We have two separate databases in the proposal, both include variance that we are going to use in our formal function. In stage 2, we considered the potential problems that we might meet when combining the two datasets and set the datatype that will avoid the potential problems we estimated. However, when we started combining the two datasets in Stage 3 and dug deeper into the standard of healthy sleep and sleep disorder, we found out that there are some potential problems that will influence the efficiency of our SQL code. So we adjusted some of the data types. Also, after we combined the two datasets and divided them into 6 separate entities, we also found that it would be more reasonable to add the userID to each of the entities, because we use the userID to combine the two datasets, and we cannot make sure all the numbers of different variables belong to the same person if the userID does not exist in each entity.

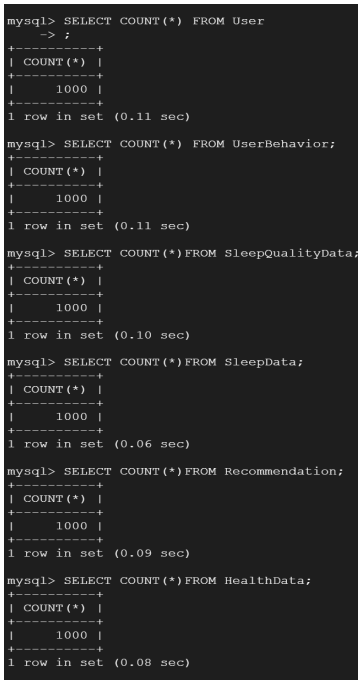
Part 1

1.Database implementation screenshots

Connection:

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to original-gasket-439919-v7.
Use “gcloud config set project [PROJECT_ID]” to change to a different project.
feihaofang5@cloudshell:~ (original-gasket-439919-v7)$ gcloud sql connect db-fa24-howdy --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 45
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE sleep;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLES;
+------------------+
| Tables_in_sleep  |
+------------------+
| HealthData       |
| Recommendation   |
| SleepData        |
| SleepQualityData |
| User             |
| UserBehavior     |
+------------------+
6 rows in set (0.01 sec)
```

Count query:



```
mysql> SELECT COUNT(*) FROM User
    -> ;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.11 sec)

mysql> SELECT COUNT(*) FROM UserBehavior;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.11 sec)

mysql> SELECT COUNT(*)FROM SleepQualityData;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.10 sec)

mysql> SELECT COUNT(*)FROM SleepData;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.06 sec)

mysql> SELECT COUNT(*)FROM Recommendation;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.09 sec)

mysql> SELECT COUNT(*)FROM HealthData;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.08 sec)
```

2. Data Definition Language (DDL) commands

User table:

```
CREATE TABLE User (
    UserID INT PRIMARY KEY,
    Password VARCHAR(255) NOT NULL,
    Gender VARCHAR(50),
    Age INT,
    ContactInfo VARCHAR(255),
    Occupation VARCHAR(255),
    Consent BOOLEAN
);
```

SleepQualityData table:

```
CREATE TABLE SleepQualityData (
    UserID INT PRIMARY KEY,
    SleepQualityScore DOUBLE,
    LightExposureHours DOUBLE,
    MovementDuringSleep DOUBLE,
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

Recommendation table:

```
CREATE TABLE Recommendation (
    UserID INT PRIMARY KEY,
```

```
    RecommendStr VARCHAR(255),
    SleepDisorder VARCHAR(255),
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

SleepData table:

```
CREATE TABLE SleepData (
    UserID INT PRIMARY KEY,
    SleepDuration DOUBLE,
    BedtimeConsistency DOUBLE,
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

HealthData table:

```
CREATE TABLE HealthData (
    UserID INT PRIMARY KEY,
    BMI DOUBLE,
    BloodPressure DOUBLE,
    HeartRate DOUBLE,
    BodyTemperature DOUBLE,
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

UserBehavior table:

```
CREATE TABLE UserBehavior (
    UserID INT PRIMARY KEY,
    PhysicalActivityLevel INT,
    DailySteps INT,
    CaffeineIntakeMg DOUBLE,
    StressLevel DOUBLE,
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

4. Advanced SQL Queries

1. SELECT u.UserID, AVG(sq.SleepQualityScore) AS AvgSleepQualityScore,
   AVG(sd.SleepDuration) AS AvgSleepDuration
   FROM User u
   JOIN UserBehavior ub ON u.UserID = ub.UserID
   JOIN SleepQualityData sq ON ub.UserID = sq.UserID
   JOIN SleepData sd ON sq.UserID = sd.UserID

```
WHERE ub.StressLevel > 7
GROUP BY u.UserID
LIMIT 15;
```

2. ```
   SELECT u.UserID, u.Gender, h.BloodPressure, ub.CaffeineIntakeMg
   FROM User u
   JOIN HealthData h ON u.UserID = h.UserID
   JOIN UserBehavior ub ON u.UserID = ub.UserID
   WHERE h.BloodPressure > 130 AND ub.CaffeineIntakeMg > (
      SELECT AVG(CaffeineIntakeMg)
      FROM UserBehavior
   )
   LIMIT 15;
   ```

3. ```
   SELECT u.UserID, sq.SleepQualityScore, sd.BedtimeConsistency,
   ub.PhysicalActivityLevel
   FROM User u
   JOIN SleepQualityData sq ON u.UserID = sq.UserID
   JOIN SleepData sd ON sq.UserID = sd.UserID
   JOIN UserBehavior ub ON sd.UserID = ub.UserID
   WHERE sq.SleepQualityScore < 50
    AND (
      sd.BedtimeConsistency < (
         SELECT AVG(BedtimeConsistency)
         FROM SleepData
      )
      OR ub.PhysicalActivityLevel < '50'
    )
   LIMIT 15;
   ```

4. ```
   SELECT u.UserID, r.RecommendStr, ub.StressLevel, sd.BedtimeConsistency
   FROM User u
   JOIN UserBehavior ub ON u.UserID = ub.UserID
   ```
   - ```
     JOIN SleepData sd ON ub.UserID = sd.UserID
     ```
   ```
   JOIN Recommendation r ON sd.UserID = r.UserID
   WHERE ub.StressLevel > 7
    AND sd.BedtimeConsistency > (
      SELECT AVG(BedtimeConsistency)
      FROM SleepData
   )
   ```

LIMIT 15;

5. Execution of SQL Queries w/ Screenshots

1.

```
mysql> SELECT u.UserID, AVG(sq.SleepQualityScore) AS AvgSleepQualityScore, AVG(sd.SleepDuration) AS AvgSleepDuration
    -> FROM User u
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> JOIN SleepQualityData sq ON ub.UserID = sq.UserID
    -> JOIN SleepData sd ON sq.UserID = sd.UserID
    -> WHERE ub.StressLevel > 7
    -> GROUP BY u.UserID
    -> LIMIT 15;
+--------+---------------------+-------------------+
| UserID | AvgSleepQualityScore | AvgSleepDuration  |
+--------+---------------------+-------------------+
|      9 |   2.4092370548842656 |       9.124036581 |
|     14 |                   1 | 5.404322722330513 |
|     17 |         9.080636464 | 5.220980886784736 |
|     23 |                   1 | 6.882860089325996 |
|     27 |    4.545444348532154 | 7.785635834887587 |
|     42 |                  10 | 7.735459441452651 |
|     43 |                   1 | 6.620203095654096 |
|     48 |                   1 | 6.995617891978392 |
|     51 |                   1 | 5.442489454154301 |
|     52 |                   1 |       6.435838496 |
|     56 |   6.3118741801552485 | 6.214841859259811 |
|     58 |                   1 | 7.806583547404465 |
|     59 |                   1 | 5.696775710885866 |
|     64 |    7.757159055055291 | 6.706001831327444 |
|     68 |                   1 | 7.527602446175157 |
+--------+---------------------+-------------------+
15 rows in set (0.00 sec)
```

2.

```
mysql> SELECT u.UserID, u.Gender, h.BloodPressure, ub.CaffeineIntakeMg
    -> FROM User u
    -> JOIN HealthData h ON u.UserID = h.UserID
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> WHERE h.BloodPressure > 130 AND ub.CaffeineIntakeMg > (
    ->     SELECT AVG(CaffeineIntakeMg)
    ->     FROM UserBehavior
    -> )
    -> LIMIT 15;
+--------+--------+---------------+--------------------+
| UserID | Gender | BloodPressure | CaffeineIntakeMg   |
+--------+--------+---------------+--------------------+
|      5 |        |         130.3 | 223.28290818217448 |
|      8 |        |         132.5 | 198.77748210843384 |
|     12 |        |         139.4 | 223.35478135058656 |
|     13 |        |         137.2 | 150.87836548981477 |
|     19 |        |         135.8 | 165.53249991994014 |
|     25 |        |         137.2 |  377.7086498053304 |
|     30 |        |         131.4 |  151.4508999262654 |
|     45 |        |         134.3 |  232.8058814893309 |
|     49 |        |         133.5 | 262.18615047942524 |
|     51 |        |         133.7 | 222.90636450023644 |
|     53 |        |           137 | 184.16602319526265 |
|     62 |        |         139.2 | 254.69117944881395 |
|     75 |        |         139.4 | 170.08450931157097 |
|     80 |        |         137.5 |  297.8071898356444 |
|     85 |        |         137.2 | 245.62950020913465 |
+--------+--------+---------------+--------------------+
15 rows in set (0.00 sec)
```

3.

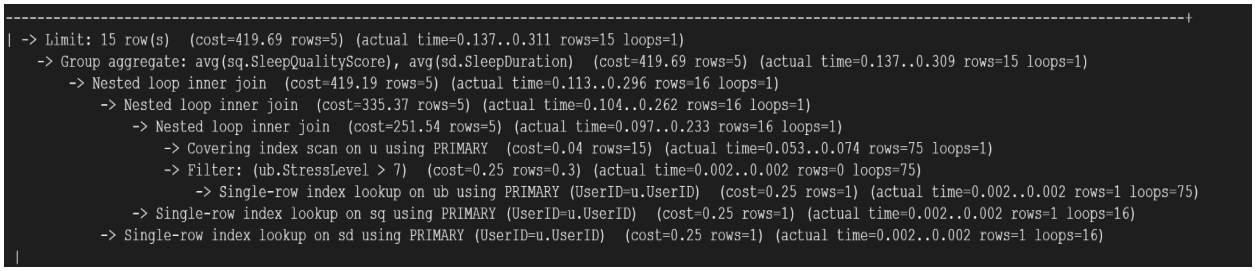
```
mysql> SELECT u.UserID, sq.SleepQualityScore, sd.BedtimeConsistency, ub.PhysicalActivityLevel
    -> FROM User u
    -> JOIN SleepQualityData sq ON u.UserID = sq.UserID
    -> JOIN SleepData sd ON sq.UserID = sd.UserID
    -> JOIN UserBehavior ub ON sd.UserID = ub.UserID
    -> WHERE sq.SleepQualityScore < 50
    ->   AND (
    ->     sd.BedtimeConsistency < (
    ->         SELECT AVG(BedtimeConsistency)
    ->         FROM SleepData
    ->     )
    ->     OR ub.PhysicalActivityLevel < '50'
    ->   )
    -> LIMIT 15;
+--------+-------------------+--------------------+----------------------+
| UserID | SleepQualityScore | BedtimeConsistency | PhysicalActivityLevel |
+--------+-------------------+--------------------+----------------------+
|      2 |                 1 | 0.1444638075259597 |                   36 |
|      4 |                 1 | 0.4532551884296549 |                   84 |
|      6 |  7.122152667218495 | 0.6553863384709657 |                   43 |
|      8 |                 1 | 0.7706907295242407 |                   39 |
|      9 | 2.4092370548842656 | 0.7693143325220593 |                   45 |
|     10 |                 1 | 0.7887340638332476 |                   45 |
|     15 |                 1 | 0.4381867526523479 |                   42 |
|     16 | 1.4858606227661448 | 0.5809646817616536 |                   35 |
|     17 |       9.080636464 | 0.4929807066430811 |                   73 |
|     18 |                 1 | 0.3292134183079888 |                   38 |
|     22 |  2.506914875172608 | 0.1749015459600334 |                   72 |
|     23 |                 1 | 0.3886442454313267 |                   49 |
|     24 |                10 | 0.5065277673381743 |                   47 |
|     28 |                 1 | 0.2745472414891705 |                   50 |
|     29 |                 1 | 0.2686117205541057 |                   33 |
+--------+-------------------+--------------------+----------------------+
15 rows in set (0.00 sec)
```

4.

```
mysql> SELECT u.UserID, r.RecommendStr, ub.StressLevel, sd.BedtimeConsistency
    -> FROM User u
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> JOIN SleepData sd ON ub.UserID = sd.UserID
    -> JOIN Recommendation r ON sd.UserID = r.UserID
    -> WHERE ub.StressLevel > 7
    ->   AND sd.BedtimeConsistency > (
    ->     SELECT AVG(BedtimeConsistency)
    ->     FROM SleepData
    -> )
    -> LIMIT 15;
+--------+--------------+-------------------+--------------------+
| UserID | RecommendStr | StressLevel       | BedtimeConsistency |
+--------+--------------+-------------------+--------------------+
|      9 |              | 7.280135692990321 | 0.7693143325220593 |
|     14 |              | 7.188359095042952 | 0.6464161902866045 |
|     27 |              | 7.143031871162077 | 0.5620945508361503 |
|     43 |              | 9.404027849319233 | 0.5921594487231935 |
|     51 |              | 7.754361873009122 | 0.6935588938006852 |
|     52 |              |       7.040073014 | 0.5797984315003392 |
|     59 |              |                10 | 0.8906214396545149 |
|     64 |              | 9.775935411049463 | 0.5234248432299703 |
|     75 |              | 8.414659619118446 | 0.6463796138468763 |
|     81 |              | 8.097278590048074 | 0.5685852754642685 |
|    125 |              | 8.060669128708822 | 0.7684360262489709 |
|    127 |              |       7.343620287 | 0.5046161317900841 |
|    128 |              | 7.996195266975569 | 0.6249807189308086 |
|    133 |              | 8.737206438326005 | 0.6184659094026687 |
|    151 |              | 8.784383521666136 | 0.6373135493506352 |
+--------+--------------+-------------------+--------------------+
15 rows in set (0.00 sec)
```

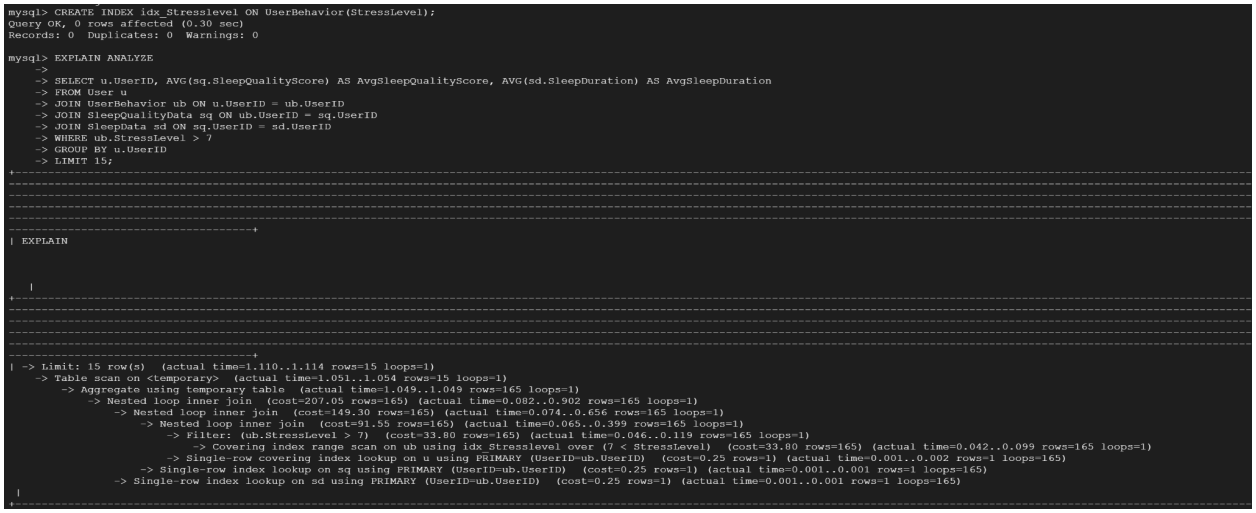
Part 2

1.For the first query, the original cost is approximately 419.69



```
| -> Limit: 15 row(s)  (cost=419.69 rows=5) (actual time=0.137..0.311 rows=15 loops=1)
    -> Group aggregate: avg(sq.SleepQualityScore), avg(sd.SleepDuration)  (cost=419.69 rows=5) (actual time=0.137..0.309 rows=15 loops=1)
        -> Nested loop inner join  (cost=419.19 rows=5) (actual time=0.113..0.296 rows=16 loops=1)
            -> Nested loop inner join  (cost=335.37 rows=5) (actual time=0.104..0.262 rows=16 loops=1)
                -> Nested loop inner join  (cost=251.54 rows=5) (actual time=0.097..0.233 rows=16 loops=1)
                    -> Covering index scan on u using PRIMARY  (cost=0.04 rows=15) (actual time=0.053..0.074 rows=75 loops=1)
                    -> Filter: (ub.StressLevel > 7)  (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=75)
                        -> Single-row index lookup on ub using PRIMARY (UserID=u.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=75)
                -> Single-row index lookup on sq using PRIMARY (UserID=u.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=16)
            -> Single-row index lookup on sd using PRIMARY (UserID=u.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=16)
 |
```

Because UserId is the primary key, only ub.StressLevel is available in the recommended attributes which we will try to index.
After indexing it, the cost is 201.05, obviously lower than the original one because this index allows the database to locate and filter data more efficiently, reducing the need for full table scans and minimizing data access time.



```
mysql> CREATE INDEX idx_Stresslevel ON UserBehavior(StressLevel);
Query OK, 0 rows affected (0.30 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    ->
    -> SELECT u.UserID, AVG(sq.SleepQualityScore) AS AvgSleepQualityScore, AVG(sd.SleepDuration) AS AvgSleepDuration
    -> FROM User u
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> JOIN SleepQualityData sq ON ub.UserID = sq.UserID
    -> JOIN SleepData sd ON sq.UserID = sd.UserID
    -> WHERE ub.StressLevel > 7
    -> GROUP BY u.UserID
    -> LIMIT 15;
+----------------------------------------+
| EXPLAIN


  |
+----------------------------------------+
| -> Limit: 15 row(s)  (actual time=1.110..1.114 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=1.051..1.054 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=1.049..1.049 rows=165 loops=1)
            -> Nested loop inner join  (cost=207.05 rows=165) (actual time=0.082..0.902 rows=165 loops=1)
                -> Nested loop inner join  (cost=149.30 rows=165) (actual time=0.074..0.656 rows=165 loops=1)
                    -> Nested loop inner join  (cost=91.55 rows=165) (actual time=0.065..0.399 rows=165 loops=1)
                        -> Filter: (ub.StressLevel > 7)  (cost=33.80 rows=165) (actual time=0.046..0.119 rows=165 loops=1)
                            -> Covering index range scan on ub using idx_Stresslevel over (7 < StressLevel)  (cost=33.80 rows=165) (actual time=0.042..0.099 rows=165 loops=1)
                        -> Single-row covering index lookup on u using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=165)
                    -> Single-row index lookup on sq using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=165)
                -> Single-row index lookup on sd using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=165)
 |
+----------------------------------------+
```

After creating a combination index on UserBehavior(StressLevel, UserID), the cost is 207.10, lower than the original one because this index allows the database to locate and filter data more efficiently, reducing the need for full table scans and minimizing data access time.

```
mysql> CREATE INDEX idx_ub_StressLevel_UserID ON UserBehavior(StressLevel, UserID);
Query OK, 0 rows affected (0.17 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, AVG(sq.SleepQualityScore) AS AvgSleepQualityScore, AVG(sd.SleepDuration) AS AvgSleepDuration
    -> FROM User u
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> JOIN SleepQualityData sq ON ub.UserID = sq.UserID
    -> JOIN SleepData sd ON sq.UserID = sd.UserID
    -> WHERE ub.StressLevel > 7
    -> GROUP BY u.UserID
    -> LIMIT 15;
+------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
-----------------------------------------------+
| EXPLAIN

   |
+------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
-----------------------------------------------+
| -> Limit: 15 row(s)  (actual time=0.970..0.972 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=0.966..0.968 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=0.964..0.964 rows=165 loops=1)
            -> Nested loop inner join  (cost=207.10 rows=165) (actual time=0.049..0.872 rows=165 loops=1)
                -> Nested loop inner join  (cost=149.35 rows=165) (actual time=0.043..0.652 rows=165 loops=1)
                    -> Nested loop inner join  (cost=91.60 rows=165) (actual time=0.037..0.419 rows=165 loops=1)
                        -> Filter: (ub.StressLevel > 7)  (cost=33.85 rows=165) (actual time=0.023..0.119 rows=165 loops=1)
                            -> Covering index range scan on ub using idx_ub_StressLevel_UserID over (7 < StressLevel)  (cost=33.85 rows=165) (actual time=0.021..0.100 rows=165 loops=1)
                        -> Single-row covering index lookup on u using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=165)
                    -> Single-row index lookup on sq using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=165)
                -> Single-row index lookup on sd using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=165)
 |
```

After indexingSleepQualityData(UserID, SleepQualityScore);
And SleepData(UserID, SleepDuration); The cost is 419.19, it is the same as the original one because these attributes are not so relevant to this query and the overhead of the index increases the cost.

```
mysql>  DROP INDEX idx_ub_StressLevel_UserID ON UserBehavior;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_sq_UserID_SleepQualityScore ON SleepQualityData(UserID, SleepQualityScore);
Query OK, 0 rows affected (0.12 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_sd_UserID_SleepDuration ON SleepData(UserID, SleepDuration);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, AVG(sq.SleepQualityScore) AS AvgSleepQualityScore, AVG(sd.SleepDuration) AS AvgSleepDuration
    -> FROM User u
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> JOIN SleepQualityData sq ON ub.UserID = sq.UserID
    -> JOIN SleepData sd ON sq.UserID = sd.UserID
    -> WHERE ub.StressLevel > 7
    -> GROUP BY u.UserID
    -> LIMIT 15;
+------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
| EXPLAIN

   |
+------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (cost=419.69 rows=5) (actual time=0.141..0.314 rows=15 loops=1)
    -> Group aggregate: avg(sq.SleepQualityScore), avg(sd.SleepDuration)  (cost=419.69 rows=5) (actual time=0.141..0.312 rows=15 loops=1)
        -> Nested loop inner join  (cost=419.19 rows=5) (actual time=0.124..0.298 rows=16 loops=1)
            -> Nested loop inner join  (cost=335.37 rows=5) (actual time=0.116..0.266 rows=16 loops=1)
                -> Nested loop inner join  (cost=251.54 rows=5) (actual time=0.109..0.233 rows=16 loops=1)
                    -> Covering index scan on u using PRIMARY  (cost=0.04 rows=15) (actual time=0.076..0.101 rows=75 loops=1)
                    -> Filter: (ub.StressLevel > 7)  (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=75)
                        -> Single-row index lookup on ub using PRIMARY (UserID=u.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=75)
                -> Single-row index lookup on sq using PRIMARY (UserID=u.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=16)
            -> Single-row index lookup on sd using PRIMARY (UserID=u.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=16)
```

So we choose to index on StressLevel because its cost is the lowest.

2.For the second query, the original cost is 267.89

```
| EXPLAIN
  |
+--------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (cost=267.89 rows=15) (actual time=1.787..2.015 rows=15 loops=1)
    -> Nested loop inner join  (cost=267.89 rows=111) (actual time=1.786..2.012 rows=15 loops=1)
        -> Nested loop inner join  (cost=151.23 rows=333) (actual time=1.702..1.812 rows=45 loops=1)
            -> Filter: (ub.CaffeineIntakeMg > (select #2))  (cost=34.58 rows=333) (actual time=1.671..1.694 rows=45 loops=1)
                -> Table scan on ub  (cost=34.58 rows=1000) (actual time=0.044..0.059 rows=85 loops=1)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(UserBehavior.CaffeineIntakeMg)  (cost=201.25 rows=1) (actual time=0.315..0.316 rows=1 loops=1)
                        -> Table scan on UserBehavior  (cost=101.25 rows=1000) (actual time=0.017..0.252 rows=1000 loops=1)
            -> Single-row index lookup on u using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=45)
        -> Filter: (h.BloodPressure > 130)  (cost=0.25 rows=0.3) (actual time=0.004..0.004 rows=0 loops=45)
            -> Single-row index lookup on h using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=45)
 |
```

After indexing both h.BloodPressure and ub.CaffeineIntakeMg, the cost becomes 145.71, which is lower because this index allows the database to locate and filter data more efficiently, reducing the need for full table scans and minimizing data access time.

```
mysql> CREATE INDEX idx_BloodPressure ON HealthData(BloodPressure);
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_CaffeineIntakeMg ON UserBehavior(CaffeineIntakeMg);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DROP INDEX idx_Stresslevel ON UserBehavior;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, u.Gender, h.BloodPressure, ub.CaffeineIntakeMg
    -> FROM User u
    -> JOIN HealthData h ON u.UserID = h.UserID
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> WHERE h.BloodPressure > 130 AND ub.CaffeineIntakeMg > (
    ->     SELECT AVG(CaffeineIntakeMg)
    ->     FROM UserBehavior
    -> )
    -> LIMIT 15;
+--------------------------------------------------------------------------------
-----------------------------------------------+
| EXPLAIN
  |
+--------------------------------------------------------------------------------
-----------------------------------------------+
| -> Limit: 15 row(s)  (cost=145.71 rows=15) (actual time=0.141..0.262 rows=15 loops=1)
    -> Nested loop inner join  (cost=145.71 rows=98) (actual time=0.140..0.260 rows=15 loops=1)
        -> Nested loop inner join  (cost=111.38 rows=98) (actual time=0.131..0.222 rows=15 loops=1)
            -> Filter: (h.BloodPressure > 130)  (cost=41.03 rows=201) (actual time=0.011..0.025 rows=25 loops=1)
                -> Covering index range scan on h using idx_BloodPressure over (130 < BloodPressure)  (cost=41.03 rows=201) (actual time=0.009..0.021 rows=25 loops=1)
            -> Filter: (ub.CaffeineIntakeMg > (select #2))  (cost=0.25 rows=0.5) (actual time=0.008..0.008 rows=1 loops=25)
                -> Single-row index lookup on ub using PRIMARY (UserID=h.UserID)  (cost=0.25 rows=1) (actual time=0.007..0.007 rows=1 loops=25)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(UserBehavior.CaffeineIntakeMg)  (cost=201.25 rows=1) (actual time=0.328..0.328 rows=1 loops=1)
                        -> Covering index scan on UserBehavior using idx_CaffeineIntakeMg  (cost=101.25 rows=1000) (actual time=0.027..0.263 rows=1000 loops=1)
        -> Single-row index lookup on u using PRIMARY (UserID=h.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
 |
```

After indexing only h.BloodPressure the cost becomes 181.73, which is lower because this index allows the database to locate and filter data more efficiently, reducing the need for full table scans and minimizing data access time.

```
mysql> DROP INDEX idx_CaffeineIntakeMg ON UserBehavior;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, u.Gender, h.BloodPressure, ub.CaffeineIntakeMg
    -> FROM User u
    -> JOIN HealthData h ON u.UserID = h.UserID
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> WHERE h.BloodPressure > 130 AND ub.CaffeineIntakeMg > (
    ->     SELECT AVG(CaffeineIntakeMg)
    ->     FROM UserBehavior
    -> )
    -> LIMIT 15;
+------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
| EXPLAIN



   |
+------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (cost=181.73 rows=15) (actual time=0.396..0.478 rows=15 loops=1)
    -> Nested loop inner join  (cost=181.73 rows=67) (actual time=0.376..0.457 rows=15 loops=1)
        -> Nested loop inner join  (cost=111.38 rows=201) (actual time=0.050..0.108 rows=25 loops=1)
            -> Filter: (h.BloodPressure > 130)  (cost=41.03 rows=201) (actual time=0.034..0.047 rows=25 loops=1)
                -> Covering index range scan on h using idx_BloodPressure over (130 < BloodPressure)  (cost=41.03 rows=201) (actual time=0.020..0.031 rows=25 loops=1)
            -> Single-row index lookup on u using PRIMARY (UserID=h.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=25)
        -> Filter: (ub.CaffeineIntakeMg > (select #2))  (cost=0.25 rows=0.3) (actual time=0.014..0.014 rows=1 loops=25)
            -> Single-row index lookup on ub using PRIMARY (UserID=h.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=25)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(UserBehavior.CaffeineIntakeMg)  (cost=201.25 rows=1) (actual time=0.276..0.276 rows=1 loops=1)
                    -> Table scan on UserBehavior  (cost=101.25 rows=1000) (actual time=0.019..0.212 rows=1000 loops=1)
```

After indexing only ub.CaffeineIntakeMg, the cost becomes 275.08, similar to the original one because this attribute is not the dominant part ,such as join and where, that affects the query and it introduces some overheads.

```
mysql> CREATE INDEX idx_CaffeineIntakeMg ON UserBehavior(CaffeineIntakeMg);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DROP INDEX idx_CaffeineIntakeMg ON HealthData;
ERROR 1091 (42000): Can't DROP 'idx_CaffeineIntakeMg'; check that column/key exists
mysql> DROP INDEX idx_BloodPressure ON HealthData;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    ->
    ->
    ->
    ->
    -> SELECT u.UserID, u.Gender, h.BloodPressure, ub.CaffeineIntakeMg
    -> FROM User u
    -> JOIN HealthData h ON u.UserID = h.UserID
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> WHERE h.BloodPressure > 130 AND ub.CaffeineIntakeMg > (
    ->     SELECT AVG(CaffeineIntakeMg)
    ->     FROM UserBehavior
    -> )
    -> LIMIT 15;
+------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
| EXPLAIN



   |
+------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (cost=275.08 rows=15) (actual time=0.050..0.119 rows=15 loops=1)
    -> Nested loop inner join  (cost=275.08 rows=163) (actual time=0.049..0.116 rows=15 loops=1)
        -> Nested loop inner join  (cost=218.15 rows=163) (actual time=0.041..0.087 rows=15 loops=1)
            -> Filter: (h.BloodPressure > 130)  (cost=101.50 rows=333) (actual time=0.027..0.040 rows=24 loops=1)
                -> Table scan on h  (cost=101.50 rows=1000) (actual time=0.025..0.032 rows=85 loops=1)
            -> Filter: (ub.CaffeineIntakeMg > (select #2))  (cost=0.25 rows=0.5) (actual time=0.002..0.002 rows=1 loops=24)
                -> Single-row index lookup on ub using PRIMARY (UserID=h.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=24)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(UserBehavior.CaffeineIntakeMg)  (cost=201.25 rows=1) (actual time=0.314..0.314 rows=1 loops=1)
                        -> Covering index scan on UserBehavior using idx_CaffeineIntakeMg  (cost=101.25 rows=1000) (actual time=0.077..0.247 rows=1000 loops=1)
```

So we choose to index on both of these two attributes because its cost is the lowest.

3..For the second query, the original cost is 451.21

```
mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, sq.SleepQualityScore, sd.BedtimeConsistency, ub.PhysicalActivityLevel
    -> FROM User u
    -> JOIN SleepQualityData sq ON u.UserID = sq.UserID
    -> JOIN SleepData sd ON sq.UserID = sd.UserID
    -> JOIN UserBehavior ub ON sd.UserID = ub.UserID
    -> WHERE sq.SleepQualityScore < 50
    ->   AND (
    ->     sd.BedtimeConsistency < (
    ->         SELECT AVG(BedtimeConsistency)
    ->         FROM SleepData
    ->     )
    ->     OR ub.PhysicalActivityLevel < '50'
    ->   )
    -> LIMIT 15;
| EXPLAIN
| -> Limit: 15 row(s)  (cost=451.21 rows=15) (actual time=0.418..0.539 rows=15 loops=1)
    -> Nested loop inner join  (cost=451.21 rows=333) (actual time=0.417..0.537 rows=15 loops=1)
        -> Nested loop inner join  (cost=334.56 rows=333) (actual time=0.090..0.175 rows=29 loops=1)
            -> Nested loop inner join  (cost=217.90 rows=333) (actual time=0.060..0.109 rows=29 loops=1)
                -> Filter: (sq.SleepQualityScore < 50)  (cost=101.25 rows=333) (actual time=0.047..0.053 rows=29 loops=1)
                    -> Table scan on sq  (cost=101.25 rows=1000) (actual time=0.045..0.048 rows=29 loops=1)
                -> Single-row covering index lookup on u using PRIMARY (UserID=sq.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=29)
            -> Single-row index lookup on sd using PRIMARY (UserID=sq.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=29)
        -> Filter: ((sd.BedtimeConsistency < (select #2)) or (ub.PhysicalActivityLevel < 50))  (cost=0.25 rows=1) (actual time=0.012..0.012 rows=1 loops=29)
            -> Single-row index lookup on ub using PRIMARY (UserID=sq.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=29)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(SleepData.BedtimeConsistency)  (cost=201.25 rows=1) (actual time=0.290..0.290 rows=1 loops=1)
                    -> Table scan on SleepData  (cost=101.25 rows=1000) (actual time=0.023..0.222 rows=1000 loops=1)
```

After indexing on SleepData(BedtimeConsistency) the cost is 451.21, the same as the original one because this attribute is not the dominant part, including join and where, that affects the query.

```
mysql> CREATE INDEX idx_sd_BedtimeConsistency ON SleepData(BedtimeConsistency);
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, sq.SleepQualityScore, sd.BedtimeConsistency, ub.PhysicalActivityLevel
    -> FROM User u
    -> JOIN SleepQualityData sq ON u.UserID = sq.UserID
    -> JOIN SleepData sd ON sq.UserID = sd.UserID
    -> JOIN UserBehavior ub ON sd.UserID = ub.UserID
    -> WHERE sq.SleepQualityScore < 50
    ->   AND (
    ->     sd.BedtimeConsistency < (
    ->         SELECT AVG(BedtimeConsistency)
    ->         FROM SleepData
    ->     )
    ->     OR ub.PhysicalActivityLevel < '50'
    ->   )
    -> LIMIT 15;
| EXPLAIN
| -> Limit: 15 row(s)  (cost=451.21 rows=15) (actual time=0.061..0.186 rows=15 loops=1)
    -> Nested loop inner join  (cost=451.21 rows=333) (actual time=0.061..0.183 rows=15 loops=1)
        -> Nested loop inner join  (cost=334.56 rows=333) (actual time=0.047..0.132 rows=29 loops=1)
            -> Nested loop inner join  (cost=217.90 rows=333) (actual time=0.041..0.089 rows=29 loops=1)
                -> Filter: (sq.SleepQualityScore < 50)  (cost=101.25 rows=333) (actual time=0.028..0.034 rows=29 loops=1)
                    -> Table scan on sq  (cost=101.25 rows=1000) (actual time=0.026..0.029 rows=29 loops=1)
                -> Single-row covering index lookup on u using PRIMARY (UserID=sq.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=29)
            -> Single-row index lookup on sd using PRIMARY (UserID=sq.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=29)
        -> Filter: ((sd.BedtimeConsistency < (select #2)) or (ub.PhysicalActivityLevel < 50))  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=29)
            -> Single-row index lookup on ub using PRIMARY (UserID=sq.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=29)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(SleepData.BedtimeConsistency)  (cost=201.25 rows=1) (actual time=0.345..0.346 rows=1 loops=1)
                    -> Covering index scan on SleepData using idx_sd_BedtimeConsistency  (cost=101.25 rows=1000) (actual time=0.037..0.270 rows=1000 loops=1)
1 row in set (0.01 sec)
```

After indexing on UserBehavior(PhysicalActivityLevel) the cost is 451.21, the same as the original one because this attribute is not the dominant part that affects the query.

```
mysql> CREATE INDEX idx_ub_PhysicalActivityLevel ON UserBehavior(PhysicalActivityLevel);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DROP INDEX idx_sd_BedtimeConsistency ON SleepData;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, sq.SleepQualityScore, sd.BedtimeConsistency, ub.PhysicalActivityLevel
    -> FROM User u
    -> JOIN SleepQualityData sq ON u.UserID = sq.UserID
    -> JOIN SleepData sd ON sq.UserID = sd.UserID
    -> JOIN UserBehavior ub ON sd.UserID = ub.UserID
    -> WHERE sq.SleepQualityScore < 50
    ->   AND (
    ->     sd.BedtimeConsistency < (
    ->         SELECT AVG(BedtimeConsistency)
    ->         FROM SleepData
    ->     )
    ->     OR ub.PhysicalActivityLevel < '50'
    ->   )
    -> LIMIT 15;
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
| EXPLAIN



   |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (cost=451.21 rows=15) (actual time=0.356..0.476 rows=15 loops=1)
    -> Nested loop inner join  (cost=451.21 rows=333) (actual time=0.355..0.474 rows=15 loops=1)
        -> Nested loop inner join  (cost=334.56 rows=333) (actual time=0.068..0.149 rows=29 loops=1)
            -> Nested loop inner join  (cost=217.90 rows=333) (actual time=0.061..0.106 rows=29 loops=1)
                -> Filter: (sq.SleepQualityScore < 50)  (cost=101.25 rows=333) (actual time=0.047..0.053 rows=29 loops=1)
                    -> Table scan on sq  (cost=101.25 rows=1000) (actual time=0.045..0.048 rows=29 loops=1)
                -> Single-row covering index lookup on u using PRIMARY (UserID=sq.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=29)
            -> Single-row index lookup on sd using PRIMARY (UserID=sq.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=29)
        -> Filter: ((sd.BedtimeConsistency < (select #2)) or (ub.PhysicalActivityLevel < 50))  (cost=0.25 rows=1) (actual time=0.011..0.011 rows=1 loops=29)
            -> Single-row index lookup on ub using PRIMARY (UserID=sq.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=29)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(SleepData.BedtimeConsistency)  (cost=201.25 rows=1) (actual time=0.263..0.263 rows=1 loops=1)
                    -> Table scan on SleepData  (cost=101.25 rows=1000) (actual time=0.017..0.202 rows=1000 loops=1)
 |
```

AFTER indexing on SleepQualityScore(SleepQualityScore), the cost is 1151.25, extremely larger than the original one. Adding the single-column index on SleepQualityData(SleepQualityScore) increased the cost because it didn't effectively optimize the complex multi-table JOIN and subquery operations; instead, it added overhead for index scanning. A better optimization might be a composite index on (UserID, SleepQualityScore) in SleepQualityData to improve both JOINs and filtering conditions.

```
mysql> DROP INDEX idx_ub_PhysicalActivityLevel ON UserBehavior;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx ON SleepQualityScore(SleepQualityScore);
ERROR 1146 (42S02): Table 'sleep.SleepQualityScore' doesn't exist
mysql> CREATE INDEX idx ON SleepQualityData(SleepQualityScore);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, sq.SleepQualityScore, sd.BedtimeConsistency, ub.PhysicalActivityLevel
    -> FROM User u
    -> JOIN SleepQualityData sq ON u.UserID = sq.UserID
    -> JOIN SleepData sd ON sq.UserID = sd.UserID
    -> JOIN UserBehavior ub ON sd.UserID = ub.UserID
    -> WHERE sq.SleepQualityScore < 50
    ->   AND (
    ->     sd.BedtimeConsistency < (
    ->         SELECT AVG(BedtimeConsistency)
    ->         FROM SleepData
    ->     )
    ->     OR ub.PhysicalActivityLevel < '50'
    ->   )
    -> LIMIT 15;
+-----------------------------------------------------------------------------------------------
| EXPLAIN

   |
+-----------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (cost=1151.25 rows=15) (actual time=0.384..0.506 rows=15 loops=1)
    -> Nested loop inner join  (cost=1151.25 rows=1000) (actual time=0.383..0.504 rows=15 loops=1)
        -> Nested loop inner join  (cost=801.25 rows=1000) (actual time=0.051..0.135 rows=29 loops=1)
            -> Nested loop inner join  (cost=451.25 rows=1000) (actual time=0.046..0.094 rows=29 loops=1)
                -> Covering index scan on u using PRIMARY  (cost=101.25 rows=1000) (actual time=0.032..0.035 rows=29 loops=1)
                -> Filter: (sq.SleepQualityScore < 50)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=29)
                    -> Single-row index lookup on sq using PRIMARY (UserID=u.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=29)
            -> Single-row index lookup on sd using PRIMARY (UserID=u.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=29)
        -> Filter: ((sd.BedtimeConsistency < (select #2)) or (ub.PhysicalActivityLevel < 50))  (cost=0.25 rows=1) (actual time=0.013..0.013 rows=1 loops=29)
            -> Single-row index lookup on ub using PRIMARY (UserID=u.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=29)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(SleepData.BedtimeConsistency)  (cost=201.25 rows=1) (actual time=0.306..0.306 rows=1 loops=1)
                    -> Table scan on SleepData  (cost=101.25 rows=1000) (actual time=0.018..0.240 rows=1000 loops=1)
```

So we will just use the default index because its cost is the lowest.

4.

For the 4th query, the original cost is 373.44

```
mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, r.RecommendStr, ub.StressLevel, sd.BedtimeConsistency
    -> FROM User u
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> JOIN SleepData sd ON ub.UserID = sd.UserID
    -> JOIN Recommendation r ON sd.UserID = r.UserID
    -> WHERE ub.StressLevel > 7
    ->   AND sd.BedtimeConsistency > (
    ->     SELECT AVG(BedtimeConsistency)
    ->     FROM SleepData
    -> )
    -> LIMIT 15;
+-----------------------------------------------------------------------------------------------
| EXPLAIN

   |
+-----------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (cost=373.44 rows=15) (actual time=0.367..1.063 rows=15 loops=1)
    -> Nested loop inner join  (cost=373.44 rows=111) (actual time=0.366..1.060 rows=15 loops=1)
        -> Nested loop inner join  (cost=334.56 rows=111) (actual time=0.352..0.490 rows=15 loops=1)
            -> Nested loop inner join  (cost=217.90 rows=333) (actual time=0.065..0.156 rows=28 loops=1)
                -> Filter: (ub.StressLevel > 7)  (cost=101.25 rows=333) (actual time=0.051..0.093 rows=28 loops=1)
                    -> Table scan on ub  (cost=101.25 rows=1000) (actual time=0.047..0.080 rows=151 loops=1)
                -> Single-row covering index lookup on u using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=28)
            -> Filter: (sd.BedtimeConsistency > (select #2))  (cost=0.25 rows=0.3) (actual time=0.012..0.012 rows=1 loops=28)
                -> Single-row index lookup on sd using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=28)
                -> Select #2 (subquery in condition; run only once)
                    -> Aggregate: avg(SleepData.BedtimeConsistency)  (cost=201.25 rows=1) (actual time=0.270..0.270 rows=1 loops=1)
                        -> Table scan on SleepData  (cost=101.25 rows=1000) (actual time=0.015..0.197 rows=1000 loops=1)
        -> Single-row index lookup on r using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.038..0.038 rows=1 loops=15)
 |
+-----------------------------------------------------------------------------------------------
```

After indexing on UserBehavior(StressLevel), the cost is 130.04, which is lower than the original one because this index allows the database to locate and filter data more efficiently, reducing the need for full table scans and minimizing data access time.

```
mysql> CREATE INDEX idx_0 ON UserBehavior(StressLevel);
Query OK, 0 rows affected (0.30 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, r.RecommendStr, ub.StressLevel, sd.BedtimeConsistency
    -> FROM User u
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> JOIN SleepData sd ON ub.UserID = sd.UserID
    -> JOIN Recommendation r ON sd.UserID = r.UserID
    -> WHERE ub.StressLevel > 7
    ->   AND sd.BedtimeConsistency > (
    ->     SELECT AVG(BedtimeConsistency)
    ->     FROM SleepData
    -> )
    -> LIMIT 15;
+-----------------------------------------------------------------------------------------------------------
-------------------+
| EXPLAIN

   |
+-----------------------------------------------------------------------------------------------------------
-------------------+
| -> Limit: 15 row(s)  (cost=130.04 rows=15) (actual time=1.130..1.364 rows=15 loops=1)
    -> Nested loop inner join  (cost=130.04 rows=55) (actual time=1.127..1.360 rows=15 loops=1)
        -> Nested loop inner join  (cost=110.80 rows=55) (actual time=1.109..1.293 rows=15 loops=1)
            -> Nested loop inner join  (cost=91.55 rows=55) (actual time=1.046..1.186 rows=15 loops=1)
                -> Filter: (ub.StressLevel > 7)  (cost=33.80 rows=165) (actual time=0.076..0.104 rows=35 loops=1)
                    -> Covering index range scan on ub using idx_0 over (7 < StressLevel)  (cost=33.80 rows=165) (actual time=0.065..0.088 rows=35 loops=1)
                -> Filter: (sd.BedtimeConsistency > (select #2))  (cost=0.25 rows=0.3) (actual time=0.031..0.031 rows=0 loops=35)
                    -> Single-row index lookup on sd using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=35)
                    -> Select #2 (subquery in condition; run only once)
                        -> Aggregate: avg(SleepData.BedtimeConsistency)  (cost=201.25 rows=1) (actual time=0.323..0.324 rows=1 loops=1)
                            -> Table scan on SleepData  (cost=101.25 rows=1000) (actual time=0.026..0.250 rows=1000 loops=1)
            -> Single-row index lookup on r using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.007..0.007 rows=1 loops=15)
        -> Single-row covering index lookup on u using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)
```

After indexing on SleepData(BedtimeConsistency), the cost is 332, larger than the original one because although it optimized the subquery's AVG(BedtimeConsistency) calculation, the overall cost increased likely due to the extra index scan overhead. In complex queries, the optimizer may sometimes increase total cost because the new index introduces additional random I/O operations.

```
mysql> CREATE INDEX idx_0 ON SleepData(BedtimeConsistency);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, r.RecommendStr, ub.StressLevel, sd.BedtimeConsistency
    -> FROM User u
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> JOIN SleepData sd ON ub.UserID = sd.UserID
    -> JOIN Recommendation r ON sd.UserID = r.UserID
    -> WHERE ub.StressLevel > 7
    ->   AND sd.BedtimeConsistency > (
    ->     SELECT AVG(BedtimeConsistency)
    ->     FROM SleepData
    -> )
    -> LIMIT 15;
+-----------------------------------------------------------------------------------------------------------
| EXPLAIN

   |
+-----------------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (cost=332.46 rows=15) (actual time=0.119..0.281 rows=15 loops=1)
    -> Nested loop inner join  (cost=332.46 rows=164) (actual time=0.118..0.278 rows=15 loops=1)
        -> Nested loop inner join  (cost=275.18 rows=164) (actual time=0.098..0.226 rows=15 loops=1)
            -> Nested loop inner join  (cost=217.90 rows=164) (actual time=0.085..0.183 rows=15 loops=1)
                -> Filter: (ub.StressLevel > 7)  (cost=101.25 rows=333) (actual time=0.059..0.098 rows=28 loops=1)
                    -> Table scan on ub  (cost=101.25 rows=1000) (actual time=0.049..0.080 rows=151 loops=1)
                -> Filter: (sd.BedtimeConsistency > (select #2))  (cost=0.25 rows=0.5) (actual time=0.003..0.003 rows=1 loops=28)
                    -> Single-row index lookup on sd using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=28)
                    -> Select #2 (subquery in condition; run only once)
                        -> Aggregate: avg(SleepData.BedtimeConsistency)  (cost=201.25 rows=1) (actual time=0.423..0.423 rows=1 loops=1)
                            -> Covering index scan on SleepData using idx_0  (cost=101.25 rows=1000) (actual time=0.077..0.342 rows=1000 loops=1)
            -> Single-row index lookup on r using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=15)
        -> Single-row covering index lookup on u using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=15)
```

After indexing on both of the two attributes above, the cost is 148.26, because indexing on the StressLevel decreases the cost but indexing on BedtimeConsistency increases the cost, so it is a little bit larger than the first case but smaller than the original one.

```
Database changed
mysql> CREATE INDEX idx_0 ON UserBehavior(StressLevel);
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_0 ON SleepData(BedtimeConsistency);
ERROR 1061 (42000): Duplicate key name 'idx_0'
mysql> EXPLAIN ANALYZE
    -> SELECT u.UserID, r.RecommendStr, ub.StressLevel, sd.BedtimeConsistency
    -> FROM User u
    -> JOIN UserBehavior ub ON u.UserID = ub.UserID
    -> JOIN SleepData sd ON ub.UserID = sd.UserID
    -> JOIN Recommendation r ON sd.UserID = r.UserID
    -> WHERE ub.StressLevel > 7
    ->   AND sd.BedtimeConsistency > (
    ->     SELECT AVG(BedtimeConsistency)
    ->     FROM SleepData
    -> )
    -> LIMIT 15;
+--------------------------------------------------------------------------------
----------------------------------------+
| EXPLAIN

   |
+--------------------------------------------------------------------------------
----------------------------------------+
| -> Limit: 15 row(s)  (cost=148.26 rows=15) (actual time=0.076..0.362 rows=15 loops=1)
    -> Nested loop inner join  (cost=148.26 rows=81) (actual time=0.074..0.359 rows=15 loops=1)
        -> Nested loop inner join  (cost=119.90 rows=81) (actual time=0.062..0.276 rows=15 loops=1)
            -> Nested loop inner join  (cost=91.55 rows=81) (actual time=0.050..0.206 rows=15 loops=1)
                -> Filter: (ub.StressLevel > 7)  (cost=33.80 rows=165) (actual time=0.032..0.051 rows=35 loops=1)
                    -> Covering index range scan on ub using idx_0 over (7 < StressLevel)  (cost=33.80 rows=165) (actual time=0.029..0.045 rows=35 loops=1)
                -> Filter: (sd.BedtimeConsistency > (select #2))  (cost=0.25 rows=0.5) (actual time=0.004..0.004 rows=0 loops=35)
                    -> Single-row index lookup on sd using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=35)
                    -> Select #2 (subquery in condition; run only once)
                        -> Aggregate: avg(SleepData.BedtimeConsistency)  (cost=201.25 rows=1) (actual time=0.339..0.339 rows=1 loops=1)
                            -> Covering index scan on SleepData using idx_0  (cost=101.25 rows=1000) (actual time=0.046..0.239 rows=1000 loops=1)
            -> Single-row index lookup on r using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=15)
        -> Single-row covering index lookup on u using PRIMARY (UserID=ub.UserID)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=15)
 |
```

So we will choose to index on UserBehavior(StressLevel) because its cost is the lowest.