Database Design

1. Database Implementation

FROM THIS STAGE WE RENAME TABLE "ILLNESSES" TO "CONDITIONS" FOR CONSISTENCY, SORRY FOR THE CONFUSION

In this stage we decide to implement our database on both local and GCP. We fixed the "Missed the Updates Relationship in the Schema." problem in the previous stage by specifying the Update/Delete conditions. The following code is the DDL we used.

```sql
CREATE TABLE USER(
user_id INT,
sex  VARCHAR(255),
country  VARCHAR(255),
FirstName   VARCHAR(255),
LastName   VARCHAR(255),
password   VARCHAR(255),
age INT,
PRIMARY KEY(user_id)
);

CREATE TABLE CONDITIONS(
trackable_id INT,
name   VARCHAR(255),
description   VARCHAR(255),
treatment_options   VARCHAR(255),
amount_of_patient INT,
average_age INT,
PRIMARY KEY(trackable_id)
);

CREATE TABLE SYMPTOMS(
trackable_id INT,
amount_of_patient INT,
average_age INT,
name  VARCHAR(255),
location  VARCHAR(255),
description  VARCHAR(255),
PRIMARY KEY(trackable_id)
);
```

```sql
CREATE TABLE REPORTING(
report_id INT,
report_type VARCHAR(255),
reported_symptom VARCHAR(255),
reported_description VARCHAR(255),
PRIMARY KEY(report_id)
);

CREATE TABLE WISHLIST(
wishlist_id INT,
illness_name VARCHAR(255),
illness_description VARCHAR(255),
user_id INT,
subscription_email VARCHAR(255),
PRIMARY KEY(wishlist_id),
FOREIGN KEY(user_id) REFERENCES USER(user_id) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE relate_to(
condition_id INT,
symptom_id  INT,
condition_name VARCHAR(255),
symptom_name VARCHAR(255),
relation_count INT,
PRIMARY KEY(condition_id, symptom_id),
FOREIGN KEY(condition_id) REFERENCES CONDITIONS(trackable_id) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(symptom_id) REFERENCES SYMPTOMS(trackable_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

```sql
CREATE TABLE diagnosed_with(
condition_id INT,
user_id  int,
PRIMARY KEY(condition_id, user_id),
FOREIGN KEY(condition_id) REFERENCES CONDITIONS(trackable_id) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(user_id) REFERENCES USER(user_id) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE suffer_from(
user_id  INT,
symptom_id  INT,
PRIMARY KEY(user_id, symptom_id),
FOREIGN KEY(user_id) REFERENCES USER(user_id) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY(symptom_id) REFERENCES SYMPTOMS(trackable_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

All of the tables in the database is shown here:

```
mysql> SHOW TABLES;
+----------------+
| Tables_in_cs411 |
+----------------+
| CONDITIONS     |
| REPORTING      |
| SYMPTOMS       |
| USER           |
| WISHLIST       |
| diagnosed_with |
| relate_to      |
| suffer_from    |
+----------------+
8 rows in set (0.00 sec)
```

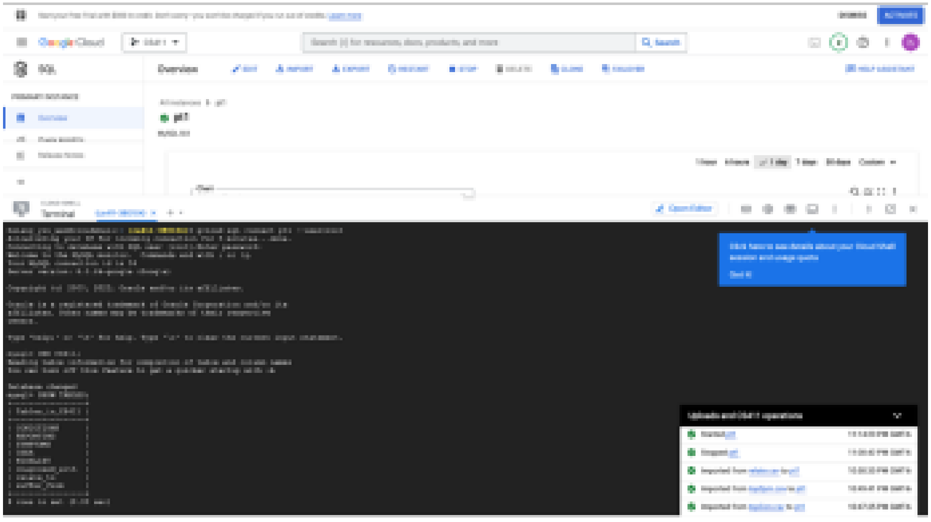
Here is some deep look into the structure of three of our tables:

```
mysql> DESCRIBE SYMPTOMS;
+-------------------+--------------+------+-----+---------+-------+
| Field             | Type         | Null | Key | Default | Extra |
+-------------------+--------------+------+-----+---------+-------+
| trackable_id      | int(11)      | NO   | PRI | NULL    |       |
| amount_of_patient | int(11)      | YES  |     | NULL    |       |
| average_age       | int(11)      | YES  |     | NULL    |       |
| name              | varchar(255) | YES  |     | NULL    |       |
| location          | varchar(255) | YES  |     | NULL    |       |
| description       | varchar(255) | YES  |     | NULL    |       |
+-------------------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)

mysql> DESCRIBE ILLNESES;
+-------------------+--------------+------+-----+---------+-------+
| Field             | Type         | Null | Key | Default | Extra |
+-------------------+--------------+------+-----+---------+-------+
| trackable_id      | int(11)      | NO   | PRI | NULL    |       |
| name              | varchar(255) | YES  |     | NULL    |       |
| description       | varchar(255) | YES  |     | NULL    |       |
| treatment_options | varchar(255) | YES  |     | NULL    |       |
| amount_of_patient | int(11)      | YES  |     | NULL    |       |
| average_age       | int(11)      | YES  |     | NULL    |       |
+-------------------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)

mysql> DESCRIBE relate_to;
+------------+---------+------+-----+---------+-------+
| Field      | Type    | Null | Key | Default | Extra |
+------------+---------+------+-----+---------+-------+
| illness_id | int(11) | NO   | PRI | NULL    |       |
| symptom_id | int(11) | NO   | PRI | NULL    |       |
+------------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> DESCRIBE CONDITIONS;
+-------------------+--------------+------+-----+---------+-------+
| Field             | Type         | Null | Key | Default | Extra |
+-------------------+--------------+------+-----+---------+-------+
| trackable_id      | int(11)      | NO   | PRI | NULL    |       |
| name              | varchar(255) | YES  |     | NULL    |       |
| description       | varchar(255) | YES  |     | NULL    |       |
| treatment_options | varchar(255) | YES  |     | NULL    |       |
| amount_of_patient | int(11)      | YES  |     | NULL    |       |
| average_age       | int(11)      | YES  |     | NULL    |       |
+-------------------+--------------+------+-----+---------+-------+
6 rows in set (0.04 sec)
```

```
mysql> DESCRIBE relate_to;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| condition_id   | int(11)      | NO   | PRI | NULL    |       |
| symptom_id     | int(11)      | NO   | PRI | NULL    |       |
| condition_name | varchar(255) | YES  |     | NULL    |       |
| symptom_name   | varchar(255) | YES  |     | NULL    |       |
| relation_count | int(11)      | YES  |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

For GCP it's all the same, here's a screenshot of the console:



2. Data Pre-Processing

The dataset we used is rich in information yet lacks organization, so we spent a great amount of time pre-processing it to make it usable. We took the following steps to organize the dataset into separate clean data.

<u>Conditions table and Symptoms table</u>

```
Total numer of unique conditions ('trackable_name') tracked are 9443
Total numer of unique symptons ('trackable_name') tracked are 23157
```

Initially, we got 9443 unique conditions and 23157 symptoms in the dataset. To simplify the case, we choose the top 1000 conditions/symptoms by the number of patients reporting having those. Then we calculate the average age of those conditions/symptoms based on the original dataset.

Step 1: Cleaning up and splitting the dataset

```python
import pandas as pd
import numpy as np
## STEP 1: CLEAN UP AND SPLIT THE DATASET
res = pd.read_csv('export.csv')
df = pd.DataFrame(res)
df['user_id'] = pd.Categorical(df['user_id'])
df['user_id']= df.user_id.cat.codes
df["age"] = df.age.replace(0.0,np.nan)

# Save data with trackable_type=="Symptom" or trackable_type=="Condition"
df_filtered_Symptom = df[df.trackable_type=="Symptom"]
df_filtered_Symptom.to_csv('Symptom.csv')
df_filtered_Condition = df[df.trackable_type=="Condition"]
df_filtered_Condition.to_csv('Condition.csv')
```

Step 2: Fetch top 1000 symptoms (same for conditions)

```python
# # STEP 2: GET TOP 1000 SYMPTOMS
res = pd.read_csv('Symptom.csv')
# Create a DataFrame and filter for rows with "trackable_type" equal to "Symptom"
df = pd.DataFrame(res[res.trackable_type=="Symptom"])
# Get the top 10 frequently occurring symptoms
output = df.trackable_name.value_counts().head(1000)
# Calculate the mean age for each symptom in the top 10 list
mean_age = df[df.trackable_name.isin(output.index)].groupby('trackable_name').mean().sort_values(by='age')
# Print the mean age of each symptom in the top 1000 list, sorted by the top 1000 order
out = mean_age.loc[output.index]
out.to_csv('Sym1000age.csv')
```

Step 3: Count top 1000 conditions' number of patients (same for symptoms)

```python
# STEP 4: GET TOP 1000 CONDITIONS COUNTS
res = pd.read_csv('Condition.csv')
df = pd.DataFrame(res)
print("Total numer of unique Condition ('trackable_name') tracked are",df[df.trackable_type=="Condition"].trackable_name.nunique())
count = df[df.trackable_type=="Condition"].trackable_name.value_counts().head(1000)
count.to_csv('topCon.csv')
```

Step 4: Combine the above data and round up data to make things neat. <u>**Conditions-Symptoms relation table**</u>

For simplicity, we randomly chose 400000 data from the original dataset. Then we generated a table for the correlated relationship with the Symptoms and Conditions by counting how many patients reported having both simultaneously. We chose the correlations that have more than 500 records.

```python
res = pd.read_csv('export.csv')
df = pd.DataFrame(res)
df = df.sample(n = 400000)
# Create two DataFrames, one for Conditions and one for Symptoms
conditions = df[df['trackable_type'] == 'Condition'].rename(columns={'trackable_id': 'condition_id', 'trackable_name': 'condition_name'})
symptoms = df[df['trackable_type'] == 'Symptom'].rename(columns={'trackable_id': 'symptom_id', 'trackable_name': 'symptom_name'})
# Merge the two DataFrames on the user_id column
merged = pd.merge(conditions, symptoms, on='user_id')
# Group the merged DataFrame by user_id, condition_id, symptom_id, condition_name, and symptom_name, then count the occurrences
counts = merged.groupby(['user_id', 'condition_id', 'symptom_id', 'condition_name', 'symptom_name']).size().reset_index(name='count')
# Filter the counts DataFrame to only include rows where count is greater than or equal to 2
correlated = counts[counts['count'] >= 500]
print(correlated.shape[0])
correlated.to_csv("relate.csv")
```

3. Data Insertion

In this stage, we finished inserting data into table "CONDITIONS", "SYMPTOMS" nad "relate_to". The following shows the SQL commands we used for our local database. For the GCP, we simply click import.

```sql
LOAD DATA LOCAL INFILE "/Users/mac/Desktop/topCon.csv" INTO TABLE CS411.CONDITIONS
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 LINES;

LOAD DATA LOCAL INFILE "/Users/mac/Desktop/topSym.csv" INTO TABLE CS411.SYMPTOMS
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 LINES;

LOAD DATA LOCAL INFILE "/Users/mac/Desktop/relate.csv" INTO TABLE CS411.relate_to
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

Here are some screenshots of our database:

Insert 1000 rows into TABLE "CONDITIONS"

```
mysql> SELECT COUNT(*) FROM CONDITIONS;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT * FROM CONDITIONS LIMIT 15;
+--------------+------------------------+-------------+--------------------+-------------------+-------------+
| trackable_id | name                   | description | treatment_options  | amount_of_patient | average_age |
+--------------+------------------------+-------------+--------------------+-------------------+-------------+
|            2 | ADHD                   |             |                    |              3768 |          31 |
|            7 | Achalasia              |             |                    |               223 |          49 |
|            8 | Achilles tendinitis    |             |                    |               178 |          41 |
|           10 | Acne                   |             |                    |              1833 |          38 |
|           15 | Acute sinusitis        |             |                    |               647 |          45 |
|           20 | Addison's disease      |             |                    |               316 |          33 |
|           21 | Adenomyosis            |             |                    |               473 |          26 |
|           26 | Adult ADHD             |             |                    |              1245 |          34 |
|           29 | Agoraphobia            |             |                    |               591 |          33 |
|           38 | Allergies              |             |                    |              8722 |          34 |
|           47 | Anal fissure           |             |                    |               152 |          34 |
|           50 | Anaphylaxis            |             |                    |               119 |          33 |
|           51 | Anemia                 |             |                    |              1067 |          33 |
|           56 | Ankylosing spondylitis |             |                    |              3026 |          36 |
|           57 | Anorexia nervosa       |             |                    |               448 |          24 |
+--------------+------------------------+-------------+--------------------+-------------------+-------------+
15 rows in set (0.00 sec)
```

Insert 1000 rows into TABLE “SYMPTOMS”

```
mysql> SELECT COUNT(*) FROM SYMPTOMS;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT * FROM SYMPTOMS LIMIT 15;
+--------------+-------------------+-------------+---------------------+----------+-------------+
| trackable_id | amount_of_patient | average_age | name                | location | description |
+--------------+-------------------+-------------+---------------------+----------+-------------+
|            1 |             33275 |          31 | Abdominal pain      |          |             |
|            3 |              1652 |          31 | Allergy             |          |             |
|            5 |               564 |          45 | Anal itching        |          |             |
|            6 |               510 |          34 | Anemia              |          |             |
|            8 |             61545 |          34 | Anxiety             |          |             |
|            9 |               985 |          37 | Aphasia             |          |             |
|           11 |              6806 |          34 | Ankle pain          |          |             |
|           12 |             37830 |          32 | Back pain           |          |             |
|           13 |               499 |          32 | Bad breath          |          |             |
|           16 |              4417 |          33 | Arm pain            |          |             |
|           17 |               835 |          36 | Bladder spasms      |          |             |
|           18 |               512 |          35 | Bleeding gums       |          |             |
|           22 |               625 |          30 | Bloody nose         |          |             |
|           25 |              6659 |          37 | Blurred vision      |          |             |
|           26 |              1263 |          44 | Bowel incontinence  |          |             |
+--------------+-------------------+-------------+---------------------+----------+-------------+
15 rows in set (0.03 sec)
```

Insert 1000+ rows into TABLE “relate_to”

```
mysql> SELECT COUNT(*) FROM relate_to;
+----------+
| COUNT(*) |
+----------+
|     1188 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT * FROM relate_to ORDER BY RAND() LIMIT 20;
+--------------+------------+------------------------------+-----------------------------------+----------------+
| condition_id | symptom_id | condition_name               | symptom_name                      | relation_count |
+--------------+------------+------------------------------+-----------------------------------+----------------+
|         1199 |        242 | Acid Reflux                  | Fatigue                           |           5250 |
|         1186 |        279 | Mast Cell Activation Syndrome | Muscle spasms                    |            572 |
|         1174 |        121 | joint pain                   | Joint pain                        |           4389 |
|         1132 |        331 | POTS                         | muscle weakness                   |            520 |
|          421 |      13432 | Gastroparesis                | being sick                        |            506 |
|           80 |         45 | Asthma                       | Constipation                      |            638 |
|          222 |       1890 | Chronic hives (urticaria)    | Nose sores                        |            525 |
|          291 |         35 | Depression                   | Chest pain                        |            528 |
|          397 |       2762 | Fibromyalgia                 | hand and wrist pain               |            504 |
|         6154 |      12247 | bowel not working very well  | got shuff pouring out underneath  |            675 |
|          532 |         54 | Idiopathic hypersomnia       | Depression                        |           1089 |
|          291 |        515 | Depression                   | exhaustion                        |            528 |
|         1164 |        329 | Fatigue                      | impaired cognition                |           1276 |
|           80 |       7004 | Asthma                       | GI Distress                       |            506 |
|         1201 |        674 | Chronic Pain                 | Neuropathy                        |            961 |
|          485 |      11505 | Hepatitis C                  | How Do You Feel Today?            |            720 |
|         1493 |       8518 | ME/CFS                       | overexertion                      |           1421 |
|          269 |        243 | Crohn's disease              | Stomach Pain                      |            532 |
|         1158 |        142 | OCD                          | Mood swings                       |            912 |
|          728 |        415 | Osteoarthritis               | Achiness                          |            754 |
+--------------+------------+------------------------------+-----------------------------------+----------------+
20 rows in set (0.01 sec)
```

4. Advanced Queries

<u>QUERY1:</u> Condition and symptom names that are related to each other

This query finds the average age of patients who have at least two symptoms in common, along with the names of the conditions and symptoms. This is useful because we want to find patients that have similar histories and then we can extrapolate more information based off of those similar histories. Perhaps even find a cause for the symptoms or derive risk factors from them.

The commands:

SELECT c.name AS condition_name, s.name AS symptom_name, AVG(s.average_age) AS avg_age

FROM CONDITIONS c JOIN relate_to r1 ON c.trackable_id = r1.condition_id

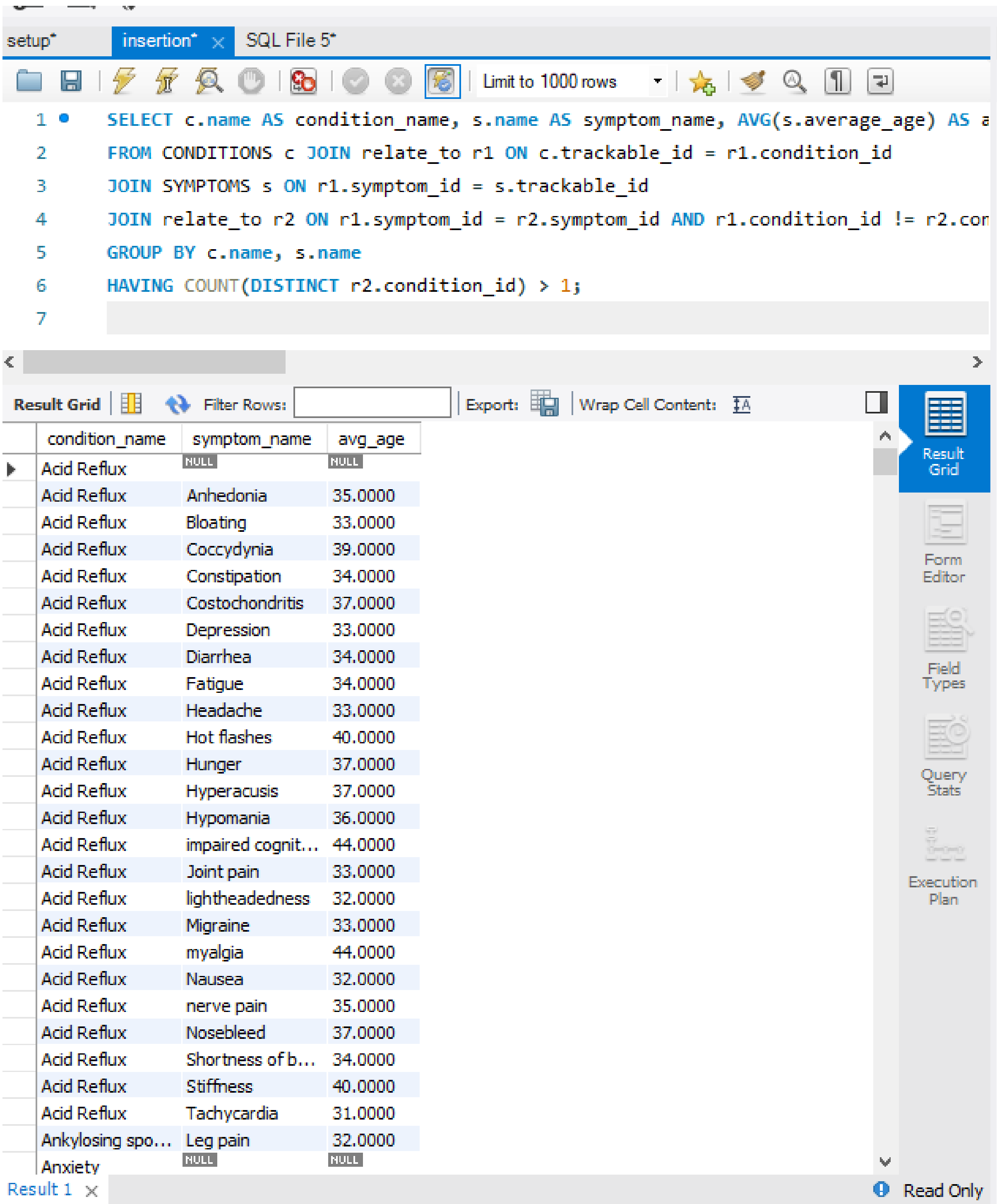
JOIN SYMPTOMS s ON r1.symptom_id = s.trackable_id

JOIN relate_to r2 ON r1.symptom_id = r2.symptom_id AND r1.condition_id != r2.condition_id

GROUP BY c.name, s.name

HAVING COUNT(DISTINCT r2.condition_id) > 1;

The results screenshot:



<u>Indexing Analysis</u>

Default query:

```
-> Filter: (count(distinct relate_to.condition_id) > 1)  (actual time=21.133..26.635 rows=956 loops=1)
    -> Group aggregate: count(distinct relate_to.condition_id), avg(symptoms.average_age)  (actual time=21.132..26.566 rows=1039 loops=1)
        -> Sort: c.`name`, s.`name`  (actual time=21.117..21.801 rows=11736 loops=1)
            -> Stream results  (cost=1972.17 rows=4780) (actual time=0.041..13.004 rows=11736 loops=1)
                -> Nested loop inner join  (cost=1972.17 rows=4780) (actual time=0.039..10.026 rows=11736 loops=1)
                    -> Nested loop inner join  (cost=1141.35 rows=1193) (actual time=0.032..2.646 rows=1193 loops=1)
                        -> Nested loop inner join  (cost=723.80 rows=1193) (actual time=0.028..1.872 rows=1193 loops=1)
                            -> Covering index scan on r1 using rs_idx  (cost=127.30 rows=1193) (actual time=0.020..0.311 rows=1193 loops=1)
                            -> Single-row index lookup on c using PRIMARY (trackable_id=r1.condition_id)  (cost=0.40 rows=1) (actual time=0.001..0.001 rows=1 loops=1193)
                        -> Single-row index lookup on s using PRIMARY (trackable_id=r1.symptom_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1193)
                    -> Filter: (r1.condition_id <> r2.condition_id)  (cost=0.25 rows=4) (actual time=0.002..0.006 rows=10 loops=1193)
                        -> Covering index lookup on r2 using rs_idx (symptom_id=r1.symptom_id)  (cost=0.25 rows=4) (actual time=0.002..0.005 rows=11 loops=1193)
```

So this is the EXPLAIN ANALYZE result of the original query.

Index 1: CREATE INDEX symptomname_idx on symptoms(name);

```
-> Filter: (count(distinct relate_to.condition_id) > 1)  (actual time=22.808..28.489 rows=956 loops=1)
    -> Group aggregate: count(distinct relate_to.condition_id), avg(symptoms.average_age)  (actual time=22.806..28.416 rows=1039 loops=1)
        -> Sort: c.`name`, s.`name`  (actual time=22.791..23.526 rows=11736 loops=1)
            -> Stream results  (cost=1972.17 rows=4780) (actual time=0.069..14.320 rows=11736 loops=1)
                -> Nested loop inner join  (cost=1972.17 rows=4780) (actual time=0.065..11.225 rows=11736 loops=1)
                    -> Nested loop inner join  (cost=1141.35 rows=1193) (actual time=0.057..3.207 rows=1193 loops=1)
                        -> Nested loop inner join  (cost=723.80 rows=1193) (actual time=0.053..2.089 rows=1193 loops=1)
                            -> Covering index scan on r1 using rs_idx  (cost=127.30 rows=1193) (actual time=0.037..0.330 rows=1193 loops=1)
                            -> Single-row index lookup on c using PRIMARY (trackable_id=r1.condition_id)  (cost=0.40 rows=1) (actual time=0.001..0.001 rows=1 loops=1193)
                        -> Single-row index lookup on s using PRIMARY (trackable_id=r1.symptom_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1193)
                    -> Filter: (r1.condition_id <> r2.condition_id)  (cost=0.25 rows=4) (actual time=0.002..0.006 rows=10 loops=1193)
                        -> Covering index lookup on r2 using rs_idx (symptom_id=r1.symptom_id)  (cost=0.25 rows=4) (actual time=0.002..0.005 rows=11 loops=1193)
```

To begin we make a simple index on the symptom name to see if it makes any difference. You can see that the times actually increase, but the costs themselves are the exact same as before. So this is not actually improving the efficiency of the query but seems to be making the times worse anyway.

Index 2: CREATE INDEX cidx on relate_to(condition_id, symptom_id)

Let's try making an index for the relate_to table instead. We will also try using two columns this time, condition_id and symptom_id

```
-> Filter: (count(distinct relate_to.condition_id) > 1)  (actual time=20.481..25.758 rows=956 loops=1)
    -> Group aggregate: count(distinct relate_to.condition_id), avg(symptoms.average_age)  (actual
    time=20.479..25.686 rows=1039 loops=1)
        -> Sort: c.`name`, s.`name`  (actual time=20.463..21.170 rows=11736 loops=1)
            -> Stream results  (cost=1966.17 rows=4780) (actual time=0.079..12.541 rows=11736 loops=1)
                -> Nested loop inner join  (cost=1966.17 rows=4780) (actual time=0.075..9.661 rows=11736 loops=1)
                    -> Nested loop inner join  (cost=1135.35 rows=1193) (actual time=0.067..2.621 rows=1193
                    loops=1)
                        -> Nested loop inner join  (cost=717.80 rows=1193) (actual time=0.049..1.913 rows=1193
                        loops=1)
                            -> Covering index scan on r1 using rs_idx  (cost=121.30 rows=1193) (actual
                            time=0.036..0.325 rows=1193 loops=1)
                            -> Single-row index lookup on c using PRIMARY (trackable_id=r1.condition_id)  (
                            cost=0.40 rows=1) (actual time=0.001..0.001 rows=1 loops=1193)
                        -> Single-row index lookup on s using PRIMARY (trackable_id=r1.symptom_id)  (cost=0.25
                        rows=1) (actual time=0.000..0.000 rows=1 loops=1193)
                    -> Filter: (r1.condition_id <> r2.condition_id)  (cost=0.25 rows=4) (actual
                    time=0.002..0.005 rows=10 loops=1193)
                        -> Covering index lookup on r2 using rs_idx (symptom_id=r1.symptom_id)  (cost=0.25
                        rows=4) (actual time=0.002..0.005 rows=11 loops=1193)
```

So immediately there appears to be a boost in overall time (from 21.1 seconds to 20.4), and also we see the costs have lowered (from 1972 to 1966 for the top level operation. Some of the other operations have also decreased (1141 to 1135 for the nested inner loop join), 723 to 717, etc. lots of slight improvements). The structure of the query still hasn't changed though, there are still 12 operations total.

Index 3: CREATE INDEX cdidx on c(condition_id, symptom_id)

```
-> Filter: (count(distinct relate_to.condition_id) > 1)  (actual time=21.018..26.373 rows=956 loops=1)
    -> Group aggregate: count(distinct relate_to.condition_id), avg(symptoms.average_age)  (actual
    time=21.016..26.303 rows=1039 loops=1)
        -> Sort: c.`name`, s.`name`  (actual time=20.997..21.679 rows=11736 loops=1)
            -> Stream results  (cost=1787.22 rows=4780) (actual time=0.045..12.581 rows=11736 loops=1)
                -> Nested loop inner join  (cost=1787.22 rows=4780) (actual time=0.043..9.651 rows=11736 loops=1)
                    -> Nested loop inner join  (cost=956.40 rows=1193) (actual time=0.033..2.594 rows=1193
                    loops=1)
                        -> Nested loop inner join  (cost=538.85 rows=1193) (actual time=0.029..1.907 rows=1193
                        loops=1)
                            -> Covering index scan on r1 using rs_idx  (cost=121.30 rows=1193) (actual
                            time=0.021..0.309 rows=1193 loops=1)
                            -> Single-row index lookup on c using PRIMARY (trackable_id=r1.condition_id)  (
                            cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1193)
                        -> Single-row index lookup on s using PRIMARY (trackable_id=r1.symptom_id)  (cost=0.25
                        rows=1) (actual time=0.000..0.000 rows=1 loops=1193)
                    -> Filter: (r1.condition_id <> r2.condition_id)  (cost=0.25 rows=4) (actual
                    time=0.002..0.005 rows=10 loops=1193)
                        -> Covering index lookup on r2 using rs_idx (symptom_id=r1.symptom_id)  (cost=0.25
                        rows=4) (actual time=0.002..0.005 rows=11 loops=1193)
```

With this last attempt, the time actually goes up again, but the costs are significantly lower (by almost 200 points). You can see that the first "Stream results" operation goes from 1966 to 1787, which is better than the previous two indexes and also better than the original query as well. Similarly, all the other operations have much lower costs (except for the last few which seem to be constant at a cost of 0.25). But overall if we had to choose one query it would be this one since it is the one showing the biggest improvements from the original.

QUERY 2: Select symptom that occurs in the most conditions

This query retrieves the possible symptoms in the database, ordered by the frequency of their occurrence in conjunction with the conditions. Since the relate_to table, brings the symptom and condition data together, we utilized it to find how many instances of the symptoms occur across the various conditions using COUNT.

```sql
SELECT r.symptom_id , COUNT(r.symptom_id) AS cnt
FROM relate_to r
GROUP BY r.symptom_id
ORDER BY COUNT(r.symptom_id) DESC;
```

CLOUD SHELL
Terminal (cs411-380104)

```
+------------+-----+
| symptom_id | cnt |
+------------+-----+
|         98 |  39 |
|        152 |  38 |
|        242 |  34 |
|         45 |  25 |
|         63 |  23 |
|          8 |  22 |
|        121 |  22 |
|        145 |  21 |
|        275 |  21 |
|         54 |  19 |
|         12 |  17 |
|         35 |  17 |
|         56 |  14 |
|        399 |  14 |
|        257 |  13 |
|        515 |  13 |
|        191 |  12 |
|        329 |  11 |
|       2605 |  11 |
|         84 |  10 |
|        265 |  10 |
|        681 |  10 |
```

Indexing Analysis

Default first:

```
# EXPLAIN
-> Sort: cnt DESC  (actual time=1.262..1.269 rows=268 loops=1)
    -> Stream results  (cost=240.60 rows=268) (actual time=0.400..0.754 rows=268 loops=1)
        -> Group aggregate: count(r.symptom_id)  (cost=240.60 rows=268) (actual time=0.395..0.721 rows=268
        loops=1)
            -> Covering index scan on r using symptom_id  (cost=121.30 rows=1193) (actual time=0.260..0.526
            rows=1193 loops=1)
```

So about 1.2 seconds

Index 1: CREATE INDEX rs_idx on relate_to(symptom_id);

Let's try indexing symptom_id:

```
# EXPLAIN
-> Sort: cnt DESC  (actual time=0.518..0.527 rows=268 loops=1)
    -> Stream results  (cost=240.60 rows=268) (actual time=0.106..0.448 rows=268 loops=1)
        -> Group aggregate: count(r.symptom_id)  (cost=240.60 rows=268) (actual time=0.103..0.418 rows=268
        loops=1)
            -> Covering index scan on r using rs_idx  (cost=121.30 rows=1193) (actual time=0.097..0.349
            rows=1193 loops=1)
```

Okay, so the time dropped about 0.7 seconds. Quite good.
Index 2: CREATE INDEX rc_idx on relate_to(condition_id);

Well, there's not much else to index since in the query above we only really checked one field. So we will just try other fields in the relate_to table and see if changing them gives any difference.

```
# EXPLAIN
-> Sort: cnt DESC  (actual time=0.518..0.527 rows=268 loops=1)
    -> Stream results  (cost=240.60 rows=268) (actual time=0.106..0.448 rows=268 loops=1)
        -> Group aggregate: count(r.symptom_id)  (cost=240.60 rows=268) (actual time=0.103..0.418 rows=268
        loops=1)
            -> Covering index scan on r using rs_idx  (cost=121.30 rows=1193) (actual time=0.097..0.349
            rows=1193 loops=1)
```

Strangely enough, it seems to still speed up the queries.
Index 3: CREATE INDEX rsn_idx on relate_to(symptom_name);

```
# EXPLAIN
-> Sort: cnt DESC  (actual time=0.629..0.636 rows=268 loops=1)
    -> Stream results  (cost=240.60 rows=268) (actual time=0.055..0.537 rows=268 loops=1)
        -> Group aggregate: count(r.symptom_id)  (cost=240.60 rows=268) (actual time=0.053..0.490 rows=268
        loops=1)
            -> Covering index scan on r using rs_idx  (cost=121.30 rows=1193) (actual time=0.046..0.389
            rows=1193 loops=1)
```

And for the final one, it's a bit slower at 0.629 sec. The costs for all three are the same, so there's not much to comment on there. The query itself is perhaps a little too simple to benefit much from indexing. Or perhaps, we should try combinations of columns from different tables. But with the current strategies, we don't seem to get much improvement.