ANSI SQL Using MySQL Exercises

1.User Upcoming Events
Show a list of all upcoming events a user is registered for in their city, sorted by date.

```
mysql> SELECT e.event_id, e.title, e.city, e.start_date, e.end_date, e.status
   -> FROM Users u
   -> JOIN Registrations r ON u.user_id = r.user_id
   -> JOIN Events1 e ON r.event_id = e.event_id
   -> WHERE e.status = 'upcoming'
   -> AND e.city = u.city
   -> AND u.user_id = 1
   -> ORDER BY e.start_date;
```

2. **Top Rated Events**
Identify events with the highest average rating, considering only those that have received at least 10 feedback submissions.

```
mysql> SELECT
   -> e.event_id,
   -> e.title,
   -> e.city,
   -> AVG(f.rating) AS average_rating,
   -> COUNT(f.feedback_id) AS feedback_count
   -> FROM Events1 e
   -> JOIN Feedback f ON e.event_id = f.event_id
   -> GROUP BY e.event_id, e.title, e.city
   -> HAVING COUNT(f.feedback_id) >= 10
   -> ORDER BY average_rating DESC;
```

3. **Inactive Users**
Retrieve users who have not registered for any events in the last 90 days.

```
mysql> SELECT u.user_id, u.full_name, u.email, u.city, u.registration_date
   -> FROM Users u
   -> LEFT JOIN Registrations r
   -> ON u.user_id = r.user_id
   -> AND r.registration_date >= CURDATE() - INTERVAL 90 DAY
   -> WHERE r.registration_id IS NULL;
```

4. **Peak Session Hours**
Count how many sessions are scheduled between 10 AM to 12 PM for each event.

```
mysql> SELECT event_id,COUNT(*) AS sessions_between_10_and_12
   -> FROM Sessions
   -> WHERE TIME(start_time) >= '10:00:00'
   -> AND TIME(end_time) <= '12:00:00'
   -> GROUP BY event_id;
```

5. **Most Active Cities**
List the top 5 cities with the highest number of distinct user registrations.

```
mysql> SELECT u.city,COUNT(DISTINCT r.user_id) AS distinct_user_count
    -> FROM Users u
    -> JOIN Registrations r ON u.user_id = r.user_id
    -> GROUP BY u.city
    -> ORDER BY distinct_user_count DESC
    -> LIMIT 5;
```

6. **Event Resource Summary**
Generate a report showing the number of resources (PDFs, images, links) uploaded for each event.

```
mysql> SELECT e.event_id,e.title,
    -> SUM(CASE WHEN r.resource_type = 'pdf' THEN 1 ELSE 0 END) AS pdf_count,
    -> SUM(CASE WHEN r.resource_type = 'image' THEN 1 ELSE 0 END) AS image_count,
    -> SUM(CASE WHEN r.resource_type = 'link' THEN 1 ELSE 0 END) AS link_count
    -> FROM Events1 e
    -> LEFT JOIN Resources r ON e.event_id = r.event_id
    -> GROUP BY e.event_id, e.title
    -> ORDER BY e.event_id;
```

7. **Low Feedback Alerts**
List all users who gave feedback with a rating less than 3, along with their comments and associated event names.

```
mysql> SELECT u.user_id,u.full_name,f.rating,f.comments,e.title AS event_name
    -> FROM Feedback f
    -> JOIN Users u ON f.user_id = u.user_id
    -> JOIN Events1 e ON f.event_id = e.event_id
    -> WHERE f.rating < 3;
```

8. **Sessions per Upcoming Event**
Display all upcoming events with the count of sessions scheduled for them.

```
mysql> SELECT e.event_id,e.title,e.status,COUNT(s.session_id) AS session_count
    -> FROM Events1 e
    -> LEFT JOIN Sessions s ON e.event_id = s.event_id
    -> WHERE e.status = 'upcoming'
    -> GROUP BY e.event_id, e.title, e.status
    -> ORDER BY e.event_id;
```

9. **Organizer Event Summary**
For each event organizer, show the number of events created and their current status (upcoming, completed, cancelled).

```
mysql> SELECT u.user_id,u.full_name,e.status,COUNT(e.event_id) AS event_count
    -> FROM Users u
    -> JOIN Events1 e ON u.user_id = e.organizer_id
    -> GROUP BY u.user_id, u.full_name, e.status
    -> ORDER BY u.user_id, e.status;
```

10. **Feedback Gap**
Identify events that had registrations but received no feedback at all.

```
mysql> SELECT e.event_id, e.title, e.city
    -> FROM Events1 e
    -> JOIN Registrations r ON e.event_id = r.event_id
    -> LEFT JOIN Feedback f ON e.event_id = f.event_id
    -> GROUP BY e.event_id, e.title, e.city
    -> HAVING COUNT(f.feedback_id) = 0;
```

11. **Daily New User Count**
Find the number of users who registered each day in the last 7 days.

```
mysql> SELECT registration_date,COUNT(user_id) AS new_user_count
    -> FROM Users
    -> WHERE registration_date >= CURDATE() - INTERVAL 7 DAY
    -> GROUP BY registration_date
    -> ORDER BY registration_date;
```

12. **Event with Maximum Sessions**
List the event(s) with the highest number of sessions.

```
mysql> SELECT e.event_id,e.title,COUNT(s.session_id) AS session_count
    -> FROM Events1 e
    -> JOIN Sessions s ON e.event_id = s.event_id
    -> GROUP BY e.event_id, e.title
    -> HAVING session_count = (
    -> SELECT MAX(session_counts) FROM (
    -> SELECT COUNT(session_id) AS session_counts
    -> FROM Sessions
    -> GROUP BY event_id
    -> ) AS counts
    -> );
```

13. **Average Rating per City**
Calculate the average feedback rating of events conducted in each city.

```
mysql> SELECT e.city,AVG(f.rating) AS average_rating
    -> FROM Events1 e
    -> JOIN Feedback f ON e.event_id = f.event_id
    -> GROUP BY e.city
    -> ORDER BY average_rating DESC;
```

14. **Most Registered Events**
List top 3 events based on the total number of user registrations.

```
mysql> SELECT e.event_id,e.title,COUNT(r.registration_id) AS total_registrations
    -> FROM Events1 e
    -> JOIN Registrations r ON e.event_id = r.event_id
    -> GROUP BY e.event_id, e.title
    -> ORDER BY total_registrations DESC
    -> LIMIT 3;
```

15. **Event Session Time Conflict**
Identify overlapping sessions within the same event (i.e., session start and end times that conflict).

```
mysql> SELECT s1.event_id,
    -> s1.session_id AS session1_id,
    -> s1.title AS ss1_title,
    -> s1.start_time AS ss1_start,
    -> s1.end_time AS ss1_end,
    -> s2.session_id AS ss2_id,
    -> s2.title AS ss2_title,
    -> s2.start_time AS ss2_start,
    -> s2.end_time AS ss2_end
    -> FROM Sessions s1
    -> JOIN Sessions s2
    -> ON s1.event_id = s2.event_id
    -> AND s1.session_id < s2.session_id
    -> WHERE
    -> s1.start_time < s2.end_time
    -> AND s2.start_time < s1.end_time;
```

16. **Unregistered Active Users**
Find users who created an account in the last 30 days but haven't registered for any events.

```
mysql> SELECT u.user_id,u.full_name,u.email,u.registration_date
    -> FROM Users u
    -> LEFT JOIN Registrations r ON u.user_id = r.user_id
    -> WHERE u.registration_date >= CURDATE() - INTERVAL 30 DAY
    -> AND r.registration_id IS NULL;
```

17. **Multi-Session Speakers**
Identify speakers who are handling more than one session across all events.

```
mysql> SELECT speaker_name,COUNT(session_id) AS session_count
    -> FROM Sessions
    -> GROUP BY speaker_name
    -> HAVING session_count > 1;
```

18. **Resource Availability Check**
List all events that do not have any resources uploaded.

```
mysql> SELECT e.event_id,e.title,e.city,e.start_date,e.end_date
    -> FROM Events1 e
    -> LEFT JOIN Resources r ON e.event_id = r.event_id
    -> WHERE r.resource_id IS NULL;
```

19. **Completed Events with Feedback Summary**
For completed events, show total registrations and average feedback rating.

```
mysql> SELECT e.event_id,e.title,
    -> COUNT(DISTINCT r.registration_id) AS total_registration,
    -> ROUND(AVG(f.rating), 2) AS average_rating
    -> FROM Events1 e
    -> LEFT JOIN Registrations r ON e.event_id = r.event_id
    -> LEFT JOIN Feedback f ON e.event_id = f.event_id
    -> WHERE e.status = 'completed'
    -> GROUP BY e.event_id, e.title;
```

20. **User Engagement Index**
For each user, calculate how many events they attended and how many feedbacks they submitted.

```
mysql> SELECT u.user_id,u.full_name,
    -> COUNT(DISTINCT r.event_id) AS events_attended,
    -> COUNT(DISTINCT f.feedback_id) AS feedbacks_submitted
    -> FROM Users u
    -> LEFT JOIN Registrations r ON u.user_id = r.user_id
    -> LEFT JOIN Feedback f ON u.user_id = f.user_id
    -> GROUP BY u.user_id, u.full_name;
```

21. **Top Feedback Providers**
List top 5 users who have submitted the most feedback entries.

```
mysql> SELECT u.user_id,u.full_name,u.email,
    -> COUNT(f.feedback_id) AS feedback_count
    -> FROM Users u
    -> JOIN Feedback f ON u.user_id = f.user_id
    -> GROUP BY u.user_id, u.full_name, u.email
    -> ORDER BY feedback_count DESC
    -> LIMIT 5;
```

22. **Duplicate Registrations Check**
Detect if a user has been registered more than once for the same event.

```
mysql> SELECT user_id,event_id,COUNT(*) AS registration_count
    -> FROM Registrations
    -> GROUP BY user_id, event_id
    -> HAVING COUNT(*) > 1;
```

23. **Registration Trends**
Show a month-wise registration count trend over the past 12 months.

```
mysql> SELECT DATE_FORMAT(registration_date, '%Y-%m') AS month,
    -> COUNT(*) AS registration_count
    -> FROM Registrations
    -> WHERE registration_date >= DATE_SUB(CURDATE(), INTERVAL 12 MONTH)
    -> GROUP BY month
    -> ORDER BY month;
```

24. **Average Session Duration per Event**
Compute the average duration (in minutes) of sessions in each event.

```
mysql> SELECT e.event_id,e.title,
    -> ROUND(AVG(TIMESTAMPDIFF(MINUTE, s.start_time, s.end_time)), 2) AS
avg_duration_min
    -> FROM Events1 e
    -> JOIN Sessions s ON e.event_id = s.event_id
    -> GROUP BY e.event_id, e.title;
```

25. **Events Without Sessions**
List all events that currently have no sessions scheduled under them.

```
mysql> SELECT e.event_id,e.title,e.city,e.start_date,e.end_date
    -> FROM Events1 e
    -> LEFT JOIN Sessions s ON e.event_id = s.event_id
    -> WHERE s.session_id IS NULL;
```