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.

```
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)
);
```

```
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
);

```

```

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)
```

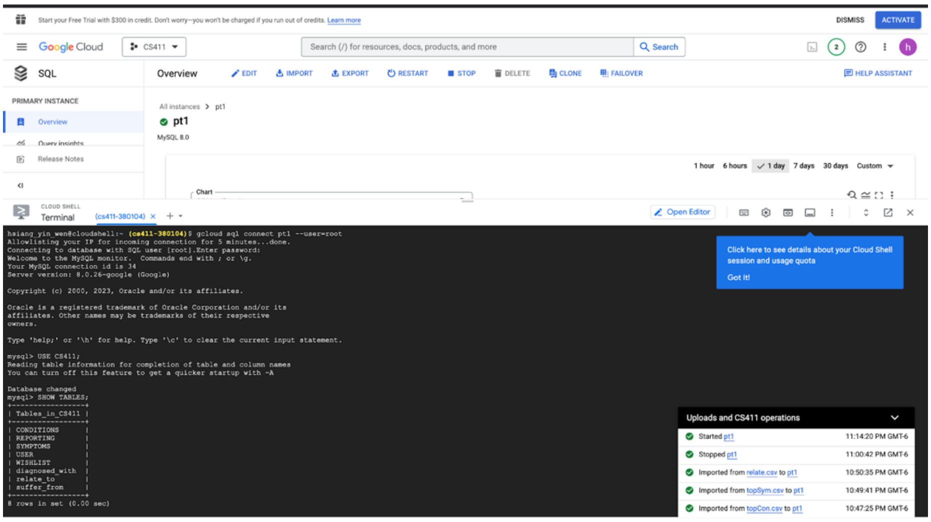
Here is some deep look into the structure of three of our tables:

```
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 symptoms ('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    |             |                   |               179 |          41 |
|           10 | Acne                   |             |                   |              1831 |          30 |
|           19 | Acute sinusitis        |             |                   |               647 |          45 |
|           20 | Addison's disease      |             |                   |               316 |          33 |
|           21 | Adenomyosis            |             |                   |               473 |          26 |
|           26 | Adult ADHD             |             |                   |              1245 |          34 |
|           29 | Agoraphobia            |             |                   |               591 |          33 |
|           36 | Allergies              |             |                   |              8722 |          34 |
|           47 | Anal fissure           |             |                   |               152 |          34 |
|           50 | Anaphylaxis            |             |                   |               119 |          33 |
|           51 | Anemia                 |             |                   |              1067 |          33 |
|           56 | Ankylosing spondylitis |             |                   |              3026 |          36 |
|           57 | Anorexia nervosa       |             |                   |               446 |          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

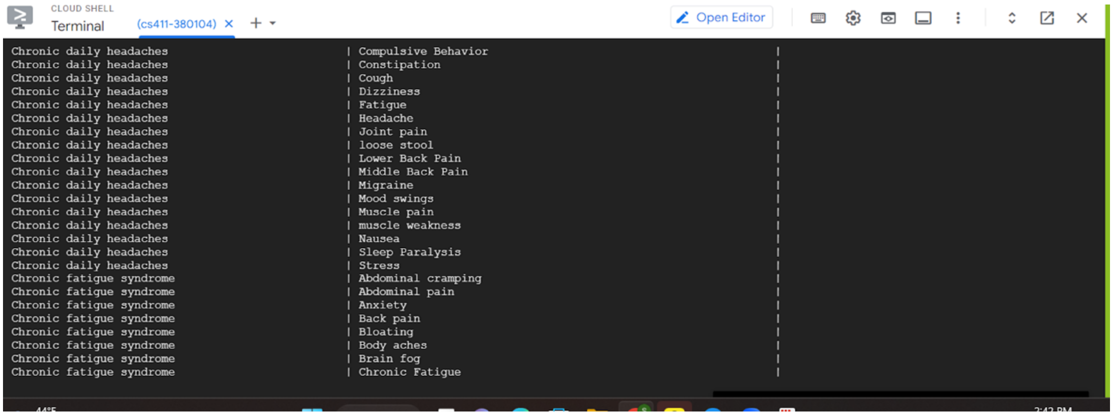
QUERY1: Condition and symptom names that are related to each other

This query retrieves distinct pairs of condition and symptom names that are related to each other. It uses joins to combine the SYMPTOMS, relate_to, and CONDITIONS tables, then groups the results by the name of the condition and symptom, and filters the results to only include rows where the trackable_id of the condition equals the condition_id of the relate_to table. Finally, the results are ordered by the name of the condition.

The commands:

```sql
SELECT DISTINCT c.name, s.name
FROM SYMPTOMS s
JOIN relate_to r ON r.symptom_name = s.name
JOIN CONDITIONS c ON c.trackable_id = r.condition_id
GROUP BY c.name, s.name, c.trackable_id, r.condition_id
HAVING c.trackable_id = r.condition_id
ORDER BY c.name;

```

The results screenshot:


```
Chronic daily headaches          | Compulsive Behavior      |
Chronic daily headaches          | Constipation             |
Chronic daily headaches          | Cough                    |
Chronic daily headaches          | Dizziness                |
Chronic daily headaches          | Fatigue                  |
Chronic daily headaches          | Headache                 |
Chronic daily headaches          | Joint pain               |
Chronic daily headaches          | loose stool              |
Chronic daily headaches          | Lower Back Pain          |
Chronic daily headaches          | Middle Back Pain         |
Chronic daily headaches          | Migraine                 |
Chronic daily headaches          | Mood swings              |
Chronic daily headaches          | Muscle pain              |
Chronic daily headaches          | muscle weakness          |
Chronic daily headaches          | Nausea                   |
Chronic daily headaches          | Sleep Paralysis          |
Chronic daily headaches          | Stress                   |
Chronic fatigue syndrome         | Abdominal cramping       |
Chronic fatigue syndrome         | Abdominal pain           |
Chronic fatigue syndrome         | Anxiety                  |
Chronic fatigue syndrome         | Back pain                |
Chronic fatigue syndrome         | Bloating                 |
Chronic fatigue syndrome         | Body aches               |
Chronic fatigue syndrome         | Brain fog                |
Chronic fatigue syndrome         | Chronic Fatigue          |
```

Indexing Analysis

Default query:

```
# EXPLAIN
-> Sort: c.`name`, s.`name`, c.trackable_id, r.condition_id  (actual time=6.750..6.802 rows=1057 loops=1)
    -> Sort with duplicate removal: c.`name`, s.`name`  (actual time=6.334..6.389 rows=1057 loops=1)
        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=5.662..5.772 rows=1057 loops=1)
            -> Temporary table with deduplication  (cost=0.00..0.00 rows=0) (actual time=5.660..5.660 rows=1057
            loops=1)
                -> Filter: (c.trackable_id = r.condition_id)  (actual time=3.842..4.912 rows=1057 loops=1)
                    -> Filter: (s.`name` = r.symptom_name)  (cost=159924.51 rows=159459) (actual
                    time=3.839..4.822 rows=1057 loops=1)
                        -> Inner hash join (<hash>(s.`name`)=<hash>(r.symptom_name))  (cost=159924.51
                        rows=159459) (actual time=3.837..4.578 rows=1057 loops=1)
                            -> Table scan on s  (cost=0.03 rows=1593) (actual time=0.055..0.490 rows=1593
                            loops=1)
                            -> Hash
                                -> Nested loop inner join  (cost=451.95 rows=1001) (actual time=0.091..3.393
                                rows=1193 loops=1)
                                    -> Table scan on c  (cost=101.60 rows=1001) (actual time=0.051..0.441
                                    rows=1001 loops=1)
                                    -> Index lookup on r using PRIMARY (condition_id=c.trackable_id)  (cost=0.25
                                    rows=1) (actual time=0.002..0.003 rows=1 loops=1001)
```

The costs have been highlighted along with the times and the number of rows/loops.Time is around 7 seconds. Now we will try three indexes.

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

```
# EXPLAIN
-> Sort: c.`name`, s.`name`, c.trackable_id, r.condition_id  (actual time=7.070..7.122 rows=1057 loops=1)
    -> Sort with duplicate removal: c.`name`, s.`name`  (actual time=6.653..6.705 rows=1057 loops=1)
        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=6.025..6.133 rows=1057 loops=1)
            -> Temporary table with deduplication  (cost=0.00..0.00 rows=0) (actual time=6.024..6.024 rows=1057
            loops=1)
                -> Filter: (c.trackable_id = r.condition_id)  (actual time=0.085..5.313 rows=1057 loops=1)
                    -> Nested loop inner join  (cost=1609.71 rows=2349) (actual time=0.084..5.228 rows=1057
                    loops=1)
                        -> Nested loop inner join  (cost=538.85 rows=1193) (actual time=0.068..1.142 rows=1193
                        loops=1)
                            -> Filter: (r.symptom_name is not null)  (cost=121.30 rows=1193) (actual
                            time=0.049..0.538 rows=1193 loops=1)
                                -> Table scan on r  (cost=121.30 rows=1193) (actual time=0.046..0.466 rows=1193
                                loops=1)
                            -> Single-row index lookup on c using PRIMARY (trackable_id=r.condition_id)  (
                            cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1193)
                        -> Covering index lookup on s using symptomname_idx (name=r.symptom_name)  (cost=0.70
                        rows=2) (actual time=0.003..0.003 rows=1 loops=1193)
```

With this first index we actually seemed to have made the performance worse (from 6.750..6.802 to 7.070..7.122). The inner hash join has been replaced with two nested loop inner joins. We’ve also gotten rid of an extra table scan. However there are now **two** index row lookups required, which is maybe why it is taking longer.

Index 2: CREATE INDEX conditonid_idx on relate_to(condition_id);

Let’s try making an index for the relate_to table instead. First we should drop the previous index like so:

```
DROP INDEX symptomname_idx on symptoms;
```

```
# EXPLAIN
-> Sort: c.`name`, s.`name`, c.trackable_id, r.condition_id  (actual time=3.837..3.889 rows=1057 loops=1)
    -> Sort with duplicate removal: c.`name`, s.`name`  (actual time=3.422..3.477 rows=1057 loops=1)
        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=2.757..2.867 rows=1057 loops=1)
            -> Temporary table with deduplication  (cost=0.00..0.00 rows=0) (actual time=2.756..2.756 rows=1057
            loops=1)
                -> Filter: (c.trackable_id = r.condition_id)  (actual time=1.204..2.141 rows=1057 loops=1)
                    -> Filter: (s.`name` = r.symptom_name)  (cost=190599.27 rows=190045) (actual
                    time=1.203..2.078 rows=1057 loops=1)
                        -> Inner hash join (<hash>(s.`name`)=<hash>(r.symptom_name))  (cost=190599.27
                        rows=190045) (actual time=1.202..1.882 rows=1057 loops=1)
                            -> Table scan on s  (cost=0.03 rows=1593) (actual time=0.035..0.449 rows=1593
                            loops=1)
                            -> Hash
                                -> Nested loop inner join  (cost=538.85 rows=1193) (actual time=0.065..0.876
                                rows=1193 loops=1)
                                    -> Table scan on r  (cost=121.30 rows=1193) (actual time=0.052..0.463
                                    rows=1193 loops=1)
                                    -> Single-row index lookup on c using PRIMARY (trackable_id=r.condition_id)
                                    (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1193)
```

So immediately there appears to be a boost in overall time, the 6-7 second times from before are around the 4 second range now. This query has the same structure as the default query except the last index lookup has been replaced with a single-row index lookup.

Index 3: CREATE INDEX conditions_idx on conditions(name, trackable_id);

Here we try doing two columns instead of 1, and we try it on the last table.

```
# EXPLAIN
-> Sort: c.`name`, s.`name`, c.trackable_id, r.condition_id  (actual time=5.735..5.809 rows=1057 loops=1)
    -> Sort with duplicate removal: c.`name`, s.`name`  (actual time=5.126..5.214 rows=1057 loops=1)
        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=4.044..4.216 rows=1057 loops=1)
            -> Temporary table with deduplication  (cost=0.00..0.00 rows=0) (actual time=4.042..4.042 rows=1057
            loops=1)
                -> Filter: (c.trackable_id = r.condition_id)  (actual time=1.424..3.029 rows=1057 loops=1)
                    -> Filter: (s.`name` = r.symptom_name)  (cost=190599.27 rows=190045) (actual
                    time=1.423..2.915 rows=1057 loops=1)
                        -> Inner hash join (<hash>(s.`name`)=<hash>(r.symptom_name))  (cost=190599.27
                        rows=190045) (actual time=1.421..2.586 rows=1057 loops=1)
                            -> Table scan on s  (cost=0.03 rows=1593) (actual time=0.035..0.763 rows=1593
                            loops=1)
                            -> Hash
                                -> Nested loop inner join  (cost=538.85 rows=1193) (actual time=0.045..1.039
                                rows=1193 loops=1)
                                    -> Table scan on r  (cost=121.30 rows=1193) (actual time=0.036..0.504
                                    rows=1193 loops=1)
                                    -> Single-row index lookup on c using PRIMARY (trackable_id=r.condition_id)
                                    (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1193)
```

So, with this last attempt our times are just under 6 seconds. This is better than the default and attempt 1 but not as good as attempt 2. Also, since we used two columns this will be more work to update. The structure of the query is the same as attempt 2 and the default query, in that there the last operation is the only difference and it is now a single-row index lookup.

<u>QUERY 2:</u> 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=0.451..0.458 rows=268 loops=1)
    -> Stream results  (cost=240.60 rows=268) (actual time=0.047..0.390 rows=268 loops=1)
        -> Group aggregate: count(r.symptom_id)  (cost=240.60 rows=268) (actual time=0.044..0.361 rows=268
        loops=1)
            -> Covering index scan on r using rs_idx  (cost=121.30 rows=1193) (actual time=0.039..0.294
            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.