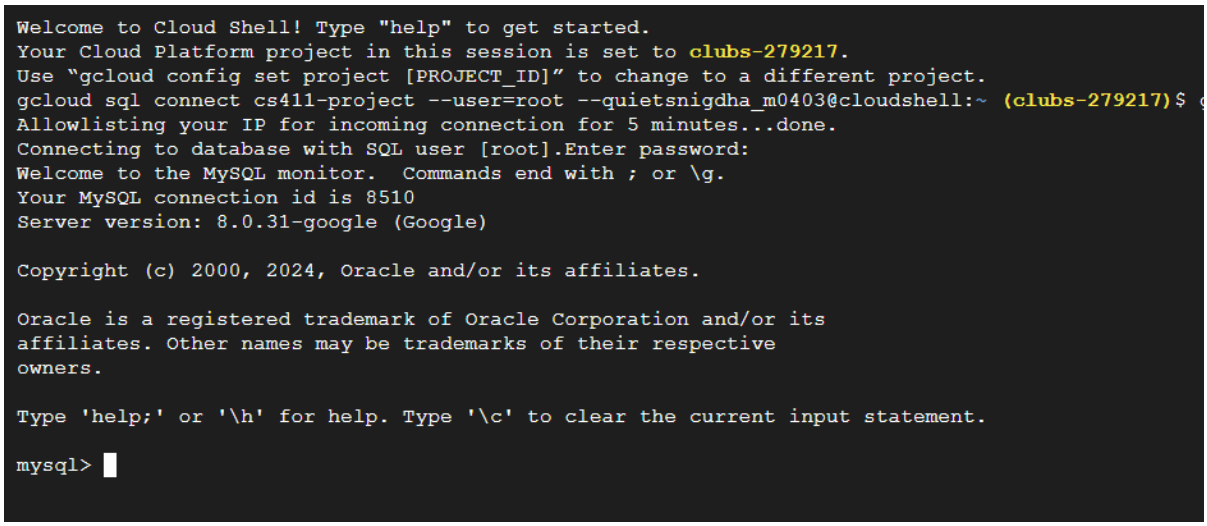
Part 1

Proof of Connection:



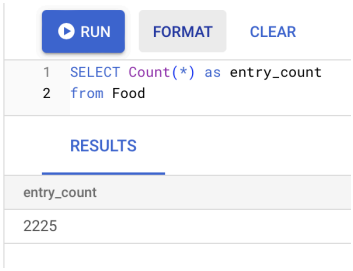
DDL Commands:

```
CREATE TABLE Users (
    User_Id INT PRIMARY KEY,
    Name VARCHAR(50),
    Password VARCHAR(50),
    Weight DECIMAL,
    Height DECIMAL
);
CREATE TABLE Food (
    Food_Id INT PRIMARY KEY,
    Name VARCHAR(50),
    Food_Category VARCHAR(50),
    Calories_Per_100g DECIMAL
);
CREATE TABLE Workouts (
    Workout_Id INT PRIMARY KEY,
    Name VARCHAR(50),
    Workout_Category VARCHAR(50),
    Body_Part VARCHAR(50),
    Equipment VARCHAR(50),
    Level VARCHAR(50)
);
```

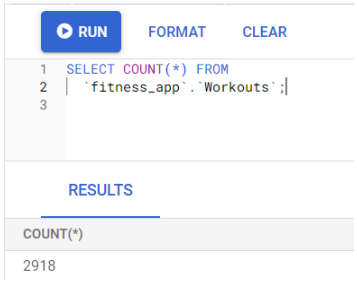
```sql
CREATE TABLE Logs (
    Log_Id INT PRIMARY KEY,
    User_Id INT,
    Log_Type VARCHAR(50),
    Date DATE,
    FOREIGN KEY (User_Id) REFERENCES Users(User_Id)
);
CREATE TABLE Meal (
    Log_Id INT,
    Food_Id INT,
    PRIMARY KEY (Log_Id, Food_Id),
    FOREIGN KEY (Log_Id) REFERENCES Logs(Log_Id),
    FOREIGN KEY (Food_Id) REFERENCES Food(Food_Id)
);
CREATE TABLE Session (
    Log_Id INT,
    Workout_id INT,
    PRIMARY KEY (Log_Id, Workout_id),
    FOREIGN KEY (Log_Id) REFERENCES Logs(Log_Id),
    FOREIGN KEY (Workout_id) REFERENCES Workouts(Workout_Id)
);
```

COUNT Query for Three Tables:

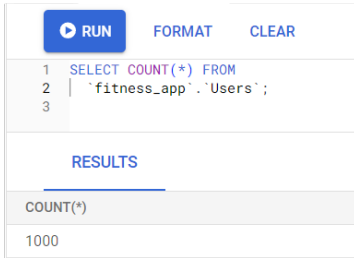
Food:



Workouts:



Users (randomly generated):



Four Advanced Queries:

Query 1: Get the total number of calories consumed in that day

```
SELECT u.User_Id,
       u.Name,
       l.Date AS Log_Date,
       SUM(f.Calories) AS
       Total_Calories_Consumed
FROM Users u
JOIN Logs l ON u.User_Id = l.User_Id
JOIN Meals m ON l.Log_Id = m.Log_Id
JOIN Food f ON m.Food_Id = f.Food_Id
WHERE l.Date = CURRENT_DATE AND l.Log_type
      = 'Meal'
GROUP BY u.User_Id, u.Name, l.Date
ORDER BY u.User_Id, l.Date
LIMIT 15;
```

User_Id	Name	Log_Date	Total_Calories_Consumed
3	User3	2024-04-02	231
178	User178	2024-04-02	241
201	User201	2024-04-02	375
253	User253	2024-04-02	45
274	User274	2024-04-02	221
311	User311	2024-04-02	137
321	User321	2024-04-02	351
358	User358	2024-04-02	60
365	User365	2024-04-02	394
390	User390	2024-04-02	355
394	User394	2024-04-02	422
415	User415	2024-04-02	89
433	User433	2024-04-02	29
456	User456	2024-04-02	159
458	User458	2024-04-02	52

Query 2: Select the top 5 most popular workouts

```
SELECT w.Name AS Workout_Name,
       COUNT(*) AS Total_Usage
FROM Session s
JOIN Workouts w ON s.Workout_id = w.Workout_Id
GROUP BY s.Workout_id
ORDER BY Total_Usage DESC
LIMIT 5;
```

Workout_Name	Total_Usage
371	6
780	5
17	5
255	4
232	4

Only received 5 results because the prompt was to select top 5 workouts

Query 3: Get users who have achieved weight loss goals

```
SELECT u.User_Id, u.Name, g.Weight AS GoalWeight,
u.Weight AS CurrentWeight
FROM Users  u
JOIN (
    SELECT g.User_Id, g.Type_of_Goal, MAX(g.Date) AS
MaxDate
    FROM Goals g
    WHERE g.Type_of_Goal = 'weight'
    GROUP BY g.User_Id, g.Type_of_Goal
) AS RecentWeightGoals ON u.User_Id =
RecentWeightGoals.User_Id
JOIN Goals g ON g.User_Id = u.User_Id AND g.Type_of_Goal = 'weight' AND g.Date
= RecentWeightGoals.MaxDate
WHERE u.Weight < g.Weight;
```

User_Id	Name	GoalWeight	CurrentWeight
189	User189	84	51
566	User566	141	95
732	User732	144	75
872	User872	87	81
884	User884	84	78
977	User977	68	65

Only received 6 results because the weights were randomly generated

Query 4: Get how many sessions each user has and their average number of workouts per session

```
SELECT u.User_Id,u.Name, COUNT(DISTINCT
l.Log_Id) AS Total_Sessions, AVG(Workout_Count)
AS Avg_Workouts_Per_Session
FROM Users u
JOIN Logs l ON u.User_Id = l.User_Id AND
l.Log_type = 'Workout'
JOIN (
       SELECT Log_Id, COUNT(Workout_Id) AS
Workout_Count
       FROM Session
       GROUP BY Log_Id
       ) AS Workout_Summary ON l.Log_Id =
Workout_Summary.Log_Id
GROUP BY u.User_Id, u.Name
LIMIT 15;
```

User_Id	Name	Total_Sessions	Avg_Workou
4	User4	1	2.0000
6	User6	1	2.0000
7	User7	2	2.0000
8	User8	1	2.0000
10	User10	1	2.0000
11	User11	1	2.0000
13	User13	2	2.0000
16	User16	1	2.0000
18	User18	1	2.0000
19	User19	3	2.0000
20	User20	2	2.0000
22	User22	1	2.0000
26	User26	1	2.0000
33	User33	1	2.0000
39	User39	3	2.0000

Part 2

Advanced Query Performance:

Query 1:

```
EXPLAIN
```

```
-> Sort: u.User_Id, u.`Name` (actual time=0.368..0.371 rows=30 loops=1) -> Table scan on <temporary> (actual time=0.342..0.347 rows=30 loops=1) -> Aggregate using temporary table (actual time=0.341..0.341 rows=30 loops=1) -> Nested loop inner join (cost=70.90 rows=90) (actual time=0.111..0.287 rows=30 loops=1) -> Nested loop inner join (cost=39.40 rows=90) (actual time=0.105..0.233 rows=30 loops=1) -> Nested loop inner join (cost=22.88 rows=30) (actual time=0.095..0.157 rows=30 loops=1) -> Filter: (l.User_Id is not null) (cost=12.38 rows=30) (actual time=0.085..0.093 rows=30 loops=1) -> Index lookup on l using idx_covering_logs (Log_Type='Meal', Date=curdate()) (cost=12.38 rows=30) (actual time=0.084..0.089 rows=30 loops=1) -> Single-row index lookup on u using PRIMARY (User_Id=l.User_Id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=30) -> Covering index lookup on m using PRIMARY (Log_Id=l.Log_Id) (cost=0.26 rows=3) (actual time=0.002..0.002 rows=1 loops=30) -> Single-row index lookup on f using PRIMARY (Food_Id=m.Food_Id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=30)
```

Query 2:

```
EXPLAIN
```

```
-> Limit: 5 row(s) (actual time=2.154..2.155 rows=5 loops=1) -> Sort: Total_Usage DESC, limit input to 5 row(s) per chunk (actual time=2.153..2.154 rows=5 loops=1) -> Stream results (cost=600.25 rows=1090) (actual time=0.099..2.046 rows=671 loops=1) -> Group aggregate: count(0) (cost=600.25 rows=1090) (actual time=0.093..1.897 rows=671 loops=1) -> Nested loop inner join (cost=491.25 rows=1090) (actual time=0.081..1.704 rows=1090 loops=1) -> Covering index scan on s using Workout_id (cost=109.75 rows=1090) (actual time=0.060..0.288 rows=1090 loops=1) -> Single-row index lookup on w using PRIMARY (Workout_Id=s.Workout_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1090)
```

Query 3:

```
EXPLAIN
```

```
-> Nested loop inner join (cost=168.50 rows=3) (actual time=4.484..8.302 rows=6 loops=1) -> Nested loop inner join (cost=165.00 rows=10) (actual time=3.521..7.079 rows=710 loops=1) -> Filter: (RecentWeightGoals.User_Id is not null) (cost=240.92..25.00 rows=200) (actual time=3.501..3.699 rows=703 loops=1) -> Table scan on RecentWeightGoals (cost=242.03..247.00 rows=200) (actual time=3.499..3.635 rows=703 loops=1) -> Materialize (cost=242.00..242.00 rows=200) (actual time=3.497..3.497 rows=703 loops=1) -> Group aggregate: max(g.`Date`) (cost=222.00 rows=200) (actual time=0.199..3.138 rows=703 loops=1) -> Filter: (g.Type_of_Goal = 'weight') (cost=202.00 rows=200) (actual time=0.192..2.887 rows=1035 loops=1) -> Index scan on g using fk_user_id (cost=202.00 rows=2000) (actual time=0.190..2.668 rows=2000 loops=1) -> Filter: ((g.`Date` = RecentWeightGoals.MaxDate) and (g.Type_of_Goal = 'weight')) (cost=0.50 rows=0.05) (actual time=0.004..0.005 rows=1 loops=703) -> Index lookup on g using fk_user_id (User_Id=RecentWeightGoals.User_Id) (cost=0.50 rows=2) (actual time=0.003..0.004 rows=2 loops=703) -> Filter: (u.Weight < g.Weight) (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=710) -> Single-row index lookup on u using PRIMARY (User_Id=RecentWeightGoals.User_Id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=710)
```

Query 4:

```
EXPLAIN
```

```
-> Limit: 15 row(s) (actual time=3.025..3.034 rows=15 loops=1) -> Group aggregate: count(distinct `Logs`.Log_Id), avg(Workout_Count) (actual time=3.024..3.033 rows=15 loops=1) -> Sort: u.User_Id, u.`Name` (actual time=3.015..3.016 rows=23 loops=1) -> Stream results (cost=1330.87 rows=561) (actual time=0.576..2.718 rows=545 loops=1) -> Nested loop inner join (cost=1330.87 rows=561) (actual time=0.573..2.552 rows=545 loops=1) -> Nested loop inner join (cost=1134.38 rows=561) (actual time=0.563..1.742 rows=545 loops=1) -> Table scan on Workout_Summary (cost=328.01..344.12 rows=1090) (actual time=0.542..0.628 rows=545 loops=1) -> Materialize (cost=328.00..328.00 rows=1090) (actual time=0.541..0.541 rows=545 loops=1) -> Group aggregate: count(`Session`.Workout_id) (cost=219.00 rows=1090) (actual time=0.056..0.469 rows=545 loops=1) -> Covering index scan on Session using PRIMARY (cost=110.00 rows=1090) (actual time=0.051..0.306 rows=1090 loops=1) -> Filter: ((l.Log_Type = 'Workout') and (l.User_Id is not null)) (cost=0.63 rows=1) (actual time=0.002..0.002 rows=1 loops=545) -> Single-row index lookup on l using PRIMARY (Log_Id=Workout_Summary.Log_Id) (cost=0.63 rows=1) (actual time=0.001..0.001 rows=1 loops=545) -> Single-row index lookup on u using PRIMARY (User_Id=l.User_Id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=545)
```

Indexing Analysis:

Query 1:

Attempt 1: `CREATE INDEX idx_logs_date_user_logtype_logid ON Logs(Date, User_Id, Log_type, Log_Id);`

EXPLAIN

-> Sort: u.User_Id, u.`Name` (actual time=0.832..0.837 rows=30 loops=1) -> Table scan on <temporary> (actual time=0.750..0.758 rows=30 loops=1) -> Aggregate using temporary table (actual time=0.748..0.748 rows=30 loops=1) -> Nested loop inner join (cost=10.85 rows=14) (actual time=0.100..0.477 rows=30 loops=1) -> Nested loop inner join (cost=5.92 rows=14) (actual time=0.091..0.384 rows=30 loops=1) -> Nested loop inner join (cost=3.33 rows=5) (actual time=0.080..0.244 rows=30 loops=1) -> Filter: ((l.Log_Type = 'Meal') and (l.User_Id is not null)) (cost=1.68 rows=5) (actual time=0.066..0.104 rows=30 loops=1) -> Covering index lookup on l using idx_logs_date_user_logtype_logid (Date=curdate()) (cost=1.68 rows=47) (actual time=0.058..0.076 rows=47 loops=1) -> Single-row index lookup on u using PRIMARY (User_Id=l.User_Id) (cost=0.27 rows=1) (actual time=0.003..0.003 rows=1 loops=30) -> Covering index lookup on m using PRIMARY (Log_Id=l.Log_Id) (cost=0.31 rows=3) (actual time=0.003..0.004 rows=1 loops=30) -> Single-row index lookup on f using PRIMARY (Food_Id=m.Food_Id) (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=30)

Attempt 2: `CREATE INDEX idx_meals_logid_foodid ON Meals(Log_Id, Food_Id);`

EXPLAIN

-> Sort: u.User_Id, u.`Name` (actual time=0.490..0.496 rows=30 loops=1) -> Table scan on <temporary> (actual time=0.386..0.391 rows=30 loops=1) -> Aggregate using temporary table (actual time=0.385..0.385 rows=30 loops=1) -> Nested loop inner join (cost=70.90 rows=90) (actual time=0.133..0.325 rows=30 loops=1) -> Nested loop inner join (cost=39.40 rows=90) (actual time=0.122..0.266 rows=30 loops=1) -> Nested loop inner join (cost=22.88 rows=30) (actual time=0.114..0.181 rows=30 loops=1) -> Filter: (l.User_Id is not null) (cost=12.38 rows=30) (actual time=0.105..0.113 rows=30 loops=1) -> Index lookup on l using idx_covering_logs (Log_Type='Meal', Date=curdate()) (cost=12.38 rows=30) (actual time=0.104..0.109 rows=30 loops=1) -> Single-row index lookup on u using PRIMARY (User_Id=l.User_Id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=30) -> Covering index lookup on m using PRIMARY (Log_Id=l.Log_Id) (cost=0.26 rows=3) (actual time=0.002..0.003 rows=1 loops=30) -> Single-row index lookup on f using PRIMARY (Food_Id=m.Food_Id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=30)

Attempt 3: `CREATE INDEX idx_food_foodid_calories ON Food(Food_Id, Calories);`

```
EXPLAIN
```

```
-> Sort: u.User_Id, u.`Name` (actual time=0.382..0.385 rows=30 loops=1) -> Table scan on <temporary> (actual time=0.358..0.362 rows=30 loops=1) -> Aggregate using temporary table (actual time=0.357..0.357 rows=30 loops=1) -> Nested loop inner join (cost=70.90 rows=90) (actual time=0.111..0.296 rows=30 loops=1) -> Nested loop inner join (cost=39.40 rows=90) (actual time=0.105..0.243 rows=30 loops=1) -> Nested loop inner join (cost=22.88 rows=30) (actual time=0.097..0.170 rows=30 loops=1) -> Filter: (l.User_Id is not null) (cost=12.38 rows=30) (actual time=0.087..0.094 rows=30 loops=1) -> Index lookup on l using idx_covering_logs (Log_Type='Meal', Date=curdate()) (cost=12.38 rows=30) (actual time=0.085..0.090 rows=30 loops=1) -> Single-row index lookup on u using PRIMARY (User_Id=l.User_Id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=30) -> Covering index lookup on m using PRIMARY (Log_Id=l.Log_Id) (cost=0.26 rows=3) (actual time=0.002..0.002 rows=1 loops=30) -> Single-row index lookup on f using PRIMARY (Food_Id=m.Food_Id) (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=30)
```

Final Design: `CREATE INDEX idx_logs_date_user_logtype_logid ON Logs(Date, User_Id, Log_type, Log_Id)` because it has the lowest cost estimate.

For attempt 1, the overall cost estimate is less than the "None" indexing design, indicating lower resource utilization. For attempt 2, the overall cost estimate is the same as the "None" indexing design, indicating similar resource utilization. For attempt 3, the overall cost estimate is the same as the "None" indexing design, indicating similar resource utilization. Therefore, from this analysis we can see that the third attempt would be the best option.

Query 2:

Attempt 1: `CREATE INDEX idx_session_workout_id ON Session(Workout_id);`

```
EXPLAIN
```

```
-> Limit: 5 row(s) (actual time=2.013..2.014 rows=5 loops=1) -> Sort: Total_Usage DESC, limit input to 5 row(s) per chunk (actual time=2.012..2.013 rows=5 loops=1) -> Stream results (cost=600.50 rows=1090) (actual time=0.058..1.909 rows=671 loops=1) -> Group aggregate: count(0) (cost=600.50 rows=1090) (actual time=0.055..1.765 rows=671 loops=1) -> Nested loop inner join (cost=491.50 rows=1090) (actual time=0.047..1.568 rows=1090 loops=1) -> Covering index scan on s using idx_session_workout_id (cost=110.00 rows=1090) (actual time=0.034..0.264 rows=1090 loops=1) -> Single-row index lookup on w using PRIMARY (Workout_Id=s.Workout_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1090)
```

Attempt 2: `CREATE INDEX idx_workouts_workout_id ON Workouts(Workout_Id);`

EXPLAIN

-> Limit: 5 row(s) (actual time=2.315..2.315 rows=5 loops=1) -> Sort: Total_Usage DESC, limit input to 5 row(s) per chunk (actual time=2.314..2.314 rows=5 loops=1) -> Table scan on <temporary> (actual time=2.150..2.234 rows=671 loops=1) -> Aggregate using temporary table (actual time=2.149..2.149 rows=671 loops=1) -> Nested loop inner join (cost=491.50 rows=1090) (actual time=0.063..1.617 rows=1090 loops=1) -> Covering index scan on s using PRIMARY (cost=110.00 rows=1090) (actual time=0.049..0.316 rows=1090 loops=1) -> Single-row index lookup on w using PRIMARY (Workout_Id=s.Workout_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1090)

Attempt 3: `CREATE INDEX idx_workouts_covering ON Workouts(Workout_Id, Name);`

EXPLAIN

-> Limit: 5 row(s) (actual time=2.465..2.466 rows=5 loops=1) -> Sort: Total_Usage DESC, limit input to 5 row(s) per chunk (actual time=2.464..2.465 rows=5 loops=1) -> Table scan on <temporary> (actual time=2.296..2.384 rows=671 loops=1) -> Aggregate using temporary table (actual time=2.294..2.294 rows=671 loops=1) -> Nested loop inner join (cost=491.50 rows=1090) (actual time=0.064..1.698 rows=1090 loops=1) -> Covering index scan on s using PRIMARY (cost=110.00 rows=1090) (actual time=0.045..0.328 rows=1090 loops=1) -> Single-row index lookup on w using PRIMARY (Workout_Id=s.Workout_id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1090)

Final Design: “None” because it has the lowest cost estimate.

For attempt 1, The overall cost estimate is higher compared to the "None" indexing design, indicating potentially higher resource utilization. For attempt 2, The overall cost estimate is the same as the "None" indexing design, indicating similar resource utilization. For attempt 3, The overall cost estimate is the same as the "None" indexing design, indicating similar resource utilization. Therefore, from this analysis we can see that the “None attempt would be the best option.

Query 3:

Attempt 1: `CREATE INDEX idx_user_id ON Users(User_Id);`

EXPLAIN

-> Nested loop inner join (cost=168.50 rows=3) (actual time=4.413..9.109 rows=6 loops=1) -> Nested loop inner join (cost=165.00 rows=10) (actual time=3.409..7.802 rows=710 loops=1) -> Filter: (RecentWeightGoals.User_Id is not null) (cost=240.92..25.00 rows=200) (actual time=3.391..3.631 rows=703 loops=1) -> Table scan on RecentWeightGoals (cost=242.03..247.00 rows=200) (actual time=3.389..3.552 rows=703 loops=1) -> Materialize (cost=242.00..242.00 rows=200) (actual time=3.386..3.386 rows=703 loops=1) -> Group aggregate: max(g.`Date`) (cost=222.00 rows=200) (actual time=0.191..3.072 rows=703 loops=1) -> Filter: (g.Type_of_Goal = 'weight') (cost=202.00 rows=200) (actual time=0.185..2.818 rows=1035 loops=1) -> Index scan on g using fk_user_id (cost=202.00 rows=2000) (actual time=0.182..2.599 rows=2000 loops=1) -> Filter: ((g.`Date` = RecentWeightGoals.MaxDate) and (g.Type_of_Goal = 'weight')) (cost=0.50 rows=0.05) (actual time=0.005..0.006 rows=1 loops=703) -> Index lookup on g using fk_user_id (User_Id=RecentWeightGoals.User_Id) (cost=0.50 rows=2) (actual time=0.004..0.005 rows=2 loops=703) -> Filter: (u.Weight < g.Weight) (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=710) -> Single-row index lookup on u using PRIMARY (User_Id=RecentWeightGoals.User_Id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=710)

Attempt 2: `CREATE INDEX idx_goals_user_type_date ON Goals(User_Id, Type_of_Goal, Date);`

EXPLAIN

-> Nested loop inner join (cost=165.81 rows=67) (actual time=3.245..7.878 rows=6 loops=1) -> Nested loop inner join (cost=95.00 rows=200) (actual time=1.474..2.913 rows=703 loops=1) -> Filter: ((RecentWeightGoals.User_Id is not null) and (RecentWeightGoals.MaxDate is not null)) (cost=240.92..25.00 rows=200) (actual time=1.460..1.746 rows=703 loops=1) -> Table scan on RecentWeightGoals (cost=242.03..247.00 rows=200) (actual time=1.459..1.632 rows=703 loops=1) -> Materialize (cost=242.00..242.00 rows=200) (actual time=1.457..1.457 rows=703 loops=1) -> Group aggregate: max(g.`Date`) (cost=222.00 rows=200) (actual time=0.064..1.127 rows=703 loops=1) -> Filter: (g.Type_of_Goal = 'weight') (cost=202.00 rows=200) (actual time=0.057..0.884 rows=1035 loops=1) -> Covering index scan on g using idx_goals_user_type_date (cost=202.00 rows=2000) (actual time=0.054..0.666 rows=2000 loops=1) -> Single-row index lookup on u using PRIMARY (User_Id=RecentWeightGoals.User_Id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=703) -> Filter: (u.Weight < g.Weight) (cost=0.25 rows=0.3) (actual time=0.007..0.007 rows=0 loops=703) -> Index lookup on g using idx_goals_user_type_date (User_Id=RecentWeightGoals.User_Id, Type_of_Goal='weight', Date=RecentWeightGoals.MaxDate) (cost=0.25 rows=1) (actual time=0.006..0.007 rows=1 loops=703)

Attempt 3: `CREATE INDEX idx_goals_covering ON Goals(User_Id, Type_of_Goal, Date, Weight);`

EXPLAIN

-> Nested loop inner join (cost=277.77 rows=67) (actual time=7.720..18.361 rows=6 loops=1) -> Nested loop inner join (cost=95.00 rows=200) (actual time=1.856..3.301 rows=703 loops=1) -> Filter: ((RecentWeightGoals.User_Id is not null) and (RecentWeightGoals.MaxDate is not null)) (cost=240.92..25.00 rows=200) (actual time=1.840..2.097 rows=703 loops=1) -> Table scan on RecentWeightGoals (cost=242.03..247.00 rows=200) (actual time=1.838..1.994 rows=703 loops=1) -> Materialize (cost=242.00..242.00 rows=200) (actual time=1.836..1.836 rows=703 loops=1) -> Group aggregate: max(g.`Date`) (cost=222.00 rows=200) (actual time=0.071..1.370 rows=703 loops=1) -> Filter: (g.Type_of_Goal = 'weight') (cost=202.00 rows=200) (actual time=0.064..1.103 rows=1035 loops=1) -> Covering index scan on g using idx_goals_covering (cost=202.00 rows=2000) (actual time=0.061..0.747 rows=2000 loops=1) -> Single-row index lookup on u using PRIMARY (User_Id=RecentWeightGoals.User_Id) (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=703) -> Filter: (u.Weight < g.Weight) (cost=0.81 rows=0.3) (actual time=0.021..0.021 rows=0 loops=703) -> Covering index lookup on g using idx_goals_covering (User_Id=RecentWeightGoals.User_Id, Type_of_Goal='weight', Date=RecentWeightGoals.MaxDate) (cost=0.81 rows=1) (actual time=0.020..0.021 rows=1 loops=703)

Final Design: `CREATE INDEX idx_goals_user_type_date ON Goals(User_Id, Type_of_Goal, Date)` because it has the lowest cost estimate.

For attempt 1, The overall cost estimate is the same as the "None" indexing design, indicating similar resource utilization. For attempt 2, The overall cost estimate is less than the "None" indexing design, indicating lower resource utilization. For attempt 3, The overall cost estimate is more than the "None" indexing design, indicating higher resource utilization. Therefore, from this analysis we can see that the third attempt would be the best option.

Query 4:

Attempt 1: `CREATE INDEX idx_logs_log_type ON Logs(Log_Type);`

EXPLAIN

-> Limit: 15 row(s) (actual time=2.957..2.968 rows=15 loops=1) -> Group aggregate: count(distinct `Logs`.Log_Id), avg(Workout_Count) (actual time=2.957..2.967 rows=15 loops=1) -> Sort: u.User_Id, u.`Name` (actual time=2.948..2.949 rows=23 loops=1) -> Stream results (cost=1330.87 rows=561) (actual time=0.540..2.653 rows=545 loops=1) -> Nested loop inner join (cost=1330.87 rows=561) (actual time=0.536..2.511 rows=545 loops=1) -> Nested loop inner join (cost=1134.38 rows=561) (actual time=0.525..1.731 rows=545 loops=1) -> Table scan on Workout_Summary (cost=328.01..344.12 rows=1090) (actual time=0.489..0.580 rows=545 loops=1) -> Materialize (cost=328.00..328.00 rows=1090) (actual time=0.488..0.488 rows=545 loops=1) -> Group aggregate: count(`Session`.Workout_id) (cost=219.00 rows=1090) (actual time=0.038..0.385 rows=545 loops=1) -> Covering index scan on Session using PRIMARY (cost=110.00 rows=1090) (actual time=0.033..0.275 rows=1090 loops=1) -> Filter: ((l.Log_Type = 'Workout') and (l.User_Id is not null)) (cost=0.63 rows=1) (actual time=0.002..0.002 rows=1 loops=545) -> Single-row index lookup on l using PRIMARY (Log_Id=Workout_Summary.Log_Id) (cost=0.63 rows=1) (actual time=0.001..0.001 rows=1 loops=545) -> Single-row index lookup on u using PRIMARY (User_Id=l.User_Id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=545)

Attempt 2: `CREATE INDEX idx_logs_user_log_type ON Logs(User_Id, Log_Type);`

EXPLAIN

-> Limit: 15 row(s) (actual time=2.756..2.766 rows=15 loops=1) -> Group aggregate: count(distinct `Logs`.Log_Id), avg(Workout_Count) (actual time=2.755..2.764 rows=15 loops=1) -> Sort: u.User_Id, u.`Name` (actual time=2.746..2.747 rows=23 loops=1) -> Stream results (cost=1330.87 rows=561) (actual time=0.553..2.505 rows=545 loops=1) -> Nested loop inner join (cost=1330.87 rows=561) (actual time=0.550..2.356 rows=545 loops=1) -> Nested loop inner join (cost=1134.38 rows=561) (actual time=0.541..1.600 rows=545 loops=1) -> Table scan on Workout_Summary (cost=328.01..344.12 rows=1090) (actual time=0.523..0.595 rows=545 loops=1) -> Materialize (cost=328.00..328.00 rows=1090) (actual time=0.521..0.521 rows=545 loops=1) -> Group aggregate: count(`Session`.Workout_id) (cost=219.00 rows=1090) (actual time=0.043..0.452 rows=545 loops=1) -> Covering index scan on Session using PRIMARY (cost=110.00 rows=1090) (actual time=0.039..0.325 rows=1090 loops=1) -> Filter: ((l.Log_Type = 'Workout') and (l.User_Id is not null)) (cost=0.63 rows=1) (actual time=0.002..0.002 rows=1 loops=545) -> Single-row index lookup on l using PRIMARY (Log_Id=Workout_Summary.Log_Id) (cost=0.63 rows=1) (actual time=0.001..0.001 rows=1 loops=545) -> Single-row index lookup on u using PRIMARY (User_Id=l.User_Id) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=545)

Attempt 3: `CREATE INDEX idx_session_log_id ON Session(Log_Id);`

EXPLAIN

-> Limit: 15 row(s) (actual time=10.378..10.395 rows=15 loops=1) -> Group aggregate: count(distinct `Logs`.Log_Id), avg(Workout_Count) (actual time=10.377..10.393 rows=15 loops=1) -> Sort: u.User_Id, u.`Name` (actual time=10.357..10.360 rows=23 loops=1) -> Stream results (cost=1330.87 rows=561) (actual time=0.810..3.877 rows=545 loops=1) -> Nested loop inner join (cost=1330.87 rows=561) (actual time=0.806..3.624 rows=545 loops=1) -> Nested loop inner join (cost=1134.38 rows=561) (actual time=0.797..2.447 rows=545 loops=1) -> Table scan on Workout_Summary (cost=328.01..344.12 rows=1090) (actual time=0.776..0.902 rows=545 loops=1) -> Materialize (cost=328.00..328.00 rows=1090) (actual time=0.774..0.774 rows=545 loops=1) -> Group aggregate: count(`Session`.Workout_id) (cost=219.00 rows=1090) (actual time=0.055..0.674 rows=545 loops=1) -> Covering index scan on Session using idx_session_log_id (cost=110.00 rows=1090) (actual time=0.049..0.493 rows=1090 loops=1) -> Filter: ((l.Log_Type = 'Workout') and (l.User_Id is not null)) (cost=0.63 rows=1) (actual time=0.002..0.003 rows=1 loops=545) -> Single-row index lookup on l using PRIMARY (Log_Id=Workout_Summary.Log_Id) (cost=0.63 rows=1) (actual time=0.002..0.002 rows=1 loops=545) -> Single-row index lookup on u using PRIMARY (User_Id=l.User_Id) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=545)

Final Design: `CREATE INDEX idx_logs_log_type ON Logs(Log_Type);` because it has the lowest cost estimate.

For attempt 1, the overall cost estimate is less than the "None" indexing design, indicating lower resource utilization. For attempt 2, the overall cost estimate is roughy the same as the “None” indexing design, indicating similar resource utilization. For attempt 3, The overall cost estimate is more than the "None" indexing design, indicating higher resource utilization. Therefore, from this analysis we can see that the first attempt would be the best option.