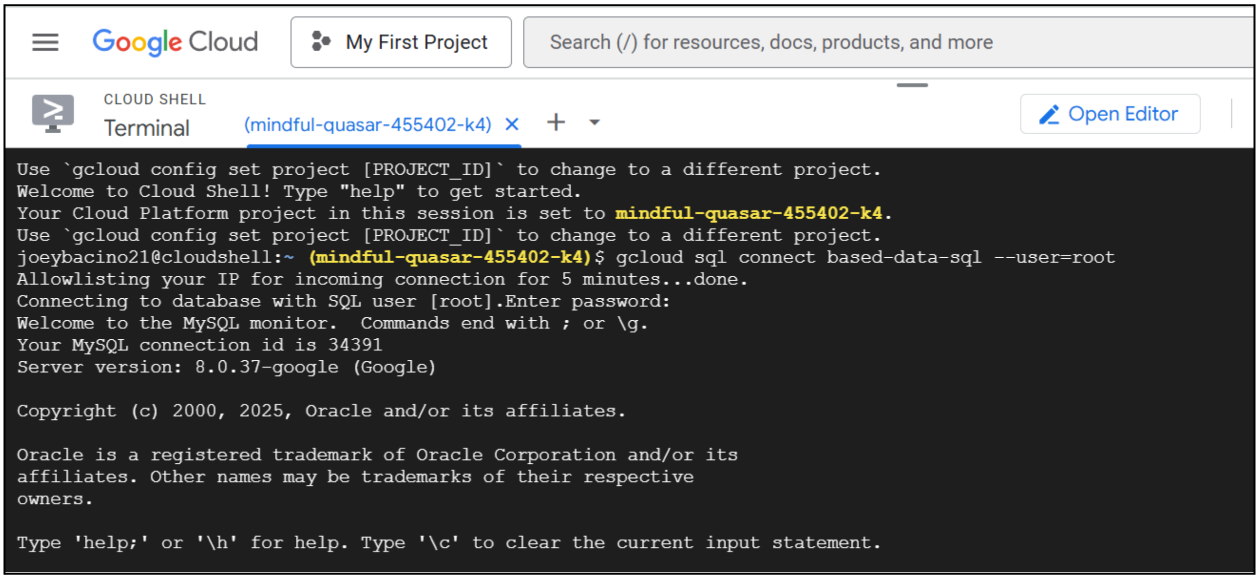
CS 411 Stage 3 – Team 64

Below are all the required screenshots for Stage 3 of our project, MedMatch.

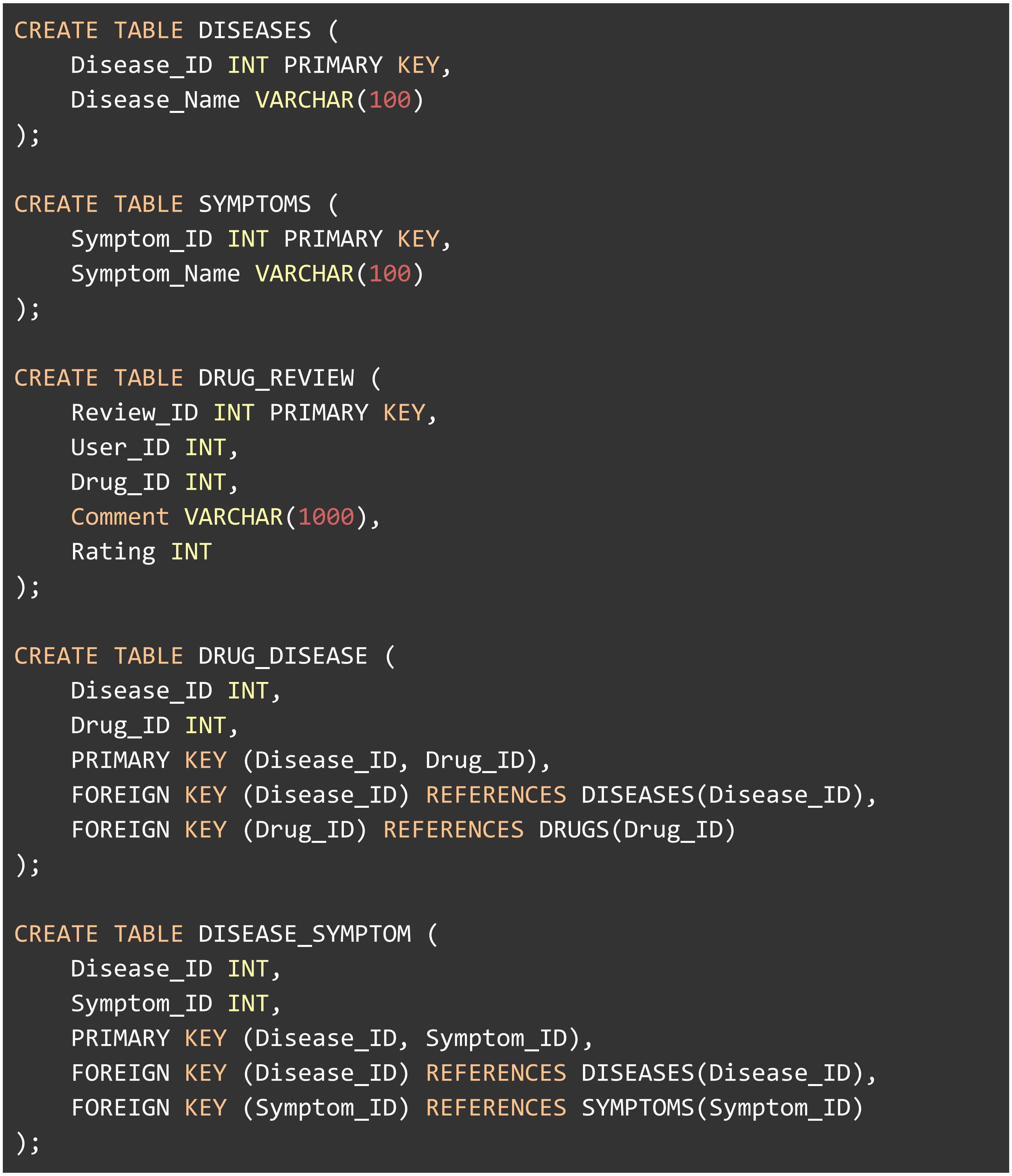
Implementation of the database tables on GCP:


Google Cloud
My First Project
Search (/) for resources, docs, products, and more
CLOUD SHELL
Terminal
(mindful-quasar-455402-k4)
Open Editor

```
Use `gcloud config set project [PROJECT_ID]` to change to a different project.
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to mindful-quasar-455402-k4.
Use `gcloud config set project [PROJECT_ID]` to change to a different project.
joeybacino21@cloudshell:~ (mindful-quasar-455402-k4)$ gcloud sql connect based-data-sql --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 34391
Server version: 8.0.37-google (Google)

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

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

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

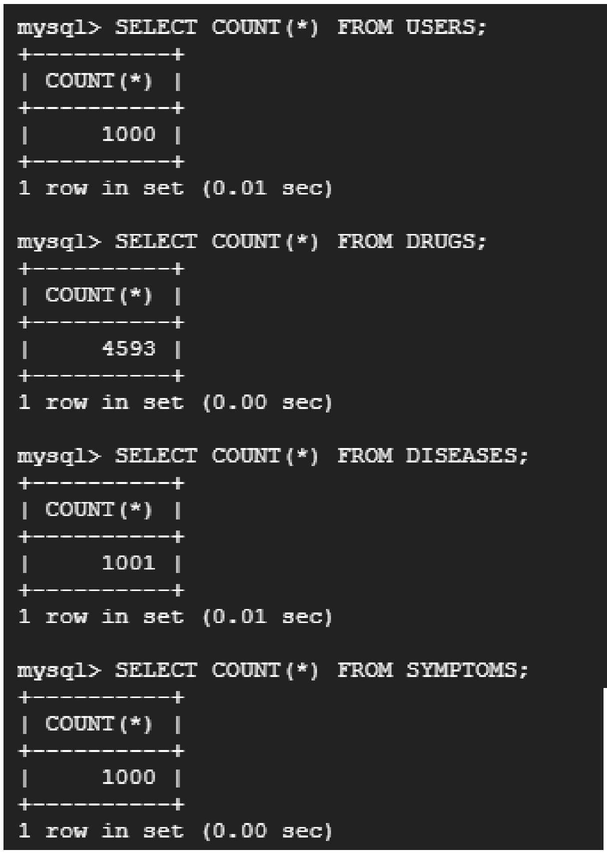
DDL commands for all tables:

```
CREATE TABLE USERS (
    User_ID INT PRIMARY KEY,
    Username VARCHAR(100),
    Age INT,
    Gender VARCHAR(6)
);

CREATE TABLE DRUGS (
    Drug_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Type VARCHAR(50)
);
```

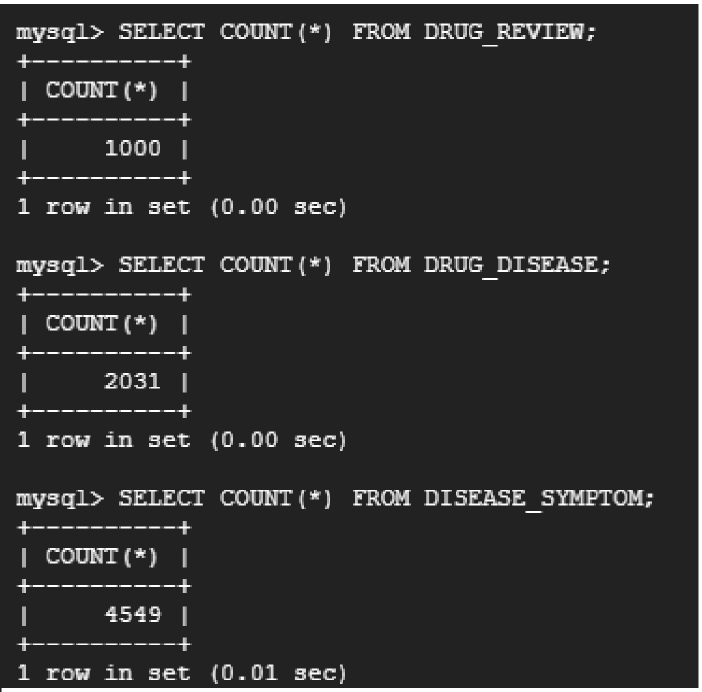


```sql
CREATE TABLE DISEASES (
    Disease_ID INT PRIMARY KEY,
    Disease_Name VARCHAR(100)
);

CREATE TABLE SYMPTOMS (
    Symptom_ID INT PRIMARY KEY,
    Symptom_Name VARCHAR(100)
);

CREATE TABLE DRUG_REVIEW (
    Review_ID INT PRIMARY KEY,
    User_ID INT,
    Drug_ID INT,
    Comment VARCHAR(1000),
    Rating INT
);

CREATE TABLE DRUG_DISEASE (
    Disease_ID INT,
    Drug_ID INT,
    PRIMARY KEY (Disease_ID, Drug_ID),
    FOREIGN KEY (Disease_ID) REFERENCES DISEASES(Disease_ID),
    FOREIGN KEY (Drug_ID) REFERENCES DRUGS(Drug_ID)
);

CREATE TABLE DISEASE_SYMPTOM (
    Disease_ID INT,
    Symptom_ID INT,
    PRIMARY KEY (Disease_ID, Symptom_ID),
    FOREIGN KEY (Disease_ID) REFERENCES DISEASES(Disease_ID),
    FOREIGN KEY (Symptom_ID) REFERENCES SYMPTOMS(Symptom_ID)
);
```

At least 1000 rows in the tables:



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

mysql> SELECT COUNT(*) FROM DRUGS;
+----------+
| COUNT(*) |
+----------+
|     4593 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM DISEASES;
+----------+
| COUNT(*) |
+----------+
|     1001 |
+----------+
1 row in set (0.01 sec)

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



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

mysql> SELECT COUNT(*) FROM DRUG_DISEASE;
+----------+
| COUNT(*) |
+----------+
|     2031 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM DISEASE_SYMPTOM;
+----------+
| COUNT(*) |
+----------+
|     4549 |
+----------+
1 row in set (0.01 sec)
```

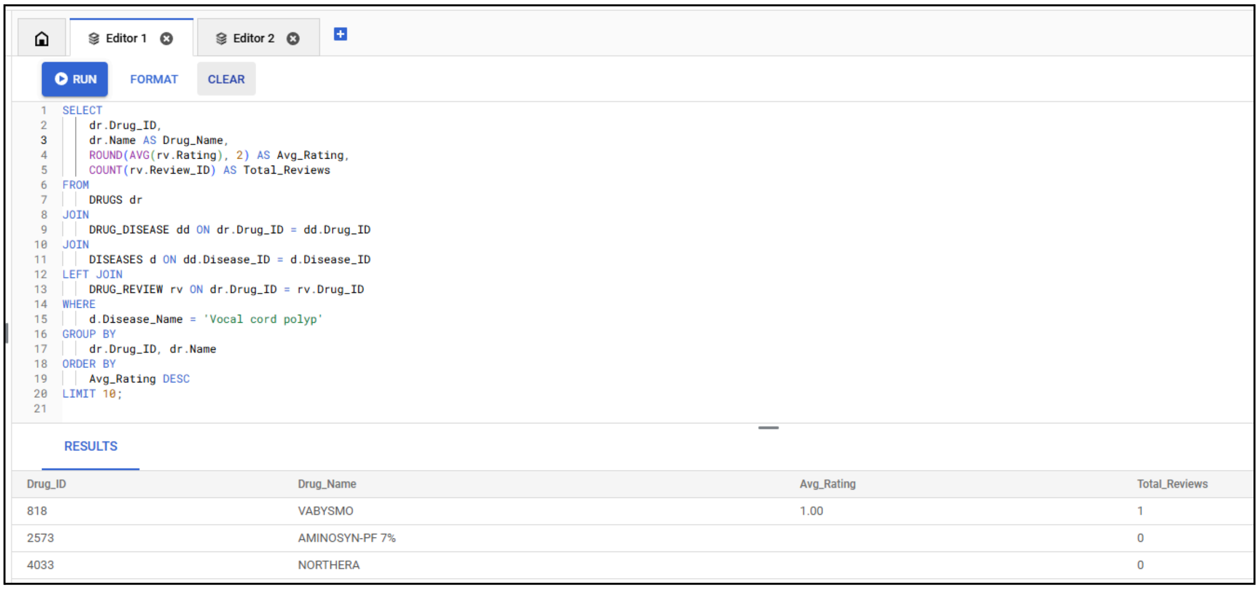
Our 4 advanced queries with screenshots of the query results:

Query 1: Drugs for a specific disease, ordered by rating (if no rating, at bottom)

```sql
SELECT dr.Drug_ID, dr.Name AS Drug_Name, ROUND(AVG(rv.Rating), 2) AS
Avg_Rating, COUNT(rv.Review_ID) AS Total_Reviews

FROM DRUGS dr

JOIN DRUG_DISEASE dd ON dr.Drug_ID = dd.Drug_ID
JOIN DISEASES d ON dd.Disease_ID = d.Disease_ID
LEFT JOIN DRUG_REVIEW rv ON dr.Drug_ID = rv.Drug_ID

WHERE d.Disease_Name = 'Vocal cord polyp'

GROUP BY dr.Drug_ID, dr.Name
```

```
ORDER BY Avg_Rating DESC

LIMIT 10;
```

Query 1 Result: We only have 3 rows displayed because there are only 3 drugs associated with the specific disease chosen.



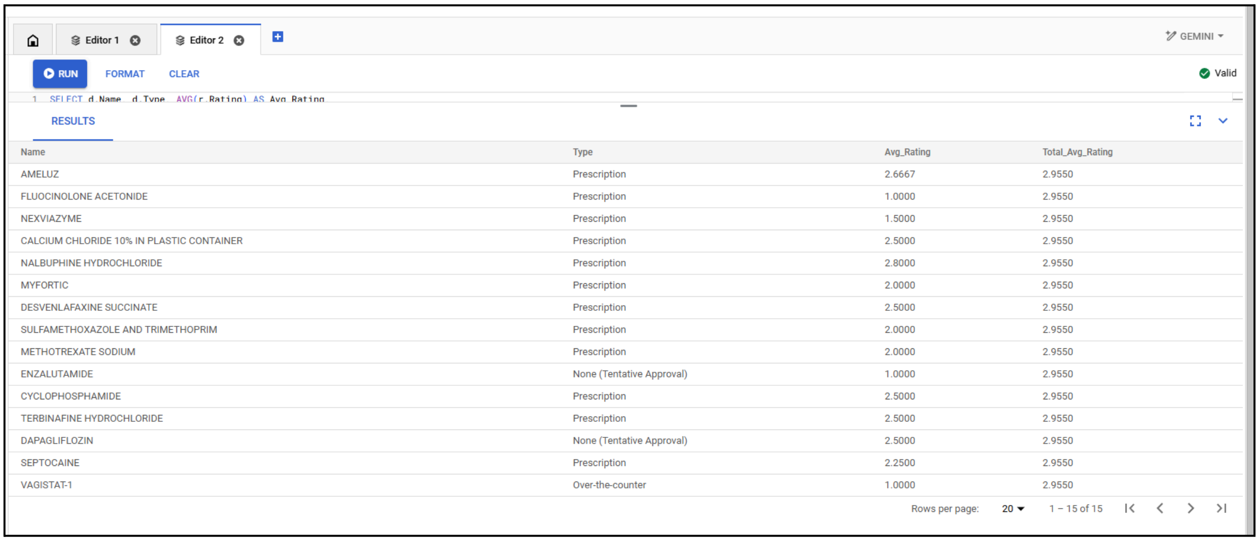
```
SELECT
    dr.Drug_ID,
    dr.Name AS Drug_Name,
    ROUND(AVG(rv.Rating), 2) AS Avg_Rating,
    COUNT(rv.Review_ID) AS Total_Reviews
FROM
    DRUGS dr
JOIN
    DRUG_DISEASE dd ON dr.Drug_ID = dd.Drug_ID
JOIN
    DISEASES d ON dd.Disease_ID = d.Disease_ID
LEFT JOIN
    DRUG_REVIEW rv ON dr.Drug_ID = rv.Drug_ID
WHERE
    d.Disease_Name = 'Vocal cord polyp'
GROUP BY
    dr.Drug_ID, dr.Name
ORDER BY
    Avg_Rating DESC
LIMIT 10;
```

Drug_ID	Drug_Name	Avg_Rating	Total_Reviews
818	VABYSMO	1.00	1
2573	AMINOSYN-PF 7%		0
4033	NORTHERA		0

Query 2: Drugs whose average rating is below the category average

```
SELECT d.Name, d.Type, AVG(r.Rating) AS Avg_Rating,
    (SELECT AVG(r3.Rating) FROM DRUG_REVIEW r3) AS Total_Avg_Rating

FROM DRUGS d

JOIN DRUG_REVIEW r ON d.Drug_ID = r.Drug_ID

GROUP BY d.Drug_ID, d.Name, d.Type

HAVING AVG(r.Rating) <
    (SELECT AVG(r2.Rating)
    FROM DRUG_REVIEW r2
    JOIN DRUGS d2 ON r2.Drug_ID = d2.Drug_ID
    WHERE d2.Type = d.Type);
```

Query 2 Result:



```
1  SELECT d.Name, d.Type, AVG(r.Rating) AS Avg_Rating
```

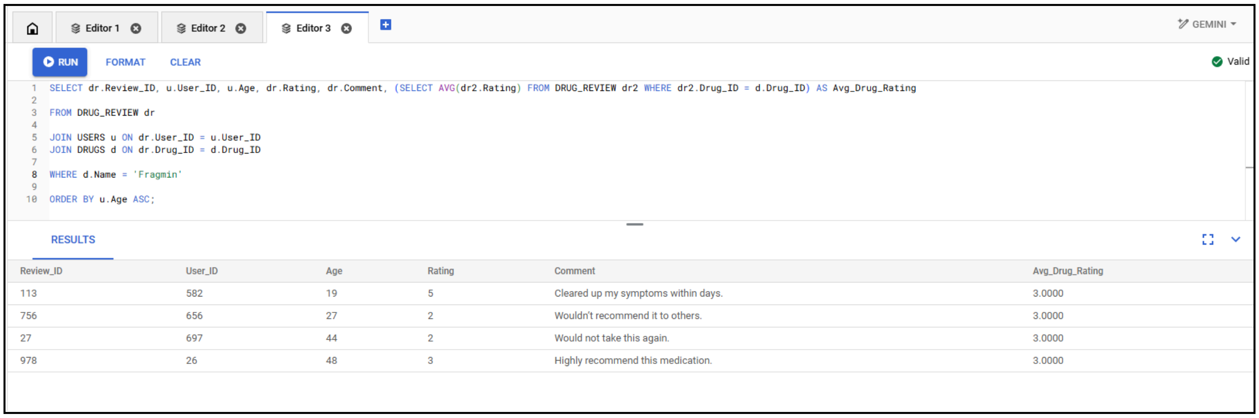
Name	Type	Avg_Rating	Total_Avg_Rating
AMELUZ	Prescription	2.6667	2.9550
FLUOCINOLONE ACETONIDE	Prescription	1.0000	2.9550
NEXVIAZYME	Prescription	1.5000	2.9550
CALCIUM CHLORIDE 10% IN PLASTIC CONTAINER	Prescription	2.5000	2.9550
NALBUPHINE HYDROCHLORIDE	Prescription	2.8000	2.9550
MYFORTIC	Prescription	2.0000	2.9550
DESVENLAFAXINE SUCCINATE	Prescription	2.5000	2.9550
SULFAMETHOXAZOLE AND TRIMETHOPRIM	Prescription	2.0000	2.9550
METHOTREXATE SODIUM	Prescription	2.0000	2.9550
ENZALUTAMIDE	None (Tentative Approval)	1.0000	2.9550
CYCLOPHOSPHAMIDE	Prescription	2.5000	2.9550
TERBINAFINE HYDROCHLORIDE	Prescription	2.5000	2.9550
DAPAGLIFLOZIN	None (Tentative Approval)	2.5000	2.9550
SEPTOCAINE	Prescription	2.2500	2.9550
VAGISTAT-1	Over-the-counter	1.0000	2.9550

Rows per page: 20 1 – 15 of 15

Query 3: All reviews for a specific drug, ordered by age with the average drug rating

```sql
SELECT dr.Review_ID, u.User_ID, u.Age, dr.Rating, dr.Comment, (SELECT AVG(dr2.Rating) FROM DRUG_REVIEW dr2 WHERE dr2.Drug_ID = d.Drug_ID) AS Avg_Drug_Rating

FROM DRUG_REVIEW dr

JOIN USERS u ON dr.User_ID = u.User_ID
JOIN DRUGS d ON dr.Drug_ID = d.Drug_ID

WHERE d.Name = 'Ibuprofen'

ORDER BY u.Age ASC;
```

Query 3 Result: There are only 4 rows due to there only being 4 reviews for this drug.



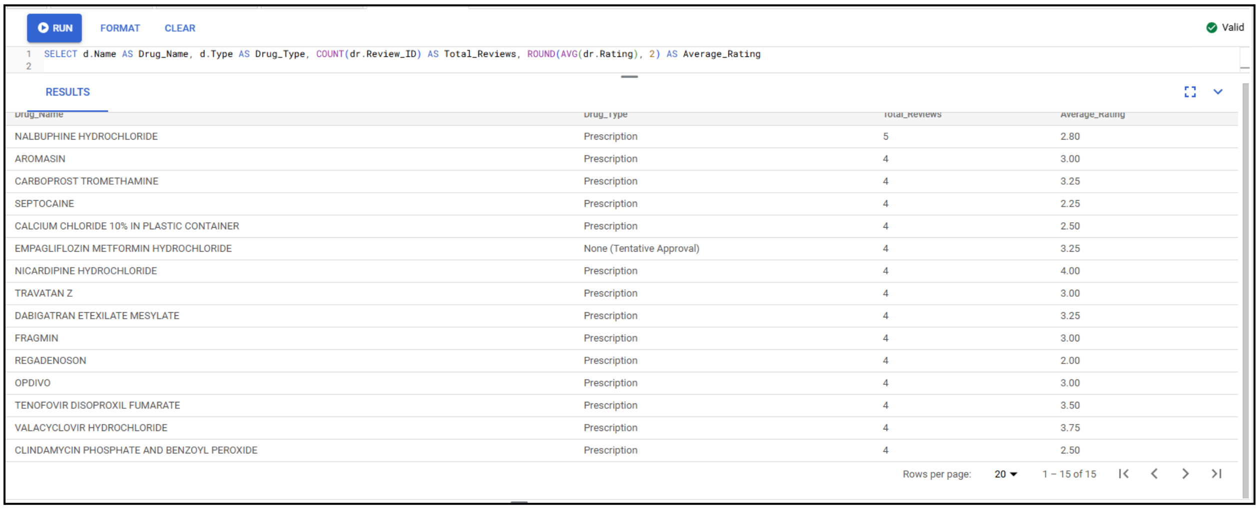
```sql
SELECT dr.Review_ID, u.User_ID, u.Age, dr.Rating, dr.Comment, (SELECT AVG(dr2.Rating) FROM DRUG_REVIEW dr2 WHERE dr2.Drug_ID = d.Drug_ID) AS Avg_Drug_Rating

FROM DRUG_REVIEW dr

JOIN USERS u ON dr.User_ID = u.User_ID
JOIN DRUGS d ON dr.Drug_ID = d.Drug_ID

WHERE d.Name = 'Fragmin'

ORDER BY u.Age ASC;
```

Review_ID	User_ID	Age	Rating	Comment	Avg_Drug_Rating
113	582	19	5	Cleared up my symptoms within days.	3.0000
756	656	27	2	Wouldn't recommend it to others.	3.0000
27	697	44	2	Would not take this again.	3.0000
978	26	48	3	Highly recommend this medication.	3.0000

Query 4: Most reviewed drugs, sorted by number of reviews, with the average rating

```sql
SELECT d.Name AS Drug_Name, d.Type AS Drug_Type, COUNT(dr.Review_ID)
AS Total_Reviews, ROUND(AVG(dr.Rating), 2) AS Average_Rating

FROM DRUGS d

JOIN DRUG_REVIEW dr ON d.Drug_ID = dr.Drug_ID

GROUP BY d.Drug_ID, d.Name, d.Type

ORDER BY Total_Reviews DESC;
```

Query 4 Result:



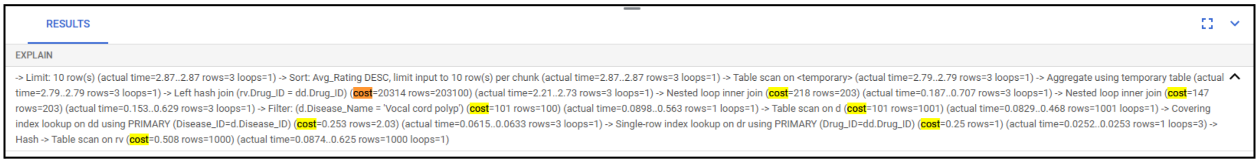
```
SELECT d.Name AS Drug_Name, d.Type AS Drug_Type, COUNT(dr.Review_ID) AS Total_Reviews, ROUND(AVG(dr.Rating), 2) AS Average_Rating
```

Drug_Name	Drug_Type	Total_Reviews	Average_Rating
NALBUPHINE HYDROCHLORIDE	Prescription	5	2.80
AROMASIN	Prescription	4	3.00
CARBOPROST TROMETHAMINE	Prescription	4	3.25
SEPTOCAINE	Prescription	4	2.25
CALCIUM CHLORIDE 10% IN PLASTIC CONTAINER	Prescription	4	2.50
EMPAGLIFLOZIN METFORMIN HYDROCHLORIDE	None (Tentative Approval)	4	3.25
NICARDIPINE HYDROCHLORIDE	Prescription	4	4.00
TRAVATAN Z	Prescription	4	3.00
DABIGATRAN ETEXILATE MESYLATE	Prescription	4	3.25
FRAGMIN	Prescription	4	3.00
REGADENOSON	Prescription	4	2.00
OPDIVO	Prescription	4	3.00
TENOFOVIR DISOPROXIL FUMARATE	Prescription	4	3.50
VALACYCLOVIR HYDROCHLORIDE	Prescription	4	3.75
CLINDAMYCIN PHOSPHATE AND BENZOYL PEROXIDE	Prescription	4	2.50

Three different indexing designs (+ default) for each query:

Query 1:

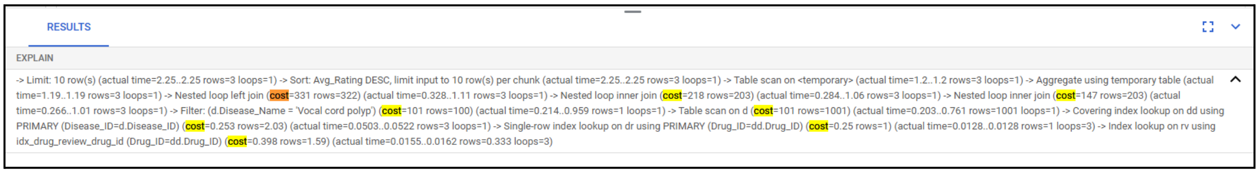
Default Index:



EXPLAIN

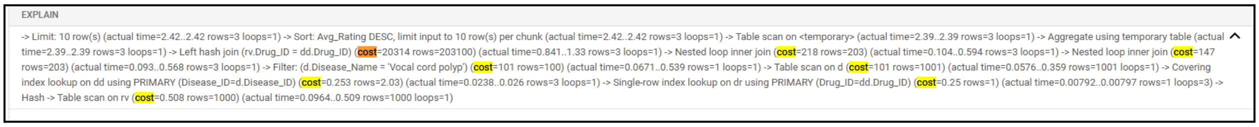
```
-> Limit: 10 row(s) (actual time=2.87..2.87 rows=3 loops=1) -> Sort: Avg_Rating DESC, limit input to 10 row(s) per chunk (actual time=2.87..2.87 rows=3 loops=1) -> Table scan on <temporary> (actual time=2.79..2.79 rows=3 loops=1) -> Aggregate using temporary table (actual time=2.79..2.79 rows=3 loops=1) -> Left hash join (rv.Drug_ID = dd.Drug_ID) (cost=20314 rows=203100) (actual time=2.21..2.73 rows=3 loops=1) -> Nested loop inner join (cost=218 rows=203) (actual time=0.187..0.707 rows=3 loops=1) -> Nested loop inner join (cost=147 rows=203) (actual time=0.153..0.629 rows=3 loops=1) -> Filter: (d.Disease_Name = 'Vocal cord polyp') (cost=101 rows=100) (actual time=0.0898..0.563 rows=1 loops=1) -> Table scan on d (cost=101 rows=1001) (actual time=0.0829..0.468 rows=1001 loops=1) -> Covering index lookup on dd using PRIMARY (Disease_ID=d.Disease_ID) (cost=0.253 rows=2.03) (actual time=0.0615..0.0633 rows=3 loops=1) -> Single-row index lookup on dr using PRIMARY (Drug_ID=dd.Drug_ID) (cost=0.25 rows=1) (actual time=0.0252..0.0253 rows=1 loops=3) -> Hash -> Table scan on rv (cost=0.508 rows=1000) (actual time=0.0874..0.625 rows=1000 loops=1)
```

DRUG_REVIEW(Drug_ID):



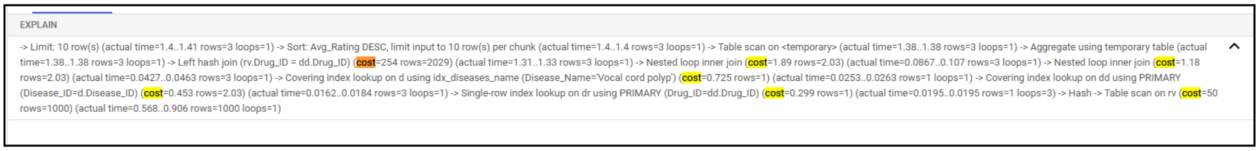
```
-> Limit: 10 row(s) (actual time=2.25..2.25 rows=3 loops=1) -> Sort: Avg_Rating DESC, limit input to 10 row(s) per chunk (actual time=2.25..2.25 rows=3 loops=1) -> Table scan on <temporary> (actual time=1.2..1.2 rows=3 loops=1) -> Aggregate using temporary table (actual
time=1.19..1.19 rows=3 loops=1) -> Nested loop left join (cost=331 rows=322) (actual time=0.328..1.11 rows=3 loops=1) -> Nested loop inner join (cost=218 rows=203) (actual time=0.284..1.06 rows=3 loops=1) -> Nested loop inner join (cost=147 rows=203) (actual
time=0.266..1.01 rows=3 loops=1) -> Filter: (d.Disease_Name = 'Vocal cord polyp') (cost=101 rows=100) (actual time=0.214..0.959 rows=1 loops=1) -> Table scan on d (cost=101 rows=1001) (actual time=0.203..0.761 rows=1001 loops=1) -> Covering index lookup on dd using
PRIMARY (Disease_ID=d.Disease_ID) (cost=0.253 rows=2.03) (actual time=0.0503..0.0522 rows=3 loops=1) -> Single-row index lookup on dr using PRIMARY (Drug_ID=dd.Drug_ID) (cost=0.25 rows=1) (actual time=0.0128..0.0128 rows=1 loops=3) -> Index lookup on rv using
idx_drug_review_drug_id (Drug_ID=dd.Drug_ID) (cost=0.398 rows=1.59) (actual time=0.0155..0.0162 rows=0.333 loops=3)
```

DRUG_DISEASE(Disease_ID, Drug_ID):



```
-> Limit: 10 row(s) (actual time=2.42..2.42 rows=3 loops=1) -> Sort: Avg_Rating DESC, limit input to 10 row(s) per chunk (actual time=2.42..2.42 rows=3 loops=1) -> Table scan on <temporary> (actual time=2.39..2.39 rows=3 loops=1) -> Aggregate using temporary table (actual
time=2.39..2.39 rows=3 loops=1) -> Left hash join (rv.Drug_ID = dd.Drug_ID) (cost=20314 rows=203100) (actual time=0.841..1.33 rows=3 loops=1) -> Nested loop inner join (cost=218 rows=203) (actual time=0.104..0.594 rows=3 loops=1) -> Nested loop inner join (cost=147
rows=203) (actual time=0.093..0.568 rows=3 loops=1) -> Filter: (d.Disease_Name = 'Vocal cord polyp') (cost=101 rows=100) (actual time=0.0671..0.539 rows=1 loops=1) -> Table scan on d (cost=101 rows=1001) (actual time=0.0576..0.359 rows=1001 loops=1) -> Covering
index lookup on dd using PRIMARY (Disease_ID=d.Disease_ID) (cost=0.253 rows=2.03) (actual time=0.0238..0.026 rows=3 loops=1) -> Single-row index lookup on dr using PRIMARY (Drug_ID=dd.Drug_ID) (cost=0.25 rows=1) (actual time=0.00792..0.00797 rows=1 loops=3) ->
Hash -> Table scan on rv (cost=0.508 rows=1000) (actual time=0.0964..0.509 rows=1000 loops=1)
```

DISEASES(Disease_Name):



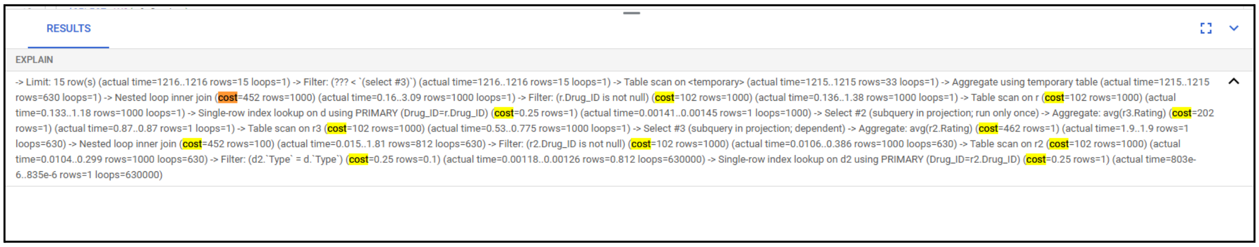
```
-> Limit: 10 row(s) (actual time=1.4..1.41 rows=3 loops=1) -> Sort: Avg_Rating DESC, limit input to 10 row(s) per chunk (actual time=1.4..1.4 rows=3 loops=1) -> Table scan on <temporary> (actual time=1.38..1.38 rows=3 loops=1) -> Aggregate using temporary table (actual
time=1.38..1.38 rows=3 loops=1) -> Left hash join (rv.Drug_ID = dd.Drug_ID) (cost=254 rows=2029) (actual time=1.31..1.33 rows=3 loops=1) -> Nested loop inner join (cost=1.89 rows=2.03) (actual time=0.0867..0.107 rows=3 loops=1) -> Nested loop inner join (cost=1.18
rows=2.03) (actual time=0.0427..0.0463 rows=3 loops=1) -> Covering index lookup on d using idx_diseases_name (Disease_Name='Vocal cord polyp') (cost=0.725 rows=1) (actual time=0.0253..0.0263 rows=1 loops=1) -> Covering index lookup on dd using PRIMARY
(Disease_ID=d.Disease_ID) (cost=0.453 rows=2.03) (actual time=0.0162..0.0184 rows=3 loops=1) -> Single-row index lookup on dr using PRIMARY (Drug_ID=dd.Drug_ID) (cost=0.299 rows=1) (actual time=0.0195..0.0195 rows=1 loops=3) -> Hash -> Table scan on rv (cost=50
rows=1000) (actual time=0.568..0.906 rows=1000 loops=1)
```

Query 1 Analysis:

For Query 1, four indexing configurations were tested using EXPLAIN ANALYZE: the default index, and custom indexes on DRUG_REVIEW(Drug_ID), DRUG_DISEASE(Disease_ID, Drug_ID), and DISEASES(Disease_Name). The default setup had the highest cost at ~21,432 due to expensive nested loop joins and table scans. Indexing DRUG_REVIEW(Drug_ID) reduced the cost to ~900, while indexing DRUG_DISEASE(Disease_ID, Drug_ID) had no impact, matching the default cost. The most effective index was DISEASES(Disease_Name), which brought the cost down to ~308. This was selected as the final index design, as it made a ~98.5% reduction in total query cost.

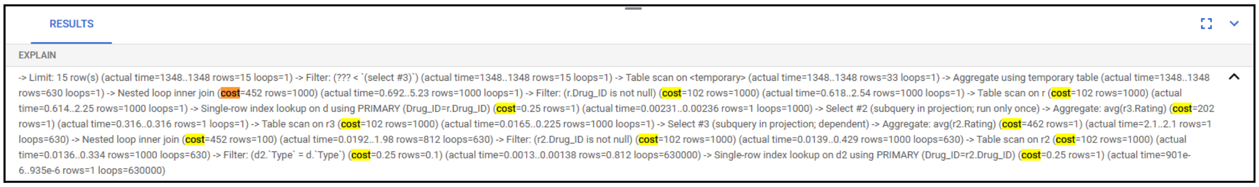
Query 2:

Default Index:



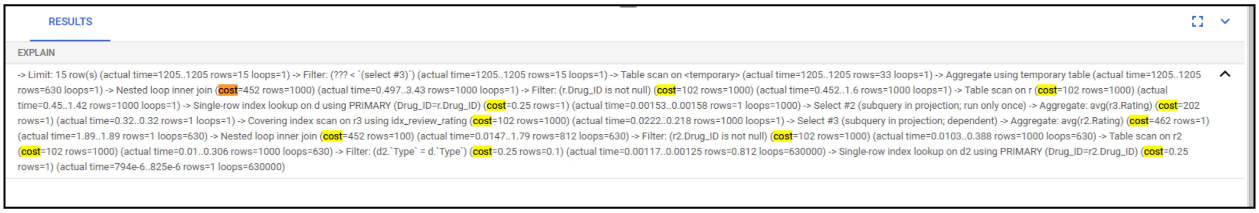
```
-> Limit: 15 row(s) (actual time=1216..1216 rows=15 loops=1) -> Filter: (??? < `(select #3)`) (actual time=1216..1216 rows=15 loops=1) -> Table scan on <temporary> (actual time=1215..1215 rows=33 loops=1) -> Aggregate using temporary table (actual time=1215..1215
rows=630 loops=1) -> Nested loop inner join (cost=452 rows=1000) (actual time=0.16..3.09 rows=1000 loops=1) -> Filter: (r.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.136..1.38 rows=1000 loops=1) -> Table scan on r (cost=102 rows=1000) (actual
time=0.133..1.18 rows=1000 loops=1) -> Single-row index lookup on d using PRIMARY (Drug_ID=r.Drug_ID) (cost=0.25 rows=1) (actual time=0.00141..0.00145 rows=1 loops=1000) -> Select #2 (subquery in projection; run only once) -> Aggregate: avg(r3.Rating) (cost=202
rows=1) (actual time=0.87..0.87 rows=1 loops=1) -> Table scan on r3 (cost=102 rows=1000) (actual time=0.53..0.775 rows=1000 loops=1) -> Select #3 (subquery in projection; dependent) -> Aggregate: avg(r2.Rating) (cost=462 rows=1) (actual time=1.9..1.9 rows=1
loops=630) -> Nested loop inner join (cost=452 rows=100) (actual time=0.015..1.81 rows=812 loops=630) -> Filter: (r2.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.0106..0.386 rows=1000 loops=630) -> Table scan on r2 (cost=102 rows=1000) (actual
time=0.0104..0.299 rows=1000 loops=630) -> Filter: (d2.`Type` = d.`Type`) (cost=0.25 rows=0.1) (actual time=0.00118..0.00126 rows=0.812 loops=630000) -> Single-row index lookup on d2 using PRIMARY (Drug_ID=r2.Drug_ID) (cost=0.25 rows=1) (actual time=803e-
6..835e-6 rows=1 loops=630000)
```

DRUG_REVIEW(Drug_ID):



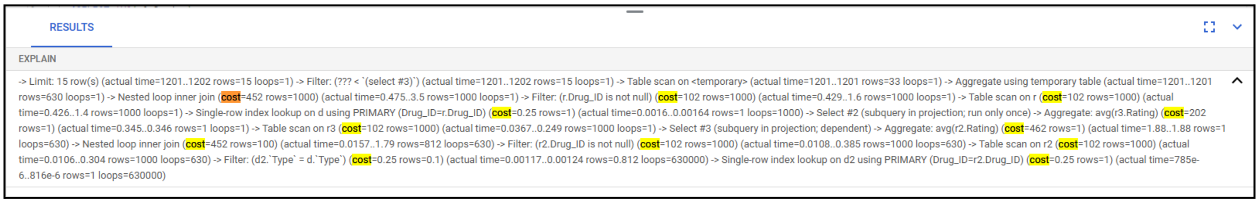
```
EXPLAIN
-> Limit: 15 row(s) (actual time=1348..1348 rows=15 loops=1) -> Filter: (??? < `(select #3)`) (actual time=1348..1348 rows=15 loops=1) -> Table scan on <temporary> (actual time=1348..1348 rows=33 loops=1) -> Aggregate using temporary table (actual time=1348..1348
rows=630 loops=1) -> Nested loop inner join (cost=452 rows=1000) (actual time=0.692..5.23 rows=1000 loops=1) -> Filter: (r.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.618..2.54 rows=1000 loops=1) -> Table scan on r (cost=102 rows=1000) (actual
time=0.614..2.25 rows=1000 loops=1) -> Single-row index lookup on d using PRIMARY (Drug_ID=r.Drug_ID) (cost=0.25 rows=1) (actual time=0.00231..0.00236 rows=1 loops=1000) -> Select #2 (subquery in projection; run only once) -> Aggregate: avg(r3.Rating) (cost=202
rows=1) (actual time=0.316..0.316 rows=1 loops=1) -> Table scan on r3 (cost=102 rows=1000) (actual time=0.0165..0.225 rows=1000 loops=1) -> Select #3 (subquery in projection; dependent) -> Aggregate: avg(r2.Rating) (cost=462 rows=1) (actual time=2.1..2.1 rows=1
loops=630) -> Nested loop inner join (cost=452 rows=100) (actual time=0.0192..1.98 rows=812 loops=630) -> Filter: (r2.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.0139..0.429 rows=1000 loops=630) -> Table scan on r2 (cost=102 rows=1000) (actual
time=0.0136..0.334 rows=1000 loops=630) -> Filter: (d2.`Type` = d.`Type`) (cost=0.25 rows=0.1) (actual time=0.0013..0.00138 rows=0.812 loops=630000) -> Single-row index lookup on d2 using PRIMARY (Drug_ID=r2.Drug_ID) (cost=0.25 rows=1) (actual time=901e-
6..935e-6 rows=1 loops=630000)
```

DRUG_REVIEW(Rating):



```
EXPLAIN
-> Limit: 15 row(s) (actual time=1205..1205 rows=15 loops=1) -> Filter: (??? < `(select #3)`) (actual time=1205..1205 rows=15 loops=1) -> Table scan on <temporary> (actual time=1205..1205 rows=33 loops=1) -> Aggregate using temporary table (actual time=1205..1205
rows=630 loops=1) -> Nested loop inner join (cost=452 rows=1000) (actual time=0.497..3.43 rows=1000 loops=1) -> Filter: (r.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.452..1.6 rows=1000 loops=1) -> Table scan on r (cost=102 rows=1000) (actual
time=0.45..1.42 rows=1000 loops=1) -> Single-row index lookup on d using PRIMARY (Drug_ID=r.Drug_ID) (cost=0.25 rows=1) (actual time=0.00153..0.00158 rows=1 loops=1000) -> Select #2 (subquery in projection; run only once) -> Aggregate: avg(r3.Rating) (cost=202
rows=1) (actual time=0.32..0.32 rows=1 loops=1) -> Covering index scan on r3 using idx_review_rating (cost=102 rows=1000) (actual time=0.0222..0.218 rows=1000 loops=1) -> Select #3 (subquery in projection; dependent) -> Aggregate: avg(r2.Rating) (cost=462 rows=1)
(actual time=1.89..1.89 rows=1 loops=630) -> Nested loop inner join (cost=452 rows=100) (actual time=0.0147..1.79 rows=812 loops=630) -> Filter: (r2.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.0103..0.388 rows=1000 loops=630) -> Table scan on r2
(cost=102 rows=1000) (actual time=0.01..0.306 rows=1000 loops=630) -> Filter: (d2.`Type` = d.`Type`) (cost=0.25 rows=0.1) (actual time=0.00117..0.00125 rows=0.812 loops=630000) -> Single-row index lookup on d2 using PRIMARY (Drug_ID=r2.Drug_ID) (cost=0.25
rows=1) (actual time=794e-6..825e-6 rows=1 loops=630000)
```

DRUGS(Type):



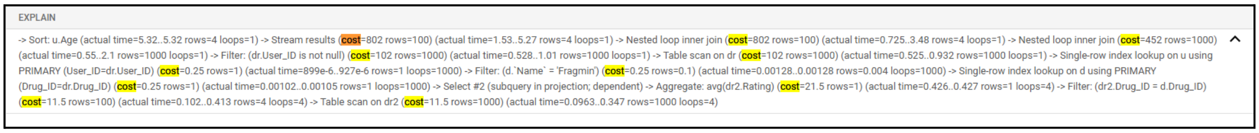
```
EXPLAIN
-> Limit: 15 row(s) (actual time=1201..1202 rows=15 loops=1) -> Filter: (??? < `(select #3)`) (actual time=1201..1202 rows=15 loops=1) -> Table scan on <temporary> (actual time=1201..1201 rows=33 loops=1) -> Aggregate using temporary table (actual time=1201..1201
rows=630 loops=1) -> Nested loop inner join (cost=452 rows=1000) (actual time=0.475..3.5 rows=1000 loops=1) -> Filter: (r.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.429..1.6 rows=1000 loops=1) -> Table scan on r (cost=102 rows=1000) (actual
time=0.426..1.4 rows=1000 loops=1) -> Single-row index lookup on d using PRIMARY (Drug_ID=r.Drug_ID) (cost=0.25 rows=1) (actual time=0.0016..0.00164 rows=1 loops=1000) -> Select #2 (subquery in projection; run only once) -> Aggregate: avg(r3.Rating) (cost=202
rows=1) (actual time=0.345..0.346 rows=1 loops=1) -> Table scan on r3 (cost=102 rows=1000) (actual time=0.0367..0.249 rows=1000 loops=1) -> Select #3 (subquery in projection; dependent) -> Aggregate: avg(r2.Rating) (cost=462 rows=1) (actual time=1.88..1.88 rows=1
loops=630) -> Nested loop inner join (cost=452 rows=100) (actual time=0.0157..1.79 rows=812 loops=630) -> Filter: (r2.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.0108..0.385 rows=1000 loops=630) -> Table scan on r2 (cost=102 rows=1000) (actual
time=0.0106..0.304 rows=1000 loops=630) -> Filter: (d2.`Type` = d.`Type`) (cost=0.25 rows=0.1) (actual time=0.00117..0.00124 rows=0.812 loops=630000) -> Single-row index lookup on d2 using PRIMARY (Drug_ID=r2.Drug_ID) (cost=0.25 rows=1) (actual time=785e-
6..816e-6 rows=1 loops=630000)
```

Query 2 Analysis:

For Query 2, four indexing setups were tested using EXPLAIN ANALYZE: the default index, and custom indexes on DRUG_REVIEW(Drug_ID), DRUG_REVIEW(Rating), and DRUGS(Type). All four had the same total cost of ~2079, with no noticeable improvement or slowdown/speedup in performance. This suggests that the indexes didn’t help because the query plan already used efficient lookups, and the indexed columns weren’t used in a way that made a big difference. Since the custom indexes had no effect, the default index was kept as the final choice for Query 2.

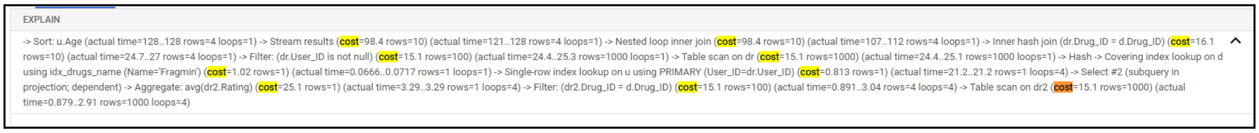
Query 3:

Default Index:



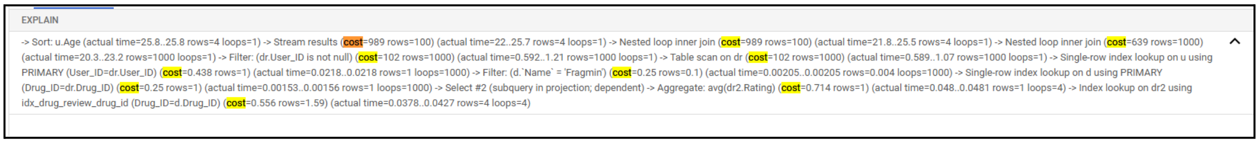
```
EXPLAIN
-> Sort: u.Age (actual time=5.32..5.32 rows=4 loops=1) -> Stream results (cost=802 rows=100) (actual time=1.53..5.27 rows=4 loops=1) -> Nested loop inner join (cost=802 rows=100) (actual time=0.725..3.48 rows=4 loops=1) -> Nested loop inner join (cost=452 rows=1000)
(actual time=0.55..2.1 rows=1000 loops=1) -> Filter: (dr.User_ID is not null) (cost=102 rows=1000) (actual time=0.528..1.01 rows=1000 loops=1) -> Table scan on dr (cost=102 rows=1000) (actual time=0.525..0.932 rows=1000 loops=1) -> Single-row index lookup on u using
PRIMARY (User_ID=dr.User_ID) (cost=0.25 rows=1) (actual time=899e-6..927e-6 rows=1 loops=1000) -> Filter: (d.`Name` = 'Fragmin') (cost=0.25 rows=0.1) (actual time=0.00128..0.00128 rows=0.004 loops=1000) -> Single-row index lookup on d using PRIMARY
(Drug_ID=dr.Drug_ID) (cost=0.25 rows=1) (actual time=0.00102..0.00105 rows=1 loops=1000) -> Select #2 (subquery in projection; dependent) -> Aggregate: avg(dr2.Rating) (cost=21.5 rows=1) (actual time=0.426..0.427 rows=1 loops=4) -> Filter: (dr2.Drug_ID = d.Drug_ID)
(cost=11.5 rows=100) (actual time=0.102..0.413 rows=4 loops=4) -> Table scan on dr2 (cost=11.5 rows=1000) (actual time=0.0963..0.347 rows=1000 loops=4)
```

DRUGS(Name):



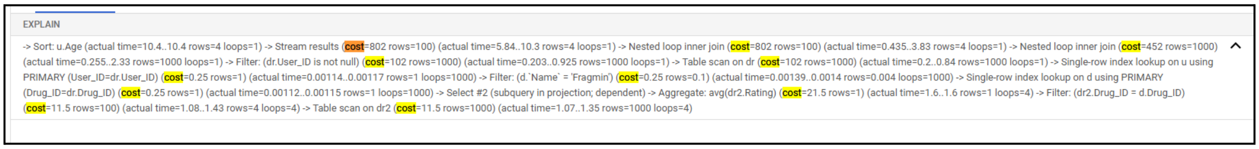
```
EXPLAIN
-> Sort: u.Age (actual time=128..128 rows=4 loops=1) -> Stream results (cost=98.4 rows=10) (actual time=121..128 rows=4 loops=1) -> Nested loop inner join (cost=98.4 rows=10) (actual time=107..112 rows=4 loops=1) -> Inner hash join (dr.Drug_ID = d.Drug_ID) (cost=16.1
rows=10) (actual time=24.7..27 rows=4 loops=1) -> Filter: (dr.User_ID is not null) (cost=15.1 rows=100) (actual time=24.4..25.3 rows=1000 loops=1) -> Table scan on dr (cost=15.1 rows=1000) (actual time=24.4..25.1 rows=1000 loops=1) -> Hash -> Covering index lookup on d
using idx_drugs_name (Name='Fragmin') (cost=1.02 rows=1) (actual time=0.0666..0.0717 rows=1 loops=1) -> Single-row index lookup on u using PRIMARY (User_ID=dr.User_ID) (cost=0.813 rows=1) (actual time=21.2..21.2 rows=1 loops=4) -> Select #2 (subquery in
projection; dependent) -> Aggregate: avg(dr2.Rating) (cost=25.1 rows=1) (actual time=3.29..3.29 rows=1 loops=4) -> Filter: (dr2.Drug_ID = d.Drug_ID) (cost=15.1 rows=100) (actual time=0.891..3.04 rows=4 loops=4) -> Table scan on dr2 (cost=15.1 rows=1000) (actual
time=0.879..2.91 rows=1000 loops=4)
```

DRUG_REVIEW(Drug_ID):



```
EXPLAIN
-> Sort: u.Age (actual time=25.8..25.8 rows=4 loops=1) -> Stream results (cost=989 rows=100) (actual time=22..25.7 rows=4 loops=1) -> Nested loop inner join (cost=989 rows=100) (actual time=21.8..25.5 rows=4 loops=1) -> Nested loop inner join (cost=639 rows=1000)
(actual time=20.3..23.2 rows=1000 loops=1) -> Filter: (dr.User_ID is not null) (cost=102 rows=1000) (actual time=0.592..1.21 rows=1000 loops=1) -> Table scan on dr (cost=102 rows=1000) (actual time=0.589..1.07 rows=1000 loops=1) -> Single-row index lookup on u using
PRIMARY (User_ID=dr.User_ID) (cost=0.438 rows=1) (actual time=0.0218..0.0218 rows=1 loops=1000) -> Filter: (d.`Name` = 'Fragmin') (cost=0.25 rows=0.1) (actual time=0.00205..0.00205 rows=0.004 loops=1000) -> Single-row index lookup on d using PRIMARY
(Drug_ID=dr.Drug_ID) (cost=0.25 rows=1) (actual time=0.00153..0.00156 rows=1 loops=1000) -> Select #2 (subquery in projection; dependent) -> Aggregate: avg(dr2.Rating) (cost=0.714 rows=1) (actual time=0.048..0.0481 rows=1 loops=4) -> Index lookup on dr2 using
idx_drug_review_drug_id (Drug_ID=d.Drug_ID) (cost=0.556 rows=1.59) (actual time=0.0378..0.0427 rows=4 loops=4)
```

USERS(User_ID, Age):



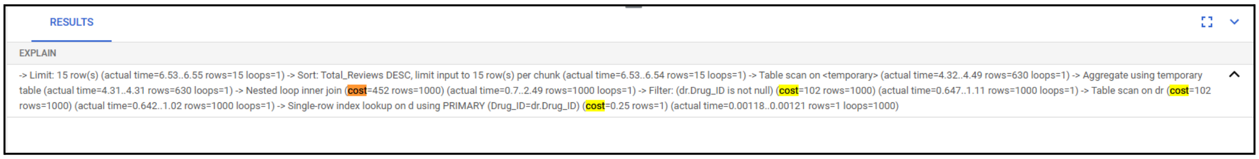
```
EXPLAIN
-> Sort: u.Age (actual time=10.4..10.4 rows=4 loops=1) -> Stream results (cost=802 rows=100) (actual time=5.84..10.3 rows=4 loops=1) -> Nested loop inner join (cost=802 rows=100) (actual time=0.435..3.83 rows=4 loops=1) -> Nested loop inner join (cost=452 rows=1000)
(actual time=0.255..2.33 rows=1000 loops=1) -> Filter: (dr.User_ID is not null) (cost=102 rows=1000) (actual time=0.203..0.925 rows=1000 loops=1) -> Table scan on dr (cost=102 rows=1000) (actual time=0.2..0.84 rows=1000 loops=1) -> Single-row index lookup on u using
PRIMARY (User_ID=dr.User_ID) (cost=0.25 rows=1) (actual time=0.00114..0.00117 rows=1 loops=1000) -> Filter: (d.`Name` = 'Fragmin') (cost=0.25 rows=0.1) (actual time=0.00139..0.0014 rows=0.004 loops=1000) -> Single-row index lookup on d using PRIMARY
(Drug_ID=dr.Drug_ID) (cost=0.25 rows=1) (actual time=0.00112..0.00115 rows=1 loops=1000) -> Select #2 (subquery in projection; dependent) -> Aggregate: avg(dr2.Rating) (cost=21.5 rows=1) (actual time=1.6..1.6 rows=1 loops=4) -> Filter: (dr2.Drug_ID = d.Drug_ID)
(cost=11.5 rows=100) (actual time=1.08..1.43 rows=4 loops=4) -> Table scan on dr2 (cost=11.5 rows=1000) (actual time=1.07..1.35 rows=1000 loops=4)
```

Query 3 Analysis:

For Query 3, four indexing setups were tested using EXPLAIN ANALYZE: the default index, and custom indexes on DRUGS(Name), DRUG_REVIEW(Drug_ID), and USERS(User_ID, Age). The default setup had a total cost of ~2305, mostly due to costly nested loop joins and table scans. Indexing USERS(User_ID, Age) made no difference, with the same cost as the default. Indexing DRUG_REVIEW(Drug_ID) actually increased the cost to ~2823, likely because of an inefficient change in the query plan. The best result came from indexing DRUGS(Name), which brought the cost down to ~300 by speeding up filtering on drug names. This was chosen as the final index design, with an ~87% drop in total query cost.

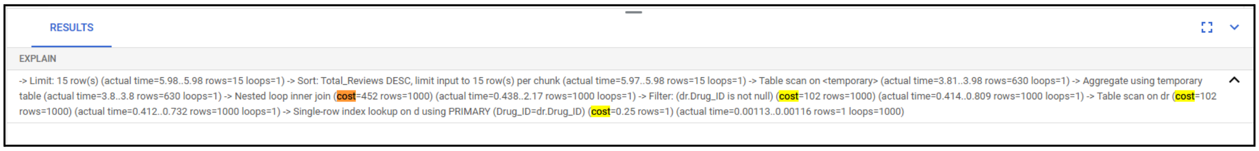
Query 4:

Default Index:



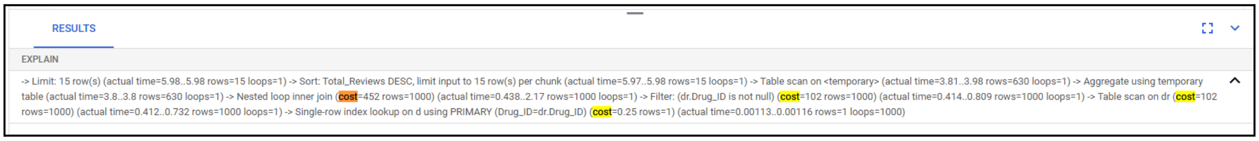
```
RESULTS
EXPLAIN
-> Limit: 15 row(s) (actual time=6.53..6.55 rows=15 loops=1) -> Sort: Total_Reviews DESC, limit input to 15 row(s) per chunk (actual time=6.53..6.54 rows=15 loops=1) -> Table scan on <temporary> (actual time=4.32..4.49 rows=630 loops=1) -> Aggregate using temporary
table (actual time=4.31..4.31 rows=630 loops=1) -> Nested loop inner join (cost=452 rows=1000) (actual time=0.7..2.49 rows=1000 loops=1) -> Filter: (dr.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.647..1.11 rows=1000 loops=1) -> Table scan on dr (cost=102
rows=1000) (actual time=0.642..1.02 rows=1000 loops=1) -> Single-row index lookup on d using PRIMARY (Drug_ID=dr.Drug_ID) (cost=0.25 rows=1) (actual time=0.00118..0.00121 rows=1 loops=1000)
```

DRUG_REVIEW(Drug_ID):



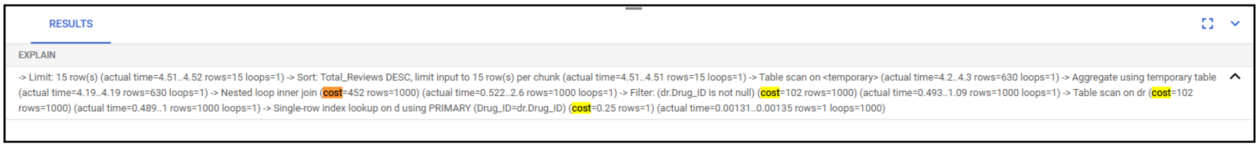
```
RESULTS
EXPLAIN
-> Limit: 15 row(s) (actual time=5.98..5.98 rows=15 loops=1) -> Sort: Total_Reviews DESC, limit input to 15 row(s) per chunk (actual time=5.97..5.98 rows=15 loops=1) -> Table scan on <temporary> (actual time=3.81..3.98 rows=630 loops=1) -> Aggregate using temporary
table (actual time=3.8..3.8 rows=630 loops=1) -> Nested loop inner join (cost=452 rows=1000) (actual time=0.438..2.17 rows=1000 loops=1) -> Filter: (dr.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.414..0.809 rows=1000 loops=1) -> Table scan on dr (cost=102
rows=1000) (actual time=0.412..0.732 rows=1000 loops=1) -> Single-row index lookup on d using PRIMARY (Drug_ID=dr.Drug_ID) (cost=0.25 rows=1) (actual time=0.00113..0.00116 rows=1 loops=1000)
```

Drugs(Drug_ID):



```
RESULTS
EXPLAIN
-> Limit: 15 row(s) (actual time=5.98..5.98 rows=15 loops=1) -> Sort: Total_Reviews DESC, limit input to 15 row(s) per chunk (actual time=5.97..5.98 rows=15 loops=1) -> Table scan on <temporary> (actual time=3.81..3.98 rows=630 loops=1) -> Aggregate using temporary
table (actual time=3.8..3.8 rows=630 loops=1) -> Nested loop inner join (cost=452 rows=1000) (actual time=0.438..2.17 rows=1000 loops=1) -> Filter: (dr.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.414..0.809 rows=1000 loops=1) -> Table scan on dr (cost=102
rows=1000) (actual time=0.412..0.732 rows=1000 loops=1) -> Single-row index lookup on d using PRIMARY (Drug_ID=dr.Drug_ID) (cost=0.25 rows=1) (actual time=0.00113..0.00116 rows=1 loops=1000)
```

DRUGS(Name, Type):



EXPLAIN

```
-> Limit: 15 row(s) (actual time=4.51..4.52 rows=15 loops=1) -> Sort: Total_Reviews DESC, limit input to 15 row(s) per chunk (actual time=4.51..4.51 rows=15 loops=1) -> Table scan on <temporary> (actual time=4.2..4.3 rows=630 loops=1) -> Aggregate using temporary table
(actual time=4.19..4.19 rows=630 loops=1) -> Nested loop inner join (cost=452 rows=1000) (actual time=0.522..2.6 rows=1000 loops=1) -> Filter: (dr.Drug_ID is not null) (cost=102 rows=1000) (actual time=0.493..1.09 rows=1000 loops=1) -> Table scan on dr (cost=102
rows=1000) (actual time=0.489..1 rows=1000 loops=1) -> Single-row index lookup on d using PRIMARY (Drug_ID=dr.Drug_ID) (cost=0.25 rows=1) (actual time=0.00131..0.00135 rows=1 loops=1000)
```

Query 4 Analysis:

For Query 4, four indexing setups were tested using EXPLAIN ANALYZE: the default index, DRUG_REVIEW(Drug_ID), DRUGS(Drug_ID), and DRUGS(Name, Type). All configurations resulted in the same total cost of ~656, not showing any improvement or decline in performance. This suggests that the indexes had no effect because the query plan was already optimized, likely using primary key lookups and simple joins. Since none of the custom indexes made a difference, we kept the default index as the final design for Query 4.