

PYTHON -HEXAWARE FOUNDATION TRAINING

ASSIGNMENT

MYSQL

STUDENT INFORMATION SYSTEM

Name: Mathew P

Batch: 04

TASK -1

DATABASE DESIGN

1.Create the database named “SISDB”

Query: CREATE DATABASE SISDB;

The screenshot shows the MySQL Workbench interface. In the top-left corner, the title bar says "MySQL Workbench Local instance MySQL80". The main area has a "Query 1" tab open with the following SQL code:

```
CREATE DATABASE SISDB;
SHOW DATABASES;
```

Below the query window, the "Result Grid" shows the output of the "SHOW DATABASES" command, listing the databases: information_schema, mysql, performance_schema, sisdb, and sys.

In the bottom-right corner of the result grid, there is a note: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

2. Define the schema for the Students, Courses, Enrollments, Teacher, and Payments tables based on the provided schema. Write SQL scripts to create the mentioned tables with appropriate data types, constraints, and relationships. a. Students b. Courses c. Enrollments d. Teacher e. Payments

Query:

```
USE SISDB;
```

```
CREATE TABLE Teacher(
    teacher_id VARCHAR(10),
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255),
    email VARCHAR(255));
```

```
CREATE TABLE Students(  
    student_id VARCHAR(10),  
    first_name VARCHAR(255) NOT NULL,  
    last_name VARCHAR(255),  
    date_of_birth DATE NOT NULL,  
    email VARCHAR(255),  
    phone_number BIGINT NOT NULL );
```

```
CREATE TABLE Courses(  
    course_id VARCHAR(10),  
    course_name VARCHAR(255) NOT NULL,  
    credits INT,  
    teacher_id VARCHAR(10)  
);
```

```
CREATE TABLE Enrollments(  
    enrollment_id INT PRIMARY KEY,  
    student_id VARCHAR(10),  
    course_id VARCHAR(10),  
    enrollment_date DATE NOT NULL  
);
```

```
CREATE TABLE Payments(  
    payment_id INT ,  
    student_id VARCHAR(10),
```

```
amount INT NOT NULL,  
payment_date DATE NOT NULL);
```

```
SHOW TABLES;
```

Query 1 ×

```
1 • USE SISDB;  
2  
3 • CREATE TABLE Teacher(  
4     teacher_id VARCHAR(10),  
5     first_name VARCHAR(255) NOT NULL,  
6     last_name VARCHAR(255),  
7     email VARCHAR(255) );  
8  
9 • CREATE TABLE Students(  
10    student_id VARCHAR(10),  
11    first_name VARCHAR(255) NOT NULL,  
12    last_name VARCHAR(255),  
13    date_of_birth DATE NOT NULL,  
14    email VARCHAR(255),  
15    phone_number BIGINT NOT NULL );  
16  
17 • CREATE TABLE Courses(  
18    course_id VARCHAR(10),  
19    course_name VARCHAR(255) NOT NULL,  
20    credits INT,
```

Result Grid | Filter Rows: _____ | Export: _____ | Wrap Cell Content: Result 24 × Read Only

Tables_in_sisdb
courses
enrollments
payments
students
teacher

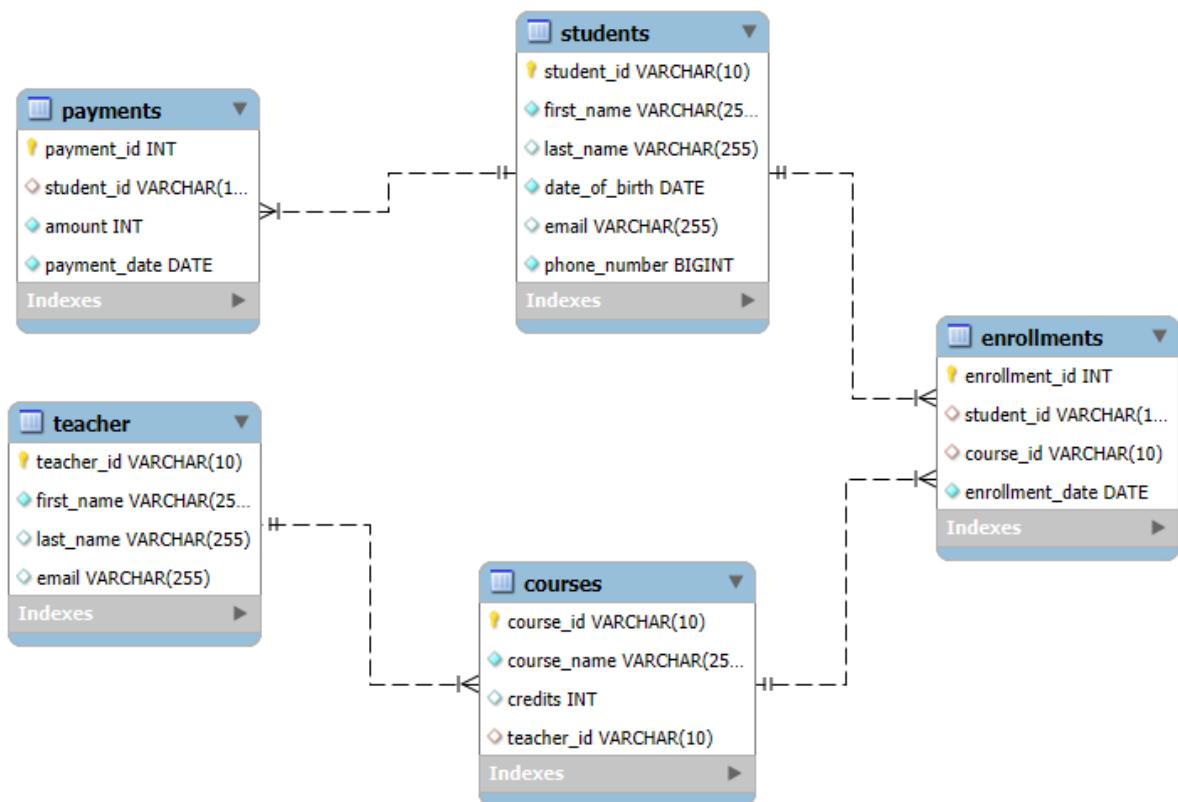
Query 2 ×

```
19    course_name VARCHAR(255) NOT NULL,  
20    credits INT,  
21    teacher_id VARCHAR(10)  
22 );  
23  
24 • CREATE TABLE Enrollments(  
25    enrollment_id INT PRIMARY KEY,  
26    student_id VARCHAR(10),  
27    course_id VARCHAR(10),  
28    enrollment_date DATE NOT NULL  
29 );  
30  
31 • CREATE TABLE Payments(  
32    payment_id INT ,  
33    student_id VARCHAR(10),  
34    amount INT NOT NULL,  
35    payment_date DATE NOT NULL);  
36  
37 • SHOW TABLES;  
38
```

Result Grid | Filter Rows: _____ | Export: _____ | Wrap Cell Content: Result 24 × Read Only

Tables_in_sisdb
courses
enrollments
payments
students
teacher

3. Create an ERD (Entity Relationship Diagram) for the database.



4. Create appropriate Primary Key and Foreign Key constraints for referential integrity.

Query:

Primary key constraint:

```
ALTER TABLE Students ADD CONSTRAINT PRIMARY KEY (student_id);
```

```
ALTER TABLE Courses ADD CONSTRAINT PRIMARY KEY (course_id);
```

```
ALTER TABLE Enrollments ADD CONSTRAINT PRIMARY KEY (enrollment_id);
```

```
ALTER TABLE Teacher ADD CONSTRAINT PRIMARY KEY (teacher_id);
```

```
ALTER TABLE Payments ADD CONSTRAINT PRIMARY KEY (payment_id);
```

Query 1

```

1 • ALTER TABLE Students ADD CONSTRAINT PRIMARY KEY (student_id);
2 • ALTER TABLE Courses ADD CONSTRAINT PRIMARY KEY (course_id);
3 • ALTER TABLE Enrollments ADD CONSTRAINT PRIMARY KEY (enrollment_id);
4 • ALTER TABLE Teacher ADD CONSTRAINT PRIMARY KEY (teacher_id);
5 • ALTER TABLE Payments ADD CONSTRAINT PRIMARY KEY (payment_id);

6
7
8
9
10
11
12
13
14
15

```

Output

Action Output	Message
# Time Action	
1 11:11:19 ALTER TABLE Students ADD CONSTRAINT PRIMARY KEY (student_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
2 11:11:19 ALTER TABLE Courses ADD CONSTRAINT PRIMARY KEY (course_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
3 11:11:19 ALTER TABLE Enrollments ADD CONSTRAINT PRIMARY KEY (enrollment_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
4 11:11:19 ALTER TABLE Teacher ADD CONSTRAINT PRIMARY KEY (teacher_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
5 11:11:19 ALTER TABLE Payments ADD CONSTRAINT PRIMARY KEY (payment_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Foreign Key Constraint:

`ALTER TABLE Courses ADD CONSTRAINT FOREIGN KEY(teacher_id)
REFERENCES Teacher(teacher_id);`

`ALTER TABLE Enrollments ADD CONSTRAINT FOREIGN KEY(student_id)
REFERENCES Students(student_id);`

`ALTER TABLE Enrollments ADD CONSTRAINT FOREIGN KEY(course_id)
REFERENCES Courses(course_id);`

`ALTER TABLE Payments ADD CONSTRAINT FOREIGN KEY(student_id)
REFERENCES Students(student_id);`

Query 1

```

1 • ALTER TABLE Courses ADD CONSTRAINT FOREIGN KEY(teacher_id) REFERENCES Teacher(teacher_id);
2 • ALTER TABLE Enrollments ADD CONSTRAINT FOREIGN KEY(student_id) REFERENCES Students(student_id);
3 • ALTER TABLE Enrollments ADD CONSTRAINT FOREIGN KEY(course_id) REFERENCES Courses(course_id);
4 • ALTER TABLE Payments ADD CONSTRAINT FOREIGN KEY(student_id) REFERENCES Students(student_id);

5
6
7
8
9
10
11
12

```

Output

Action Output	Message
# Time Action	
1 11:21:48 ALTER TABLE Courses ADD CONSTRAINT FOREIGN KEY(teacher_id) REFERENCES Teacher(teacher_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
2 11:21:48 ALTER TABLE Enrollments ADD CONSTRAINT FOREIGN KEY(student_id) REFERENCES Students(student_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
3 11:21:48 ALTER TABLE Enrollments ADD CONSTRAINT FOREIGN KEY(course_id) REFERENCES Courses(course_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
4 11:21:48 ALTER TABLE Payments ADD CONSTRAINT FOREIGN KEY(student_id) REFERENCES Students(student_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

5. Insert at least 10 sample records into each of the following tables. i. Students ii Courses iii. Enrollments iv. Teacher v. Payments

Query:

5.i. Students Table:

```
INSERT INTO Students(student_id, first_name, last_name, date_of_birth, email, phone_number)
VALUES('S001', 'Angella', 'Thomson', '1995-08-12', 'angella@example.com', '9078563412'),
('S002', 'Bellatrix', 'Lestrange', '1993-10-11', 'bella@example.com', '8079685746'),
('S003', 'Hermoine', 'Granger', '1994-11-17', 'hermo@example.com', '3524138079'),
('S004', 'Harry', 'Potter', '1994-10-12', 'potter@example.com', '0987654321'),
('S005', 'Ron', 'Weasly', '1994-05-15', 'ron@example.com', '0099887766'),
('S006', 'Draco', 'Malfoy', '1994-12-12', 'draco@example.com', '9988776655'),
('S007', 'Tom', 'Riddle', '1995-09-06', 'tom@example.com', '8877665544'),
('S008', 'Ginny', 'Strange', '1995-04-01', 'potter@example.com', '7766554433'),
('S009', 'Luna', 'Lovegood', '1995-07-20', 'potter@example.com', '6655443322'),
('S010', 'Danaerys', 'Targaryenr', '1995-05-17', 'potter@example.com', '5544332211');
```

```
SELECT * FROM Students;
```

The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1". The query editor contains the following SQL code:

```
1 • INSERT INTO Students(student_id, first_name, last_name, date_of_birth, email, phone_number)
2 VALUES('S001', 'Angella', 'Thomson', '1995-08-12', 'angella@example.com', '9078563412'),
3 ('S002', 'Bellatrix', 'Lestrange', '1993-10-11', 'bella@example.com', '8079685746'),
4 ('S003', 'Hermoine', 'Granger', '1994-11-17', 'hermo@example.com', '3524138079'),
5 ('S004', 'Harry', 'Potter', '1994-10-12', 'potter@example.com', '0987654321'),
6 ('S005', 'Ron', 'Weasly', '1994-05-15', 'ron@example.com', '0099887766'),
7 ('S006', 'Draco', 'Malfoy', '1994-12-12', 'draco@example.com', '9988776655'),
8 ('S007', 'Tom', 'Riddle', '1995-09-06', 'tom@example.com', '8877665544'),
9 ('S008', 'Ginny', 'Strange', '1995-04-01', 'potter@example.com', '7766554433'),
10 ('S009', 'Luna', 'Lovegood', '1995-07-20', 'potter@example.com', '6655443322'),
11 ('S010', 'Danaerys', 'Targaryenr', '1995-05-17', 'potter@example.com', '5544332211');
12
13 • SELECT * FROM Students;
```

The "Result Grid" tab displays the 10 inserted student records:

student_id	first_name	last_name	date_of_birth	email	phone_number
S001	Angella	Thomson	1995-08-12	angella@example.com	9078563412
S002	Bellatrix	Lestrange	1993-10-11	bella@example.com	8079685746
S003	Hermoine	Granger	1994-11-17	hermo@example.com	3524138079
S004	Harry	Potter	1994-10-12	potter@example.com	0987654321
S005	Ron	Weasly	1994-05-15	ron@example.com	0099887766
S006	Draco	Malfoy	1994-12-12	draco@example.com	9988776655
S007	Tom	Riddle	1995-09-06	tom@example.com	8877665544
S008	Ginny	Strange	1995-04-01	potter@example.com	7766554433
S009	Luna	Lovegood	1995-07-20	potter@example.com	6655443322
S010	Danaerys	Targaryenr	1995-05-17	potter@example.com	5544332211

The "Output" tab shows the message: "10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0".

5.ii. Courses Table:

```
INSERT INTO Courses(course_id, course_name, credits, teacher_id)
VALUES('C001', 'Biology', 3,'T001'),
('C002', 'Aerospace', 4,'T002' ),
('C003', 'Aeronautical',4, 'T003'),
('C004','Neurology',5,'T004'),
('C005','Electronics',3, 'T005' ),
('C006','Chemical',2, 'T006' ),
('C007','Computer Science',4, 'T007' ),
('C008','Social Science',1, 'T008' ),
('C009','Economics',2,'T009' ),
('C010','Quantum Mechanics',5,'T010' );
```

```
SELECT * FROM Courses;
```

The screenshot shows the MySQL Workbench interface with two queries in the SQL editor:

```
2 • INSERT INTO Courses(course_id, course_name, credits, teacher_id)
3   VALUES('C001', 'Biology', 3,'T001'),
4   ('C002', 'Aerospace', 4,'T002' ),
5   ('C003', 'Aeronautical',4, 'T003'),
6   ('C004','Neurology',5,'T004'),
7   ('C005','Electronics',3, 'T005' ),
8   ('C006','Chemical',2, 'T006' ),
9   ('C007','Computer Science',4, 'T007' ),
10  ('C008','Social Science',1, 'T008' ),
11  ('C009','Economics',2,'T009' ),
12  ('C010','Quantum Mechanics',5,'T010' );
13
14 • SELECT * FROM Courses;
```

The results grid displays the data inserted into the Courses table:

course_id	course_name	credits	teacher_id
C001	Biology	3	T001
C002	Aerospace	4	T002
C003	Aeronautical	4	T003
C004	Neurology	5	T004
C005	Electronics	3	T005
C006	Chemical	2	T006
C007	Computer Science	4	T007
C008	Social Science	1	T008
C009	Economics	2	T009
C010	Quantum Mechanics	5	T010

The status bar at the bottom indicates: Courses 36 x, Output, Action Output, # Time Action, 3 16:33:26 INSERT INTO Courses(course_id, course_name, credits, teacher_id) VALUES(C001, 'Biology', 3, 'T001'), (C002, 'Aeros...', 10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0.

5.iii.Enrollments Table:

```
INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date) VALUES  
(1, 'S001', 'C001', '2024-06-01'),  
(2, 'S002', 'C002', '2024-06-02'),  
(3, 'S003', 'C003', '2024-06-03'),  
(4, 'S004', 'C004', '2024-06-04'),  
(5, 'S005', 'C005', '2024-06-05'),  
(6, 'S006', 'C006', '2024-06-06'),  
(7, 'S007', 'C007', '2024-06-07'),  
(8, 'S008', 'C008', '2024-06-08'),  
(9, 'S009', 'C009', '2024-06-09'),  
(10, 'S010', 'C010', '2024-06-10);
```

```
SELECT*FROM Enrollments;
```

The screenshot shows the MySQL Workbench interface with a query editor and results grid.

Query Editor (Query 1):

```
Query 1 ×  
Limit to 1000 rows  
1 • INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date) VALUES  
2     (1, 'S001', 'C001', '2024-06-01'),  
3     (2, 'S002', 'C002', '2024-06-02'),  
4     (3, 'S003', 'C003', '2024-06-03'),  
5     (4, 'S004', 'C004', '2024-06-04'),  
6     (5, 'S005', 'C005', '2024-06-05'),  
7     (6, 'S006', 'C006', '2024-06-06'),  
8     (7, 'S007', 'C007', '2024-06-07'),  
9     (8, 'S008', 'C008', '2024-06-08'),  
10    (9, 'S009', 'C009', '2024-06-09'),  
11    (10, 'S010', 'C010', '2024-06-10);  
12  
13 • SELECT * FROM Enrollments;  
*4
```

Result Grid:

enrollment_id	student_id	course_id	enrollment_date
1	S001	C001	2024-06-01
2	S002	C002	2024-06-02
3	S003	C003	2024-06-03
4	S004	C004	2024-06-04
5	S005	C005	2024-06-05
6	S006	C006	2024-06-06
7	S007	C007	2024-06-07
8	S008	C008	2024-06-08
9	S009	C009	2024-06-09
10	S010	C010	2024-06-10
NULL	NULL	NULL	NULL

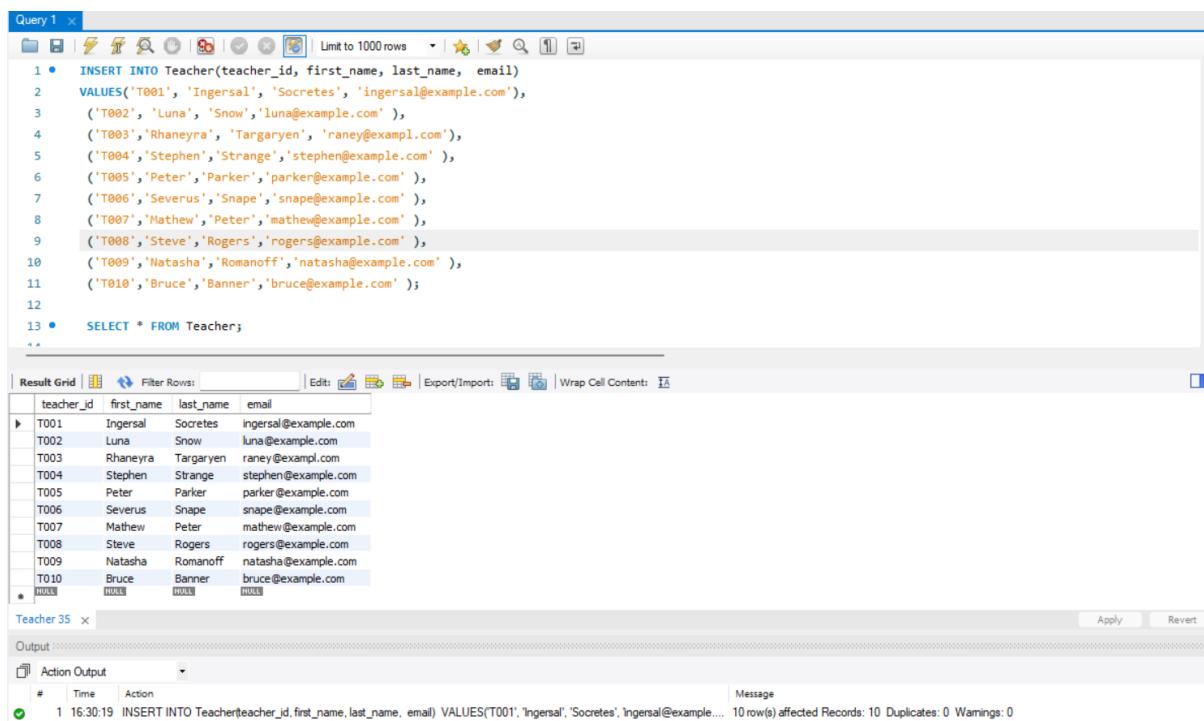
Action Output:

#	Time	Action
5	16:35:48	INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date) VALUES (1, 'S001', 'C001', '2024-06-01... 10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0

5.iv. Teacher table:

```
INSERT INTO Teacher(teacher_id, first_name, last_name, email)
VALUES('T001', 'Ingersal', 'Socrates', 'ingersal@example.com'),
('T002', 'Luna', 'Snow', 'luna@example.com' ),
('T003', 'Rhaneyra', 'Targaryen', 'raney@example.com'),
('T004', 'Stephen', 'Strange', 'stephen@example.com' ),
('T005', 'Peter', 'Parker', 'parker@example.com' ),
('T006', 'Severus', 'Snape', 'snape@example.com' ),
('T007', 'Mathew', 'Peter', 'mathew@example.com' ),
('T008', 'Steve', 'Rogers', 'rogers@example.com' ),
('T009', 'Natasha', 'Romanoff', 'natasha@example.com' ),
('T010', 'Bruce', 'Banner', 'bruce@example.com');
```

```
SELECT * FROM Teacher;
```



The screenshot shows the MySQL Workbench interface with a query editor and results grid.

Query Editor:

```
Query 1
1 • INSERT INTO Teacher(teacher_id, first_name, last_name, email)
2   VALUES('T001', 'Ingersal', 'Socrates', 'ingersal@example.com')
3   ('T002', 'Luna', 'Snow', 'luna@example.com' ),
4   ('T003', 'Rhaneyra', 'Targaryen', 'raney@example.com'),
5   ('T004', 'Stephen', 'Strange', 'stephen@example.com' ),
6   ('T005', 'Peter', 'Parker', 'parker@example.com' ),
7   ('T006', 'Severus', 'Snape', 'snape@example.com' ),
8   ('T007', 'Mathew', 'Peter', 'mathew@example.com' ),
9   ('T008', 'Steve', 'Rogers', 'rogers@example.com' ),
10  ('T009', 'Natasha', 'Romanoff', 'natasha@example.com' ),
11  ('T010', 'Bruce', 'Banner', 'bruce@example.com');
12
13 • SELECT * FROM Teacher;
```

Result Grid:

teacher_id	first_name	last_name	email
T001	Ingersal	Socrates	ingersal@example.com
T002	Luna	Snow	luna@example.com
T003	Rhaneyra	Targaryen	raney@example.com
T004	Stephen	Strange	stephen@example.com
T005	Peter	Parker	parker@example.com
T006	Severus	Snape	snape@example.com
T007	Mathew	Peter	mathew@example.com
T008	Steve	Rogers	rogers@example.com
T009	Natasha	Romanoff	natasha@example.com
T010	Bruce	Banner	bruce@example.com
NULL	NULL	NULL	NULL

Action Output:

#	Time	Action
1	16:30:19	INSERT INTO Teacher(teacher_id, first_name, last_name, email) VALUES('T001', 'Ingersal', 'Socrates', 'ingersal@example.com')

Message: 10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0

5.v. Payments Table:

```
INSERT INTO Payments (payment_id, student_id, amount, payment_date)
VALUES
(1, 'S001', 1000, '2024-06-01'),
(2, 'S002', 2000, '2024-06-02'),
(3, 'S003', 1500, '2024-06-03'),
(4, 'S004', 2500, '2024-06-04'),
(5, 'S005', 1800, '2024-06-05'),
(6, 'S006', 2100, '2024-06-06'),
(7, 'S007', 1700, '2024-06-07'),
(8, 'S008', 2300, '2024-06-08'),
(9, 'S009', 1900, '2024-06-09'),
(10, 'S010', 2200, '2024-06-10');
```

```
SELECT * FROM Payments;
```

The screenshot shows a database interface with two tabs: 'Query 1' and 'Payments 38'. The 'Query 1' tab contains the SQL code for inserting data into the 'Payments' table and a subsequent SELECT statement to retrieve all records. The 'Payments 38' tab displays the results of the SELECT query as a grid, showing 10 rows of payment details. The bottom section of the interface shows the 'Action Output' log, which includes the timestamp of the insert operation and a message indicating 10 rows affected.

payment_id	student_id	amount	payment_date
1	S001	1000	2024-06-01
2	S002	2000	2024-06-02
3	S003	1500	2024-06-03
4	S004	2500	2024-06-04
5	S005	1800	2024-06-05
6	S006	2100	2024-06-06
7	S007	1700	2024-06-07
8	S008	2300	2024-06-08
9	S009	1900	2024-06-09
10	S010	2200	2024-06-10

TASK-02

Select, Where, Between, AND, LIKE:

1. Write an SQL query to insert a new student into the "Students" table with the following details:

- a. First Name: John
- b. Last Name: Doe
- c. Date of Birth: 1995-08-15
- d. Email: john.doe@example.com
- e. Phone Number: 1234567890

Query:

```
INSERT INTO Students(student_id, first_name, last_name, date_of_birth, email, phone_number)
VALUES ('S011', 'John', 'Doe', '1995-08-15', 'john.doe@example.com', 1234567890);
```

```
SELECT * FROM Students;
```

The screenshot shows the MySQL Workbench interface with three panes:

- Query 1:** Contains the SQL statements:

```
1 • INSERT INTO Students(student_id, first_name, last_name, date_of_birth, email, phone_number)
2 VALUES ('S011', 'John', 'Doe', '1995-08-15', 'john.doe@example.com', 1234567890);
3
4 • SELECT * FROM Students;
5
```
- Result Grid:** Displays the contents of the Students table.

student_id	first_name	last_name	date_of_birth	email	phone_number
S001	Angella	Thomson	1995-08-12	angella@example.com	9078563412
S002	Bellatrix	Lestrange	1993-10-11	bella@example.com	8079685746
S003	Hermoine	Granger	1994-11-17	hermo@example.com	3524138079
S004	Harry	Potter	1994-10-12	potter@example.com	987654321
S005	Ron	Weasley	1994-05-15	ron@example.com	99887766
S006	Draco	Malfoy	1994-12-12	draco@example.com	9988776655
S007	Tom	Riddle	1995-09-06	tom@example.com	8877665544
S008	Ginny	Strange	1995-04-01	ginny@example.com	7766554433
S009	Luna	Lovegood	1995-07-20	loveg@example.com	6655443322
S010	Danaerys	Targaryen	1995-05-17	targ@example.com	5544332211
S011	John	Doe	1995-08-15	john.doe@example.com	1234567890
• NULL	NULL	NULL	NULL	NULL	NULL
- Action Output:** Shows the log of actions taken:

#	Time	Action	Message
1	16:54:55	INSERT INTO Students(student_id, first_name, last_name, date_of_birth, email, phone_number) VALUES ('S011', 'John', 'Doe', '1995-08-15', 'john.doe@example.com', 1234567890)	1 row(s) affected
2	16:54:55	SELECT * FROM Students LIMIT 0, 1000	11 row(s) returned

2. Write an SQL query to enroll a student in a course. Choose an existing student and course and insert a record into the "Enrollments" table with the enrollment date.

Query:

```
INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date)
VALUES (11, 'S011', 'C001', '2024-06-11');
```

```
SELECT * FROM Enrollments;
```

The screenshot shows a database interface with two tabs: 'Query 1' and 'Result Grid'. The 'Query 1' tab contains the following SQL code:

```
1 • INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date)
2   VALUES (11, 'S011', 'C001', '2024-06-11');
3
4 • SELECT * FROM Enrollments;
```

The 'Result Grid' tab displays the data from the 'Enrollments' table:

enrollment_id	student_id	course_id	enrollment_date
1	S001	C001	2024-06-01
2	S002	C002	2024-06-02
3	S003	C003	2024-06-03
4	S004	C004	2024-06-04
5	S005	C005	2024-06-05
6	S006	C006	2024-06-06
7	S007	C007	2024-06-07
8	S008	C008	2024-06-08
9	S009	C009	2024-06-09
10	S010	C010	2024-06-10
11	S011	C001	2024-06-11
*	NULL	NULL	NULL

3. Update the email address of a specific teacher in the "Teacher" table. Choose any teacher and modify their email address.

Query:

```
UPDATE Teacher SET email = 'snape_updated@example.com' WHERE teacher_id = 'T006';
```

```
SELECT * FROM Teacher;
```

Query 1

```
1 UPDATE Teacher SET email = 'snape_updated@example.com' WHERE teacher_id = 'T006';
2 • SELECT * FROM Teachers;
```

Result Grid

teacher_id	first_name	last_name	email
T001	Ingersal	Socrates	ingersal@example.com
T002	Luna	Snow	luna@example.com
T003	Rhanevra	Targaryen	raney@example.com
T004	Stephen	Strange	stephen@example.com
T005	Peter	Parker	parker@example.com
T006	Severus	Snape	snape_updated@example.com
T007	Mathew	Peter	mathew@example.com
T008	Steve	Rogers	rogers@example.com
T009	Natasha	Romanoff	natasha@example.com
T010	Bruce	Banner	bruce@example.com
• NULL	NULL	NULL	NULL

Teacher 44

Output

Action Output

#	Time	Action	Message
1	17:19:50	UPDATE Teacher SET email = 'snape_updated@example.com' WHERE teacher_id = 'T006'	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
2	17:19:50	SELECT * FROM Teacher LIMIT 0, 1000	10 row(s) returned

4. Write an SQL query to delete a specific enrollment record from the "Enrollments" table. Select an enrollment record based on the student and course.

Query:

```
DELETE FROM Enrollments WHERE student_id = 'S004' AND course_id = 'C004';
```

```
SELECT * FROM Enrollments;
```

Query 1

```
1 • DELETE FROM Enrollments WHERE student_id = 'S004' AND course_id = 'C004';
2 • SELECT * FROM Enrollments;
3 |
```

Result Grid

enrollment_id	student_id	course_id	enrollment_date
1	S001	C001	2024-06-01
2	S002	C002	2024-06-02
3	S003	C003	2024-06-03
5	S005	C005	2024-06-05
6	S006	C006	2024-06-06
7	S007	C007	2024-06-07
8	S008	C008	2024-06-08
9	S009	C009	2024-06-09
10	S010	C010	2024-06-10
11	S011	C001	2024-06-11
• NULL	NULL	NULL	NULL

Enrollments 46

Output

Action Output

#	Time	Action	Message
1	17:26:16	DELETE FROM Enrollments WHERE student_id = 'S004' AND course_id = 'C004'	1 row(s) affected
2	17:26:16	SELECT * FROM Enrollments LIMIT 0, 1000	10 row(s) returned

5. Update the "Courses" table to assign a specific teacher to a course. Choose any course and teacher from the respective tables.

Query:

```
UPDATE Courses SET teacher_id = 'T007' WHERE course_id = 'C010';
```

```
SELECT * FROM Courses;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a 'Query 1' tab with the following content:

```
1 • UPDATE Courses SET teacher_id = 'T007' WHERE course_id = 'C010';
2 • SELECT * FROM Courses;
3 •
```

Below this is a 'Result Grid' showing the contents of the 'Courses' table:

course_id	course_name	credits	teacher_id
C001	Biology	3	T001
C002	Aerospace	4	T002
C003	Aeronautical	4	T003
C004	Neurology	5	T004
C005	Electronics	3	T005
C006	Chemical	2	T006
C007	Computer Science	4	T007
C008	Social Science	1	T008
C009	Economics	2	T009
C010	Quantum Mechanics	5	T007

At the bottom, there is a 'Courses 48' tab showing the activity log:

#	Time	Action
1	17:33:47	UPDATE Courses SET teacher_id = 'T007' WHERE course_id = 'C010'
2	17:33:47	SELECT * FROM Courses LIMIT 0, 1000

Message: 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
10 row(s) returned

6. Delete a specific student from the "Students" table and remove all their enrollment records from the "Enrollments" table. Be sure to maintain referential integrity.

Explanation:

To delete a specific student, by considering “Referential Integrity”, we need to delete the child table data of the student and then finally delete the Parent table data of the particular student.

For an illustration, lets consider the deletion of S002 student “Bellatrix Lestrange”.

Step 1: we need to delete the student’s data from Enrollments table (Child table).

Query:

```
DELETE FROM Enrollments WHERE student_id = 'S002';
```

```
SELECT * FROM Enrollments;
```

Query 1 ×

```

1 • DELETE FROM Enrollments WHERE student_id = 'S002';
2 • SELECT * FROM Enrollments;
3

```

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

	enrollment_id	student_id	course_id	enrollment_date
▶	1	S001	C001	2024-06-01
	3	S003	C003	2024-06-03
	5	S005	C005	2024-06-05
	6	S006	C006	2024-06-06
	7	S007	C007	2024-06-07
	8	S008	C008	2024-06-08
	9	S009	C009	2024-06-09
	10	S010	C010	2024-06-10
	11	S011	C001	2024-06-11
*	NULL	NULL	NULL	NULL

Enrollments 50 ×

Output

Action Output	#	Time	Action	Message
3 17:36:26	SELECT * FROM Students LIMIT 0, 1000			11 row(s) returned
4 17:43:40	DELETE FROM Enrollments WHERE student_id = 'S002'			1 row(s) affected
5 17:43:40	SELECT * FROM Enrollments LIMIT 0, 1000			9 row(s) returned

Step 2: Deletion of Student ‘S002’ from the Payments Table (also a Child Table).

Query: DELETE FROM Payments WHERE student_id = 'S002';

SELECT * FROM Payments;

Query 1 ×

```

1 • DELETE FROM Payments WHERE student_id = 'S002';
2 • SELECT * FROM Payments;
3

```

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

	payment_id	student_id	amount	payment_date
▶	1	S001	1000	2024-06-01
	3	S003	1500	2024-06-03
	4	S004	2500	2024-06-04
	5	S005	1800	2024-06-05
	6	S006	2100	2024-06-06
	7	S007	1700	2024-06-07
	8	S008	2300	2024-06-08
	9	S009	1900	2024-06-09
	10	S010	2200	2024-06-10
*	NULL	NULL	NULL	NULL

Payments 52 ×

Output

Action Output	#	Time	Action	Message
6 17:51:29	SELECT * FROM Payments LIMIT 0, 1000			10 row(s) returned
7 17:52:14	DELETE FROM Payments WHERE student_id = 'S002'			1 row(s) affected
8 17:52:14	SELECT * FROM Payments LIMIT 0, 1000			9 row(s) returned

Step 3: Finally we can delete Student ‘S002’ from the parent table “Students table”

Query:

```
DELETE FROM Students WHERE student_id = 'S002';
```

```
SELECT * FROM Students;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a toolbar with various icons. Below the toolbar, the 'Query 1' tab is active, displaying the following SQL code:

```
1 • DELETE FROM Students WHERE student_id = 'S002';
2 • SELECT * FROM Students;
3
```

Below the code, the 'Result Grid' shows the 'Students' table with 11 rows of data. The columns are: student_id, first_name, last_name, date_of_birth, email, and phone_number. The data includes entries for Harry Potter, Ron Weasley, Hermione Granger, Luna Lovegood, and others.

At the bottom, the 'Students 54' tab is active, showing the 'Action Output' pane which logs the following actions:

#	Time	Action	Message
9	17:55:24	SELECT * FROM Students LIMIT 0, 1000	11 row(s) returned
10	17:56:13	DELETE FROM Students WHERE student_id = 'S002'	1 row(s) affected
11	17:56:13	SELECT * FROM Students LIMIT 0, 1000	10 row(s) returned

7. Update the payment amount for a specific payment record in the "Payments" table. Choose any payment record and modify the payment amount.

Query:

```
UPDATE Payments SET amount = 2000 WHERE payment_id = 5;
```

```
SELECT * FROM Payments;
```

Query 1 ×

Limit to 1000 rows

```
1 • UPDATE Payments SET amount = 2000 WHERE payment_id = 5;
2 • SELECT * FROM Payments;
3 |
```

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

	payment_id	student_id	amount	payment_date
▶	1	S001	1000	2024-06-01
	3	S003	1500	2024-06-03
	4	S004	2500	2024-06-04
	5	S005	2000	2024-06-05
	6	S006	2100	2024-06-06
	7	S007	1700	2024-06-07
	8	S008	2300	2024-06-08
	9	S009	1900	2024-06-09
	10	S010	2200	2024-06-10
•	HULL	HULL	HULL	HULL

Payments 56 ×

Output

Action Output

#	Time	Action
1	18:00:41	UPDATE Payments SET amount = 2000 WHERE payment_id = 5
2	18:00:41	SELECT * FROM Payments LIMIT 0, 1000

Message

1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

9 row(s) returned

TASK – 03

Aggregate functions, Having, Order By, GroupBy and Joins:

1. Write an SQL query to calculate the total payments made by a specific student. You will need to join the "Payments" table with the "Students" table based on the student's ID.

Query:

```
SELECT Students.student_id, Students.first_name, Students.last_name,
SUM(Payments.amount) AS Total_Payment FROM Students
JOIN Payments ON Students.student_id = Payments.student_id
WHERE Students.student_id = 'S007'
GROUP BY Students.student_id, Students.first_name, Students.last_name;
```

The screenshot shows the MySQL Workbench interface with three main panes: 'Query 1', 'Result Grid', and 'Result 27'.

Query 1:

```
1  SELECT Students.student_id, Students.first_name, Students.last_name, SUM(Payments.amount) AS Total_Payment FROM Students
2  JOIN Payments ON Students.student_id = Payments.student_id
3  WHERE Students.student_id = 'S007'
4  GROUP BY Students.student_id, Students.first_name, Students.last_name;
```

Result Grid:

student_id	first_name	last_name	Total_Payment
S007	Tom	Riddle	1700

Result 27:

Action Output

#	Time	Action	Message
1	12:30:31	SELECT Students.student_id, Students.first_name, Students.last_name, SUM(Payments.amount) AS Total_P... 1 row(s) returned	

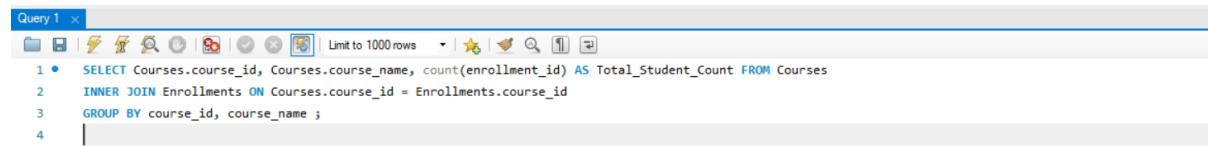
2. Write an SQL query to retrieve a list of courses along with the count of students enrolled in each course. Use a JOIN operation between the "Courses" table and the "Enrollments" table.

Query:

```
SELECT Courses.course_id, Courses.course_name, count(enrollment_id) AS Total_Student_Count FROM Courses
```

```
INNER JOIN Enrollments ON Courses.course_id = Enrollments.course_id
```

```
GROUP BY course_id, course_name ;
```



```
Query 1 ×
SELECT Courses.course_id, Courses.course_name, count(enrollment_id) AS Total_Student_Count FROM Courses
INNER JOIN Enrollments ON Courses.course_id = Enrollments.course_id
GROUP BY course_id, course_name ;
```



course_id	course_name	Total_Student_Count
C001	Biology	2
C003	Aeronautical	1
C005	Electronics	1
C006	Chemical	1
C007	Computer Science	1
C008	Social Science	1
C009	Economics	1
C010	Quantum Mechanics	1



Result 3 × Read Only

#	Time	Action
1	10:30:41	SELECT Courses.course_id, Courses.course_name, count(enrollment_id) AS Total_Student_Count FROM Courses INNER JOIN Enrollments ON Courses.course_id = Enrollments.course_id GROUP BY course_id, course_name ; 8 row(s) returned

3. Write an SQL query to find the names of students who have not enrolled in any course. Use a LEFT JOIN between the "Students" table and the "Enrollments" table to identify students without enrollments.

Query:

```
SELECT Students.student_id, Students.first_name, Students.last_name FROM Students
```

```
LEFT JOIN Enrollments ON Students.student_id = Enrollments.student_id
```

```
WHERE Enrollments.enrollment_id IS NULL;
```

The screenshot shows a SQL query editor interface with three main sections: Query Editor, Result Grid, and Action Output.

Query Editor:

```

Query 1 ×
Limit to 1000 rows
1 • SELECT Students.student_id, Students.first_name, Students.last_name FROM Students
2   LEFT JOIN Enrollments ON Students.student_id = Enrollments.student_id
3   WHERE Enrollments.enrollment_id IS NULL;
4

```

Result Grid:

student_id	first_name	last_name
5004	Harry	Potter

Action Output:

#	Time	Action	Message
1	10:45:51	SELECT Students.student_id, Students.first_name, Students.last_name FROM Students LEFT JOIN Enrollments ON Stu...	1 row(s) returned

4. Write an SQL query to retrieve the first name, last name of students, and the names of the courses they are enrolled in. Use JOIN operations between the "Students" table and the "Enrollments" and "Courses" tables.

Query:

```
SELECT Students.first_name, Students.last_name, Courses.course_name FROM Students
```

```
JOIN Enrollments ON Students.student_id = Enrollments.student_id
```

```
JOIN Courses ON Enrollments.course_id = Courses.Course_id
```

```
ORDER BY Students.first_name ASC;
```

Query 1 x

1 • SELECT Students.first_name, Students.last_name, Courses.course_name FROM Students
 2 JOIN Enrollments ON Students.student_id = Enrollments.student_id
 3 JOIN Courses ON Enrollments.course_id = Courses.Course_id
 4 ORDER BY Students.first_name ASC;
 5

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

	first_name	last_name	course_name
▶	Angella	Thomson	Biology
	Danaeys	Targaryen	Quantum Mechanics
	Draco	Malfoy	Chemical
	Ginny	Strange	Social Science
	Hermoine	Granger	Aeronautical
	John	Doe	Biology
	Luna	Lovegood	Economics
	Ron	Weasley	Electronics
	Tom	Riddle	Computer Science

Result 10 x Read Only

Output :: Action Output

#	Time	Action	Message
1	10:57:20	SELECT Students.first_name, Students.last_name, Courses.course_name FROM Students JOIN Enrollments ...	9 row(s) returned

5. Create a query to list the names of teachers and the courses they are assigned to. Join the "Teacher" table with the "Courses" table.

Query:

```
SELECT Teacher.first_name, Teacher.last_name, Courses.course_name FROM Teacher
JOIN Courses ON Teacher.teacher_id = Courses.teacher_id
ORDER BY Teacher.first_name ASC;
```

The screenshot shows a MySQL query editor interface. At the top, there is a toolbar with various icons for file operations, search, and help. Below the toolbar, a query window titled "Query 1" contains the following SQL code:

```

1 •  SELECT Teacher.first_name, Teacher.last_name, Courses.course_name FROM Teacher
2   JOIN Courses ON Teacher.teacher_id = Courses.teacher_id
3   ORDER BY Teacher.first_name ASC;
4

```

Below the query window is a "Result Grid" table with three columns: "first_name", "last_name", and "course_name". The data is as follows:

first_name	last_name	course_name
Ingersal	Socrates	Biology
Luna	Snow	Aerospace
Mathew	Peter	Computer Science
Mathew	Peter	Quantum Mechanics
Natasha	Romanoff	Economics
Peter	Parker	Electronics
Rhaneyra	Targaryen	Aeronautical
Severus	Snape	Chemical
Stephen	Strange	Neurology
Steve	Rogers	Social Science

At the bottom of the interface, there is a "Result 11" window showing the output of the query. It includes a "Read Only" status indicator and a log entry:

Output ::

Action Output

#	Time	Action	Message
1	11:03:54	SELECT Teacher.first_name, Teacher.last_name, Courses.course_name FROM Teacher JOIN Courses ON Teacher.teacher_id = Courses.teacher_id ORDER BY Teacher.first_name ASC;	10 row(s) returned

6. Retrieve a list of students and their enrollment dates for a specific course. You'll need to join the "Students" table with the "Enrollments" and "Courses" tables.

Query:

```

SELECT Students.first_name, Students.last_name, Courses.course_name,
Enrollments.enrollment_date FROM Students
JOIN Enrollments ON Students.student_id = Enrollments.student_id
JOIN Courses ON Enrollments.course_id = Courses.course_id
ORDER BY Students.first_name ASC;

```

Query 1

```

1 •  SELECT Students.first_name, Students.last_name, Courses.course_name, Enrollments.enrollment_date FROM Students
2   JOIN Enrollments ON Students.student_id = Enrollments.student_id
3   JOIN Courses ON Enrollments.course_id = Courses.course_id
4   ORDER BY Students.first_name ASC;
5

```

Result Grid

first_name	last_name	course_name	enrollment_date
Angella	Thomson	Biology	2024-06-01
Danaveryn	Targaryen	Quantum Mechanics	2024-06-10
Draco	Malfoy	Chemical	2024-06-06
Ginny	Strange	Social Science	2024-06-08
Hermoine	Granger	Aeronautical	2024-06-03
John	Doe	Biology	2024-06-11
Luna	Lovegood	Economics	2024-06-09
Ron	Weasley	Electronics	2024-06-05
Tom	Riddle	Computer Science	2024-06-07

Result 13

Output

#	Time	Action	Message
1	11:11:37	SELECT Students.first_name, Students.last_name, Courses.course_name, Enrollments.enrollment_date FRO...	9 row(s) returned

7. Find the names of students who have not made any payments. Use a LEFT JOIN between the "Students" table and the "Payments" table and filter for students with NULL payment records.

Query:

```

SELECT Students.first_name, Students.last_name FROM Students
LEFT JOIN Payments ON Students.student_id = Payments.student_id
WHERE Payments.payment_id IS NULL;

```

Query 1 x

```
1  SELECT Students.first_name, Students.last_name FROM Students
2  LEFT JOIN Payments ON Students.student_id = Payments.student_id
3  WHERE Payments.payment_id IS NULL;
4
```

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

first_name	last_name
John	Doe

Result 17 x Read Only

Output ::

Action Output

#	Time	Action	Message
1	11:21:44	SELECT Students.first_name, Students.last_name FROM Students LEFT JOIN Payments ON Students.stude...	1 row(s) returned

8. Write a query to identify courses that have no enrollments. You'll need to use a LEFT JOIN between the "Courses" table and the "Enrollments" table and filter for courses with NULL enrollment records.

Query:

```
SELECT Courses.course_id, Courses.course_name FROM Courses
LEFT JOIN Enrollments ON Courses.course_id = Enrollments.course_id
WHERE Enrollments.enrollment_id IS NULL;
```

The screenshot shows the MySQL Workbench interface with three main panes:

- Query 1**: Contains the SQL query:


```

1 •  SELECT Courses.course_id, Courses.course_name FROM Courses
2   LEFT JOIN Enrollments ON Courses.course_id = Enrollments.course_id
3   WHERE Enrollments.enrollment_id IS NULL;
4
      
```
- Result Grid**: Displays the results of the query as a table:

course_id	course_name
C002	Aerospace
C004	Neurology
- Result 18**: Shows the execution log:

#	Time	Action	Message
1	11:28:56	SELECT Courses.course_id, Courses.course_name FROM Courses LEFT JOIN Enrollments ON Courses.course_id = Enrollments.course_id WHERE Enrollments.enrollment_id IS NULL;	2 row(s) returned

9. Identify students who are enrolled in more than one course. Use a self-join on the "Enrollments" table to find students with multiple enrollment records.

Query:

```

SELECT A.student_id, B.course_id FROM Enrollments A
JOIN Enrollments B ON A.student_id = B.student_id
AND A.enrollment_id <> B.enrollment_id;
      
```

The screenshot shows the MySQL Workbench interface. In the top panel, a query window titled "Query 1" contains the following SQL code:

```
1 •  SELECT A.student_id,B.course_id FROM Enrollments A
2   JOIN Enrollments B ON A.student_id = B.student_id
3   AND A.enrollment_id <> B.enrollment_id;
4
5
```

The bottom panel displays the "Result Grid" with the following schema:

student_id	course_id

In the "Output" section, there is one entry in the "Action Output" log:

#	Time	Action	Message
1	11:59:15	SELECT A.student_id,B.course_id FROM Enrollments A JOIN Enrollments B ON A.student_id = B.student_id ...	0 row(s) returned

At my current data entry, the query result shows empty. So let us insert a extra enrollment for same student.

```
INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date)
```

```
VALUES (12, 'S001', 'C002', '2024-06-12');
```

The screenshot shows the MySQL Workbench interface with three main panes:

- Query 1**: Contains the SQL query:


```

1 •  SELECT A.student_id,B.course_id FROM Enrollments A
2   JOIN Enrollments B ON A.student_id = B.student_id
3   AND A.enrollment_id <> B.enrollment_id;
4
5
      
```
- Result Grid**: Displays the results of the query in a tabular format:

student_id	course_id
S001	C002
S001	C001
- Result 25**: Shows the history of actions taken:

#	Time	Action	Message
1	11:59:15	SELECT A.student_id,B.course_id FROM Enrollments A JOIN Enrollments B ON A.student_id = B.student_id ...	0 row(s) returned
2	12:06:10	INSERT INTO Enrollments (enrollment_id,student_id,course_id,enrollment_date) VALUES (12,'S001','C002...)	1 row(s) affected
3	12:06:37	SELECT A.student_id,B.course_id FROM Enrollments A JOIN Enrollments B ON A.student_id = B.student_id ...	2 row(s) returned

10. Find teachers who are not assigned to any courses. Use a LEFT JOIN between the "Teacher" table and the "Courses" table and filter for teachers with NULL course assignments.

Query:

```
SELECT Teacher.teacher_id , Teacher.first_name, Teacher.last_name From Teacher
```

```
LEFT JOIN Courses ON Teacher.teacher_id = Courses.teacher_id
```

```
WHERE Courses.course_id IS NULL;
```

Query 1 ×

Limit to 1000 rows

```
1 •  SELECT Teacher.teacher_id , Teacher.first_name, Teacher.last_name From Teacher
2   LEFT JOIN Courses ON Teacher.teacher_id = Courses.teacher_id
3   WHERE Courses.course_id IS NULL;
4
```

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

teacher_id	first_name	last_name
T010	Bruce	Banner

Result 26 × Read Only

Output

Action Output

#	Time	Action	Message
1	12:21:12	SELECT Teacher.teacher_id , Teacher.first_name, Teacher.last_name From Teacher LEFT JOIN Courses O...	1 row(s) returned

TASK – 04

Subquery and its type

1. Write an SQL query to calculate the average number of students enrolled in each course. Use aggregate functions and subqueries to achieve this.

Query:

```
SELECT AVG(Course_count) AS Average_Student_Enrollment  
FROM ( SELECT e2.student_id, COUNT(e2.course_id) AS Course_count  
FROM Enrollments e2  
GROUP BY e2.student_id  
) AS Subquery;
```

The screenshot shows the MySQL Workbench interface with two panes. The top pane, titled 'Query 1', contains the SQL code. The bottom pane, titled 'Result Grid', displays the output of the query, which is a single row with the value '1.1111'.

Average_Student_Enrollment
1.1111

The 'Result 4' pane at the bottom shows the action log with one entry:

#	Time	Action	Message
1	09:27:34	SELECT AVG(Course_count) AS Average_Student_Enrollment FROM (SELECT e2.student_id, COUNT(e2....	1 row(s) returned

2. Identify the student(s) who made the highest payment. Use a subquery to find the maximum payment amount and then retrieve the student(s) associated with that amount.

Query:

```
SELECT s.student_id, s.first_name, s.last_name, p.amount
FROM Students s
JOIN Payments p ON s.student_id = p.student_id
WHERE p.amount = ( SELECT MAX(p2.amount) AS Max_Payment
From Payments p2
);
```

The screenshot shows a MySQL query editor window titled "Query 1". The query is identical to the one above. Below the query, the "Result Grid" shows a single row of data:

student_id	first_name	last_name	amount
S004	Harry	Potter	2500

The screenshot shows a MySQL query editor window titled "Result 11". The output section displays the following message:

Action Output

#	Time	Action	Message
1	09:44:19	SELECT s.student_id, s.first_name, s.last_name, p.amount FROM Students s JOIN Payments p ON s.student_id = p.student_id WHERE p.amount = (SELECT MAX(p2.amount) AS Max_Payment From Payments p2);	1 row(s) returned

3. Retrieve a list of courses with the highest number of enrollments. Use subqueries to find the course(s) with the maximum enrollment count.

Query :

```
SELECT c.course_id, c.course_name, COUNT(e.enrollment_id) AS
enroll_count
FROM Courses c
```

```

JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name
HAVING COUNT(e.enrollment_id) = (
    SELECT MAX(course_count) FROM (
        SELECT COUNT(enrollment_id) AS course_count
        FROM Enrollments
        GROUP BY course_id
    ) AS subquery
);

```

Query 1

```

1 •  SELECT c.course_id, c.course_name, COUNT(e.enrollment_id) AS enroll_count
2   FROM Courses c
3   JOIN Enrollments e ON c.course_id = e.course_id
4   GROUP BY c.course_id, c.course_name
5   HAVING COUNT(e.enrollment_id) = (
6       SELECT MAX(course_count) FROM (
7           SELECT COUNT(enrollment_id) AS course_count
8           FROM Enrollments
9           GROUP BY course_id
10      ) AS subquery
11  );
12
13
14
15

```

Result Grid

course_id	course_name	enroll_count
C001	Biology	2

Result 72

#	Time	Action	Message
2	15:19:14	SELECT * FROM Teacher LIMIT 0, 1000	10 row(s) returned
3	15:19:43	SELECT * FROM Courses LIMIT 0, 1000	10 row(s) returned
4	16:03:50	SELECT c.course_id, c.course_name, COUNT(e.enrollment_id) AS enroll_count FROM Courses c JOIN Enrollments e ON c.course_id = e.course_id GROUP BY c.course_id, c.course_name HAVING COUNT(e.enrollment_id) = (SELECT MAX(course_count) FROM (SELECT COUNT(enrollment_id) AS course_count FROM Enrollments GROUP BY course_id) AS subquery);	1 row(s) returned

4. Calculate the total payments made to courses taught by each teacher. Use subqueries to sum payments for each teacher's courses.

Query:

```
SELECT t.teacher_id, t.first_name, t.last_name,
```

```

(SELECT SUM(p.amount)

FROM Payments p

JOIN Enrollments e ON p.student_id = e.student_id

JOIN Courses c ON e.course_id = c.course_id

WHERE c.teacher_id = t.teacher_id

) AS total_payments

FROM Teacher t;

```

The screenshot shows the MySQL Workbench interface. The top window is titled "Query 1" and contains the SQL code provided above. The bottom window is titled "Result Grid" and displays the query results as a table:

teacher_id	first_name	last_name	total_payments
T001	Ingersal	Socrates	1000
T002	Luna	Snow	1000
T003	Rhaneyra	Targaryen	1500
T004	Stephen	Strange	NULL
T005	Peter	Parker	2000
T006	Severus	Snape	2100
T007	Mathew	Peter	3900
T008	Steve	Rogers	2300
T009	Natasha	Romanoff	1900
T010	Bruce	Banner	NULL

The output window below shows the execution log:

```

Output :: Action Output
# Time Action Message
1 15:05:35 SELECT t.teacher_id, t.first_name, t.last_name, (SELECT SUM(p.amount) FROM Payments p JOIN E... 10 row(s) returned

```

5. Identify students who are enrolled in all available courses. Use subqueries to compare a student's enrollments with the total number of courses.

Query:

```
SELECT s.student_id, s.first_name, s.last_name
```

```
FROM Students s
```

```

JOIN Enrollments e ON s.student_id = e.student_id
GROUP BY s.student_id, s.first_name, s.last_name
HAVING COUNT(DISTINCT e.course_id) = (SELECT COUNT(*) FROM Courses);

```

Query 1

```

1 •  SELECT s.student_id, s.first_name, s.last_name
2   FROM Students s
3   JOIN Enrollments e ON s.student_id = e.student_id
4   GROUP BY s.student_id, s.first_name, s.last_name
5   HAVING COUNT(DISTINCT e.course_id) = (SELECT COUNT(*) FROM Courses);
6
7

```

Result Grid

student_id	first_name	last_name
5001	Angella	Thomson

Result 83

Action	Time	Action	Message
2	16:25:21	SELECT DISTINCT e.student_id FROM Enrollments e GROUP BY student_id HAVING COUNT(DISTINCT ...	1 row(s) returned
3	16:26:35	SELECT s.student_id, s.first_name, COUNT(DISTINCT e.course_id) AS enrolled_courses FROM Students s ...	9 row(s) returned
4	16:27:21	SELECT s.student_id, s.first_name, s.last_name FROM Students s JOIN Enrollments e ON s.student_id = e....	1 row(s) returned

6. Retrieve the names of teachers who have not been assigned to any courses. Use subqueries to find teachers with no course assignments.

Query:

```

SELECT t.teacher_id, t.first_name, t.last_name
FROM Teacher t
WHERE t.teacher_id NOT IN (
    SELECT c.teacher_id

```

```

FROM Courses c
WHERE c.teacher_id IS NOT NULL);

```

Query 1

The screenshot shows the MySQL Workbench interface. At the top is the Query 1 window containing the SQL code. Below it is the Result Grid window displaying the output of the query.

```

1 •  SELECT t.teacher_id, t.first_name, t.last_name
2   FROM Teacher t
3   WHERE t.teacher_id NOT IN (
4     SELECT c.teacher_id
5       FROM Courses c
6      WHERE c.teacher_id IS NOT NULL);
7
8
9
10

```

teacher_id	first_name	last_name
T010	Bruce	Banner
NULL	NULL	NULL

Teacher 47

The screenshot shows the Teacher 47 window, which displays the log of actions taken. It shows a single entry for the executed query.

#	Time	Action	Message
1	13:15:25	SELECT t.teacher_id, t.first_name, t.last_name FROM Teacher t WHERE t.teacher_id NOT IN (SELECT c.teacher_id FROM Courses c WHERE c.teacher_id IS NOT NULL);	1 row(s) returned

7. Calculate the average age of all students. Use subqueries to calculate the age of each student based on their date of birth.

Query:

```
SELECT AVG(Age) AS Average_Student_Age
```

```
FROM ( SELECT s.student_id, TIMESTAMPDIFF(YEAR,s.date_of_birth,
CURDATE()) AS Age
```

```
FROM Students s
```

```
GROUP BY s.student_id ) AS Subquery;
```

Query 1

```
1 • SELECT AVG(Age) AS Average_Student_Age
2   FROM ( SELECT s.student_id, TIMESTAMPDIFF(YEAR,s.date_of_birth, CURDATE()) AS Age
3         FROM Students s
4       GROUP BY s.student_id ) AS Subquery;
5
6
7
8
9
10
```

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

Average_Student_Age
29.7000

Result 45

Output

Action Output

#	Time	Action	Message
1	17:58:17	SELECT AVG(Age) AS Average_Student_Age FROM (SELECT s.student_id, TIMESTAMPDIFF(YEAR,s.dat...	1 row(s) returned

8. Identify courses with no enrollments. Use subqueries to find courses without enrollment records.

Query:

```
SELECT c.course_id, c.course_name, c.credits
FROM Courses c
WHERE c.course_id NOT IN (
  SELECT e.course_id
  FROM Enrollments e
  WHERE e.course_id IS NOT NULL );
```

Query 1

```

1 •  SELECT c.course_id, c.course_name, c.credits
2   FROM Courses c
3   WHERE c.course_id NOT IN (  SELECT e.course_id
4     FROM Enrollments e
5     WHERE e.course_id IS NOT NULL );
6
7
8
9
10

```

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

course_id	course_name	credits
C004	Neurology	5
NULL	NULL	NULL

Courses 52

Action Output

#	Time	Action	Message
1	13:29:39	SELECT c.course_id, c.course_name, c.credits FROM Courses c WHERE c.course_id NOT IN (SELECT e...	1 row(s) returned

9. Calculate the total payments made by each student for each course they are enrolled in. Use subqueries and aggregate functions to sum payments.

Query:

```

SELECT s.student_id, s.first_name, s.last_name, e.course_id, c.course_name,
p.amount AS Amount_paid

FROM Students s

JOIN Enrollments e ON s.student_id = e.student_id

JOIN Courses c ON e.course_id = c.course_id

JOIN Payments p ON s.student_id = p.student_id

WHERE (s.student_id, p.amount) IN ( SELECT p1.student_id,
SUM(p1.amount) FROM Payments p1 GROUP BY p1.student_id )

ORDER BY s.student_id ASC;

```

Query 1

```

1 •  SELECT s.student_id, s.first_name, s.last_name, e.course_id, c.course_name, p.amount AS Amount_paid
2   FROM Students s
3   JOIN Enrollments e ON s.student_id = e.student_id
4   JOIN Courses c ON e.course_id = c.course_id
5   JOIN Payments p ON s.student_id = p.student_id
6   WHERE (s.student_id , p.amount) IN ( SELECT p1.student_id, SUM(p1.amount) FROM Payments p1 GROUP BY p1.student_id )
7   ORDER BY s.student_id ASC;

```

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

student_id	first_name	last_name	course_id	course_name	Amount_paid
S001	Angella	Thomson	C001	Biology	1000
S001	Angella	Thomson	C002	Aerospace	1000
S001	Angella	Thomson	C003	Aeronautical	1000
S001	Angella	Thomson	C004	Neurology	1000
S001	Angella	Thomson	C005	Electronics	1000
S001	Angella	Thomson	C006	Chemical	1000
S001	Angella	Thomson	C007	Computer Science	1000
S001	Angella	Thomson	C008	Social Science	1000
S001	Angella	Thomson	C009	Economics	1000
S001	Angella	Thomson	C010	Quantum Mechanics	1000
S003	Hermoine	Granger	C003	Aeronautical	1500
S005	Ron	Weasley	C005	Electronics	2000
S006	Draco	Malfoy	C006	Chemical	2100
S007	Tom	Riddle	C007	Computer Science	1700
S008	Ginny	Strange	C008	Social Science	2300
S009	Luna	Lovegood	C009	Economics	1900
S010	Danaerys	Targaryen	C010	Quantum Mechanics	2200

Result 84 × Read Only

Output

#	Time	Action	Message
3	16:26:35	SELECT s.student_id, s.first_name, COUNT(DISTINCT e.course_id) AS enrolled_courses FROM Students s...	9 row(s) returned
4	16:27:21	SELECT e.student_id, e.first_name, e.last_name FROM Students e JOIN Enrollments e ON e.student_id = a...	1 row(s) returned

10. Identify students who have made more than one payment. Use subqueries and aggregate functions to count payments per student and filter for those with counts greater than one.

Query:

```

SELECT s.student_id, s.first_name, s.last_name
FROM Students s
WHERE s.student_id IN (
    SELECT p.student_id
    FROM Payments p
    GROUP BY p.student_id
    HAVING COUNT(p.payment_id) > 1
);

```

Query 1

```

1 • SELECT s.student_id, s.first_name, s.last_name
   Execute the selected portion of the script or everything, if there is no selection
2 FROM Students s
3 WHERE s.student_id IN (
4     SELECT p.student_id
5     FROM Payments p
6     GROUP BY p.student_id
7     HAVING COUNT(p.payment_id) > 1
8 );
9

```

Result Grid

student_id	first_name	last_name
S001	Angella	Thomson
S003	Hermoine	Granger
NULL	NULL	NULL

Students 88

Output

Action Output

#	Time	Action	Message
1	16:39:01	SELECT s.student_id, s.first_name, s.last_name FROM Students s WHERE s.student_id IN (SELECT p.stu... 2 row(s) returned

11. Write an SQL query to calculate the total payments made by each student. Join the "Students" table with the "Payments" table and use GROUP BY to calculate the sum of payments for each student.

Query:

```

SELECT s.student_id, s.first_name, s.last_name, SUM(p.amount) AS
total_payments
FROM Students s
JOIN Payments p ON s.student_id = p.student_id
GROUP BY s.student_id, s.first_name, s.last_name;

```

The screenshot shows the MySQL Workbench environment. At the top is the Query Editor titled "Query 1" containing the following SQL code:

```

1 •  SELECT s.student_id, s.first_name, s.last_name, SUM(p.amount) AS total_payments
2   FROM Students s
3   JOIN Payments p ON s.student_id = p.student_id
4   GROUP BY s.student_id, s.first_name, s.last_name;
5
6

```

Below the Query Editor is the "Result Grid" which displays the following data:

student_id	first_name	last_name	total_payments
S001	Angella	Thomson	2000
S003	Hermoine	Granger	2900
S004	Harry	Potter	2500
S005	Ron	Weasley	2000
S006	Draco	Malfoy	2100
S007	Tom	Riddle	1700
S008	Ginny	Strange	2300
S009	Luna	Lovegood	1900
S010	Danaerys	Targaryenr	2200

At the bottom is the "Action History" titled "Result 90" showing two log entries:

#	Time	Action	Message
2	16:40:22	SELECT s.student_id, s.first_name, s.last_name, SUM(p.amount) AS total_payments FROM Students s JOI...	9 row(s) returned
3	16:40:23	SELECT s.student_id, s.first_name, s.last_name, SUM(p.amount) AS total_payments FROM Students s JOI...	9 row(s) returned

12. Retrieve a list of course names along with the count of students enrolled in each course. Use JOIN operations between the "Courses" table and the "Enrollments" table and GROUP BY to count enrollments.

Query:

```

SELECT      c.course_id,      c.course_name,      COUNT(e.student_id)      AS
Enrollment_Count

FROM Courses c

JOIN Enrollments e ON c.course_id = e.course_id

GROUP BY c.course_id, c.course_name;

```

The screenshot shows the MySQL Workbench environment. At the top is the Query Editor window titled "Query 1" containing the following SQL code:

```

1 •  SELECT c.course_id, c.course_name, COUNT(e.student_id) AS Enrollment_Count
2   FROM Courses c
3   JOIN Enrollments e ON c.course_id = e.course_id
4   GROUP BY c.course_id, c.course_name;
5
6

```

Below the Query Editor is the Result Grid window, which displays the following data:

course_id	course_name	Enrollment_Count
C001	Biology	2
C002	Aerospace	1
C003	Aeronautical	2
C004	Neurology	1
C005	Electronics	2
C006	Chemical	2
C007	Computer Science	2
C008	Social Science	2
C009	Economics	2
C010	Quantum Mechanics	2

At the bottom is the History Grid window titled "Result 91" showing the execution details of the previous query:

#	Time	Action	Message
3	16:40:23	SELECT s.student_id, s.first_name, s.last_name, SUM(p.amount) AS total_payments FROM Students s JOIN Payments p ON s.student_id = p.student_id GROUP BY s.student_id;	9 row(s) returned
4	16:42:24	SELECT c.course_id, c.course_name, COUNT(e.student_id) AS Enrollment_Count FROM Courses c JOIN Enrollments e ON c.course_id = e.course_id GROUP BY c.course_id, c.course_name;	10 row(s) returned

13. Calculate the average payment amount made by students. Use JOIN operations between the "Students" table and the "Payments" table and GROUP BY to calculate the average.

Query:

```

SELECT AVG(Total_amount) AS Average_Payment_Amount
FROM(SELECT s.student_id, Sum(p.amount) AS Total_amount
FROM Students s
JOIN Payments p ON s.student_id = p.student_id
GROUP BY s.student_id) AS Subquery;

```

Query 1 ×

The screenshot shows a database query editor window titled "Query 1". The query itself is as follows:

```
1 •  SELECT AVG(Total_amount) AS Average_Payment_Amount
2   FROM(SELECT s.student_id, Sum(p.amount) AS Total_amount
3   FROM Students s
4   JOIN Payments p ON s.student_id = p.student_id
5   GROUP BY s.student_id) AS Subquery;
6
7
```

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

A single row of data is displayed in the result grid:

Average_Payment_Amount
2177.7778

Result 92 × Read Only

Output ::

Action Output

#	Time	Action	Message
4	16:42:24	SELECT c.course_id, c.course_name, COUNT(e.student_id) AS Enrollment_Count FROM Courses c JOIN Enrollment e ON c.course_id = e.course_id GROUP BY c.course_id, c.course_name;	10 row(s) returned
5	16:43:33	SELECT AVG(Total_amount) AS Average_Payment_Amount FROM(SELECT s.student_id, Sum(p.amount) AS Total_amount FROM Students s JOIN Payments p ON s.student_id = p.student_id GROUP BY s.student_id) AS Subquery;	1 row(s) returned