

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

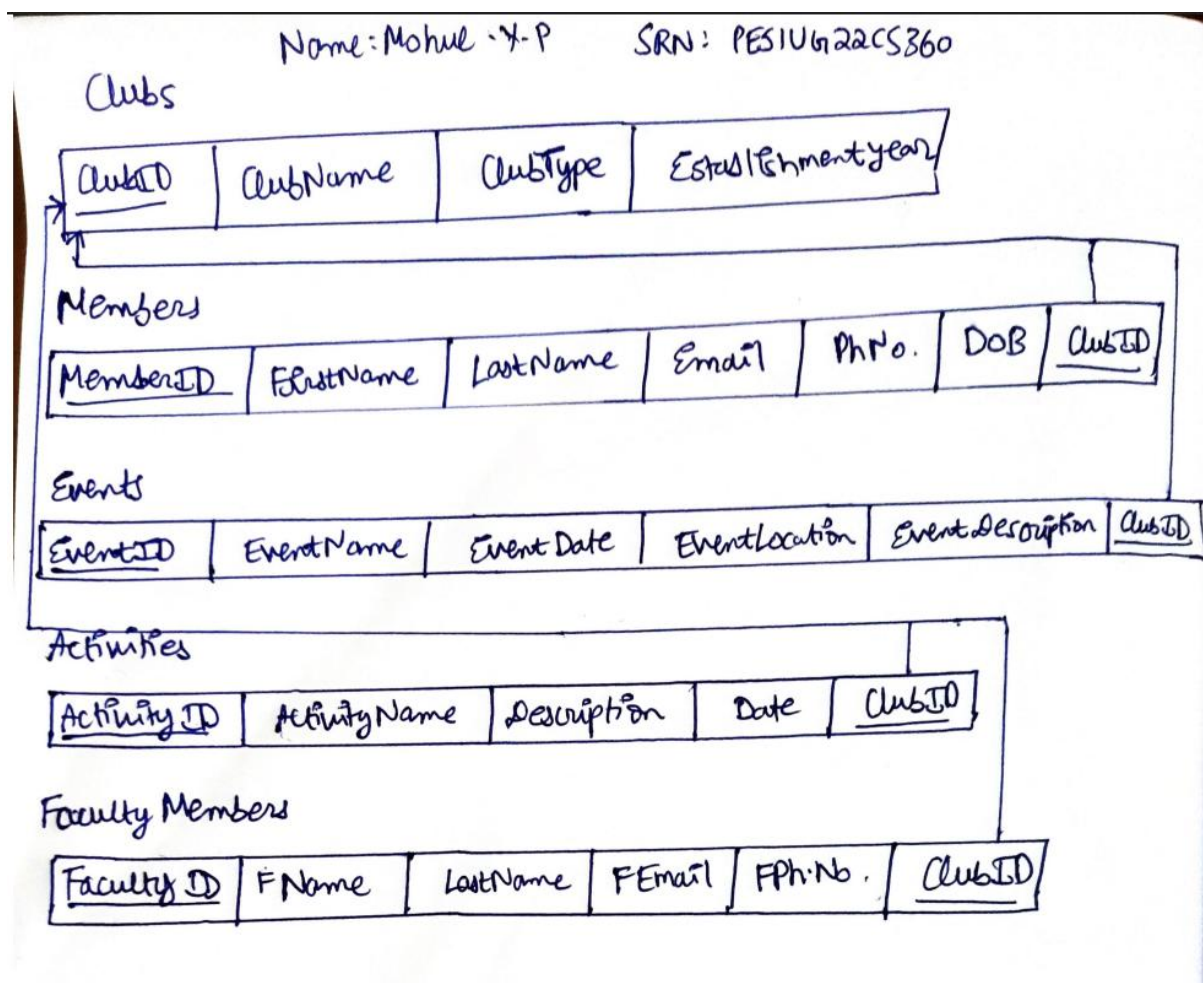
- **ActivityID** (primary key)
- **ActivityName** (required)
- **ActivityDescription**

- 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;

The screenshot shows a database query editor with a toolbar at the top. The SQL editor contains the following queries:

```
1 -- Select all data from Clubs table
2 • SELECT * FROM Clubs;
3
4 -- Count rows in Clubs table
5 • SELECT COUNT(*) FROM Clubs;
6
7 -- Select all data from Members table
8 • SELECT * FROM Members;
```

The result grid displays the data for the 'Clubs' table:

ClubID	ClubName	ClubType	EstablishmentYear
1	Club 1 Sports	Sports	1999
2	Club 2 Music	Music	2021
3	Club 3 Debate	Debate	2022
4	Club 4 Drama	Drama	2018
5	Club 5 Social	Social	2019
6	Club 6 Technical	Technical	2024
7	Club 7 Debate	Debate	2022
8	Club 8 Danc	Club 7 Debate	1992
9	Club 9 Environmental	Environmental	1998
10	Club 10 Art	Art	2002

The bottom of the window shows a tabbed interface with tabs for 'Clubs 1', 'Result 2', 'Members 3', 'Result 4', 'Events 5', 'Result 6', 'Activities 7', 'Result 8', 'Apply', and 'Revert'.

The screenshot shows the same database query editor as above, but with the result grid displaying the output of the 'SELECT COUNT(*) FROM Clubs;' query:

COUNT(*)
100

The bottom of the window shows a tabbed interface with tabs for 'Clubs 1', 'Result 2', 'Members 3', 'Result 4', 'Events 5', 'Result 6', 'Activities 7', 'Result 8', 'Read Only', and 'emb'.

select and count queries on “member” table

SELECT * FROM Members;

SELECT COUNT(*) FROM Members;

The screenshot shows a database client window with a toolbar at the top. The SQL editor contains the following queries:

```
7 -- Select all data from Members table
8 • SELECT * FROM Members;
9
10 -- Count rows in Members table
11 • SELECT COUNT(*) FROM Members;
12
13 -- Select all data from Events table
14 • SELECT * FROM Events;
```

The results pane displays a table with 8 columns: MemberID, FirstName, LastName, Email, PhoneNumber, DOB, and ClubID. The table contains 10 rows of data.

MemberID	FirstName	LastName	Email	PhoneNumber	DOB	ClubID
1	Mohul	YP	mohulyp@gmail.com	555-1234	2004-03-10	54
2	Sheryl	Ortega	sheryl.ortega337@example.com	555-7832	1999-04-16	61
3	Richard	Thayer	richard.thayer619@example.com	555-4063	2006-09-30	42
4	Glenn	Burnett	glenn.burnett824@example.com	555-3646	1997-10-10	83
5	Frances	Duplessis	frances.duplessis766@example.com	555-7918	2001-08-18	44
6	Lionel	Walker	lionel.walker651@example.com	555-6264	1999-09-25	92
7	Nancy	Glidewell	nancy.glidewell277@example.com	555-2256	1995-11-06	33
8	Columbus	Deatherage	columbus.deatherage695@example.com	555-1506	2005-11-19	28
9	Tina	Enger	tina.enger393@example.com	555-6517	1996-02-26	61
10	Kathy	Duressell	kathy.duressell708@example.com	555-4545	2003-05-17	12

The bottom of the window shows a tabbed interface with tabs for Result 2, Members 3, Result 4, Events 5, Result 6, Activities 7, Result 8, Faculty, and Apply. The Results pane is currently displaying the data for the first query.

The screenshot shows the same database client window, but the results pane now displays the output of the second query, which is a single row with the count of rows in the Members table.

COUNT(*)
10000

The bottom of the window shows the same tabbed interface, but the active tab is now Result 4, which displays the count of rows in the Members table.

select and count queries on “events” table

SELECT * FROM Events;

SELECT COUNT(*) FROM Events;

The screenshot shows a database query tool interface. The SQL editor contains the following queries:

```
13 -- Select all data from Events table
14 • SELECT * FROM Events;
15
16 -- Count rows in Events table
17 • SELECT COUNT(*) FROM Events;
18
19 -- Select all data from Activities table
20 • SELECT * FROM Activities;
```

The result grid for the 'Events' table is displayed below the queries:

EventID	EventName	EventDate	EventLocation	EventDescription
1	Weekly Session 2024	2024-11-09	Theater	This is a description for the Weekly Session. It ...
2	Weekly Meetup 2024	2024-07-27	Online	This is a description for the Weekly Meetup. It ...
3	Local Fair 2024	2025-07-29	Cafeteria	This is a description for the Local Fair. It will be ...
4	Annual Competition 2023	2025-12-23	Student Center	This is a description for the Annual Competition...
5	Intra-College Seminar 2024	2026-01-22	Gymnasium	This is a description for the Intra-College Semin...
6	Quarterly Symposium 2023	2025-02-11	Court	This is a description for the Quarterly Symposiu...
7	Summer Competition 2025	2024-05-13	Court	This is a description for the Summer Competition..
8	Fall Gathering 2024	2024-11-10	Recreation Center	This is a description for the Fall Gathering. It will..
9	Fall Camp 2024	2024-03-26	Court	This is a description for the Fall Camp. It will be

The interface also includes a toolbar with icons for file operations, a 'Limit to 50000 rows' dropdown, and a sidebar with 'Result Grid', 'Form Editor', and 'Field Types' options.

The screenshot shows the same database query tool interface, but now displaying the result of the 'COUNT(*)' query. The SQL editor contains the same queries as the previous screenshot:

```
13 -- Select all data from Events table
14 • SELECT * FROM Events;
15
16 -- Count rows in Events table
17 • SELECT COUNT(*) FROM Events;
18
19 -- Select all data from Activities table
20 • SELECT * FROM Activities;
```

The result grid for the 'COUNT(*)' query is displayed below the queries:

COUNT(*)
10000

The interface also includes a toolbar with icons for file operations, a 'Limit to 50000 rows' dropdown, and a sidebar with 'Result Grid', 'Form Editor', and 'Field Types' options.

select and count queries on “activities” table

```
SELECT * FROM Activities;
```

```
SELECT COUNT(*) FROM Activities;
```

b

Limit to 50000 rows

```
-- Select all data from Activities table
20 • SELECT * FROM Activities;
21
22 -- Count rows in Activities table
23 • SELECT COUNT(*) FROM Activities;
24
25 -- Select all data from FacultyMembers table
26 • SELECT * FROM FacultyMembers;
```

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

	ActivityID	ActivityName	ActivityDescription	ActivityDate	ClubID
▶	1	Beginner Rehearsal 98	This is a description for the Beginner Rehearsal ...	2024-12-22	95
	2	Annual Workshop 10	This is a description for the Annual Workshop ac...	2024-10-15	87
	3	Special Mentoring 12	This is a description for the Special Mentoring ac...	2024-12-19	33
	4	Collaborative Design 65	This is a description for the Collaborative Design...	2025-02-01	94
	5	External Mentoring 47	This is a description for the External Mentoring ...	2025-05-20	70
	6	Group Study 56	This is a description for the Group Study activity...	2024-09-25	38
	7	Annual Volunteering 92	This is a description for the Annual Volunteering...	2025-08-15	40
	8	Weekly Meeting 5	This is a description for the Weekly Meeting acti...	2024-11-24	45
	9	Group Volunteering 61	This is a description for the Group Volunteering ...	2024-11-02	73
	10	Weekly Social 18	This is a description for the Weekly Social activit...	2025-05-14	75

Result 2 Members 3 Result 4 Events 5 Result 6 Activities 7 × Result 8 Faculty Apply Revert

Result Grid
Form Editor
Field Types

The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, a search icon, and a 'Limit to 50000 rows' dropdown. The SQL editor contains the following code:

```
19 -- Select all data from Activities table
20 • SELECT * FROM Activities;
21
22 -- Count rows in Activities table
23 • SELECT COUNT(*) FROM Activities;
24
25 -- Select all data from FacultyMembers table
26 • SELECT * FROM FacultyMembers;
27
```

Below the editor is the 'Result Grid' section. It has a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The result grid displays the following data:

	COUNT(*)
▶	10000

On the right side of the IDE, there is a vertical toolbar with icons for 'Result Grid', 'Form Editor', and 'Field Types'.

select and count queries on “faculty members” table

SELECT * FROM FacultyMembers;

SELECT COUNT(*) FROM FacultyMembers;

The screenshot shows a database application window with a SQL query editor and a result grid. The query editor contains the following SQL code:

```
23 • SELECT COUNT(*) FROM Activities;
24
25 -- Select all data from FacultyMembers table
26 • SELECT * FROM FacultyMembers;
27
28 -- Count rows in FacultyMembers table
29 • SELECT COUNT(*) FROM FacultyMembers;
30
```

The result grid displays the results of the queries. The first query result is a single row with the value 10000. The second query result is a table with 7 columns: FacultyID, FirstName, LastName, Email, PhoneNumber, and ClubID. The table contains 10 rows of data.

FacultyID	FirstName	LastName	Email	PhoneNumber	ClubID
1	Randy	Holcomb	prof.randy.holcomb794@university.edu	555-2771	25
2	Kristie	Spurlock	prof.kristie.spurlock460@university.edu	555-7815	6
3	Sara	Wright	prof.sara.wright985@university.edu	555-6320	95
4	Myrtle	Delia	prof.myrtle.delia204@university.edu	555-4692	16
5	Virginia	Goodwin	prof.virginia.goodwin969@university.edu	555-4880	39
6	Michelle	Shelton	prof.michelle.shelton968@university.edu	555-2458	36
7	Benjamin	Hegge	prof.benjamin.hegge581@university.edu	555-4081	2
8	Andrea	Kettler	prof.andrea.kettler499@university.edu	555-1101	88
9	Orville	Eaton	prof.orville.eaton43@university.edu	555-9885	42
10	Johnny	Mason	prof.johnny.mason783@university.edu	555-0770	52

The screenshot shows a database application window with a SQL query editor and a result grid. The query editor contains the following SQL code:

```
23 • SELECT COUNT(*) FROM Activities;
24
25 -- Select all data from FacultyMembers table
26 • SELECT * FROM FacultyMembers;
27
28 -- Count rows in FacultyMembers table
29 • SELECT COUNT(*) FROM FacultyMembers;
30
```

The result grid displays the results of the queries. The first query result is a single row with the value 10000. The second query result is a table with 7 columns: FacultyID, FirstName, LastName, Email, PhoneNumber, and ClubID. The table contains 10 rows of data.

FacultyID	FirstName	LastName	Email	PhoneNumber	ClubID
1	Randy	Holcomb	prof.randy.holcomb794@university.edu	555-2771	25
2	Kristie	Spurlock	prof.kristie.spurlock460@university.edu	555-7815	6
3	Sara	Wright	prof.sara.wright985@university.edu	555-6320	95
4	Myrtle	Delia	prof.myrtle.delia204@university.edu	555-4692	16
5	Virginia	Goodwin	prof.virginia.goodwin969@university.edu	555-4880	39
6	Michelle	Shelton	prof.michelle.shelton968@university.edu	555-2458	36
7	Benjamin	Hegge	prof.benjamin.hegge581@university.edu	555-4081	2
8	Andrea	Kettler	prof.andrea.kettler499@university.edu	555-1101	88
9	Orville	Eaton	prof.orville.eaton43@university.edu	555-9885	42
10	Johnny	Mason	prof.johnny.mason783@university.edu	555-0770	52

INDEX SCAN

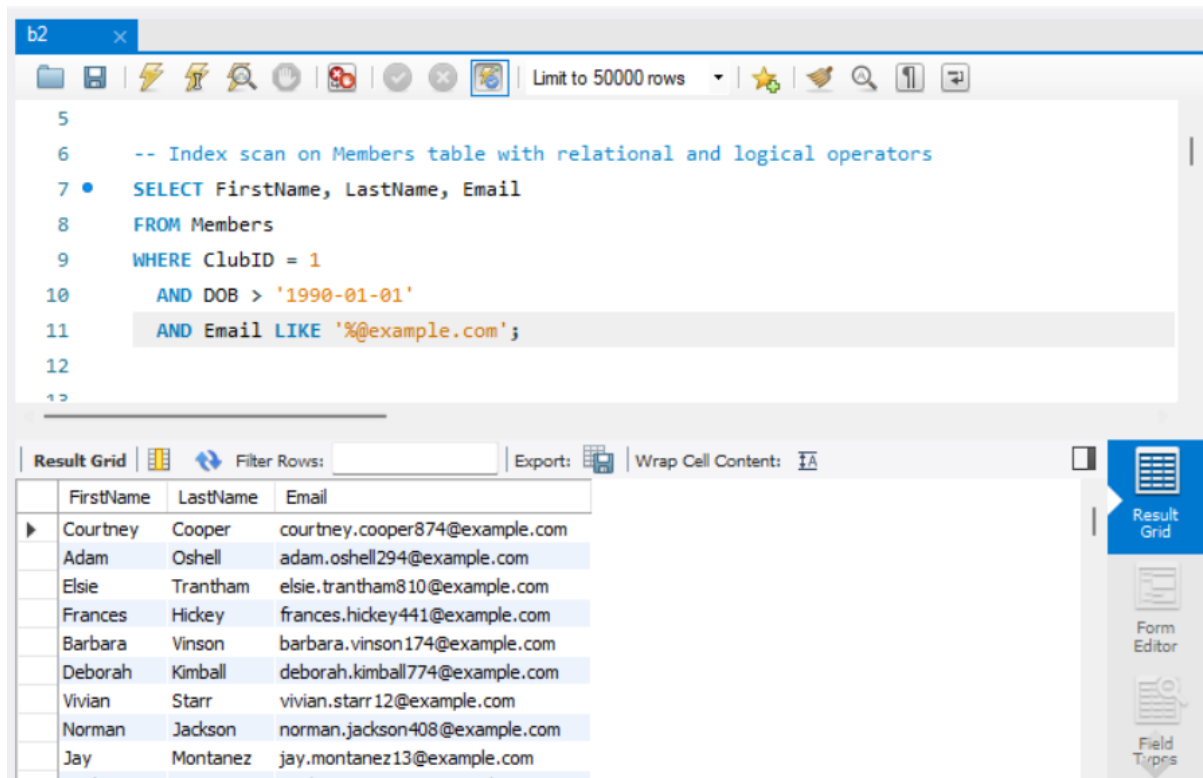
1) **SELECT** FirstName, LastName, Email

FROM Members

WHERE ClubID = 1

AND DOB > '1990-01-01'

AND Email **LIKE** '%@example.com';



The screenshot shows a database query editor window titled 'b2'. The query is as follows:

```
5
6  -- Index scan on Members table with relational and logical operators
7  • SELECT FirstName, LastName, Email
8  FROM Members
9  WHERE ClubID = 1
10     AND DOB > '1990-01-01'
11     AND Email LIKE '@example.com';
12
13
```

Below the query editor, the 'Result Grid' is displayed, showing a table with 3 columns: FirstName, LastName, and Email. The table contains 10 rows of data:

FirstName	LastName	Email
Courtney	Cooper	courtney.cooper874@example.com
Adam	Oshell	adam.oshell294@example.com
Elsie	Trantham	elsie.trantham810@example.com
Frances	Hickey	frances.hickey441@example.com
Barbara	Vinson	barbara.vinson174@example.com
Deborah	Kimball	deborah.kimball774@example.com
Vivian	Starr	vivian.starr12@example.com
Norman	Jackson	norman.jackson408@example.com
Jay	Montanez	jay.montanez13@example.com
Madison	Tran	madison.tran365@example.com

The right sidebar of the application contains buttons for 'Result Grid', 'Form Editor', and 'Field Types'.

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;

The screenshot shows a database query editor with a query window and a result grid. The query is a table scan on the Members table, filtering for members born after 1990 who are not in sports clubs established after 2000, ordered by first name.

```
69 -- table scan on members to find members born after 1990 ordered by first name who are not in sports clubs established after
70 SELECT *
71 FROM Members m
72 WHERE m.ClubID NOT IN (
73     SELECT c.ClubID
74     FROM Clubs c
75     WHERE c.ClubType = 'Sports' AND c.EstablishmentYear > 2000
76 )
77 AND m.DOB > '1990-01-01'
78 ORDER BY m.FirstName;
79
```

The result grid displays the following data:

MemberID	FirstName	LastName	Email	PhoneNumber	DOB	ClubID
8415	Aaron	Myers	aaron.myers869@example.com	555-7708	2003-09-26	26
8785	Aaron	Ransom	aaron.ransom608@example.com	555-9925	1997-01-10	47
8513	Aaron	Yang	aaron.yang962@example.com	555-7990	2006-11-09	65
9103	Aaron	Dalton	aaron.dalton520@example.com	555-7195	2005-05-12	84
8009	Aaron	Sullivan	aaron.sullivan492@example.com	555-5811	1997-08-09	88
7116	Aaron	Goddard	aaron.goddard884@example.com	555-7755	2004-03-09	44
5694	Aaron	Nichols	aaron.nichols164@example.com	555-4599	2005-03-27	4
2502	Aaron	Melvin	aaron.melvin435@example.com	555-5193	1997-10-03	68
618	Aaron	Prieto	aaron.prieto868@example.com	555-2865	2001-11-19	34
7009	Aaron	Huehner	aaron.huehner803@example.com	555-7446	2005-06-28	45

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

SELECT a.ActivityName, COUNT(m.MemberID) AS MemberCount

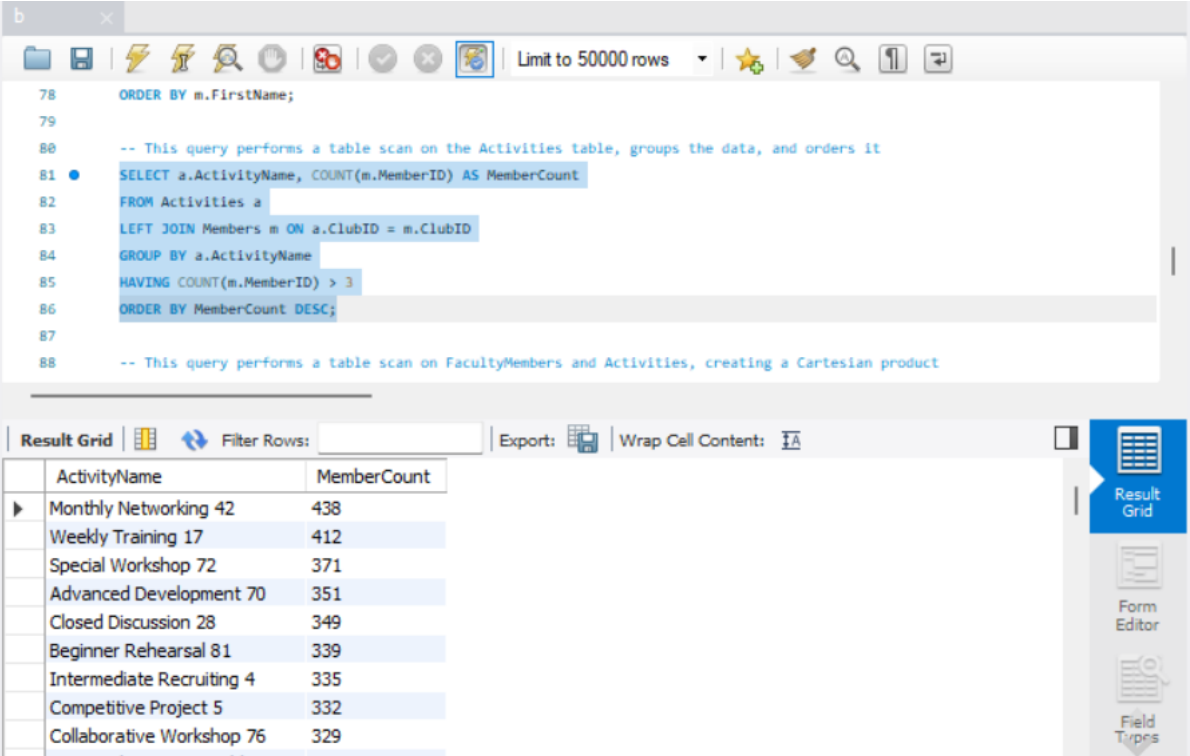
FROM Activities a

LEFT JOIN Members m ON a.ClubID = m.ClubID

GROUP BY a.ActivityName

HAVING COUNT(m.MemberID) > 3

ORDER BY MemberCount DESC;



The screenshot shows a database query editor with a SQL query and its results. The query is as follows:

```
78 ORDER BY m.FirstName;
79
80 -- This query performs a table scan on the Activities table, groups the data, and orders it
81 SELECT a.ActivityName, COUNT(m.MemberID) AS MemberCount
82 FROM Activities a
83 LEFT JOIN Members m ON a.ClubID = m.ClubID
84 GROUP BY a.ActivityName
85 HAVING COUNT(m.MemberID) > 3
86 ORDER BY MemberCount DESC;
87
88 -- This query performs a table scan on FacultyMembers and Activities, creating a Cartesian product
```

The results are displayed in a grid with the following data:

ActivityName	MemberCount
Monthly Networking 42	438
Weekly Training 17	412
Special Workshop 72	371
Advanced Development 70	351
Closed Discussion 28	349
Beginner Rehearsal 81	339
Intermediate Recruiting 4	335
Competitive Project 5	332
Collaborative Workshop 76	329
Intermediate Team Building 4	326

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;

The screenshot shows a database query editor window. The query text is as follows:

```
87
88 -- This query performs a table scan on FacultyMembers and Activities, creating
89 • SELECT f.FirstName, f.LastName, a.ActivityName
90 FROM FacultyMembers f
91 CROSS JOIN Activities a;
92
93
94
```

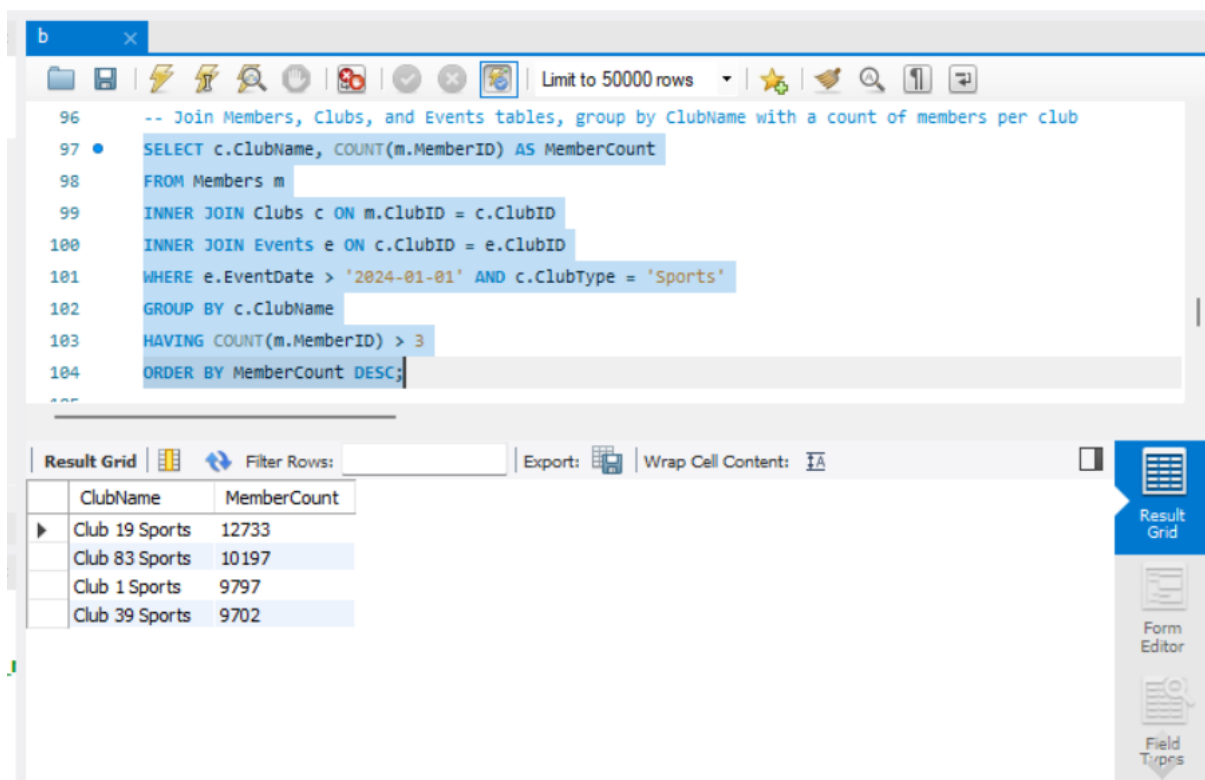
Below the query editor, the results are displayed in a table with the following data:

ActivityName	MemberCount
Monthly Networking 42	438
Weekly Training 17	412
Special Workshop 72	371
Advanced Development 70	351
Closed Discussion 28	349
Beginner Rehearsal 81	339
Intermediate Recruiting 4	335
Competitive Project 5	332
Collaborative Workshop 76	329
Intermediate Team Building 1	328

The interface includes a toolbar at the top with icons for file operations, a 'Limit to 50000 rows' dropdown, and a 'Result Grid' button on the right. The bottom of the window shows a 'Filter Rows' field and an 'Export' button.

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;
```



The screenshot shows a database query editor with a SQL query and its results. The query is as follows:

```
-- Join Members, Clubs, and Events tables, group by ClubName with a count of members per club
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;
```

The results are displayed in a grid with the following data:

ClubName	MemberCount
Club 19 Sports	12733
Club 83 Sports	10197
Club 1 Sports	9797
Club 39 Sports	9702

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 (

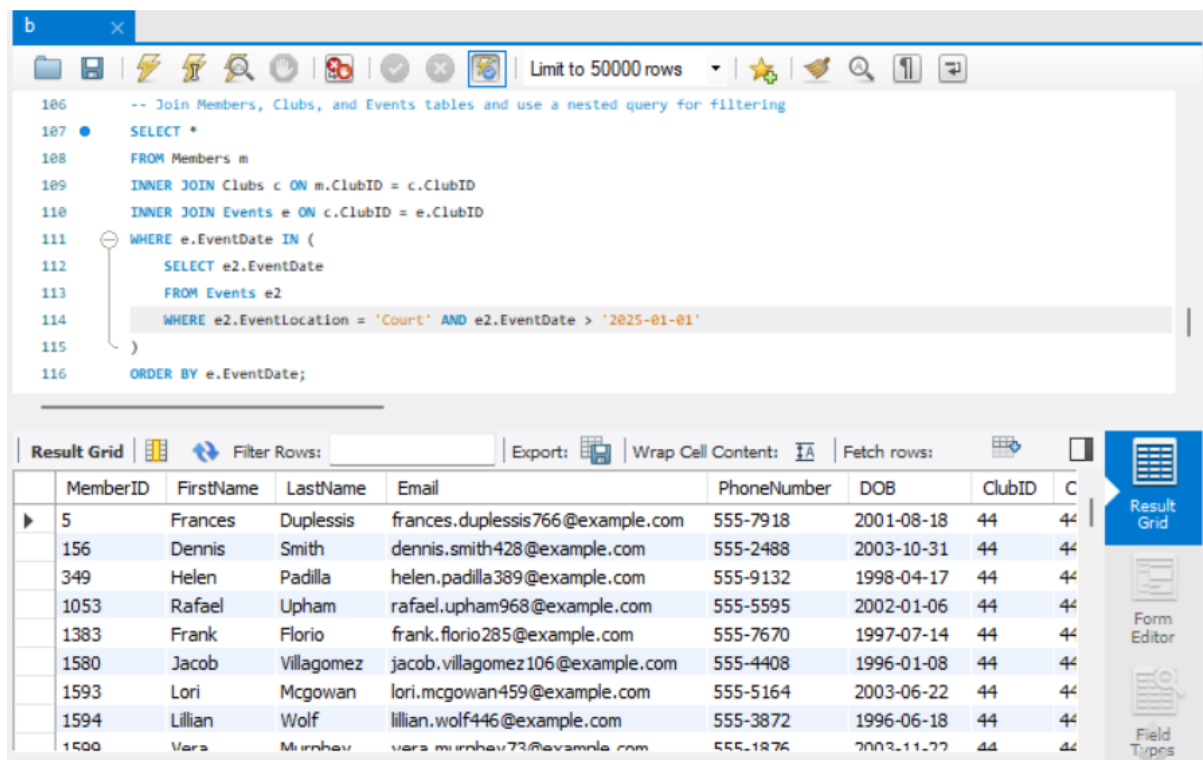
SELECT e2.EventDate

FROM Events e2

WHERE e2.EventLocation = 'Hall A' AND e2.EventDate > '2025-01-01'

)

ORDER BY e.EventDate;



The screenshot shows a database query editor with a SQL query and its results. The query is as follows:

```
106 -- Join Members, Clubs, and Events tables and use a nested query for filtering
107 SELECT *
108 FROM Members m
109 INNER JOIN Clubs c ON m.ClubID = c.ClubID
110 INNER JOIN Events e ON c.ClubID = e.ClubID
111 WHERE e.EventDate IN (
112     SELECT e2.EventDate
113     FROM Events e2
114     WHERE e2.EventLocation = 'Court' AND e2.EventDate > '2025-01-01'
115 )
116 ORDER BY e.EventDate;
```

The results are displayed in a grid with the following columns: MemberID, FirstName, LastName, Email, PhoneNumber, DOB, ClubID, and C. The grid shows 10 rows of data.

MemberID	FirstName	LastName	Email	PhoneNumber	DOB	ClubID	C
5	Frances	Duplessis	frances.duplessis766@example.com	555-7918	2001-08-18	44	44
156	Dennis	Smith	dennis.smith428@example.com	555-2488	2003-10-31	44	44
349	Helen	Padilla	helen.padilla389@example.com	555-9132	1998-04-17	44	44
1053	Rafael	Upham	rafael.upham968@example.com	555-5595	2002-01-06	44	44
1383	Frank	Florio	frank.florio285@example.com	555-7670	1997-07-14	44	44
1580	Jacob	Villagomez	jacob.villagomez106@example.com	555-4408	1996-01-08	44	44
1593	Lori	Mcgowan	lori.mcgowan459@example.com	555-5164	2003-06-22	44	44
1594	Lillian	Wolf	lillian.wolf446@example.com	555-3872	1996-06-18	44	44
1500	Vera	Murney	vera.murney73@example.com	555-1876	2003-11-22	44	44

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';

The screenshot shows a database management tool interface. The top section displays a SQL script with the following queries:

```
119
120 CREATE VIEW ClubMembers AS
121 SELECT c.ClubName, m.FirstName, m.LastName, m.Email
122 FROM Clubs c
123 INNER JOIN Members m ON c.ClubID = m.ClubID;
124
125 -- Query using the view to select a subset of columns with filtering
126 SELECT ClubName, FirstName, LastName
127 FROM ClubMembers
128 WHERE ClubName = 'Club 54 Dance' AND LastName = 'YP';
129
```

The bottom section shows the results of the second query in a table format:

ClubName	FirstName	LastName
Club 54 Dance	Mohul	YP

The interface also includes a toolbar with various icons, a 'Limit to 50000 rows' dropdown, and a sidebar with options like 'Result Grid', 'Form Editor', and 'Field Types'.

EXPLAIN ANALYZE on each of the queries mentioned above :

INDEX SCAN

1) EXPLAIN ANALYZE

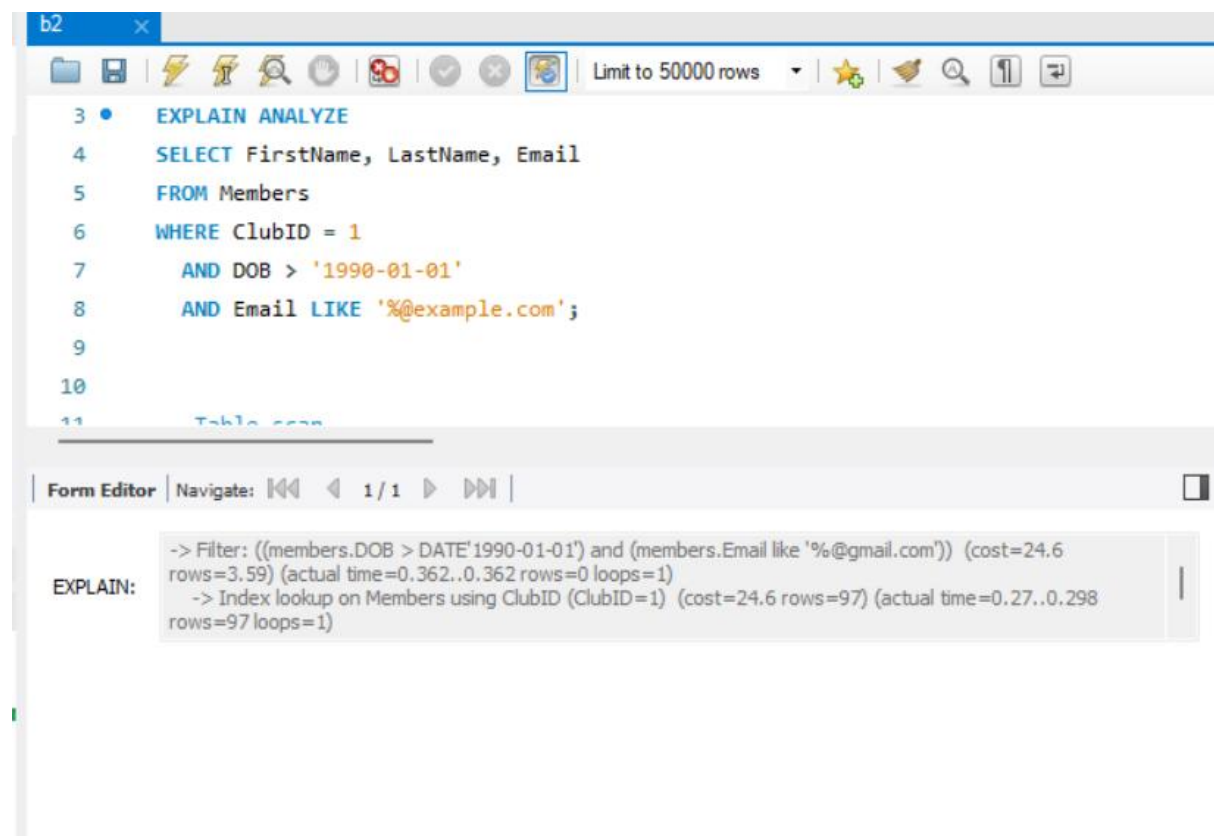
SELECT FirstName, LastName, Email

FROM Members

WHERE ClubID = 1

AND DOB > '1990-01-01'

AND Email LIKE '%@example.com';



The screenshot shows a database IDE window titled 'b2'. The SQL editor contains the following query:

```
3 • EXPLAIN ANALYZE
4 SELECT FirstName, LastName, Email
5 FROM Members
6 WHERE ClubID = 1
7     AND DOB > '1990-01-01'
8     AND Email LIKE '%@example.com';
9
10
11
```

Below the editor, the 'Form Editor' tab is active, displaying the 'EXPLAIN:' results:

```
-> Filter: ((members.DOB > DATE'1990-01-01') and (members.Email like '%@gmail.com')) (cost=24.6
rows=3.59) (actual time=0.362..0.362 rows=0 loops=1)
-> Index lookup on Members using ClubID (ClubID=1) (cost=24.6 rows=97) (actual time=0.27..0.298
rows=97 loops=1)
```

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;

The screenshot shows a database IDE window titled 'b2'. The SQL editor contains the following query:

```
13 EXPLAIN ANALYZE
14 SELECT *
15 FROM Members m
16 WHERE m.ClubID NOT IN (
17     SELECT c.ClubID
18     FROM Clubs c
19     WHERE c.ClubType = 'Sports' AND c.EstablishmentYear > 2000
20 )
21 AND m.DOB > '1990-01-01'
```

The IDE interface includes a toolbar with icons for file operations, a 'Limit to 50000 rows' dropdown, and a 'Form Editor' tab. Below the editor, the 'EXPLAIN:' section displays the execution plan:

```
EXPLAIN:
-> Sort: m.FirstName (cost=1031 rows=10067) (actual time=53.6..56.2 rows=9684 loops=1)
-> Filter: (<in_optimizer>(m.ClubID,m.ClubID in (select #2) is false) and (m.DOB > DATE'1990-01-01'))
(cost=1031 rows=10067) (actual time=0.222..34.5 rows=9684 loops=1)
-> Table scan on m (cost=1031 rows=10067) (actual time=0.102..13.2 rows=10000 loops=1)
```

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;

The screenshot shows a SQL IDE window with a query editor and an execution plan viewer. The query editor contains the following SQL code:

```
25 • EXPLAIN ANALYZE
26 SELECT a.ActivityName, COUNT(m.MemberID) AS MemberCount
27 FROM Activities a
28 LEFT JOIN Members m ON a.ClubID = m.ClubID
29 GROUP BY a.ActivityName
30 HAVING COUNT(m.MemberID) > 3
31 ORDER BY MemberCount DESC;
32
33
```

The execution plan viewer shows the following details:

Form Editor | Navigate: ⏮ ⏪ ⏩ ⏭

EXPLAIN:

- > Sort: MemberCount DESC (actual time=19525..19527 rows=8917 loops=1)
- > Filter: ('count(m.MemberID)' > 3) (actual time=19501..19510 rows=8917 loops=1)
- > Table scan on <temporary> (actual time=19501..19508 rows=8917 loops=1)
- > Aggregate using temporary table (actual time=19501..19501 rows=8917 loops=1)

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;

The screenshot shows a database IDE window titled 'b2'. The SQL editor contains the following query:

```
44 -- Join Members, Clubs, and Events tables, group by ClubName with a count of member
45 EXPLAIN ANALYZE
46 SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount
47 FROM Members m
48 INNER JOIN Clubs c ON m.ClubID = c.ClubID
49 INNER JOIN Events e ON c.ClubID = e.ClubID
50 WHERE e.EventDate > '2024-01-01' AND c.ClubType = 'Sports'
51 GROUP BY c.ClubName
52 HAVING COUNT(m.MemberID) > 3
```

The IDE interface includes a toolbar with icons for file operations, a 'Limit to 50000 rows' dropdown, and a 'Form Editor' tab. Below the SQL editor, the 'EXPLAIN ANALYZE' results are displayed:

EXPLAIN:

- > Sort: MemberCount DESC (actual time=75..75 rows=4 loops=1)
- > Filter: ('count(m.MemberID)' > 3) (actual time=74.8..74.8 rows=4 loops=1)
- > Table scan on <temporary> (actual time=74.8..74.8 rows=4 loops=1)
- > Aggregate using temporary table (actual time=74.8..74.8 rows=4 loops=1)

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 (

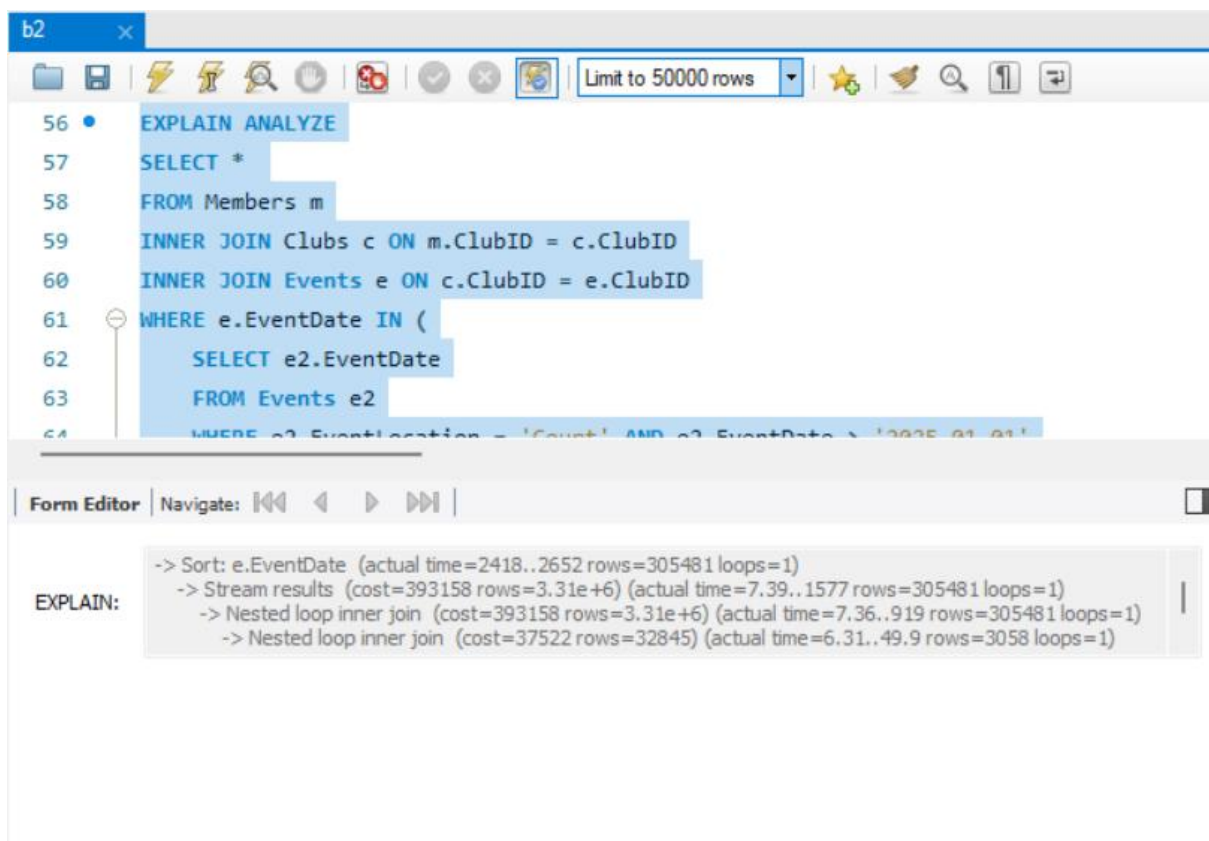
SELECT e2.EventDate

FROM Events e2

WHERE e2.EventLocation = 'Court' AND e2.EventDate > '2025-01-01'

)

ORDER BY e.EventDate;



The screenshot shows a database query editor window titled 'b2'. The query is as follows:

```
56 EXPLAIN ANALYZE
57 SELECT *
58 FROM Members m
59 INNER JOIN Clubs c ON m.ClubID = c.ClubID
60 INNER JOIN Events e ON c.ClubID = e.ClubID
61 WHERE e.EventDate IN (
62     SELECT e2.EventDate
63     FROM Events e2
64     WHERE e2.EventLocation = 'Court' AND e2.EventDate > '2025-01-01'
65 )
66 ORDER BY e.EventDate;
```

The execution plan (EXPLAIN) is displayed below the query:

```
EXPLAIN:
-> Sort: e.EventDate (actual time=2418..2652 rows=305481 loops=1)
-> Stream results (cost=393158 rows=3.31e+6) (actual time=7.39..1577 rows=305481 loops=1)
-> Nested loop inner join (cost=393158 rows=3.31e+6) (actual time=7.36..919 rows=305481 loops=1)
-> Nested loop inner join (cost=37522 rows=32845) (actual time=6.31..49.9 rows=3058 loops=1)
```

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';

The screenshot shows a database IDE window titled 'b2*' with a toolbar at the top. The SQL editor contains the following code:

```
70  
71 • CREATE VIEW ClubMembers AS  
72   SELECT c.ClubName, m.FirstName, m.LastName, m.Email  
73   FROM Clubs c  
74   INNER JOIN Members m ON c.ClubID = m.ClubID;  
75  
76 • EXPLAIN ANALYZE  
77   -- Query using the view to select a subset of columns with filtering  
78   SELECT ClubName, FirstName, LastName
```

Below the editor is a 'Form Editor' section with a 'Navigate' bar. The 'EXPLAIN:' section displays the execution plan:

```
-> Nested loop inner join (cost=363 rows=101) (actual time=0.465..0.533 rows=1 loops=1)  
  -> Filter: (c.ClubName = 'Club 54 Dance') (cost=10.2 rows=10) (actual time=0.137..0.163 rows=1 loops=1)  
    -> Table scan on c (cost=10.2 rows=100) (actual time=0.102..0.134 rows=100 loops=1)  
      -> Filter: (m.LastName = 'YP') (cost=25.3 rows=10.1) (actual time=0.325..0.363 rows=1 loops=1)
```

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;

The screenshot shows a SQL IDE window with a query editor and an execution plan. The query is as follows:

```

7 • EXPLAIN ANALYZE
8 SELECT *
9 FROM Members m
10 WHERE m.ClubID NOT IN (
11     SELECT c.ClubID
12     FROM Clubs c
13     WHERE c.ClubType = 'Sports' AND c.EstablishmentYear > 2000
14 )
15 AND m.DOB > '1990-01-01'

```

The execution plan (EXPLAIN) is shown below the query:

```

-> Sort: m.FirstName (cost=1031 rows=10067) (actual time=32.6..35.4 rows=9684 loops=1)
-> Filter: (<in_optimizer>(m.ClubID,m.ClubID in (select #2) is false) and (m.DOB > DATE'1990-01-01'))
(cost=1031 rows=10067) (actual time=0.646..15.8 rows=9684 loops=1)
-> Table scan on m (cost=1031 rows=10067) (actual time=0.241..6.11 rows=10000 loops=1)

```

Old query with indexes :

The screenshot shows a SQL IDE window with a query and its execution plan. The query is as follows:

```

13 • EXPLAIN ANALYZE
14 SELECT *
15 FROM Members m
16 WHERE m.ClubID NOT IN (
17     SELECT c.ClubID
18     FROM Clubs c
19     WHERE c.ClubType = 'Sports' AND c.EstablishmentYear > 2000
20 )
21 AND m.DOB > '1990-01-01'

```

The execution plan (EXPLAIN) is shown below the query:

```

-> Sort: m.FirstName (cost=1031 rows=10067) (actual time=53.6..56.2 rows=9684 loops=1)
-> Filter: (<in_optimizer>(m.ClubID,m.ClubID in (select #2) is false) and (m.DOB > DATE'1990-01-01'))
(cost=1031 rows=10067) (actual time=0.222..34.5 rows=9684 loops=1)
-> Table scan on m (cost=1031 rows=10067) (actual time=0.102..13.2 rows=10000 loops=1)

```

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;

The screenshot shows a SQL IDE window with a query editor and an execution plan. The query is as follows:

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

The execution plan is displayed in the 'Form Editor' tab. It shows the following steps:

- > Sort: MemberCount DESC (actual time=54.5..54.5 rows=4 loops=1)
- > Filter: ('count(m.MemberID)' > 3) (actual time=54.5..54.5 rows=4 loops=1)
- > Table scan on <temporary> (actual time=54.5..54.5 rows=4 loops=1)
- > Aggregate using temporary table (actual time=54.5..54.5 rows=4 loops=1)

The IDE interface includes a toolbar at the top with various icons, a 'Limit to 50000 rows' dropdown, and a sidebar on the right with buttons for 'Result Grid', 'Form Editor', and 'Field Types'.

Old query without indexes:

The screenshot shows a SQL IDE window with a query editor and an execution plan. The query is as follows:

```
44 -- Join Members, Clubs, and Events tables, group by ClubName with a count of member
45 • EXPLAIN ANALYZE
46 SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount
47 FROM Members m
48 INNER JOIN Clubs c ON m.ClubID = c.ClubID
49 INNER JOIN Events e ON c.ClubID = e.ClubID
50 WHERE e.EventDate > '2024-01-01' AND c.ClubType = 'Sports'
51 GROUP BY c.ClubName
52 HAVING COUNT(m.MemberID) > 3
```

The execution plan is displayed in the 'Form Editor' tab. It shows the following steps:

- > Sort: MemberCount DESC (actual time=75..75 rows=4 loops=1)
- > Filter: ('count(m.MemberID)' > 3) (actual time=74.8..74.8 rows=4 loops=1)
- > Table scan on <temporary> (actual time=74.8..74.8 rows=4 loops=1)
- > Aggregate using temporary table (actual time=74.8..74.8 rows=4 loops=1)

The IDE interface includes a toolbar at the top with various icons, a 'Limit to 50000 rows' dropdown, and a sidebar on the right with buttons for 'Result Grid', 'Form Editor', and 'Field Types'.

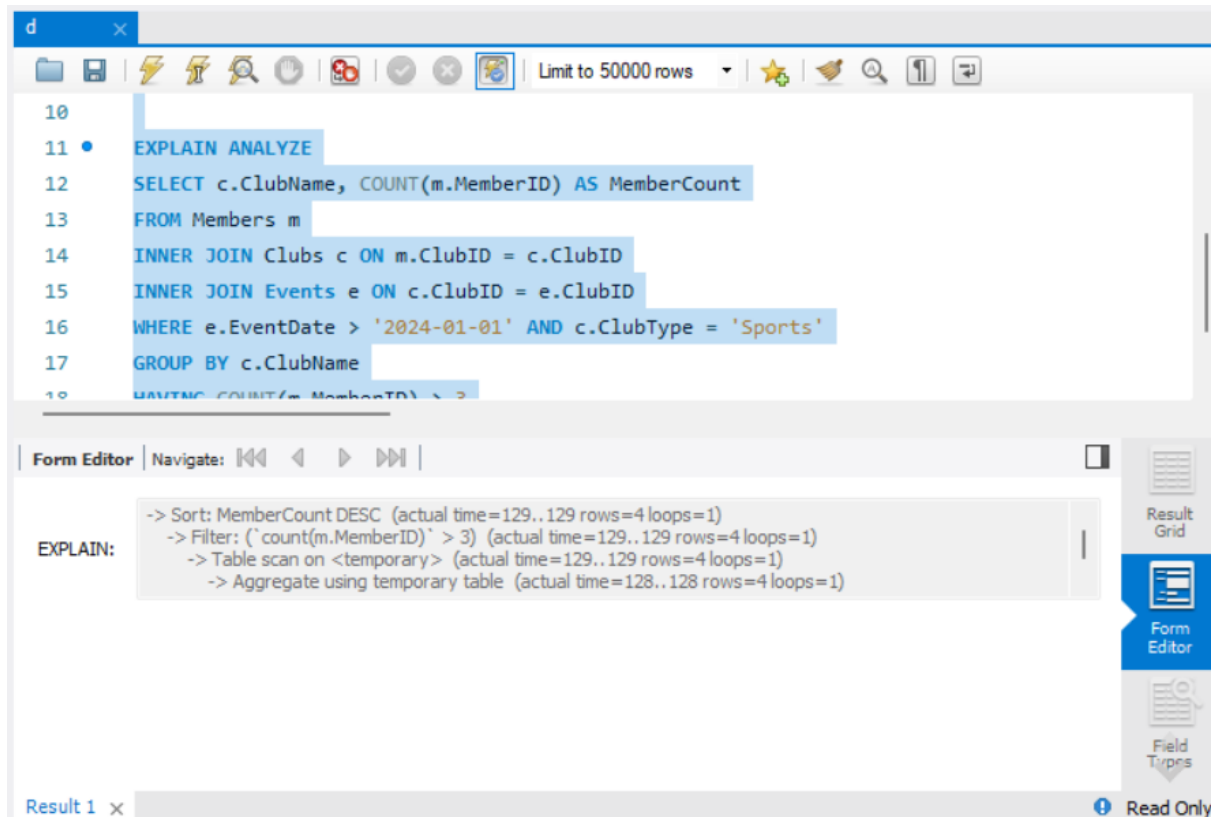
As we can see, execution time has reduced from 75 sec to 54.5 sec

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 :



The screenshot shows a database IDE interface. The top pane displays a SQL query with line numbers 10 through 18. The query is as follows:

```
10
11 • EXPLAIN ANALYZE
12 SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount
13 FROM Members m
14 INNER JOIN Clubs c ON m.ClubID = c.ClubID
15 INNER JOIN Events e ON c.ClubID = e.ClubID
16 WHERE e.EventDate > '2024-01-01' AND c.ClubType = 'Sports'
17 GROUP BY c.ClubName
18 HAVING COUNT(m.MemberID) > 3
```

The bottom pane shows the execution plan for the query, labeled "EXPLAIN:". The plan includes the following steps:

- > Sort: MemberCount DESC (actual time=129..129 rows=4 loops=1)
- > Filter: ('count(m.MemberID)' > 3) (actual time=129..129 rows=4 loops=1)
- > Table scan on <temporary> (actual time=129..129 rows=4 loops=1)
- > Aggregate using temporary table (actual time=128..128 rows=4 loops=1)

The IDE interface also shows a "Form Editor" tab, a "Result Grid" button, and a "Field Types" button. The bottom status bar indicates "Result 1" and "Read Only".

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;

The screenshot shows a database query editor with a SQL query and its execution plan. The query is as follows:

```
10  
11 • EXPLAIN ANALYZE  
12 SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount  
13 FROM Members m  
14 INNER JOIN Clubs c ON m.ClubID = c.ClubID  
15 INNER JOIN Events e ON c.ClubID = e.ClubID  
16 WHERE e.EventDate > '2024-01-01' AND c.ClubType = 'Sports'  
17 GROUP BY c.ClubName  
18 HAVING COUNT(m.MemberID) > 3
```

The execution plan (EXPLAIN) is shown below the query:

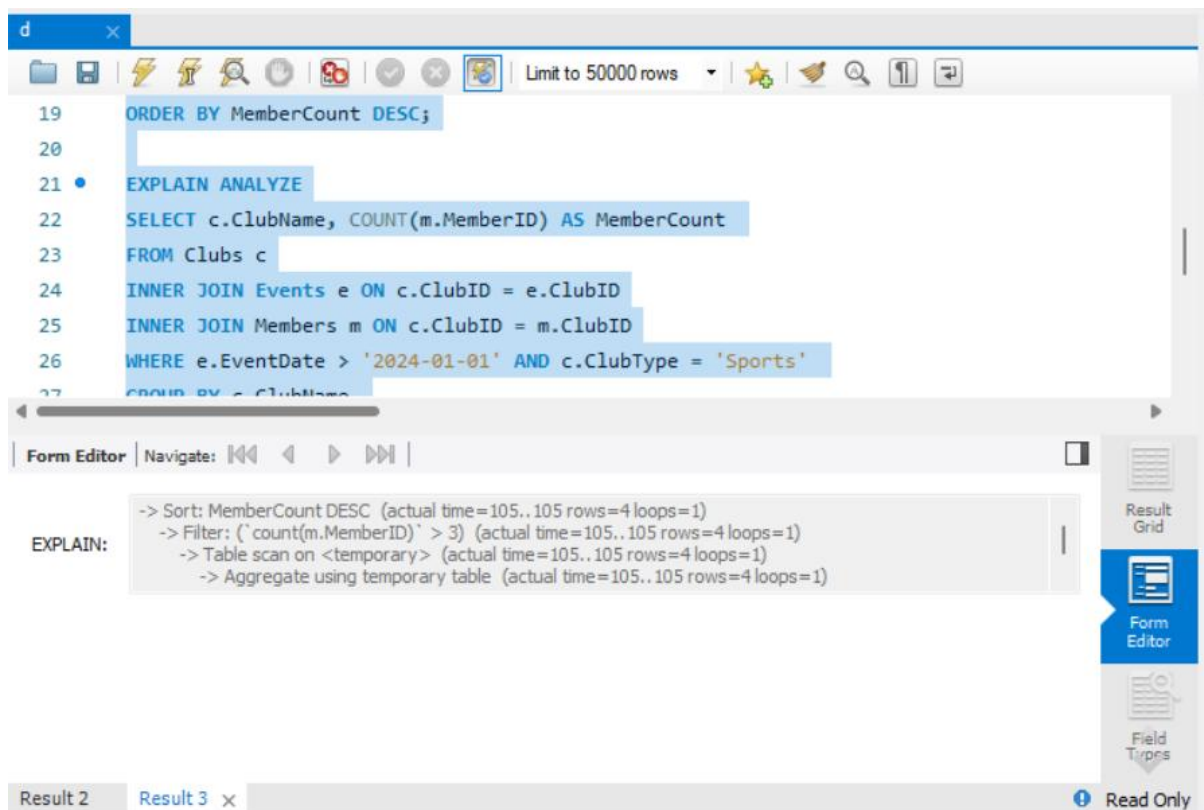
```
EXPLAIN:  
-> Sort: MemberCount DESC (actual time=38.1..38.1 rows=4 loops=1)  
-> Filter: ('count(m.MemberID)' > 3) (actual time=38.1..38.1 rows=4 loops=1)  
-> Table scan on <temporary> (actual time=38..38 rows=4 loops=1)  
-> Aggregate using temporary table (actual time=38..38 rows=4 loops=1)
```

The interface includes a toolbar at the top with icons for file operations, a 'Limit to 50000 rows' dropdown, and a 'Form Editor' tab. On the right side, there are buttons for 'Result Grid', 'Form Editor', and 'Field Types'. At the bottom, there are tabs for 'Result 2' and 'Result 3', and a 'Read Only' indicator.

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;
```



The screenshot shows a SQL IDE window with a query editor and an execution plan. The query is as follows:

```
19 ORDER BY MemberCount DESC;
20
21 EXPLAIN ANALYZE
22 SELECT c.ClubName, COUNT(m.MemberID) AS MemberCount
23 FROM Clubs c
24 INNER JOIN Events e ON c.ClubID = e.ClubID
25 INNER JOIN Members m ON c.ClubID = m.ClubID
26 WHERE e.EventDate > '2024-01-01' AND c.ClubType = 'Sports'
27 GROUP BY c.ClubName
```

The execution plan (EXPLAIN ANALYZE) is shown below the query:

```
EXPLAIN:
-> Sort: MemberCount DESC (actual time=105..105 rows=4 loops=1)
-> Filter: ('count(m.MemberID)' > 3) (actual time=105..105 rows=4 loops=1)
-> Table scan on <temporary> (actual time=105..105 rows=4 loops=1)
-> Aggregate using temporary table (actual time=105..105 rows=4 loops=1)
```

The IDE interface includes a toolbar at the top with icons for file operations, a 'Limit to 50000 rows' dropdown, and a 'Form Editor' tab at the bottom. The bottom status bar shows 'Result 2', 'Result 3', and a 'Read Only' indicator.

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

Incorporating a variety of join types such as subqueries to diversify optimization approaches.

1)

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

```
FROM Clubs c
```

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

```
WHERE c.ClubType = 'Sports'
```

```
AND c.ClubID IN (
```

```
    SELECT DISTINCT e.ClubID
```

```
    FROM Events e
```

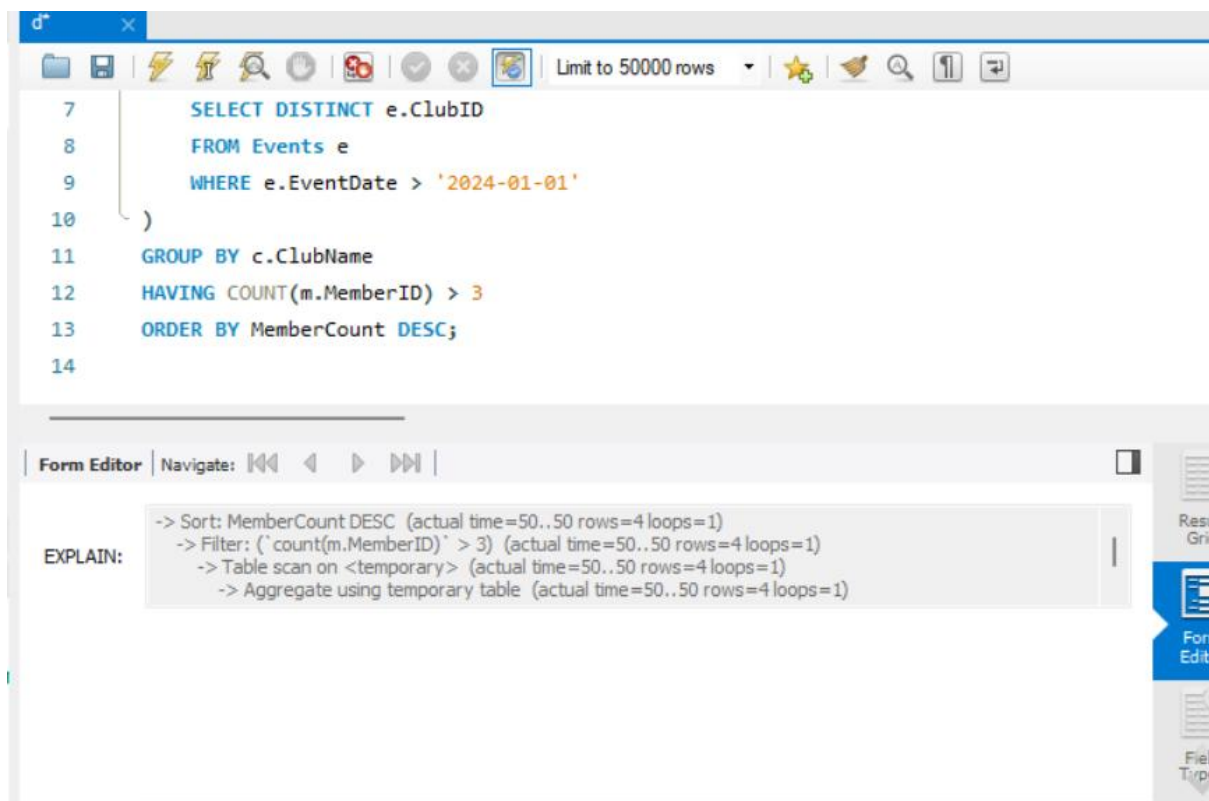
```
    WHERE e.EventDate > '2024-01-01'
```

```
)
```

```
GROUP BY c.ClubName
```

```
HAVING COUNT(m.MemberID) > 3
```

```
ORDER BY MemberCount DESC;
```



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 50000 rows' dropdown. The main editor displays a SQL query with line numbers 7 through 14. The query is a complex SELECT statement involving a subquery. Below the editor, the 'Form Editor' tab is active, showing an 'EXPLAIN:' section with the execution plan. The plan details the sort operation, filter, table scan, and aggregate function used.

```
7      SELECT DISTINCT e.ClubID
8      FROM Events e
9      WHERE e.EventDate > '2024-01-01'
10     )
11     GROUP BY c.ClubName
12     HAVING COUNT(m.MemberID) > 3
13     ORDER BY MemberCount DESC;
14
```

EXPLAIN:

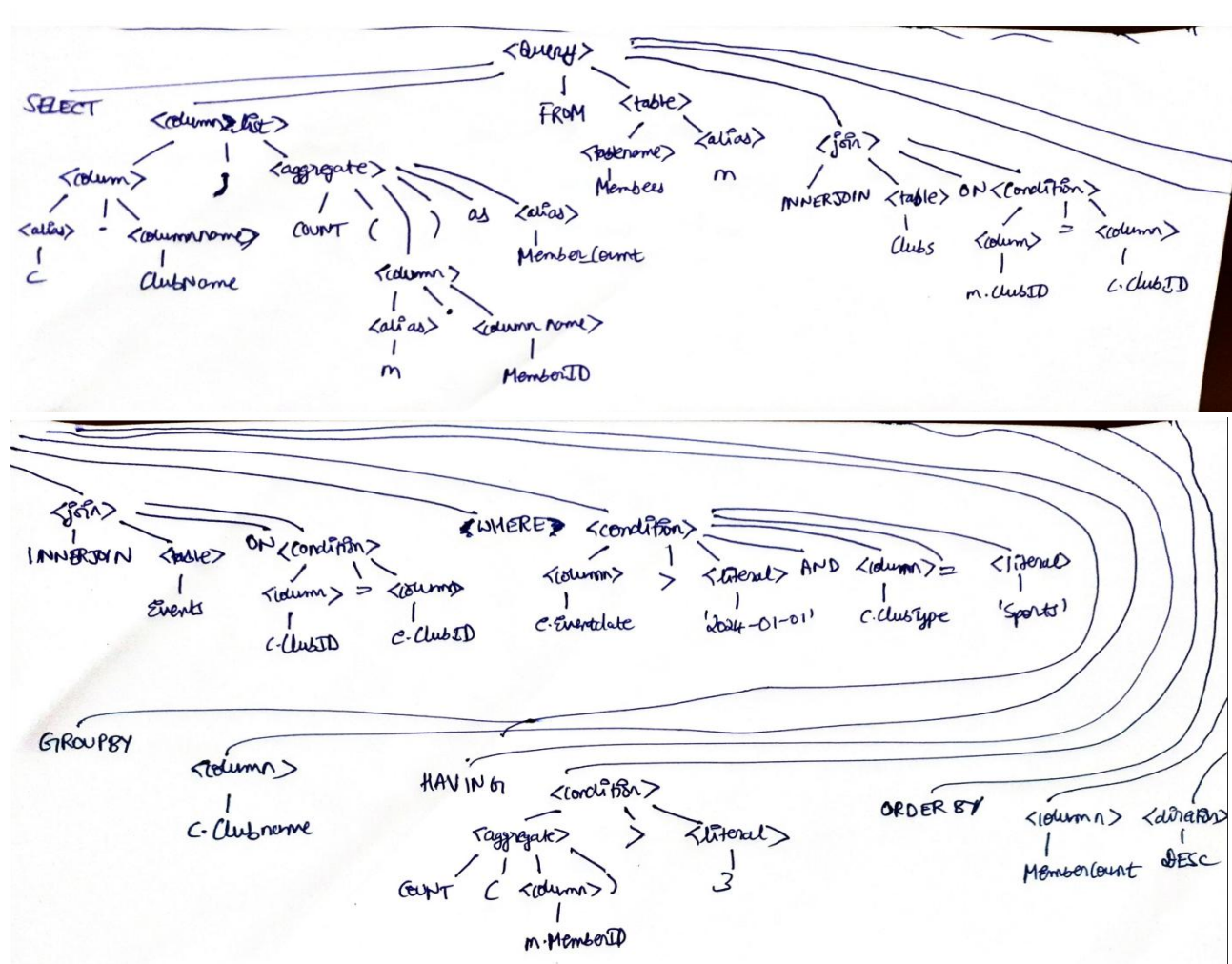
- > Sort: MemberCount DESC (actual time=50..50 rows=4 loops=1)
- > Filter: ('count(m.MemberID)' > 3) (actual time=50..50 rows=4 loops=1)
- > Table scan on <temporary> (actual time=50..50 rows=4 loops=1)
- > Aggregate using temporary table (actual time=50..50 rows=4 loops=1)

As we can see the execution time has reduced from 129 sec to 50 sec

e)

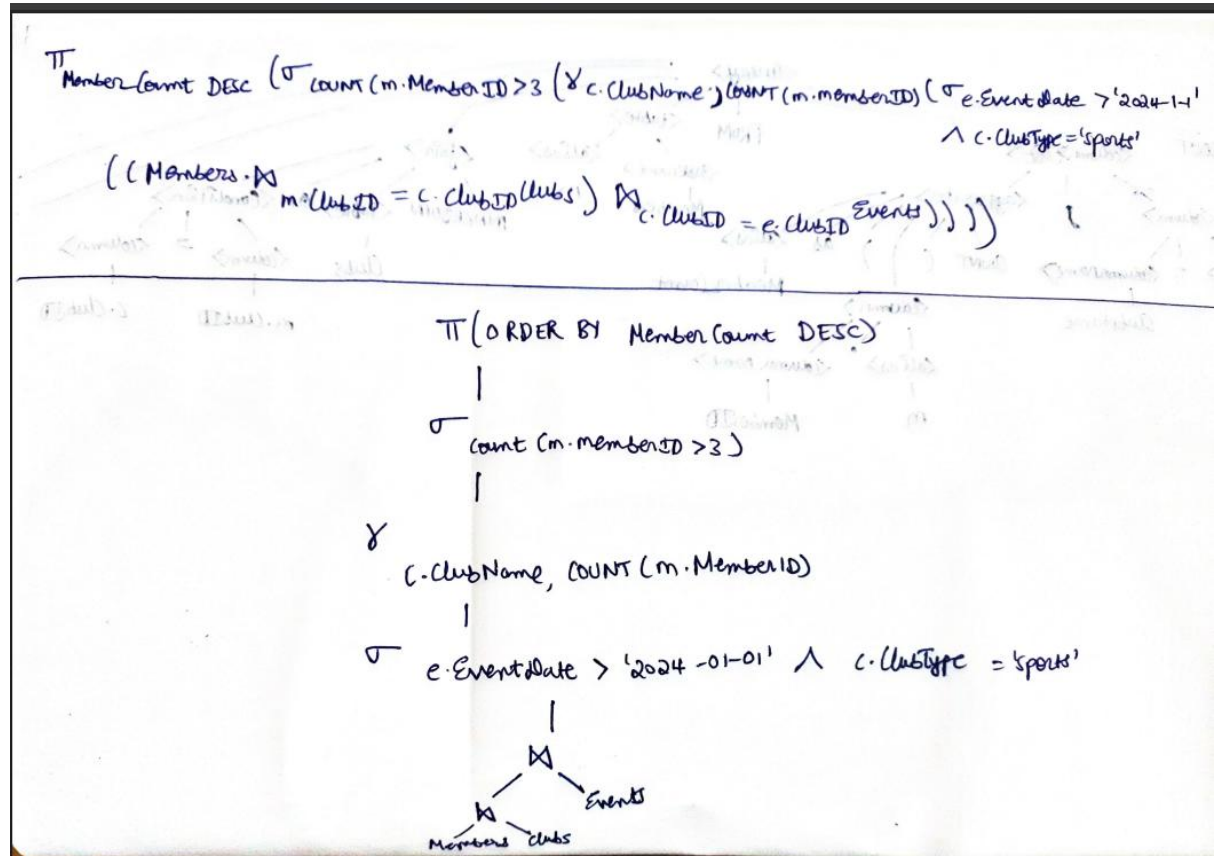
Part 1:

Parse tree:

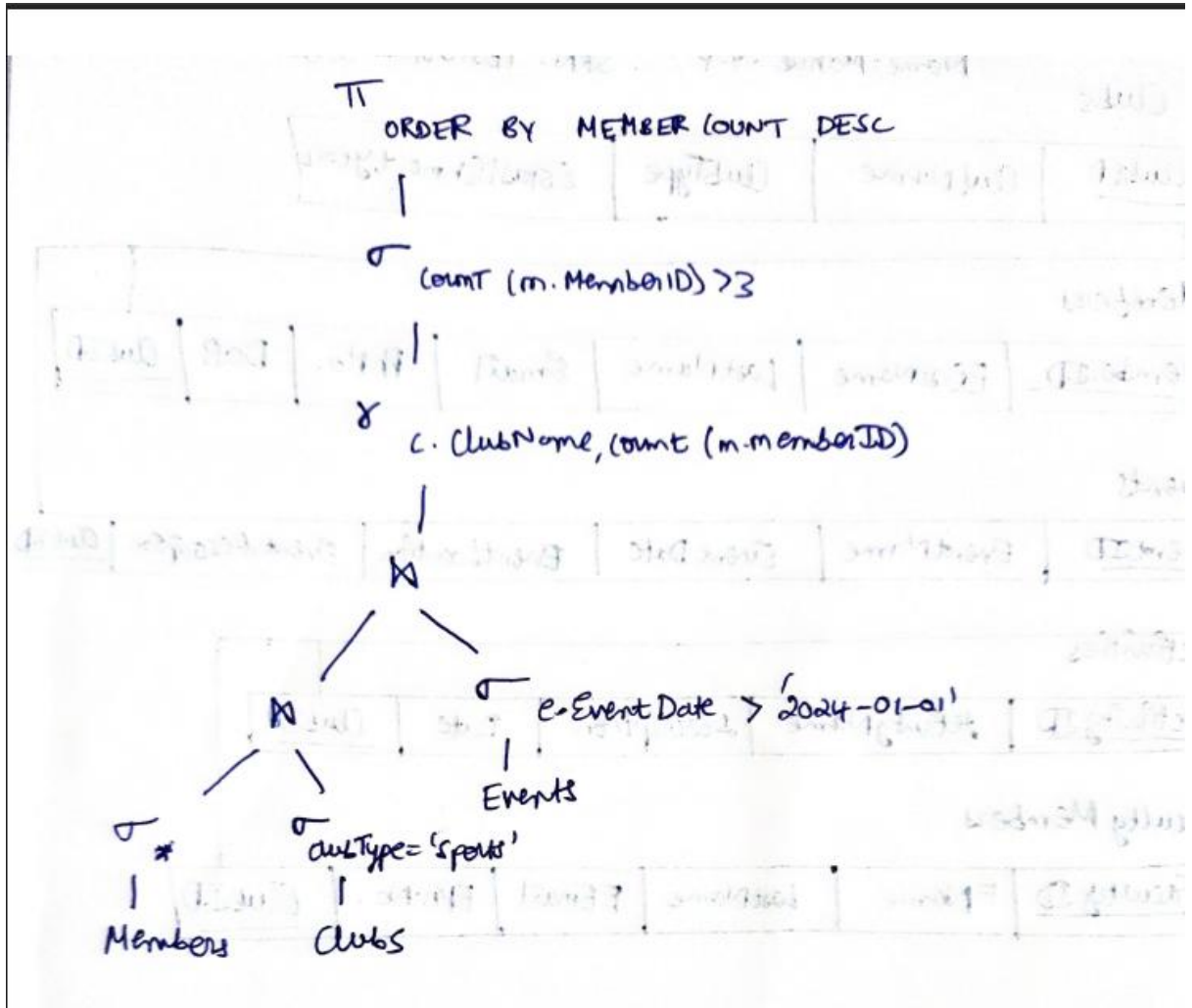


The long lines drawn on the 2nd screenshot are all coming from <Query>. I didn't have space to draw it on 1 single sheet which is why I combined 2 screenshots in this manner.

Relational Algebraic Expression and Initial query tree



Part 2:



Optimization : Pushing Selections Down

- In the initial query tree, the selection $\sigma (e.\text{EventDate} > '2024-01-01' \wedge c.\text{ClubType} = 'sports')$ 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.