SOLUTION DETAILS

CONDUCTED BY: DEENA AL BUSAIDI

DATE: 29 JUNE 2025

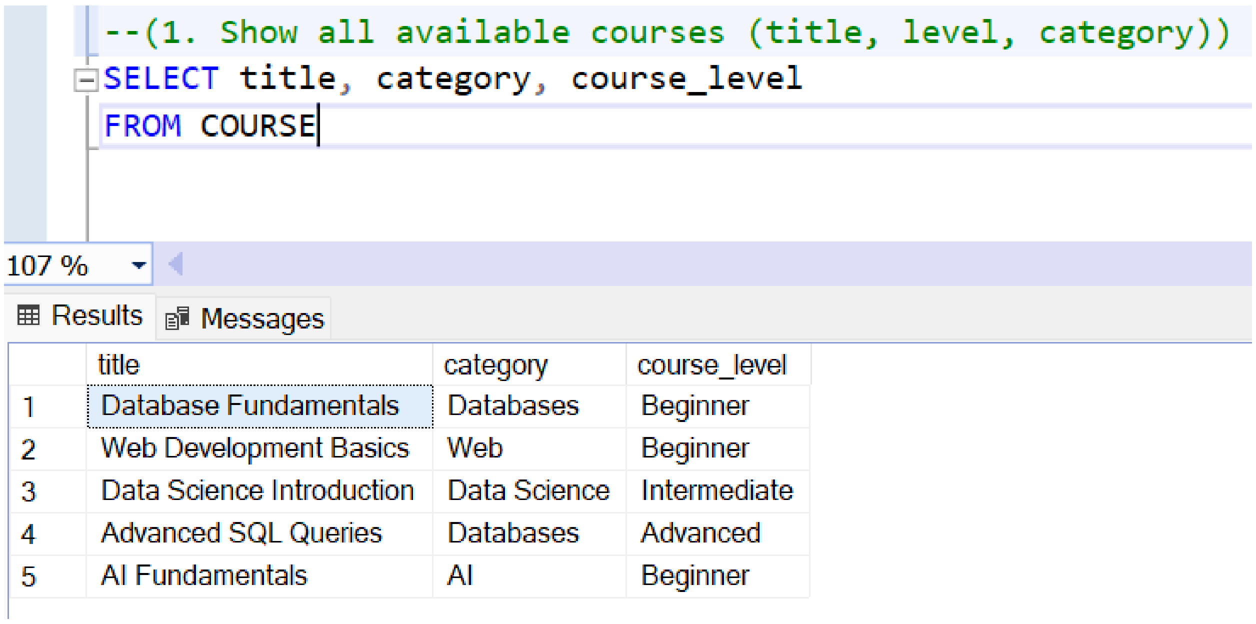
Contents

Trainee Perspective

QUESTION 1

Show all available courses (title, level, category)

- **Retrieve a list of all courses offered by the institute, including the course title, level (e.g., Beginner), and category (e.g., Web, Databases).**



```
--(1. Show all available courses (title, level, category))
SELECT title, category, course_level
FROM COURSE
```

107 %

Results | Messages

	title	category	course_level
1	Database Fundamentals	Databases	Beginner
2	Web Development Basics	Web	Beginner
3	Data Science Introduction	Data Science	Intermediate
4	Advanced SQL Queries	Databases	Advanced
5	AI Fundamentals	AI	Beginner

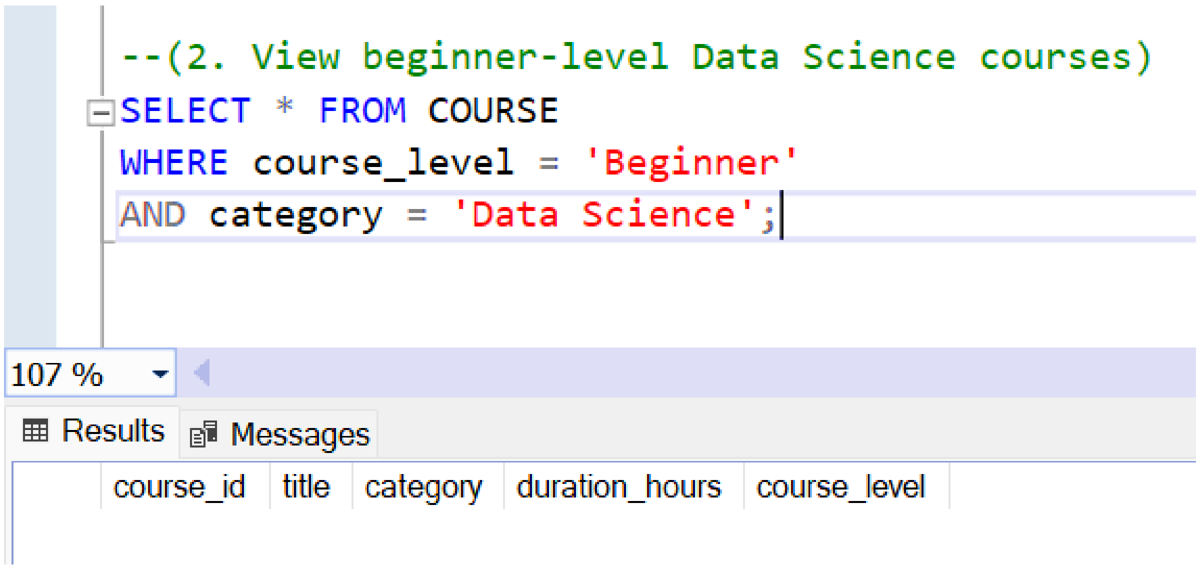
EXPLANATION:

I chose to write the SQL code **SELECT title, category, course_level FROM COURSE;** because the question asks to show all available courses, including their title, level, and category. This code helps me get exactly that information from the database. The **SELECT** part tells the database what information I want to see. In this case, the course title, category, and level. The **FROM COURSE** part tells the database where to find this information, which is in the table named **COURSE**. This simple query shows a list of all the courses offered by the institute, along with the details needed to answer the question.

QUESTION 2

View beginner-level Data Science courses

- **Filter the course list to only show beginner-level courses in the "Data Science" category.**



```
--(2. View beginner-level Data Science courses)
SELECT * FROM COURSE
WHERE course_level = 'Beginner'
AND category = 'Data Science';
```

107 %

Results Messages

course_id	title	category	duration_hours	course_level

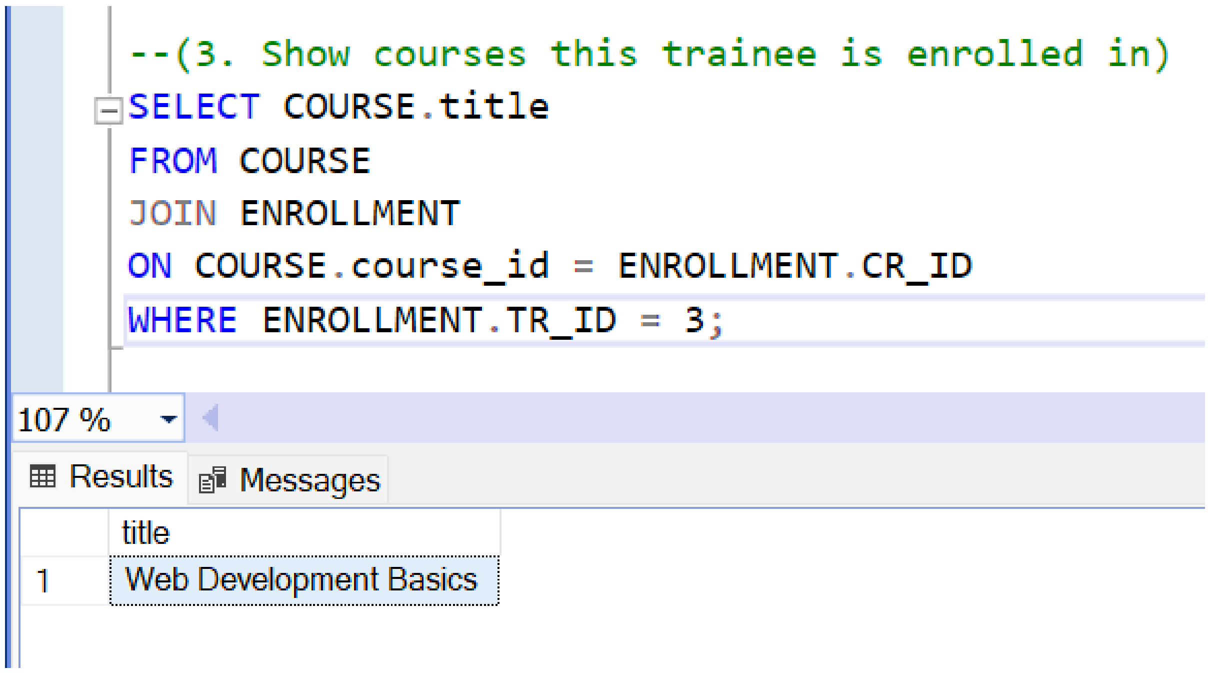
EXPLANATION:

I wrote the SQL code **SELECT * FROM COURSE WHERE course_level = 'Beginner' AND category = 'Data Science';** to answer the question asking for beginner-level courses in the "Data Science" category. I used **SELECT *** to get all the details about the courses that match the conditions. The **FROM COURSE** part tells the database to look in the **COURSE** table. The **WHERE** clause is used to filter the results. I added **course_level = 'Beginner'** to show only beginner-level courses, and **AND category = 'Data Science'** to make sure the courses are from the Data Science category. This way, the code gives exactly the list of courses needed for the question.

QUESTION 3

Show courses this trainee is enrolled in

- **Display the titles of the courses the logged-in trainee has registered for using their trainee ID.**



```
--(3. Show courses this trainee is enrolled in)
SELECT COURSE.title
FROM COURSE
JOIN ENROLLMENT
ON COURSE.course_id = ENROLLMENT.CR_ID
WHERE ENROLLMENT.TR_ID = 3;
```

107 %

Results | Messages

	title
1	Web Development Basics

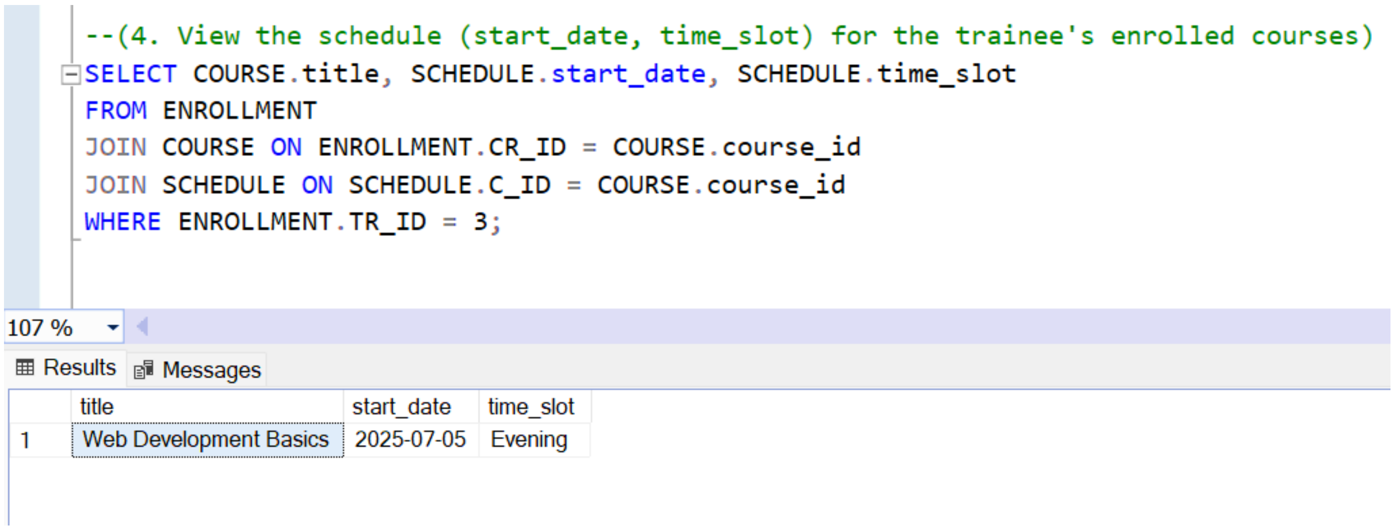
EXPLANATION:

I wrote the SQL code **SELECT COURSE.title FROM COURSE JOIN ENROLLMENT ON COURSE.course_id = ENROLLMENT.CR_ID WHERE ENROLLMENT.TR_ID = 3;** to answer the question that asks for the titles of the courses a specific trainee is enrolled in. This code helps to find only the courses that the trainee with ID number 3 has registered for. I used **SELECT COURSE.title** to show just the names of the courses. Then, I used **FROM COURSE** to start from the table that holds course details. I added a **JOIN** with the **ENROLLMENT** table using **ON COURSE.course_id = ENROLLMENT.CR_ID** to connect each course with its matching enrollment record. Finally, the **WHERE ENROLLMENT.TR_ID = 3** part makes sure the result only includes courses that this particular trainee has enrolled in. This way, the code gives exactly what the question is asking for.

QUESTION 4

View the schedule (start_date, time_slot) for the trainee's enrolled courses

- **Show the scheduled start dates and time slots for all the courses the trainee is enrolled in.**



```
--(4. View the schedule (start_date, time_slot) for the trainee's enrolled courses)
SELECT COURSE.title, SCHEDULE.start_date, SCHEDULE.time_slot
FROM ENROLLMENT
JOIN COURSE ON ENROLLMENT.CR_ID = COURSE.course_id
JOIN SCHEDULE ON SCHEDULE.C_ID = COURSE.course_id
WHERE ENROLLMENT.TR_ID = 3;
```

	title	start_date	time_slot
1	Web Development Basics	2025-07-05	Evening

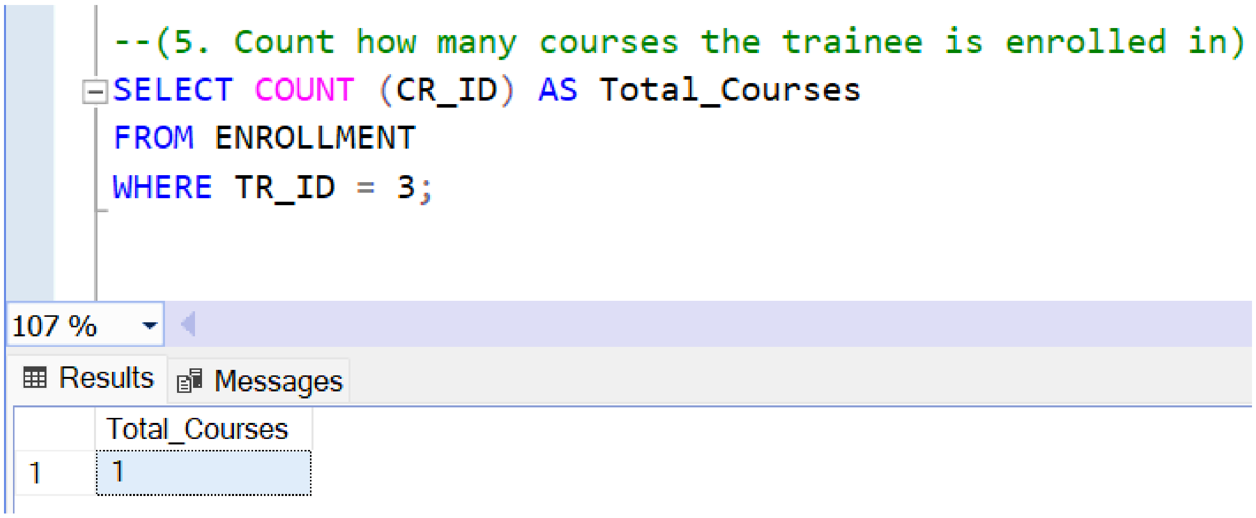
EXPLANATION:

I wrote the SQL code **SELECT COURSE.title, SCHEDULE.start_date, SCHEDULE.time_slot FROM ENROLLMENT JOIN COURSE ON ENROLLMENT.CR_ID = COURSE.course_id JOIN SCHEDULE ON SCHEDULE.C_ID = COURSE.course_id WHERE ENROLLMENT.TR_ID = 3;** to answer the question that asks for the schedule of courses a trainee is enrolled in. This includes the course title, start date, and time slot. I used **SELECT COURSE.title, SCHEDULE.start_date, SCHEDULE.time_slot** to show the needed information. Then I joined the **ENROLLMENT** table with the **COURSE** table using **ENROLLMENT.CR_ID = COURSE.course_id** to find the courses the trainee is registered for. I also joined the **SCHEDULE** table using **SCHEDULE.C_ID = COURSE.course_id** to get the schedule details for those courses. Finally, I used **WHERE ENROLLMENT.TR_ID = 3** to filter the results for trainee number 3. This code gives the schedule information for all the courses that the trainee is currently enrolled in.

QUESTION 5

Count how many courses the trainee is enrolled in

- ➢ **Write a query that returns the number of courses the trainee is currently enrolled in.**



```
--(5. Count how many courses the trainee is enrolled in)
SELECT COUNT (CR_ID) AS Total_Courses
FROM ENROLLMENT
WHERE TR_ID = 3;
```

107 %

Results | Messages

```
   Total_Courses
1  1
```

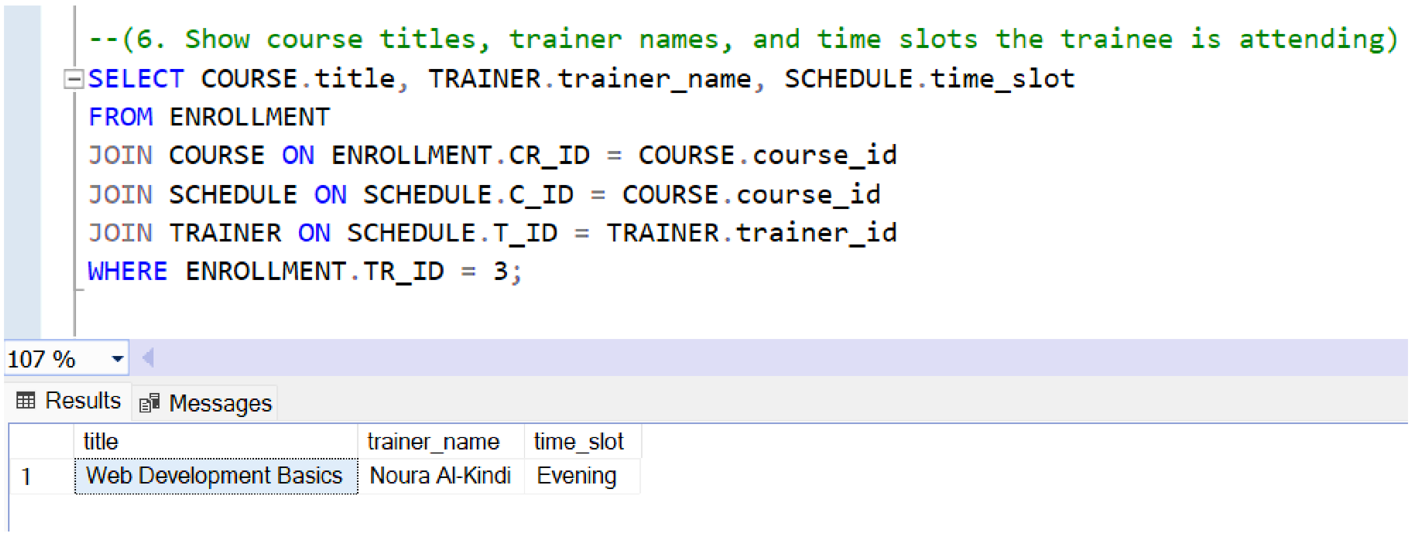
EXPLANATION:

I wrote the SQL code **SELECT COUNT(CR_ID) AS Total_Courses FROM ENROLLMENT WHERE TR_ID = 3;** to answer the question that asks how many courses a trainee is enrolled in. I used **COUNT(CR_ID)** to count the number of courses the trainee has registered for. I gave it a name using **AS Total_Courses** so the result will be easier to read. I used **FROM ENROLLMENT** because the enrollment records are stored in that table. The **WHERE TR_ID = 3** part makes sure we are only counting the courses for trainee number 3. This code gives the total number of courses the trainee is currently enrolled in, just as the question asks.

QUESTION 6

Show course titles, trainer names, and time slots the trainee is attending

- **Join the relevant tables to show which trainer is teaching each course the trainee attends, along with the course title and session time.**



```
--(6. Show course titles, trainer names, and time slots the trainee is attending)
SELECT COURSE.title, TRAINER.trainer_name, SCHEDULE.time_slot
FROM ENROLLMENT
JOIN COURSE ON ENROLLMENT.CR_ID = COURSE.course_id
JOIN SCHEDULE ON SCHEDULE.C_ID = COURSE.course_id
JOIN TRAINER ON SCHEDULE.T_ID = TRAINER.trainer_id
WHERE ENROLLMENT.TR_ID = 3;
```

	title	trainer_name	time_slot
1	Web Development Basics	Noura Al-Kindi	Evening

EXPLANATION:

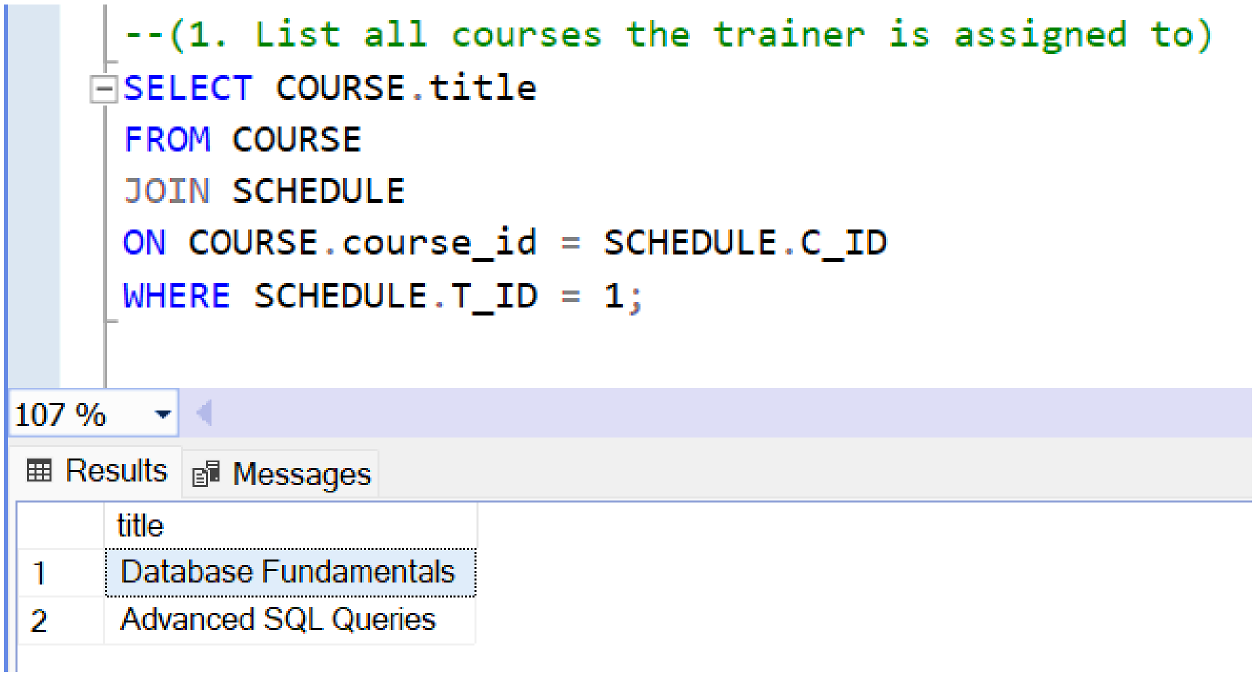
I wrote the SQL code **SELECT COURSE.title, TRAINER.trainer_name, SCHEDULE.time_slot FROM ENROLLMENT JOIN COURSE ON ENROLLMENT.CR_ID = COURSE.course_id JOIN SCHEDULE ON SCHEDULE.C_ID = COURSE.course_id JOIN TRAINER ON SCHEDULE.T_ID = TRAINER.trainer_id WHERE ENROLLMENT.TR_ID = 3;** to answer the question that asks for the course titles, trainer names, and time slots for the courses the trainee is attending. I selected the course title, trainer's name, and the time slot to show all the important details. I joined the **ENROLLMENT** table with the **COURSE** table to find the courses the trainee is enrolled in. Then, I joined the **SCHEDULE** table to get the session times for those courses. Finally, I joined the **TRAINER** table to find out which trainer is teaching each course. The **WHERE ENROLLMENT.TR_ID = 3** part filters the information for trainee number 3. This code shows all the courses the trainee attends, who teaches them, and when the sessions happen.

Trainer Perspective

QUESTION 1

List all courses the trainer is assigned to

- ➢ **Show all course titles where the trainer is responsible for teaching, using the trainer's ID.**



```
--(1. List all courses the trainer is assigned to)
SELECT COURSE.title
FROM COURSE
JOIN SCHEDULE
ON COURSE.course_id = SCHEDULE.C_ID
WHERE SCHEDULE.T_ID = 1;
```

```
107 %

Results   Messages

   title
1  Database Fundamentals
2  Advanced SQL Queries
```

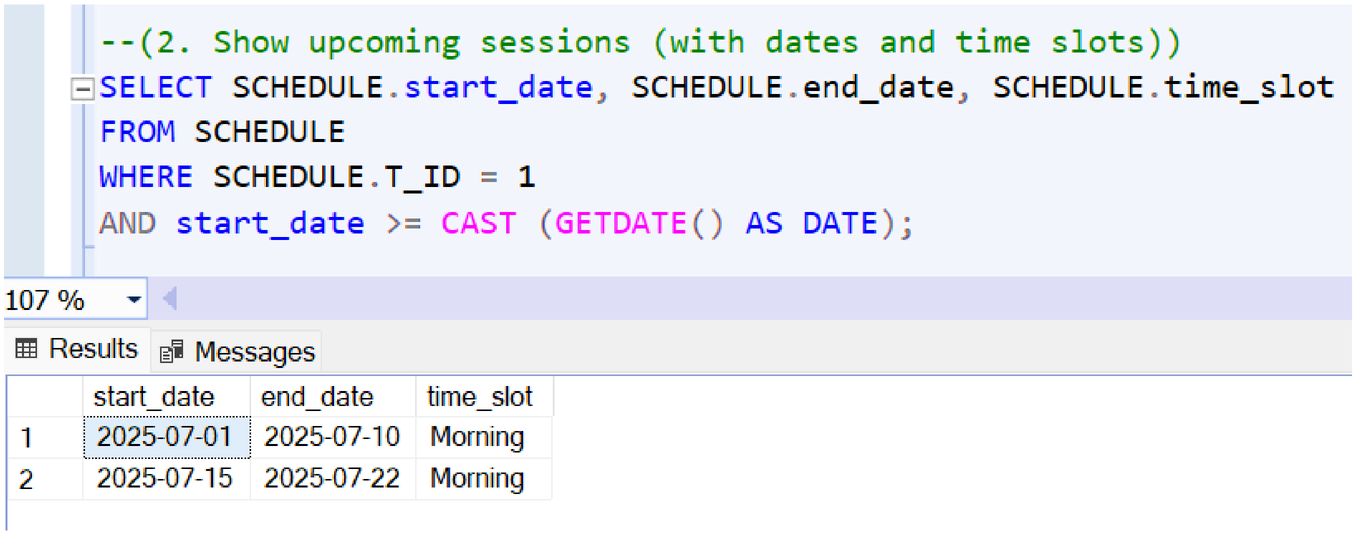
EXPLANATION:

I wrote the SQL code **SELECT COURSE.title FROM COURSE JOIN SCHEDULE ON COURSE.course_id = SCHEDULE.C_ID WHERE SCHEDULE.T_ID = 1;** to answer the question that asks for all courses a trainer is assigned to. I used **SELECT COURSE.title** to show the names of the courses. I joined the **COURSE** table with the **SCHEDULE** table using **ON COURSE.course_id = SCHEDULE.C_ID** to connect each course with its schedule. Then, I used **WHERE SCHEDULE.T_ID = 1** to find only the courses taught by the trainer with ID number 1. This way, the code gives a list of all course titles that the trainer is responsible for teaching.

QUESTION 2

Show upcoming sessions (with dates and time slots)

- **Display future sessions this trainer is conducting, including start and end dates, and the time slot (Morning, Evening, etc.).**



```sql
--(2. Show upcoming sessions (with dates and time slots))
SELECT SCHEDULE.start_date, SCHEDULE.end_date, SCHEDULE.time_slot
FROM SCHEDULE
WHERE SCHEDULE.T_ID = 1
AND start_date >= CAST (GETDATE() AS DATE);
```

107 %

Results | Messages

	start_date	end_date	time_slot
1	2025-07-01	2025-07-10	Morning
2	2025-07-15	2025-07-22	Morning

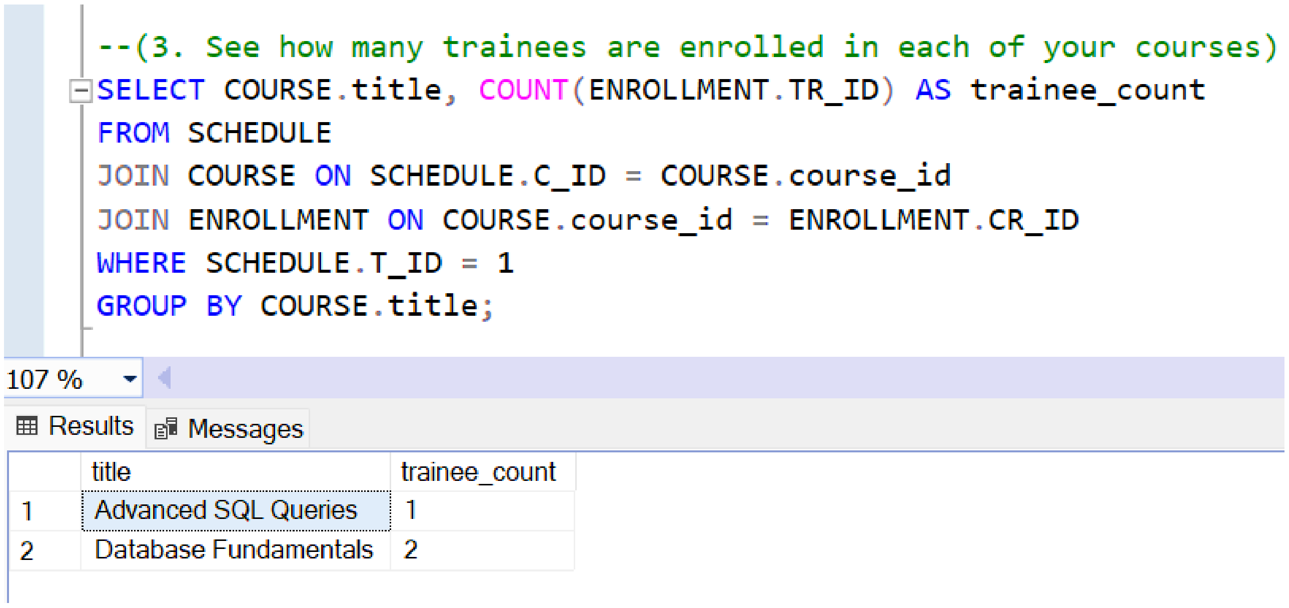
EXPLANATION:

I wrote the SQL code **SELECT SCHEDULE.start_date, SCHEDULE.end_date, SCHEDULE.time_slot FROM SCHEDULE WHERE SCHEDULE.T_ID = 1 AND start_date >= CAST(GETDATE() AS DATE);** to answer the question that asks for upcoming sessions the trainer is teaching. I selected the start date, end date, and time slot to show all the details about these sessions. I used **FROM SCHEDULE** because the schedule information is in that table. The **WHERE SCHEDULE.T_ID = 1** part makes sure the sessions are only for the trainer with ID 1. I also added **AND start_date >= CAST(GETDATE() AS DATE)** to show only future sessions, meaning sessions that start today or later. This code helps to list all upcoming sessions the trainer will conduct with their dates and times.

QUESTION 3

See how many trainees are enrolled in each of your courses

➢ **Count and display how many trainees are registered in each of the trainer's courses.**



```
--(3. See how many trainees are enrolled in each of your courses)
SELECT COURSE.title, COUNT(ENROLLMENT.TR_ID) AS trainee_count
FROM SCHEDULE
JOIN COURSE ON SCHEDULE.C_ID = COURSE.course_id
JOIN ENROLLMENT ON COURSE.course_id = ENROLLMENT.CR_ID
WHERE SCHEDULE.T_ID = 1
GROUP BY COURSE.title;
```

	title	trainee_count
1	Advanced SQL Queries	1
2	Database Fundamentals	2

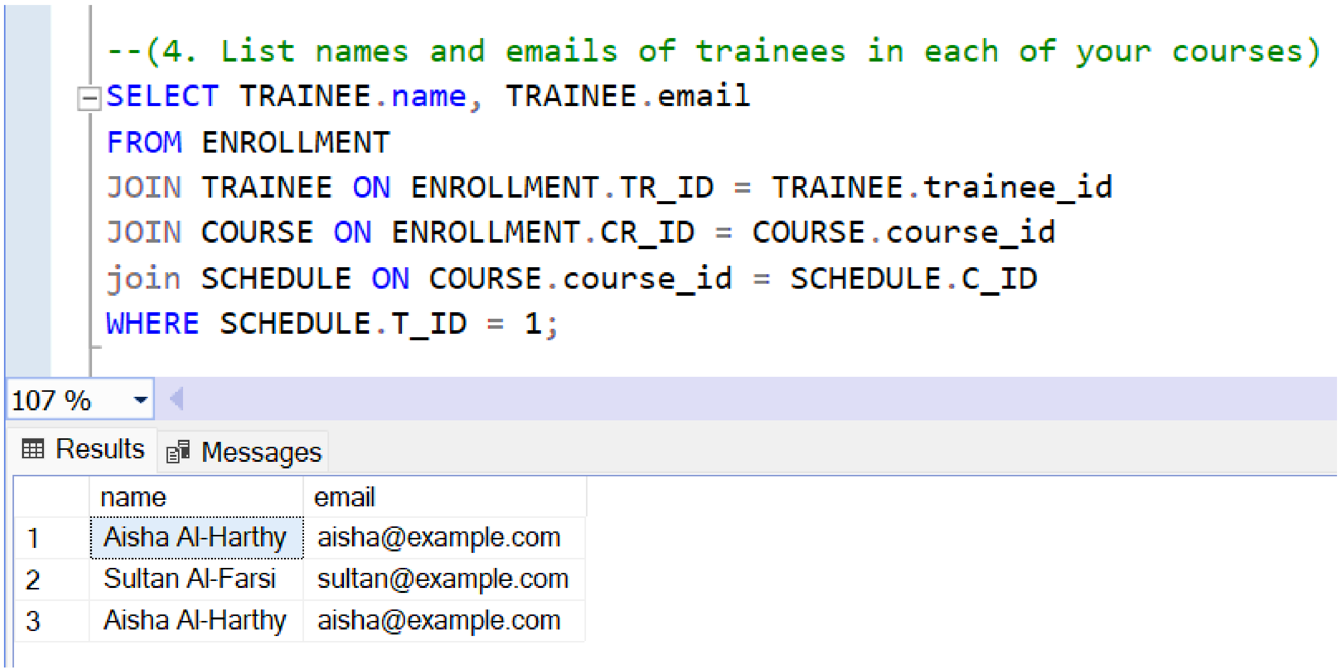
EXPLANATION:

I wrote the SQL code **SELECT COURSE.title, COUNT(ENROLLMENT.TR_ID) AS trainee_count FROM SCHEDULE JOIN COURSE ON SCHEDULE.C_ID = COURSE.course_id JOIN ENROLLMENT ON COURSE.course_id = ENROLLMENT.CR_ID WHERE SCHEDULE.T_ID = 1 GROUP BY COURSE.title;** to answer the question that asks how many trainees are enrolled in each course taught by the trainer. I selected the course title and used **COUNT(ENROLLMENT.TR_ID)** to count the number of trainees in each course. I joined the **SCHEDULE** table with the **COURSE** table to find the courses the trainer teaches, and then joined the **ENROLLMENT** table to count how many trainees are registered for those courses. The **WHERE SCHEDULE.T_ID = 1** part makes sure I only look at courses taught by the trainer with ID 1. I used **GROUP BY COURSE.title** to count the trainees for each course separately. This code shows the number of trainees in every course the trainer teaches.

QUESTION 4

List names and emails of trainees in each of your courses

- **Join Enrollment and Trainee tables to list trainee details for each course taught by the trainer.**



```sql
--(4. List names and emails of trainees in each of your courses)
SELECT TRAINEE.name, TRAINEE.email
FROM ENROLLMENT
JOIN TRAINEE ON ENROLLMENT.TR_ID = TRAINEE.trainee_id
JOIN COURSE ON ENROLLMENT.CR_ID = COURSE.course_id
join SCHEDULE ON COURSE.course_id = SCHEDULE.C_ID
WHERE SCHEDULE.T_ID = 1;
```

	name	email
1	Aisha Al-Harthy	aisha@example.com
2	Sultan Al-Farsi	sultan@example.com
3	Aisha Al-Harthy	aisha@example.com

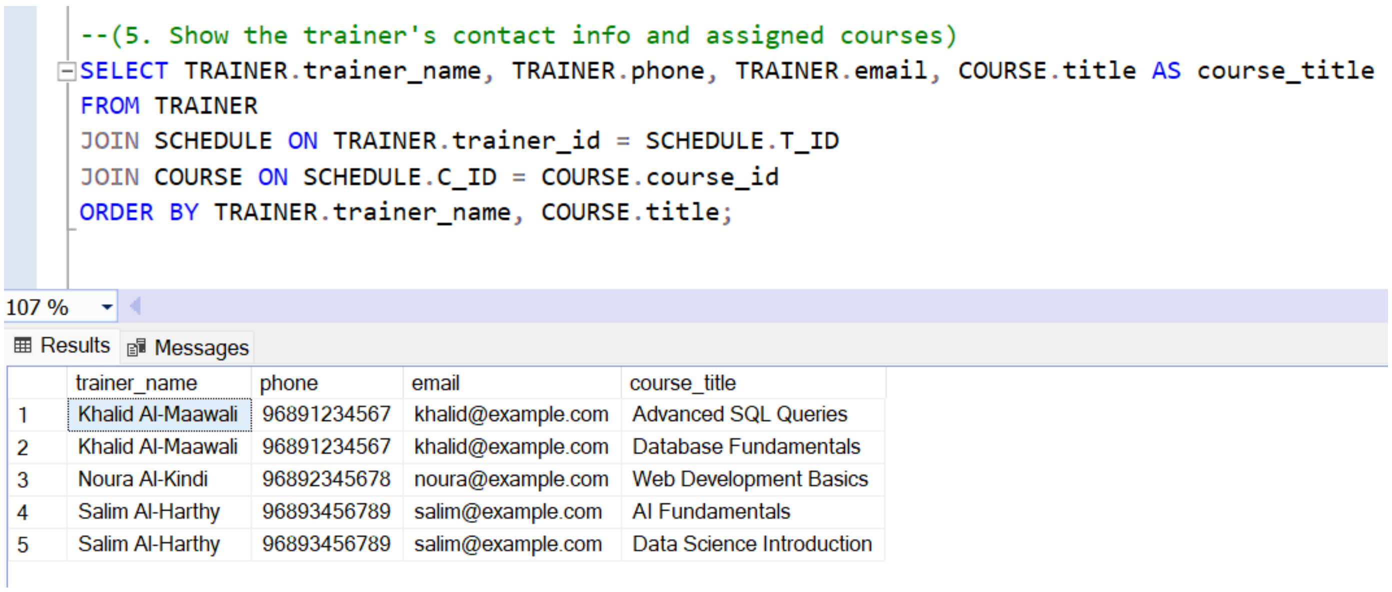
EXPLANATION:

I wrote the SQL code **SELECT TRAINEE.name, TRAINEE.email FROM ENROLLMENT JOIN TRAINEE ON ENROLLMENT.TR_ID = TRAINEE.trainee_id JOIN COURSE ON ENROLLMENT.CR_ID = COURSE.course_id JOIN SCHEDULE ON COURSE.course_id = SCHEDULE.C_ID WHERE SCHEDULE.T_ID = 1;** to answer the question that asks for the names and emails of trainees in the trainer's courses. I selected the trainee's name and email to show their details. I joined the **ENROLLMENT** table with the **TRAINEE** table to connect each enrollment to the trainee's information. Then, I joined the **COURSE** and **SCHEDULE** tables to find the courses the trainer teaches. The **WHERE SCHEDULE.T_ID = 1** part makes sure I only list trainees in courses taught by the trainer with ID 1. This code gives a list of all trainees' names and emails for the trainer's courses.

QUESTION 5

Show the trainer's contact info and assigned courses

- **Return the trainer's phone and email along with a list of their assigned courses.**



```sql
--(5. Show the trainer's contact info and assigned courses)
SELECT TRAINER.trainer_name, TRAINER.phone, TRAINER.email, COURSE.title AS course_title
FROM TRAINER
JOIN SCHEDULE ON TRAINER.trainer_id = SCHEDULE.T_ID
JOIN COURSE ON SCHEDULE.C_ID = COURSE.course_id
ORDER BY TRAINER.trainer_name, COURSE.title;
```

107 %

Results | Messages

	trainer_name	phone	email	course_title
1	Khalid Al-Maawali	96891234567	khalid@example.com	Advanced SQL Queries
2	Khalid Al-Maawali	96891234567	khalid@example.com	Database Fundamentals
3	Noura Al-Kindi	96892345678	noura@example.com	Web Development Basics
4	Salim Al-Harthy	96893456789	salim@example.com	AI Fundamentals
5	Salim Al-Harthy	96893456789	salim@example.com	Data Science Introduction

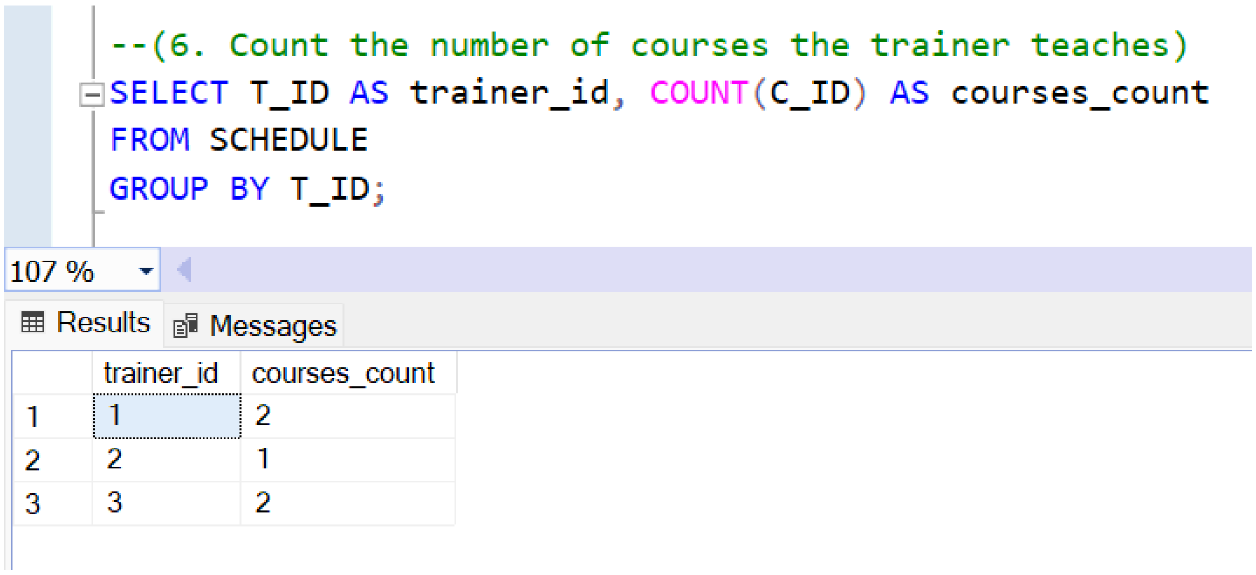
EXPLANATION:

I wrote the SQL code **SELECT TRAINER.trainer_name, TRAINER.phone, TRAINER.email, COURSE.title AS course_title FROM TRAINER JOIN SCHEDULE ON TRAINER.trainer_id = SCHEDULE.T_ID JOIN COURSE ON SCHEDULE.C_ID = COURSE.course_id ORDER BY TRAINER.trainer_name, COURSE.title;** to answer the question that asks for the trainer's contact information and the courses they teach. I selected the trainer's name, phone number, and email to show their contact details. I also selected the course title to show which courses are assigned to each trainer. I joined the **TRAINER** table with the **SCHEDULE** table to connect trainers to their schedules, and then joined the **COURSE** table to get the course details. Finally, I used **ORDER BY** to organize the results by the trainer's name and course title, making the list easy to read. This code gives a clear list of trainers with their contact info and their assigned courses.

QUESTION 6

Count the number of courses the trainer teaches

- **Write a query to find how many different courses the trainer is teaching.**



```
--(6. Count the number of courses the trainer teaches)
SELECT T_ID AS trainer_id, COUNT(C_ID) AS courses_count
FROM SCHEDULE
GROUP BY T_ID;
```

	trainer_id	courses_count
1	1	2
2	2	1
3	3	2

EXPLANATION:

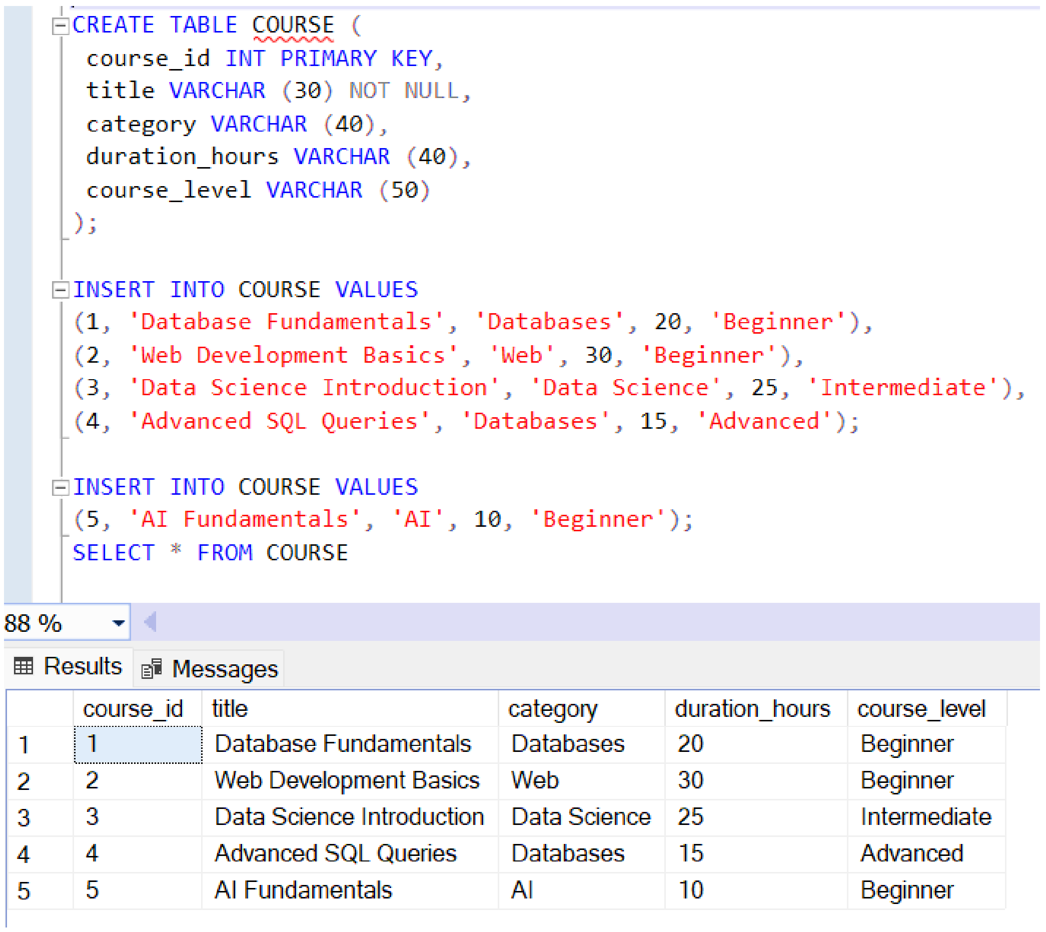
I wrote the SQL code **SELECT T_ID AS trainer_id, COUNT(C_ID) AS courses_count FROM SCHEDULE GROUP BY T_ID;** to answer the question that asks how many courses each trainer is teaching. I selected the trainer's ID using **T_ID** and counted the number of courses with **COUNT(C_ID).** I gave the count a clear name, courses_count, so it is easy to understand. I used the **SCHEDULE** table because it shows which trainers teach which courses. The **GROUP BY T_ID** part groups the results by each trainer, so the code counts the number of courses for every trainer separately. This way, the code shows how many different courses each trainer teaches.

Admin Perspective

QUESTION 1

Add a new course (INSERT statement)

- **Write an SQL INSERT command to add a new course to the Course table with details like title, category, duration, and level.**



```
CREATE TABLE COURSE (
 course_id INT PRIMARY KEY,
 title VARCHAR (30) NOT NULL,
 category VARCHAR (40),
 duration_hours VARCHAR (40),
 course_level VARCHAR (50)
);

INSERT INTO COURSE VALUES
(1, 'Database Fundamentals', 'Databases', 20, 'Beginner'),
(2, 'Web Development Basics', 'Web', 30, 'Beginner'),
(3, 'Data Science Introduction', 'Data Science', 25, 'Intermediate'),
(4, 'Advanced SQL Queries', 'Databases', 15, 'Advanced');

INSERT INTO COURSE VALUES
(5, 'AI Fundamentals', 'AI', 10, 'Beginner');
SELECT * FROM COURSE
```

88 %

Results | Messages

	course_id	title	category	duration_hours	course_level
1	1	Database Fundamentals	Databases	20	Beginner
2	2	Web Development Basics	Web	30	Beginner
3	3	Data Science Introduction	Data Science	25	Intermediate
4	4	Advanced SQL Queries	Databases	15	Advanced
5	5	AI Fundamentals	AI	10	Beginner

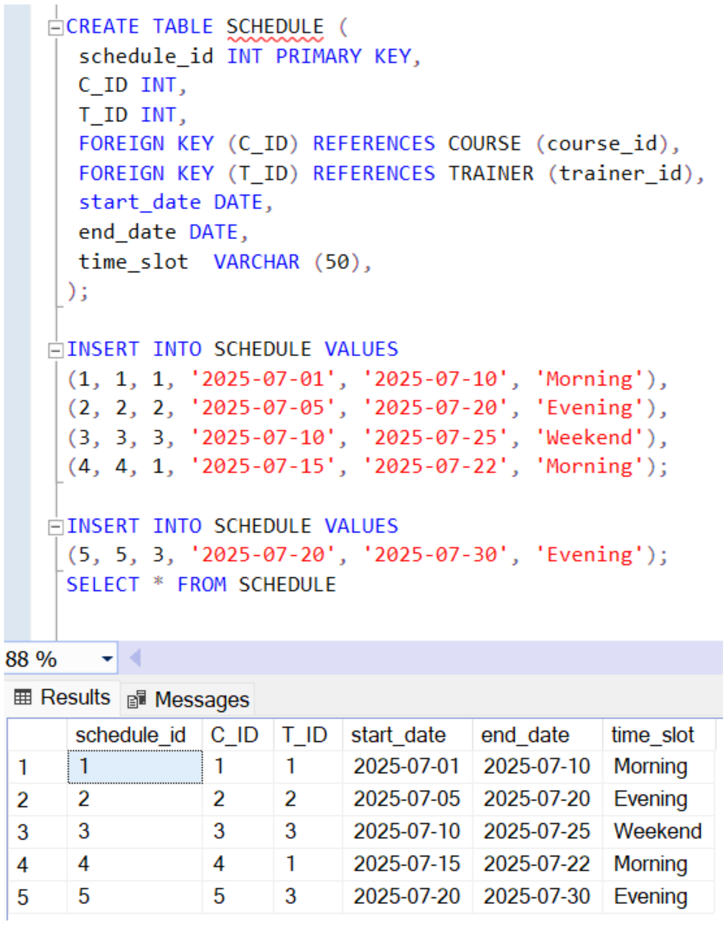
EXPLANATION:

I wrote the SQL code **INSERT INTO COURSE VALUES (5, 'AI Fundamentals', 'AI', 10, 'Beginner');** to answer the question that asks to add a new course to the Course table. This code adds a new row with all the course details: the course ID is 5, the title is "AI Fundamentals," the category is "AI," the duration is 10 (for example, hours or weeks), and the level is "Beginner." I used the **INSERT INTO** command because it adds new information to the table. This way, the new course is added exactly as the question asks.

QUESTION 2

Create a new schedule for a trainer

- **Write an INSERT statement to schedule a course by assigning a trainer, course, start/end dates, and time slot in the Schedule table.**



```
CREATE TABLE SCHEDULE (
 schedule_id INT PRIMARY KEY,
 C_ID INT,
 T_ID INT,
 FOREIGN KEY (C_ID) REFERENCES COURSE (course_id),
 FOREIGN KEY (T_ID) REFERENCES TRAINER (trainer_id),
 start_date DATE,
 end_date DATE,
 time_slot  VARCHAR (50),
);

INSERT INTO SCHEDULE VALUES
(1, 1, 1, '2025-07-01', '2025-07-10', 'Morning'),
(2, 2, 2, '2025-07-05', '2025-07-20', 'Evening'),
(3, 3, 3, '2025-07-10', '2025-07-25', 'Weekend'),
(4, 4, 1, '2025-07-15', '2025-07-22', 'Morning');

INSERT INTO SCHEDULE VALUES
(5, 5, 3, '2025-07-20', '2025-07-30', 'Evening');
SELECT * FROM SCHEDULE
```

	schedule_id	C_ID	T_ID	start_date	end_date	time_slot
1	1	1	1	2025-07-01	2025-07-10	Morning
2	2	2	2	2025-07-05	2025-07-20	Evening
3	3	3	3	2025-07-10	2025-07-25	Weekend
4	4	4	1	2025-07-15	2025-07-22	Morning
5	5	5	3	2025-07-20	2025-07-30	Evening

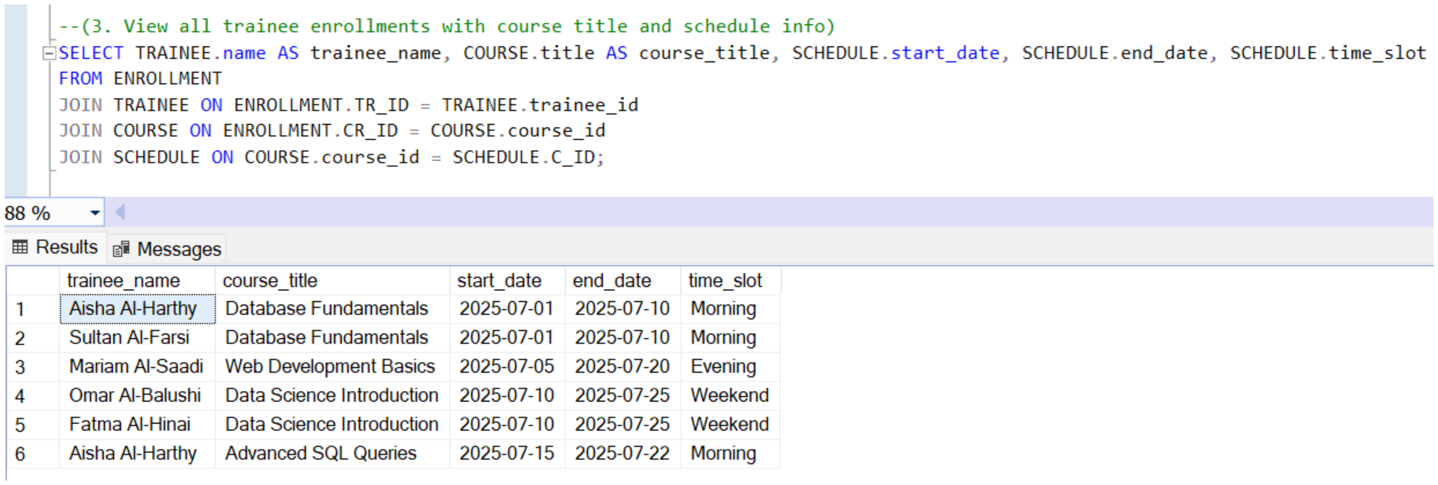
EXPLANATION:

I wrote the SQL code **INSERT INTO SCHEDULE VALUES (5, 5, 3, '2025-07-20', '2025-07-30', 'Evening');** to answer the question that asks to create a new schedule for a trainer. This code adds a new row to the Schedule table with all the details needed: the schedule ID is 5, the course ID is 5, the trainer ID is 3, the start date is July 20, 2025, the end date is July 30, 2025, and the time slot is “Evening.” I used the **INSERT INTO** command because it adds this new schedule information to the table. This way, the schedule is set up as the question asks.

QUESTION 3

View all trainee enrollments with course title and schedule info

- **Create a joined query across Enrollment, Course, and Schedule to display which trainees are enrolled in which courses, along with scheduling info.**



```
--(3. View all trainee enrollments with course title and schedule info)
SELECT TRAINEE.name AS trainee_name, COURSE.title AS course_title, SCHEDULE.start_date, SCHEDULE.end_date, SCHEDULE.time_slot
FROM ENROLLMENT
JOIN TRAINEE ON ENROLLMENT.TR_ID = TRAINEE.trainee_id
JOIN COURSE ON ENROLLMENT.CR_ID = COURSE.course_id
JOIN SCHEDULE ON COURSE.course_id = SCHEDULE.C_ID;
```

88 %

Results | Messages

	trainee_name	course_title	start_date	end_date	time_slot
1	Aisha Al-Harthy	Database Fundamentals	2025-07-01	2025-07-10	Morning
2	Sultan Al-Farsi	Database Fundamentals	2025-07-01	2025-07-10	Morning
3	Mariam Al-Saadi	Web Development Basics	2025-07-05	2025-07-20	Evening
4	Omar Al-Balushi	Data Science Introduction	2025-07-10	2025-07-25	Weekend
5	Fatma Al-Hinai	Data Science Introduction	2025-07-10	2025-07-25	Weekend
6	Aisha Al-Harthy	Advanced SQL Queries	2025-07-15	2025-07-22	Morning

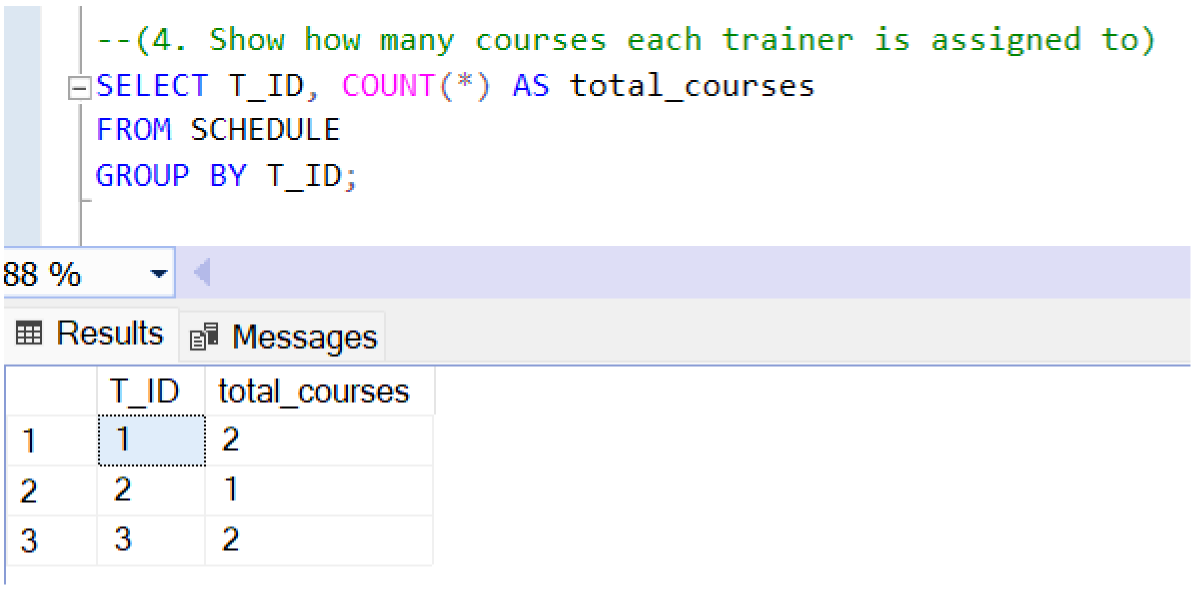
EXPLANATION:

I wrote the SQL code **SELECT TRAINEE.name AS trainee_name, COURSE.title AS course_title, SCHEDULE.start_date, SCHEDULE.end_date, SCHEDULE.time_slot FROM ENROLLMENT JOIN TRAINEE ON ENROLLMENT.TR_ID = TRAINEE.trainee_id JOIN COURSE ON ENROLLMENT.CR_ID = COURSE.course_id JOIN SCHEDULE ON COURSE.course_id = SCHEDULE.C_ID;** to answer the question that asks to show all trainee enrollments with course titles and schedule information. I selected the trainee's name, course title, start date, end date, and time slot to show all important details. I joined the **ENROLLMENT** table with the **TRAINEE** table to connect trainees with their information. Then, I joined the **COURSE** table to get the course titles for each enrollment. Finally, I joined the **SCHEDULE** table to add the schedule details for those courses. This way, the code shows a complete list of which trainees are in which courses, along with when and at what time the courses happen.

QUESTION 4

Show how many courses each trainer is assigned to

- **Write a query that counts the number of courses assigned to each trainer.**



```
--(4. Show how many courses each trainer is assigned to)
SELECT T_ID, COUNT(*) AS total_courses
FROM SCHEDULE
GROUP BY T_ID;
```

88 %

Results Messages

	T_ID	total_courses
1	1	2
2	2	1
3	3	2

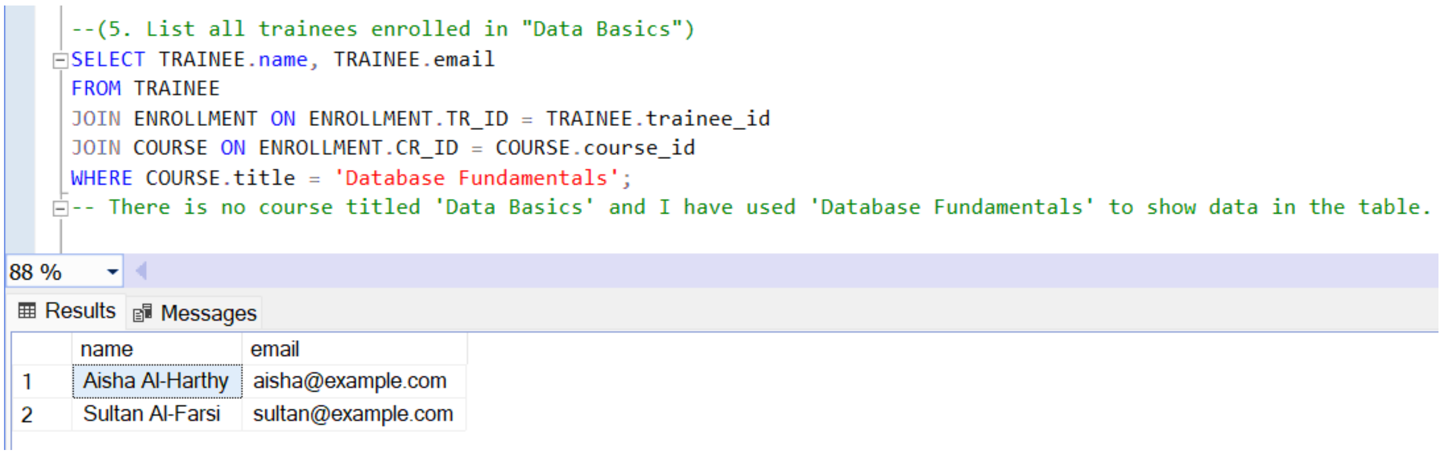
EXPLANATION:

I wrote the SQL code **SELECT T_ID, COUNT(*) AS total_courses FROM SCHEDULE GROUP BY T_ID;** to answer the question that asks how many courses each trainer is assigned to. I selected the trainer ID using **T_ID** and counted the total number of courses with **COUNT(*).** I gave this count the name total_courses so it is easy to understand. I used the **SCHEDULE** table because it shows which trainers teach which courses. The **GROUP BY T_ID** part groups the results by each trainer, so the count shows the number of courses for each trainer separately. This code helps to see how many courses each trainer is teaching.

QUESTION 5

List all trainees enrolled in "Data Basics"

- **Retrieve trainee names and emails for those enrolled in the course titled "Data Basics".**



```
--(5. List all trainees enrolled in "Data Basics")
SELECT TRAINEE.name, TRAINEE.email
FROM TRAINEE
JOIN ENROLLMENT ON ENROLLMENT.TR_ID = TRAINEE.trainee_id
JOIN COURSE ON ENROLLMENT.CR_ID = COURSE.course_id
WHERE COURSE.title = 'Database Fundamentals';
-- There is no course titled 'Data Basics' and I have used 'Database Fundamentals' to show data in the table.
```

88 %

Results | Messages

	name	email
1	Aisha Al-Harthy	aisha@example.com
2	Sultan Al-Farsi	sultan@example.com

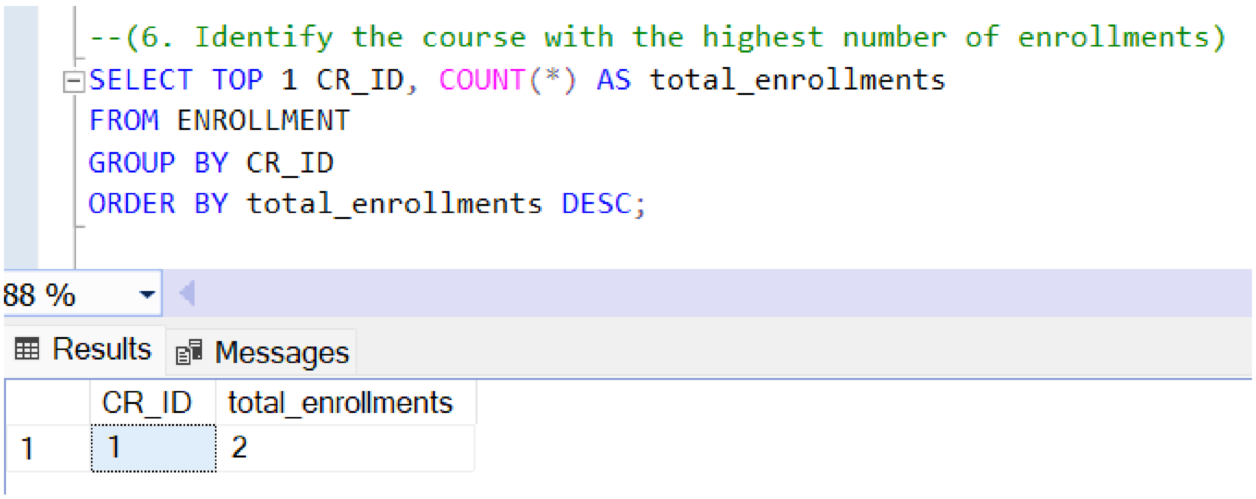
EXPLANATION:

I wrote the SQL code **SELECT TRAINEE.name, TRAINEE.email FROM TRAINEE JOIN ENROLLMENT ON ENROLLMENT.TR_ID = TRAINEE.trainee_id JOIN COURSE ON ENROLLMENT.CR_ID = COURSE.course_id WHERE COURSE.title = 'Database Fundamentals';** to answer the question that asks for the names and emails of trainees enrolled in the course called "Data Basics." Since there is no course with the exact title "Data Basics" in the table, I used "Database Fundamentals," which is a similar course title, to show how the data can be found. I joined the **TRAINEE** table with **ENROLLMENT** to connect trainees with their courses, and then joined **COURSE** to check the course title. This way, the code finds and lists the trainees enrolled in that course.

QUESTION 6

Identify the course with the highest number of enrollments

- **Write a query that ranks courses by enrollment count and displays the one with the highest number.**



```
--(6. Identify the course with the highest number of enrollments)
SELECT TOP 1 CR_ID, COUNT(*) AS total_enrollments
FROM ENROLLMENT
GROUP BY CR_ID
ORDER BY total_enrollments DESC;
```

	CR_ID	total_enrollments
1	1	2

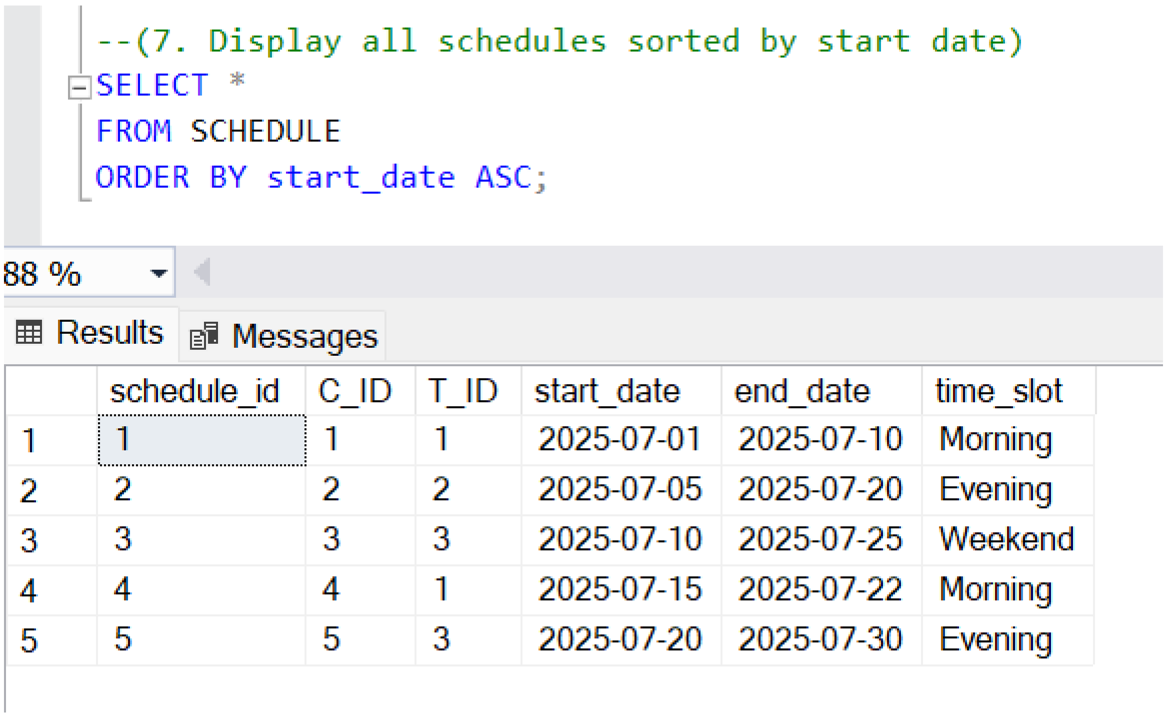
EXPLANATION:

I wrote the SQL code **SELECT TOP 1 CR_ID, COUNT(*) AS total_enrollments FROM ENROLLMENT GROUP BY CR_ID ORDER BY total_enrollments DESC;** to answer the question that asks for the course with the highest number of enrollments. I counted how many times each course ID appears in the **ENROLLMENT** table using **COUNT(*)** and gave it the name total_enrollments to make it clear. I grouped the results by course ID with **GROUP BY CR_ID** to count enrollments for each course separately. Then, I used **ORDER BY total_enrollments DESC** to sort the courses from the most enrollments to the least. Finally, I added **TOP 1** to show only the course with the highest number of enrollments. This code helps find the most popular course based on how many trainees joined it.

QUESTION 7

Display all schedules sorted by start date

➢ **Select all rows from the Schedule table and sort them in ascending order based on start date.**



```
--(7. Display all schedules sorted by start date)
SELECT *
FROM SCHEDULE
ORDER BY start_date ASC;
```

88 %

Results Messages

	schedule_id	C_ID	T_ID	start_date	end_date	time_slot
1	1	1	1	2025-07-01	2025-07-10	Morning
2	2	2	2	2025-07-05	2025-07-20	Evening
3	3	3	3	2025-07-10	2025-07-25	Weekend
4	4	4	1	2025-07-15	2025-07-22	Morning
5	5	5	3	2025-07-20	2025-07-30	Evening

EXPLANATION:

I wrote the SQL code **SELECT * FROM SCHEDULE ORDER BY start_date ASC;** to answer the question that asks to show all schedules sorted by start date. I used **SELECT *** to get all the information from the Schedule table. Then, I used **ORDER BY start_date ASC** to arrange the schedules from the earliest start date to the latest. This way, the schedules are shown in order by when they begin, making it easy to see the timeline of all sessions.