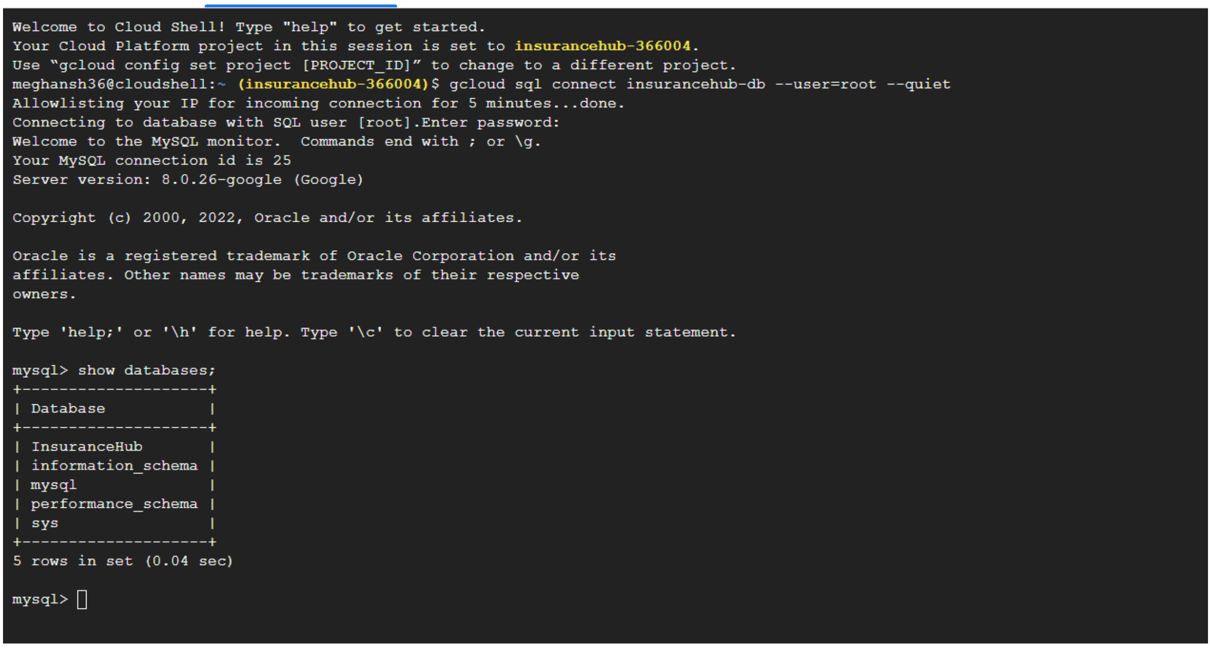
DATABASE IMPLEMENTATION

Database creation



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to insurancehub-366004.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
meghansh36@cloudshell:~ (insurancehub-366004)$ gcloud sql connect insurancehub-db --user=root --quiet
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 25
Server version: 8.0.26-google (Google)

Copyright (c) 2000, 2022, 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.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| InsuranceHub       |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.04 sec)

mysql> 
```

Data Definition Language (DDL) commands-

1. CREATE TABLE SecurityQuestion (
   id INT PRIMARY KEY AUTO_INCREMENT,
   question VARCHAR(1000) NOT NULL
   );

2. CREATE TABLE Role (
   id INT PRIMARY KEY AUTO_INCREMENT,
   role VARCHAR(255) NOT NULL
   );

3. CREATE TABLE User (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(255) NOT NULL,
   email VARCHAR(255) NOT NULL UNIQUE,
   password VARCHAR(1000) NOT NULL,
   security_question INT not null,
   security_answer VARCHAR(255) NOT NULL,
   first_name VARCHAR(255) NOT NULL,
   middle_name VARCHAR(255),
   last_name VARCHAR(255) NOT NULL,
   city VARCHAR(255) NOT NULL,
   state VARCHAR(255) NOT NULL,
   country VARCHAR(255) NOT NULL,
   zip VARCHAR(10) NOT NULL,

```
marital_status INT NOT NULL,
user_type INT,
FOREIGN KEY(security_question) REFERENCES SecurityQuestion(id),
FOREIGN KEY(user_type) REFERENCES Role(id)
);
```

4.
   ```
   CREATE TABLE Company(
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(255) NOT NULL,
   email VARCHAR(255) NOT NULL UNIQUE,
   address VARCHAR(500) NOT NULL,
   image BLOB,
   phone INT NOT NULL,
   website VARCHAR(1000) NOT NULL,
   description VARCHAR(5000)
   );
   ```

5.
   ```
   CREATE TABLE PolicyType (
   id INT PRIMARY KEY AUTO_INCREMENT,
   type VARCHAR(100) NOT NULL
   );
   ```

6.
   ```
   CREATE TABLE InsurancePolicy(
   id INT PRIMARY KEY AUTO_INCREMENT,
   type INT,
   name VARCHAR(255) NOT NULL,
   cover_amount INT,
   premium_per_month INT,
   premium_per_annum INT,
   creation_date DATE NOT NULL,
   company_id INT,
   FOREIGN KEY (company_id) REFERENCES Company(id) ON DELETE CASCADE,
   FOREIGN KEY (type) REFERENCES PolicyType(id) ON DELETE CASCADE
   );
   ```

7.
   ```
   CREATE TABLE Tag (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(255) NOT NULL,
   description VARCHAR(1000)
   );
   ```

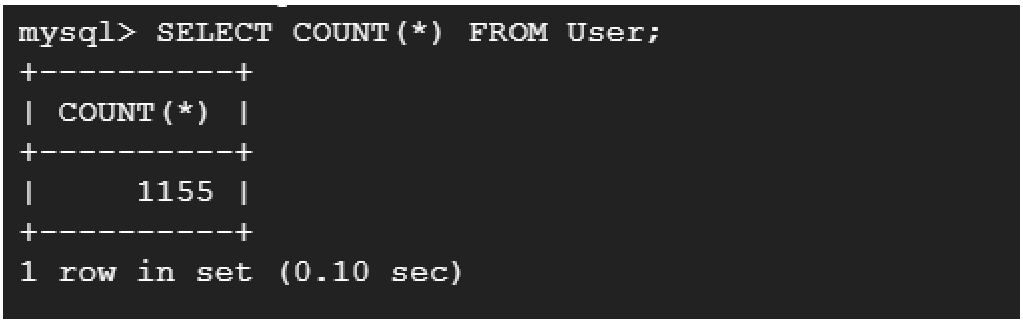
8.
   ```
   CREATE TABLE PolicyTag (
   policy_id INT,
   tag_id INT,
   PRIMARY KEY (policy_id, tag_id),
   FOREIGN KEY (policy_id) REFERENCES InsurancePolicy(id) ON DELETE
   CASCADE,
   FOREIGN KEY (tag_id)  REFERENCES Tag(id) ON DELETE CASCADE
   );
   ```

9. CREATE TABLE Rating (
   policy_id INT,
   user_id INT,
   creation_date DATE NOT NULL,
   rating INT NOT NULL,
   PRIMARY KEY (policy_id, user_id),
   FOREIGN KEY (policy_id) REFERENCES InsurancePolicy(id) ON DELETE CASCADE,
   FOREIGN KEY (user_id) REFERENCES User(id) ON DELETE CASCADE
   );

10. CREATE TABLE UserActivity (
    id INT PRIMARY KEY AUTO_INCREMENT,
    policy_id INT,
    user_id INT,
    creation_date TIMESTAMP NOT NULL,
    search_string VARCHAR(1000) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES User(id) ON DELETE CASCADE,
    FOREIGN KEY (policy_id) REFERENCES InsurancePolicy(id) ON DELETE CASCADE
    );

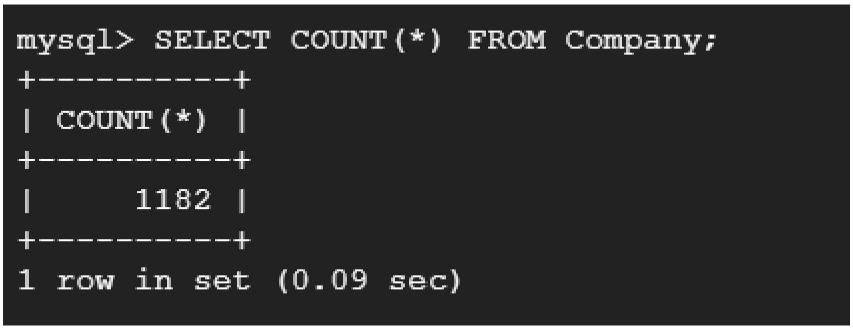
Database Dummy data:

- User Table



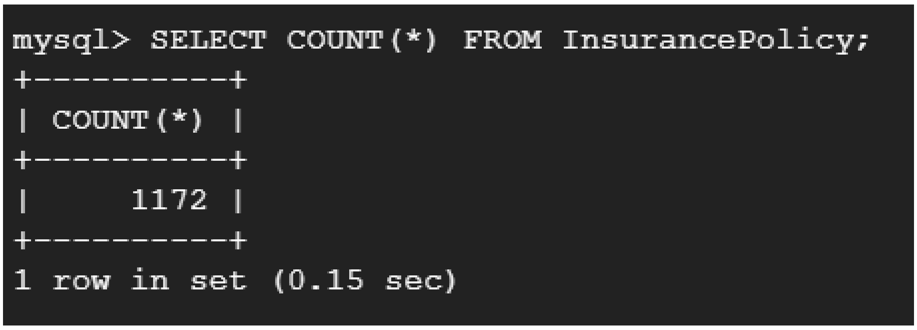
```
mysql> SELECT COUNT(*) FROM User;
+----------+
| COUNT(*) |
+----------+
|     1155 |
+----------+
1 row in set (0.10 sec)
```

- Company Table



```
mysql> SELECT COUNT(*) FROM Company;
+----------+
| COUNT(*) |
+----------+
|     1182 |
+----------+
1 row in set (0.09 sec)
```

- Insurance Policy Table

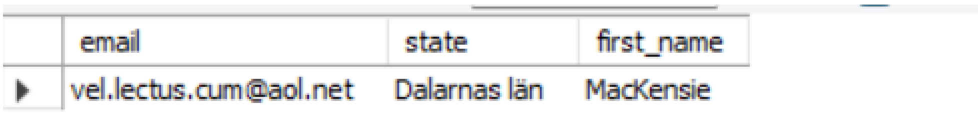


```
mysql> SELECT COUNT(*) FROM InsurancePolicy;
+----------+
| COUNT(*) |
+----------+
|     1172 |
+----------+
1 row in set (0.15 sec)
```

ADVANCED SQL QUERIES IMPLEMENTATION

- Find users who have searched for a policy having the max cover amount.

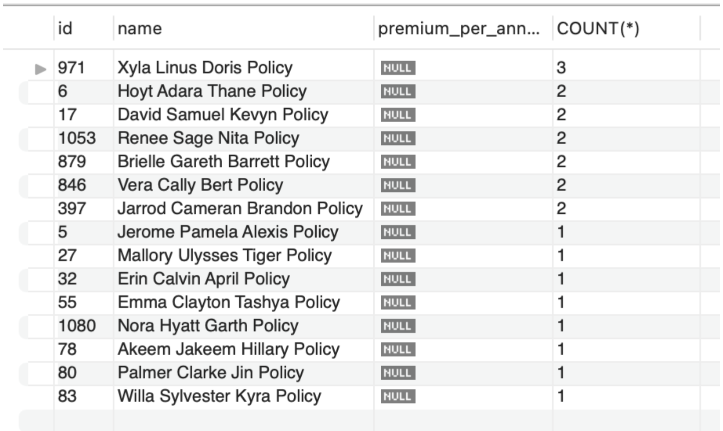
  SELECT u.email, u.state, u.first_name FROM InsuranceHub.UserActivity ua JOIN InsuranceHub.User u ON (ua.user_id = u.id) JOIN (SELECT id FROM InsuranceHub.InsurancePolicy WHERE cover_amount = (SELECT max(cover_amount) FROM InsuranceHub.InsurancePolicy)) AS temp ON (ua.policy_id = temp.id);



	email	state	first_name
▶	vel.lectus.cum@aol.net	Dalarnas län	MacKensie

- Find the top fifteen policies which have the maximum ratings most number of times.

  SELECT ip.id, ip.name, ip.premium_per_annum, COUNT(*) FROM Rating ra JOIN InsurancePolicy ip ON ra.policy_id = ip.id
  WHERE ra.rating = (Select MAX(rating) FROM Rating) GROUP BY ra.policy_id
  ORDER BY COUNT(*) DESC LIMIT 15

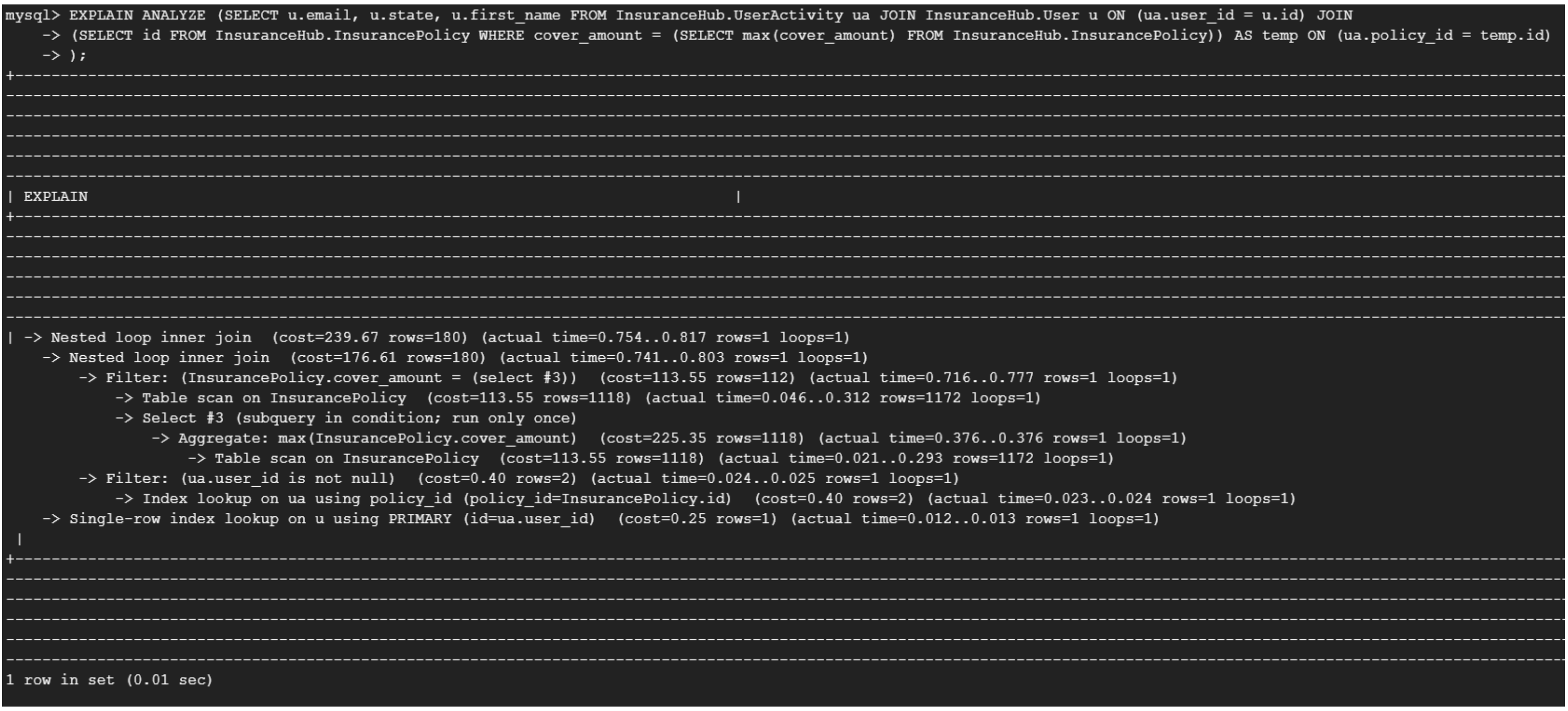


	id	name	premium_per_ann...	COUNT(*)
▶	971	Xyla Linus Doris Policy	NULL	3
	6	Hoyt Adara Thane Policy	NULL	2
	17	David Samuel Kevyn Policy	NULL	2
	1053	Renee Sage Nita Policy	NULL	2
	879	Brielle Gareth Barrett Policy	NULL	2
	846	Vera Cally Bert Policy	NULL	2
	397	Jarrod Cameran Brandon Policy	NULL	2
	5	Jerome Pamela Alexis Policy	NULL	1
	27	Mallory Ulysses Tiger Policy	NULL	1
	32	Erin Calvin April Policy	NULL	1
	55	Emma Clayton Tashya Policy	NULL	1
	1080	Nora Hyatt Garth Policy	NULL	1
	78	Akeem Jakeem Hillary Policy	NULL	1
	80	Palmer Clarke Jin Policy	NULL	1
	83	Willa Sylvester Kyra Policy	NULL	1

Indexing:

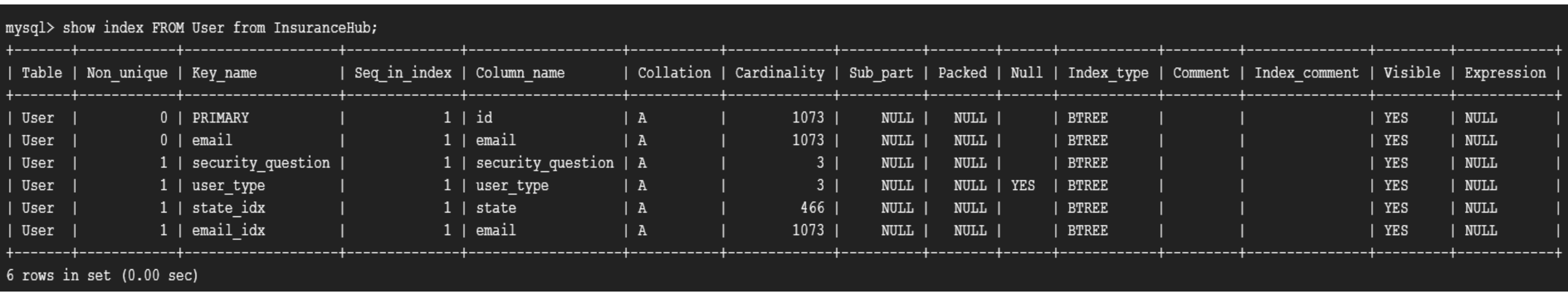
1. Adding indexes to the first query

The below image depicts the performance of the query before creating the index on cover_amount, first name, email, and state. Note that this performance is with indexes already added for PKs and FKs, which were added automatically by MySQL.



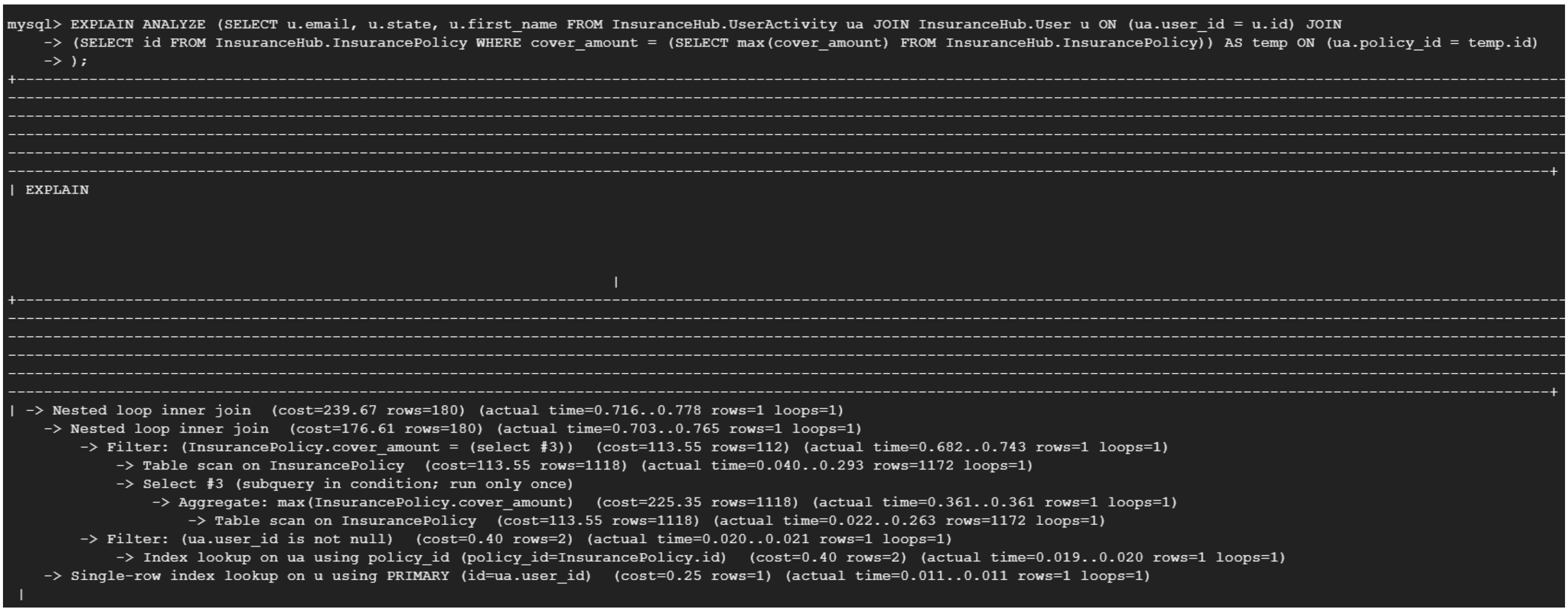
```
mysql> EXPLAIN ANALYZE (SELECT u.email, u.state, u.first_name FROM InsuranceHub.UserActivity ua JOIN InsuranceHub.User u ON (ua.user_id = u.id) JOIN
    -> (SELECT id FROM InsuranceHub.InsurancePolicy WHERE cover_amount = (SELECT max(cover_amount) FROM InsuranceHub.InsurancePolicy)) AS temp ON (ua.policy_id = temp.id)
    -> );
+------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN                                                                        |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=239.67 rows=180) (actual time=0.754..0.817 rows=1 loops=1)
    -> Nested loop inner join  (cost=176.61 rows=180) (actual time=0.741..0.803 rows=1 loops=1)
        -> Filter: (InsurancePolicy.cover_amount = (select #3))  (cost=113.55 rows=112) (actual time=0.716..0.777 rows=1 loops=1)
            -> Table scan on InsurancePolicy  (cost=113.55 rows=1118) (actual time=0.046..0.312 rows=1172 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: max(InsurancePolicy.cover_amount)  (cost=225.35 rows=1118) (actual time=0.376..0.376 rows=1 loops=1)
                    -> Table scan on InsurancePolicy  (cost=113.55 rows=1118) (actual time=0.021..0.293 rows=1172 loops=1)
        -> Filter: (ua.user_id is not null)  (cost=0.40 rows=2) (actual time=0.024..0.025 rows=1 loops=1)
            -> Index lookup on ua using policy_id (policy_id=InsurancePolicy.id)  (cost=0.40 rows=2) (actual time=0.023..0.024 rows=1 loops=1)
    -> Single-row index lookup on u using PRIMARY (id=ua.user_id)  (cost=0.25 rows=1) (actual time=0.012..0.013 rows=1 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

First, we have added an index on state and email. However, we didn't see any performance updates as email and state are used only in select, so they don't provide a performance boost.



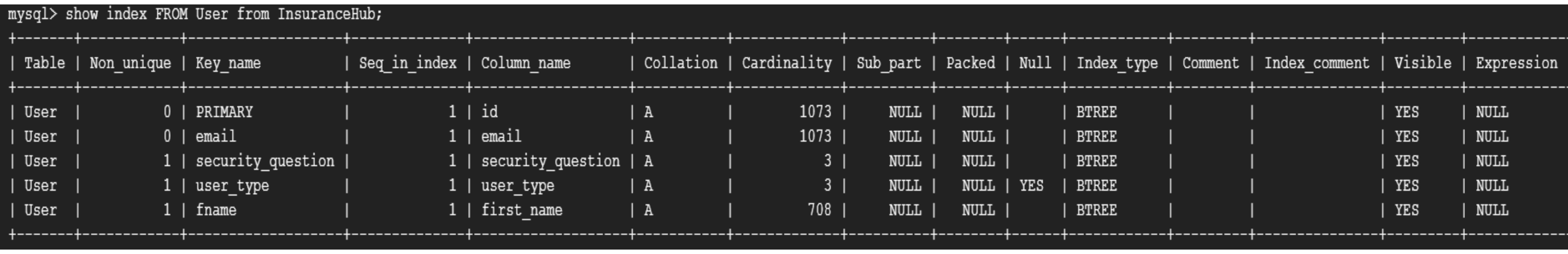
```
mysql> show index FROM User from InsuranceHub;
+-------+------------+-------------------+--------------+-------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table | Non_unique | Key_name          | Seq_in_index | Column_name       | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-------+------------+-------------------+--------------+-------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| User  |          0 | PRIMARY           |            1 | id                | A         |        1073 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| User  |          0 | email             |            1 | email             | A         |        1073 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| User  |          1 | security_question |            1 | security_question | A         |           3 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| User  |          1 | user_type         |            1 | user_type         | A         |           3 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL       |
| User  |          1 | state_idx         |            1 | state             | A         |         466 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| User  |          1 | email_idx         |            1 | email             | A         |        1073 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+-------+------------+-------------------+--------------+-------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
6 rows in set (0.00 sec)
```

This is the performance after indexes on **state** and **email**.



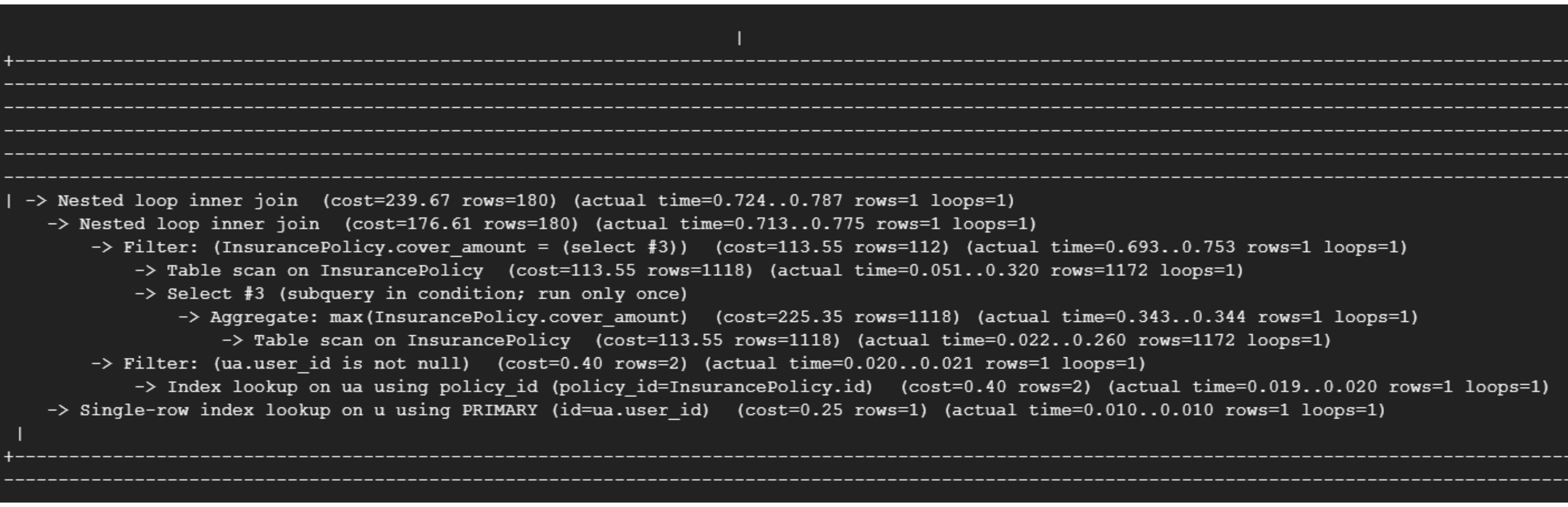
```
mysql> EXPLAIN ANALYZE (SELECT u.email, u.state, u.first_name FROM InsuranceHub.UserActivity ua JOIN InsuranceHub.User u ON (ua.user_id = u.id) JOIN
    -> (SELECT id FROM InsuranceHub.InsurancePolicy WHERE cover_amount = (SELECT max(cover_amount) FROM InsuranceHub.InsurancePolicy)) AS temp ON (ua.policy_id = temp.id)
    -> );
+-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN




                                                    |
+-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=239.67 rows=180) (actual time=0.716..0.778 rows=1 loops=1)
    -> Nested loop inner join  (cost=176.61 rows=180) (actual time=0.703..0.765 rows=1 loops=1)
        -> Filter: (InsurancePolicy.cover_amount = (select #3))  (cost=113.55 rows=112) (actual time=0.682..0.743 rows=1 loops=1)
            -> Table scan on InsurancePolicy  (cost=113.55 rows=1118) (actual time=0.040..0.293 rows=1172 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: max(InsurancePolicy.cover_amount)  (cost=225.35 rows=1118) (actual time=0.361..0.361 rows=1 loops=1)
                    -> Table scan on InsurancePolicy  (cost=113.55 rows=1118) (actual time=0.022..0.263 rows=1172 loops=1)
        -> Filter: (ua.user_id is not null)  (cost=0.40 rows=2) (actual time=0.020..0.021 rows=1 loops=1)
            -> Index lookup on ua using policy_id (policy_id=InsurancePolicy.id)  (cost=0.40 rows=2) (actual time=0.019..0.020 rows=1 loops=1)
    -> Single-row index lookup on u using PRIMARY (id=ua.user_id)  (cost=0.25 rows=1) (actual time=0.011..0.011 rows=1 loops=1)
 |
```

Next, we try adding an index on the **first name** attribute. However, this also didn't provide a performance boost as the first name is used only in select.



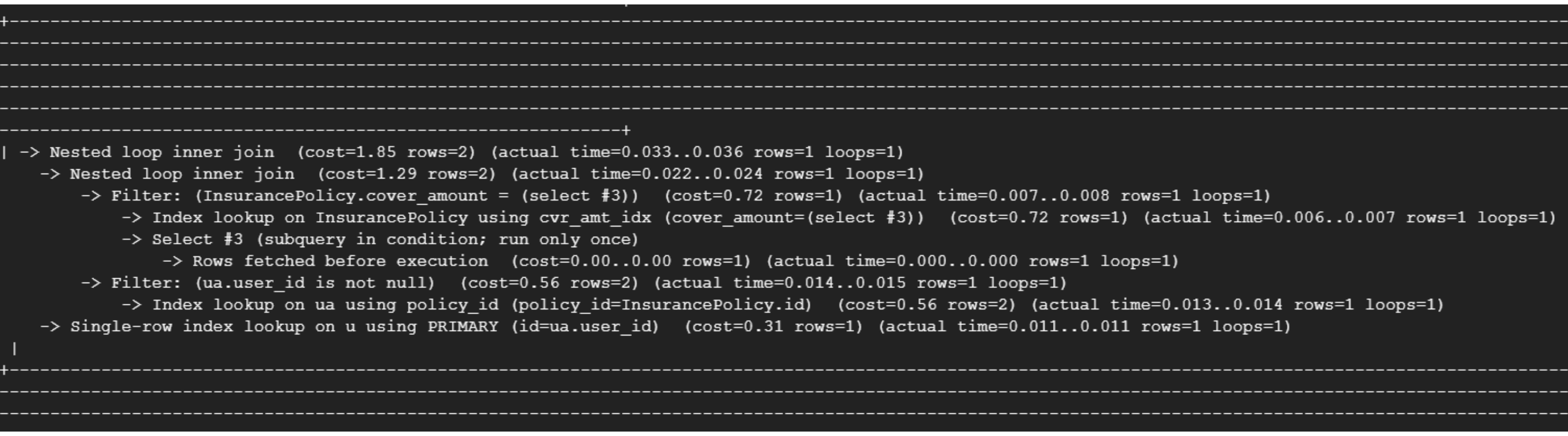
```
mysql> show index FROM User from InsuranceHub;
+-------+------------+-------------------+--------------+-------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------
| Table | Non_unique | Key_name          | Seq_in_index | Column_name       | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression
+-------+------------+-------------------+--------------+-------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------
| User  |          0 | PRIMARY           |            1 | id                | A         |        1073 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL
| User  |          0 | email             |            1 | email             | A         |        1073 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL
| User  |          1 | security_question |            1 | security_question | A         |           3 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL
| User  |          1 | user_type         |            1 | user_type         | A         |           3 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL
| User  |          1 | fname             |            1 | first_name        | A         |         708 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL
+-------+------------+-------------------+--------------+-------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------
```

Below is the performance after adding the index on the **first name**.



```
                                                    |
+-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=239.67 rows=180) (actual time=0.724..0.787 rows=1 loops=1)
    -> Nested loop inner join  (cost=176.61 rows=180) (actual time=0.713..0.775 rows=1 loops=1)
        -> Filter: (InsurancePolicy.cover_amount = (select #3))  (cost=113.55 rows=112) (actual time=0.693..0.753 rows=1 loops=1)
            -> Table scan on InsurancePolicy  (cost=113.55 rows=1118) (actual time=0.051..0.320 rows=1172 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: max(InsurancePolicy.cover_amount)  (cost=225.35 rows=1118) (actual time=0.343..0.344 rows=1 loops=1)
                    -> Table scan on InsurancePolicy  (cost=113.55 rows=1118) (actual time=0.022..0.260 rows=1172 loops=1)
        -> Filter: (ua.user_id is not null)  (cost=0.40 rows=2) (actual time=0.020..0.021 rows=1 loops=1)
            -> Index lookup on ua using policy_id (policy_id=InsurancePolicy.id)  (cost=0.40 rows=2) (actual time=0.019..0.020 rows=1 loops=1)
    -> Single-row index lookup on u using PRIMARY (id=ua.user_id)  (cost=0.25 rows=1) (actual time=0.010..0.010 rows=1 loops=1)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
```

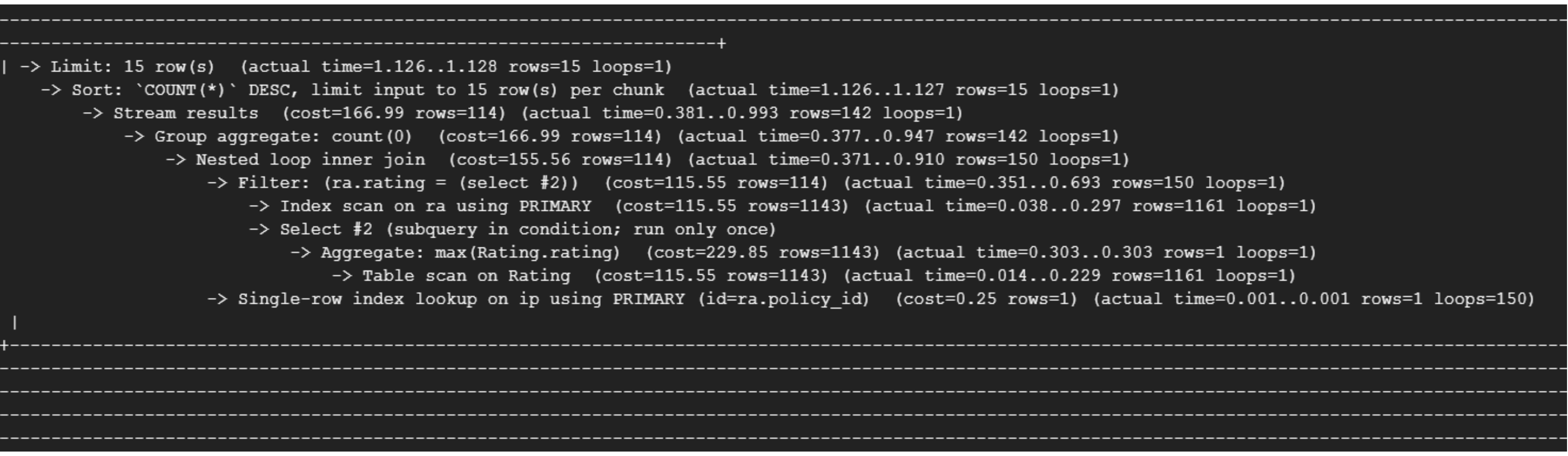
OPTIMISED ADVANCED QUERY 1: Next, we add an index to the **cover amount**. The below image depicts the performance of the query after adding an index on the cover_amount. As evident, the cost to fetch the max cover_amount has significantly reduced to 0.72 as now we are only doing an Index scan.



```
| -> Nested loop inner join  (cost=1.85 rows=2) (actual time=0.033..0.036 rows=1 loops=1)
    -> Nested loop inner join  (cost=1.29 rows=2) (actual time=0.022..0.024 rows=1 loops=1)
        -> Filter: (InsurancePolicy.cover_amount = (select #3))  (cost=0.72 rows=1) (actual time=0.007..0.008 rows=1 loops=1)
            -> Index lookup on InsurancePolicy using cvr_amt_idx (cover_amount=(select #3))  (cost=0.72 rows=1) (actual time=0.006..0.007 rows=1 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
        -> Filter: (ua.user_id is not null)  (cost=0.56 rows=2) (actual time=0.014..0.015 rows=1 loops=1)
            -> Index lookup on ua using policy_id (policy_id=InsurancePolicy.id)  (cost=0.56 rows=2) (actual time=0.013..0.014 rows=1 loops=1)
    -> Single-row index lookup on u using PRIMARY (id=ua.user_id)  (cost=0.31 rows=1) (actual time=0.011..0.011 rows=1 loops=1)
 |
```

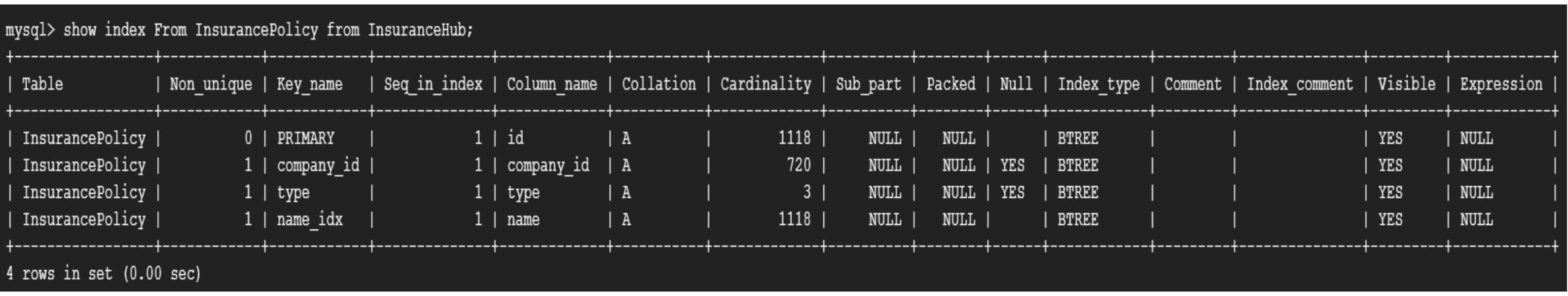
2. Adding an index to the second query-

The below image depicts the performance of the query before creating the index on ratings. Note that this performance is with indexes already added for PKs and FKs, which were added automatically by MySQL. As you can see, the performance overheads are significant.



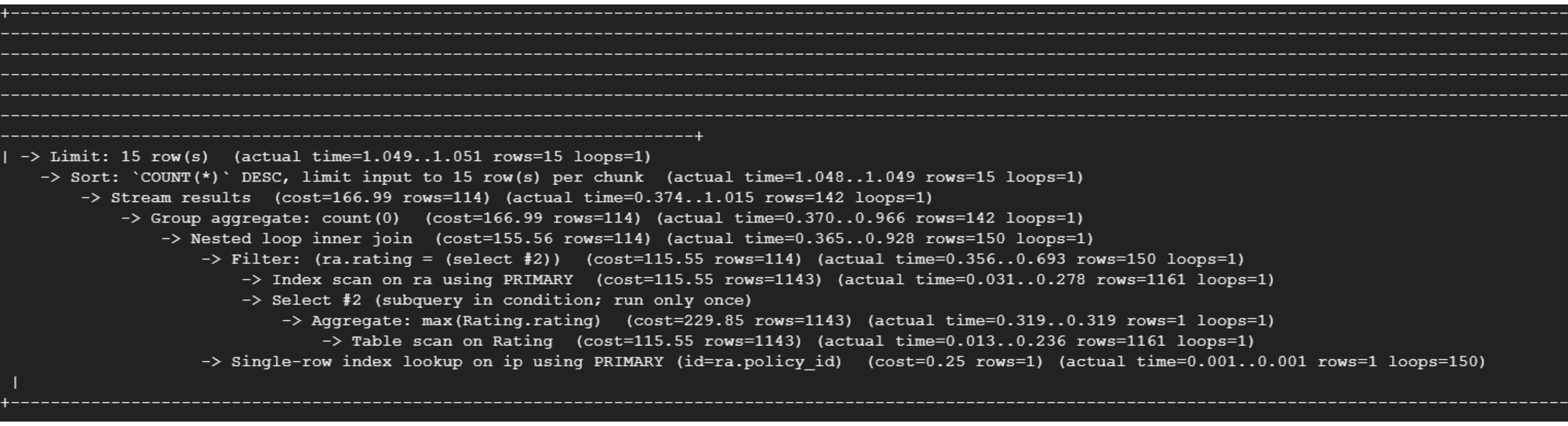
```
| -> Limit: 15 row(s)  (actual time=1.126..1.128 rows=15 loops=1)
    -> Sort: `COUNT(*)` DESC, limit input to 15 row(s) per chunk  (actual time=1.126..1.127 rows=15 loops=1)
        -> Stream results  (cost=166.99 rows=114) (actual time=0.381..0.993 rows=142 loops=1)
            -> Group aggregate: count(0)  (cost=166.99 rows=114) (actual time=0.377..0.947 rows=142 loops=1)
                -> Nested loop inner join  (cost=155.56 rows=114) (actual time=0.371..0.910 rows=150 loops=1)
                    -> Filter: (ra.rating = (select #2))  (cost=115.55 rows=114) (actual time=0.351..0.693 rows=150 loops=1)
                        -> Index scan on ra using PRIMARY  (cost=115.55 rows=1143) (actual time=0.038..0.297 rows=1161 loops=1)
                        -> Select #2 (subquery in condition; run only once)
                            -> Aggregate: max(Rating.rating)  (cost=229.85 rows=1143) (actual time=0.303..0.303 rows=1 loops=1)
                                -> Table scan on Rating  (cost=115.55 rows=1143) (actual time=0.014..0.229 rows=1161 loops=1)
                    -> Single-row index lookup on ip using PRIMARY (id=ra.policy_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=150)
 |
```

Now we try adding an index on the **name of the policy** on the InsurancePolicy table.



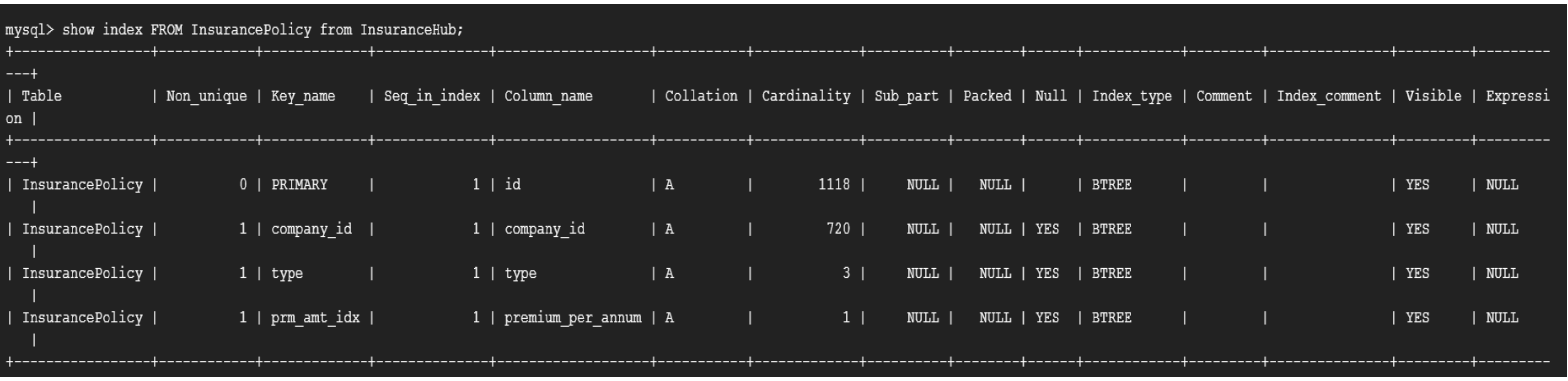
```
mysql> show index From InsurancePolicy from InsuranceHub;
+-----------------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table           | Non_unique | Key_name   | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-----------------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| InsurancePolicy |          0 | PRIMARY    |            1 | id          | A         |        1118 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| InsurancePolicy |          1 | company_id |            1 | company_id  | A         |         720 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL       |
| InsurancePolicy |          1 | type       |            1 | type        | A         |           3 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL       |
| InsurancePolicy |          1 | name_idx   |            1 | name        | A         |        1118 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+-----------------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
4 rows in set (0.00 sec)
```

As evident from the performance screenshot below, it didn’t provide any gains as the name is used only in select query and doesn’t impact performance.

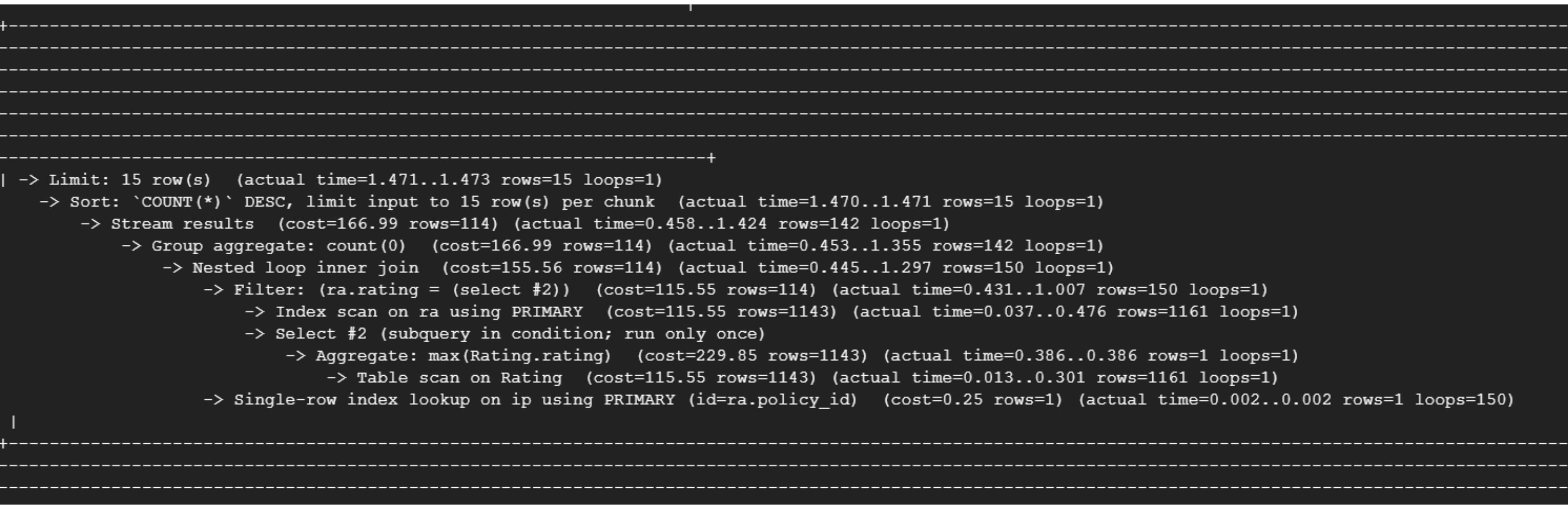


```
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=1.049..1.051 rows=15 loops=1)
    -> Sort: `COUNT(*)` DESC, limit input to 15 row(s) per chunk  (actual time=1.048..1.049 rows=15 loops=1)
        -> Stream results  (cost=166.99 rows=114) (actual time=0.374..1.015 rows=142 loops=1)
            -> Group aggregate: count(0)  (cost=166.99 rows=114) (actual time=0.370..0.966 rows=142 loops=1)
                -> Nested loop inner join  (cost=155.56 rows=114) (actual time=0.365..0.928 rows=150 loops=1)
                    -> Filter: (ra.rating = (select #2))  (cost=115.55 rows=114) (actual time=0.356..0.693 rows=150 loops=1)
                        -> Index scan on ra using PRIMARY  (cost=115.55 rows=1143) (actual time=0.031..0.278 rows=1161 loops=1)
                        -> Select #2 (subquery in condition; run only once)
                            -> Aggregate: max(Rating.rating)  (cost=229.85 rows=1143) (actual time=0.319..0.319 rows=1 loops=1)
                                -> Table scan on Rating  (cost=115.55 rows=1143) (actual time=0.013..0.236 rows=1161 loops=1)
                    -> Single-row index lookup on ip using PRIMARY (id=ra.policy_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=150)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Now we try adding an index on the **premium amount per annum attribute**. But it also doesn’t have any impact on the performance as it's only used in the select query.

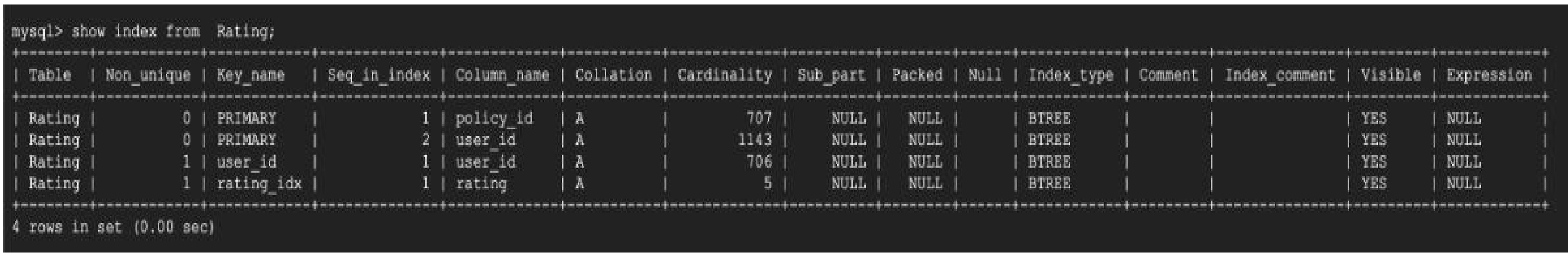


```
mysql> show index FROM InsurancePolicy from InsuranceHub;
+-----------------+------------+-------------+--------------+-------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+--------
---+
| Table           | Non_unique | Key_name    | Seq_in_index | Column_name       | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expressi
on |
+-----------------+------------+-------------+--------------+-------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+--------
---+
| InsurancePolicy |          0 | PRIMARY     |            1 | id                | A         |        1118 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL
   |
| InsurancePolicy |          1 | company_id  |            1 | company_id        | A         |         720 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL
   |
| InsurancePolicy |          1 | type        |            1 | type              | A         |           3 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL
   |
| InsurancePolicy |          1 | prm_amt_idx |            1 | premium_per_annum | A         |           1 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL
   |
+-----------------+------------+-------------+--------------+-------------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+--------
```



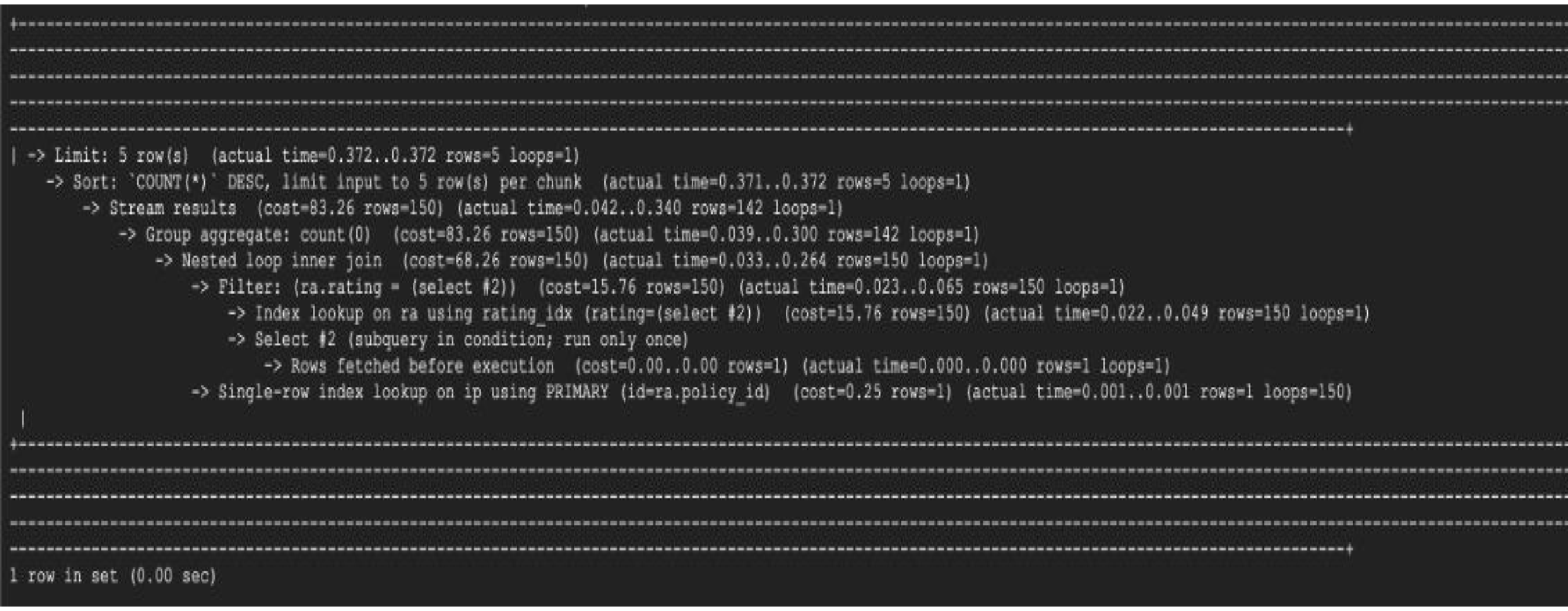
```
+--------------------------------------------------------------------------------
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
--------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=1.471..1.473 rows=15 loops=1)
    -> Sort: `COUNT(*)` DESC, limit input to 15 row(s) per chunk  (actual time=1.470..1.471 rows=15 loops=1)
        -> Stream results  (cost=166.99 rows=114) (actual time=0.458..1.424 rows=142 loops=1)
            -> Group aggregate: count(0)  (cost=166.99 rows=114) (actual time=0.453..1.355 rows=142 loops=1)
                -> Nested loop inner join  (cost=155.56 rows=114) (actual time=0.445..1.297 rows=150 loops=1)
                    -> Filter: (ra.rating = (select #2))  (cost=115.55 rows=114) (actual time=0.431..1.007 rows=150 loops=1)
                        -> Index scan on ra using PRIMARY  (cost=115.55 rows=1143) (actual time=0.037..0.476 rows=1161 loops=1)
                        -> Select #2 (subquery in condition; run only once)
                            -> Aggregate: max(Rating.rating)  (cost=229.85 rows=1143) (actual time=0.386..0.386 rows=1 loops=1)
                                -> Table scan on Rating  (cost=115.55 rows=1143) (actual time=0.013..0.301 rows=1161 loops=1)
                    -> Single-row index lookup on ip using PRIMARY (id=ra.policy_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=150)
 |
+--------------------------------------------------------------------------------
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
```

OPTIMISED ADVANCED QUERY 2: We have added an index to the **rating attribute** in the Ratings table. Note that since policy_id and user_id are PKs and FKs, they are automatically added as indexes by MYSQL.



```
mysql> show index from  Rating;
+--------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table  | Non_unique | Key_name   | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+--------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Rating |          0 | PRIMARY    |            1 | policy_id   | A         |         707 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| Rating |          0 | PRIMARY    |            2 | user_id     | A         |        1143 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| Rating |          1 | user_id    |            1 | user_id     | A         |         706 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| Rating |          1 | rating_idx |            1 | rating      | A         |           5 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+--------+------------+------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
4 rows in set (0.00 sec)
```

The below image depicts the performance of the query after adding an index on the rating. As evident, the cost to fetch the max rating has significantly reduced to 15.76 as now we are only doing an Index scan.



```
+--------------------------------------------------------------------------------
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| -> Limit: 5 row(s)  (actual time=0.372..0.372 rows=5 loops=1)
    -> Sort: `COUNT(*)` DESC, limit input to 5 row(s) per chunk  (actual time=0.371..0.372 rows=5 loops=1)
        -> Stream results  (cost=83.26 rows=150) (actual time=0.042..0.340 rows=142 loops=1)
            -> Group aggregate: count(0)  (cost=83.26 rows=150) (actual time=0.039..0.300 rows=142 loops=1)
                -> Nested loop inner join  (cost=68.26 rows=150) (actual time=0.033..0.264 rows=150 loops=1)
                    -> Filter: (ra.rating = (select #2))  (cost=15.76 rows=150) (actual time=0.023..0.065 rows=150 loops=1)
                        -> Index lookup on ra using rating_idx (rating=(select #2))  (cost=15.76 rows=150) (actual time=0.022..0.049 rows=150 loops=1)
                        -> Select #2 (subquery in condition; run only once)
                            -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
                    -> Single-row index lookup on ip using PRIMARY (id=ra.policy_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=150)
 |
+--------------------------------------------------------------------------------
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```