PART A

1- Find the names and surnames of all users registering in all sessions with a session capacity of over 100.

SessionCapacity ← π_session_id (σ_capacity>100(SESSION))
RegUsers ← π_user_id, session_id (REGISTRATION ÷ SessionCapacity)
Result ← π_first_name, last_name (USER * RegUsers)

2-Retrieve datetime of sessions attended by users with user_id = 3 or user_id = 6, excluding sessions attended by user_id = 4. Use the 'Minus (set difference)' operation for this question.
Sessions_3_6 ← π_session_id (σ_user_id = 3 ∨ user_id = 6 (REGISTRATION))
Sessions_4 ← π_session_id (σ_user_id = 4 (REGISTRATION))
Sessions_Attended_3_6 ← Sessions_3_6 - Sessions_4
Session_Times ← π_date_time (Sessions_Attended_3_6 * SESSION)

3- Retrieve the full names of all users who have registered for sessions with a registration status of 'approved' and have provided a review for at least one session.

Approved ← π_user_id (σ_approval_status = 'approved' (REGISTRATION))
UsersWithReviews ← π_user_id (REVIEW)
ApprovedAndReviews ← Approved ∩ UsersWithReviews
FullNames ← π_first_name, last_name (ApprovedAndReviews * USER)

4- Find the most popular session among all the sessions that happened between 1.1.2024 and 5.1.2024 (status of the session must be “past”)(The most popular one)

MostPopularSessions ← π_session_id, COUNT(reg_id) (σ_datetime >= '2024-01-01' ∧ datetime < '2024-05-01' ∧ session_status = 'past' (SESSION *REGISTRATION))
MaxPopularSessions ← MAX(MostPopularSessions)
PopularSessions ← σ_COUNT(reg_id) = MaxPopularSessions (MostPopularSessions)

5-Find the session names that had 5 ratings from the users. Additionally, find the session names that have status as 'past'. Retrieve the combined list of session names. Use the ‘Union’ operation for this question.

SessionsWith5Ratings ← π_title (σ_rating = 5 (SESSION * REVIEW))
SessionsPast ← π_title (σ_session_status = 'past' (SESSION))
CombinedSessions ← SessionsWith5Ratings ∪ SessionsPast

PART B

1-Identify the top 5 most registered sessions based on the number of registered users. List the session title, date, location, and the total number of registered users for each session.

session_title	session_date	location	total_registered_users
UI/UX Design Principles	2025-01-05	Conference Room	9
Digital Marketing Strategies	2024-05-20	Conference Room	7
Machine Learning Fundamentals	2024-05-05	Auditorium	6
Web Development Bootcamp	2024-06-10	Room C301	6
Public Speaking Mastery	2024-08-01	Room A205	6

```
SELECT s.title AS session_title, s.datetime AS session_date, s.location, COUNT(r.user_id)
AS total_registered_users
FROM session s
JOIN registration r ON s.session_id = r.session_id
GROUP BY s.session_id
ORDER BY total_registered_users DESC
LIMIT 5;
```

2-

a) Create a view named 'ActiveUsersWithMultipleRegistrations' to find the users who have registered more than 3 times in the next 6 months. Retrieve their user IDs, phone numbers, email addresses, and account types. (You may use CURDATE(), DATE_ADD() operations)

user_id	phone_number	email_address	account_type
12	0123456789	alexander@example.com	private
13	5451234567	emma_clark@example.com	public
16	5454567890	david@example.com	private
45	6663456789	luna@example.com	public

```
CREATE VIEW ActiveUsersWithMultipleRegistrations AS
SELECT u.user_id, u.phone_number, u.email_address, u.account_type
FROM `user` u
JOIN registration r ON u.user_id = r.user_id
JOIN session s ON r.session_id = s.session_id
WHERE r.reg_date BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 6
MONTH)
GROUP BY u.user_id
HAVING COUNT(DISTINCT r.session_id) > 3;
SELECT * FROM met.activeuserswithmultipleregistrations;
```

b) Create a trigger named "PreventSessionOverCapacity" to prevent users from registering for sessions that have reached their maximum capacity. Raise an error if a user attempts to

register for a session that is already full. (To make your trigger work, do not forget to use a delimiter at the start and at the end of statement)

You may find an example below for inserting an exception and printing the error in case of preventing voters from voting for themselves.

Example:

```
CREATE TRIGGER prevent_self_votes BEFORE INSERT ON votes
  FOR EACH ROW
    BEGIN
      IF(new.user_id = (SELECT author_id FROM posts WHERE id=new.post_id)) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "you cannot vote for yourself";
      END IF;
    END;
```

```
DELIMITER //

CREATE TRIGGER PreventSessionOverCapacity
BEFORE INSERT ON registration
FOR EACH ROW
BEGIN
    DECLARE session_capacity INT;
    DECLARE current_attendants INT;

    -- Get the session's maximum capacity
    SELECT capacity INTO session_capacity
    FROM session
    WHERE session_id = NEW.session_id;

    -- Get the current number of attendants registered for the session
    SELECT COUNT(*) INTO current_attendants
    FROM registration
    WHERE session_id = NEW.session_id;

    -- Check if the session has reached its maximum capacity
    IF current_attendants >= session_capacity THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'This session has reached its maximum capacity. Registration
is not allowed.';
    END IF;
END//
```

```
DELIMITER ;
```

3- List the session titles that require immediate attention due to low registration numbers. Retrieve the session titles and the total count of registrations lower than three for each session that has a registration status set to 'approved'.

session_title	total_registrations
Data Visualization Techniques	1
Machine Learning Fundamentals	2
Frontend Development Seminar	1
Introduction to Data Science	2
Leadership Workshop	2
E-commerce Strategies	1
Financial Planning Seminar	2
Artificial Intelligence Ethics	1
Marketing Analytics Seminar	1

```
SELECT s.title AS session_title, COUNT(r.reg_id) AS total_registrations
FROM session s
LEFT JOIN registration r ON s.session_id = r.session_id
WHERE r.approval_status = 'approved'
GROUP BY s.session_id
HAVING total_registrations < 3;
```

4- Calculate the average rating for sessions conducted by each speaker. Retrieve the top 5 speaker's name and the average ratings in descending order.

speaker_id	speaker_first_name	speaker_last_name	average_rating
17	Daniel	Young	5.0000
14	Linda	Walker	5.0000
9	William	Thomas	5.0000
18	Patricia	King	4.5000
20	Karen	Scott	4.3333

```
SELECT
    sp.speaker_id,
    sp.first_name AS speaker_first_name,
    sp.last_name AS speaker_last_name,
    AVG(rv.rating) AS average_rating
FROM
    speaker sp
JOIN
    speaks_at sa ON sp.speaker_id = sa.s_id
JOIN
    session se ON sa.session_id = se.session_id
```

```
JOIN
    review rv ON se.session_id = rv.session_id
GROUP BY
    sp.speaker_id, sp.first_name, sp.last_name
ORDER BY
    average_rating desc
LIMIT 5;
```

5- Identify the users who have interests in “programming” or “computer” and match the requirements specified in the registration table. Retrieve these users' user IDs, first names, and last names.

user_id	first_name	last_name
1	John	Smith
4	Jennifer	Davis
15	Charlotte	Lewis

```
SELECT u.user_id, u.first_name, u.last_name
FROM `user` u
JOIN interests i ON u.user_id = i.user_id
JOIN registration r ON u.user_id = r.user_id
WHERE i.interest IN ('programming', 'computer')
  AND r.requirements IN ('students with programming skills', 'technical departments only')
GROUP BY u.user_id, u.first_name, u.last_name;
```