Department of Computer Science

CSL 220: DBMS Lab

Class: BSAI-4A

LAB Journal 13

Lab 13: Open Ended Lab

Name: Ayesha Javed - 009

Zulqarnain Munir- 050

Time: 8:30 – 11:30
NC LAB - III

Instructor: Ms. Samia Kiran

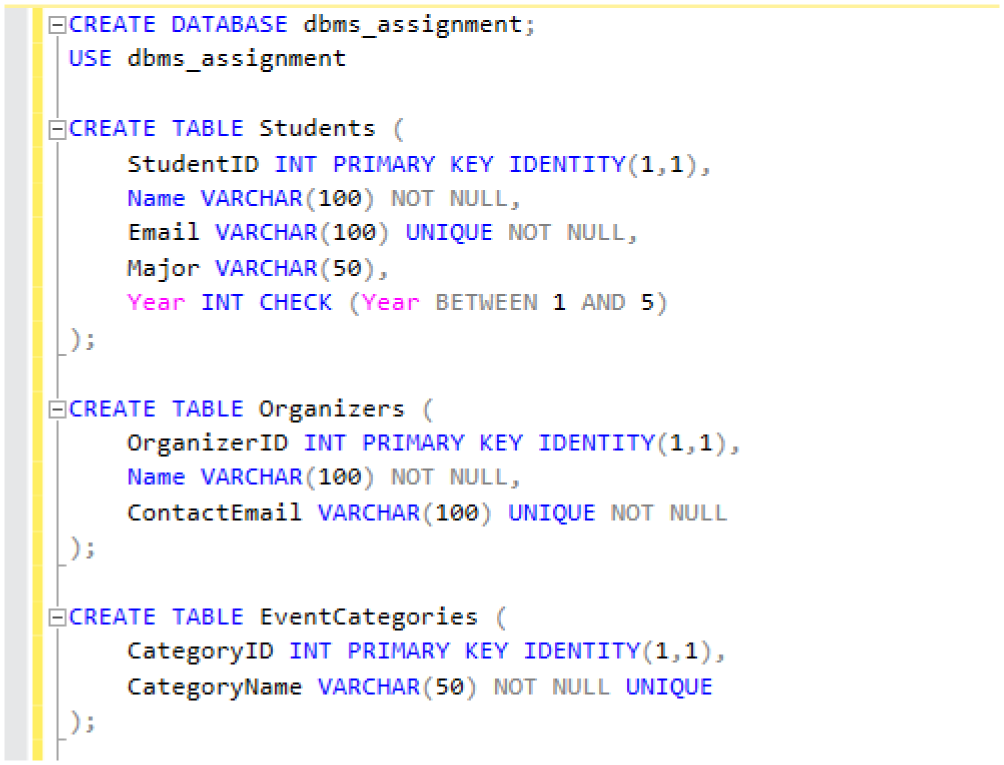
BuzzBoard – Student Event Tracker

Scenario:

University students often struggle to stay updated on the exciting events happening around campus, be it tech workshops, music nights, sports tournaments, or club meetups. BuzzBoard is a conceptual backend platform designed to digitally manage and explore these events in a structured manner. Your task is to design and implement a relational database system that forms the backbone of this event tracker. This platform should allow students to discover, register for, and track events while enabling organizers to analyze participation trends.

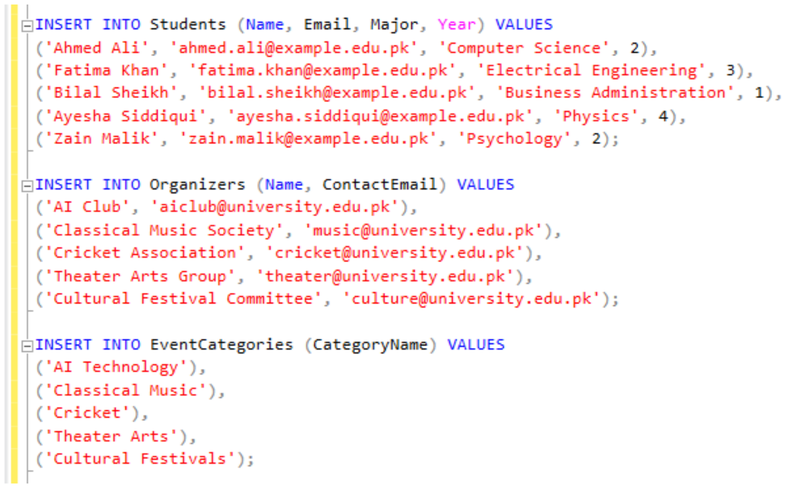
Design and implement the backend database structure for the BuzzBoard Event Tracker.

TABLES:

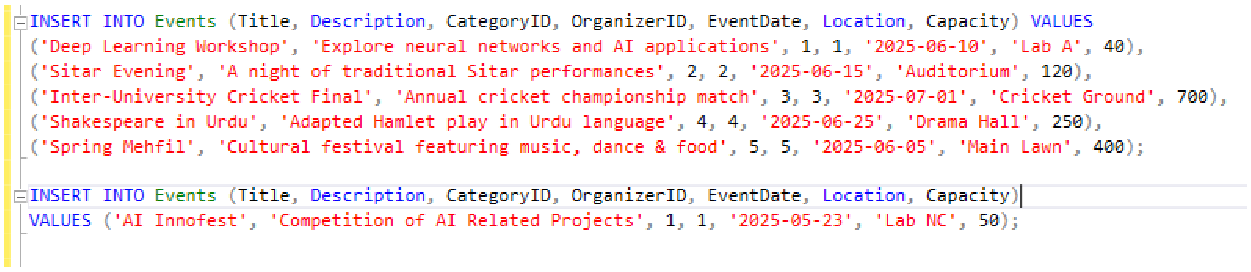


```sql
CREATE DATABASE dbms_assignment;
USE dbms_assignment

CREATE TABLE Students (
    StudentID INT PRIMARY KEY IDENTITY(1,1),
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Major VARCHAR(50),
    Year INT CHECK (Year BETWEEN 1 AND 5)
);

CREATE TABLE Organizers (
    OrganizerID INT PRIMARY KEY IDENTITY(1,1),
    Name VARCHAR(100) NOT NULL,
    ContactEmail VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE EventCategories (
    CategoryID INT PRIMARY KEY IDENTITY(1,1),
    CategoryName VARCHAR(50) NOT NULL UNIQUE
);
```

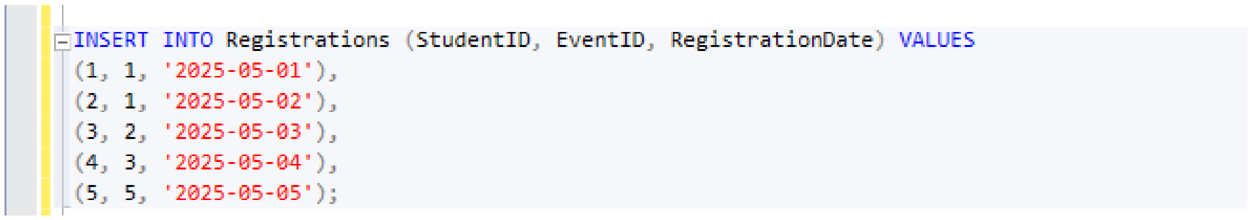
INSERTING DATA:



```sql
INSERT INTO Students (Name, Email, Major, Year) VALUES
('Ahmed Ali', 'ahmed.ali@example.edu.pk', 'Computer Science', 2),
('Fatima Khan', 'fatima.khan@example.edu.pk', 'Electrical Engineering', 3),
('Bilal Sheikh', 'bilal.sheikh@example.edu.pk', 'Business Administration', 1),
('Ayesha Siddiqui', 'ayesha.siddiqui@example.edu.pk', 'Physics', 4),
('Zain Malik', 'zain.malik@example.edu.pk', 'Psychology', 2);

INSERT INTO Organizers (Name, ContactEmail) VALUES
('AI Club', 'aiclub@university.edu.pk'),
('Classical Music Society', 'music@university.edu.pk'),
('Cricket Association', 'cricket@university.edu.pk'),
('Theater Arts Group', 'theater@university.edu.pk'),
('Cultural Festival Committee', 'culture@university.edu.pk');

INSERT INTO EventCategories (CategoryName) VALUES
('AI Technology'),
('Classical Music'),
('Cricket'),
('Theater Arts'),
('Cultural Festivals');
```



```sql
INSERT INTO Events (Title, Description, CategoryID, OrganizerID, EventDate, Location, Capacity) VALUES
('Deep Learning Workshop', 'Explore neural networks and AI applications', 1, 1, '2025-06-10', 'Lab A', 40),
('Sitar Evening', 'A night of traditional Sitar performances', 2, 2, '2025-06-15', 'Auditorium', 120),
('Inter-University Cricket Final', 'Annual cricket championship match', 3, 3, '2025-07-01', 'Cricket Ground', 700),
('Shakespeare in Urdu', 'Adapted Hamlet play in Urdu language', 4, 4, '2025-06-25', 'Drama Hall', 250),
('Spring Mehfil', 'Cultural festival featuring music, dance & food', 5, 5, '2025-06-05', 'Main Lawn', 400);

INSERT INTO Events (Title, Description, CategoryID, OrganizerID, EventDate, Location, Capacity)
VALUES ('AI Innofest', 'Competition of AI Related Projects', 1, 1, '2025-05-23', 'Lab NC', 50);
```



```sql
INSERT INTO Registrations (StudentID, EventID, RegistrationDate) VALUES
(1, 1, '2025-05-01'),
(2, 1, '2025-05-02'),
(3, 2, '2025-05-03'),
(4, 3, '2025-05-04'),
(5, 5, '2025-05-05');
```

<u>Explanation of Tables:</u>

Students

- **Purpose**: Stores information about students who can register for events.
- **Primary Key**: *Student_id* – unique ID for each student (auto-incremented).
- **Important Columns**:
    - **name** – full name of the student.
    - **email** – must be **unique**, ensures no duplicate student accounts.
    - **major** – student's field of study (e.g., Computer Science).
    - **year_of_study** – integer value indicating the academic year (e.g., 1, 2, 3, 4).
- **Constraints**: Email is set as **UNIQUE**; other fields are **NOT NULL**.

Organizers

- **Purpose**: Stores societies and clubs that organize events.
- **Primary Key**: **organizer_id** – uniquely identifies each club.
- **Important Column**:
  - **name** – must be **unique**, so no two organizers have the same name.
- **Constraint**: Organizer name is unique to avoid duplicates like "AI Club" appearing twice.

EventCategories

- **Purpose**: Defines types of events, such as "AI Technology", "Music Jam", etc.
- **Primary Key**: **category_id.**
- **Important Column**:
  - **category_name** – must be **unique** and **not null**.
- **Constraint**: Uniqueness ensures each category is clearly distinguishable.

Events

- **Purpose**: Stores all the events posted by organizers.
- **Primary Key**: **event_id.**
- **Important Columns**:
  - **title, description** – event details.
  - **event_date** – must be provided (not null).
  - **capacity** – maximum number of participants allowed (not null).
  - **organizer_id** – **foreign key** referencing organizers.organizer_id.
  - **category_id** – **foreign key** referencing event_categories.category_id.
- **Relationships**:
  - Many-to-One with **organizers.**
  - Many-to-One with **event_categories.**

Registrations

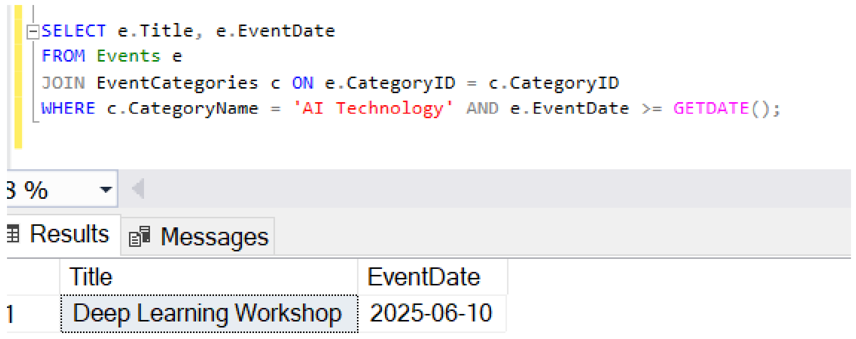
- **Purpose**: Tracks which students are registered for which events.
- **Primary Key**: **registration_id.**
- **Important Columns**:

- **student_id** – **foreign key** referencing students.student_id.
- **event_id** – **foreign key** referencing events.event_id.
- **registration_date** – the date the student registered (not null).

- **Relationship**:
  - Many-to-One with both students and events.

Queries:

1. Upcoming AI Technology Event

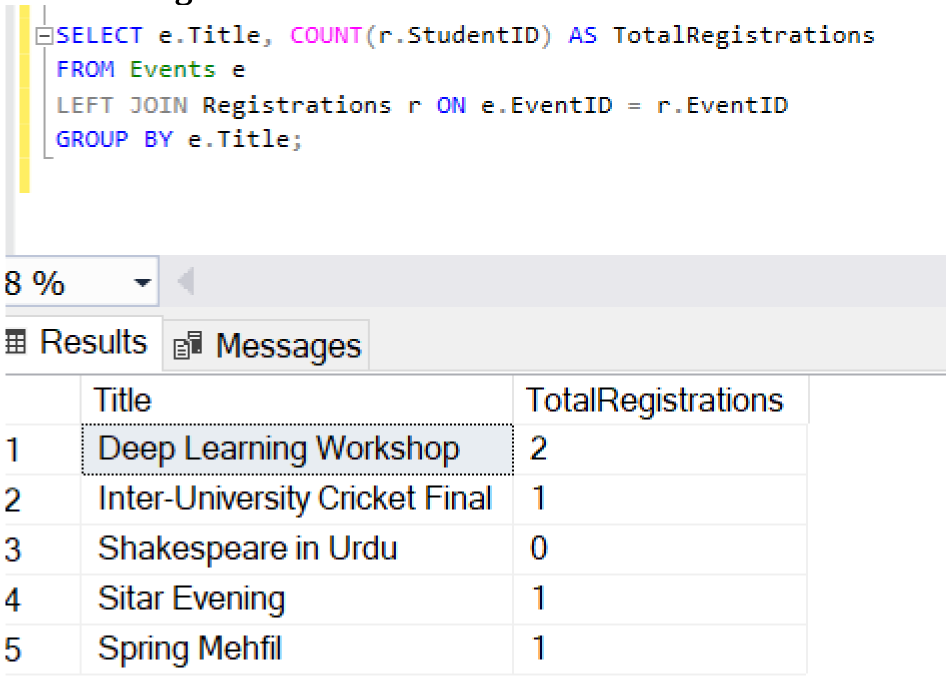


```
SELECT e.Title, e.EventDate
FROM Events e
JOIN EventCategories c ON e.CategoryID = c.CategoryID
WHERE c.CategoryName = 'AI Technology' AND e.EventDate >= GETDATE();
```

```
  Title                   EventDate
1 Deep Learning Workshop  2025-06-10
```

- **JOIN Applied**: Between **Events.CategoryID** and **EventCategories.CategoryID.**
- **WHERE Clause**: Filters only events that are in the future (**e.EventDate >= GETDATE()**) and in the '**AI Technology**' category.
- **Purpose**: Returns a list of future events specifically related to AI, helping students stay updated with tech-related opportunities.

2. Total Registrations Per Event



```
SELECT e.Title, COUNT(r.StudentID) AS TotalRegistrations
FROM Events e
LEFT JOIN Registrations r ON e.EventID = r.EventID
GROUP BY e.Title;
```

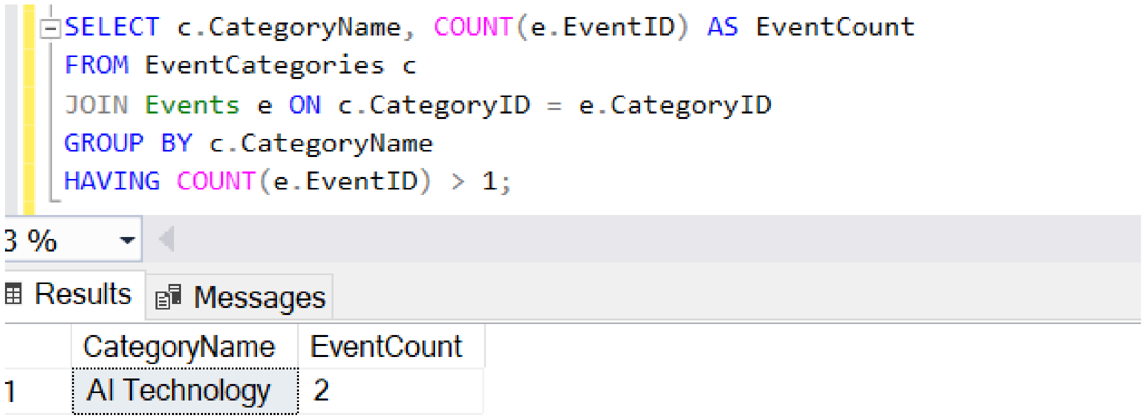
8 %

Results Messages

	Title	TotalRegistrations
1	Deep Learning Workshop	2
2	Inter-University Cricket Final	1
3	Shakespeare in Urdu	0
4	Sitar Evening	1
5	Spring Mehfil	1

- **JOIN applied:** LEFT JOIN between Events.EventID and Registrations.EventID to count registrations, including events with zero registrations.
- **COUNT aggregation applied on:** RegistrationID from Registrations to count how many students registered per event.
- **GROUP BY applied on:** Events.EventID and Events.Title to aggregate registrations event-wise.
- **Purpose:** Provides total number of students registered for each event, useful for organizers to monitor participation.

3. Categories With More Than One Event



```
SELECT c.CategoryName, COUNT(e.EventID) AS EventCount
FROM EventCategories c
JOIN Events e ON c.CategoryID = e.CategoryID
GROUP BY c.CategoryName
HAVING COUNT(e.EventID) > 1;
```

3 %

Results Messages

	CategoryName	EventCount
1	AI Technology	2

- **JOIN:** Between **EventCategories.CategoryID** and **Events.CategoryID**. It connects each event to its category.

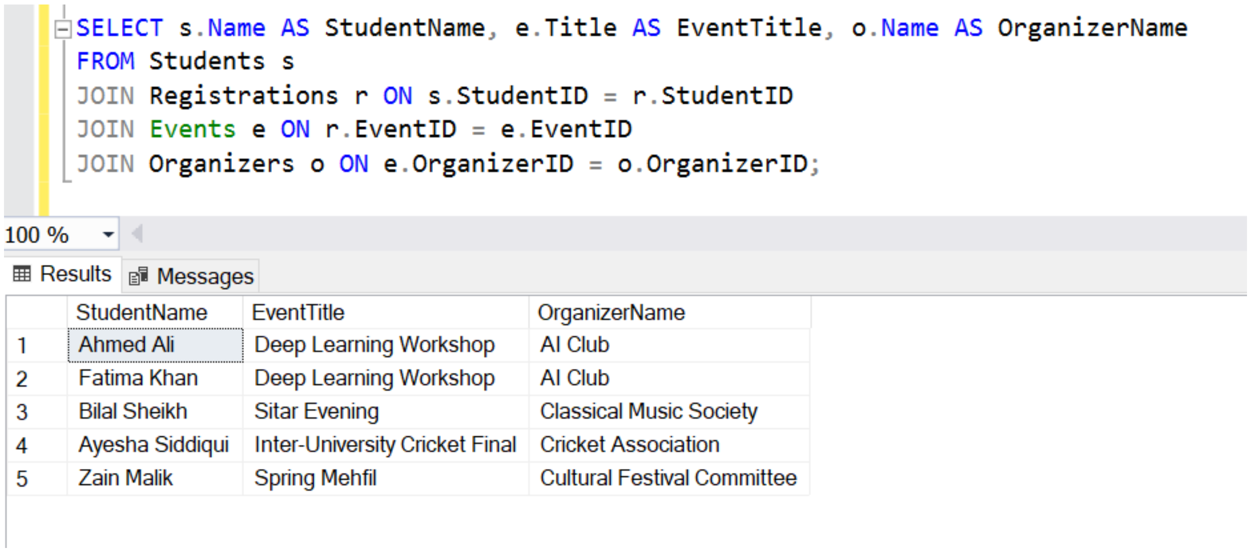
- **COUNT Applied:** On **e.EventID** to count how many events are in each category.

- **GROUP BY Applied:** On **c.CategoryName** to group all events by their category name.

- **HAVING Clause Applied:** On the result of **COUNT(e.EventID) > 1** to filter out categories with only one or zero events, keeping only categories with multiple events.

- **Purpose:** To find categories that have more than one event, showing popular or frequently used categories.

4. List of Students Registered With Organizers



```sql
SELECT s.Name AS StudentName, e.Title AS EventTitle, o.Name AS OrganizerName
FROM Students s
JOIN Registrations r ON s.StudentID = r.StudentID
JOIN Events e ON r.EventID = e.EventID
JOIN Organizers o ON e.OrganizerID = o.OrganizerID;
```

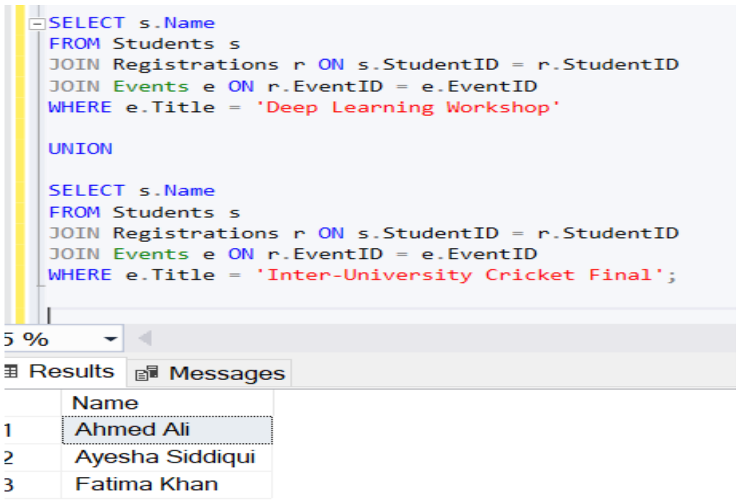
	StudentName	EventTitle	OrganizerName
1	Ahmed Ali	Deep Learning Workshop	AI Club
2	Fatima Khan	Deep Learning Workshop	AI Club
3	Bilal Sheikh	Sitar Evening	Classical Music Society
4	Ayesha Siddiqui	Inter-University Cricket Final	Cricket Association
5	Zain Malik	Spring Mehfil	Cultural Festival Committee

- **JOIN Applied:**

➢ **Students.StudentID = Registrations.StudentID** → to find each student's registrations
➢ **Registrations.EventID = Events.EventID** → to get event details for those registrations
➢ **Events.OrganizerID = Organizers.OrganizerID** → to get organizer info for each event

- **Purpose:** To list each student, which event they registered for, and who organized that event

5. Students Who Attended Either "Deep Learning Workshop" or "Inter-University Cricket Final"


```
SELECT s.Name
FROM Students s
JOIN Registrations r ON s.StudentID = r.StudentID
JOIN Events e ON r.EventID = e.EventID
WHERE e.Title = 'Deep Learning Workshop'

UNION

SELECT s.Name
FROM Students s
JOIN Registrations r ON s.StudentID = r.StudentID
JOIN Events e ON r.EventID = e.EventID
WHERE e.Title = 'Inter-University Cricket Final';
```

	Name
1	Ahmed Ali
2	Ayesha Siddiqui
3	Fatima Khan

- **JOINs:**

➢ **Students.StudentID = Registrations.StudentID** : connects students to their registrations.
➢ **Registrations.EventID = Events.EventID** : connects registrations to the events they registered for.
➢ **Events.OrganizerID = Organizers.OrganizerID** : connects each event to its organizer.

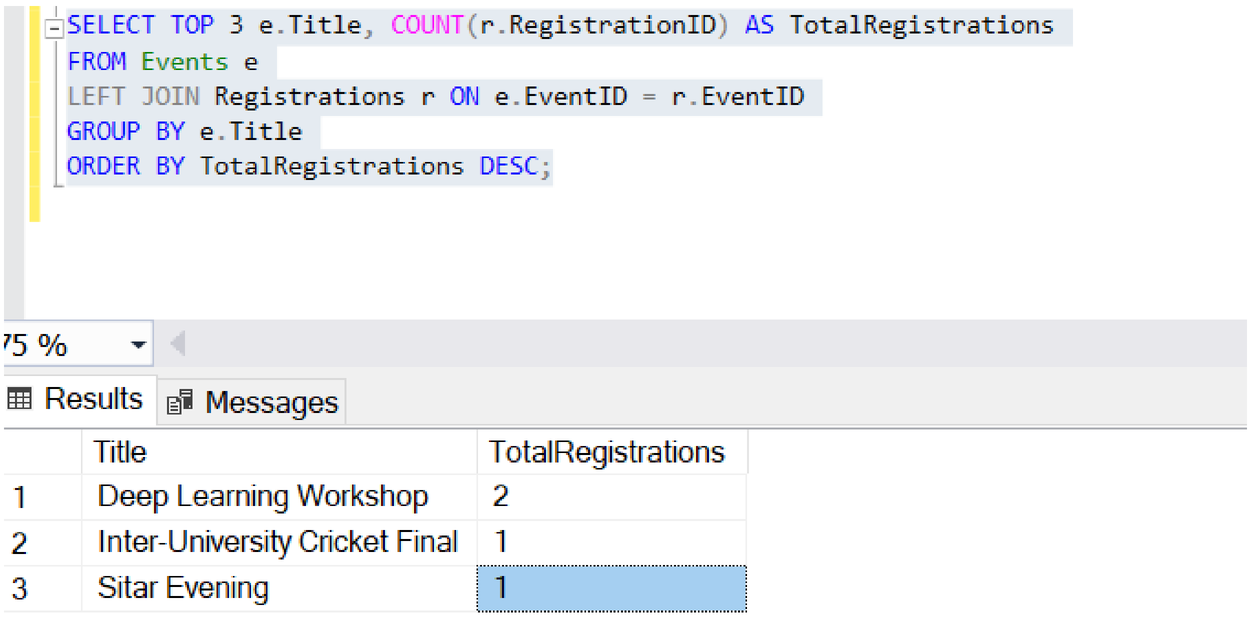
- **WHERE Filtering:**

➢ The first query filters only registrations for events where the organizer's Name is **Deep Learning Workshop"**
➢ The second query filters similarly but for events where the organizer's Name is **"Inter-University Cricket Final"**

- **SELECT columns:** Selecting distinct StudentID and Name from students who registered for these events.

- **UNION:** Combines these two lists of students **without duplicates**

6. Top 3 Most Popular Events



```sql
SELECT TOP 3 e.Title, COUNT(r.RegistrationID) AS TotalRegistrations
FROM Events e
LEFT JOIN Registrations r ON e.EventID = r.EventID
GROUP BY e.Title
ORDER BY TotalRegistrations DESC;
```

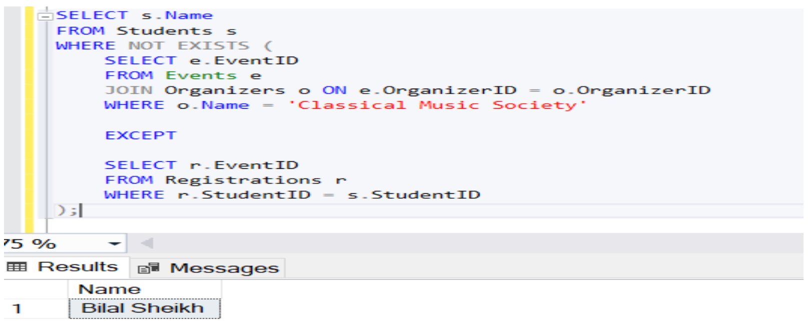
	Title	TotalRegistrations
1	Deep Learning Workshop	2
2	Inter-University Cricket Final	1
3	Sitar Evening	1

- **LEFT JOIN Applied:**

➢ Between Events.EventID and Registrations.EventID to count registrations per event including those with zero.

- **COUNT Applied:** On r.RegistrationID to count total registrations per event.
- **GROUP BY Applied:**On e.Title to aggregate registrations by event.
- **ORDER BY Applied:** Sort results descending by TotalRegistrations so the most popular events are on top.
- **TOP 3 Applied:** Limits output to only the 3 events with highest registrations.

7. Students Who Attended All 'Classical Music Society' Events



```sql
SELECT s.Name
FROM Students s
WHERE NOT EXISTS (
    SELECT e.EventID
    FROM Events e
    JOIN Organizers o ON e.OrganizerID = o.OrganizerID
    WHERE o.Name = 'Classical Music Society'

    EXCEPT

    SELECT r.EventID
    FROM Registrations r
    WHERE r.StudentID = s.StudentID
);
```

	Name
1	Bilal Sheikh

- **Purpose**:
  Identifies students who have registered for **every event** organized by the **'Classical Music Society'**.

- **Join Applied**:

➢ Between Events.OrganizerID and Organizers.OrganizerID.
➢ This connects each event with its respective organizing society.

- **WHERE Clause**: Applied to filter only those events where the organizer's name is 'Classical Music Society'.

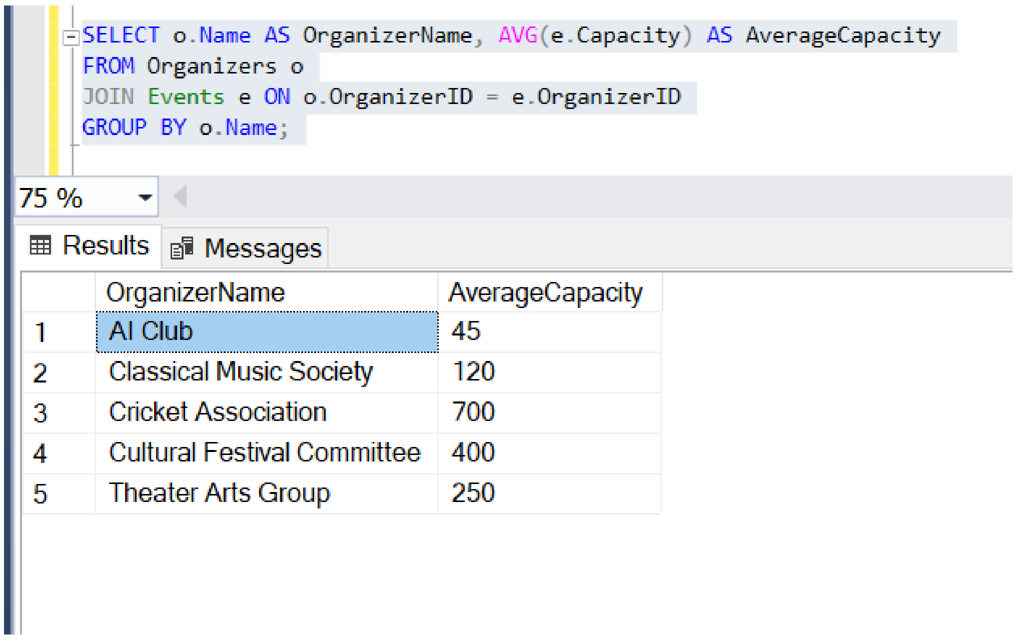
- **Set Operation - EXCEPT**:

➢ Compares the set of **Classical Music Society's events** with the set of **events a student has registered for**.
➢ Returns only the events that the student **missed**.

- **NOT EXISTS Clause**:

➢ Ensures that **no events are missing** from the student's registrations.
➢ If the EXCEPT clause returns nothing, the student is included in the final result (i.e., they attended all the events of the society).

8. Average Capacity of each event



```
SELECT o.Name AS OrganizerName, AVG(e.Capacity) AS AverageCapacity
FROM Organizers o
JOIN Events e ON o.OrganizerID = e.OrganizerID
GROUP BY o.Name;
```

	OrganizerName	AverageCapacity
1	AI Club	45
2	Classical Music Society	120
3	Cricket Association	700
4	Cultural Festival Committee	400
5	Theater Arts Group	250

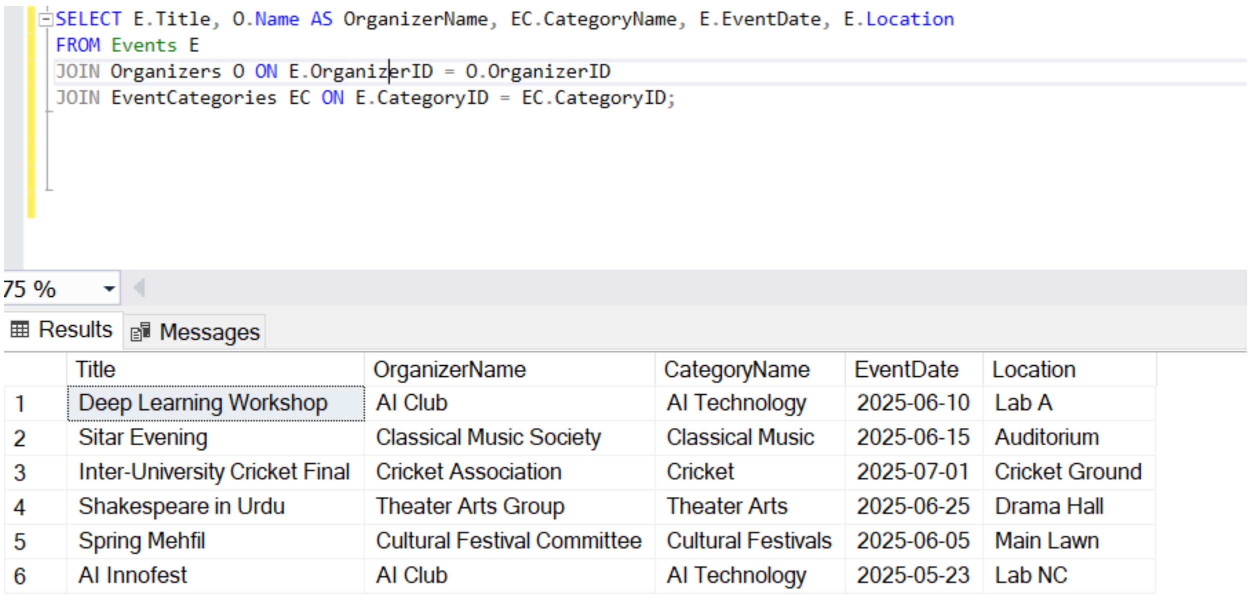
• **Purpose**: Shows how large events typically are for each organizer by calculating the **average capacity**.

• **JOIN Applied**: Between Organizers.OrganizerID and Events.OrganizerID to match each organizer with their events.

•**AVG() Applied On: e.Capacity** – to calculate the mean seating capacity for each organizer's events.

• **GROUP BY**: Applied on o.Name to group results by organizer.

9. Students Who Registered for Events from At Least 2 Different Categories



```
SELECT E.Title, O.Name AS OrganizerName, EC.CategoryName, E.EventDate, E.Location
FROM Events E
JOIN Organizers O ON E.OrganizerID = O.OrganizerID
JOIN EventCategories EC ON E.CategoryID = EC.CategoryID;
```

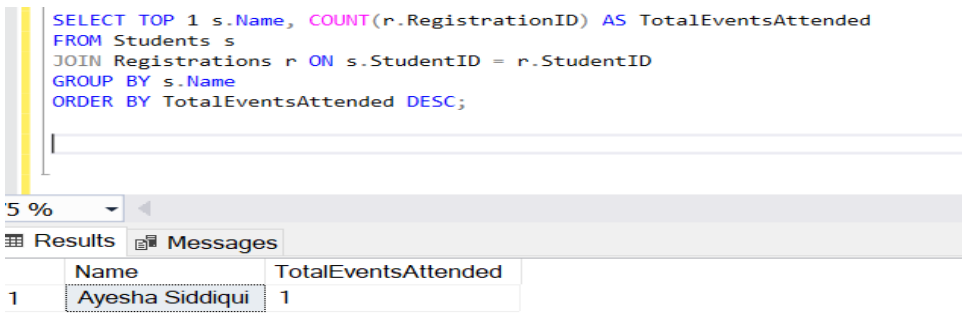
	Title	OrganizerName	CategoryName	EventDate	Location
1	Deep Learning Workshop	AI Club	AI Technology	2025-06-10	Lab A
2	Sitar Evening	Classical Music Society	Classical Music	2025-06-15	Auditorium
3	Inter-University Cricket Final	Cricket Association	Cricket	2025-07-01	Cricket Ground
4	Shakespeare in Urdu	Theater Arts Group	Theater Arts	2025-06-25	Drama Hall
5	Spring Mehfil	Cultural Festival Committee	Cultural Festivals	2025-06-05	Main Lawn
6	AI Innofest	AI Club	AI Technology	2025-05-23	Lab NC

- **JOIN applied:**

➢ Between Events.OrganizerID and Organizers.OrganizerID for organizer details.
➢ Between Events.CategoryID and EventCategories.CategoryID for category details.

- **Purpose:**Provides a complete list of events with their organizer and category, useful for browsing all events.

10. Most Active Student

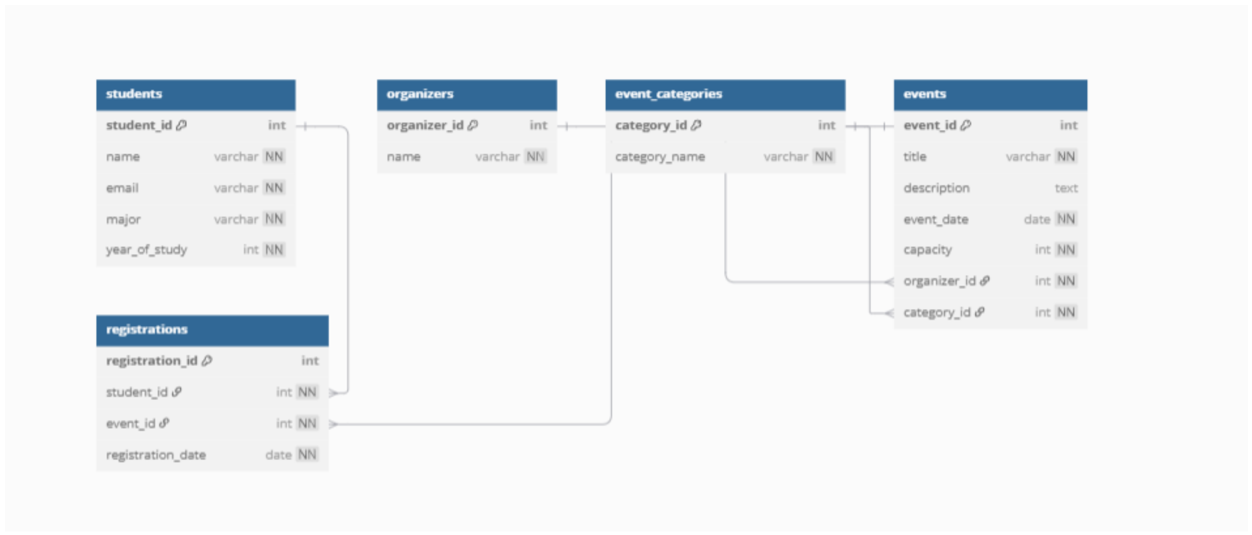


```sql
SELECT TOP 1 s.Name, COUNT(r.RegistrationID) AS TotalEventsAttended
FROM Students s
JOIN Registrations r ON s.StudentID = r.StudentID
GROUP BY s.Name
ORDER BY TotalEventsAttended DESC;
```

```
'5 %
Results   Messages
    Name              TotalEventsAttended
1   Ayesha Siddiqui   1
```

- **JOIN Applied:** Between Students.StudentID and Registrations.StudentID.
- **COUNT Applied:** On r.RegistrationID to count how many events each student attended.
- **GROUP BY Applied:** On s.Name to count per student.
- **ORDER BY Applied:** Sorts students by descending attendance.
- **TOP 1 Applied:** Returns only the student with highest attendance.
- **Purpose:** To find the most active student in terms of event participation.

<u>ER- DIAGRAM:</u>



- **Students and Registrations**

  - Relationship: One-to-Many
  - Each student can register for multiple events (many registrations).
  - Each registration belongs to exactly one student.

- **Events and Registrations**

  - Relationship: One-to-Many
  - Each event can have many student registrations.

    - Each registration refers to exactly one event.

- **Organizers and Events**

    - Relationship: One-to-Many
    - Each organizer (club/society) can organize multiple events.
    - Each event is organized by exactly one organizer.

- **EventCategories and Events**

    - Relationship: One-to-Many
    - Each category (e.g., AI Technology, Music Jam) includes many events.
    - Each event belongs to exactly one category.

- **Summary of cardinalities:**

    - Students ↔ Registrations = 1 : Many
    - Events ↔ Registrations = 1 : Many
    - Organizers ↔ Events = 1 : Many
    - EventCategories ↔ Events = 1 : Many