DBT Assignment 1 : College Club Management

Name : Mohul Y P

SRN : PES1UG22CS360

a)Database preparation

Clubs Table: The central entity that stores information about different clubs

- **ClubID (primary key)**
- **ClubName (required)**
- **ClubType (categorizes clubs, e.g., "Sports")**
- **EstablishmentYear (when the club was founded)**

Members Table: Stores student/member information with a link to their club

- **MemberID (primary key)**
- **FirstName, LastName (required)**
- **Email (required, must be unique)**
- **PhoneNumber**
- **DOB (date of birth)**
- **ClubID (foreign key to Clubs table)**

Events Table: Tracks events organized by clubs

- **EventID (primary key)**
- **EventName (required)**
- **EventDate, EventLocation**
- **EventDescription (TEXT type for longer descriptions)**
- **ClubID (foreign key to Clubs table)**

Activities Table: Records specific activities conducted by clubs

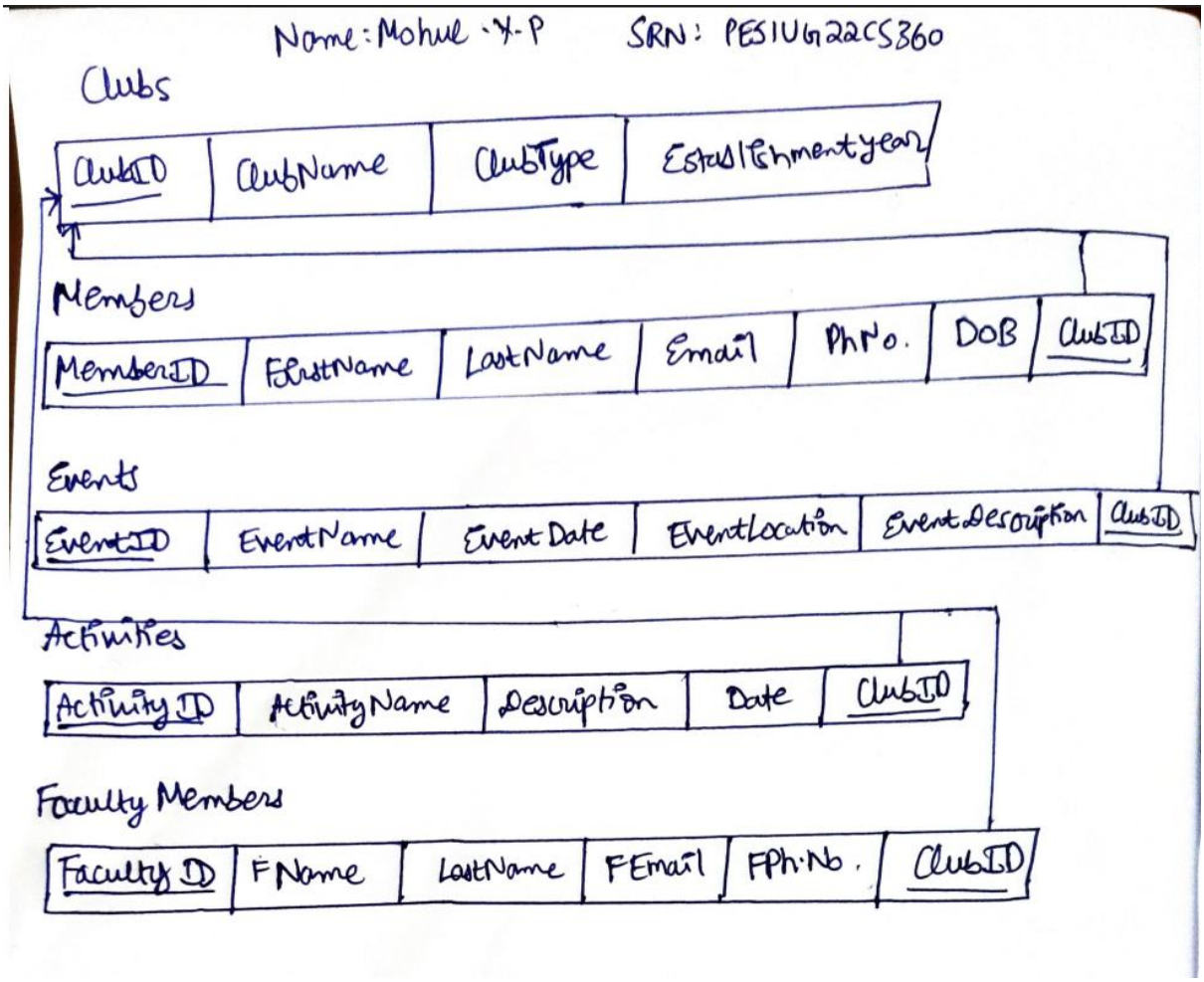
- **ActivityID (primary key)**
- **ActivityName (required)**
- **ActivityDescription**

- **ActivityDate**
- **ClubID (foreign key to Clubs table)**

FacultyMembers Table: Stores information about faculty members associated with clubs

- **FacultyID (primary key)**
- **FirstName, LastName (required)**
- **Email (required, must be unique)**
- **PhoneNumber**
- **ClubID (foreign key to Clubs table)**

Relational Schema:

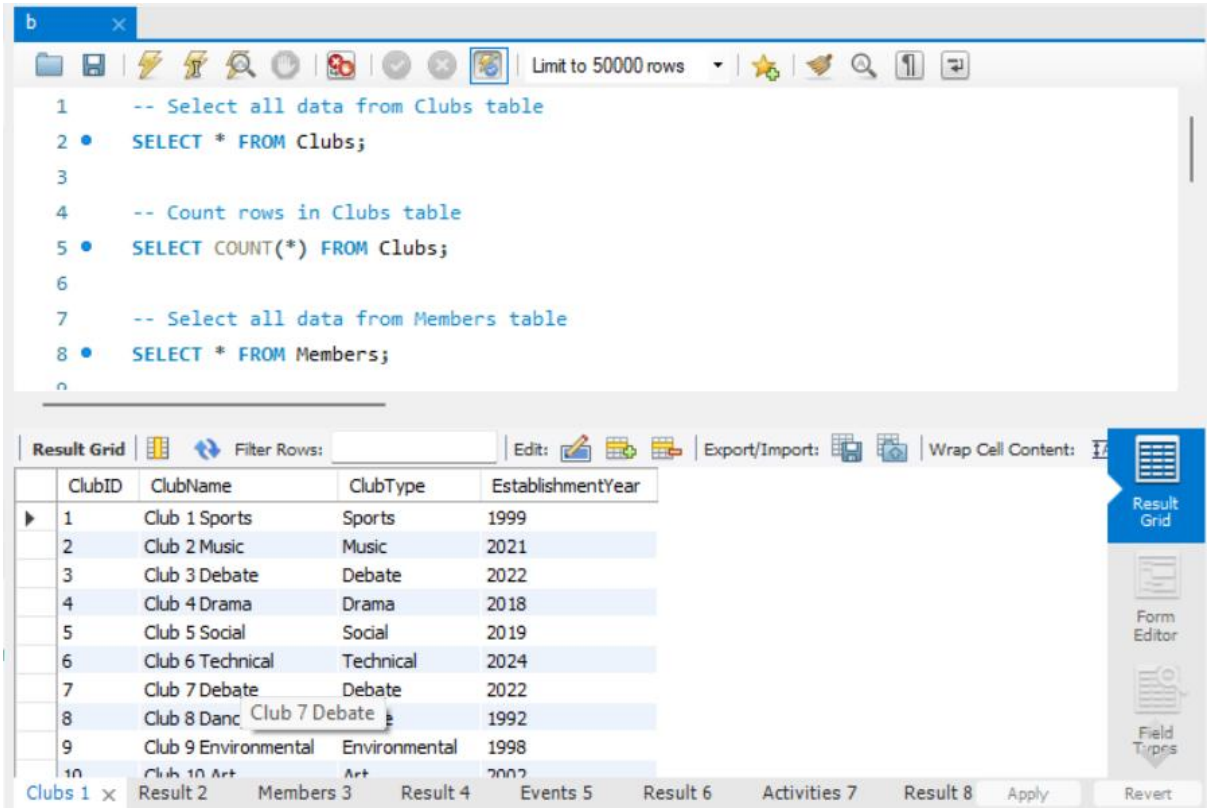


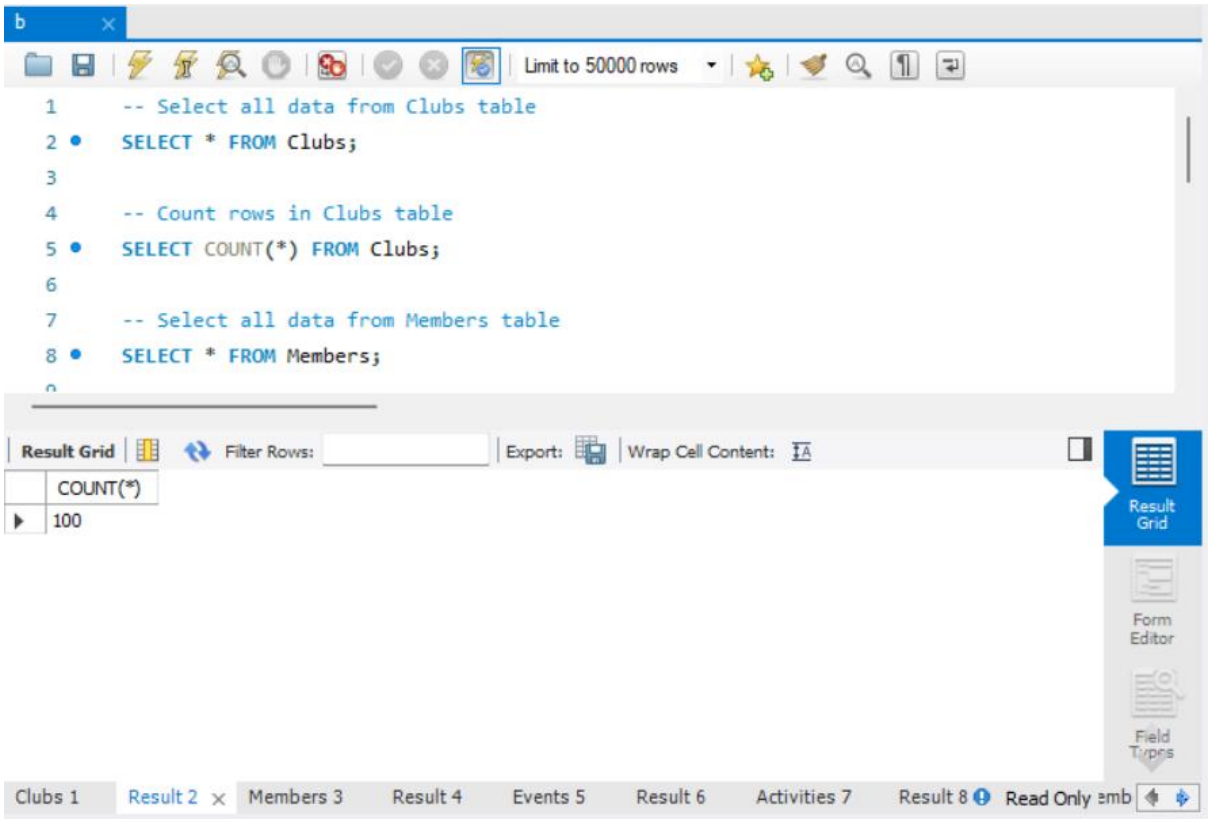
To fill the tables with 10,000 rows, I used a python script which will be in the folder along with this submission.

b)**select and count queries on “clubs” table**

SELECT * FROM Clubs;

SELECT COUNT(*) FROM Clubs;

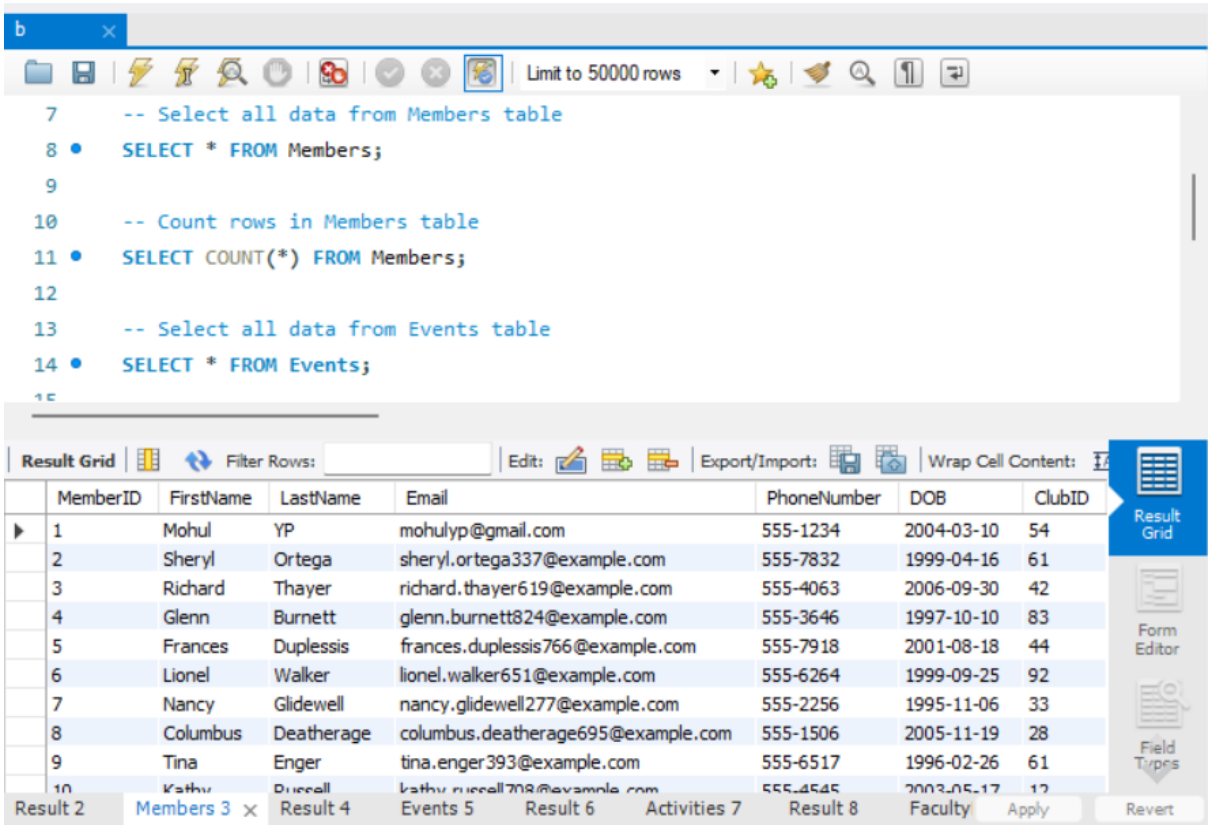


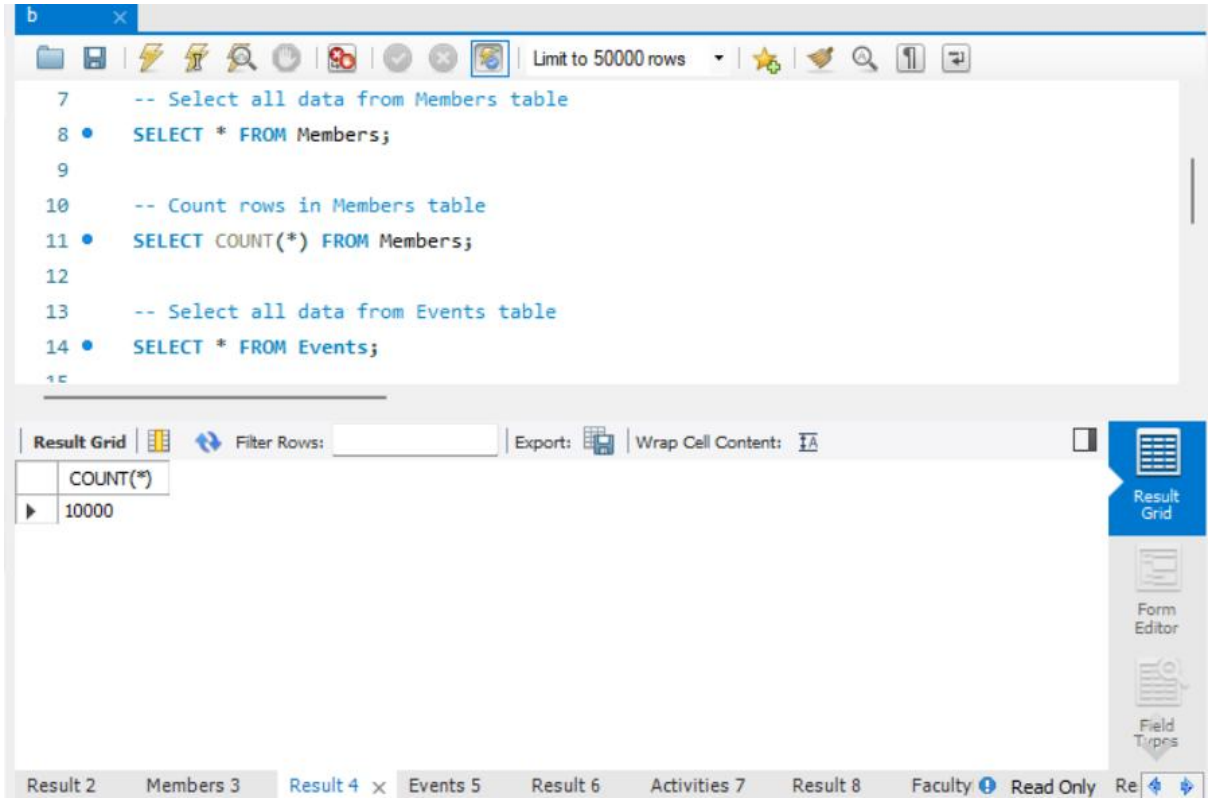


select and count queries on “member” table

SELECT * FROM Members;

SELECT COUNT(*) FROM Members;

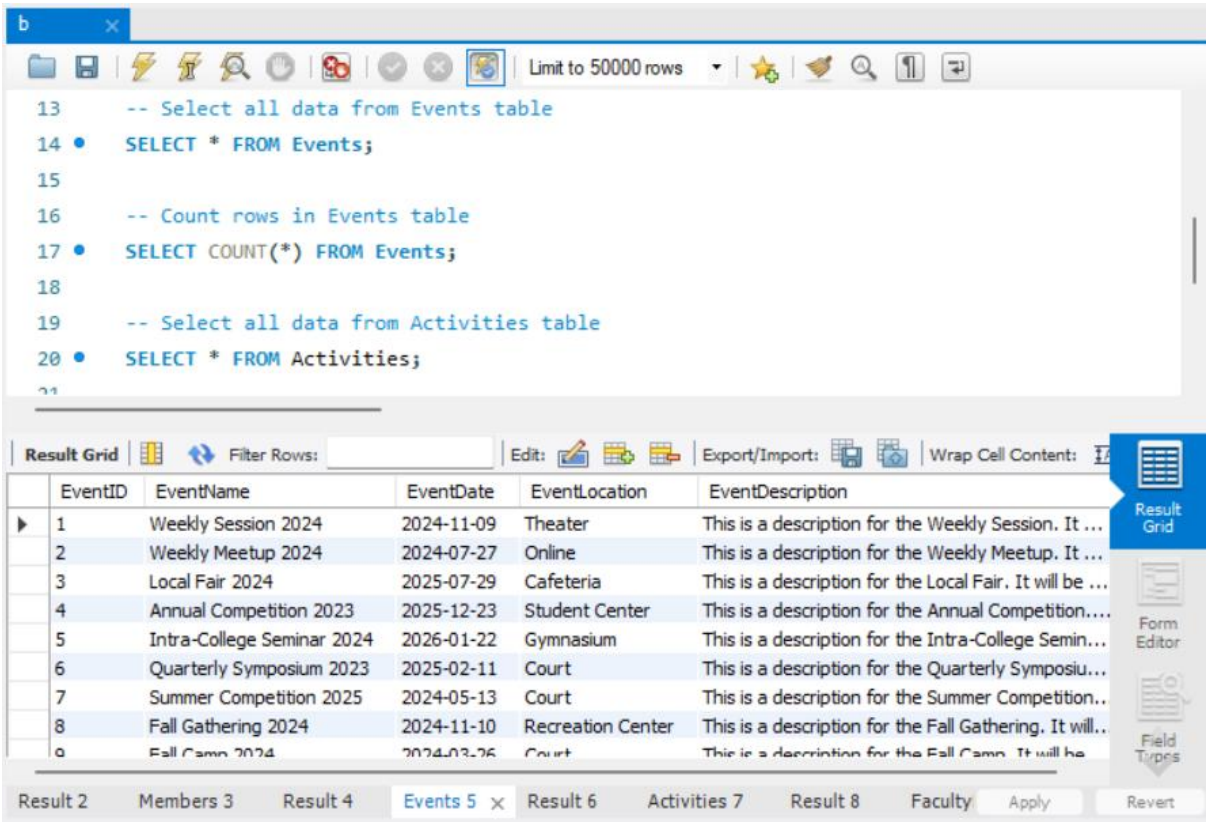


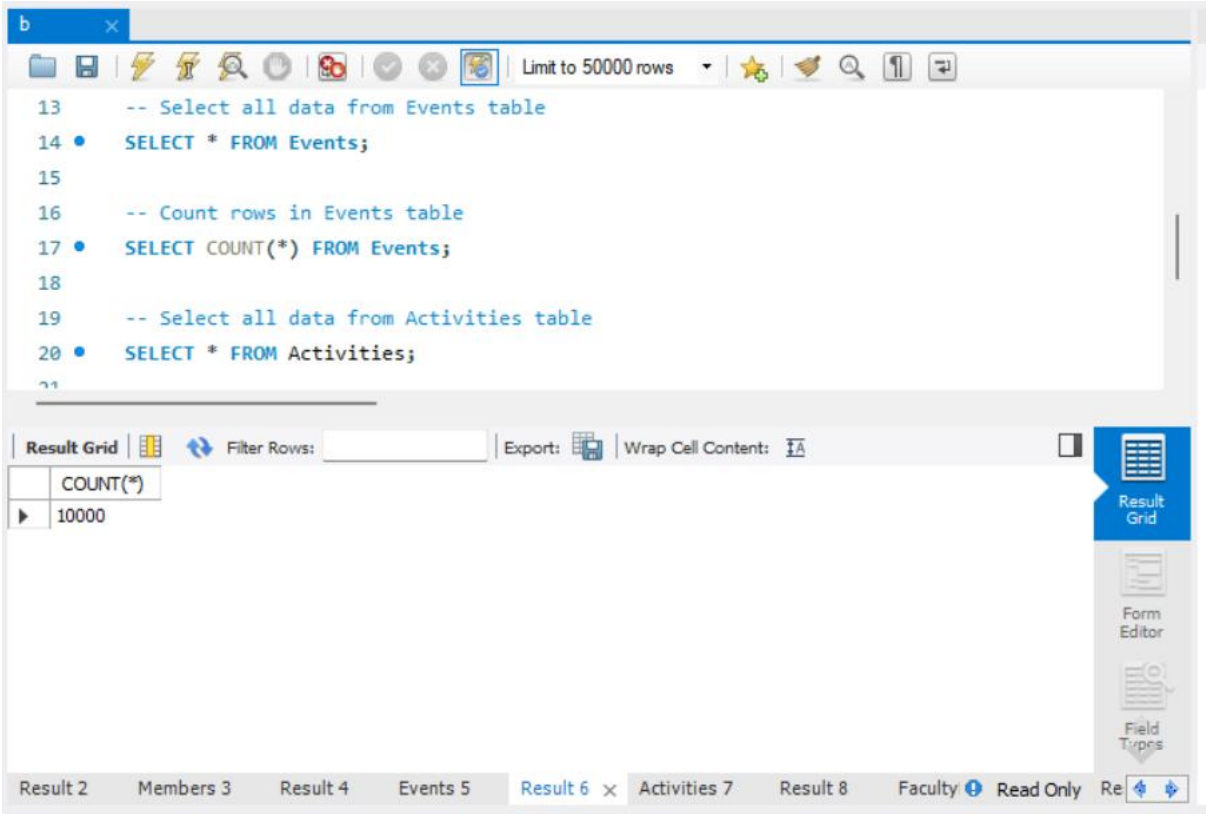


select and count queries on “events” table

SELECT * FROM Events;

SELECT COUNT(*) FROM Events;

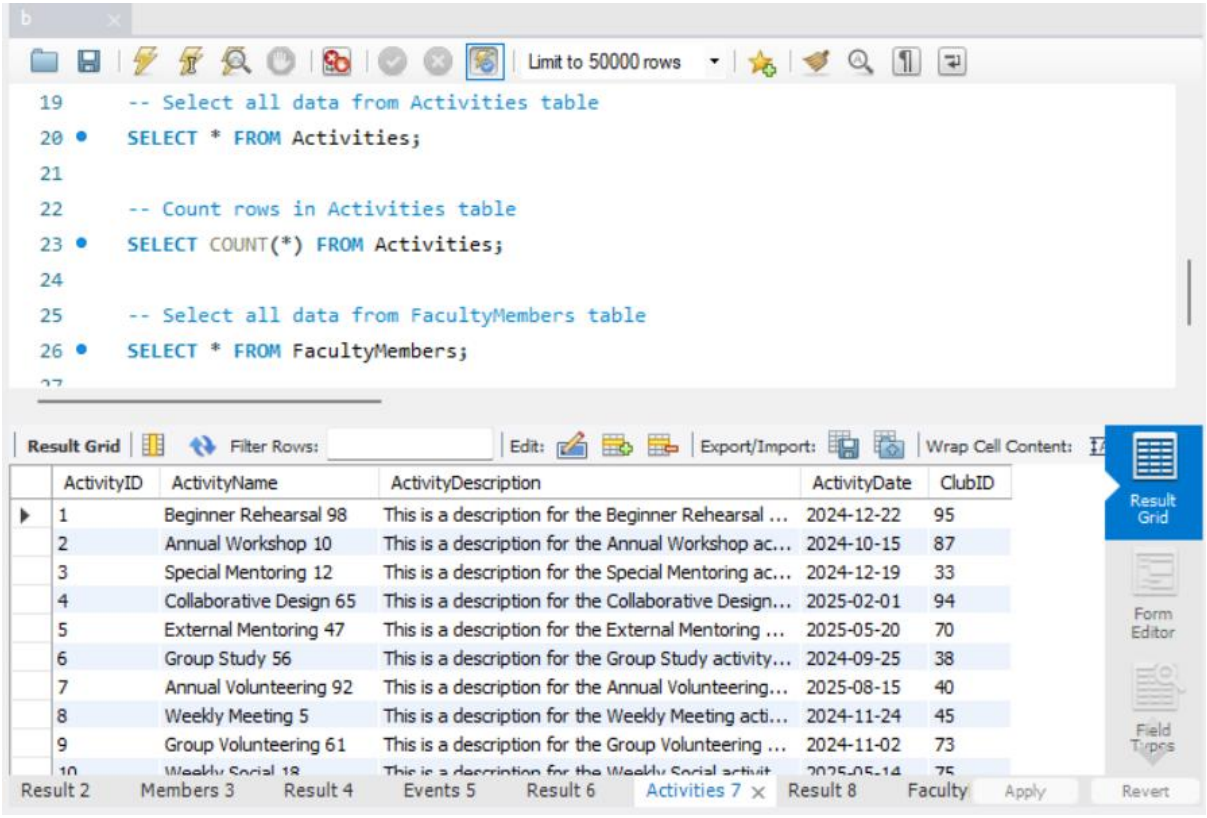


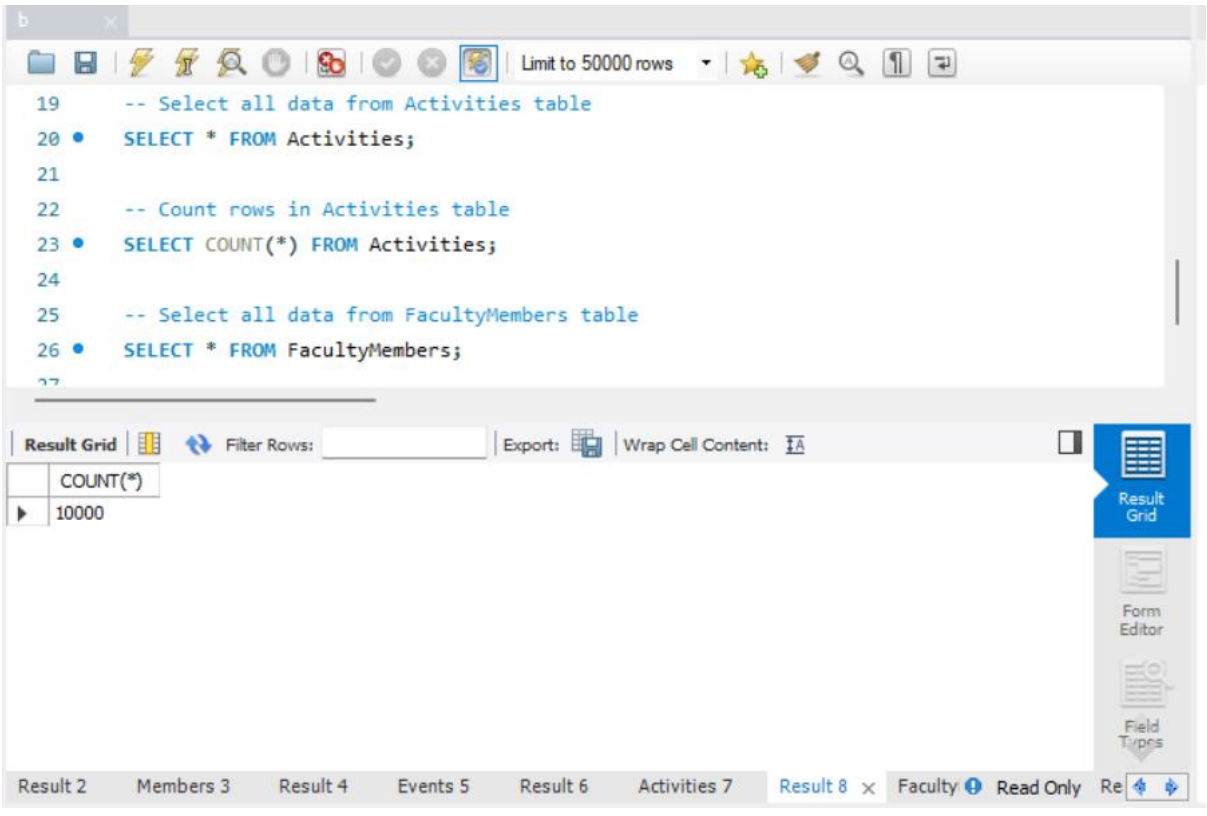


select and count queries on “activities” table

SELECT * FROM Activities;

SELECT COUNT(*) FROM Activities;

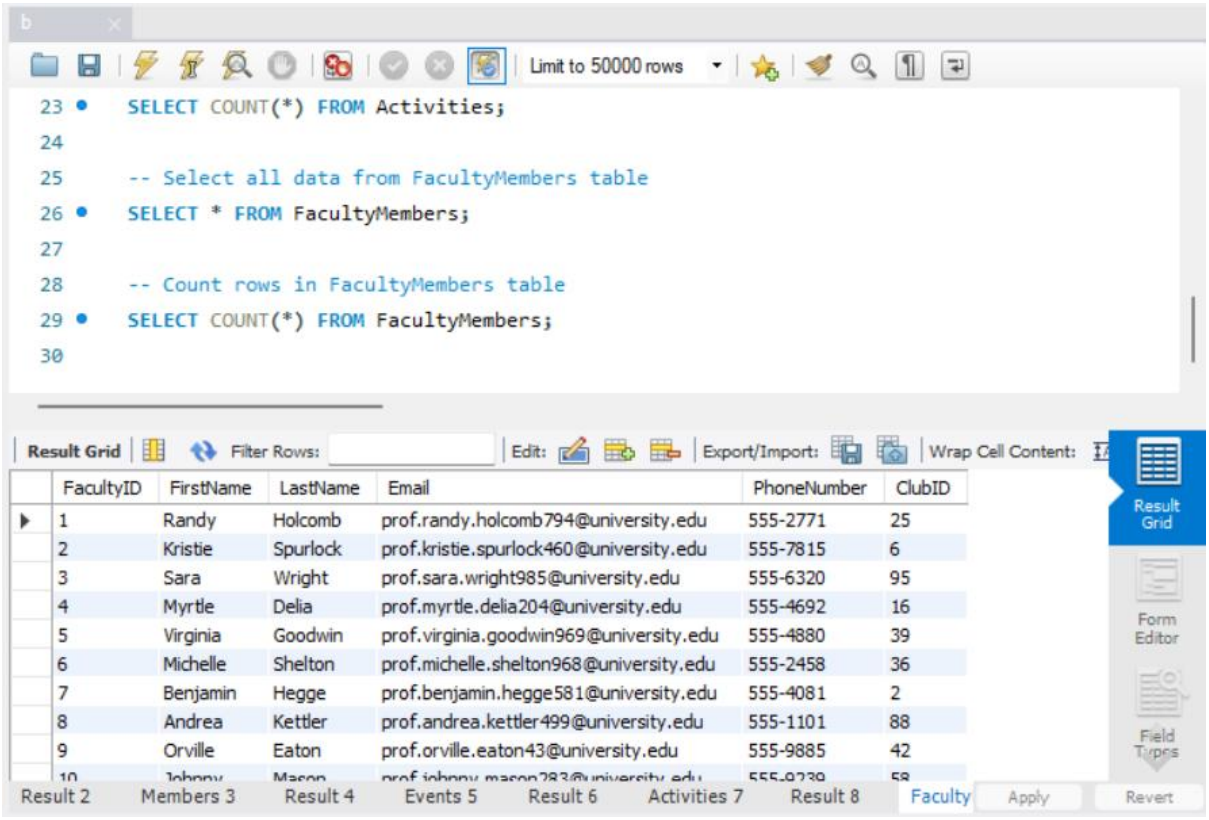


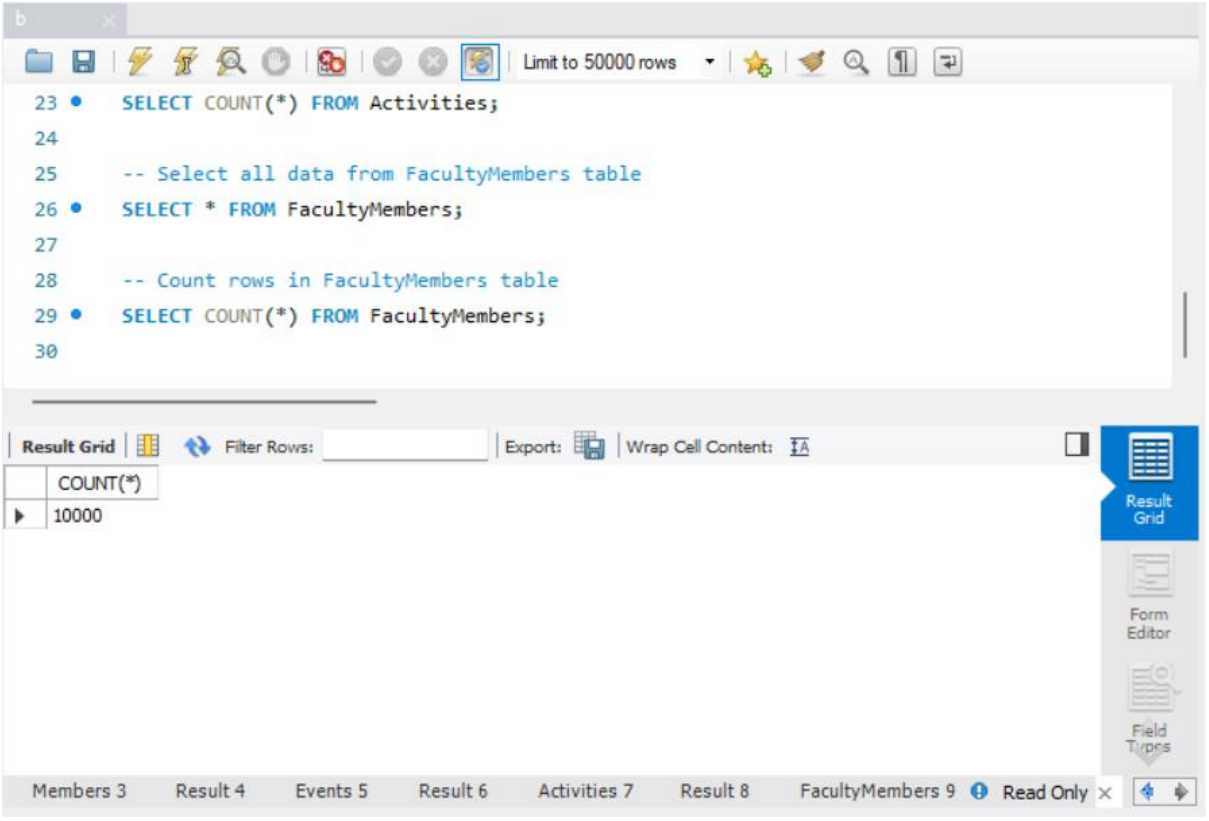


select and count queries on “faculty members” table

SELECT * FROM FacultyMembers;

SELECT COUNT(*) FROM FacultyMembers;





INDEX SCAN

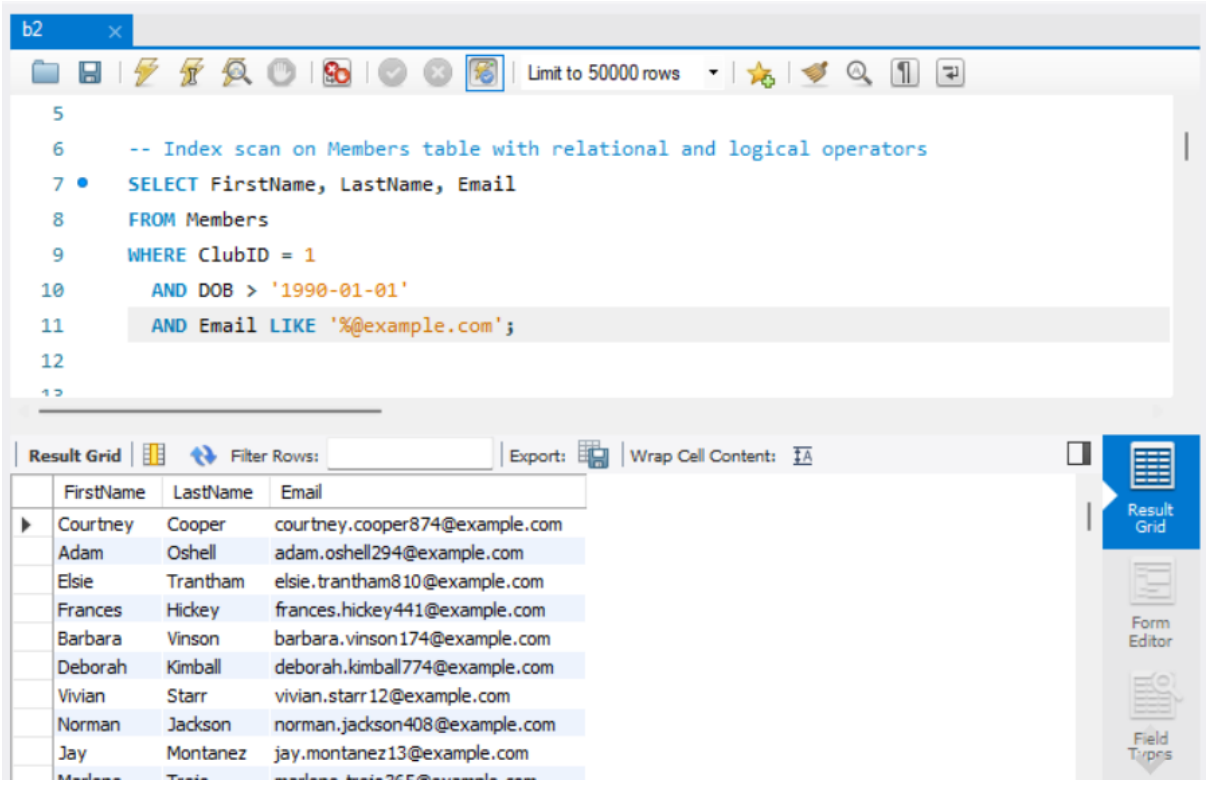
1) SELECT FirstName, LastName, Email

FROM Members

WHERE ClubID = 1

AND DOB > '1990-01-01'

AND Email LIKE '%@example.com';

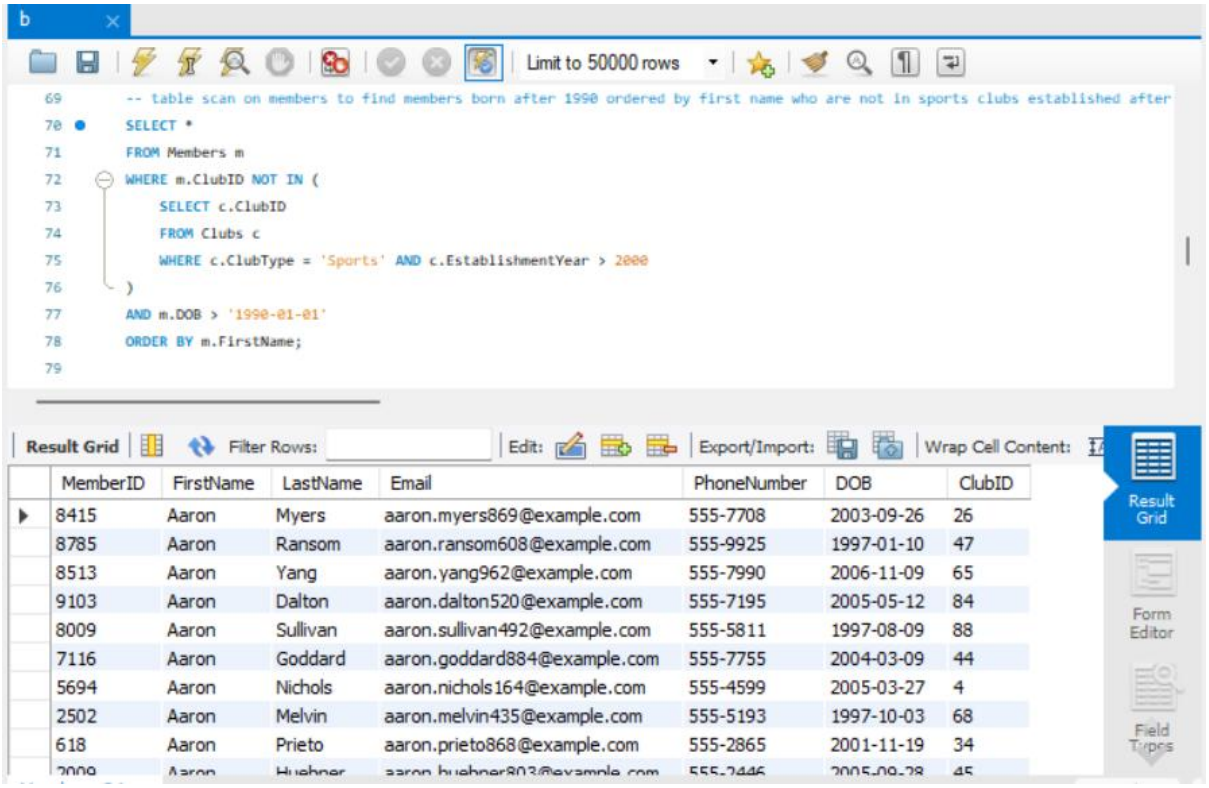


ClubID is automatically an index as it is a primary key

TABLE SCAN

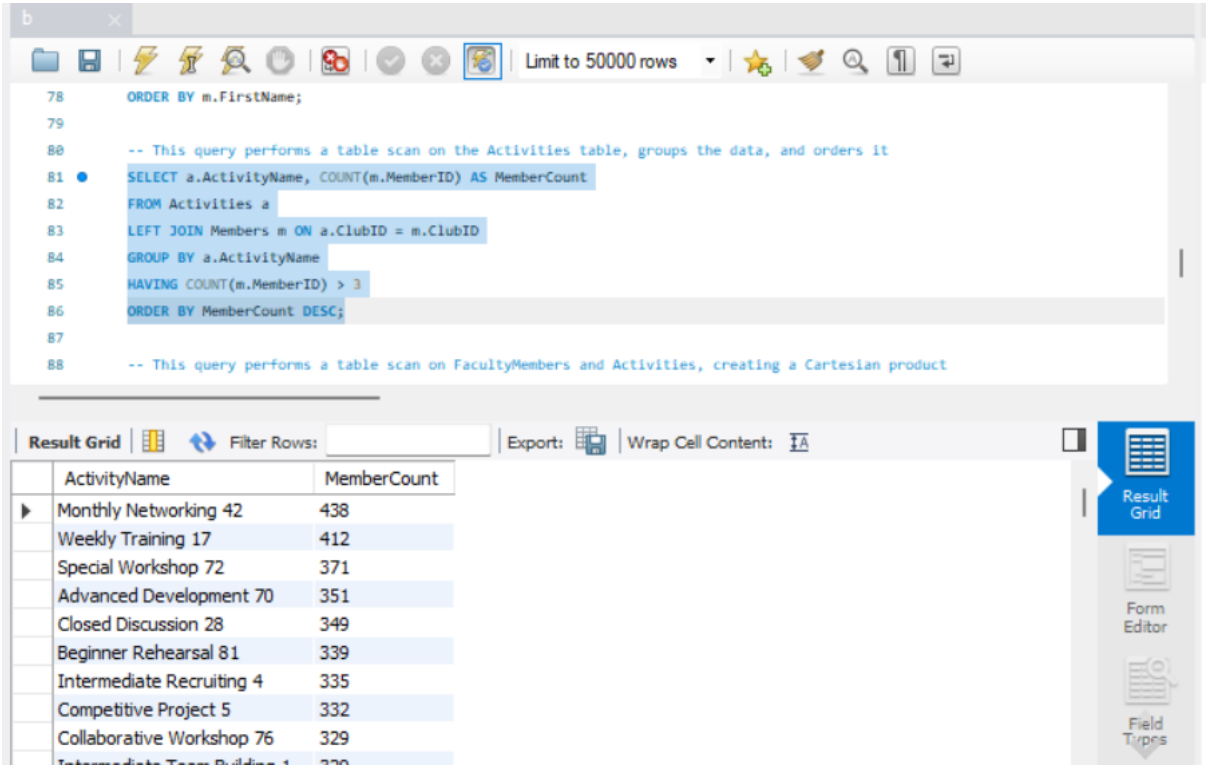
1) **– table scan on members to find members born after 1990 ordered by first name who are not in sports clubs established after 2000**

```
SELECT *
FROM Members m
WHERE m.ClubID NOT IN (
  SELECT c.ClubID
  FROM Clubs c
  WHERE c.ClubType = 'Sports' AND c.EstablishmentYear > 2000
)
AND m.DOB > '1990-01-01'
ORDER BY m.FirstName;
```



2) --table scan on join of activities and members table.

```
SELECT a.ActivityName, COUNT(m.MemberID) AS MemberCount
FROM Activities a
LEFT JOIN Members m ON a.ClubID = m.ClubID
GROUP BY a.ActivityName
HAVING COUNT(m.MemberID) > 3
ORDER BY MemberCount DESC;
```

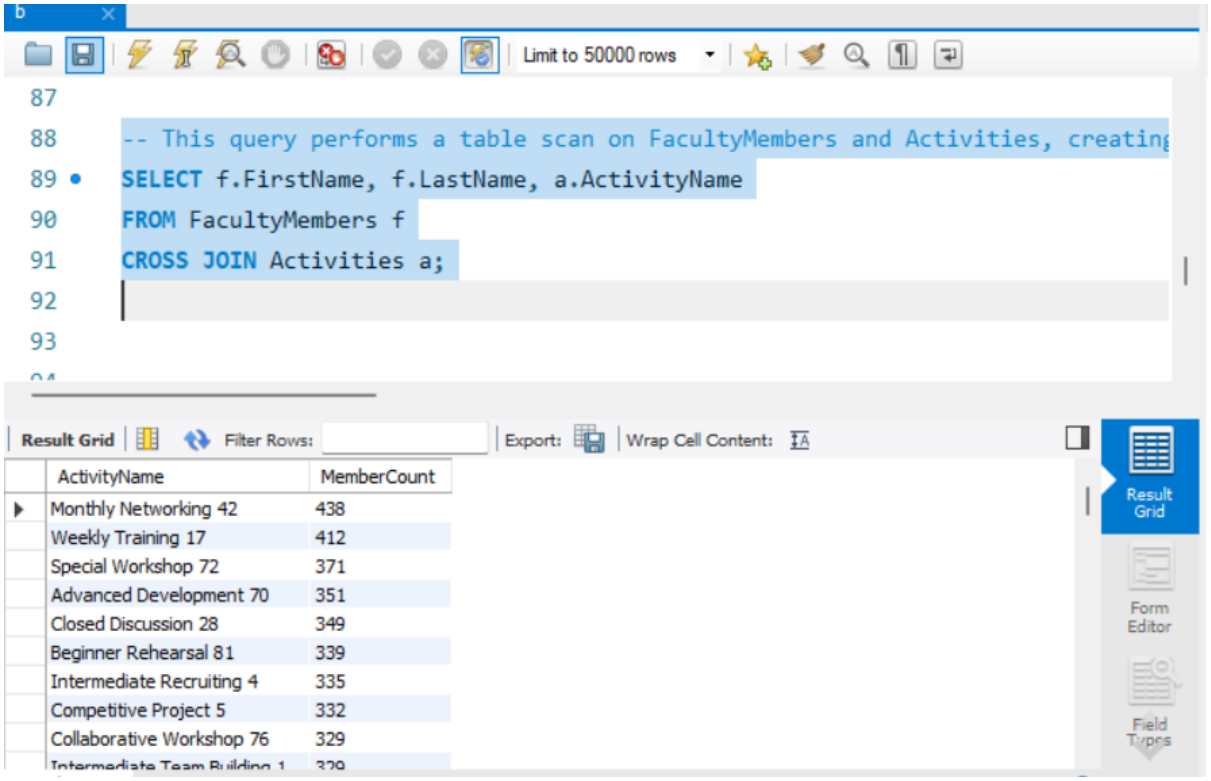


3) -- This query performs a table scan on FacultyMembers and Activities, creating a Cartesian product

SELECT f.FirstName, f.LastName, a.ActivityName

FROM FacultyMembers f

CROSS JOIN Activities a;



queries with multi-table joins involving 3 tables; including both "SELECT *" and conditional "SELECT" queries with a subset of columns.

1) SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount

FROM Members m

INNER JOIN Clubs c ON m.ClubID = c.ClubID

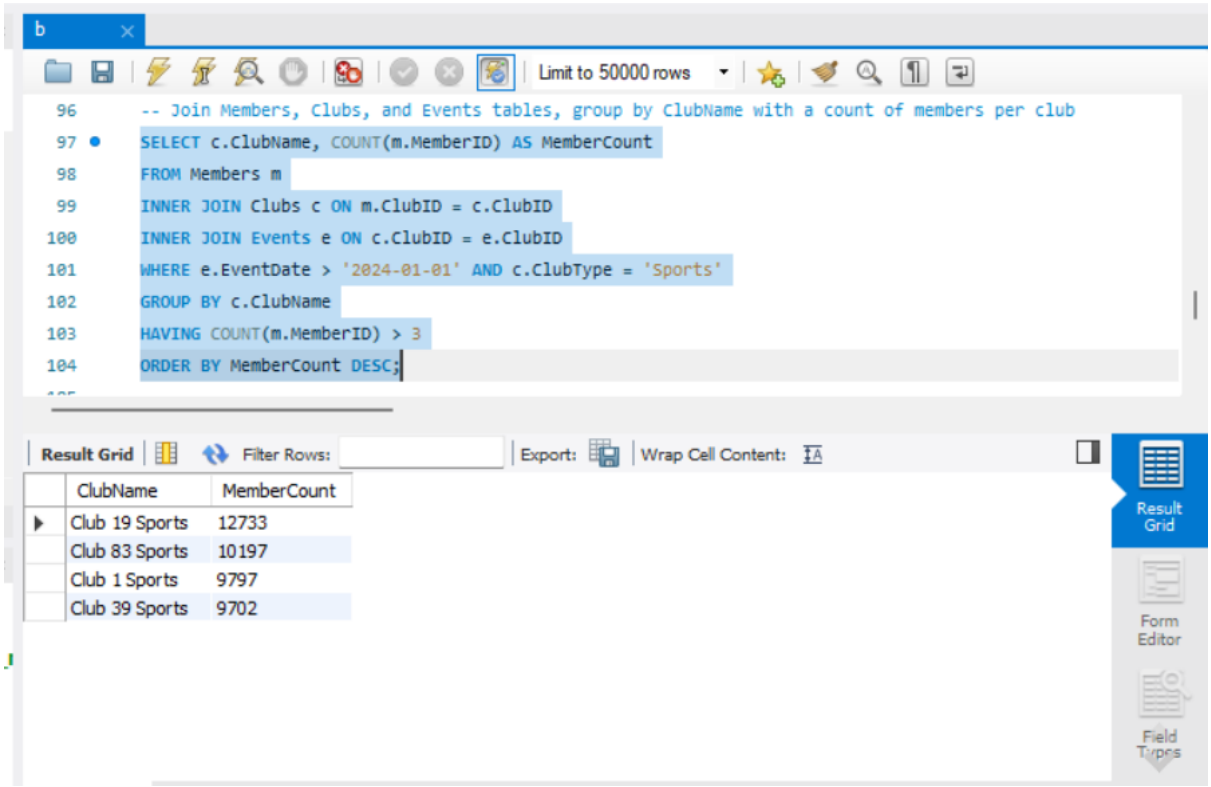
INNER JOIN Events e ON c.ClubID = e.ClubID

WHERE e.EventDate > '2024-01-01' AND c.ClubType = 'Sports'

GROUP BY c.ClubName

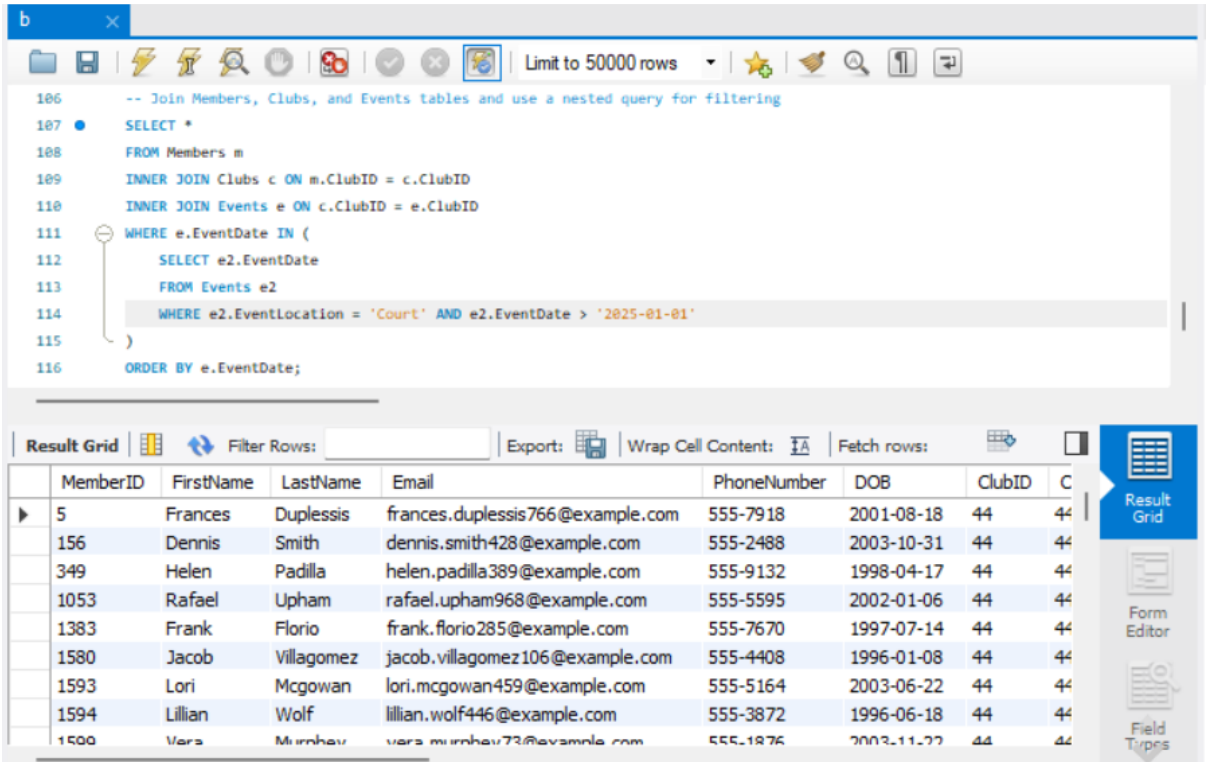
HAVING COUNT(m.MemberID) > 3

ORDER BY MemberCount DESC;



2)

```
SELECT *
FROM Members m
INNER JOIN Clubs c ON m.ClubID = c.ClubID
INNER JOIN Events e ON c.ClubID = e.ClubID
WHERE e.EventDate IN (
  SELECT e2.EventDate
  FROM Events e2
  WHERE e2.EventLocation = 'Hall A' AND e2.EventDate > '2025-01-01'
)
ORDER BY e.EventDate;
```



3) CREATE VIEW ClubMembers AS

SELECT c.ClubName, m.FirstName, m.LastName, m.Email

FROM Clubs c

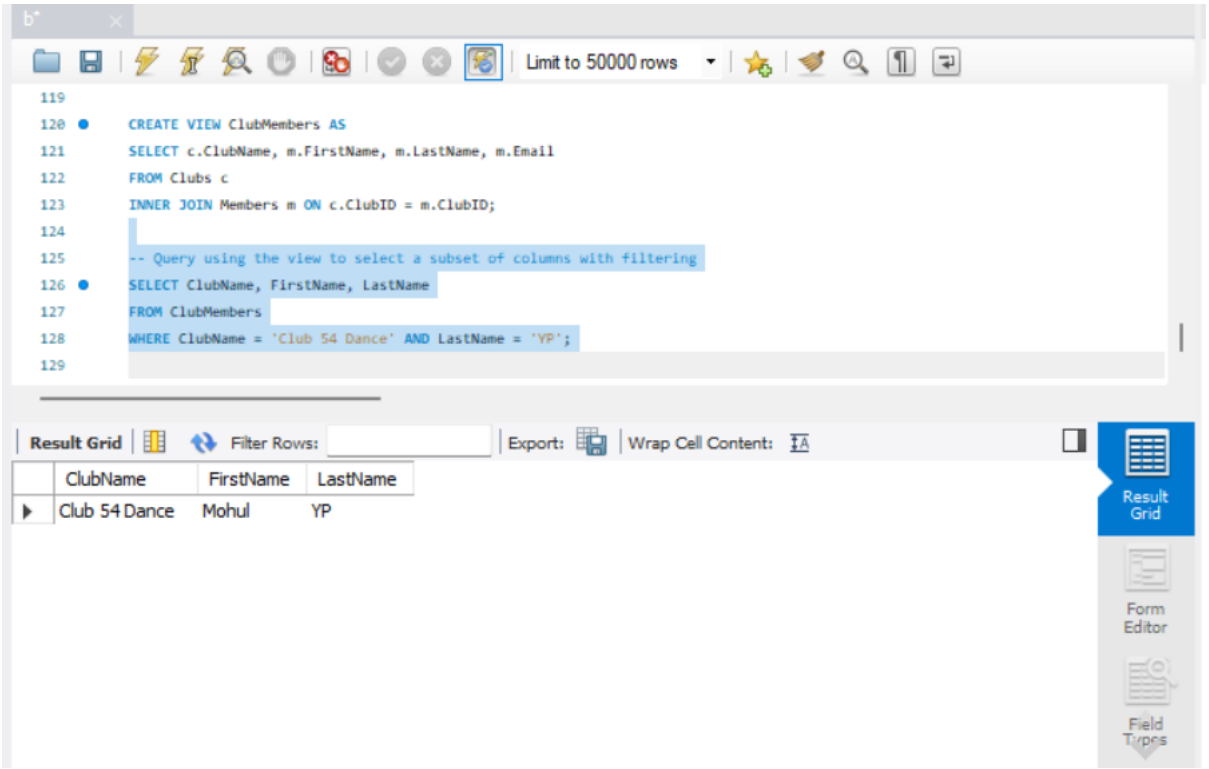
INNER JOIN Members m ON c.ClubID = m.ClubID;

-- Query using the view to select a subset of columns with filtering

SELECT ClubName, FirstName, LastName

FROM ClubMembers

WHERE ClubName = 'Club 54 Dance' AND LastName = 'YP';



EXPLAIN ANALYZE on each of the queries mentioned above :

INDEX SCAN

1) EXPLAIN ANALYZE

SELECT FirstName, LastName, Email

FROM Members

WHERE ClubID = 1

AND DOB > '1990-01-01'

AND Email LIKE '%@example.com';

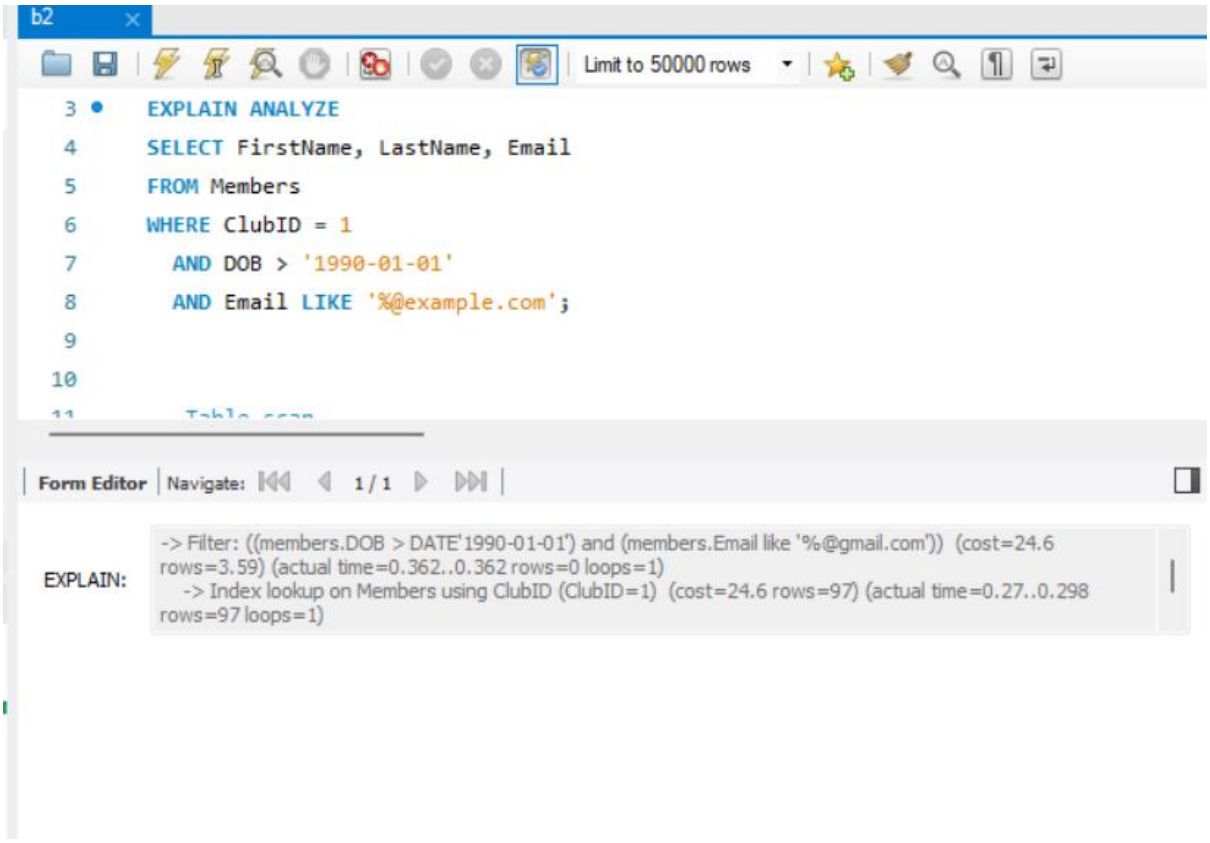
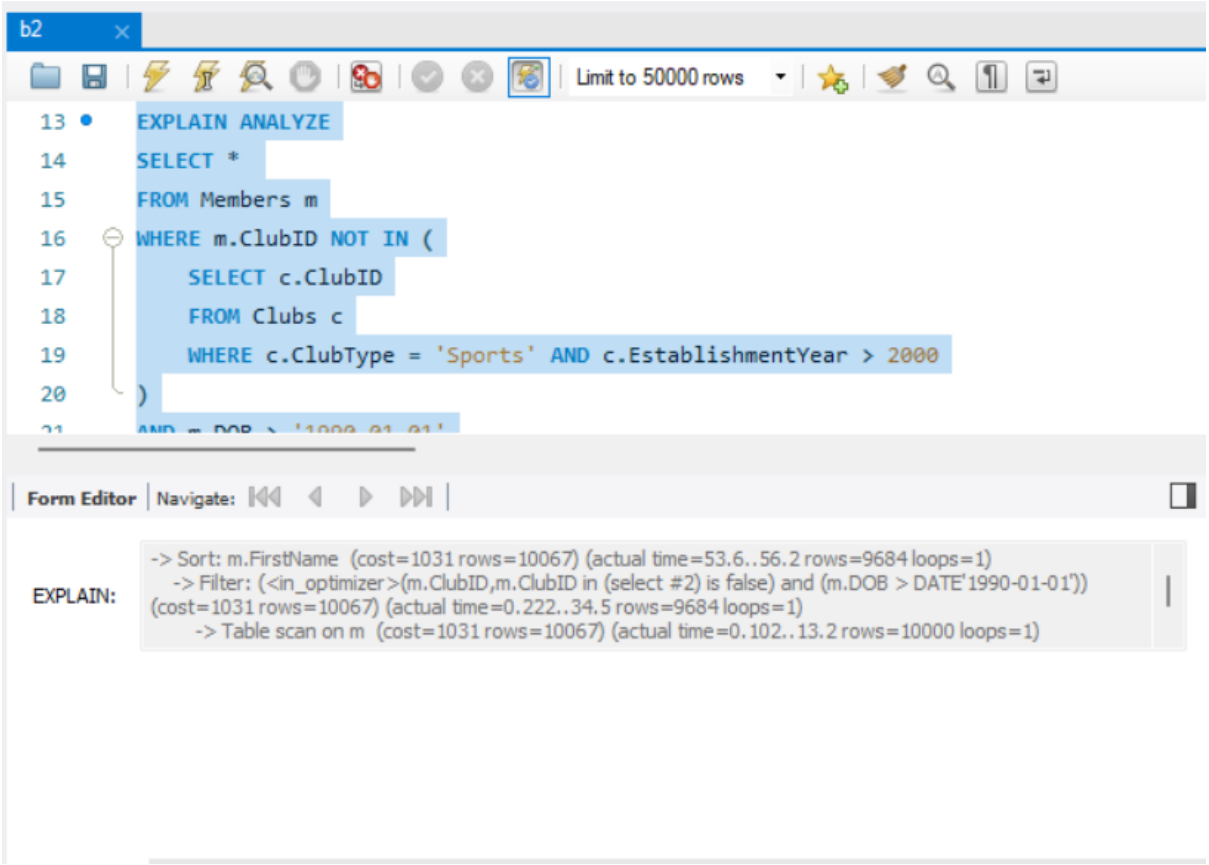


TABLE SCAN

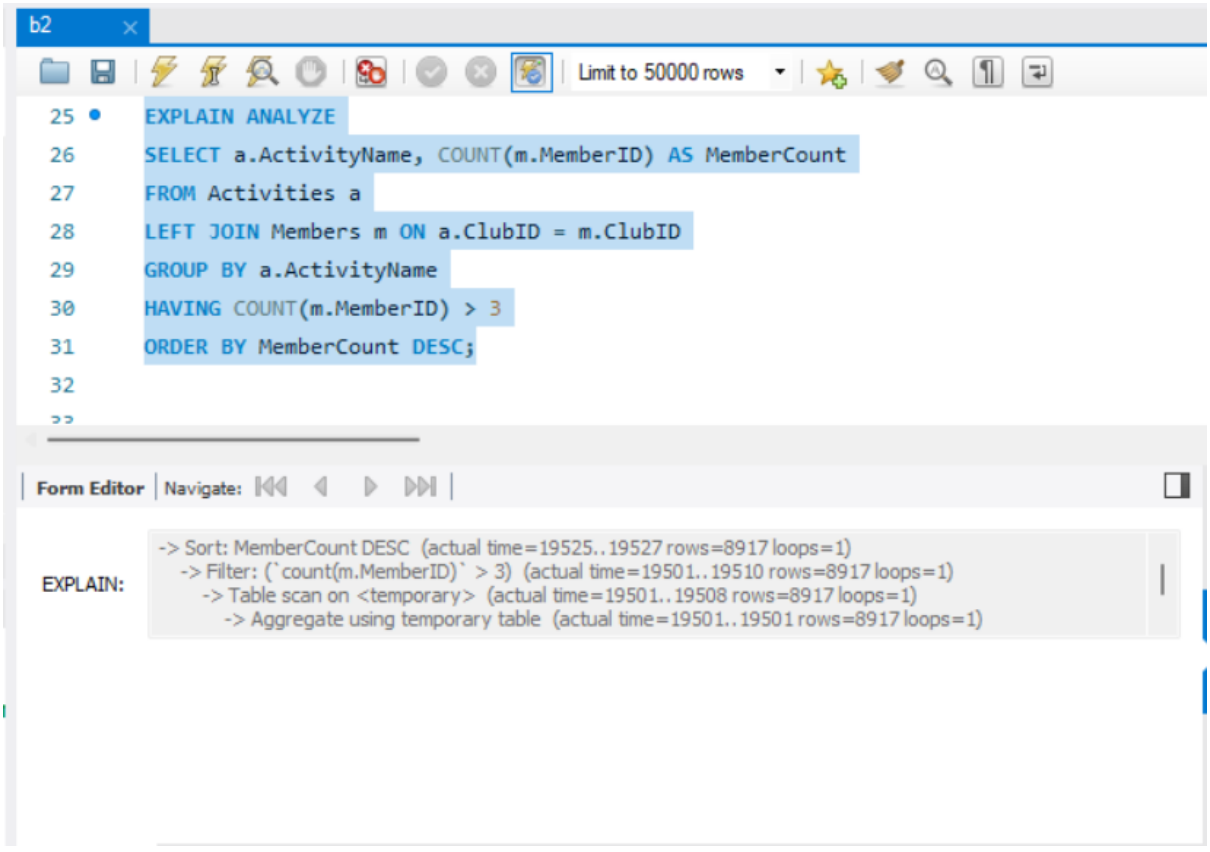
1)

```
EXPLAIN ANALYZE
SELECT *
FROM Members m
WHERE m.ClubID NOT IN (
   SELECT c.ClubID
   FROM Clubs c
   WHERE c.ClubType = 'Sports' AND c.EstablishmentYear > 2000
)
AND m.DOB > '1990-01-01'
ORDER BY m.FirstName;
```



2)

```
EXPLAIN ANALYZE
SELECT a.ActivityName, COUNT(m.MemberID) AS MemberCount
FROM Activities a
LEFT JOIN Members m ON a.ClubID = m.ClubID
GROUP BY a.ActivityName
HAVING COUNT(m.MemberID) > 3
ORDER BY MemberCount DESC;
```



3)

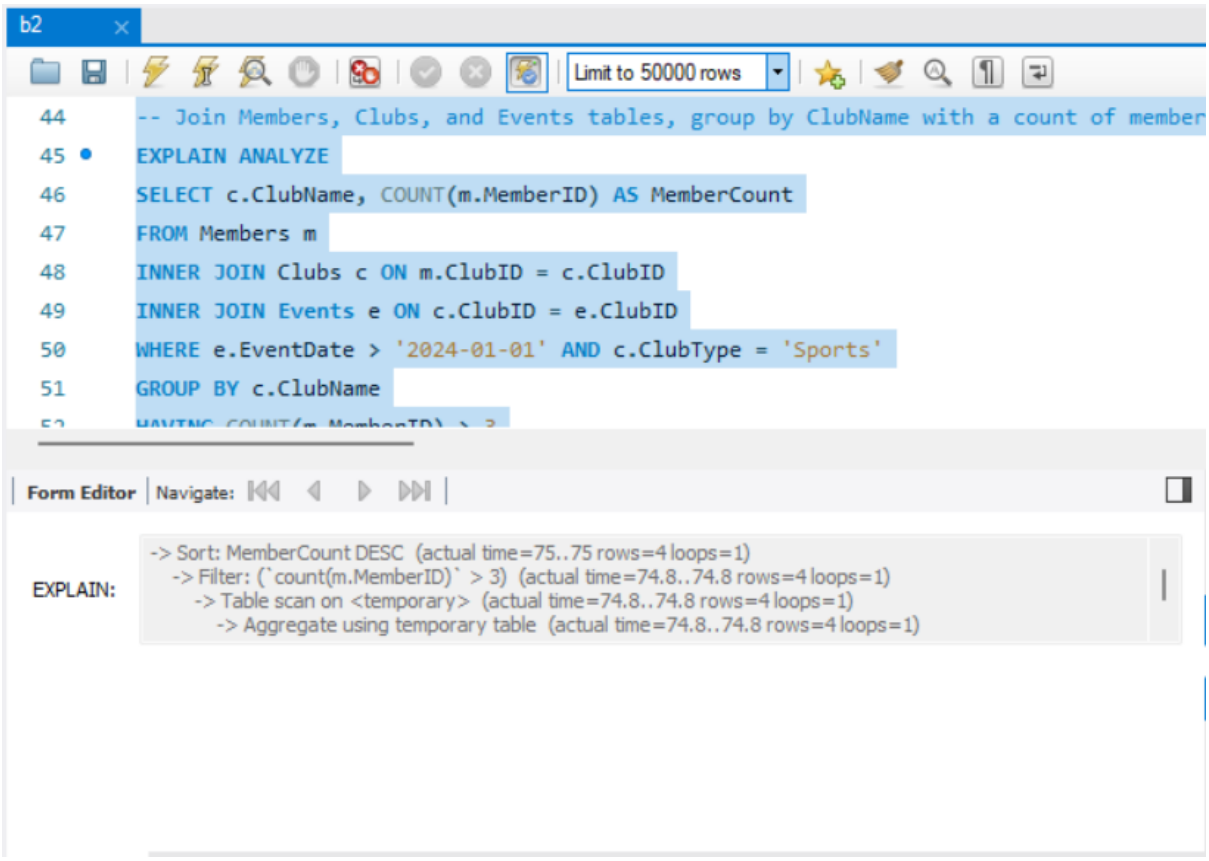
```
EXPLAIN ANALYZE

SELECT f.FirstName, f.LastName, a.ActivityName

FROM FacultyMembers f

CROSS JOIN Activities a;
```

Since we are doing a cross join(approximately 100000000 rows), the execution time is a lot and exceeds the limit set by the SQL server. Therefore, EXPLAIN ANALYZE could not be run

queries with multi-table joins involving 3 tables; including both "SELECT *" and conditional "SELECT" queries with a subset of columns.

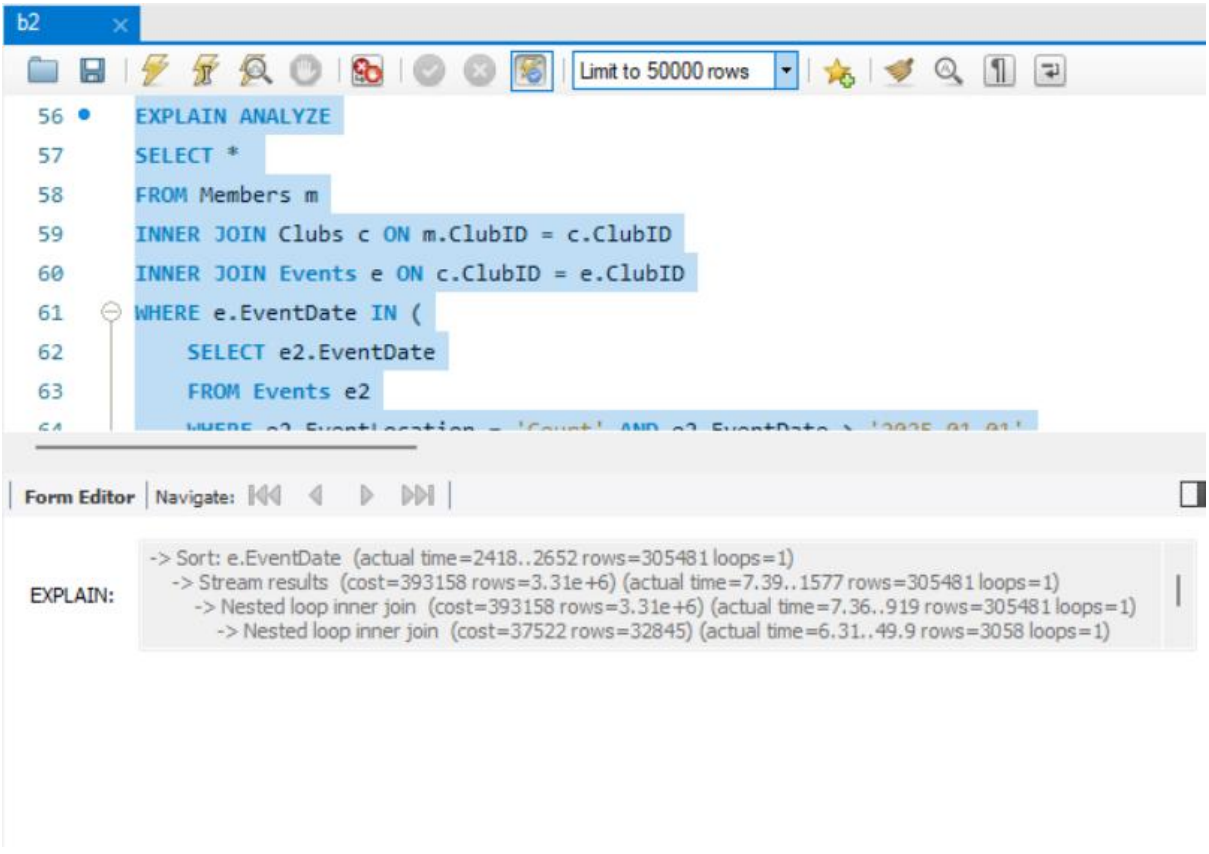
1)

```
EXPLAIN ANALYZE
SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount
FROM Members m
INNER JOIN Clubs c ON m.ClubID = c.ClubID
INNER JOIN Events e ON c.ClubID = e.ClubID
WHERE e.EventDate > '2024-01-01' AND c.ClubType = 'Sports'
GROUP BY c.ClubName
HAVING COUNT(m.MemberID) > 3
ORDER BY MemberCount DESC;
```



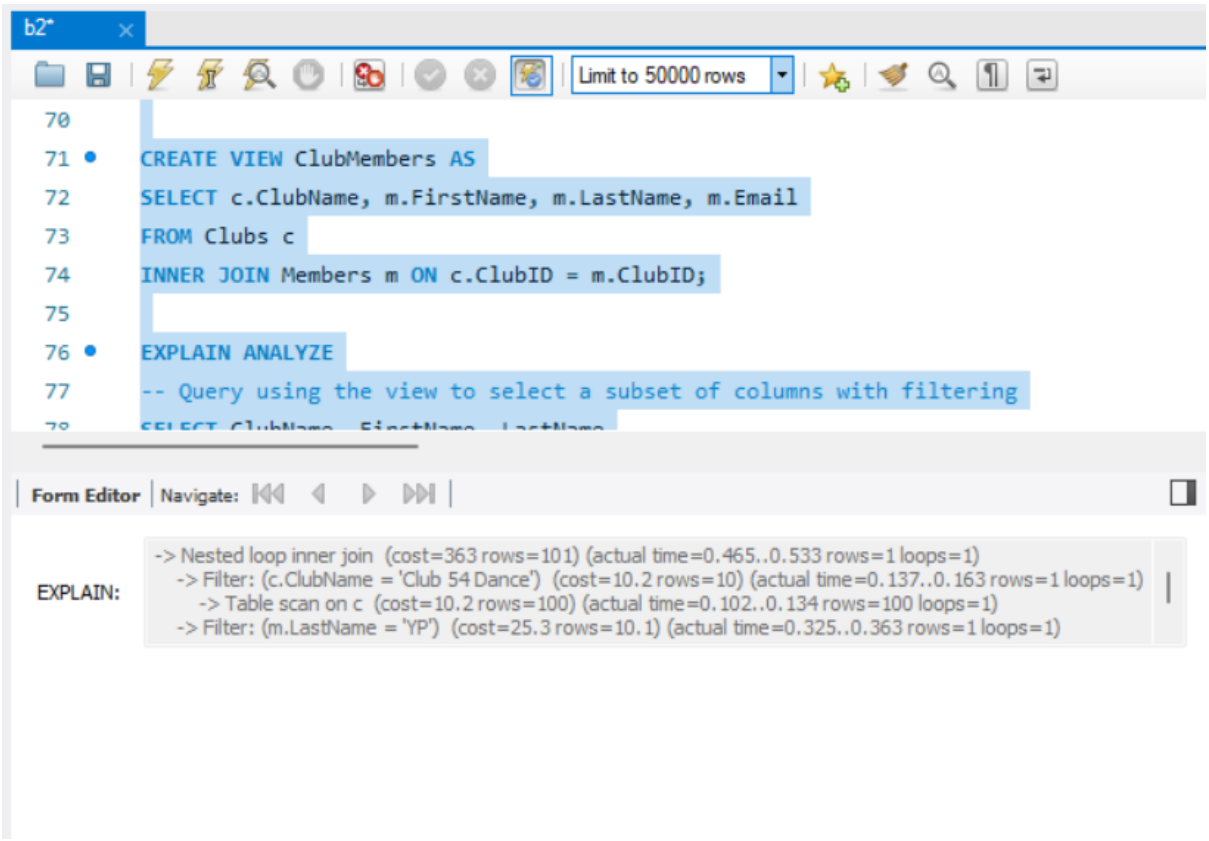
2)

```
EXPLAIN ANALYZE
SELECT *
FROM Members m
INNER JOIN Clubs c ON m.ClubID = c.ClubID
INNER JOIN Events e ON c.ClubID = e.ClubID
WHERE e.EventDate IN (
  SELECT e2.EventDate
  FROM Events e2
  WHERE e2.EventLocation = 'Court' AND e2.EventDate > '2025-01-01'
)
ORDER BY e.EventDate;
```



3)

```
CREATE VIEW ClubMembers AS
SELECT c.ClubName, m.FirstName, m.LastName, m.Email
FROM Clubs c
INNER JOIN Members m ON c.ClubID = m.ClubID;

EXPLAIN ANALYZE
-- Query using the view to select a subset of columns with filtering
SELECT ClubName, FirstName, LastName
FROM ClubMembers
WHERE ClubName = 'Club 54 Dance' AND LastName = 'YP';
```



c)

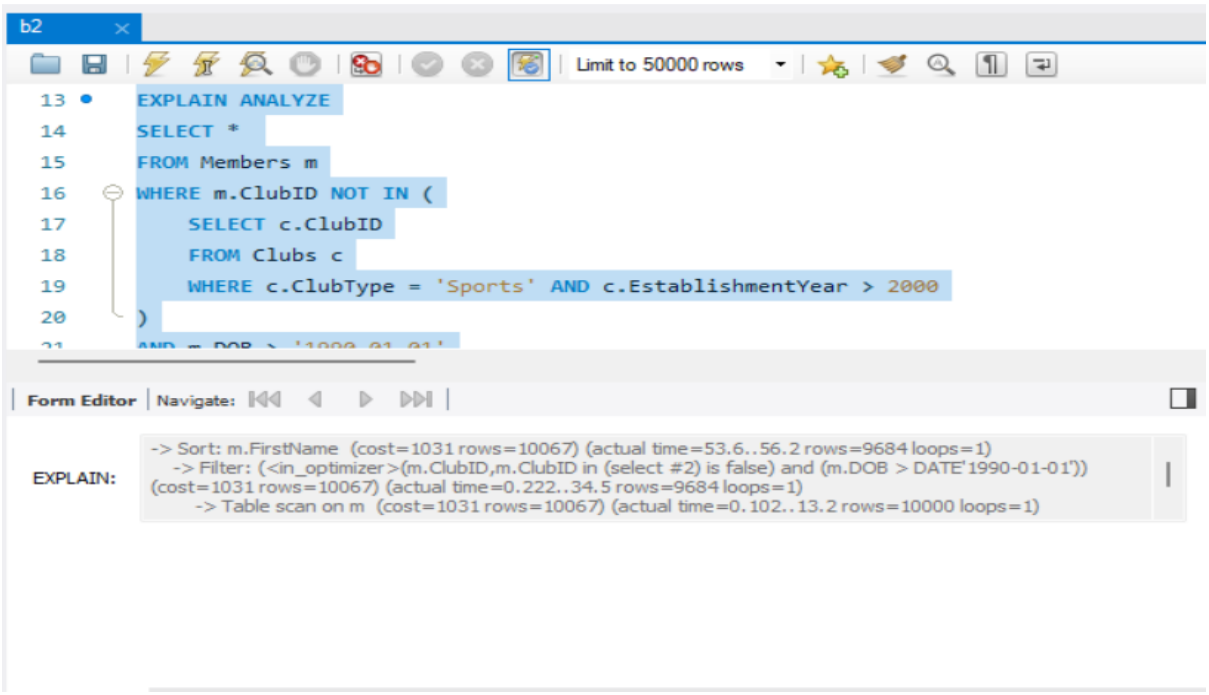
Indexes made to improve performance :

```
CREATE INDEX idx_members_dob_firstname ON Members(DOB, FirstName);

CREATE INDEX idx_clubs_clubtype_establishmentyear ON Clubs(ClubType,
EstablishmentYear);

CREATE INDEX idx_clubs_clubname ON Clubs(ClubName);

CREATE INDEX idx_events_eventdate ON Events(EventDate);
```

1)analyzing below query :

```
SELECT *
FROM Members m
WHERE m.ClubID NOT IN (
  SELECT c.ClubID
  FROM Clubs c
  WHERE c.ClubType = 'Sports' AND c.EstablishmentYear > 2000
)
AND m.DOB > '1990-01-01'
ORDER BY m.FirstName;
```

Old query with indexes :



As we can see the execution time has reduced from 53.6 sec to 32.6 sec

2)analyzing below query :

SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount

FROM Members m

INNER JOIN Clubs c ON m.ClubID = c.ClubID

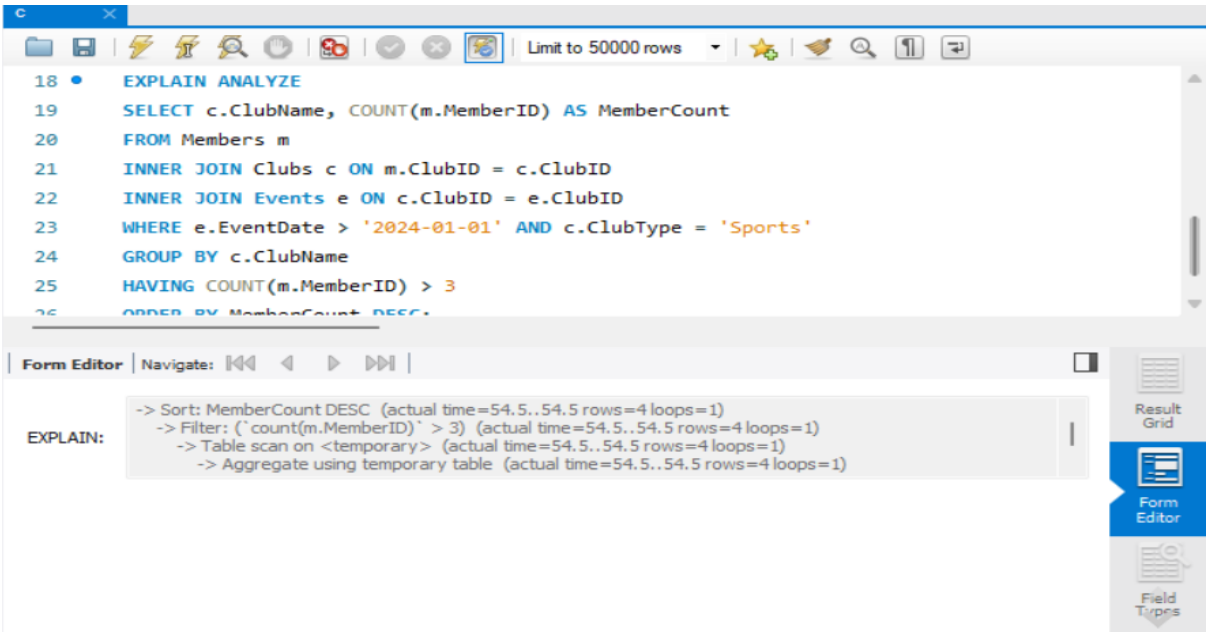
INNER JOIN Events e ON c.ClubID = e.ClubID

WHERE e.EventDate > '2024-01-01' AND c.ClubType = 'Sports'

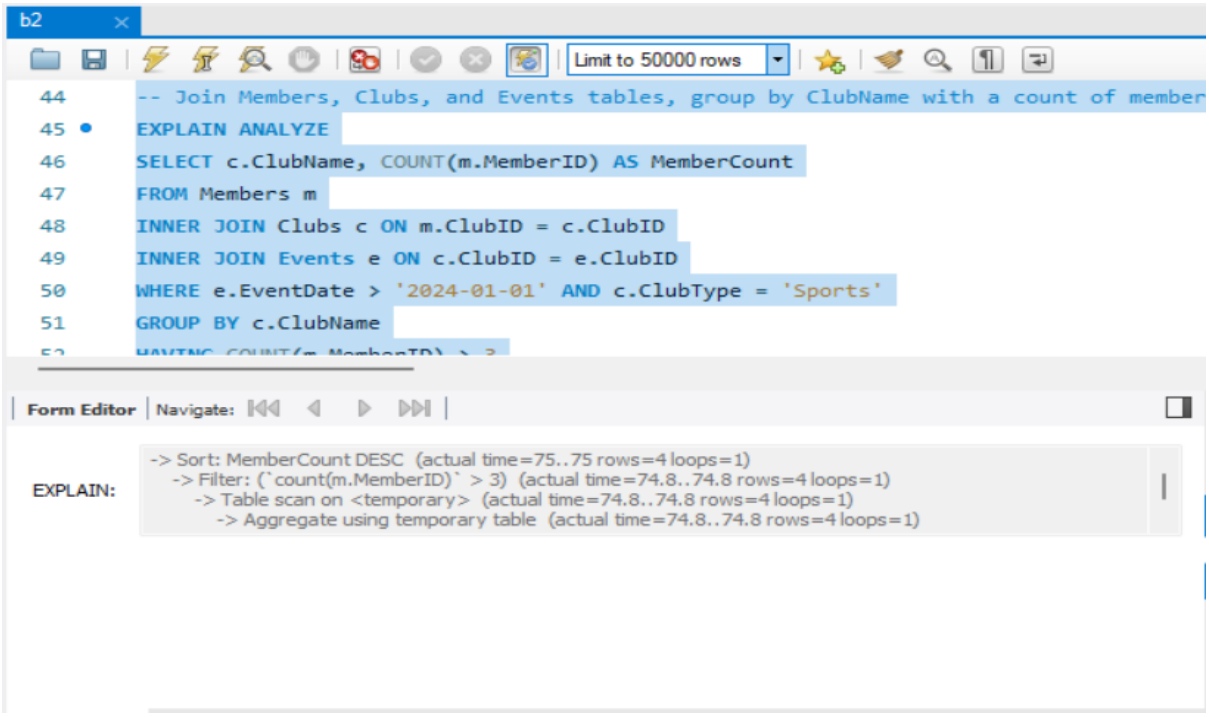
GROUP BY c.ClubName

HAVING COUNT(m.MemberID) > 3

ORDER BY MemberCount DESC;



Old query without indexes:



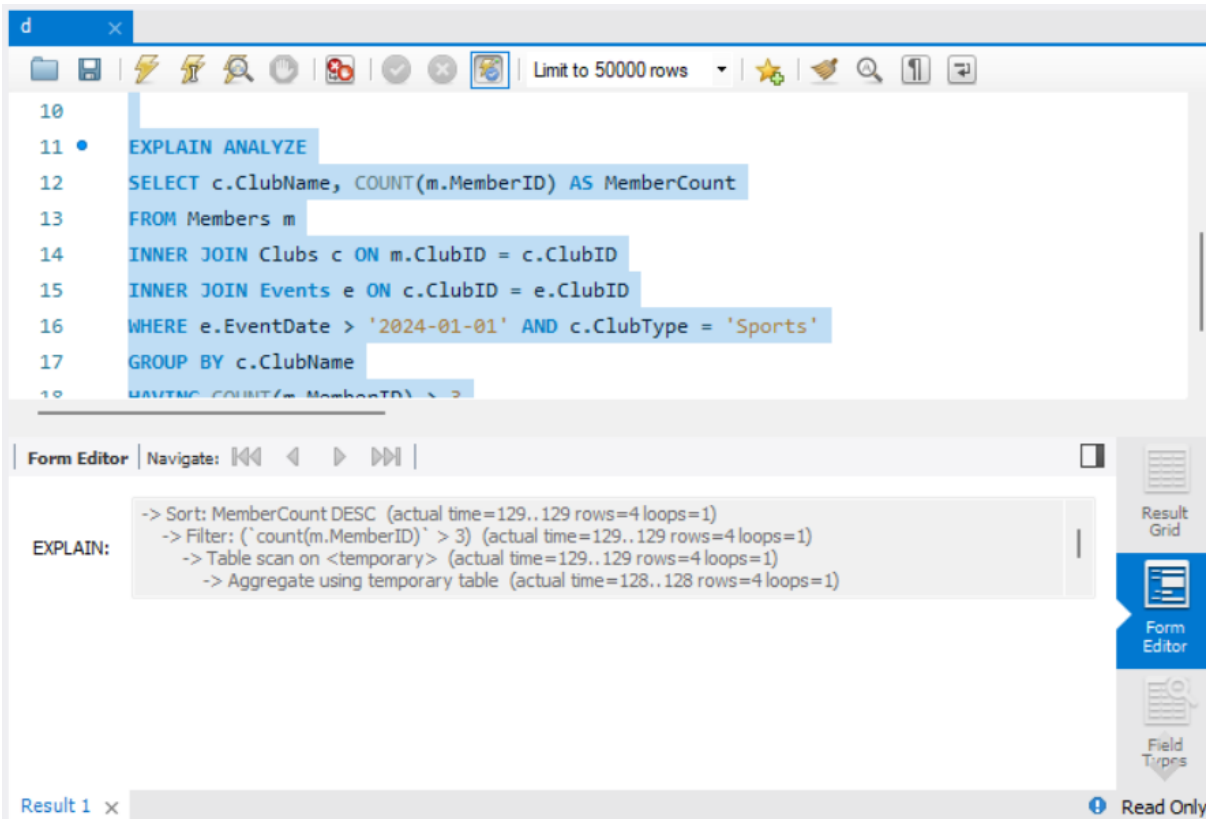
As we can see, execution time has reduced from 75 sec to 54.5 sec

d) The multi join query to optimize :

1)

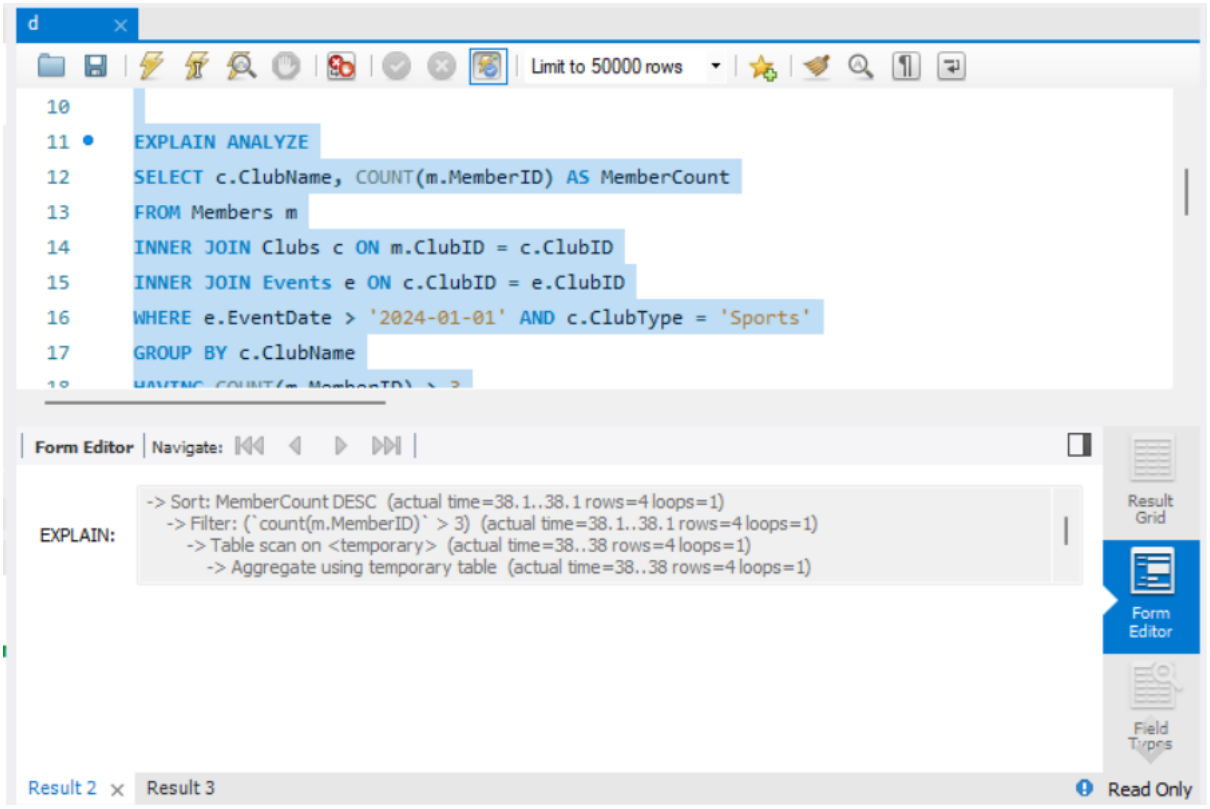
```
SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount
FROM Members m
INNER JOIN Clubs c ON m.ClubID = c.ClubID
INNER JOIN Events e ON c.ClubID = e.ClubID
WHERE e.EventDate > '2024-01-01' AND c.ClubType = 'Sports'
GROUP BY c.ClubName
HAVING COUNT(m.MemberID) > 3
ORDER BY MemberCount DESC;
```

Running EXPLAIN ANALYZE on the above query :



Join order variation 1 : Start with clubs

```
SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount
FROM Clubs c
INNER JOIN Events e ON c.ClubID = e.ClubID
INNER JOIN Members m ON c.ClubID = m.ClubID
WHERE e.EventDate > '2024-01-01' AND c.ClubType = 'Sports'
GROUP BY c.ClubName
HAVING COUNT(m.MemberID) > 3
ORDER BY MemberCount DESC;
```



As we can compare, the execution time after optimization has reduced from 129 sec to 38 sec

Join order variation 2 : Start with events

SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount

FROM Events e

INNER JOIN Clubs c ON e.ClubID = c.ClubID

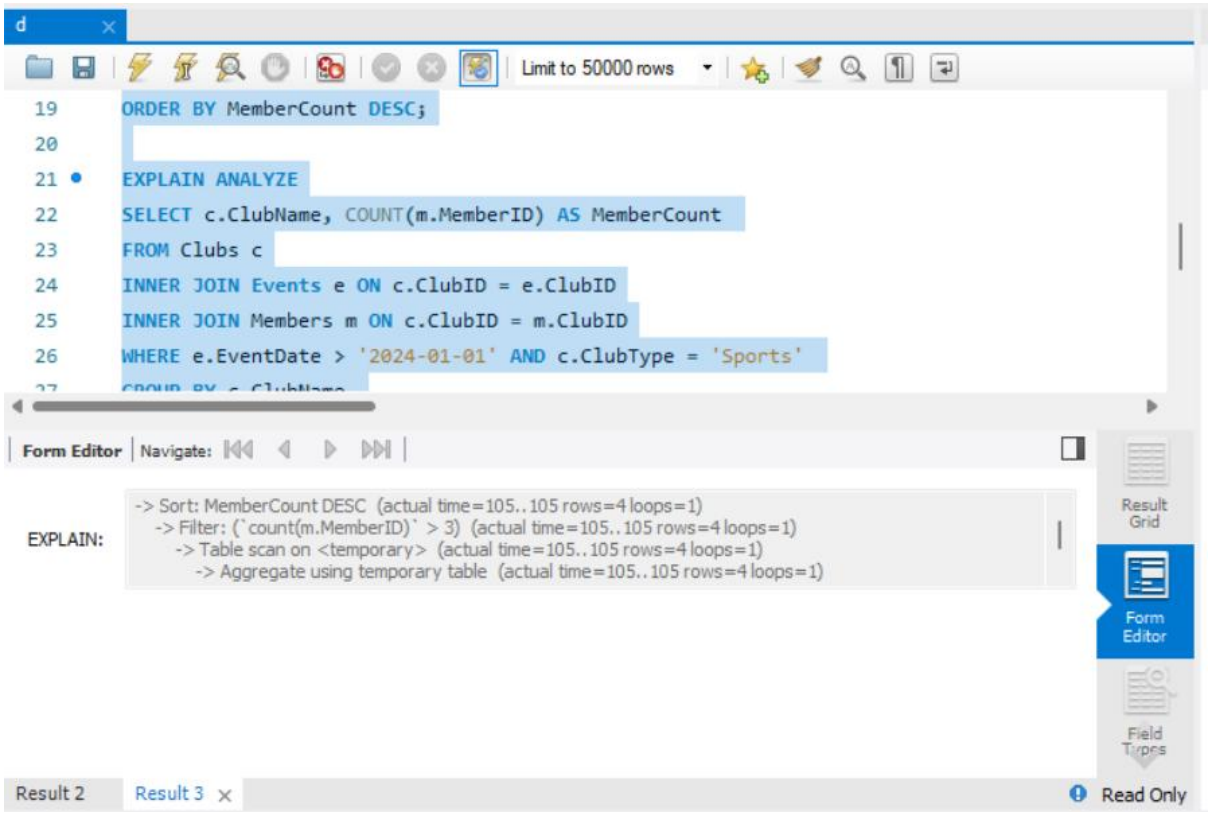
INNER JOIN Members m ON c.ClubID = m.ClubID

WHERE e.EventDate > '2024-01-01' AND c.ClubType = 'Sports'

GROUP BY c.ClubName

HAVING COUNT(m.MemberID) > 3

ORDER BY MemberCount DESC;

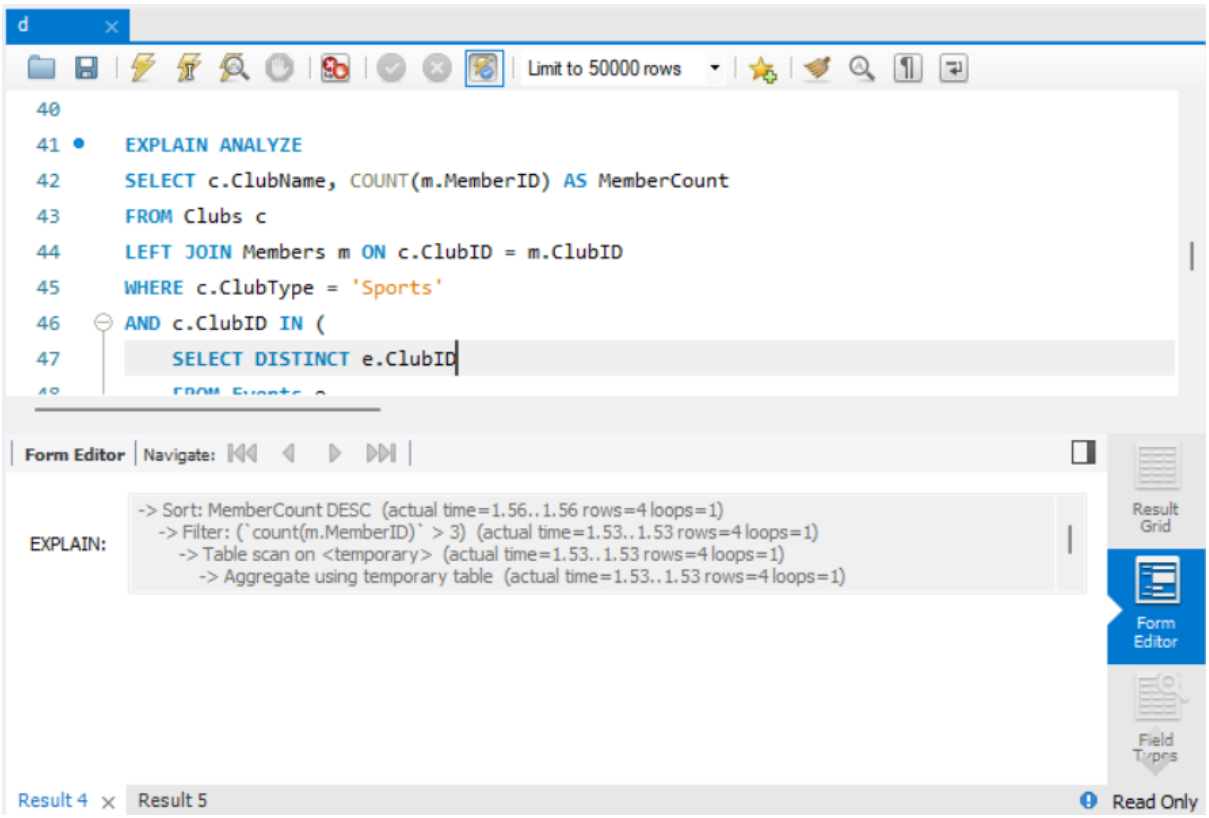


As we can see the execution time has reduced from 129 sec to 105 sec

Incorporating a variety of join types such as outer joins, subqueries, etc., to diversify optimization approaches.

1) Using LEFT JOIN with Subquery Filtering

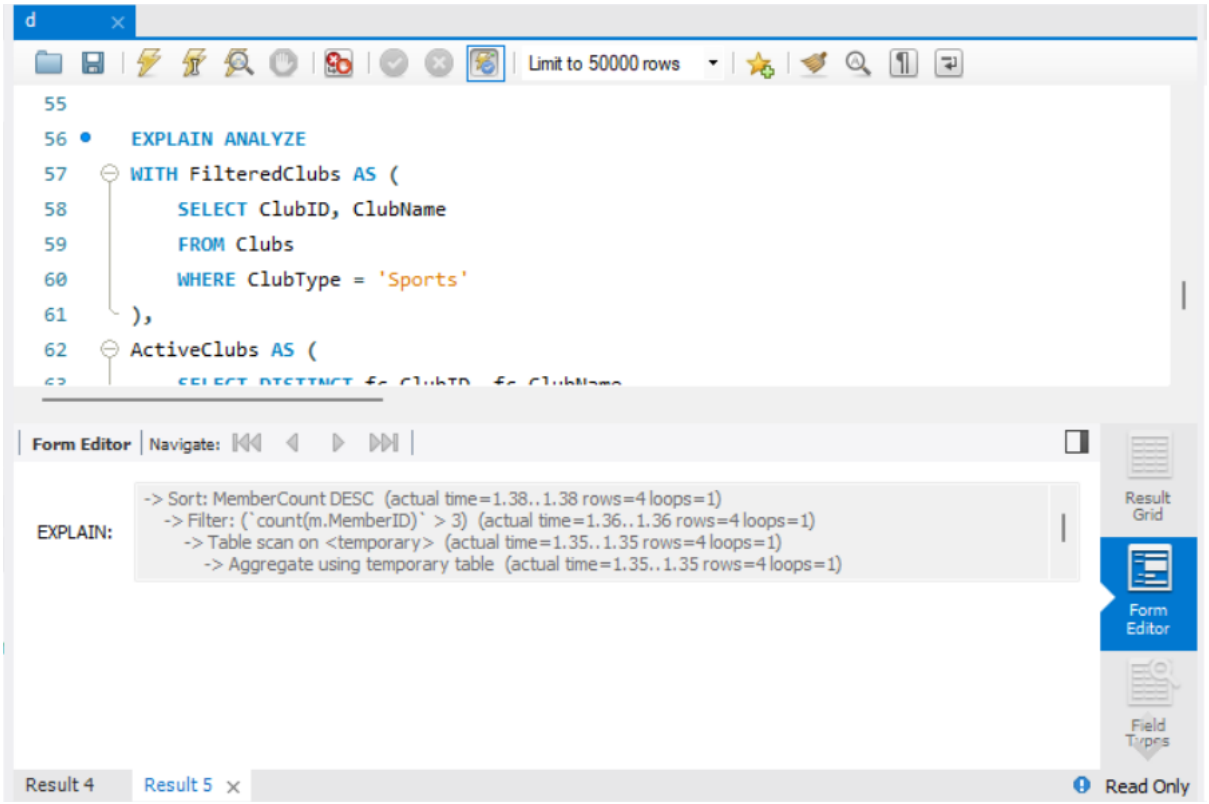
```
SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount
FROM Clubs c
LEFT JOIN Members m ON c.ClubID = m.ClubID
WHERE c.ClubType = 'Sports'
AND c.ClubID IN (
   SELECT DISTINCT e.ClubID
   FROM Events e
   WHERE e.EventDate > '2024-01-01'
)
GROUP BY c.ClubName
HAVING COUNT(m.MemberID) > 3
ORDER BY MemberCount DESC;
```



As we can see the execution time has reduced from 129 sec to 1.56 sec

2) **Using Common Table Expressions with INNER JOIN**

```
WITH FilteredClubs AS (
  SELECT ClubID, ClubName
  FROM Clubs
  WHERE ClubType = 'Sports'
),
ActiveClubs AS (
  SELECT DISTINCT fc.ClubID, fc.ClubName
  FROM FilteredClubs fc
  INNER JOIN Events e ON fc.ClubID = e.ClubID
  WHERE e.EventDate > '2024-01-01'
)
SELECT ac.ClubName, COUNT(m.MemberID) AS MemberCount
FROM ActiveClubs ac
INNER JOIN Members m ON ac.ClubID = m.ClubID
GROUP BY ac.ClubName
HAVING COUNT(m.MemberID) > 3
ORDER BY MemberCount DESC;
```



As we can see the execution time has reduced from 129 to 1.38

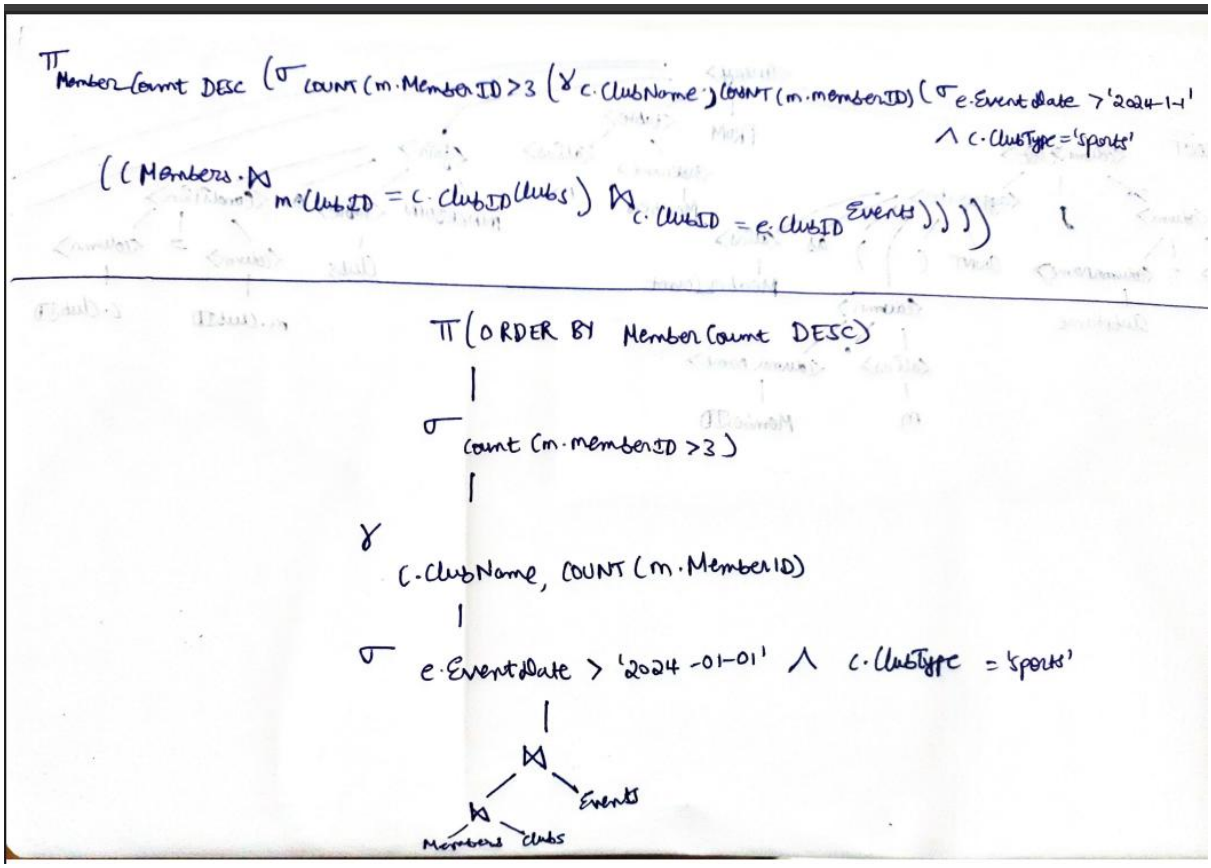
e)

Part 1:

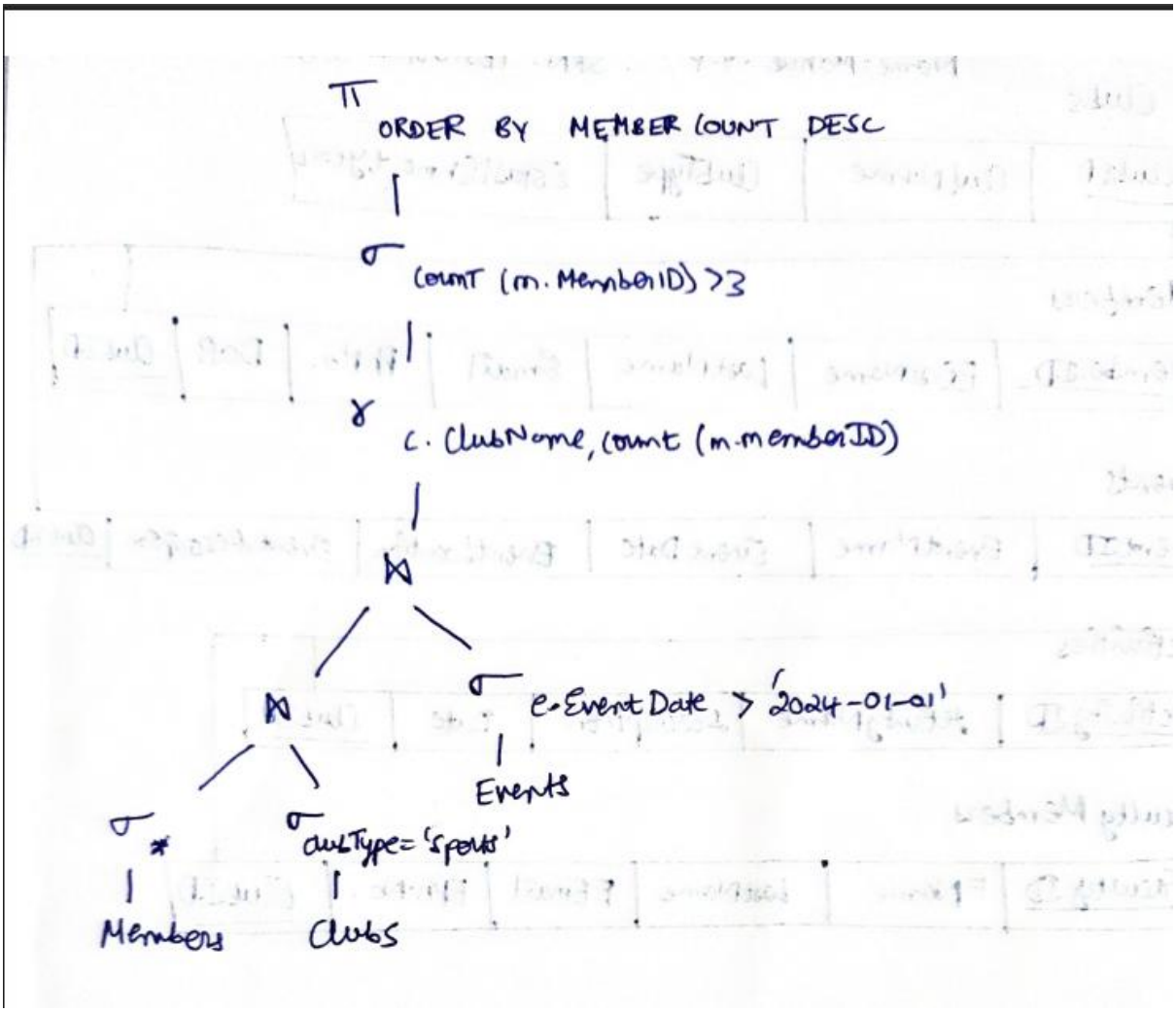
Parse tree:

The long lines drawn on the 2nd screenshot are all coming from <Query>. I didn't have space to draw it on 1 single sheet which is why I combined 2 screenshots in this manner.

Relational Algebric Expression and Initial query tree



Part 2:



Optimization : Pushing Selections Down

- **In the initial query tree, the selection σ (e.EventDate > '2024-01-01' ∧ c.ClubType = 'sports') is applied after the joins.**
- **In the optimized query tree, these selection operations are pushed down and applied earlier on Events and Clubs before the joins.**
- **This reduces the number of tuples participating in the joins, making the query more efficient.**