Project Track 1, Stage 3
Database Implementation and Indexing

Database Implementation:

Screenshot of our GCP connection to the database, and an example of 10 rows for the Medicines table.

```
+--------------------------+
| Tables_in_stage2_schema |
+--------------------------+
| carries                  |
| causedBy                 |
| currentlyHas             |
| medicines                |
| pharmacies               |
| side_effects             |
| symptoms                 |
| treatedBy                |
| users                    |
+--------------------------+
9 rows in set (0.00 sec)

mysql> SELECT * FROM medicines LIMIT 10;
+----------------------+---------------------------------+-------------------------------------------------------------------------+--------+
| name                 | manufacturer                    | composition                                                             | rating |
+----------------------+---------------------------------+-------------------------------------------------------------------------+--------+
| A Doxid 100mg Capsule | Eskon Pharma                    | Doxycycline (100mg)                                                     |   3.44 |
| A Ret HC Cream       | A. Menarini India Pvt Ltd       | Hydroquinone (2% w/w) + Tretinoin (0.05% w/w) + Hydrocortisone (1% w/w) |   2.44 |
| A-CN Gel             | Eskon Pharma                    | Clindamycin (1% w/w) + Nicotinamide (4% w/w)                            |   3.66 |
| A-Ret 0.025% Gel     | A. Menarini India Pvt Ltd       | Tretinoin (0.025% w/w)                                                  |   3.00 |
| A-Ret 0.05% Gel      | A. Menarini India Pvt Ltd       | Tretinoin (0.05% w/w)                                                   |   3.06 |
| A-Ret 0.1% Gel       | A. Menarini India Pvt Ltd       | Tretinoin (0.1% w/w)                                                    |   3.00 |
| A-Ret 0.5% Cream     | A. Menarini India Pvt Ltd       | Tretinoin (0.5% w/w)                                                    |   3.16 |
| AA 5 Tablet          | Blaze Remedies                  | Levocetirizine (5mg)                                                    |   5.00 |
| AB Phylline Capsule  | Sun Pharmaceutical Industries Ltd | Acebrophylline (100mg)                                                  |   3.10 |
| AB Phylline N Tablet | Sun Pharmaceutical Industries Ltd | Acebrophylline (100mg) + Acetylcysteine (600mg)                         |   3.38 |
+----------------------+---------------------------------+-------------------------------------------------------------------------+--------+
10 rows in set (0.00 sec)
```

DDL commands:

FOR ENTITY TABLES:

```
CREATE TABLE IF NOT EXISTS users (
state VARCHAR(255),
city VARCHAR(255),
address VARCHAR(255),
  email VARCHAR(255) PRIMARY KEY,
  preferred_pharmacy_id INT,
  FOREIGN KEY (preferred_pharmacy_id) REFERENCES pharmacies(id) ON DELETE SET
NULL
);

CREATE TABLE IF NOT EXISTS symptoms (
  name VARCHAR(255) PRIMARY KEY,
  description VARCHAR(255)
);

CREATE TABLE IF NOT EXISTS medicines (
  name VARCHAR(255) PRIMARY KEY,
  manufacturer VARCHAR(255),
```

```
  composition VARCHAR(255),
  rating DECIMAL(3,2)
);

CREATE TABLE IF NOT EXISTS pharmacies (
  state VARCHAR(255),
  address VARCHAR(255),
  city VARCHAR(255),
  id INT AUTO_INCREMENT PRIMARY KEY
);

CREATE TABLE IF NOT EXISTS side_effects (
  name VARCHAR(255) PRIMARY KEY,
  description VARCHAR(255)
);
```

FOR RELATIONSHIP TABLES:

```
CREATE TABLE IF NOT EXISTS currentlyHas(
  user_email VARCHAR(255),
  symptom_name VARCHAR(255),
  PRIMARY KEY (user_email, symptom_name),
  FOREIGN KEY (symptom_name) REFERENCES symptoms(name) ON DELETE CASCADE,
  FOREIGN KEY (user_email) REFERENCES users(email) ON DELETE CASCADE
);


CREATE TABLE IF NOT EXISTS treatedBy(
  medicine_name VARCHAR(255),
  symptom_name VARCHAR(255),
  PRIMARY KEY (symptom_name, medicine_name),
  FOREIGN KEY (symptom_name) REFERENCES symptoms(name) ON DELETE CASCADE,
  FOREIGN KEY (medicine_name) REFERENCES medicines(name) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS causedBy(
  medicine_name VARCHAR(255),
  side_effect_name VARCHAR(255),
  PRIMARY KEY (side_effect_name, medicine_name),
  FOREIGN KEY (side_effect_name) REFERENCES side_effects(name) ON DELETE
CASCADE,
  FOREIGN KEY (medicine_name) REFERENCES medicines(name) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS carries(
```

```
  medicine_name VARCHAR(255),
  pharmacy_id INT,
  PRIMARY KEY (pharmacy_id, medicine_name),
  FOREIGN KEY (pharmacy_id) REFERENCES pharmacies(id) ON DELETE CASCADE,
  FOREIGN KEY (medicine_name) REFERENCES medicines(name) ON DELETE CASCADE
);
```

INSERT >=1000 ROWS INTO ALL TABLES:

```
mysql> SELECT COUNT(*) FROM medicines;
+----------+
| COUNT(*) |
+----------+
|    11499 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(*) FROM pharmacies;
+----------+
| COUNT(*) |
+----------+
|    62973 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(*) FROM symptoms;
+----------+
| COUNT(*) |
+----------+
|     1005 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(*) FROM side_effects;
+----------+
| COUNT(*) |
+----------+
|     1160 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM users;
+----------+
| COUNT(*) |
+----------+
|     2001 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM currentlyHas;
+----------+
| COUNT(*) |
+----------+
|     4000 |
+----------+
1 row in set (0.00 sec)
```

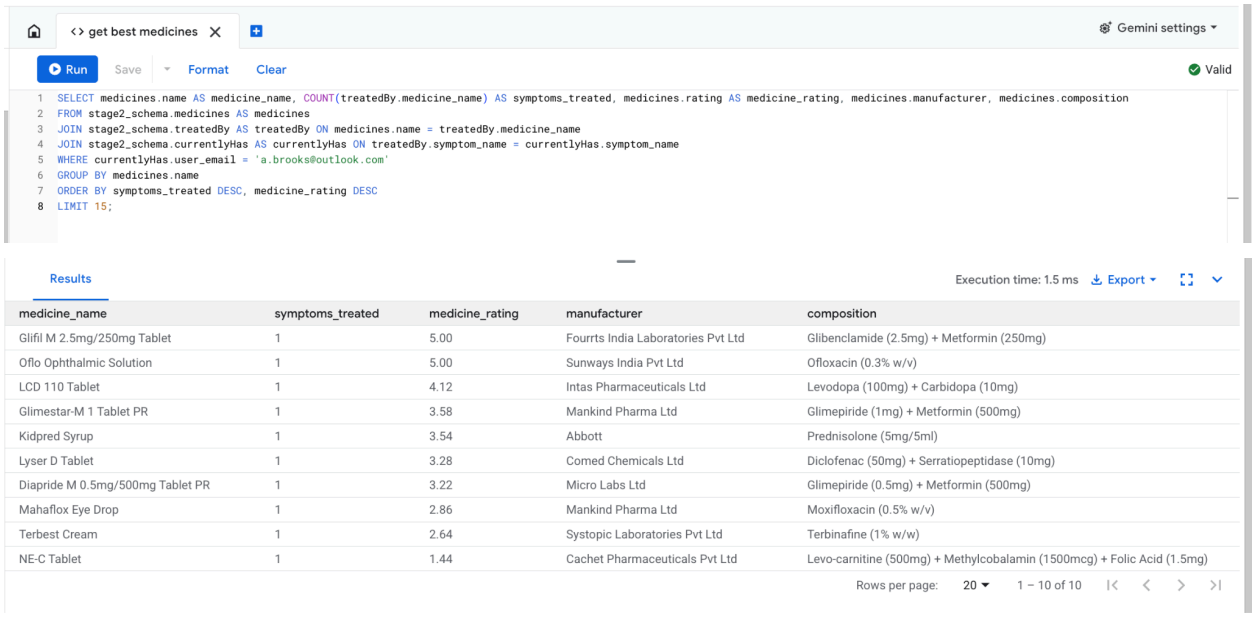
```
mysql> SELECT COUNT(*) FROM treatedBy;
+----------+
| COUNT(*) |
+----------+
|     5020 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(*) FROM causedBy;
+----------+
| COUNT(*) |
+----------+
|    13795 |
+----------+
1 row in set (0.04 sec)
```

```
mysql> SELECT COUNT(*) FROM carries;
+----------+
| COUNT(*) |
+----------+
|   628207 |
+----------+
1 row in set (0.05 sec)
```

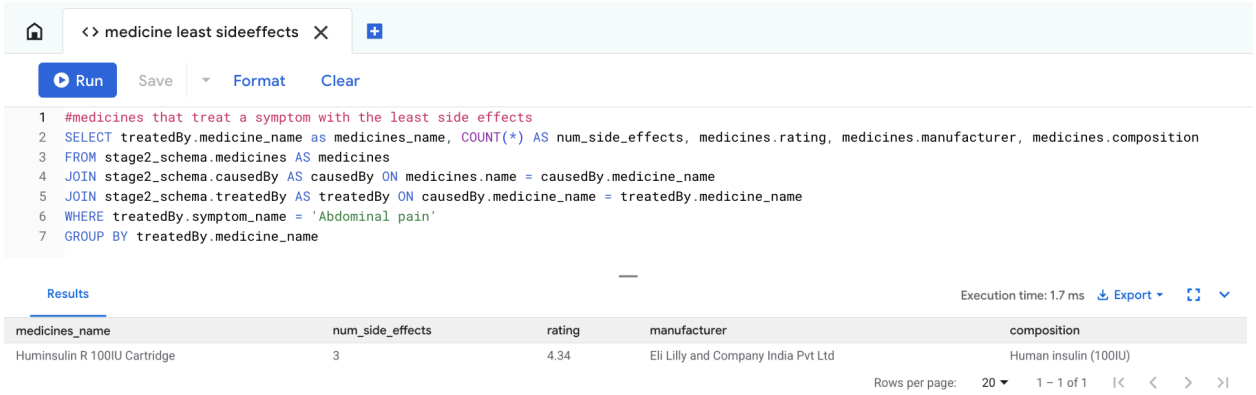
ADVANCED QUERIES (below)

1) GET BEST RATED MEDICINES



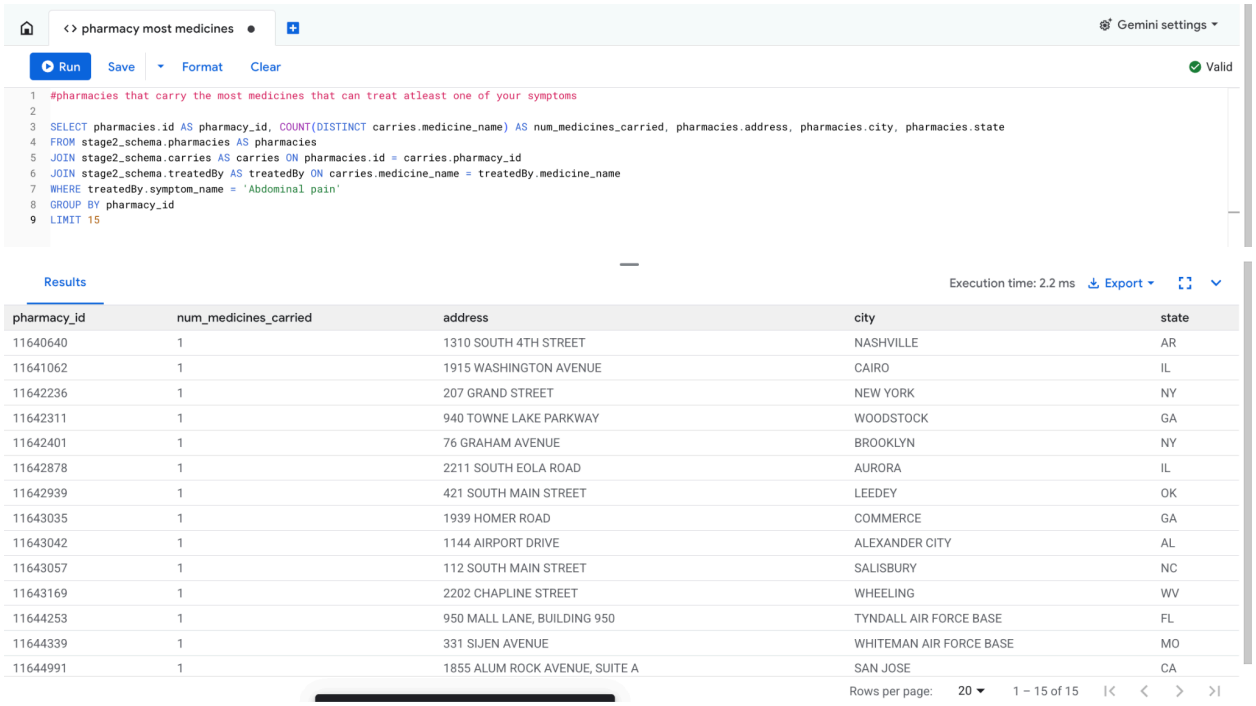
We think we get less than 15 results because of pure probability. We have 1005 symptoms and 11000 medicines. However, the symptoms that they treat are not optimally distributed, so it is not ~1000 medicines per symptom, but much less, especially taking into account that many user-reported symptoms are niche, such as “Abdominal Pain”.

2) GET MEDICINES WITH LEAST SIDE EFFECTS



Related to the reasoning for “get best rated medicine”, each integer band of side effects is broad. So there would only be a few medicines with the minimum number of side-effects. Here, Abdominal Pain only has around 5 medicines that treat it, and out of that, just having 1 with a minimum number makes sense. For symptoms such as cough, we would expect larger numbers, as many medicines treat it, and will tie for a minimum number of side effects. Even then, we expect not more than 8-10 tying medicines that all treat the same cure with the same minimum amount of side_effects. We had some GCP credit and bucket issues - but will create larger datasets as we get that sorted for stage 4. However, this advanced query, as with all our others, works as expected.

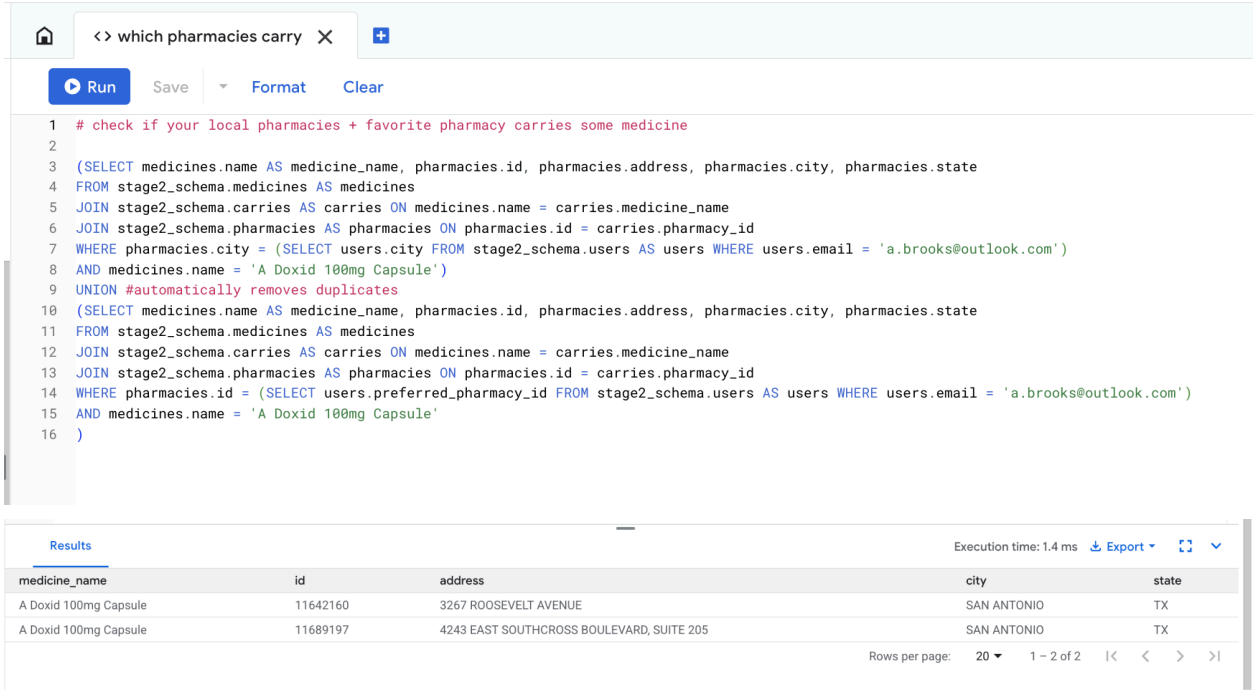
3) GET THE PHARMACIES IN USER’S CITY/STATE WITH THE MOST MEDICINES



Notice that each resulting query has only 1 medicine that treats the user. Each Pharmacy only carries a small set {5,15} medicines, in our synthetic data. Given that are ~63,000 pharmacies that carry a random sample of {5,15} out of ~11,500 medicines, the

probability that a user's symptom is treated by pharmacies in a given state {on average, a state seems to have 500 pharmacies}, is low. However, in stage 4, we will synthesize a larger number of medicines per pharmacy, and this number should be much larger. However, even with the current data, we have 274 pharmacies this user can go to in their US state/territory. So we expect to get 15+ for this, unless the state/territory itself has less than 100 pharmacies.

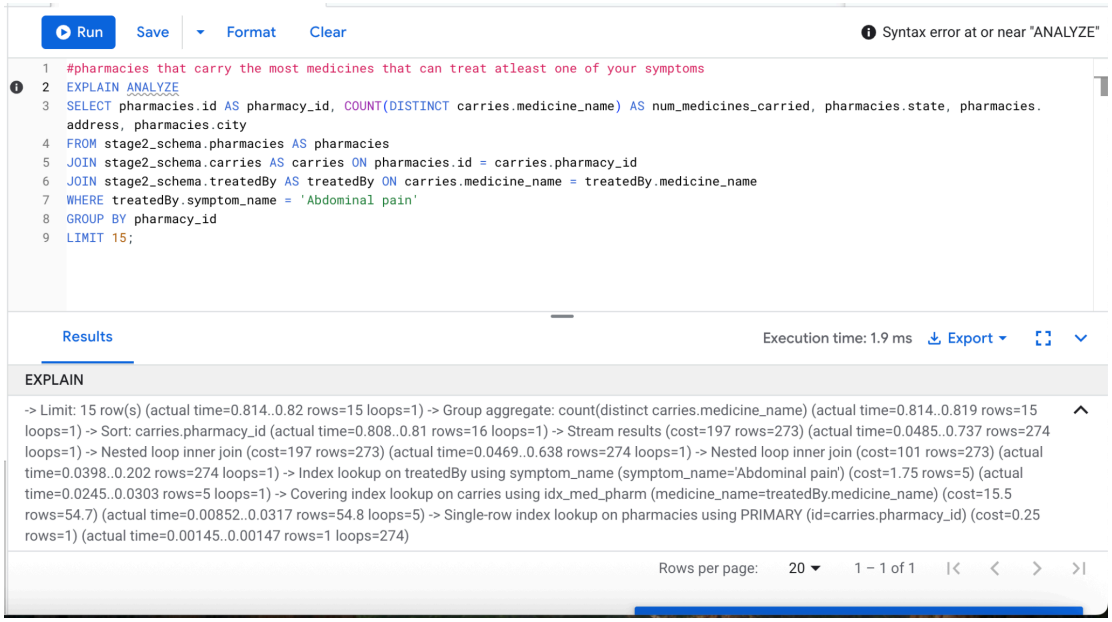
4) GET THE PHARMACIES THAT CARRY MEDICINES TO CURE USER



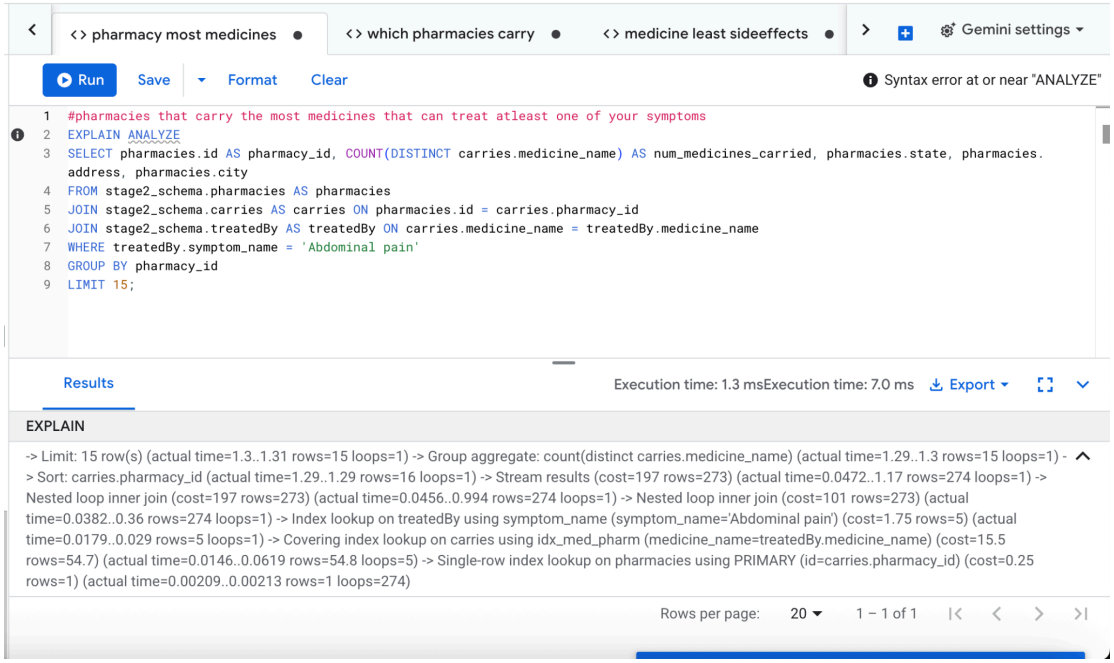
This is the most specific of our queries, and has a lot of specific predicates to check whether a certain favorite pharmacy of the current user carries medicine that treats the current user's current symptom. We expected this to be the least, but as we will synthesize extensions for a lot of our datasets, we expect to see a few more queries. However, we think that 15+ will still be unlikely for this one.

INDEX ANALYSIS:

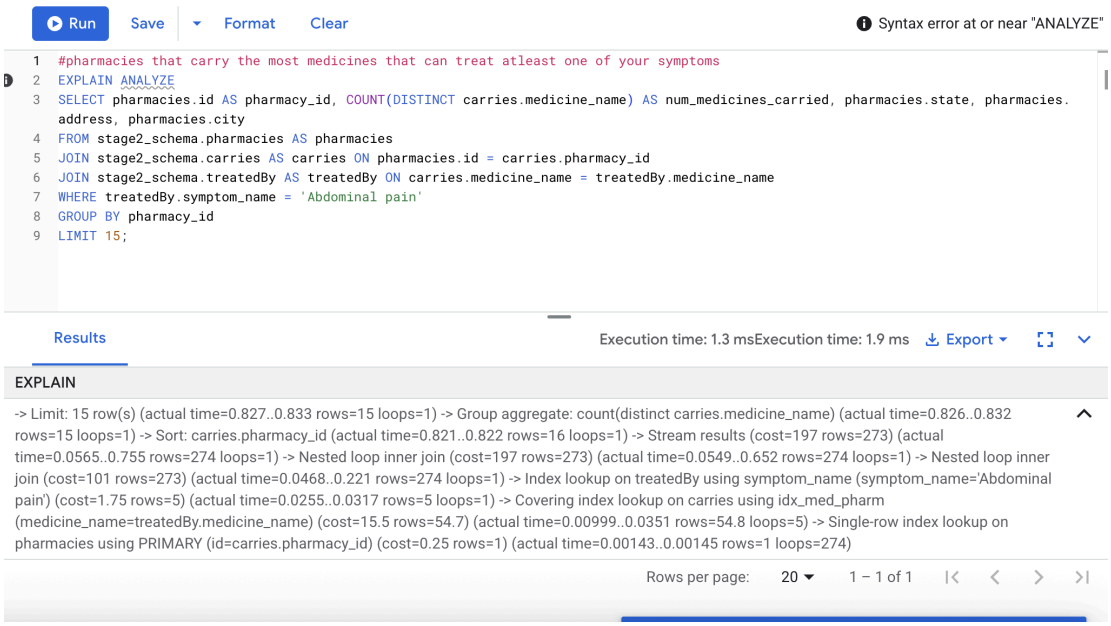
Pharmacy Most Medicine Default:



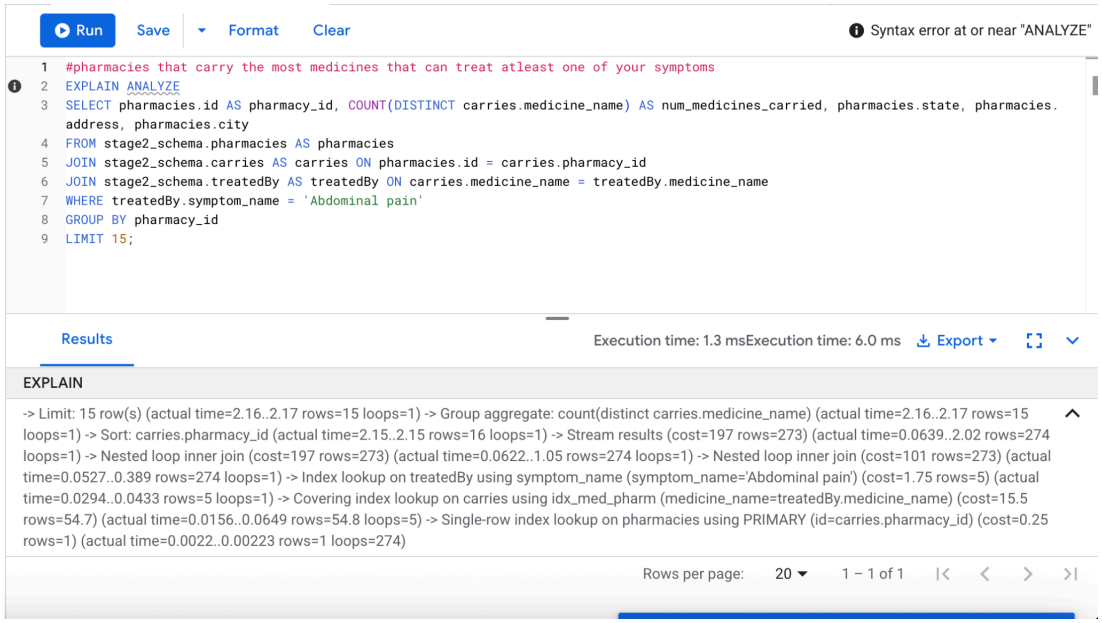
Pharmacy Most Medicine + Pharmacy.Address:



Pharmacy Most Medicine + Users.city + Pharmacies.city

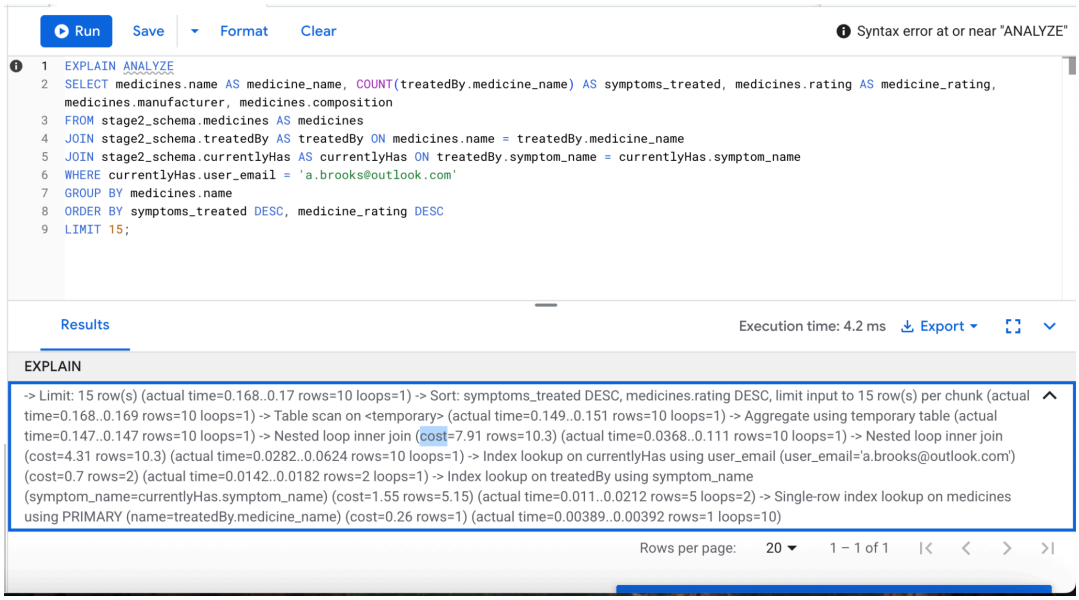


Pharmacy Most Medicine + Users.state + Pharmacies.state

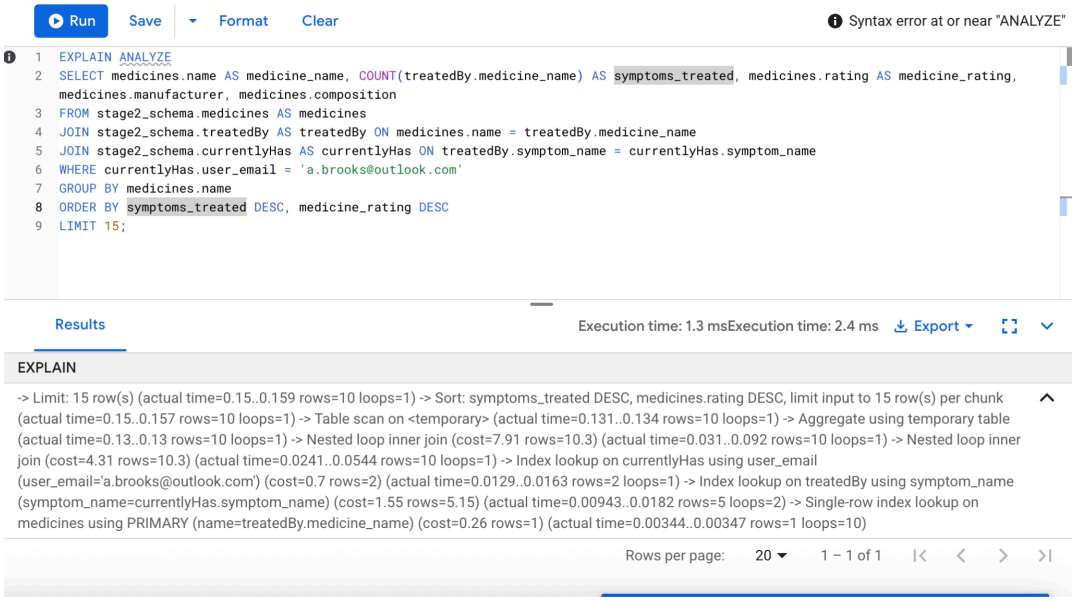


For the pharmacy most medicine query, the default cost is 197. We experimented with indexes on pharmacy.address, users.city + pharmacies.city, and users.state + pharmacies.state. None of these indexing options reduced the query cost. This suggests that the query planner already chooses an optimal access path—likely dominated by join operations or aggregates where filtering by address, city, or state contributes little selectivity. Therefore, the default indexing strategy remains optimal.

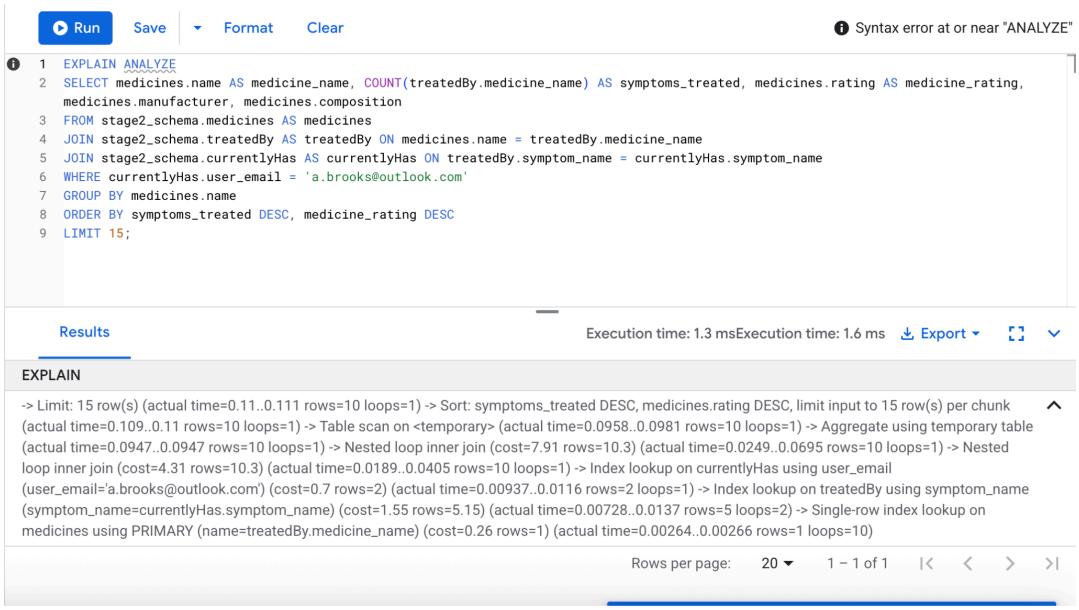
Get Best Medicine Default:



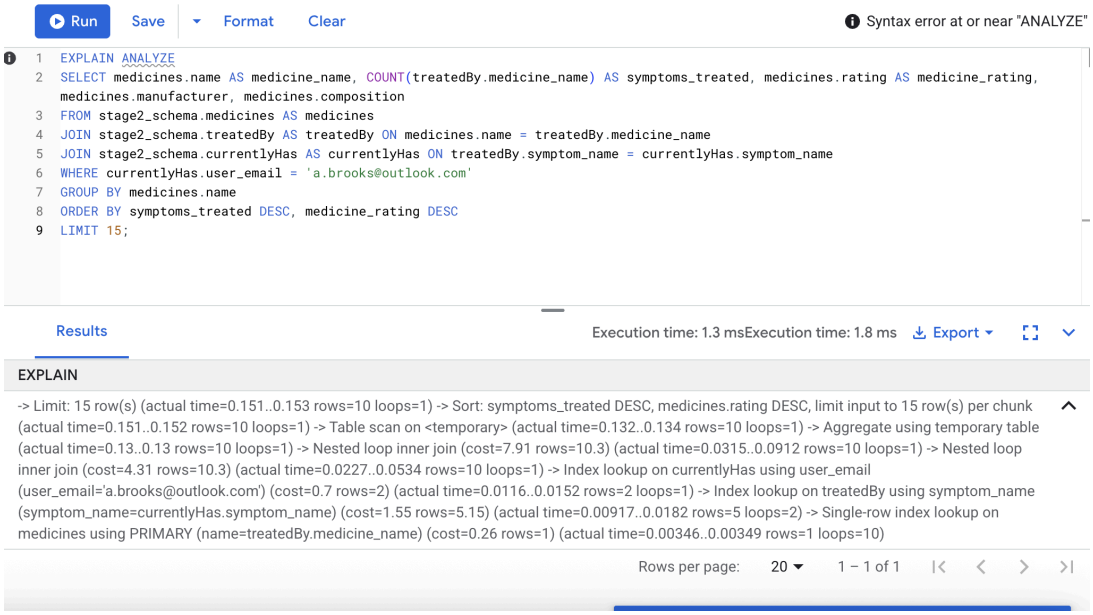
Get Best Medicine + Medicine.ratings:



Get Best Medicine + Medicine.manufacturer:

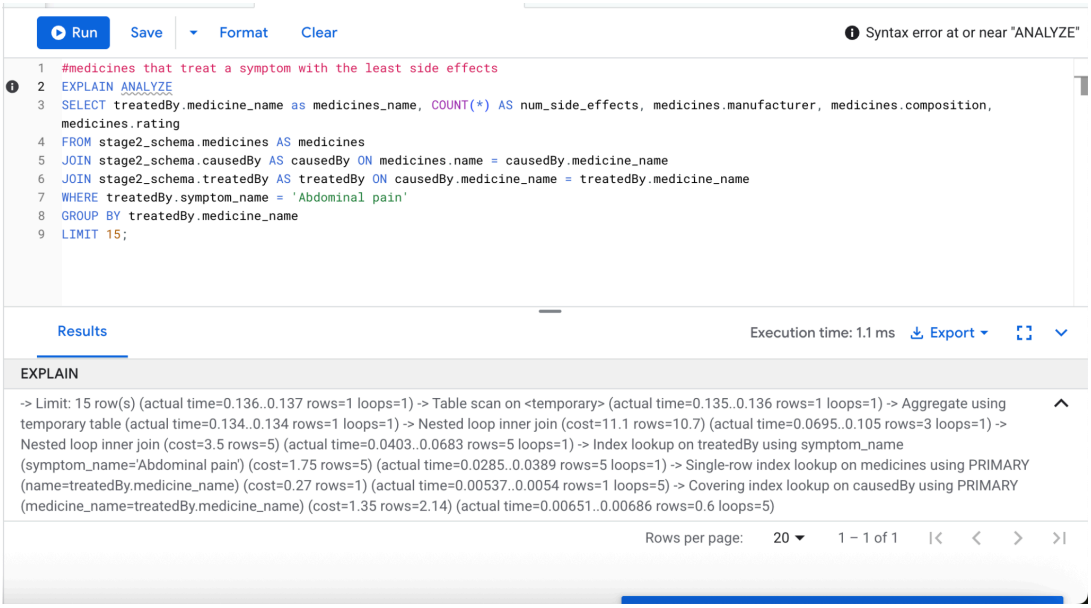


Get Best Medicine + Medicine.composition:

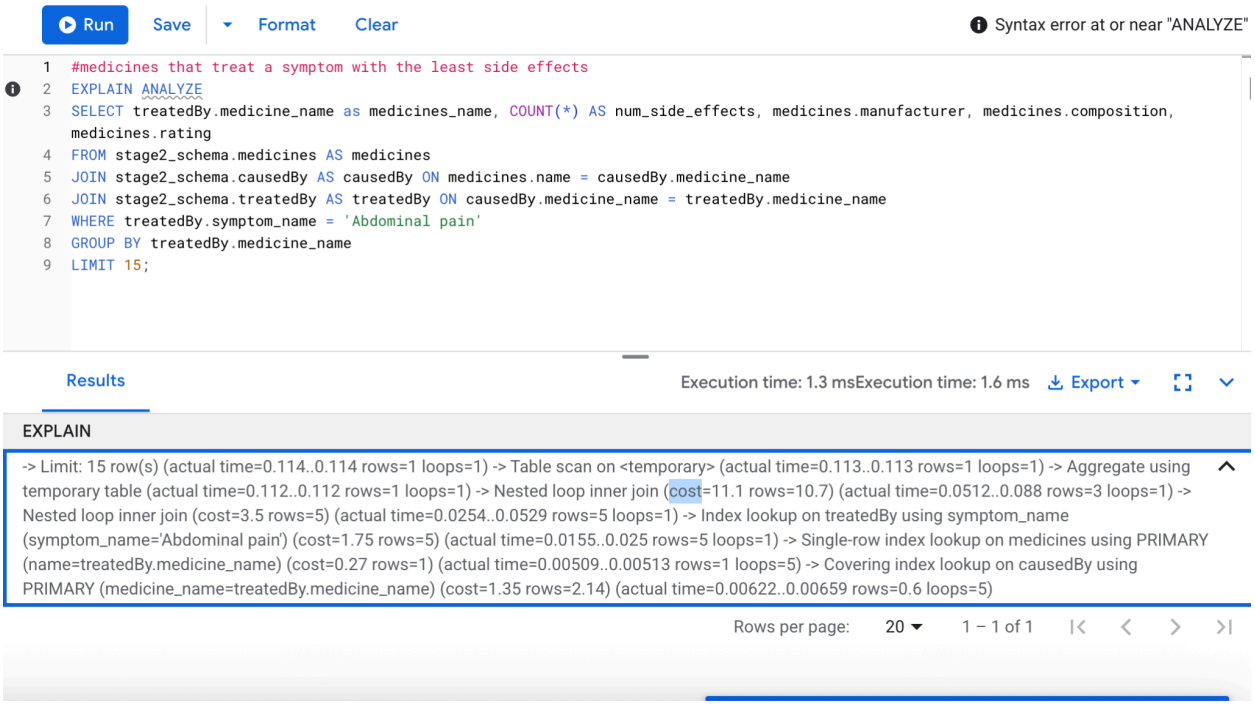


For the get best medicine query, the default cost is 7.91. We tested additional indexes on medicine.ratings, medicine.manufacturer, and medicine.composition. Since all versions retained the same estimated cost, these attributes do not constrain the query enough to make index lookups more efficient than the existing plan. Hence, the default indexing is already optimal for this query.

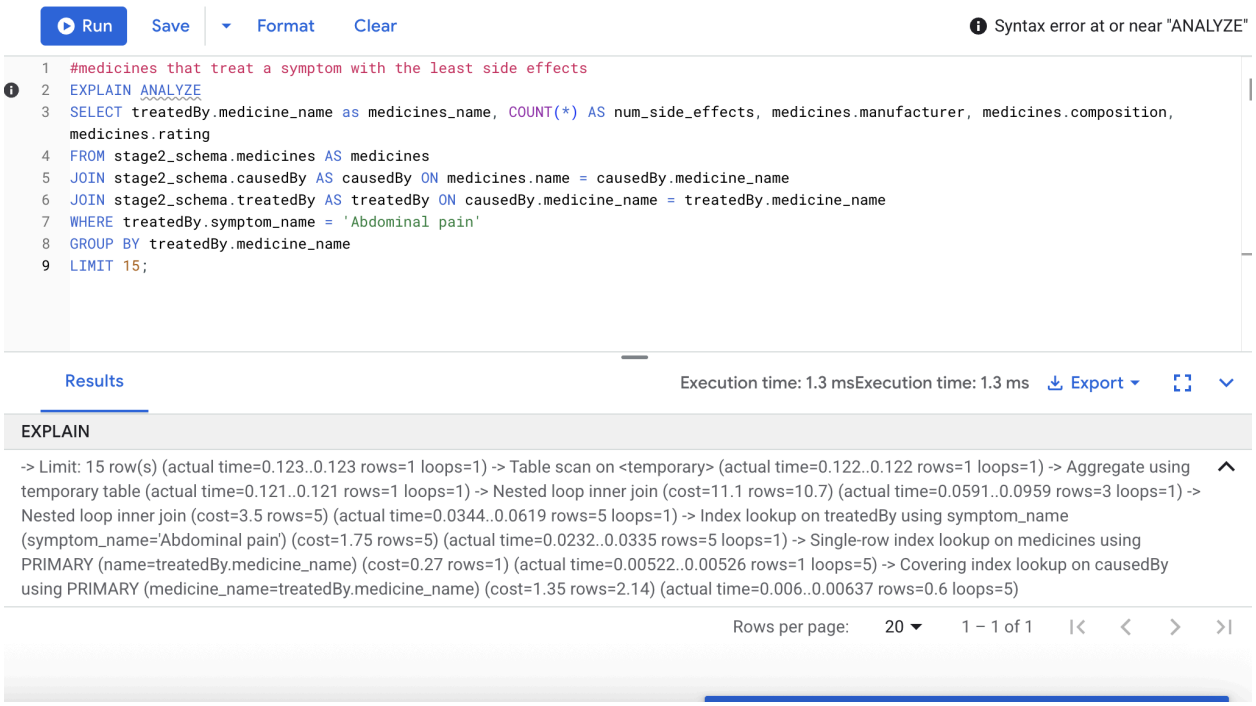
Medicine Least Side Effects Default:



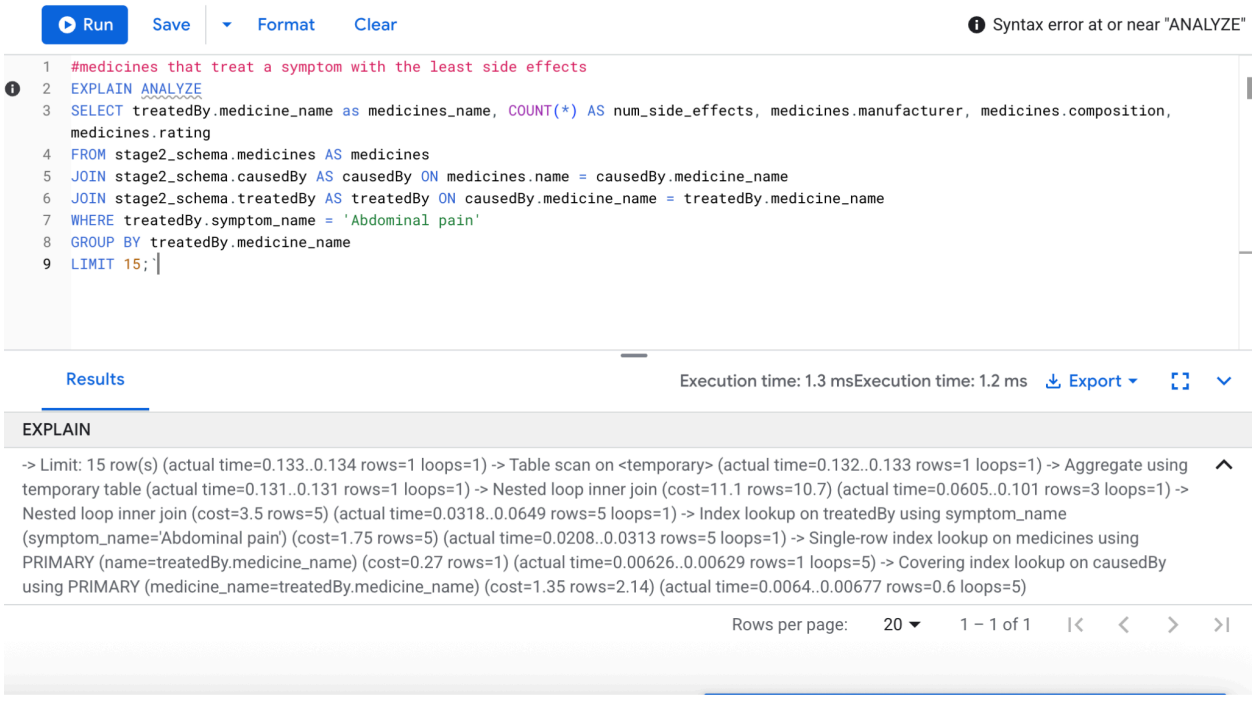
Medicine Least Side Effects + Medicine.ratings:



Medicine Least Side Effects + Medicine.composition:



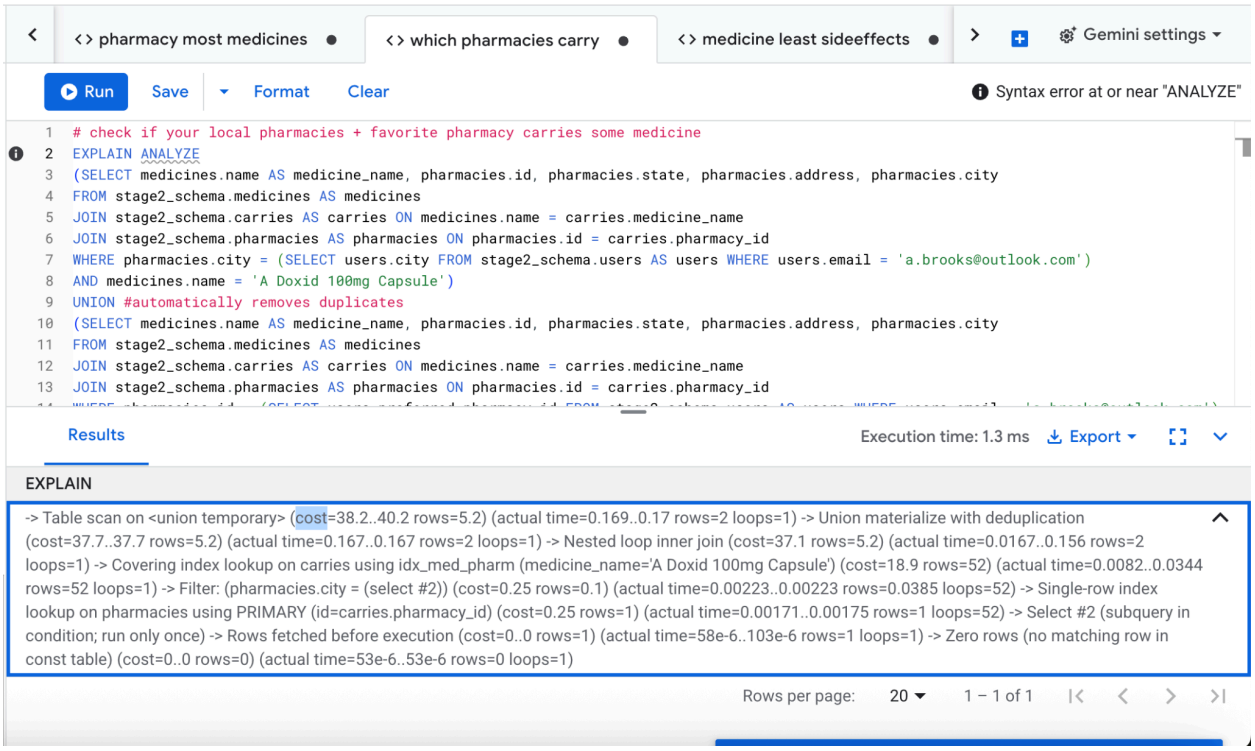
Medicine Least Side Effects + Medicine.manufacturer:



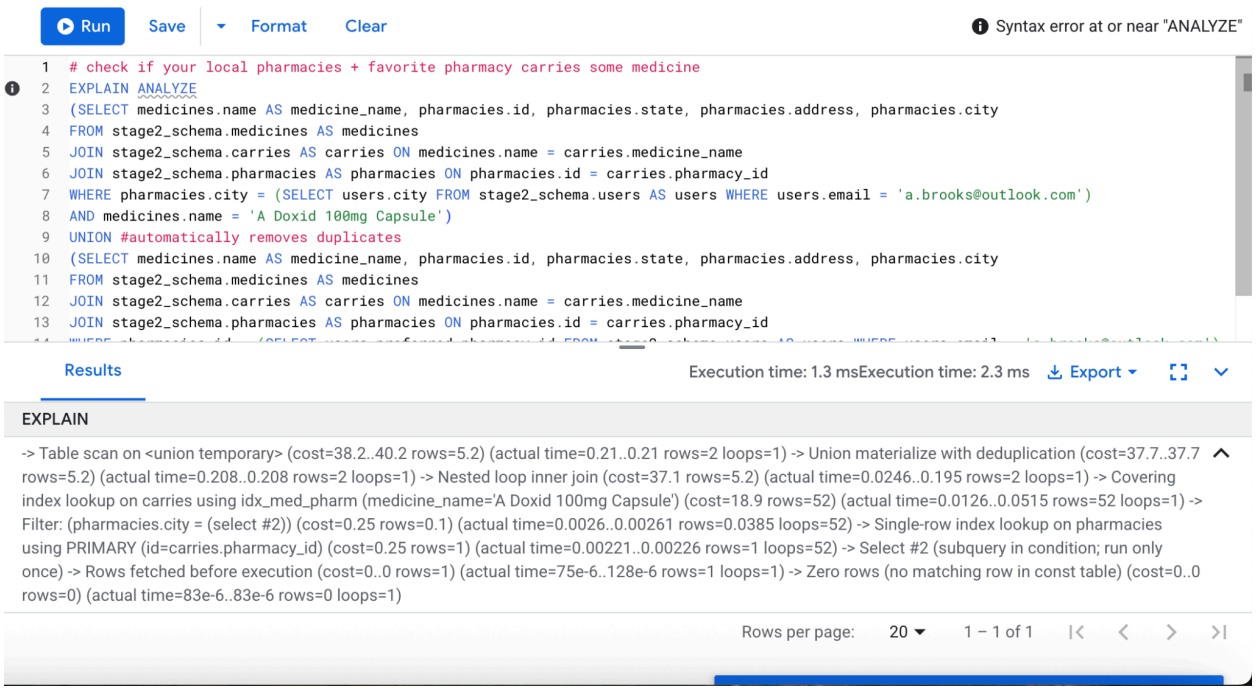
For the medicine least side effects query, the default cost is 11.1. We tried indexing medicine.ratings, medicine.composition and medicine.manufacturer. Neither option improved the cost, indicating that the query's filtering or ordering criteria are already efficiently handled

without additional indexes. The query optimizer continues to prefer sequential or existing indexed access paths, so the default indexing remains optimal.

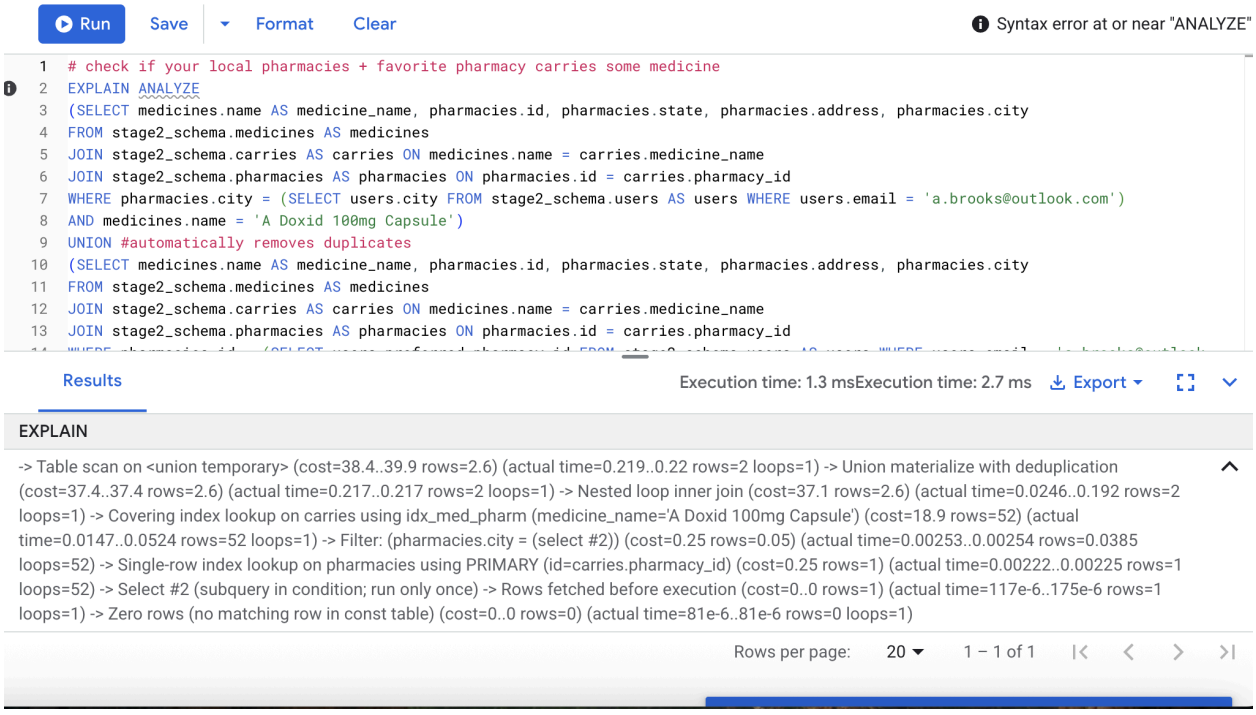
Which Pharmacies Carry Default:



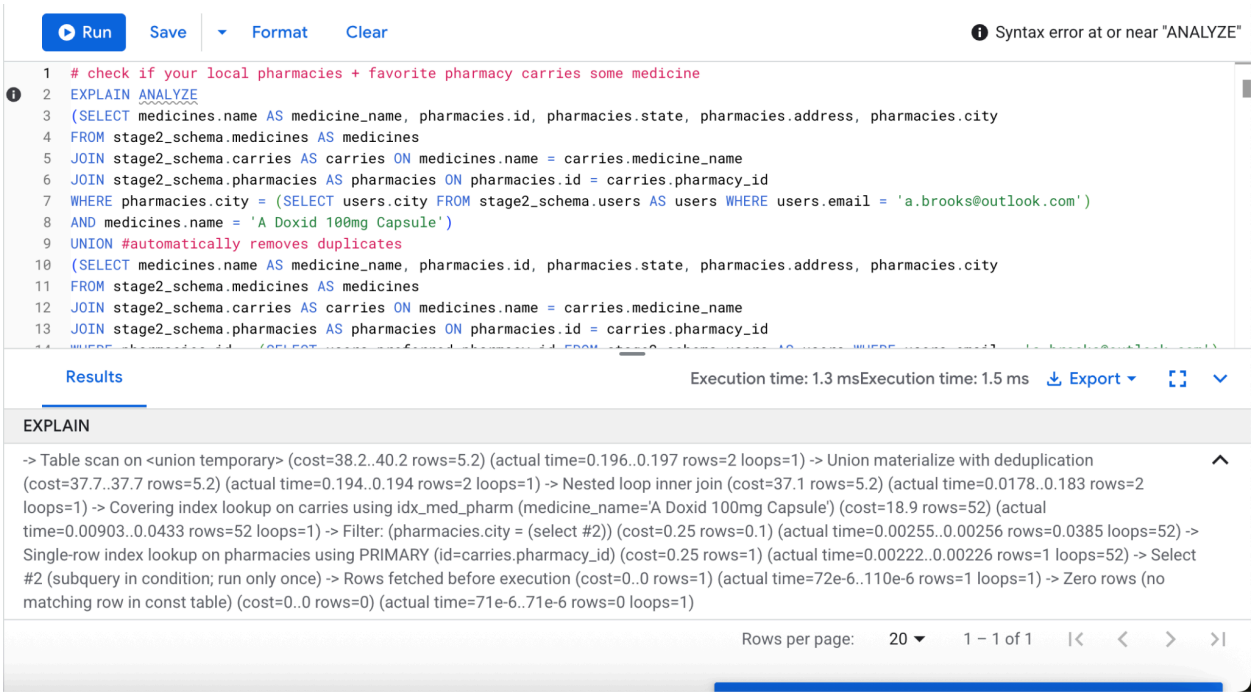
Which Pharmacies Carry + Pharmacy.address:



Which Pharmacies Carry + Pharmacies.city + Users.city:



Which Pharmacies Carry + Pharmacies.state + Users.state:



For the which pharmacies carry query, the default cost is 38.2. We experimented with indexes on pharmacy.address, pharmacies.city + users.city, and users.state + pharmacies.state. Among these, indexing on pharmacies.city + users.city slightly increased the cost to 38.4. This minor increase is likely due to the query's use of a UNION operator, which requires a deduplication step involving sorting or hashing across both result sets. Since the additional index adds lookup overhead without improving selectivity, the planner estimates a slightly higher cost. Overall, the default indexing strategy is still the most efficient for this query.