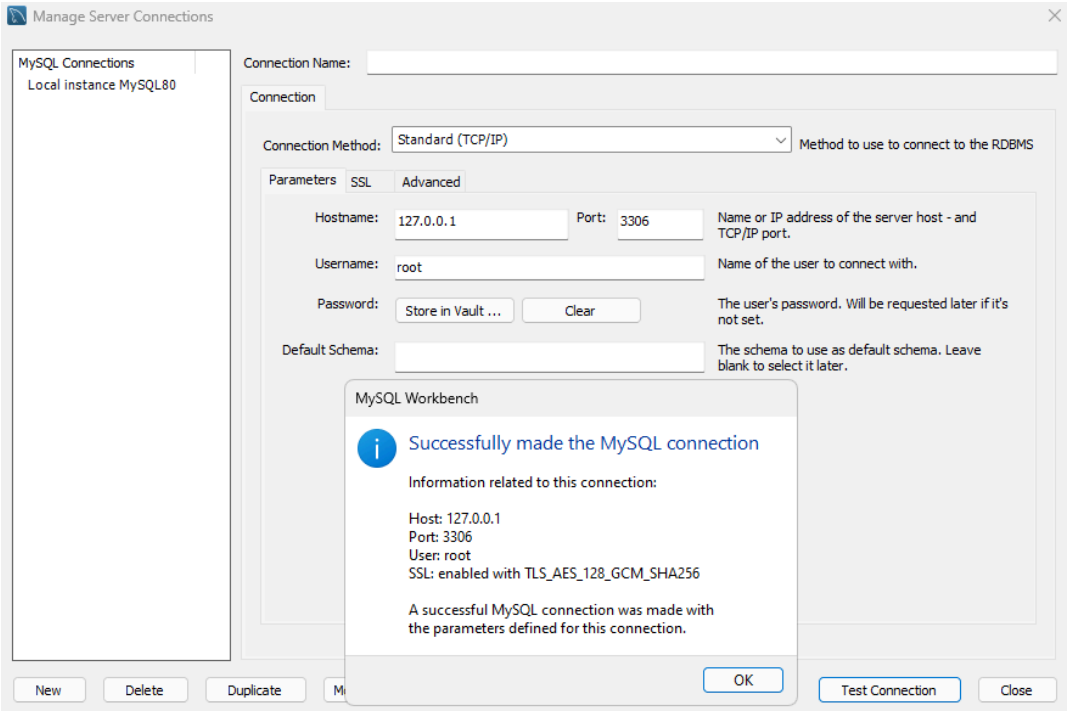
Part 1: Database Implementation

Database Connection (Local):

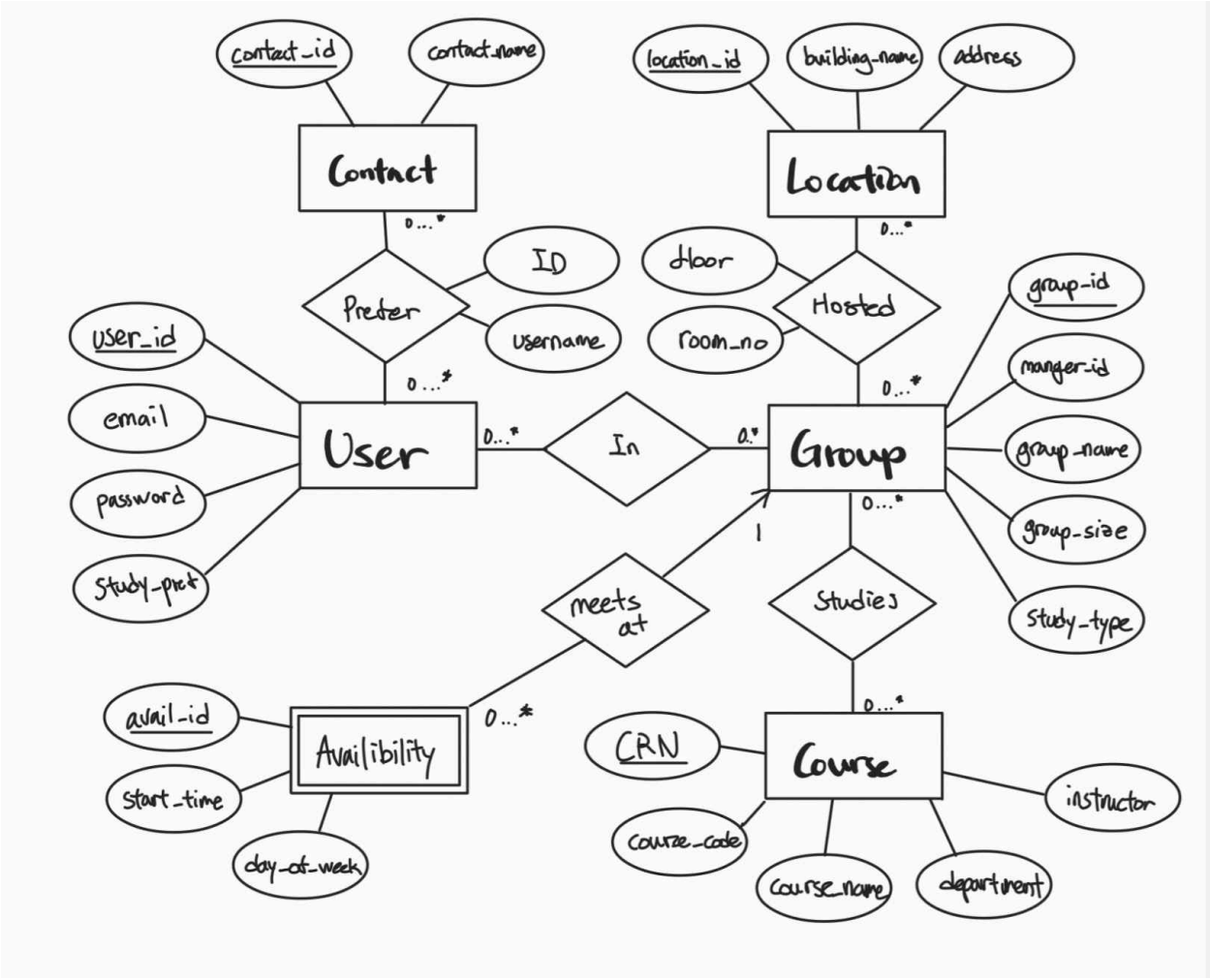


DDL Implementation

```
DROP DATABASE IF EXISTS StudyGroup;
CREATE DATABASE StudyGroup;
USE StudyGroup;

CREATE TABLE Users(
    user_id INT PRIMARY KEY,
    email VARCHAR(50),
    password CHAR(100),
    study_pref CHAR(100)
);

CREATE TABLE StudyGroup (
    group_id INT PRIMARY KEY,
```

```
    manager_id INT NOT NULL,
    group_name VARCHAR(50),
    group_size INT,
    study_type CHAR(100),
    FOREIGN KEY(manager_id) REFERENCES Users(user_id) ON DELETE CASCADE
);

CREATE TABLE Membership (
    user_id INT NOT NULL,
    group_id INT NOT NULL,
    role VARCHAR(50),
    PRIMARY KEY(user_id, group_id),
    FOREIGN KEY(user_id) REFERENCES Users(user_id) ON DELETE CASCADE,
    FOREIGN KEY(group_id) REFERENCES StudyGroup(group_id) ON DELETE CASCADE
);

CREATE TABLE Availability (
    avail_id INT NOT NULL,
    start_time TIME,
    day_of_week INT,
    group_id INT NOT NULL,
    PRIMARY KEY(avail_id, group_id),
    FOREIGN KEY(group_id) REFERENCES StudyGroup(group_id) ON DELETE CASCADE
);

CREATE TABLE Courses (
    CRN INT PRIMARY KEY,
    department VARCHAR(10),
    course_code INT,
    course_title VARCHAR(50),
    couese_name VARCHAR(10),
    instructor VARCHAR(50)
);


CREATE TABLE Group_Courses (
    group_id INT NOT NULL,
    CRN INT NOT NULL,
    PRIMARY KEY(group_id, CRN),
    FOREIGN KEY(group_id) REFERENCES StudyGroup(group_id) ON DELETE CASCADE,
    FOREIGN KEY(CRN) REFERENCES Courses(CRN) ON DELETE CASCADE
);

CREATE TABLE Locations (
    location_id INT PRIMARY KEY,
```

```
    building_name VARCHAR(50),
    address VARCHAR(50)
);


CREATE TABLE Group_Location (
    group_id INT NOT NULL,
    location_id INT NOT NULL,
    floor INT,
    room_no INT,
    PRIMARY KEY(group_id, location_id),
    FOREIGN KEY(group_id) REFERENCES StudyGroup(group_id) ON DELETE CASCADE,
    FOREIGN KEY(location_id) REFERENCES Locations(location_id) ON DELETE CASCADE
);

CREATE TABLE Contacts (
    contact_id INT PRIMARY KEY,
    contact_name VARCHAR(50)
);

CREATE TABLE User_Contact(
    contact_id INT NOT NULL,
    user_id INT NOT NULL,
    ID VARCHAR(50),
    username VARCHAR(50),
    PRIMARY KEY(contact_id, user_id),
    FOREIGN KEY(contact_id) REFERENCES Contacts(contact_id) ON DELETE CASCADE,
    FOREIGN KEY(user_id) REFERENCES Users(user_id) ON DELETE CASCADE
);
```



>=1000 Rows Insertion

```
1 • SELECT COUNT(user_id) FROM studygroup.users;
2 • SELECT COUNT(group_id) FROM studygroup.studygroup;
3 • SELECT COUNT(CRN) FROM studygroup.courses;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

COUNT(user_id)
1000

Result 4

Output

Action Output

#	Time	Action	Message
1	13:21:05	SELECT COUNT(user_id) FROM studygroup.users LIMIT 0, 2000	1 row(s) returned

```
1 •  SELECT COUNT(user_id) FROM studygroup.users;
2 •  SELECT COUNT(group_id) FROM studygroup.studygroup;
3 •  SELECT COUNT(CRN) FROM studygroup.courses;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

COUNT(group_id)
1000

Result 5

Output

Action Output

#	Time	Action	Message
1	13:21:05	SELECT COUNT(user_id) FROM studygroup.users LIMIT 0, 2000	1 row(s) returned
2	13:21:59	SELECT COUNT(group_id) FROM studygroup.studygroup LIMIT 0, 2000	1 row(s) returned

```
1 •  SELECT COUNT(user_id) FROM studygroup.users;
2 •  SELECT COUNT(group_id) FROM studygroup.studygroup;
3 •  SELECT COUNT(CRN) FROM studygroup.courses;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

COUNT(CRN)
2736

Result 6

Output

Action Output

#	Time	Action	Message
1	13:21:05	SELECT COUNT(user_id) FROM studygroup.users LIMIT 0, 2000	1 row(s) returned
2	13:21:59	SELECT COUNT(group_id) FROM studygroup.studygroup LIMIT 0, 2000	1 row(s) returned
3	13:22:20	SELECT COUNT(CRN) FROM studygroup.courses LIMIT 0, 2000	1 row(s) returned

Part 2: Advanced Queries

Query 1

1. Selects the number of groups in each department that study courses whose course code is between 500 and 600
    a. Example Query for search filter

```
SELECT COUNT(g.group_id), c.department
FROM studygroup AS g
JOIN group_courses AS gc
ON g.group_id = gc.group_id
JOIN courses AS c
ON gc.CRN = c.CRN
WHERE c.course_code BETWEEN 500 and 600
GROUP BY c.department;
```

Result:

COUNT(g.group_id)	department
53	ACCY
2	AE
2	ARCH
1	ATMS
35	BADM
2	BDI
2	BIOE
2	CHBE
3	CHEM
5	CHLH
2	CI
14	CS
13	ECE

Query 2

2. Selects most common subjects, that offers more than 40 courses
    a. Example Query for searching for classes when making a study group

```
SELECT DISTINCT course_name, department Subject FROM Courses
WHERE department IN
(SELECT department FROM Courses
GROUP BY department
HAVING COUNT(department) > 40);
```

Result:

course_name	Subject
undergraduate open seminar	MATH
Mgmt and Organizational Beh	BADM
Operations Strategy	BADM
Strategic Human Res Management	BADM
Fundamentals of Accounting	ACCY
Accounting Analysis I A	ACCY
Accounting Analysis II	ACCY
Multinational Management	BADM
Quantitative Analysis Lecture	CHEM
Inorganic Chemistry	CHEM
Instrumental Characterization	CHEM
Elementary Organic Chem II	CHEM
Electronic Music Synthesis	ECE
Computer Organization & Design	ECE
Physical Organic Chemistry	CHEM
Senior Design Project Lab	ECE
Power Electronics	ECE

Query 3

3. Searches based on a specific compatible time
    a. Example Query for search filter

```
SELECT u.user_id, email FROM Users AS u
JOIN Membership AS m
ON u.user_id = m.user_id
JOIN studygroup AS s
ON m.group_id = s.group_id
WHERE s.group_id IN
(SELECT DISTINCT group_id
FROM Availability AS a
WHERE a.start_time
IN ('20:00:00', '21:00:00', '22:00:00')
AND day_of_week IN (1,2));
```

Result:

user_id	email
147	skxelb87@illinois.edu
152	goqize14@illinois.edu
157	bvsfit92@illinois.edu
162	pmekgy36@illinois.edu
167	wdxpbz61@illinois.edu
172	nyxeac79@illinois.edu
177	oghvrb32@illinois.edu
182	lgwtas34@illinois.edu
187	tcpwsd45@illinois.edu
192	rzsfwj82@illinois.edu
197	iffwby92@illinois.edu
202	muhmsc91@illinois.edu
207	yxuixx77@illinois.edu
212	ymraox20@illinois.edu
217	utujld84@illinois.edu
222	nnljvi81@illinois.edu
227	daikkn83@illinois.edu
232	bhibur68@illinois.edu
237	bvucgl77@illinois.edu
242	cgjczc30@illinois.edu
247	uekqgg68@illinois.edu

Query 4

4. Finds out info of all classes the user joined a group of 3 or more people with
    a. Example Query for User

```
SELECT gc.group_id, c.CRN, c.course_name, c.department
FROM Courses AS c
JOIN Group_Courses AS gc ON c.CRN = gc.CRN
JOIN StudyGroup AS g ON g.group_id = gc.group_id
WHERE g.group_size >= 3 AND g.group_id IN
(SELECT m.group_id
FROM Membership AS m
JOIN Users AS u
ON u.user_id = m.user_id
WHERE u.email = "hxpwue64@illinois.edu");
```

Result:

group_id	CRN	course_name	department
1	41758	AAS 100	AAS

Part 3: Index Analysis

Query 1

EXPLAIN ANALYZE:

```
EXPLAIN: -> Table scan on <temporary>  (actual time=4.19..4.2 rows=13 loops=1)
           -> Aggregate using temporary table  (actual time=4.19..4.19 rows=13 loops=1)
             -> Nested loop inner join  (cost=483 rows=297) (actual time=0.239..3.96 rows=136 loops=1)
               -> Nested loop inner join  (cost=379 rows=297) (actual time=0.206..3.55 rows=136 loops=1)
```

After adding

1. For WHERE

- CREATE INDEX idx_courses_course_code ON Courses (course_code);

```
EXPLAIN: -> Table scan on <temporary>  (actual time=2.07..2.07 rows=13 loops=1)
           -> Aggregate using temporary table  (actual time=2.06..2.06 rows=13 loops=1)
             -> Nested loop inner join  (cost=492 rows=117) (actual time=0.0946..1.96 rows=136 loops=1)
               -> Nested loop inner join  (cost=451 rows=117) (actual time=0.0912..1.77 rows=136 loops=1)
```

2. For GROUP BY

- CREATE INDEX idx_courses_department ON Courses (department);

```
EXPLAIN: -> Table scan on <temporary>  (actual time=5.78..5.79 rows=13 loops=1)
           -> Aggregate using temporary table  (actual time=5.78..5.78 rows=13 loops=1)
             -> Nested loop inner join  (cost=483 rows=297) (actual time=0.214..4.49 rows=136 loops=1)
               -> Nested loop inner join  (cost=379 rows=297) (actual time=0.206..4.14 rows=136 loops=1)
```

Analysis:

1. The index for WHERE increases the cost, probably because there are now more course codes to iterate through than just 500-600.
2. The index for GROUP BY also does not decrease cost. Using the reasoning for #1

Result: We should not use an index for optimal performance.

Query 2

EXPLAIN ANALYZE:

```
EXPLAIN: -> Table scan on <temporary>  (cost=542..578 rows=2673) (actual time=13.7..13.8 rows=625 loops=1)
    -> Temporary table with deduplication  (cost=542..542 rows=2673) (actual time=13.7..13.7 rows=625 loops=1)
        -> Filter: <in_optimizer>(courses.department,courses.department in (select #2))  (cost=275 rows=2673) (actual time=6.79..12.6 rows=1471 loops=1)
            -> Table scan on Courses  (cost=275 rows=2673) (actual time=0.236..3.54 rows=2736 loops=1)
```

After adding

For GROUP BY

- CREATE INDEX idx_courses_department ON Courses (department);

```
EXPLAIN: -> Table scan on <temporary>  (cost=542..578 rows=2673) (actual time=7.99..8.1 rows=625 loops=1)
    -> Temporary table with deduplication  (cost=542..542 rows=2673) (actual time=7.99..7.99 rows=625 loops=1)
        -> Filter: <in_optimizer>(courses.department,courses.department in (select #2))  (cost=275 rows=2673) (actual time=2.75..6.98 rows=1471 loops=1)
            -> Table scan on Courses  (cost=275 rows=2673) (actual time=0.199..2.1 rows=2736 loops=1)
```

For HAVING

- CREATE INDEX idx_courses_course_name ON Courses (course_name);

```
EXPLAIN: -> Table scan on <temporary>  (cost=542..578 rows=2673) (actual time=9.69..9.93 rows=625 loops=1)
    -> Temporary table with deduplication  (cost=542..542 rows=2673) (actual time=9.69..9.69 rows=625 loops=1)
        -> Filter: <in_optimizer>(courses.department,courses.department in (select #2))  (cost=275 rows=2673) (actual time=3.75..8.59 rows=1471 loops=1)
            -> Table scan on Courses  (cost=275 rows=2673) (actual time=0.216..2.46 rows=2736 loops=1)
```

Analysis:

1. “department” is used in both filtering (`WHERE`) and grouping (`GROUP BY`). However, the subquery aggregates all the rows by `department`, and the cost is dominated by the aggregation step rather than by locating specific values.
2. The DISTINCT operations require sorting to ensure uniqueness of “course_name”. The index on “course_name” does not reduce the cost of this operation because the DISTINCT operations still need to evaluate all unique values across all rows.

Result: Indices are unnecessary for this query

Query 3

EXPLAIN ANALYZE:

EXPLAIN:

```
-> Nested loop inner join  (cost=11.9 rows=10) (actual time=0.0613..0.0815 rows=6 loops=1)
    -> Nested loop inner join  (cost=10.6 rows=10) (actual time=0.0573..0.0724 rows=6 loops=1)
        -> Nested loop inner join  (cost=9.39 rows=10) (actual time=0.049..0.0585 rows=6 loops=1)
            -> Table scan on <subquery2>  (cost=3.53..5.89 rows=10) (actual time=0.0421..0.0426 rows=6 loops=1)
```

After Adding:

1. FOR WHERE
    - CREATE INDEX group_idx ON studygroup(group_id);

EXPLAIN:

```
-> Nested loop inner join  (cost=11.9 rows=10) (actual time=0.0765..0.119 rows=6 loops=1)
    -> Nested loop inner join  (cost=10.6 rows=10) (actual time=0.0717..0.105 rows=6 loops=1)
        -> Nested loop inner join  (cost=9.39 rows=10) (actual time=0.0648..0.0872 rows=6 loops=1)
            -> Table scan on <subquery2>  (cost=3.53..5.89 rows=10) (actual time=0.055..0.0563 rows=6 loops=1)
```

2. FOR WHERE (in SELECT subquery)
    - CREATE INDEX day_idx ON availability(day_of_week);
    - CREATE INDEX time_idx ON availability(start_time);

EXPLAIN:

```
-> Nested loop inner join  (cost=11.9 rows=10) (actual time=0.0765..0.119 rows=6 loops=1)
    -> Nested loop inner join  (cost=10.6 rows=10) (actual time=0.0717..0.105 rows=6 loops=1)
        -> Nested loop inner join  (cost=9.39 rows=10) (actual time=0.0648..0.0872 rows=6 loops=1)
            -> Table scan on <subquery2>  (cost=3.53..5.89 rows=10) (actual time=0.055..0.0563 rows=6 loops=1)
```

3. FOR JOIN
    - CREATE INDEX user_idx ON users(user_id);

EXPLAIN:

```
-> Nested loop inner join  (cost=11.9 rows=10) (actual time=0.227..0.276 rows=6 loops=1)
    -> Nested loop inner join  (cost=10.6 rows=10) (actual time=0.222..0.26 rows=6 loops=1)
        -> Nested loop inner join  (cost=9.39 rows=10) (actual time=0.211..0.237 rows=6 loops=1)
            -> Table scan on <subquery2>  (cost=3.53..5.89 rows=10) (actual time=0.18..0.182 rows=6 loops=1)
```

<u>Analysis:</u>

1. The index for WHERE does not reduce the cost, likely because group_id is a primary key
2. The first index for WHERE in the SELECT subquery also does not reduce the cost, likely because the SELECT DISTINCT already makes the query check for unique values
3. The second index for WHERE in the SELECT subquery also does not reduce the cost, using the same reasoning as above.

4. The index in JOIN does not reduce the cost, likely as there already is an index for the primary key of users.

Result: For optimal performance, we should not use an index

Query 4

Index: CREATE INDEX email_idx ON users(email);

EXPLAIN ANALYZE

EXPLAIN:

```
-> Nested loop inner join  (cost=10918 rows=9060) (actual time=2.41..3.37 rows=30 loops=1)
  -> Inner hash join (m.group_id = `<subquery2>`.group_id)  (cost=9987 rows=9060) (actual time=2.38..3.28 rows=30 loops=1)
    -> Filter: (m.user_id is not null)  (cost=0.11 rows=100) (actual time=0.0177..0.885 rows=1000 loops=1)
      -> Table scan on m  (cost=0.11 rows=1000) (actual time=0.0164..0.784 rows=1000 loops=1)
```

After adding

1. for WHERE (in SELECT subquery)

- CREATE INDEX email_idx ON users(email);

EXPLAIN:

```
-> Nested loop inner join  (cost=2.13 rows=1) (actual time=0.237..0.253 rows=1 loops=1)
  -> Remove duplicate (gc, g) rows using temporary table (weedout)  (cost=1.78 rows=1) (actual time=0.207..0.223 rows=1 loops=1)
    -> Nested loop inner join  (cost=1.78 rows=1) (actual time=0.197..0.213 rows=1 loops=1)
      -> Nested loop inner join  (cost=1.43 rows=1) (actual time=0.155..0.17 rows=1 loops=1)
```

Analysis:

- By providing an index for a column in a subquery, I was able to make the subquery more efficient. Because the subquery was in several nested loops due from inner joins, the overall query became a lot more efficient.

Result: We need the following index for enhanced performance:

- CREATE INDEX email_idx ON users(email);