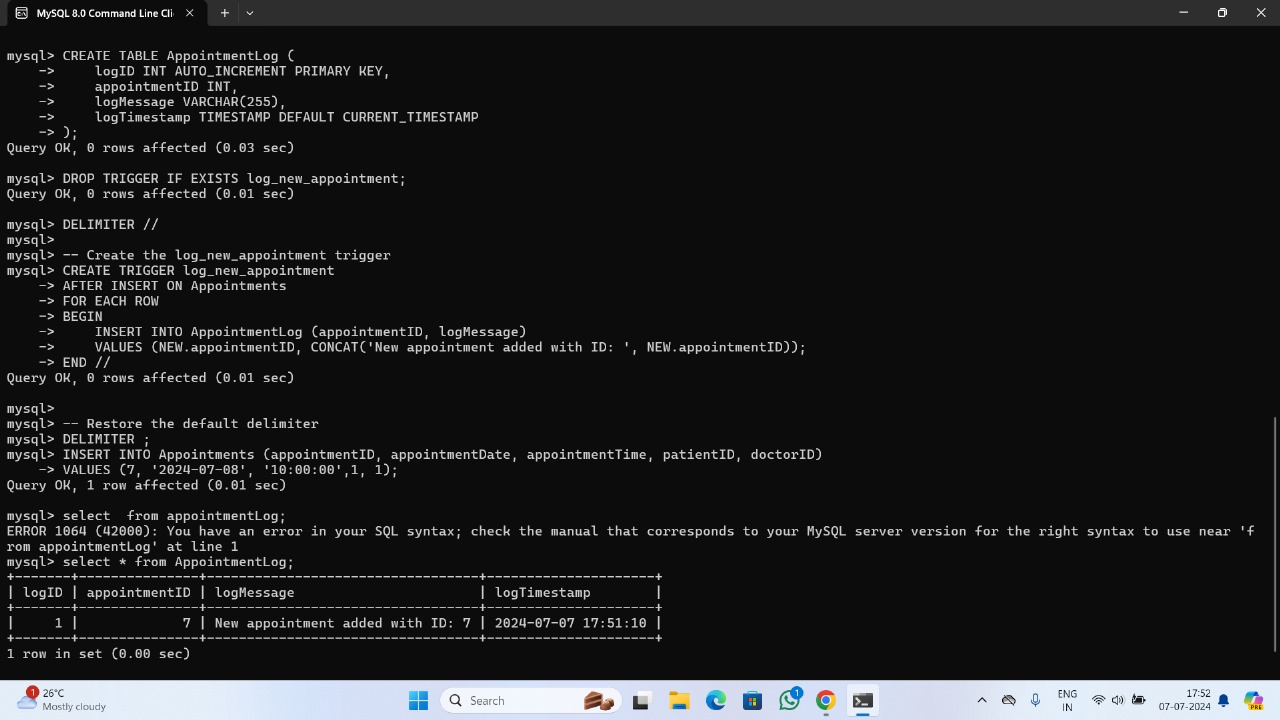
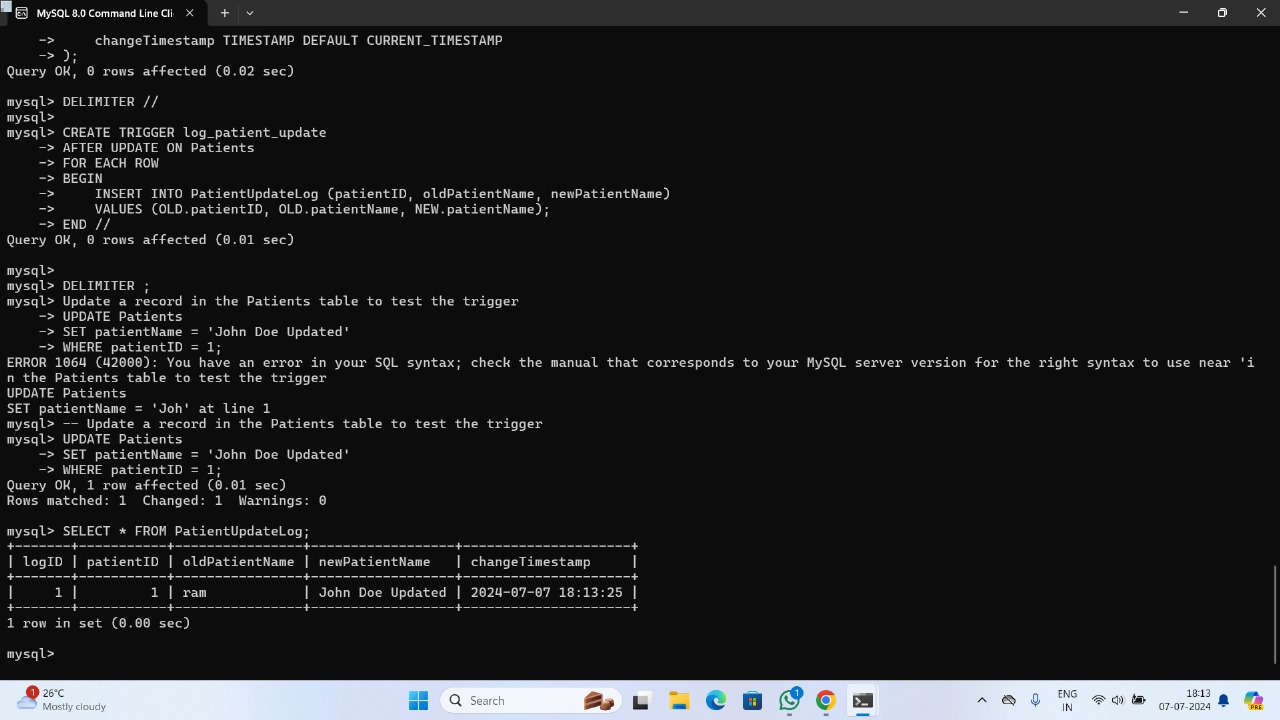
**1.Trigger to create log\_new\_appointment**



The SQL script sets up a trigger to log new appointments into an AppointmentLog table. First, it creates the AppointmentLog table with columns for a unique log ID, the appointment ID, a log message, and a timestamp of the log entry. The delimiter is then changed to // to define a multi-statement trigger. The log\_new\_appointment trigger is created to activate after any insert operation on the Appointments table. When a new appointment is added, the trigger captures the appointment ID and inserts a log entry into the AppointmentLog table with a message indicating the new appointment's ID. The delimiter is reset to the default ;. To test the trigger, a new appointment is inserted into the Appointments table, and a query to the AppointmentLog table confirms that the trigger correctly logged the new appointment by showing the appointment ID and the corresponding log message along with the timestamp.

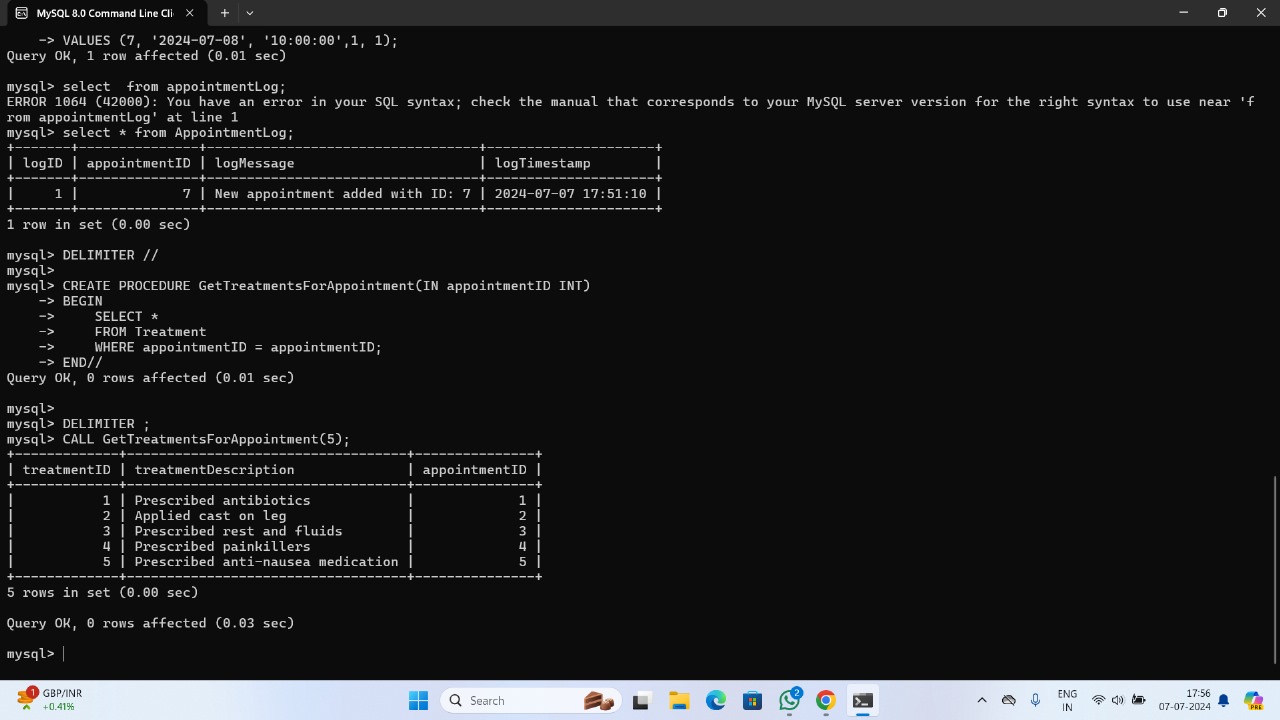
**2.Trigger to create log\_patient\_update**



The provided SQL script sets up a trigger to log updates to the Patients table into a PatientUpdateLog table. First, it creates the PatientUpdateLog table with columns for a unique log ID, patient ID, old and new patient names, and a timestamp of the change. The delimiter is then changed to // to define a multi-statement trigger. The log\_patient\_update trigger is created to activate after any update on the Patients table, capturing the old and new patient names and inserting these details into the PatientUpdateLog table. The delimiter is reset to the default ;. To test the trigger, a patient's name is updated, and a subsequent query to the PatientUpdateLog table confirms that the trigger correctly logged the change by showing the old and new patient names along with the update timestamp.

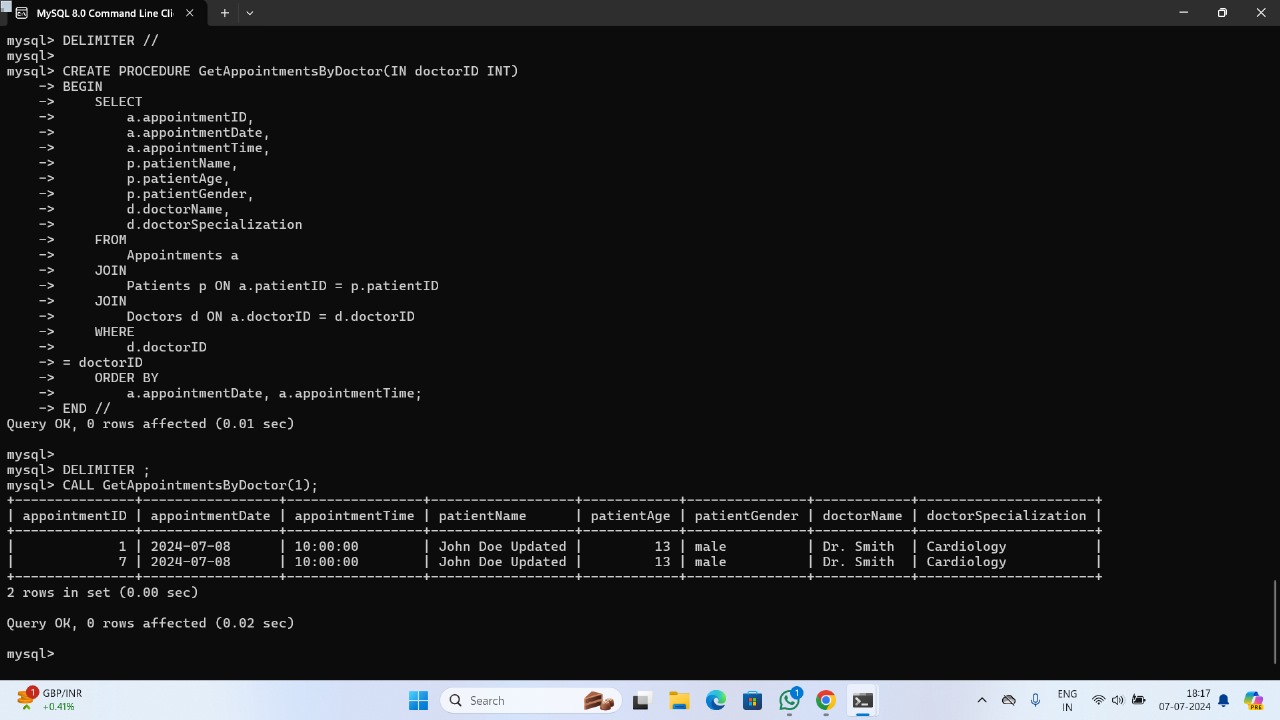
**Stored procedure**

**1.Stored procedure to retrieve records for specific appointments**



The SQL script defines a stored procedure named GetTreatmentsForAppointment that retrieves all treatment records for a specific appointment based on the provided appointment ID. First, the delimiter is changed to // to allow for the multi-statement procedure definition. The GetTreatmentsForAppointment procedure accepts an input parameter appointmentID of type INT. Inside the procedure, a SELECT statement is executed to fetch all records from the Treatment table where the appointmentID matches the input parameter. The delimiter is then reset to ;. To use this stored procedure, the CALL statement with an example appointment ID of 5 is provided, which will return all treatments associated with appointment ID 5.

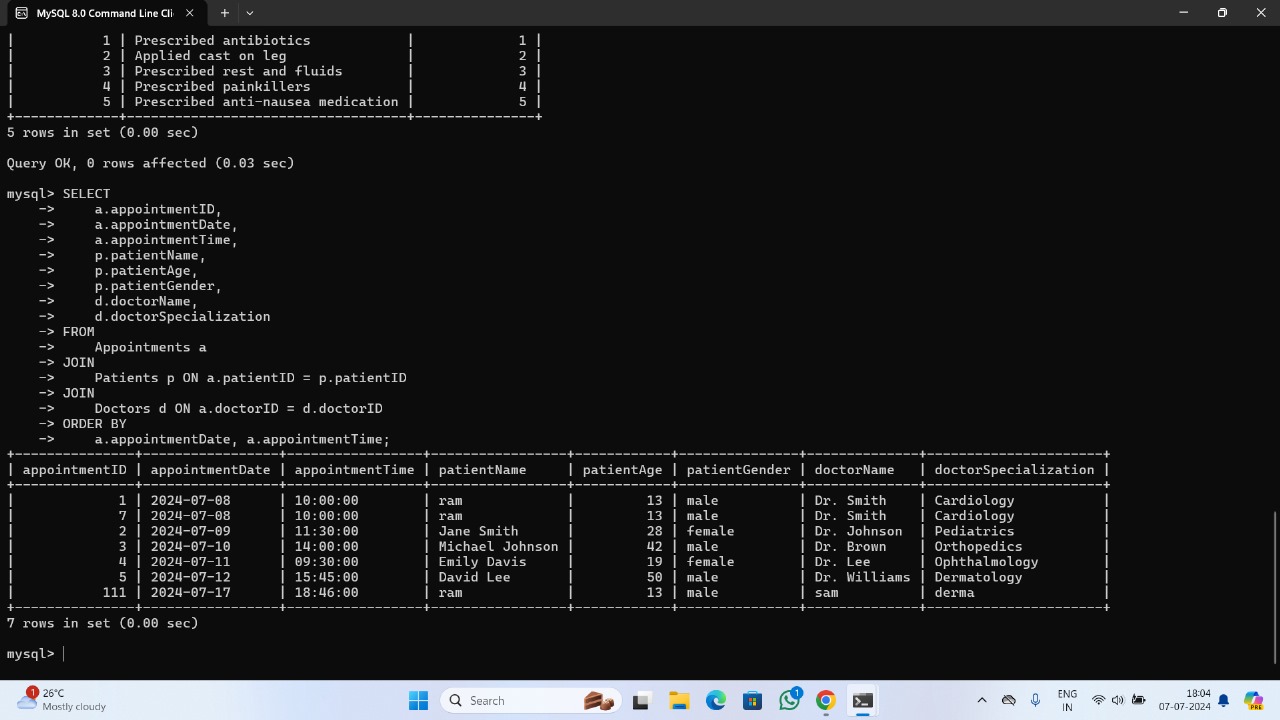
**2.Stored procedure to retrieve records for appointments for a specific doctor**



The provided SQL script defines a stored procedure called GetAppointmentsByDoctor which retrieves detailed information about all appointments for a specific doctor, identified by their doctor ID. Initially, the delimiter is changed to // to handle the multi-statement procedure definition. The procedure accepts an input parameter doctorID of type INT. Inside the procedure, a SELECT statement is executed to fetch appointment details, including appointment ID, date, time, patient name, age, gender, and doctor name and specialization. This information is gathered by joining the Appointments, Patients, and Doctors tables based on the patient and doctor IDs. The results are ordered by appointment date and time. After defining the procedure, the delimiter is reset to ;. Finally, the procedure is tested by calling it with a doctor ID of 1, which will return all appointments for the doctor with ID 1 along with the associated patient and doctor details.

**Report generation**

**1.Report showing all the appointments along with the associated patient and doctor details, ordered by date and time.**



SELECT clause: Specifies the columns to include in the report.

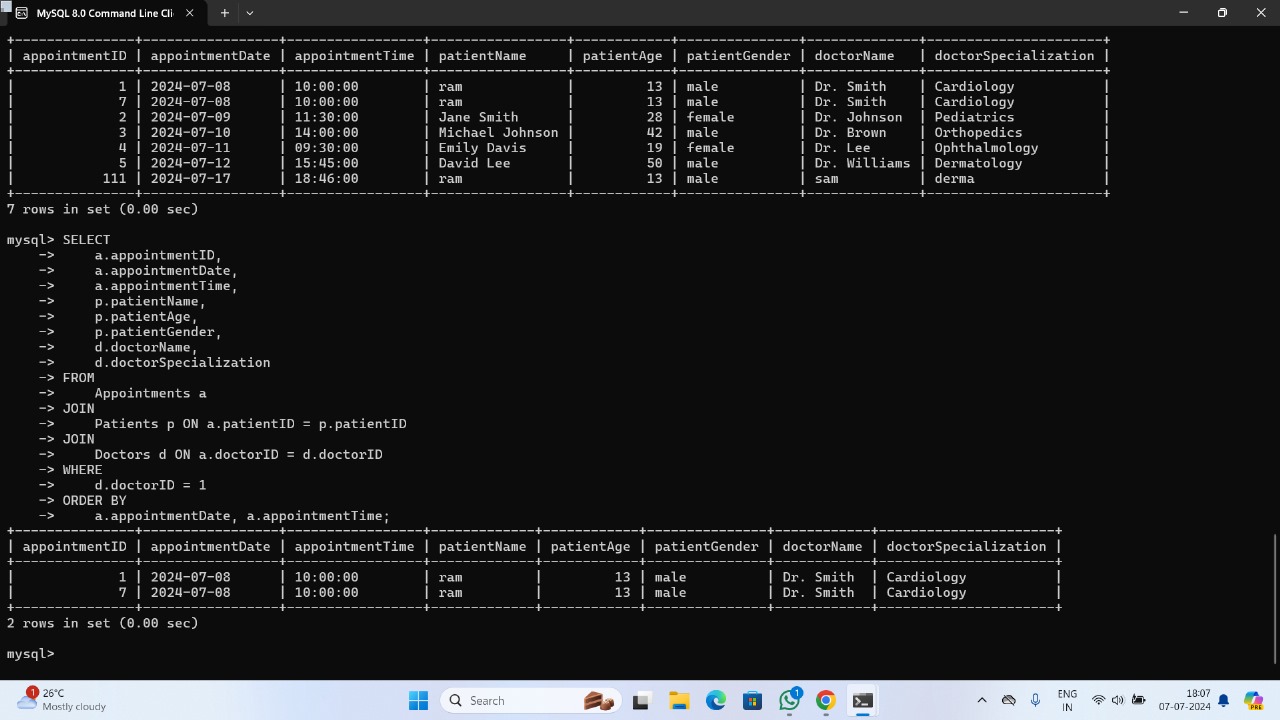
FROM clause: Specifies the main table to select data from (Appointments).

JOIN clauses: Joins the Patients and Doctors tables with the Appointments table to include patient and doctor details.

ORDER BY clause: Orders the results by appointment date and time for a clear chronological listing.

The output table is the report, showing all the appointments along with the associated patient and doctor details, ordered by date and time.

**2.Report showing all the appointments for the doctor with ID 1 , including the patient details and appointment times, ordered by date and time.**

****

SELECT clause: Specifies the columns to include in the report.

FROM clause: Specifies the main table to select data from (Appointments).

JOIN clauses: Joins the Patients and Doctors tables with the Appointments table to include patient and doctor details.

WHERE clause: Filters the results to include only appointments where the doctorID is 1.

ORDER BY clause: Orders the results by appointment date and time for a clear chronological listing.

The output table is a report showing all the appointments for the doctor with ID 1 , including the patient details and appointment times, ordered by date and time.