Stage 3: Database Implementation and indexing

Database Implementation on GCP

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| healthconnect      |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> use healthconnect
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_healthconnect      |
+------------------------------+
| Doctor                       |
| Fitness                      |
| MedicalRecord                |
| Patient                      |
| Prescription                 |
| Review                       |
| Treat                        |
| User                         |
| staging_dailyActivity_merged |
| staging_demographic          |
| staging_diet                 |
| staging_examination          |
| staging_labs                 |
| staging_medications          |
| staging_sleepDay_merged      |
+------------------------------+
15 rows in set (0.01 sec)

mysql>
```

Note: All the staging tables aren't part of the tables in our UML/ database - they are just used to populate each of our entities with the appropriate data and we will drop those tables once all the entities(tables) have been populated.

DDL for Tables

```
CREATE TABLE User (
   userID INT PRIMARY KEY,
   userName VARCHAR(20),
   password VARCHAR(20),
   role INT
);


CREATE TABLE Patient (
   patientID INT PRIMARY KEY,
   gender INT,
   ageInYears INT,
   ageInMonths INT,
   pregnancyStatus INT
);

CREATE TABLE Doctor (
   docID INT PRIMARY KEY,
   docName VARCHAR(30),
   specialization VARCHAR(30)
);

CREATE TABLE MedicalRecord (
   recordID INT PRIMARY KEY,
   date DATE,
   historyQuestions VARCHAR(30),
   medications VARCHAR(30),
   symptomName VARCHAR(30),
   patientID INT,
   FOREIGN KEY (patientID) REFERENCES Patient(patientID) ON DELETE
CASCADE
);
```

```
CREATE TABLE Prescription (
   prescriptionID INT PRIMARY KEY,
   medicineName VARCHAR(30),
   dosage VARCHAR(30),
   startDate DATE,
   endDate DATE,
   patientID INT,
   docID INT,
   recordID INT,
   FOREIGN KEY (patientID) REFERENCES Patient(patientID) ON DELETE
CASCADE,
   FOREIGN KEY (docID) REFERENCES Doctor(docID) ON DELETE CASCADE,
   FOREIGN KEY (recordID) REFERENCES MedicalRecord(recordID) ON DELETE
CASCADE
);

CREATE TABLE Fitness (
   fitnessID INT PRIMARY KEY,
   caloriesBurned REAL,
   steps INT,
   sleepDuration REAL,
   date DATE,
   patientID INT,
   FOREIGN KEY (patientID) REFERENCES Patient(patientID) ON DELETE
CASCADE
);

CREATE TABLE Treat (
   docID INT,
   patientID INT,
   PRIMARY KEY (docID, patientID),
   FOREIGN KEY (docID) REFERENCES Doctor(docID) ON DELETE CASCADE,
   FOREIGN KEY (patientID) REFERENCES Patient(patientID) ON DELETE
CASCADE
);
```

```
CREATE TABLE Review (
   docID INT,
   recordID INT,
   PRIMARY KEY (docID, recordID),
   FOREIGN KEY (docID) REFERENCES Doctor(docID) ON DELETE CASCADE,
   FOREIGN KEY (recordID) REFERENCES MedicalRecord(recordID) ON DELETE
CASCADE
);
```

Tables and data count

1. User

```
mysql> describe User;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| userID   | int         | NO   | PRI | NULL    |       |
| userName | varchar(20) | YES  |     | NULL    |       |
| password | varchar(20) | YES  |     | NULL    |       |
| role     | int         | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM User;
+----------+
| COUNT(*) |
+----------+
|     1523 |
+----------+
1 row in set (0.02 sec)

mysql>
```

2. Doctor

```
mysql> describe Doctor;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| docID          | int         | NO   | PRI | NULL    |       |
| docName        | varchar(30) | YES  |     | NULL    |       |
| specialization | varchar(30) | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Doctor;
+----------+
| COUNT(*) |
+----------+
|      627 |
+----------+
1 row in set (0.02 sec)

mysql>
```

3. Patient

```
mysql> describe Patient;
+-----------------+------+------+-----+---------+-------+
| Field           | Type | Null | Key | Default | Extra |
+-----------------+------+------+-----+---------+-------+
| patientID       | int  | NO   | PRI | NULL    |       |
| gender          | int  | YES  |     | NULL    |       |
| ageInYears      | int  | YES  |     | NULL    |       |
| ageInMonths     | int  | YES  |     | NULL    |       |
| pregnancyStatus | int  | YES  |     | NULL    |       |
+-----------------+------+------+-----+---------+-------+
5 rows in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Patient;
+----------+
| COUNT(*) |
+----------+
|     1100 |
+----------+
1 row in set (0.01 sec)

mysql>
```

4. Fitness

```
mysql> describe Fitness;
+----------------+------+------+-----+---------+----------------+
| Field          | Type | Null | Key | Default | Extra          |
+----------------+------+------+-----+---------+----------------+
| fitnessID      | int  | NO   | PRI | NULL    | auto_increment |
| caloriesBurned | int  | YES  |     | NULL    |                |
| steps          | int  | YES  |     | NULL    |                |
| sleepDuration  | int  | YES  |     | NULL    |                |
| date           | date | YES  |     | NULL    |                |
| patientID      | int  | YES  | MUL | NULL    |                |
+----------------+------+------+-----+---------+----------------+
6 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Fitness;
+----------+
| COUNT(*) |
+----------+
|    12441 |
+----------+
1 row in set (0.00 sec)

mysql>
```

5. Treat

```
mysql> describe Treat;
+-----------+------+------+-----+---------+-------+
| Field     | Type | Null | Key | Default | Extra |
+-----------+------+------+-----+---------+-------+
| docID     | int  | NO   | PRI | NULL    |       |
| patientID | int  | NO   | PRI | NULL    |       |
+-----------+------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Treat;
+----------+
| COUNT(*) |
+----------+
|     1100 |
+----------+
1 row in set (0.00 sec)

mysql>
```

6. MedicalRecord

```
mysql> describe MedicalRecord;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| recordID         | int         | NO   | PRI | NULL    |       |
| date             | date        | YES  |     | NULL    |       |
| historyQuestions | varchar(30) | YES  |     | NULL    |       |
| medications      | varchar(30) | YES  |     | NULL    |       |
| symptomName      | varchar(30) | YES  |     | NULL    |       |
| patientID        | int         | YES  | MUL | NULL    |       |
+------------------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)

mysql> SELECT COUNT(*) FROM MedicalRecord;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.01 sec)

mysql>
```

7. Prescription

```
mysql> describe Prescription;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| prescriptionID | int         | NO   | PRI | NULL    |       |
| medicineName   | varchar(30) | YES  |     | NULL    |       |
| dosage         | varchar(30) | YES  |     | NULL    |       |
| startDate      | date        | YES  |     | NULL    |       |
| endDate        | date        | YES  |     | NULL    |       |
| patientID      | int         | YES  | MUL | NULL    |       |
| docID          | int         | YES  | MUL | NULL    |       |
| recordID       | int         | YES  | MUL | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
8 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Prescription;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.01 sec)

mysql>
```

8. Review

```
mysql> describe Review;
+----------+------+------+-----+---------+-------+
| Field    | Type | Null | Key | Default | Extra |
+----------+------+------+-----+---------+-------+
| docID    | int  | NO   | PRI | NULL    |       |
| recordID | int  | NO   | PRI | NULL    |       |
+----------+------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Review;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)

mysql>
```

The requirement for this stage was to “implement at least 5 main tables” and insert at least 1000 rows in three different tables→ We implemented all 8 tables from our DDL commands, and inserted more than 1000 rows in 4 of our main tables as shown in the above screenshots (User, Patient, Fitness, and Treat).

Advanced Queries

- **Query 1:** List all doctors along with the total number of patients assigned to each, ordered by the number of patients in descending order.

  ```
  SELECT d.docID, d.docName, COUNT(p.patientID) AS total_patients
  FROM Doctor d
  JOIN Treat t ON d.docID = t.docID
  JOIN Patient p ON t.patientID = p.patientID
  GROUP BY d.docID, d.docName
  ORDER BY total_patients DESC, d.docID
  LIMIT 15;
  ```

```
mysql> SELECT d.docID, d.docName, COUNT(p.patientID) AS total_patients
    -> FROM Doctor d
    -> JOIN Treat t ON d.docID = t.docID
    -> JOIN Patient p ON t.patientID = p.patientID
    -> GROUP BY d.docID, d.docName
    -> ORDER BY total_patients DESC, d.docID
    -> LIMIT 15;
+-------+----------------------+----------------+
| docID | docName              | total_patients |
+-------+----------------------+----------------+
|   325 | Rudd Zukerman        |              7 |
|   481 | Evey Organer         |              7 |
|   544 | Zebulen Wane         |              7 |
|   412 | Olag Chaytor         |              6 |
|    94 | Anthiathia Carville  |              5 |
|   147 | Blinnie Terbrugge    |              5 |
|   179 | Conny Dilawey        |              5 |
|   242 | Alejandrina Radcliffe|              5 |
|   270 | Meggi Ohrt           |              5 |
|   309 | Obadiah Radmer       |              5 |
|   385 | Rycca Barefoot       |              5 |
|   398 | Sanders Cornill      |              5 |
|   455 | Velma Joseph         |              5 |
|   490 | Garv Monkhouse       |              5 |
|   572 | Kristoffer Fernan    |              5 |
+-------+----------------------+----------------+
15 rows in set (0.01 sec)

mysql>
```

- **Query 2:** List the average sleep duration of patients for each doctor specializing in "Neurologist"

  SELECT d.docID, d.docName, AVG(f.sleepDuration) AS avg_sleep_duration
  FROM Doctor d
  JOIN Treat t ON d.docID = t.docID
  JOIN Patient p ON t.patientID = p.patientID
  JOIN Fitness f ON p.patientID = f.patientID
  WHERE d.specialization = 'Neurologist'
  GROUP BY d.docID, d.docName
  LIMIT 15;

```
mysql> SELECT d.docID, d.docName, AVG(f.sleepDuration) AS avg_sleep_duration
    -> FROM Doctor d
    -> JOIN Treat t ON d.docID = t.docID
    -> JOIN Patient p ON t.patientID = p.patientID
    -> JOIN Fitness f ON p.patientID = f.patientID
    -> WHERE d.specialization = 'Neurologist'
    -> GROUP BY d.docID, d.docName LIMIT 15;
+-------+--------------------+--------------------+
| docID | docName            | avg_sleep_duration |
+-------+--------------------+--------------------+
|    14 | Fina Dufore        |           464.7500 |
|    28 | Sindee Medler      |           431.3333 |
|    40 | Val Arro           |           435.3333 |
|    41 | Fairfax Lamburne   |           527.0000 |
|    44 | Alene Crucetti     |           460.1667 |
|    47 | Brod Bountiff      |           394.5000 |
|    77 | Neila Woliter      |           464.0000 |
|    80 | Roth Slarke        |           534.5000 |
|    84 | Lawrence Cundict   |           437.0000 |
|    94 | Anthiathia Carville |           468.0000 |
|   102 | Skyler Limpenny    |           409.6667 |
|   138 | Tybi Larter        |           452.0000 |
|   145 | Willi Sansbury     |           413.5000 |
|   149 | Aundrea Turbayne   |           440.6000 |
|   167 | Jennifer Pywell    |           342.0000 |
+-------+--------------------+--------------------+
15 rows in set (0.00 sec)
```

- **Query 3:** Get the total number of fitness records for each patient who has logged at least two high-activity sessions, defined as having over 5,000 steps and more than 300 calories burned.

  SELECT p.patientID, COUNT(f.fitnessID) AS total_fitness_records
  FROM Patient p
  JOIN Fitness f ON p.patientID = f.patientID
  WHERE f.steps > 5000 AND f.caloriesBurned > 300
  GROUP BY p.patientID
  HAVING COUNT(f.fitnessID) >= 2
  LIMIT 15;

```
mysql> SELECT p.patientID, COUNT(f.fitnessID) AS total_fitness_records
    -> FROM Patient p
    -> JOIN Fitness f ON p.patientID = f.patientID
    -> WHERE f.steps > 5000 AND f.caloriesBurned > 300
    -> GROUP BY p.patientID
    -> HAVING COUNT(f.fitnessID) >= 2
    -> LIMIT 15;
+-----------+-----------------------+
| patientID | total_fitness_records |
+-----------+-----------------------+
|     74641 |                     2 |
|     73889 |                     2 |
|     73717 |                     2 |
|     73819 |                     2 |
|     73688 |                     2 |
|     73672 |                     2 |
|     74127 |                     2 |
|     74498 |                     2 |
|     74280 |                     3 |
|     73947 |                     2 |
|     73575 |                     3 |
|     73969 |                     2 |
|     73841 |                     2 |
|     74008 |                     3 |
|     74286 |                     2 |
+-----------+-----------------------+
15 rows in set (0.01 sec)

mysql>
```

- **Query 4**: Identify patients with an average calorie burn above the system-wide average.

```
SELECT p.patientID, p.gender, AVG(f.caloriesBurned) AS avg_calories_burned
FROM Patient p
JOIN Fitness f ON p.patientID = f.patientID
GROUP BY p.patientID, p.gender
HAVING (AVG(f.caloriesBurned) > (
   SELECT AVG(f2.caloriesBurned)
   FROM Fitness f2
))
LIMIT 15;
```

```
mysql> SELECT p.patientID, p.gender, AVG(f.caloriesBurned) AS avg_calories_burned
    -> FROM Patient p
    -> JOIN Fitness f ON p.patientID = f.patientID
    -> GROUP BY p.patientID, p.gender
    -> HAVING (AVG(f.caloriesBurned) > (
    ->     SELECT AVG(f2.caloriesBurned)
    ->     FROM Fitness f2
    -> ))
    -> LIMIT 15;
+-----------+--------+---------------------+
| patientID | gender | avg_calories_burned |
+-----------+--------+---------------------+
|     73558 |      1 |           4092.0000 |
|     73559 |      1 |           3644.0000 |
|     73560 |      1 |           3236.5000 |
|     73561 |      2 |           4236.0000 |
|     73567 |      1 |           3109.3333 |
|     73573 |      1 |           3721.0000 |
|     73574 |      2 |           3025.3333 |
|     73579 |      2 |           4005.0000 |
|     73580 |      2 |           3369.0000 |
|     73583 |      2 |           3783.0000 |
|     73584 |      1 |           4079.0000 |
|     73589 |      1 |           4163.0000 |
|     73594 |      1 |           3430.0000 |
|     73595 |      1 |           3167.2500 |
|     73597 |      2 |           3024.0000 |
+-----------+--------+---------------------+
15 rows in set (0.01 sec)
```

Indexing Analysis

- **Query 1:**
  SELECT d.docID, d.docName, COUNT(p.patientID) AS total_patients
  FROM Doctor d
  JOIN Treat t ON d.docID = t.docID
  JOIN Patient p ON t.patientID = p.patientID
  GROUP BY d.docID, d.docName
  ORDER BY total_patients DESC, d.docID
  LIMIT 15;

  Option 0: No indexing

```
| EXPLAIN
                                                                                                              |
+-------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=18.6..18.7 rows=15 loops=1)
    -> Sort: total_patients DESC, d.docID, limit input to 15 row(s) per chunk  (actual time=18.6..18.6 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=18.5..18.6 rows=518 loops=1)
            -> Aggregate using temporary table  (actual time=18.5..18.5 rows=518 loops=1)
                -> Nested loop inner join  (cost=2293 rows=1331) (actual time=9.57..17.9 rows=1100 loops=1)
                    -> Nested loop inner join  (cost=828 rows=1331) (actual time=5.61..9.08 rows=1100 loops=1)
                        -> Table scan on d  (cost=66.7 rows=627) (actual time=3.51..5.09 rows=627 loops=1)
                        -> Covering index lookup on t using PRIMARY (docID=d.docID)  (cost=1 rows=2.12) (actual time=0.00574..0.00621 rows=1.75 loops=627)
                    -> Single-row covering index lookup on p using PRIMARY (patientID=t.patientID)  (cost=1 rows=1) (actual time=0.00786..0.00788 rows=1 loops=1100)
 |
+-------------------------------------------------------------------------------------------------------------+
1 row in set (0.02 sec)
```

  Option 1: Index on docName and docID

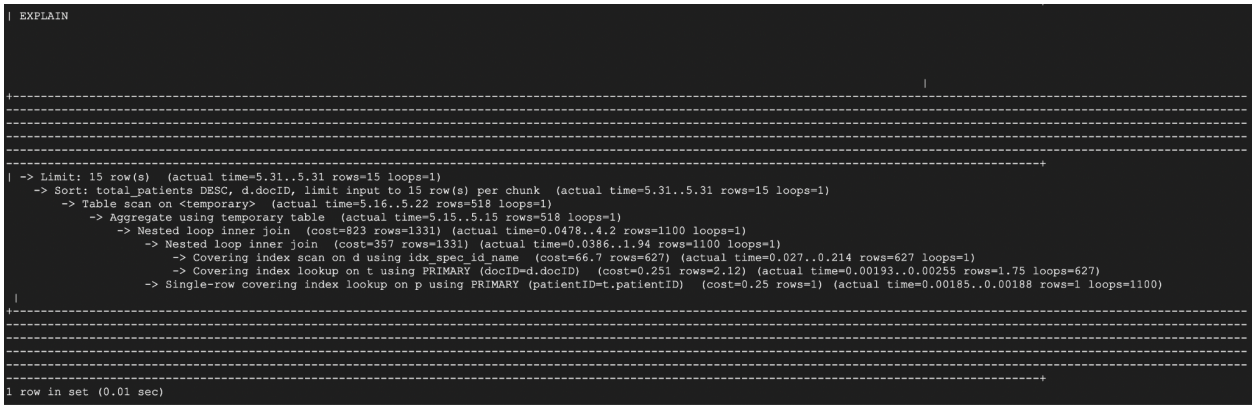
  CREATE INDEX idx_doc_name_id ON Doctor(docName, docID);

```
| EXPLAIN
                                                                                                              |
+-------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=3.2..3.2 rows=15 loops=1)
    -> Sort: total_patients DESC, d.docID, limit input to 15 row(s) per chunk  (actual time=3.2..3.2 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=3.1..3.14 rows=518 loops=1)
            -> Aggregate using temporary table  (actual time=3.1..3.1 rows=518 loops=1)
                -> Nested loop inner join  (cost=820 rows=1331) (actual time=0.0332..2.5 rows=1100 loops=1)
                    -> Nested loop inner join  (cost=354 rows=1331) (actual time=0.0276..1.2 rows=1100 loops=1)
                        -> Covering index scan on d using idx_doc_name_id  (cost=63.7 rows=627) (actual time=0.0178..0.137 rows=627 loops=1)
                        -> Covering index lookup on t using PRIMARY (docID=d.docID)  (cost=0.251 rows=2.12) (actual time=0.00114..0.00156 rows=1.75 loops=627)
                    -> Single-row covering index lookup on p using PRIMARY (patientID=t.patientID)  (cost=0.25 rows=1) (actual time=0.00104..0.00106 rows=1 loops=1100)
 |
+-------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

We see a small improvement from cost=66.7 to cost=63.7 using this index. This is likely because docName has similar selectivity to docID. This index helps a little by optimizing the access of docName and docID. The cost of the nested loop inner join decreases from 2293,828 to 820,354 which indicates improved query performance overall.

Option 2: Index on patientID

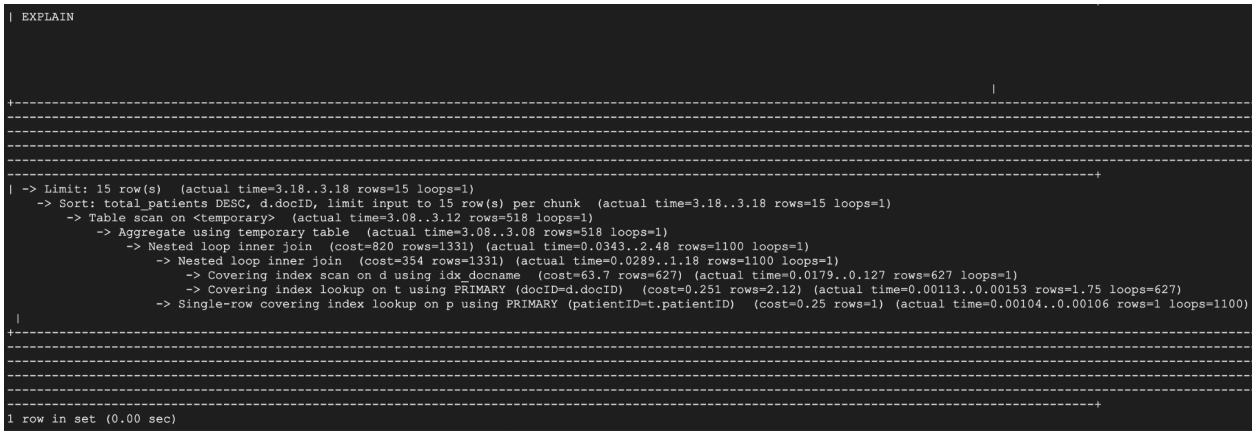
CREATE INDEX idx_patientID ON Patient(patientID);



idx_patientID likely does not provide significant performance gains in this query, and hence the database prefers to use primary key lookup instead.

Option 3: Index on docName

CREATE INDEX idx_docname ON Doctor(docName);

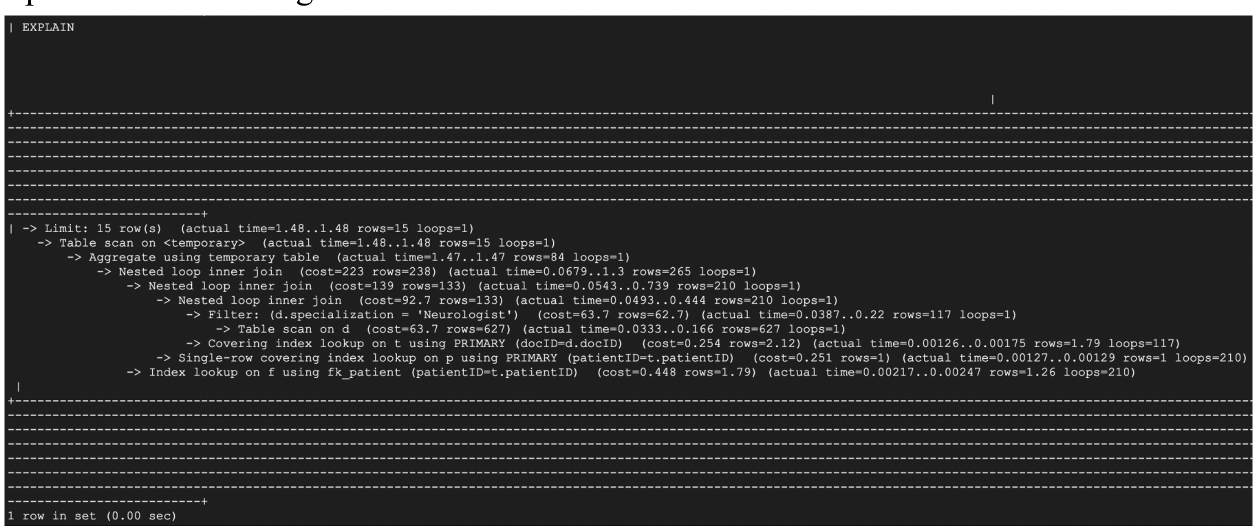


docName as an attribute is not involved in this query, so it makes no difference to performance in comparison to the options discussed earlier.

We went with option 1 as it achieved a slight decrease in costs(from 66.7 to 63.7) and decreased nested inner loop costs as well.

- **Query 2:**
  SELECT d.docID, d.docName, AVG(f.sleepDuration) AS avg_sleep_duration
  FROM Doctor d
  JOIN Treat t ON d.docID = t.docID
  JOIN Patient p ON t.patientID = p.patientID
  JOIN Fitness f ON p.patientID = f.patientID
  WHERE d.specialization = 'Neurologist'
  GROUP BY d.docID, d.docName LIMIT 15;

  Option 0: No indexing



```
| EXPLAIN
+-----------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=1.48..1.48 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=1.48..1.48 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=1.47..1.47 rows=84 loops=1)
            -> Nested loop inner join  (cost=223 rows=238) (actual time=0.0679..1.3 rows=265 loops=1)
                -> Nested loop inner join  (cost=139 rows=133) (actual time=0.0543..0.739 rows=210 loops=1)
                    -> Nested loop inner join  (cost=92.7 rows=133) (actual time=0.0493..0.444 rows=210 loops=1)
                        -> Filter: (d.specialization = 'Neurologist')  (cost=63.7 rows=62.7) (actual time=0.0387..0.22 rows=117 loops=1)
                            -> Table scan on d  (cost=63.7 rows=627) (actual time=0.0333..0.166 rows=627 loops=1)
                        -> Covering index lookup on t using PRIMARY (docID=d.docID)  (cost=0.254 rows=2.12) (actual time=0.00126..0.00175 rows=1.79 loops=117)
                    -> Single-row covering index lookup on p using PRIMARY (patientID=t.patientID)  (cost=0.251 rows=1) (actual time=0.00127..0.00129 rows=1 loops=210)
                -> Index lookup on f using fk_patient (patientID=t.patientID)  (cost=0.448 rows=1.79) (actual time=0.00217..0.00247 rows=1.26 loops=210)
 |
+-----------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Option 1: Index on Specialization

CREATE INDEX idx_specialization ON Doctor(specialization);

```
| EXPLAIN
+------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=1.36..1.36 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=1.36..1.36 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=1.36..1.36 rows=84 loops=1)
            -> Nested loop inner join  (cost=311 rows=444) (actual time=0.14..1.19 rows=265 loops=1)
                -> Nested loop inner join  (cost=156 rows=248) (actual time=0.129..0.635 rows=210 loops=1)
                    -> Nested loop inner join  (cost=68.9 rows=248) (actual time=0.124..0.357 rows=210 loops=1)
                        -> Index lookup on d using idx_specialization (specialization='Neurologist')  (cost=14.7 rows=117) (actual time=0.117..0.15 rows=117 loops=1)
                        -> Covering index lookup on t using PRIMARY (docID=d.docID)  (cost=0.252 rows=2.12) (actual time=0.00122..0.00162 rows=1.79 loops=117)
                    -> Single-row covering index lookup on p using PRIMARY (patientID=t.patientID)  (cost=0.25 rows=1) (actual time=0.00118..0.0012 rows=1 loops=210)
                -> Index lookup on f using fk_patient (patientID=t.patientID)  (cost=0.448 rows=1.79) (actual time=0.00216..0.00246 rows=1.26 loops=210)
 |
+------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

We see a significant improvement from cost=63.7 to cost=14.7 using this index. This is because the database is now able to directly retrieve rows for doctors with specialization in Neurology instead of scanning the entire table to do this.

Option 2: Index on patientID and sleepDuration

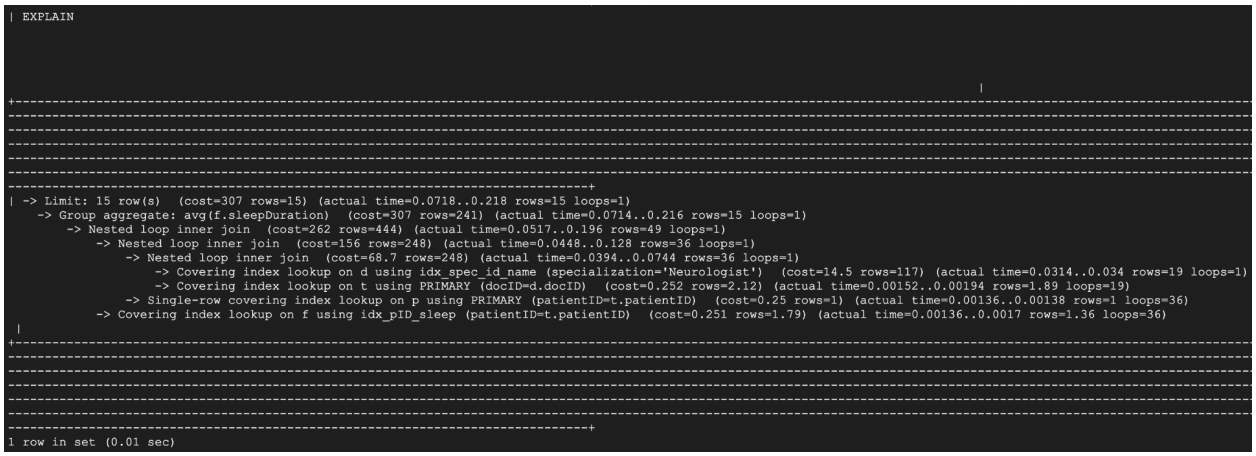
CREATE INDEX idx_pID_sleep ON Fitness(patientID, sleepDuration);

```
| EXPLAIN
+------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=15.9..15.9 rows=15 loops=1)
    -> Table scan on <temporary>  (actual time=15.9..15.9 rows=15 loops=1)
        -> Aggregate using temporary table  (actual time=15.9..15.9 rows=84 loops=1)
            -> Nested loop inner join  (cost=449 rows=444) (actual time=14.2..15.7 rows=265 loops=1)
                -> Nested loop inner join  (cost=156 rows=248) (actual time=0.13..0.638 rows=210 loops=1)
                    -> Nested loop inner join  (cost=68.9 rows=248) (actual time=0.125..0.368 rows=210 loops=1)
                        -> Index lookup on d using idx_specialization (specialization='Neurologist')  (cost=14.7 rows=117) (actual time=0.119..0.157 rows=117 loops=1)
                        -> Covering index lookup on t using PRIMARY (docID=d.docID)  (cost=0.252 rows=2.12) (actual time=0.00122..0.00165 rows=1.79 loops=117)
                    -> Single-row covering index lookup on p using PRIMARY (patientID=t.patientID)  (cost=0.25 rows=1) (actual time=0.00114..0.00116 rows=1 loops=210)
                -> Covering index lookup on f using idx_pID_sleep (patientID=t.patientID)  (cost=1 rows=1.79) (actual time=0.0711..0.0715 rows=1.26 loops=210)
 |
+------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.02 sec)
```

We realized that the cost increases from 0.448 to 1 for lookup on Fitness using this index because calculating average sleepDuration requires reading through all values for each patientID.

Option 3: Index of specialization, docID, and docName

CREATE INDEX idx_spec_id_name ON Doctor(specialization, docID, docName);



The slight improvement from 14.7 to 14.5 between options 1 and 3 is probably due to the fact that applying the index on specialization itself caused most of the increase in efficiency, and the added benefit from adding docName and docID is minimal.

We decided to apply Option 3 because it provides the best possible optimization, reducing the cost significantly.

- **Query 3**

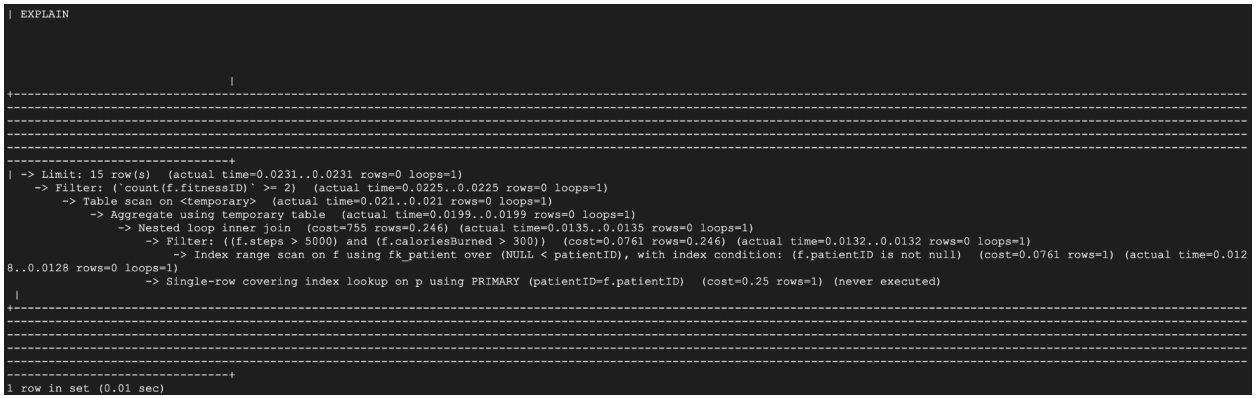
  SELECT p.patientID, COUNT(f.fitnessID) AS total_fitness_records
  FROM Patient p
  JOIN Fitness f ON p.patientID = f.patientID
  WHERE f.steps > 5000 AND f.caloriesBurned > 300
  GROUP BY p.patientID
  HAVING COUNT(f.fitnessID) >= 2
  LIMIT 15;

  Option 0: No indexing

```
| EXPLAIN
                                                                                                                      |
+-----------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=512 rows=15) (actual time=10.7..10.7 rows=0 loops=1)
    -> Filter: (count(f.fitnessID) >= 2)  (cost=512 rows=122) (actual time=10.7..10.7 rows=0 loops=1)
        -> Group aggregate: count(f.fitnessID), count(f.fitnessID)  (cost=512 rows=122) (actual time=10.7..10.7 rows=0 loops=1)
            -> Nested loop inner join  (cost=500 rows=122) (actual time=10.7..10.7 rows=0 loops=1)
                -> Covering index scan on p using PRIMARY  (cost=115 rows=1100) (actual time=5.14..9.17 rows=1100 loops=1)
                -> Filter: ((f.steps > 5000) and (f.caloriesBurned > 300))  (cost=0.25 rows=0.111) (actual time=0.0013..0.0013 rows=0 loops=1100)
                    -> Index lookup on f using fk_patient (patientID=p.patientID)  (cost=0.25 rows=1) (actual time=0.00122..0.00122 rows=0 loops=1100)
 |
+-----------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Option 1: Index on steps

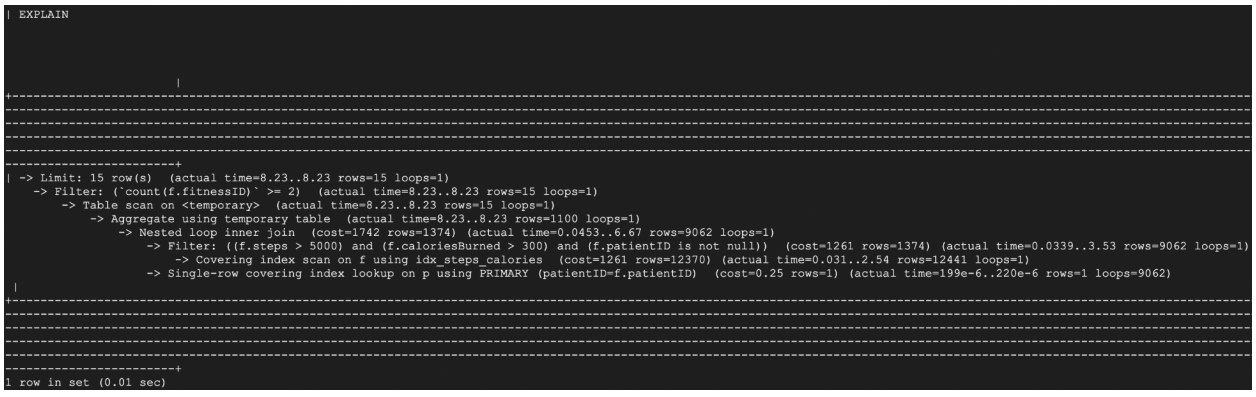
CREATE INDEX idx_steps ON Fitness(steps);



We see a notable improvement from cost=0.25 to cost=0.0761 using this index. This would be especially useful if there were a lot of entries with <=5000 steps. This index allows the database to skip irrelevant rows, thereby improving performance.

Option 2: Index on patientID, steps, and caloriesBurned

CREATE INDEX idx_steps_calories ON Fitness(patientID, steps, caloriesBurned);



The increase in cost to 1261 is likely due to the increased overhead of creating and maintaining this index; additionally, the uniqueness of patientID does not aid the filter on steps and caloriesBurned, which are less selective.

Option 3: Covering Index

CREATE INDEX idx_fitness_covering ON Fitness(patientID, steps, caloriesBurned, fitnessID);

```
| EXPLAIN
                                                                                                                                                                          |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=2570 rows=15) (actual time=0.0531..0.13 rows=15 loops=1)
    -> Filter: (count(f.fitnessID) >= 2)  (cost=2570 rows=1100) (actual time=0.0527..0.128 rows=15 loops=1)
        -> Group aggregate: count(f.fitnessID), count(f.fitnessID)  (cost=2570 rows=1100) (actual time=0.0521..0.127 rows=15 loops=1)
            -> Nested loop inner join  (cost=2433 rows=1374) (actual time=0.042..0.118 rows=122 loops=1)
                -> Covering index scan on p using PRIMARY  (cost=111 rows=1100) (actual time=0.0254..0.0269 rows=16 loops=1)
                -> Filter: ((f.steps > 5000) and (f.caloriesBurned > 300))  (cost=0.986 rows=1.25) (actual time=0.00366..0.00517 rows=7.62 loops=16)
                    -> Covering index lookup on f using idx_fitness_covering (patientID=p.patientID)  (cost=0.986 rows=11.2) (actual time=0.00304..0.00425 rows=10.8 loops=16)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

The increase in cost to 0.986 is likely due to the increased overhead of creating and maintaining this index, especially since Fitness is updated often; additionally, the uniqueness of patientID does not aid the filter on steps and caloriesBurned, which are less selective.

We decided to apply Option 1 (idx_steps to Fitness) due to the decrease in cost.

- **Query 4:**
  SELECT p.patientID, p.gender, AVG(f.caloriesBurned) AS avg_calories_burned
  FROM Patient p
  JOIN Fitness f ON p.patientID = f.patientID
  GROUP BY p.patientID, p.gender
  HAVING (AVG(f.caloriesBurned) > (
     SELECT AVG(f2.caloriesBurned)
     FROM Fitness f2
  )) LIMIT 15;

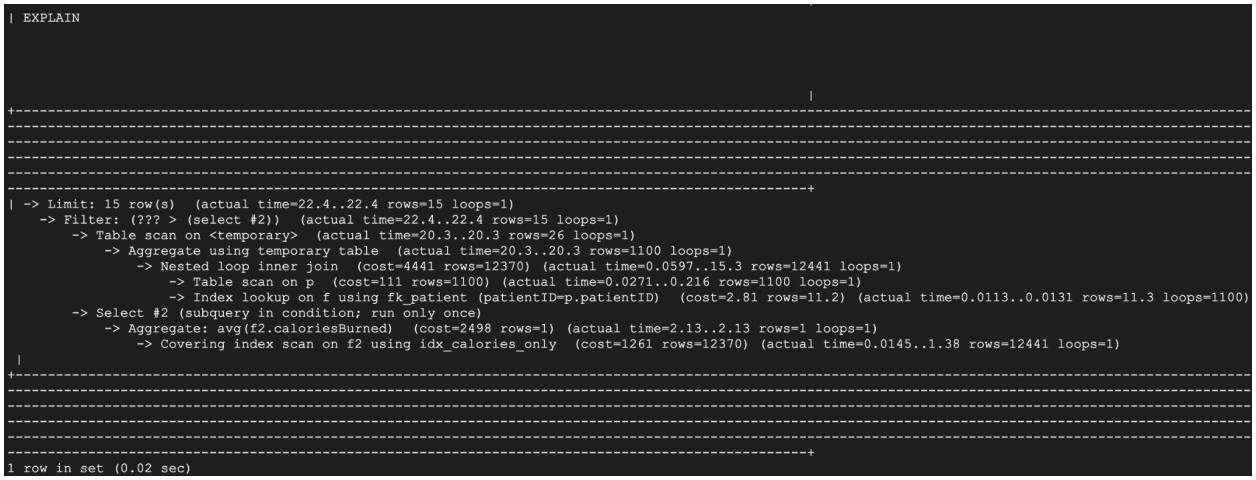
  Option 0: No indexing

```
| EXPLAIN

+------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (actual time=68.9..69 rows=15 loops=1)
    -> Filter: (??? > (select #2))  (actual time=68.9..69 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=66.6..66.6 rows=31 loops=1)
            -> Aggregate using temporary table  (actual time=66.6..66.6 rows=1100 loops=1)
                -> Nested loop inner join  (cost=13718 rows=12370) (actual time=12..61.4 rows=12441 loops=1)
                    -> Covering index scan on p using idx_patient_gender_age  (cost=111 rows=1100) (actual time=0.0169..0.206 rows=1100 loops=1)
                    -> Index lookup on f using fk_patient (patientID=p.patientID)  (cost=11.2 rows=11.2) (actual time=0.0484..0.0549 rows=11.3 loops=1100)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(f2.caloriesBurned)  (cost=2571 rows=1) (actual time=2.35..2.35 rows=1 loops=1)
                -> Table scan on f2  (cost=1334 rows=12370) (actual time=0.0173..1.6 rows=12441 loops=1)
 |
+------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
1 row in set (0.07 sec)
```

  Option 1: Index on caloriesBurned

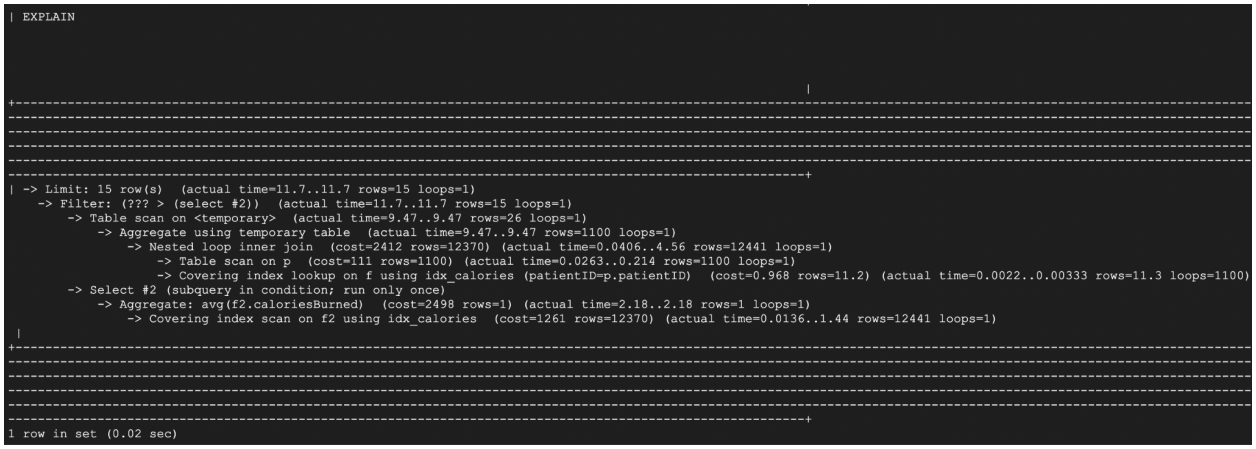
  CREATE INDEX idx_calories_only ON Fitness(caloriesBurned);



We see no change in cost(=1261) from the addition of this index because caloriesBurned likely has low selectivity, leading to the index not really speeding up the query. However, there is a slight improvement on the aggregation costs moving from 2571 without indexing towards 2498 with indexing.

Option 2: Index on patientID and caloriesBurned

CREATE INDEX idx_calories ON Fitness(patientID, caloriesBurned);



We see no change in cost (=1261) from the addition of this index because patientID has all unique values and caloriesBurned likely has low selectivity, leading to the index not really speeding up the query. No improvement compared to earlier options.

Option 3: Index on patientID and gender

CREATE INDEX idx_gender ON Patient(patientID, gender);

```
| EXPLAIN
|
| -> Limit: 15 row(s)  (cost=5678 rows=15) (actual time=2.5..2.91 rows=15 loops=1)
    -> Filter: (avg(f.caloriesBurned) > (select #2))  (cost=5678 rows=1100) (actual time=2.5..2.91 rows=15 loops=1)
        -> Group aggregate: avg(f.caloriesBurned), avg(f.caloriesBurned)  (cost=5678 rows=1100) (actual time=0.0692..0.472 rows=26 loops=1)
            -> Nested loop inner join  (cost=4441 rows=12370) (actual time=0.0405..0.432 rows=298 loops=1)
                -> Covering index scan on p using idx_gender  (cost=111 rows=1100) (actual time=0.0142..0.0173 rows=27 loops=1)
                -> Index lookup on f using fk_patient (patientID=p.patientID)  (cost=2.81 rows=11.2) (actual time=0.0128..0.0147 rows=11 loops=27)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(f2.caloriesBurned)  (cost=2498 rows=1) (actual time=2.42..2.42 rows=1 loops=1)
                -> Table scan on f2  (cost=1261 rows=12370) (actual time=0.0117..1.66 rows=12441 loops=1)
 |
1 row in set (0.01 sec)
```

We see no change in cost (=111) from the addition of this index because patientID has all unique values and gender has low cardinality, causing the combination not to be selective enough, leading to the index not speeding up the query. No improvement compared to earlier options.

All indexes do not significantly affect the performance of the query, however the reduction in cost from no indexing(2571) to indexing (2498) combined with the fact that option 2 had the lowest nested inner loop costs, made us go with option 2.