complete relational integrity.

- INSERT Statements for Faculty Records:

```sql
INSERT INTO faculty (fac_code, fac_name) VALUES
('FC', 'Faculty of Computing'),
('FKM', 'Faculty of Mechanical Engineering'),
('FS', 'Faculty of Science'),
('FKE', 'Faculty of Electrical Engineering'),
('FKT', 'Faculty of Chemical Engineering'),
('FKA', 'Faculty of Civil Engineering'),
('FABU', 'Faculty of Architecture'),
('FSSH', 'Faculty of Social Science'),
('FSS', 'Faculty of Sports Science'),
('FAI', 'Faculty of AI');
```

Figure 2.2.1: INSERT Statements for Faculty Records

Figure 2.2.1 shows the insertion of ten faculty records into the *faculty* table. These insertions serve as the foundational data for subsequent tables such as lecturer and student.

- INSERT statements for Student Records:

```
INSERT INTO student (fac_id, fullname, email, phone, enrollment_date, stud_year) VALUES
(1, 'Ali Hasan', 'ali@graduate.utm.my', '0122220001', '2023-09-01', 1),
(2, 'Nur Aina', 'aina@graduate.utm.my', '0133330002', '2022-09-01', 2),
(3, 'Kelvin Chia', 'kelvin@graduate.utm.my', '0144440003', '2021-09-01', 3),
(1, 'Siti Aminah', 'aminah@graduate.utm.my', '0155550004', '2024-02-01', 1),
(4, 'Ahmad Danish', 'danish@graduate.utm.my', 01136718507, '2023-09-01', 1),
(5, 'Mary Lee', 'mary@graduate.utm.my', '0188880006', '2022-09-01', 2),
(6, 'John Lim', 'johnlim@graduate.utm.my', '0199990007', '2021-09-01', 3),
(7, 'Lily Chan', 'lily@graduate.utm.my', '0137770008', '2020-09-01', 4),
(8, 'Samuel Tan', 'samuel@graduate.utm.my', '0116660009', '2019-09-01', 3),
(9, 'Farah Izzati', 'farah@graduate.utm.my', '0175550010', '2024-09-01', 1);
```

Figure 2.2.2: INSERT statements for Student Records

Figure 2.2.2 demonstrates the insertion of student data into the *student* table. These records include faculties, enrollment dates, and academic years while adhering to constraints such as NOT NULL, UNIQUE, and CHECK.

- INSERT Statements for Attendance Records:

```
INSERT INTO attendance (class_id, stud_id, status, mark_time, note) VALUES
(1,1,'present','2025-02-01 08:05:00',NULL),
(2,2,'late','2025-02-01 10:15:00','Stuck in traffic'),
(3,3,'absent',NULL,'No show'),
(1,4,'present','2025-02-01 08:01:00',NULL),
(4,1,'present','2025-02-08 08:03:00',NULL),
(5,2,'present','2025-02-08 10:02:00',NULL),
(6,3,'excused','2025-02-08 12:10:00','Medical leave'),
(7,5,'present','2025-02-01 14:05:00',NULL),
(8,6,'late','2025-02-01 16:20:00','Overslept'),
(9,7,'absent',NULL,'Family matters');
```

Figure 2.2.3: INSERT Statements for Attendance Records

Figure 2.2.3 shows an example of inserting attendance records, demonstrating how each attendance entry correctly references existing students and class sessions through foreign keys. These inserts demonstrate correct use of ENUM values, timestamps, and UNIQUE constraints.

- Modify Data's using ALTER TABLE

```
110 •   ALTER TABLE student
111         ADD CONSTRAINT  unique_email UNIQUE (email);
112
113 •   DESC student;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
stud_id	int	NO	PRI	NULL	auto_increment
fac_id	int	NO	MUL	NULL	
fullname	varchar(100)	NO		NULL	
email	varchar(100)	NO	UNI	NULL	
phone	varchar(20)	YES		NULL	
enrollment_date	date	NO		NULL	
stud_year	int	YES		NULL	

Figure 2.2.4 is example of editing existing table using ALTER TABLE

- Removing unnecessary tables using DROP TABLE

```
Remove unnecessary tables
107 • CREATE TABLE absence (
108       reason_id INT AUTO_INCREMENT PRIMARY KEY,
109       attendance_id INT NOT NULL,
110       reason VARCHAR(100),
111       FOREIGN KEY (attendance_id)
112         REFERENCES attendance(attendance_id)
113     );
114
115 •   DROP TABLE absence;
116
117 •   DESC absence;
```

```
11  19:44:59  CREATE TABLE absence (  reason_id INT AUTO_INCREMENT PRIMARY KEY,  attendance_id INT N...  0 row(s) affected
12  19:45:09  DROP TABLE absence                                                                          0 row(s) affected
13  19:45:13  DESC absence                                                                               Error Code: 1146. Table 'attendance_system.absence' doesn't exist
```

Figure 2.2.5 is example of dropping existing table using DROP TABLE

3.0 Query Demonstrations

This section will demonstrate the SQL queries used in the database system for the attendance system. Numerous queries will be executed, that includes retrieving data from the database to performing operations on the collected information. A screenshot and an explanation of the query's purpose and results are provided with every query.

3.1 Filtering

```
209     -- Filtering
210 •   SELECT * FROM student
211     WHERE fac_id = 1 AND stud_year BETWEEN 1 AND 3
212         AND phone IS NOT NULL;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

stud_id	fac_id	fullname	email	phone	enrollment_date	stud_year
1	1	Ali Hasan	ali@graduate.utm.my	0122220001	2023-09-01	1
4	1	Siti Aminah	aminah@graduate.utm.my	0155550004	2024-02-01	1

Figure 3.1: SQL Filtering Query in MySQL Workbench

Figure 3.1 shows the result of filtering students who belong to the Faculty of Computing, are in years 1 to 3 and have registered phone numbers.

3.2 Sorting

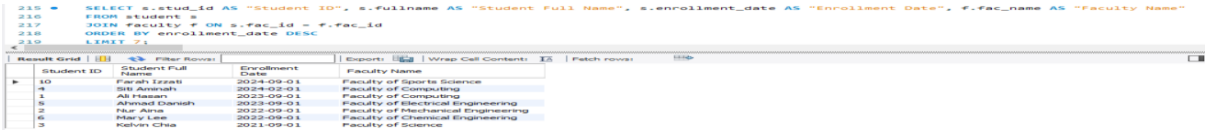


Figure 3.2: SQL Sorting Query by Enrollment Date

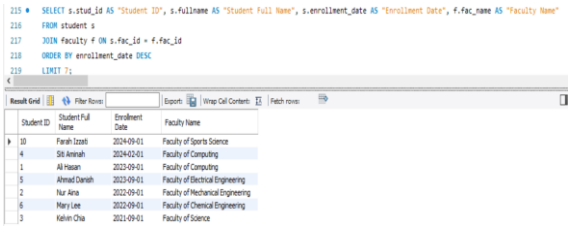


Figure 3.3: Sorting Result Table

Figure 3.2 and 3.3 show students sorted by enrollment date in descending order highlighting the newest entries in the system.

3.3 Aggregation

```
224 •   SELECT c.course_id, c.course_name,
225     MAX(section_stats.student_count) AS max_stud_by_sec,
226     MIN(section_stats.student_count) AS min_stud_by_sec,
227     AVG(section_stats.student_count) AS avg_stud_by_sec
228     FROM course c
229     JOIN section s ON c.course_id = s.course_id
230     JOIN (
231         SELECT sec_id, COUNT(*) AS student_count
232         FROM enrollment
233         GROUP BY sec_id
234     ) AS section_stats ON s.sec_id = section_stats.sec_id
235     GROUP BY c.course_id, c.course_name;
236
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_id	course_name	max_stud_by_sec	min_stud_by_sec	avg_stud_by_sec
SECP3623	Database Programming	2	2	2.0000
SECR1213	Operating Systems	1	1	1.0000
SECJ2013	Software Engineering	1	1	1.0000
SECJ3553	Artificial Intelligence	1	1	1.0000
UHLB1012	English Communication	1	1	1.0000
PHY1113	Physics I	1	1	1.0000
CHEM1013	Chemistry I	1	1	1.0000
BIO1013	Biology I	1	1	1.0000
BUS1012	Business Intro	1	1	1.0000

Figure 3.4: Aggregation of Students per Section

```
224 •    SELECT c.course_id, c.course_name,
225      MAX(section_stats.student_count) AS max_stud_by_sec,
226      MIN(section_stats.student_count) AS min_stud_by_sec,
227      AVG(section_stats.student_count) AS avg_stud_by_sec
228      FROM course c
229      JOIN section s ON c.course_id = s.course_id
230      JOIN (
231          SELECT sec_id, COUNT(*) AS student_count
232          FROM enrollment
233          GROUP BY sec_id
234      ) AS section_stats ON s.sec_id = section_stats.sec_id
235      GROUP BY c.course_id, c.course_name;
236
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_id	course_name	max_stud_by_sec	min_stud_by_sec	avg_stud_by_sec
SECP3623	Database Programming	2	2	2.0000
SECR1213	Operating Systems	1	1	1.0000
SECJ2013	Software Engineering	1	1	1.0000
SECJ3553	Artificial Intelligence	1	1	1.0000
UHLB1012	English Communication	1	1	1.0000
PHY1113	Physics I	1	1	1.0000
CHEM1013	Chemistry I	1	1	1.0000
BIO1013	Biology I	1	1	1.0000
BUS1012	Business Intro	1	1	1.0000

Figure 3.5: Aggregation Results

Figure 3.4 shows the SQL query used to calculate the maximum, minimum and average number of students in each course. Figure 3.5 shows the results, giving insights into class sizes and distribution across courses.

3.4 Grouping and Group Filtering

```
238 •    SELECT e.sec_id, c.course_name, COUNT(e.stud_id) AS total_students
239      FROM enrollment e
240      JOIN section s ON e.sec_id = s.sec_id
241      JOIN course c ON s.course_id = c.course_id
242      GROUP BY e.sec_id, c.course_name
243      HAVING total_students >= 2;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

sec_id	course_name	total_students
1	Database Programming	2

Figure 3.6: Group Students by Section with Minimum Count

Figure 3.6 shows the number of students in each section grouped by course. Only sections with 2 or more students are displayed, demonstrating how GROUP BY and HAVING clauses can be used to filter and summarize data effectively.

3.5 Functions (Numeric and String)

```
246 •    SELECT
247          UPPER(fullname) AS "Full Name (Capital)",
248          LENGTH(email) AS "Email Legth",
249          CONCAT(SUBSTR(fullname, 1, 4), '.') AS "Titles",
250          ROUND(DATEDIFF(CURDATE(),hire_date)/ 365, 1) AS "Years Teaching"
251      FROM lecturer;
252
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Full Name (Capital)	Email Legth	Titles	Years Teaching
DR. AFIF SHAQIR	16	Dr.	5.9
DR. ANIS SAFIYYA	16	Dr.	6.7
PROF. NURUL ASYIKIN	19	Prof	7.8
DR. NABIL AFLAH	17	Dr.	4.4
MR. FAIZ RAZAK	16	Mr.	3.2
DR. SITI HANA	16	Dr.	8.1
DR. JOHN TAN	16	Dr.	9.5
DR. KIM LEE	15	Dr.	10.6
DR. NADIA RAHIM	17	Dr.	2.9
PROF. JAMES WONG	17	Prof	11.3

10.6

Figure 3.7: Functions on Lecturer Data

```
246 •  SELECT
247        UPPER(fullname) AS "Full Name (Capital)",
248        LENGTH(email) AS "Email Legth",
249        CONCAT(SUBSTR(fullname, 1, 4), '') AS "Titles",
250        ROUND(DATEDIFF(CURDATE(),hire_date)/ 365, 1) AS "Years Teaching"
251    FROM lecturer;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Full Name (Capital)	Email Legth	Titles	Years Teaching
DR. AFIF SHAQIR	16	Dr.	5.9
DR. ANIS SAFIYYA	16	Dr.	6.7
PROF. NURUL ASYIKIN	19	Prof	7.8
DR. NABIL AFLAH	17	Dr.	4.4
MR. FAIZ RAZAK	16	Mr.	3.2
DR. SITI HANA	16	Dr.	8.1
DR. JOHN TAN	16	Dr.	9.5
DR. KIM LEE	15	Dr.	10.6
DR. NADIA RAHIM	17	Dr.	2.9
PROF. JAMES WONG	17	Prof	11.3

Figure 3.8: Function Query Output

Figure 3.7 shows how string and numeric functions are applied to lecturer data. Figure 3.8 demonstrates the results which the lecturer names in uppercase, email length, the first 4 letters of the name which are the titles of the lecturers and years of teaching calculated from hire date.

3.6 Conditional Logic

```
254 •  SELECT stud_id AS "Student ID", fullname AS "Full Name", stud_year AS "Student Year",
255    CASE
256        WHEN stud_year = 1 THEN 'First-year Student'
257        WHEN stud_year = 2 THEN 'Second-year Student'
258        WHEN stud_year = 3 THEN 'Third-year Student'
259        WHEN stud_year = 4 THEN 'Fourth-year Student'
260    END AS "Student Year Category"
261    FROM student;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Student ID	Full Name	Student Year	Student Year Category
1	Ali Hasan	1	First-year Student
2	Nur Aina	2	Second-year Student
3	Kelvin Chia	3	Third-year Student
4	Siti Aminah	1	First-year Student
5	Ahmad Danish	1	First-year Student
6	Mary Lee	2	Second-year Student
7	John Lim	3	Third-year Student
8	Lily Chan	4	Fourth-year Student
9	Samuel Tan	3	Third-year Student
10	Farah Izzati	1	First-year Student

Figure 3.9: CASE Query


```
254 •  SELECT stud_id AS "Student ID", fullname AS "Full Name", stud_year AS "Student Year",
255    CASE
256        WHEN stud_year = 1 THEN 'First-year Student'
257        WHEN stud_year = 2 THEN 'Second-year Student'
258        WHEN stud_year = 3 THEN 'Third-year Student'
259        WHEN stud_year = 4 THEN 'Fourth-year Student'
260    END AS "Student Year Category"
261    FROM student;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Student ID	Full Name	Student Year	Student Year Category
1	Ali Hasan	1	First-year Student
2	Nur Aina	2	Second-year Student
3	Kelvin Chia	3	Third-year Student
4	Siti Aminah	1	First-year Student
5	Ahmad Danish	1	First-year Student
6	Mary Lee	2	Second-year Student
7	John Lim	3	Third-year Student
8	Lily Chan	4	Fourth-year Student
9	Samuel Tan	3	Third-year Student
10	Farah Izzati	1	First-year Student

Figure 3.10: CASE Query Results

Figure 3.9 shows the CASE statement used to categorize students based on their study year. Figure 3.10 displays the results, showing the corresponding student year category.

3.7 Subqueries

```
264 •  SELECT * FROM course
265    WHERE credits = (SELECT MAX(credits) FROM course);
266
```

Result Grid | Filter Rows: | Edit: | Export/I

course_id	course_name	credits
BIO1013	Biology I	3
CHEM1013	Chemistry I	3
PHY1113	Physics I	3
SECJ2013	Software Engineering	3
SECJ3553	Artificial Intelligence	3
SECP3623	Database Programming	3
SECR1213	Operating Systems	3
NULL	NULL	NULL

Figure 3.11: Single-row Subquery

This figure shows a single-row subquery that returns the maximum credit value from the course table.

```
267 •  SELECT fullname AS "Students Who Were Taught by Faculty of Computing Lecturers" FROM student
268    WHERE stud_id IN(
269        SELECT e.stud_id
270        FROM enrollment e
271        JOIN section s ON e.sec_id = s.sec_id
272        JOIN lecturer l ON s.lect_id = l.lect_id
273        WHERE l.fac_id = 1
274    );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Students Who Were Taught by Faculty of Computing Lecturers
Ali Hasan
Siti Aminah
Ahmad Danish

Figure 3.12: Multiple-row Subquery

Figure 3.12 demonstrates a multiple-row subquery that returns a list of student IDs taught by lecturers from the Faculty of Computing. The outer query then retrieves the corresponding student names.

```
276 •  SELECT s.stud_id AS "Student ID", s.fullname AS "Student Full Name"
277    FROM student s
278    WHERE (
279        SELECT COUNT(*) FROM attendance a
280        WHERE a.stud_id = s.stud_id
281    ) > 1;
282
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Student ID	Student Full Name
1	Ali Hasan
2	Nur Aina
3	Kelvin Chia

Figure 3.13: Correlated Subquery

Figure 3.13 illustrates correlated subquery where the inner query counts attendance records for each student in the outer query. Only students with more than one attendance record are returned.

3.8 Set Operations

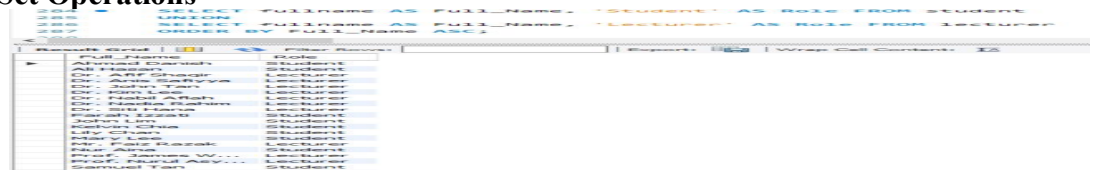


Figure 3.14: UNION Query

```
284 •  SELECT fullname AS Full_Name, 'Student' AS Role FROM student
285    UNION
286    SELECT fullname AS Full_Name, 'Lecturer' AS Role FROM lecturer
287    ORDER BY Full_Name ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Full_Name	Role
Ahmad Danish	Student
Ali Hasan	Student
Dr. Afif Shaqir	Lecturer
Dr. Anis Safiyya	Lecturer
Dr. John Tan	Lecturer
Dr. Kim Lee	Lecturer
Dr. Nabil Afiah	Lecturer
Dr. Nadia Rahim	Lecturer
Dr. Siti Hana	Lecturer
Farah Izzati	Student
John Lim	Student
Kelvin Chia	Student
Lily Chan	Student
Mary Lee	Student
Mr. Faiz Razak	Lecturer
Nur Aina	Student
Prof. James W...	Lecturer
Prof. Nurul Asy...	Lecturer
Samuel Tan	Student

Figure 3.15: UNION Result

Figure 3.14 and 3.15 shows the SQL query that combines student and lecturer names into a single list.

```
289 •    SELECT s.stud_id, s.fullname
290      FROM student s
291      WHERE NOT EXISTS(
292          SELECT * FROM attendance a
293          WHERE a.stud_id = s.stud_id
294      );
```

Result Grid | Filter Rows: | Edit:

stud_id	fullname
8	Lily Chan
9	Samuel Tan
10	Farah Izzati

Figure 3.16: NOT EXISTS Query

```
289 •   SELECT s.stud_id, s.fullname
290     FROM student s
291     WHERE NOT EXISTS(
292         SELECT * FROM attendance a
293         WHERE a.stud_id = s.stud_id
294     );
```

Result Grid | Filter Rows: | Edit:

stud_id	fullname
8	Lily Chan
9	Samuel Tan
10	Farah Izzati

Figure 3.17: NOT EXISTS Result

Figure 3.16 shows the SQL query that retrieves students who have no attendance records while Figure 3.16 displays the students with no recorded attendance.

3.9 Joins

```
297 •   SELECT fullname, course_id
298     FROM student
299     NATURAL JOIN enrollment
300     NATURAL JOIN section
301     ORDER BY fullname ASC;
```

Result Grid | Filter Rows:

fullname	course_id
Ahmad Danish	SECJ3553
Ali Hasan	SECP3623
Farah Izzati	BUS1012
John Lim	PHY1113
Kelvin Chia	SECJ2013
Lily Chan	CHEM1013
Mary Lee	UHLB1012
Nur Aina	SECR1213
Samuel Tan	BIO1013
Siti Aminah	SECP3623

Figure 3.18: NATURAL JOIN Query


```
297 •   SELECT fullname, course_id
298     FROM student
299     NATURAL JOIN enrollment
300     NATURAL JOIN section
301     ORDER BY fullname ASC;
```

Result Grid | Filter Rows:

fullname	course_id
Ahmad Danish	SECJ3553
Ali Hasan	SECP3623
Farah Izzati	BUS1012
John Lim	PHY1113
Kelvin Chia	SECJ2013
Lily Chan	CHEM1013
Mary Lee	UHLB1012
Nur Aina	SECR1213
Samuel Tan	BIO1013
Siti Aminah	SECP3623

Figure 3.19: NATURAL JOIN Result

Figure 3.20: INNER JOIN Query

```
303 •  SELECT s.fullname AS "Student Name", c.course_name AS "Course Name", l.fullname AS "Lecturer Name", e.enrolled_on AS "Enrollment Date"
304    FROM enrollment e
305    INNER JOIN student s ON e.stud_id = s.stud_id
306    INNER JOIN section sec ON e.sec_id = sec.sec_id
307    INNER JOIN course c ON sec.course_id = c.course_id
308    INNER JOIN lecturer l ON sec.lect_id = l.lect_id
309    ORDER BY s.fullname ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Student Name	Course Name	Lecturer Name	Enrollment Date
Ahmad Danish	Artificial Intelligence	Dr. Nabil Aflah	2025-01-12
Ali Hasan	Database Programming	Dr. Afif Shaqir	2025-01-10
Farah Izzati	Business Intro	Dr. Nadia Rahim	2025-01-13
John Lim	Physics I	Dr. Siti Hana	2025-01-13
Kelvin Chia	Software Engineering	Prof. Nurul Asyikin	2025-01-10
Lily Chan	Chemistry I	Dr. John Tan	2025-01-13
Mary Lee	English Communication	Mr. Faiz Razak	2025-01-12
Nur Aina	Operating Systems	Dr. Anis Safiyya	2025-01-10
Samuel Tan	Biology I	Dr. Kim Lee	2025-01-13
Siti Aminah	Database Programming	Dr. Afif Shaqir	2025-01-12

Figure 3.21: INNER JOIN Result

Figure 3.22: LEFT OUTER JOIN Query

```
311 •  SELECT s.fullname AS "Student Name", c.course_name AS "Course Name", a.status AS "Attendance Status", cl.class_date AS "Class Date"
312    FROM student s
313    LEFT OUTER JOIN enrollment e ON s.stud_id = e.stud_id
314    LEFT OUTER JOIN section sec ON e.sec_id = sec.sec_id
315    LEFT OUTER JOIN course c ON sec.course_id = c.course_id
316    LEFT OUTER JOIN attendance a ON s.stud_id = a.stud_id
317    LEFT OUTER JOIN class cl ON a.class_id = cl.class_id
318    ORDER BY s.fullname ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Student Name	Course Name	Attendance Status	Class Date
Ahmad Danish	Artificial Intelligence	present	2025-02-01
Ali Hasan	Database Programming	present	2025-02-01
Ali Hasan	Database Programming	present	2025-02-08
Farah Izzati	Business Intro	NULL	NULL
John Lim	Physics I	absent	2025-02-01
Kelvin Chia	Software Engineering	absent	2025-02-01
Kelvin Chia	Software Engineering	excused	2025-02-08
Lily Chan	Chemistry I	NULL	NULL
Mary Lee	English Communication	late	2025-02-01
Nur Aina	Operating Systems	late	2025-02-01
Nur Aina	Operating Systems	present	2025-02-08
Samuel Tan	Biology I	NULL	NULL
Siti Aminah	Database Programming	present	2025-02-01

Figure 3.23: LEFT OUTER JOIN Result

```
320 •    SELECT
321          s1.fullname AS "Student 1",
322          s2.fullname AS "Student 2",
323          c.course_name AS "Course Name",
324          l.fullname AS "Lecturer Name"
325      FROM enrollment e1
326      JOIN enrollment e2
327          ON e1.sec_id = e2.sec_id
328          AND e1.stud_id != e2.stud_id
329      JOIN student s1 ON e1.stud_id = s1.stud_id
330      JOIN student s2 ON e2.stud_id = s2.stud_id
331      JOIN section sec ON e1.sec_id = sec.sec_id
332      JOIN course c ON sec.course_id = c.course_id
333      JOIN lecturer l ON sec.lect_id = l.lect_id;
```

Result Grid | Filter Rows: | Export: | Wrap

Student 1	Student 2	Course Name	Lecturer Name
Ali Hasan	Siti Aminah	Database Programming	Dr. Afif Shaqir
Siti Aminah	Ali Hasan	Database Programming	Dr. Afif Shaqir

Figure 3.24: SELF JOIN Query

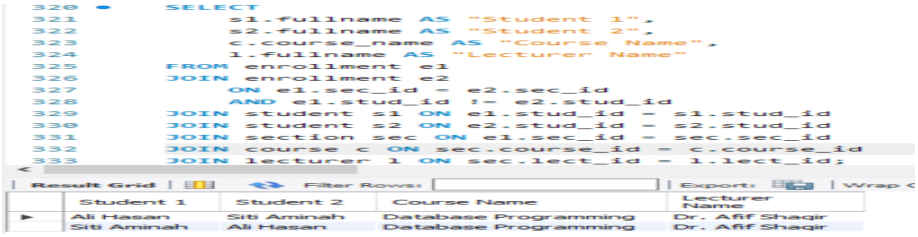


Figure 3.25: SELF JOIN Result

Figure 3.18 to 3.25 demonstrate various types of SQL joins used to retrieve related data from multiple tables in the attendance system. Figures 3.18 and 3.19 show a NATURAL JOIN between the student, enrollment and section tables, automatically linking columns with the same name and displaying students along with their enrolled sections. Figures 3.20 and 3.21 illustrate an INNER JOIN that retrieves detailed information including student name, course name, lecturer name and enrollment date, returning only records with matching entries across the tables. Figures 3.22 and 3.23 demonstrate a LEFT OUTER JOIN to include all students even if they have no attendance data. Finally, Figures 3.24 and 3.25 use a SELF JOIN on the enrollment table to pair students within the same section.

4.0 Optimization Section

4.1 BTREE Index

```
CREATE INDEX idx_attendance_history
        ON attendance(status, mark_time)
        USING BTREE;
```

Figure 4.1 Attendance History

Key usage	Before Index	After Index
Results		

Rows examined	10 rows	1 row
Possible key	NULL	idx_attendance_history

Table 4.1 Before vs After Index for Attendance History

Table 4.1, the query executed as a full table scan which examined all 10 attendance records without any possible keys available. It forced the query to sequentially check every record against both mark_time range and status conditions. Meanwhile, the after index query optimized the composite index on status and mark_time, performed a more efficient range scan that examined 1 relevant record. With the implementation of BTREE index in Figure 4.1, it transforms the operation from 10% to 100% filtering efficiency and reduction in total rows examined, showing that the index is perfectly align with the query pattern and its ability to deliver targeted data retrieval for attendance history.

4.2 FULLTEXT Index

```
CREATE FULLTEXT INDEX idx_stud
    ON student(fullname, email);
```

Figure 4.2 Student Search

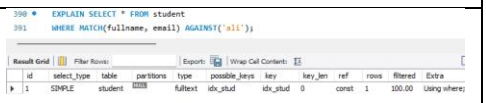
Key usage	Before Index	After Index
Results		
Rows examined	10 rows	1 row
Possible key	NULL	idx_stud

Table 4.2 Before vs After Index for Student Search

Table 4.2, before performing fulltext indexing, the search operation used a slow LIKE pattern to match ‘ali’ with all 10 student records using a full table scan, and there are no possible keys included to support the index. Meanwhile, the after index query performs a fulltext to specialize the text searching by examining only 1 relevant row, replacing the inefficient pattern matching across name and email fields.

```
CREATE FULLTEXT INDEX idx_course
    ON course(course_name);
```

Figure 4.3 Course Search

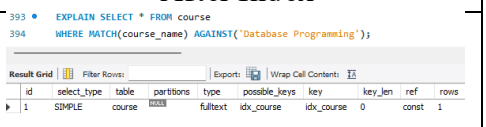
Key usage	Before Index	After Index
Results		
Rows examined	10 rows	1 row
Possible key	NULL	idx_course

Table 4.3 Before vs After Index for Course Search

Table 4.3, before implementing fulltext indexing, the course search relied on inefficient LIKE operators with a full table scan of all 10 course records and no possible keys to support the index. After implementing idx_course fulltext index, the query executes using a relevant text search by examining just 1 matching record. Thus, it transforms the slow pattern matching into faster, intelligence course discovery through advanced text indexing capabilities.

5.0 Team Work

Anis Safiyya - DDL Developer

Responsibilities:

- Designed database schema including determining entities, attribute and entity relationships.
- Developed the DDL statements using CREATE, ALTER, and DROP status
- Created all the CREATE, ALTER, and DROP statements required to build and alter the database structure in accordance with the project specifications.
- Apply constraints to the data such as PRIMARY KEY, FOREIGN KEY, CHECK, and UNIQUE. This will ensure data integrity and increase consistency.
- Review the table relationship to increase database speed and make it easier to maintain as it is scalable.

Learning Outcomes:

- Gained deeper understanding of database normalization
- Learned importance of constraint planning
- Improved team coordination skills

Afif Shaqir – DML Developer

Responsibilities:

- Designed and implemented DML operations, including inserting complete datasets for every table to ensure that each insertion followed to the constraints defined in the DDL such as FOREIGN KEY, UNIQUE and CHECK.
- Made sure data in correct form by inserting in the right order and checking updates or deletions so they didn't break the links between tables.
- Worked with teammates to check that the database structure was correct and gave feedback so that the DML scripts fit well with the overall design.

Learning Outcomes:

- Learned more on how tables depend on each other, especially how foreign keys affect the order of data operations.
- Improve SQL scripting and handling real data situations like update and records and safely deleting data.
- Improved problem solving by fixing constraints errors, planning the order of inserts, and making sure every change matched the database design
- Developed a better understanding of shared responsibility in a project with team members when dividing and handling the task.

Nurul Asyikin – Query Developer

Responsibilities:

- Wrote detailed SQL queries to retrieve, filter and aggregate data effectively.
- Used SQL clauses such as SELECT, JOIN, WHERE, GROUP BY and ORDER BY to generate accurate results.
- Ensured the accuracy and integrity when merging data from multiple tables.
- Collaborated with team members to understand data requirements and deliver useful insights.

Learning Outcomes:

- Gained better understanding of relational database concepts and table relationships.
- Improved problem-solving and logical thinking when handling complex datasets.
- Improved communication and teamwork skills with others.

Nabil Aflah – Optimization Developer

Responsibilities:

- Used query profiling and EXPLAIN to examine query performance for both before and after indexing.
- Strategically designed and implemented BTREE indexes for range search and specific query patterns.
- Developed FULLTEXT indexes for fast and efficient text searching.
- Enhanced the performance of slow-running queries by utilizing indexes and restructuring the queries.

Learning Outcomes:

- Improved understanding on how to analyze and interpret the performance of SQL queries by using EXPLAIN for before and after indexes.
- Learned that BTREE and FULLTEXT indexes not only cut down the number of rows that are being examined but also makes the queries more efficient.
- I become more knowledgeable about the principle of database indexing and their relationships with query planning and execution.

6.0 Conclusion

The attendance system is a complete solution that not only takes care of student admission management, but also course, section, class, and attendance monitoring. The system can boast an integration of a relational database design with its well-defined tables, primary and foreign keys, and constraints that help in data integrity and consistency throughout the system. It can execute a variety of queries such as data retrieval and indexing, thus aiding the lecturers and administrators to come up with meaningful reports.

The development process faced some difficulties in forming queries that were effective across a multitude of connected tables and in getting the database to return the right information for different cases. Other challenges were the performance of the queries that

were made visible through analysis and comparison and then the decision for indexing based on it. The optimization of BTREE and FULLTEXT indexes helps to increase the speed and efficiency of the queries.

Lastly, the additional functionalities that can be improved to the system are automated notifications, attendance statistics, and user-friendly interfaces for both lecturers and students to enhance the attendance system. Continuous performance assessment and periodic index updates are required to keep the system working efficiently even when the data growth enormously. Overall, the system is reliable and acts as a strong ground for a genuine attendance management solution.