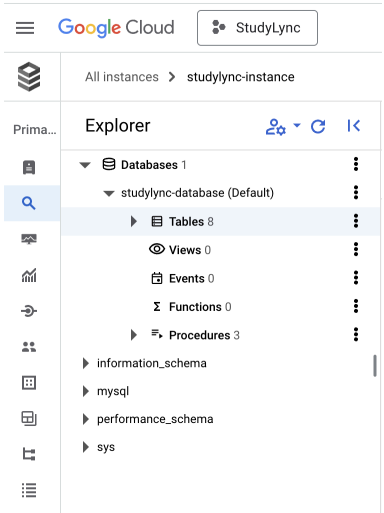
Database Implementation

Database Connection



DDL Commands for tables

1. Courses

```
Unset

CREATE TABLE Courses (

   CourseTitle VARCHAR(100) PRIMARY KEY,

   Department VARCHAR(50),

   CourseName VARVHAR(200)

);
```

2. Locations

```
Unset

CREATE TABLE Locations (

    LocationId INT AUTO_INCREMENT PRIMARY KEY,
```

```
    LocationName VARCHAR(100),
    Longitude DECIMAL(9,6),
    Latitude DECIMAL(9,6),
    Address VARCHAR(200)
);
```

3. Study Sessions

```
Unset
CREATE TABLE StudySessions (
   SessionId INT AUTO_INCREMENT PRIMARY KEY,
   CourseTitle VARCHAR(100),
   LocationId INT,
   Status VARCHAR(20),
   Description TEXT,
   FOREIGN KEY (CourseTitle) REFERENCES Courses(CourseTitle),
   FOREIGN KEY (LocationId) REFERENCES Locations(LocationId)
);
```

4. Users

```
Unset
CREATE TABLE Users (
   UserNetId VARCHAR(20) PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   Email VARCHAR(100) UNIQUE,
```

```
  Password VARCHAR(100),

  SessionId INT

 );
```

5. Chat

```
Unset

 CREATE TABLE Chat (

   MessageId INT AUTO_INCREMENT PRIMARY KEY,

   SessionId INT UNIQUE,

   Timestamp DATETIME,

   Content TEXT,

   FOREIGN KEY (SessionId) REFERENCES StudySessions(SessionId)

);
```

Inserting at least 1000

```
Unset
SELECT
  (SELECT COUNT(*) FROM Users) AS UserCount,
  (SELECT COUNT(*) FROM Locations) AS LocationCount,
  (SELECT COUNT(*) FROM Courses) AS CourseCount;
```

Output:

UserCount	LocationCount	CourseCount
1000	1245	4510

Advanced Queries

4 Advanced Queries + Screenshots

1. Count # of users at a locations

```
Unset
SELECT
  l.LocationName,
  COUNT(u.UserNetId) AS NumUsers
FROM Users u
JOIN StudySessions ss ON u.SessionId = ss.SessionId
JOIN Locations l ON ss.LocationId = l.LocationId
GROUP BY l.LocationName
ORDER BY NumUsers DESC;
```

Output (less than 15): The output is less than 15 because there are only 6 locations currently in use.

LocationName	NumUsers
Business Instructional Facility	29
Chemistry Annex	22
Illini Union	18
Main Library	13
Lincoln Hall	11
Grainger Engineering Library	7

2. Find the top 3 courses with the highest session attendance

```
Unset
SELECT
  ss.CourseTitle,
  COUNT(u.UserNetId) AS TotalAttendance
FROM Users u
JOIN StudySessions ss ON u.SessionId = ss.SessionId
WHERE ss.Status = 'Active'
GROUP BY ss.CourseTitle
ORDER BY TotalAttendance DESC
LIMIT 3;
```

Output (less than 15): The output is less than 15 because we only care about the top 3 courses. If we wanted more than 3, take out LIMIT 3.

CourseTitle	TotalAttendance
AAS 100	15
AAS 211	13
AAS 283	13

3. Which courses have ongoing sessions where the average number of users is above the system-wide average?

```
Unset
SELECT
  c.CourseTitle,
  ROUND(COUNT(u.UserNetId) / COUNT(DISTINCT ss.SessionId), 2) AS AvgUsersPerSession,
  ROUND(MAX(sys_avg.SystemAvgUsersPerSession), 4) AS SystemWideAvg
FROM Courses c
JOIN StudySessions ss ON c.CourseTitle = ss.CourseTitle
JOIN Users u ON u.SessionId = ss.SessionId
JOIN (
  SELECT
    COUNT(*) / COUNT(DISTINCT SessionId) AS SystemAvgUsersPerSession
  FROM Users
  WHERE SessionId IN (
    SELECT SessionId FROM StudySessions WHERE Status = 'Ongoing'
  )
) AS sys_avg
WHERE ss.Status = 'Ongoing'
GROUP BY c.CourseTitle
HAVING AvgUsersPerSession > SystemWideAvg
ORDER BY AvgUsersPerSession DESC;
```

Output (less than 15): The output is less than 15 because there are only 5 classes that have an average session attendance greater than the overall system average.

CourseTitle	AvgUsersPerSession	SystemWideAvg
AAS 288	4.00	3.3333
AAS 100	3.75	3.3333
AAS 275	3.67	3.3333
AAS 286	3.50	3.3333
AAS 287	3.50	3.3333

4. For each course, where do students most frequently meet?

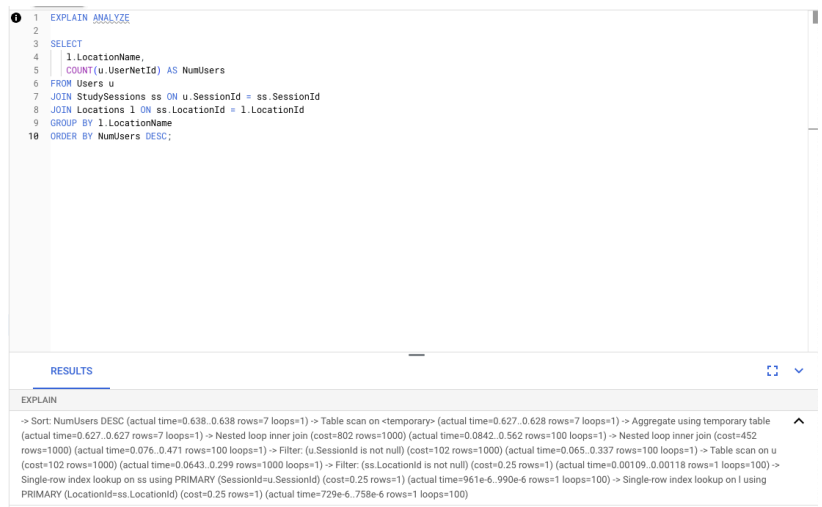
```
Unset
SELECT
 ss.CourseTitle,
 l.LocationName,
 COUNT(*) AS SessionCount
FROM StudySessions ss
JOIN Locations l ON ss.LocationId = l.LocationId
GROUP BY ss.CourseTitle, l.LocationName
HAVING COUNT(*) = (
   SELECT MAX(sub.SessionCount)
   FROM (
      SELECT
        COUNT(*) AS SessionCount
      FROM StudySessions
      WHERE CourseTitle = ss.CourseTitle
      GROUP BY LocationId
   ) AS sub
)
ORDER BY ss.CourseTitle;
```

Output (greater than 15):

CourseTitle	LocationName	SessionCount
AAS 100	Illini Union	1
AAS 100	Main Library	1
AAS 200	Business Instructional Facility	1
AAS 200	Grainger Engineering Library	1
AAS 211	Grainger Engineering Library	1
AAS 211	Illini Union	1
AAS 275	Lincoln Hall	2
AAS 283	Main Library	2
AAS 286	Business Instructional Facility	1
AAS 286	Lincoln Hall	1
AAS 287	Chemistry Annex	2
AAS 288	Business Instructional Facility	1
AAS 288	Chemistry Annex	1
AAS 297	Business Instructional Facility	2
AAS 299	Chemistry Annex	1

INDEX

1. Count # of users at a locations

Stage	Cost	Screenshot
Initial	802	

```
EXPLAIN ANALYZE

SELECT
    l.LocationName,
    COUNT(u.UserNetId) AS NumUsers
FROM Users u
JOIN StudySessions ss ON u.SessionId = ss.SessionId
JOIN Locations l ON ss.LocationId = l.LocationId
GROUP BY l.LocationName
ORDER BY NumUsers DESC;
```

RESULTS

EXPLAIN

-> Sort: NumUsers DESC (actual time=0.638..0.638 rows=7 loops=1) -> Table scan on <temporary> (actual time=0.627..0.628 rows=7 loops=1) -> Aggregate using temporary table (actual time=0.627..0.627 rows=7 loops=1) -> Nested loop inner join (cost=802 rows=1000) (actual time=0.0842..0.562 rows=100 loops=1) -> Nested loop inner join (cost=452 rows=1000) (actual time=0.076..0.471 rows=100 loops=1) -> Filter: (u.SessionId is not null) (cost=102 rows=1000) (actual time=0.065..0.337 rows=100 loops=1) -> Table scan on u (cost=102 rows=1000) (actual time=0.0643..0.299 rows=1000 loops=1) -> Filter: (ss.LocationId is not null) (cost=0.25 rows=1) (actual time=0.00109..0.00118 rows=1 loops=100) -> Single-row index lookup on ss using PRIMARY (SessionId=u.SessionId) (cost=0.25 rows=1) (actual time=961e-6..990e-6 rows=1 loops=100) -> Single-row index lookup on l using PRIMARY (LocationId=ss.LocationId) (cost=0.25 rows=1) (actual time=729e-6..758e-6 rows=1 loops=100)

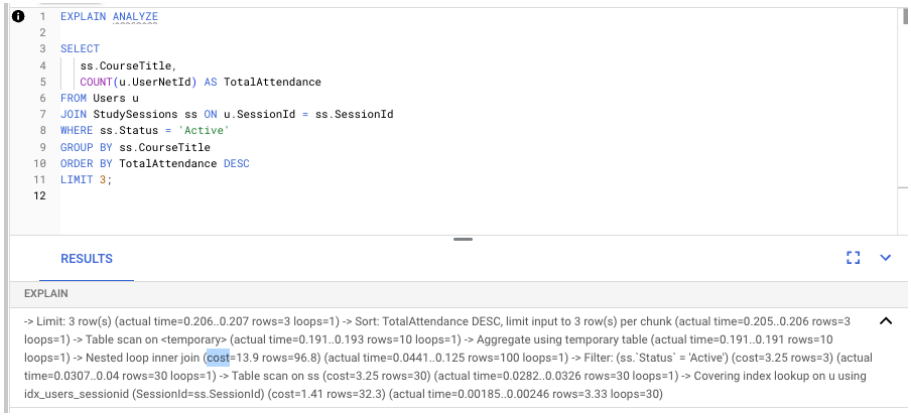
CREATE INDEX idx_locations_locationame ON Locations(LocationName);	802	 EXPLAIN ANALYZE SELECT l.LocationName, COUNT(u.UserNetId) AS NumUsers FROM Users u JOIN StudySessions ss ON u.SessionId = ss.SessionId JOIN Locations l ON ss.LocationId = l.LocationId GROUP BY l.LocationName ORDER BY NumUsers DESC; RESULTS EXPLAIN -> Sort: NumUsers DESC (actual time=0.54..0.541 rows=7 loops=1) -> Table scan on <temporary> (actual time=0.53..0.531 rows=7 loops=1) -> Aggregate using temporary table (actual time=0.53..0.53 rows=7 loops=1) -> Nested loop inner join (cost=802 rows=1000) (actual time=0.0406..0.472 rows=100 loops=1) -> Nested loop inner join (cost=452 rows=1000) (actual time=0.036..0.391 rows=100 loops=1) -> Filter: (u.SessionId is not null) (cost=102 rows=1000) (actual time=0.027..0.276 rows=100 loops=1) -> Table scan on u (cost=102 rows=1000) (actual time=0.0263..0.238 rows=1000 loops=1) -> Filter: (ss.LocationId is not null) (cost=0.25 rows=1) (actual time=903e-6..990e-6 rows=1 loops=100) -> Single-row index lookup on ss using PRIMARY (SessionId=u.SessionId) (cost=0.25 rows=1) (actual time=776e-6..804e-6 rows=1 loops=100) -> Single-row index lookup on l using PRIMARY (LocationId=ss.LocationId) (cost=0.25 rows=1) (actual time=625e-6..653e-6 rows=1 loops=100)
CREATE INDEX idx_users_sessionid ON Users(SessionId); (FOREIGN)	120	 EXPLAIN ANALYZE SELECT l.LocationName, COUNT(u.UserNetId) AS NumUsers FROM Users u JOIN StudySessions ss ON u.SessionId = ss.SessionId JOIN Locations l ON ss.LocationId = l.LocationId GROUP BY l.LocationName ORDER BY NumUsers DESC; RESULTS EXPLAIN -> Sort: NumUsers DESC (actual time=0.254..0.254 rows=7 loops=1) -> Table scan on <temporary> (actual time=0.235..0.236 rows=7 loops=1) -> Aggregate using temporary table (actual time=0.235..0.235 rows=7 loops=1) -> Nested loop inner join (cost=120 rows=968) (actual time=0.0401..0.174 rows=100 loops=1) -> Nested loop inner join (cost=13.8 rows=30) (actual time=0.0309..0.0633 rows=30 loops=1) -> Filter: (ss.LocationId is not null) (cost=3.25 rows=30) (actual time=0.0206..0.0282 rows=30 loops=1) -> Covering index scan on ss using LocationId (cost=3.25 rows=30) (actual time=0.0198..0.0249 rows=30 loops=1) -> Single-row index lookup on l using PRIMARY (LocationId=ss.LocationId) (cost=0.253 rows=1) (actual time=945e-6..976e-6 rows=1 loops=30) -> Covering index lookup on u using idx_users_sessionid (SessionId=ss.SessionId) (cost=0.44 rows=32.3) (actual time=0.00273..0.00332 rows=3.33 loops=30)
CREATE INDEX idx_studysessions_coursetitle_3 ON StudySessions(LocationID); (FOREIGN)	802	 EXPLAIN ANALYZE SELECT l.LocationName, COUNT(u.UserNetId) AS NumUsers FROM Users u JOIN StudySessions ss ON u.SessionId = ss.SessionId JOIN Locations l ON ss.LocationId = l.LocationId GROUP BY l.LocationName ORDER BY NumUsers DESC; RESULTS EXPLAIN -> Sort: NumUsers DESC (actual time=0.54..0.541 rows=7 loops=1) -> Table scan on <temporary> (actual time=0.53..0.531 rows=7 loops=1) -> Aggregate using temporary table (actual time=0.53..0.53 rows=7 loops=1) -> Nested loop inner join (cost=802 rows=1000) (actual time=0.0406..0.472 rows=100 loops=1) -> Nested loop inner join (cost=452 rows=1000) (actual time=0.036..0.391 rows=100 loops=1) -> Filter: (u.SessionId is not null) (cost=102 rows=1000) (actual time=0.027..0.276 rows=100 loops=1) -> Table scan on u (cost=102 rows=1000) (actual time=0.0263..0.238 rows=1000 loops=1) -> Filter: (ss.LocationId is not null) (cost=0.25 rows=1) (actual time=903e-6..990e-6 rows=1 loops=100) -> Single-row index lookup on ss using PRIMARY (SessionId=u.SessionId) (cost=0.25 rows=1) (actual time=776e-6..804e-6 rows=1 loops=100) -> Single-row index lookup on l using PRIMARY (LocationId=ss.LocationId) (cost=0.25 rows=1) (actual time=625e-6..653e-6 rows=1 loops=100)

Explanation:

While analyzing the execution of the query to determine the number of users per site, we considered the impact of three additional indexing techniques. The original cost of the query was 802. After adding an index to Locations(LocationName), the cost was still 802. The reason for this is probably that the LocationName column is utilized only in the GROUP BY clause, and since there are few location rows, the optimizer didn't utilize the index. Adding an index to StudySessions(LocationID) didn't help either. This is likely due to the fact that LocationID is already a foreign key optimized by the database. However, when we indexed Users(SessionId), the cost greatly improved to 120. This shows that the join between Users and StudySessions was the most performance-critical portion of the query, and indexing the foreign key SessionId in the Users table allowed for a much faster join, leading to a huge performance gain.

2. Find the top 3 courses with the highest session attendance’

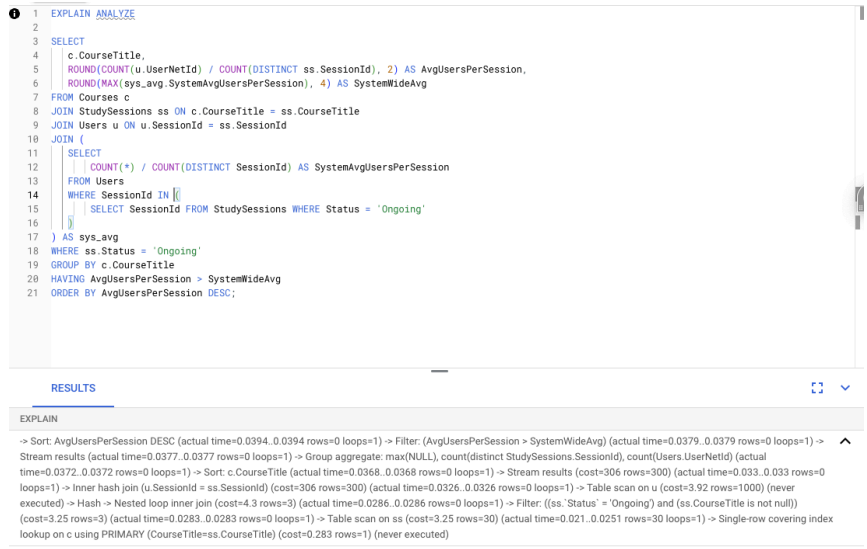
Stage	Cost	Screenshot
Initial	452	

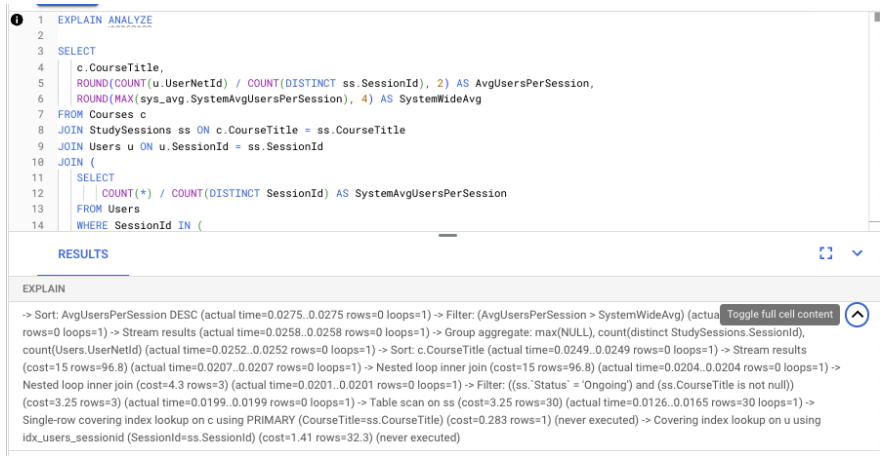
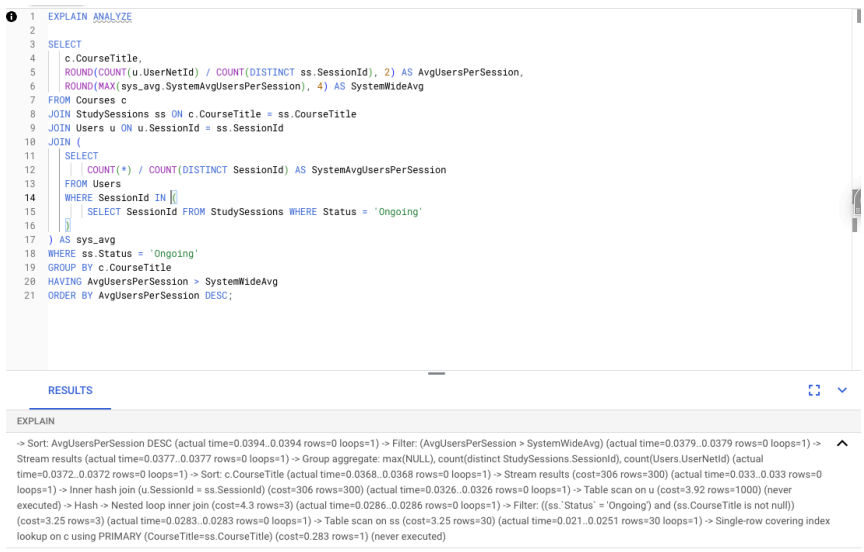
CREATE INDEX idx_users_sessionid ON Users(SessionId); (FOREIGN)	13.9	
CREATE INDEX idx_ss_status ON StudySessions(Status);	452	
CREATE INDEX idx_ss_courseTitle ON StudySessions(CourseTitle); (FOREIGN)	305	

To evaluate the performance of our query that counts total attendance per course for active sessions, we tested three indexing strategies one at a time. The initial query had a cost of 452. First, we created an index on Users(SessionId) since the query joins Users and StudySessions on that column. This change reduced the cost significantly to 13.9, showing that the join became much more efficient. Next, we created an index on StudySessions(Status) to help with filtering for active sessions, but the cost remained 452, suggesting the index did not improve the query in this case. Finally, we tested an index on StudySessions(CourseTitle), which slightly lowered the

cost to 305, indicating a small improvement likely related to the grouping step. Overall, the index on Users(SessionId) had the most noticeable impact on performance.

3. Which courses have ongoing sessions where the average number of users is above the system-wide average?

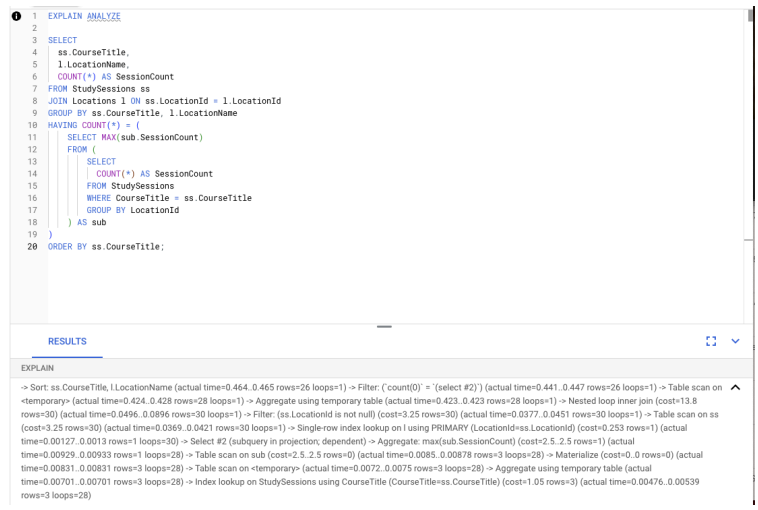
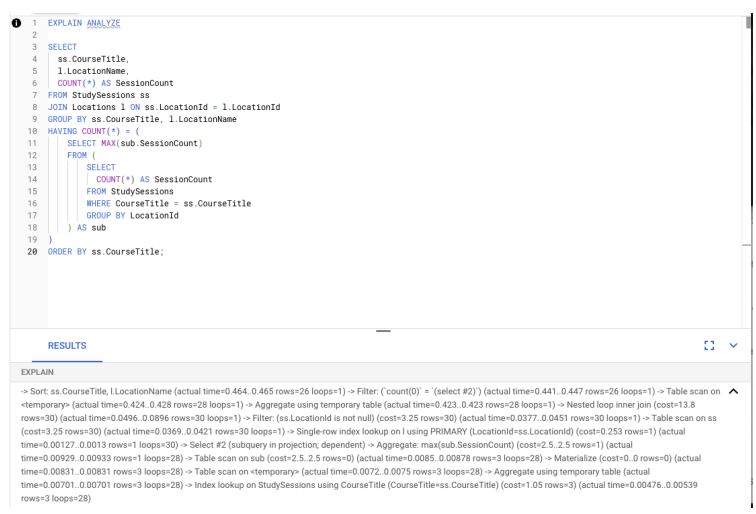
Stage	Cost	Screenshot
Initial	306	 EXPLAIN ANALYZE SELECT c.CourseTitle, ROUND(COUNT(u.UserNetId) / COUNT(DISTINCT ss.SessionId), 2) AS AvgUsersPerSession, ROUND(MAX(sys_avg.SystemAvgUsersPerSession), 4) AS SystemWideAvg FROM Courses c JOIN StudySessions ss ON c.CourseTitle = ss.CourseTitle JOIN Users u ON u.SessionId = ss.SessionId JOIN (SELECT COUNT(*) / COUNT(DISTINCT SessionId) AS SystemAvgUsersPerSession FROM Users WHERE SessionId IN (SELECT SessionId FROM StudySessions WHERE Status = 'Ongoing')) AS sys_avg WHERE ss.Status = 'Ongoing' GROUP BY c.CourseTitle HAVING AvgUsersPerSession > SystemWideAvg ORDER BY AvgUsersPerSession DESC; RESULTS EXPLAIN -> Sort: AvgUsersPerSession DESC (actual time=0.0394..0.0394 rows=0 loops=1) -> Filter: (AvgUsersPerSession > SystemWideAvg) (actual time=0.0379..0.0379 rows=0 loops=1) -> Stream results (actual time=0.0377..0.0377 rows=0 loops=1) -> Group aggregate: max(NULL), count(distinct StudySessions.SessionId), count(Users.UserNetId) (actual time=0.0372..0.0372 rows=0 loops=1) -> Sort: c.CourseTitle (actual time=0.0368..0.0368 rows=0 loops=1) -> Stream results (cost=306 rows=300) (actual time=0.033..0.033 rows=0 loops=1) -> Inner hash join (u.SessionId = ss.SessionId) (cost=306 rows=300) (actual time=0.0326..0.0326 rows=0 loops=1) -> Table scan on u (cost=3.92 rows=1000) (never executed) -> Hash -> Nested loop inner join (cost=4.3 rows=3) (actual time=0.0286..0.0286 rows=0 loops=1) -> Filter: ((ss.`Status` = 'Ongoing') and (ss.CourseTitle is not null)) (cost=3.25 rows=3) (actual time=0.0283..0.0283 rows=0 loops=1) -> Table scan on ss (cost=3.25 rows=30) (actual time=0.021..0.0251 rows=30 loops=1) -> Single-row covering index lookup on c using PRIMARY (CourseTitle=ss.CourseTitle) (cost=0.283 rows=1) (never executed)
CREATE INDEX idx_ss_status ON StudySessions(Status);	102	 EXPLAIN ANALYZE SELECT c.CourseTitle, ROUND(COUNT(u.UserNetId) / COUNT(DISTINCT ss.SessionId), 2) AS AvgUsersPerSession, ROUND(MAX(sys_avg.SystemAvgUsersPerSession), 4) AS SystemWideAvg FROM Courses c JOIN StudySessions ss ON c.CourseTitle = ss.CourseTitle JOIN Users u ON u.SessionId = ss.SessionId JOIN (SELECT COUNT(*) / COUNT(DISTINCT SessionId) AS SystemAvgUsersPerSession FROM Users WHERE SessionId IN (SELECT SessionId FROM StudySessions WHERE Status = 'Ongoing')) AS sys_avg WHERE ss.Status = 'Ongoing' GROUP BY c.CourseTitle HAVING AvgUsersPerSession > SystemWideAvg ORDER BY AvgUsersPerSession DESC; RESULTS EXPLAIN -> Sort: AvgUsersPerSession DESC (actual time=0.0168..0.0168 rows=0 loops=1) -> Filter: (AvgUsersPerSession > SystemWideAvg) (actual time=0.0151..0.0151 rows=0 loops=1) -> Stream results (actual time=0.0147..0.0147 rows=0 loops=1) -> Group aggregate: max(NULL), count(distinct StudySessions.SessionId), count(Users.UserNetId) (actual time=0.0141..0.0141 rows=0 loops=1) -> Sort: c.CourseTitle (actual time=0.0138..0.0138 rows=0 loops=1) -> Stream results (cost=102 rows=100) (actual time=0.0103..0.0103 rows=0 loops=1) -> Inner hash join (u.SessionId = ss.SessionId) (cost=102 rows=100) (actual time=0.01..0.01 rows=0 loops=1) -> Table scan on u (cost=11.8 rows=1000) (never executed) -> Hash -> Nested loop inner join (cost=0.7 rows=1) (actual time=0.00624..0.00624 rows=0 loops=1) -> Filter: (ss.CourseTitle is not null) (cost=0.35 rows=1) (actual time=0.00605..0.00605 rows=0 loops=1) -> Index lookup on ss using idx_ss_status (Status='Ongoing') (cost=0.35 rows=1) (actual time=0.00587..0.00587 rows=0 loops=1) -> Single-row covering index lookup on c using PRIMARY (CourseTitle=ss.CourseTitle) (cost=0.35 rows=1) (never executed)

CREATE INDEX idx_users_sessionid ON Users(SessionId); [FK]	15	 EXPLAIN ANALYZE SELECT c.CourseTitle, ROUND(COUNT(u.UserNetId) / COUNT(DISTINCT ss.SessionId), 2) AS AvgUsersPerSession, ROUND(MAX(sys_avg.SystemAvgUsersPerSession), 4) AS SystemWideAvg FROM Courses c JOIN StudySessions ss ON c.CourseTitle = ss.CourseTitle JOIN Users u ON u.SessionId = ss.SessionId JOIN (SELECT COUNT(*) / COUNT(DISTINCT SessionId) AS SystemAvgUsersPerSession FROM Users WHERE SessionId IN (RESULTS EXPLAIN -> Sort: AvgUsersPerSession DESC (actual time=0.0275..0.0275 rows=0 loops=1) -> Filter: (AvgUsersPerSession > SystemWideAvg) (actua Toggle full cell content rows=0 loops=1) -> Stream results (actual time=0.0258..0.0258 rows=0 loops=1) -> Group aggregate: max(NULL), count(distinct StudySessions.SessionId), count(Users.UserNetId) (actual time=0.0252..0.0252 rows=0 loops=1) -> Sort: c.CourseTitle (actual time=0.0249..0.0249 rows=0 loops=1) -> Stream results (cost=15 rows=96.8) (actual time=0.0207..0.0207 rows=0 loops=1) -> Nested loop inner join (cost=15 rows=96.8) (actual time=0.0204..0.0204 rows=0 loops=1) -> Nested loop inner join (cost=4.3 rows=3) (actual time=0.0201..0.0201 rows=0 loops=1) -> Filter: ((ss.`Status` = 'Ongoing') and (ss.CourseTitle is not null)) (cost=3.25 rows=3) (actual time=0.0199..0.0199 rows=0 loops=1) -> Table scan on ss (cost=3.25 rows=30) (actual time=0.0126..0.0165 rows=30 loops=1) -> Single-row covering index lookup on c using PRIMARY (CourseTitle=ss.CourseTitle) (cost=0.283 rows=1) (never executed) -> Covering index lookup on u using idx_users_sessionid (SessionId=ss.SessionId) (cost=1.41 rows=32.3) (never executed)
CREATE INDEX idx_ss_coursetitle ON StudySessions(CourseTitle); [FK]	306	 EXPLAIN ANALYZE SELECT c.CourseTitle, ROUND(COUNT(u.UserNetId) / COUNT(DISTINCT ss.SessionId), 2) AS AvgUsersPerSession, ROUND(MAX(sys_avg.SystemAvgUsersPerSession), 4) AS SystemWideAvg FROM Courses c JOIN StudySessions ss ON c.CourseTitle = ss.CourseTitle JOIN Users u ON u.SessionId = ss.SessionId JOIN (SELECT COUNT(*) / COUNT(DISTINCT SessionId) AS SystemAvgUsersPerSession FROM Users WHERE SessionId IN (SELECT SessionId FROM StudySessions WHERE Status = 'Ongoing')) AS sys_avg WHERE ss.Status = 'Ongoing' GROUP BY c.CourseTitle HAVING AvgUsersPerSession > SystemWideAvg ORDER BY AvgUsersPerSession DESC; RESULTS EXPLAIN -> Sort: AvgUsersPerSession DESC (actual time=0.0394..0.0394 rows=0 loops=1) -> Filter: (AvgUsersPerSession > SystemWideAvg) (actual time=0.0379..0.0379 rows=0 loops=1) -> Stream results (actual time=0.0377..0.0377 rows=0 loops=1) -> Group aggregate: max(NULL), count(distinct StudySessions.SessionId), count(Users.UserNetId) (actual time=0.0372..0.0372 rows=0 loops=1) -> Sort: c.CourseTitle (actual time=0.0368..0.0368 rows=0 loops=1) -> Stream results (cost=306 rows=300) (actual time=0.033..0.033 rows=0 loops=1) -> Inner hash join (u.SessionId = ss.SessionId) (cost=306 rows=300) (actual time=0.0326..0.0326 rows=0 loops=1) -> Table scan on u (cost=3.92 rows=1000) (never executed) -> Hash -> Nested loop inner join (cost=4.3 rows=3) (actual time=0.0286..0.0286 rows=0 loops=1) -> Filter: ((ss.`Status` = 'Ongoing') and (ss.CourseTitle is not null)) (cost=3.25 rows=3) (actual time=0.0283..0.0283 rows=0 loops=1) -> Table scan on ss (cost=3.25 rows=30) (actual time=0.021..0.0251 rows=30 loops=1) -> Single-row covering index lookup on c using PRIMARY (CourseTitle=ss.CourseTitle) (cost=0.283 rows=1) (never executed)

Query 3 Analysis:

When analyzing the performance of our query that finds courses with ongoing sessions where the average number of users is above the system-wide average, we tested three different indexing strategies individually. Initially, the query had a cost of 306. Adding an index on StudySessions(Status) reduced the cost to 102, likely because filtering by session status was a major part of the query and the index helped avoid a full table scan. In contrast, creating an index on StudySessions(CourseTitle) had no effect, as the cost remained 306. This may be because CourseTitle is only used for a simple join and the optimizer did not see a benefit in using the index. However, indexing Users(SessionId) reduced the cost significantly to 15. This shows that the join between Users and StudySessions was the main performance bottleneck, and indexing the foreign key in Users greatly improved efficiency.

4. For each course, where do students most frequently meet?

Stage/SQL Command	Cost	Screenshot
Initial	13.8	
CREATE INDEX idx_locations_locationanme ON Locations(LocationName);	13.8	
CREATE INDEX idx_ss_coursetitle ON StudySessions(CourseTitle); (FOREIGN)	13.8	

CREATE INDEX idx_studysessions_coursetitle_3 ON StudySessions(LocationID); (FOREIGN)	13.8	

```
EXPLAIN ANALYZE

SELECT
  ss.CourseTitle,
  l.LocationName,
  COUNT(*) AS SessionCount
FROM StudySessions ss
JOIN Locations l ON ss.LocationId = l.LocationId
GROUP BY ss.CourseTitle, l.LocationName
HAVING COUNT(*) = (
    SELECT MAX(sub.SessionCount)
    FROM (
        SELECT
            COUNT(*) AS SessionCount
        FROM StudySessions
        WHERE CourseTitle = ss.CourseTitle
        GROUP BY LocationId
    ) AS sub
)
ORDER BY ss.CourseTitle;
```

RESULTS

EXPLAIN

-> Sort: ss.CourseTitle, l.LocationName (actual time=0.464..0.465 rows=26 loops=1) -> Filter: (`count(0)` = `(select #2)`) (actual time=0.441..0.447 rows=26 loops=1) -> Table scan on <temporary> (actual time=0.424..0.428 rows=28 loops=1) -> Aggregate using temporary table (actual time=0.423..0.423 rows=28 loops=1) -> Nested loop inner join (cost=13.8 rows=30) (actual time=0.0496..0.0896 rows=30 loops=1) -> Filter: (ss.LocationId is not null) (cost=3.25 rows=30) (actual time=0.0377..0.0451 rows=30 loops=1) -> Table scan on ss (cost=3.25 rows=30) (actual time=0.0369..0.0421 rows=30 loops=1) -> Single-row index lookup on l using PRIMARY (LocationId=ss.LocationId) (cost=0.253 rows=1) (actual time=0.00127..0.0013 rows=1 loops=30) -> Select #2 (subquery in projection; dependent) -> Aggregate: max(sub.SessionCount) (cost=2.5..2.5 rows=1) (actual time=0.00929..0.00933 rows=1 loops=28) -> Table scan on sub (cost=2.5..2.5 rows=0) (actual time=0.0085..0.00878 rows=3 loops=28) -> Materialize (cost=0..0 rows=0) (actual time=0.00831..0.00831 rows=3 loops=28) -> Table scan on <temporary> (actual time=0.0072..0.0075 rows=3 loops=28) -> Aggregate using temporary table (actual time=0.00701..0.00701 rows=3 loops=28) -> Index lookup on StudySessions using CourseTitle (CourseTitle=ss.CourseTitle) (cost=1.05 rows=3) (actual time=0.00476..0.00539 rows=3 loops=28)

<u>Query 4 Analysis:</u>

The cost of the query did not alter since the plan was already optimal, and the additional indexes did not improve the access paths much. CourseTitle index did not do much good because the query already selects on CourseTitle, and the planner likely served this optimally without an extra index. LocationId index was not useful — while good for joins, it wasn't selective on its own. A composite index on (CourseTitle, LocationId) would have had greater impact for the GROUP BY and subquery. LocationName index wasn't used because the query doesn't search or filter on LocationName; it's only used in the SELECT and GROUP BY.