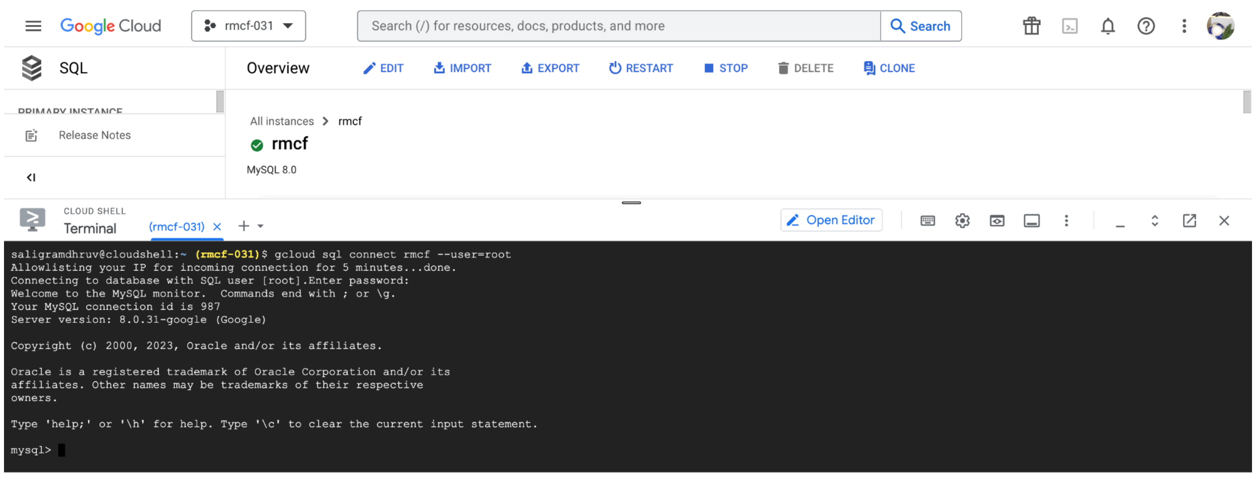
Proof of GCP Connection

We set up our database using GCP – pasted below is a screenshot of our terminal connection.



```
saligramdhruv@cloudshell:~ (rmcf-031)$ gcloud sql connect rmcf --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 987
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2023, 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>
```

DDL Commands For Table Creation

The DDL commands to create all of our tables are present below:

```sql
CREATE TABLE PoliceAreas (
      Area INT PRIMARY KEY,
      LocalPolicePhone INT
);
```

```sql
CREATE TABLE CrimeCodes (
      CrimeCode INT PRIMARY KEY,
      CrimeDescription VARCHAR(100)
);
```

```sql
CREATE TABLE Users (
      Username VARCHAR(20) PRIMARY KEY,
      Password VARCHAR(30),
      FirstName VARCHAR(50),
      LastName VARCHAR(60)
);
```

```sql
CREATE TABLE Schools (
	SchoolID INT PRIMARY KEY,
	SchoolName VARCHAR(100),
	SchoolType VARCHAR(100),
	SLongitude DECIMAL(7,4),
	SLatitude DECIMAL(6,4),
	Area INT,
	FOREIGN KEY (Area) REFERENCES PoliceAreas(Area) ON DELETE SET NULL ON
UPDATE CASCADE
);
```

```sql
CREATE TABLE Crimes (
	FileID INT PRIMARY KEY,
	CrimeDate VARCHAR(10),
	CrimeTime VARCHAR(15),
	CLongitude DECIMAL(7,4),
	CLatitude DECIMAL(6,4),
	VictimAge INT,
	WeaponUsed VARCHAR(50),
	CrimeCode INT,
	FOREIGN KEY (CrimeCode) REFERENCES CrimeCodes(CrimeCode) ON DELETE
CASCADE ON UPDATE CASCADE
);
```

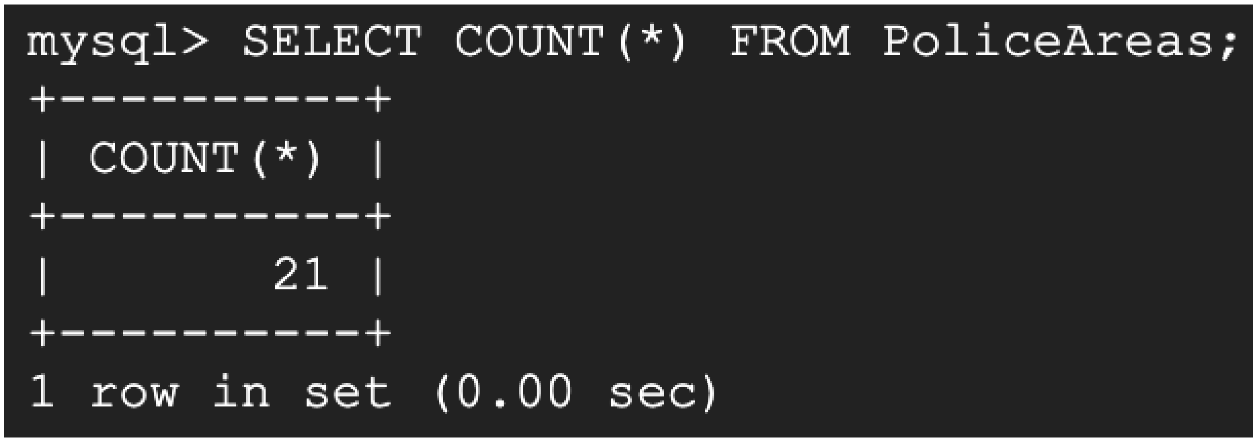
```sql
CREATE TABLE Ratings (
	Username VARCHAR(20),
	SchoolID INT,
	Rating DECIMAL(3,1),
	PRIMARY KEY (Username, SchoolID),
	FOREIGN KEY (Username) REFERENCES Users(Username) ON DELETE CASCADE ON
UPDATE CASCADE,
	FOREIGN KEY (SchoolID) REFERENCES Schools(SchoolID) ON DELETE CASCADE
ON UPDATE CASCADE
);
```

```
CREATE TABLE Favorites (
     Username VARCHAR(20),
     SchoolID INT,
     PRIMARY KEY (Username, SchoolID),
     FOREIGN KEY (Username) REFERENCES Users(Username) ON DELETE CASCADE ON
UPDATE CASCADE,
     FOREIGN KEY (SchoolID) REFERENCES Schools(SchoolID) ON DELETE CASCADE
ON UPDATE CASCADE
);
```

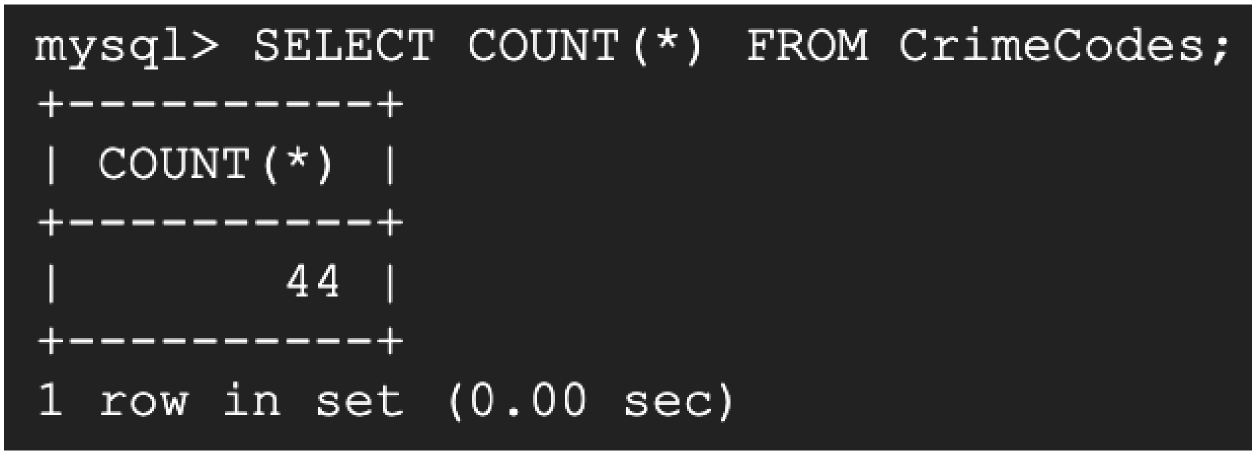
```
CREATE TABLE Suffer (
     FileID INT,
     SchoolID INT,
     PRIMARY KEY (FileID, SchoolID),
     FOREIGN KEY (FileID) REFERENCES Crimes(FileID) ON DELETE CASCADE ON
UPDATE CASCADE,
     FOREIGN KEY (SchoolID) REFERENCES Schools(SchoolID) ON DELETE CASCADE
ON UPDATE CASCADE
);
```

Table Counts

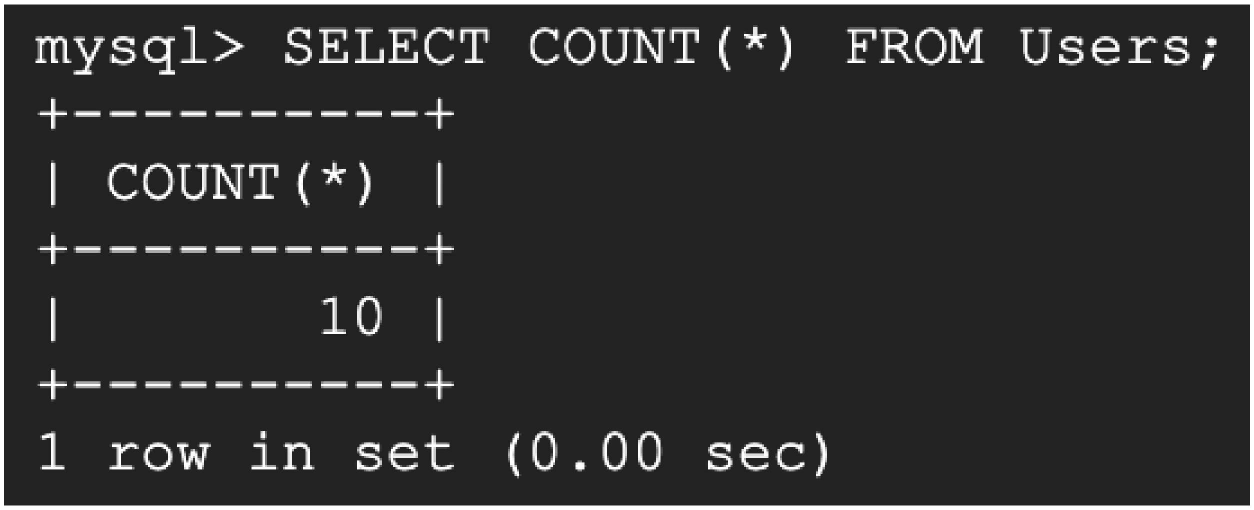
The counts for entries in each of our tables are provided below – Schools, Crimes, Ratings, Favorites, and Suffer all have 1,000+ entries.



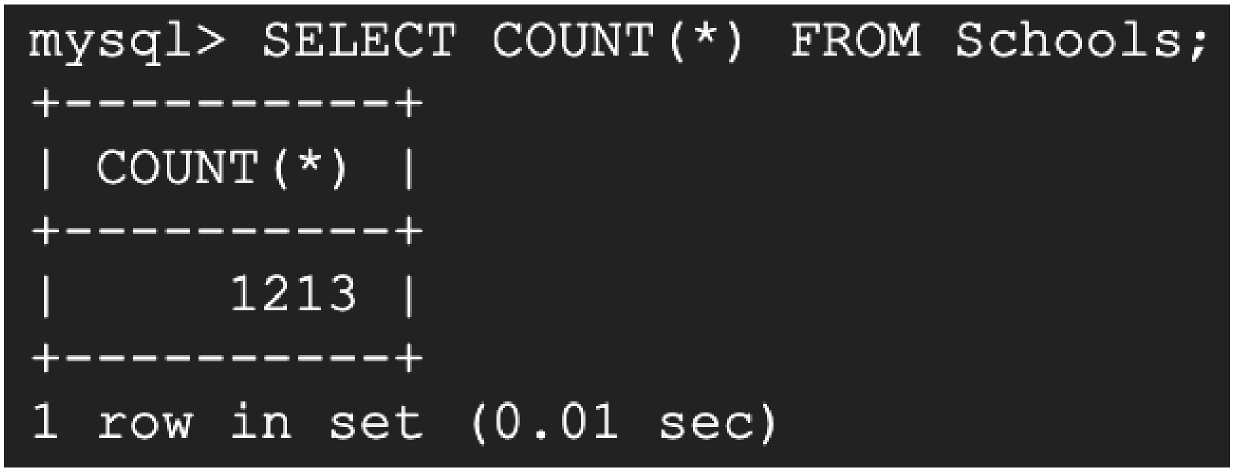
```
mysql> SELECT COUNT(*) FROM PoliceAreas;
+----------+
| COUNT(*) |
+----------+
|       21 |
+----------+
1 row in set (0.00 sec)
```



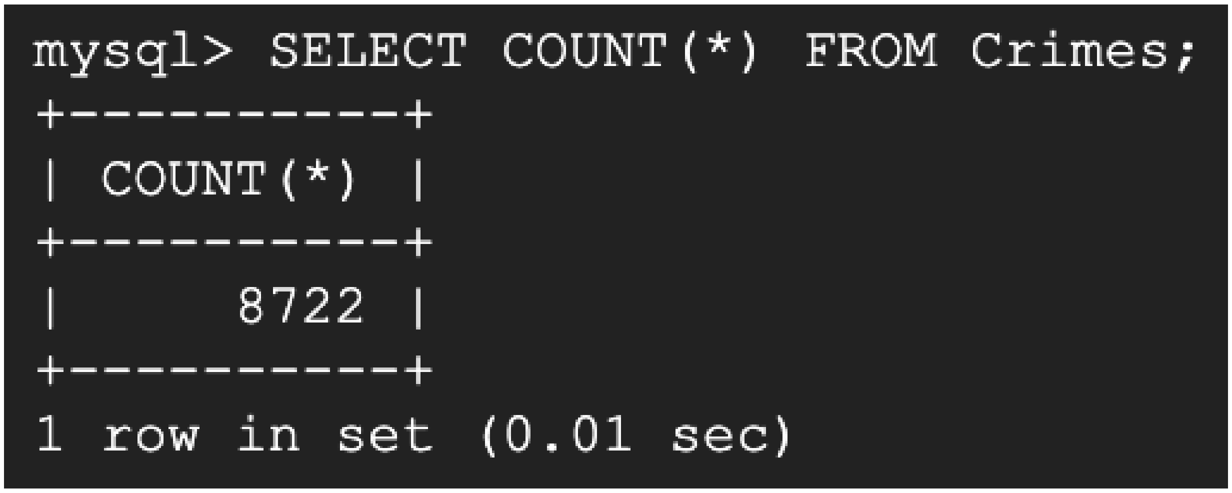
```
mysql> SELECT COUNT(*) FROM CrimeCodes;
+----------+
| COUNT(*) |
+----------+
|       44 |
+----------+
1 row in set (0.00 sec)
```



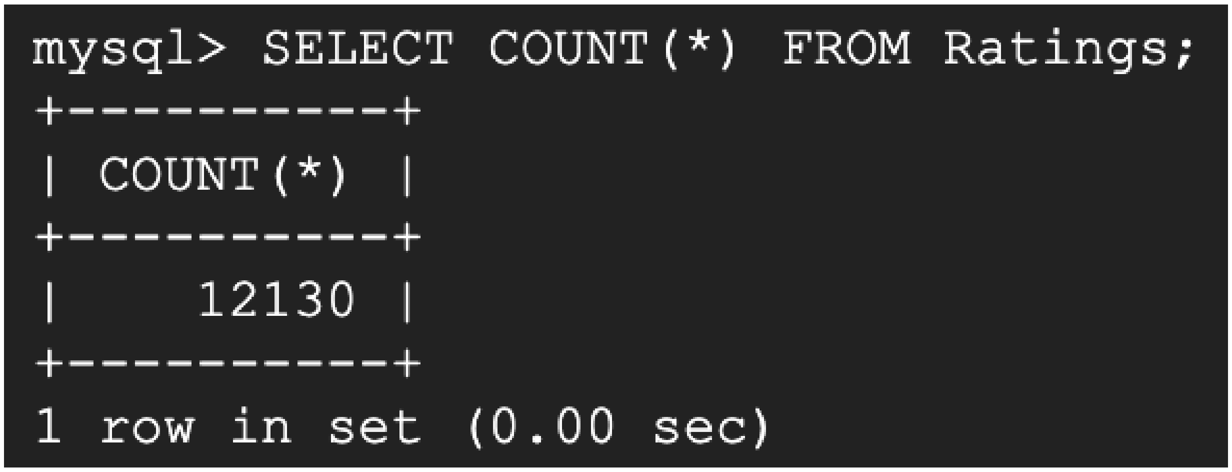
```
mysql> SELECT COUNT(*) FROM Users;
+----------+
| COUNT(*) |
+----------+
|       10 |
+----------+
1 row in set (0.00 sec)
```



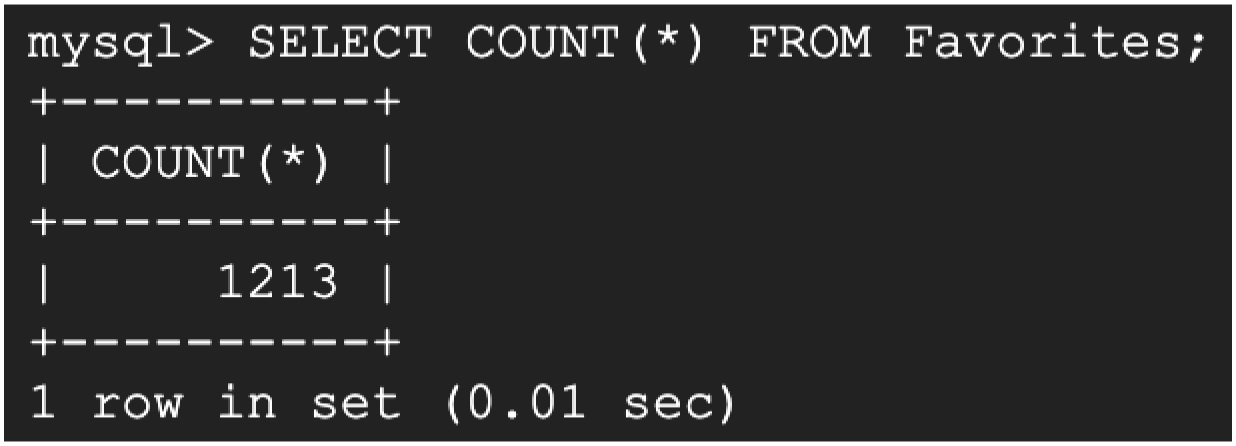
```
mysql> SELECT COUNT(*) FROM Schools;
+----------+
| COUNT(*) |
+----------+
|     1213 |
+----------+
1 row in set (0.01 sec)
```



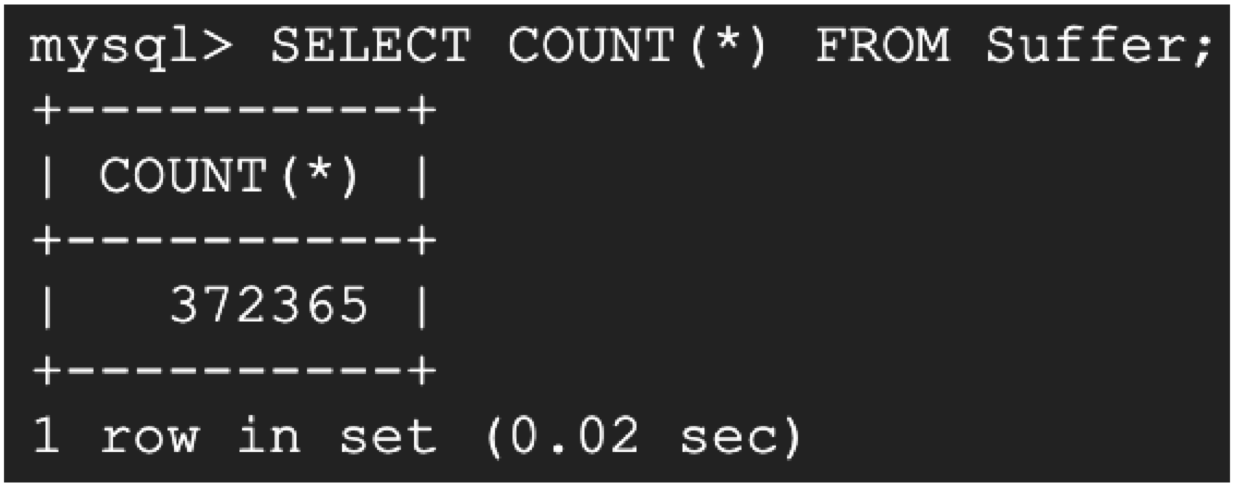
```
mysql> SELECT COUNT(*) FROM Crimes;
+----------+
| COUNT(*) |
+----------+
|     8722 |
+----------+
1 row in set (0.01 sec)
```



```
mysql> SELECT COUNT(*) FROM Ratings;
+----------+
| COUNT(*) |
+----------+
|    12130 |
+----------+
1 row in set (0.00 sec)
```



```
mysql> SELECT COUNT(*) FROM Favorites;
+----------+
| COUNT(*) |
+----------+
|     1213 |
+----------+
1 row in set (0.01 sec)
```



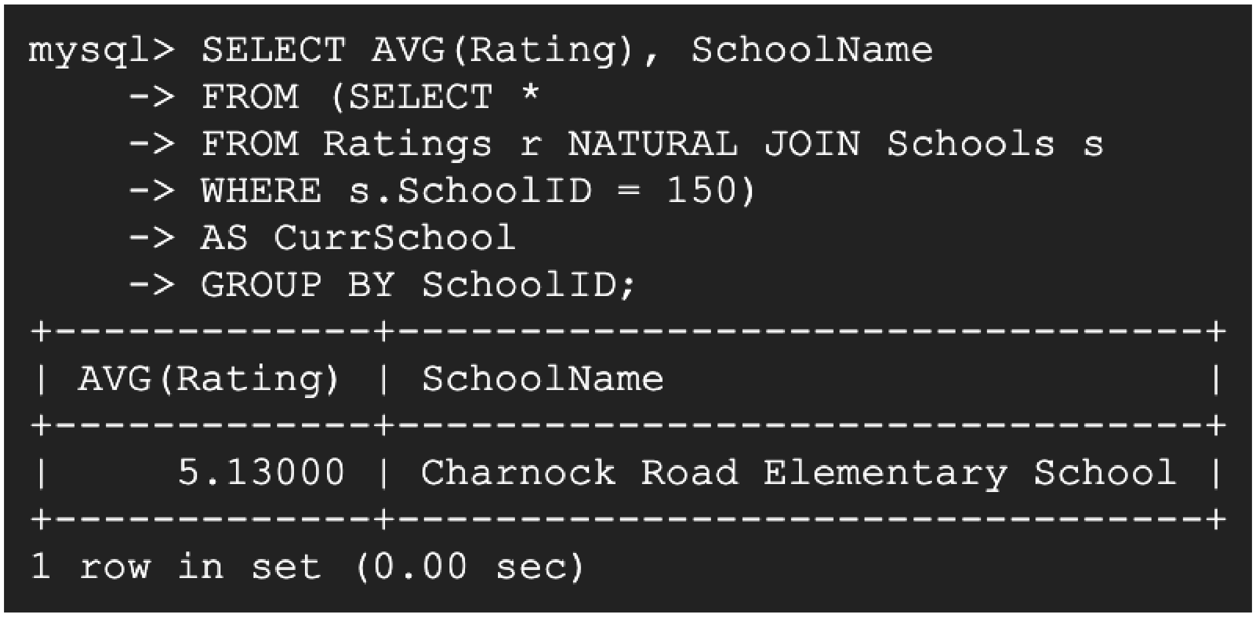
```
mysql> SELECT COUNT(*) FROM Suffer;
+----------+
| COUNT(*) |
+----------+
|   372365 |
+----------+
1 row in set (0.02 sec)
```

Advanced Query #1

Our first advanced query is as follows:

```sql
SELECT AVG(Rating), SchoolName
FROM (SELECT *
FROM Ratings r NATURAL JOIN Schools s
WHERE s.SchoolID = curr)
AS CurrSchool
GROUP BY SchoolID;
```

This advanced query involves aggregation via GROUP BY, join of multiple relations, and a subquery. In this query, “curr” will represent a SchoolID that a user has clicked on in our application. This ID will be routed to our backend, where we can execute the above query. For demonstrative purposes, “curr” has been replaced with 150 in the execution below – this would represent the output when a user clicks on the school with SchoolID 150 in our application.



```
mysql> SELECT AVG(Rating), SchoolName
    -> FROM (SELECT *
    -> FROM Ratings r NATURAL JOIN Schools s
    -> WHERE s.SchoolID = 150)
    -> AS CurrSchool
    -> GROUP BY SchoolID;
+-------------+-------------------------------------+
| AVG(Rating) | SchoolName                          |
+-------------+-------------------------------------+
|     5.13000 | Charnock Road Elementary School     |
+-------------+-------------------------------------+
1 row in set (0.00 sec)
```

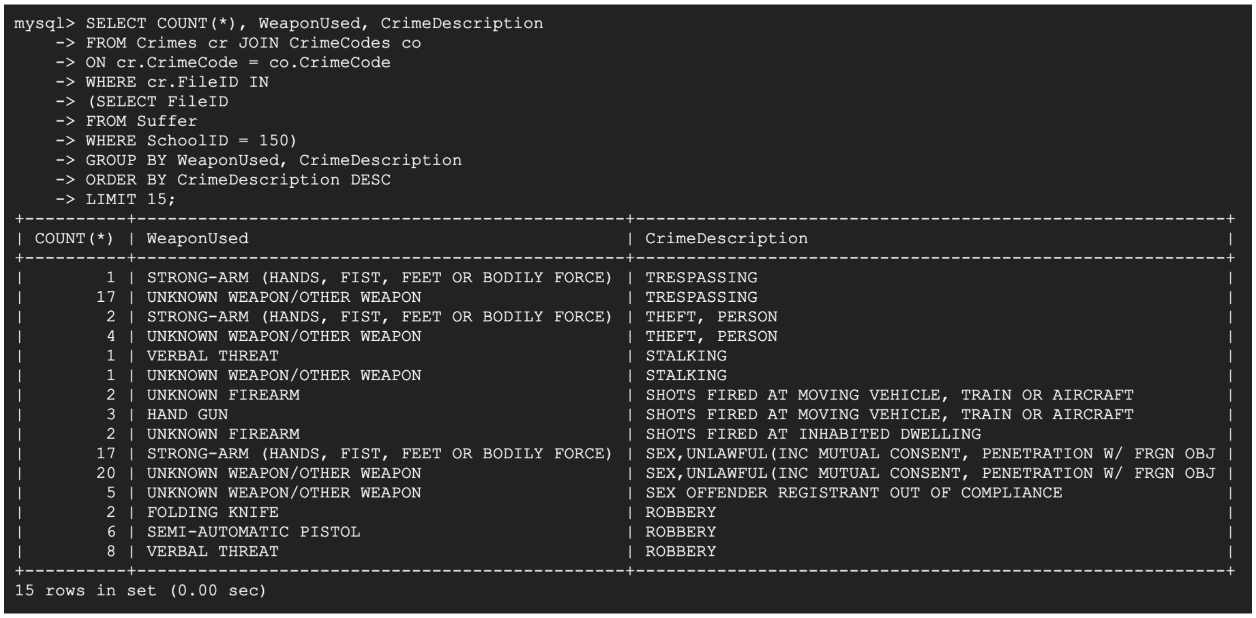
The output for this advanced query is only 1 row since we are getting the average rating for a single given school along with its full name.

Advanced Query #2

Our second advanced query is as follows:

```
SELECT COUNT(*), WeaponUsed, CrimeDescription
FROM Crimes cr JOIN CrimeCodes co
ON cr.CrimeCode = co.CrimeCode
WHERE cr.FileID IN
(SELECT FileID
FROM Suffer
WHERE SchoolID = curr)
GROUP BY WeaponUsed, CrimeDescription;
```

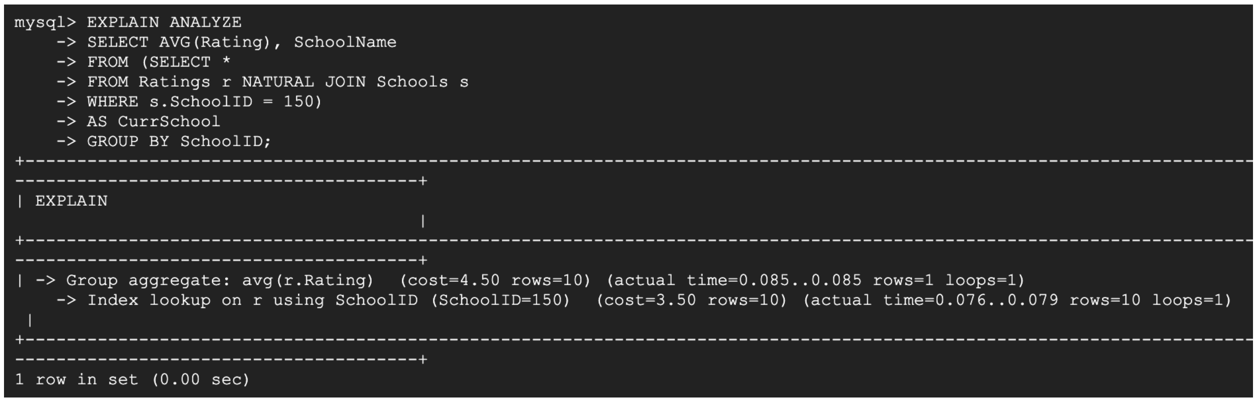
This advanced query involves aggregation via GROUP BY, join of multiple relations, and a subquery. In this query, “curr” will represent a SchoolID that a user has clicked on in our application. This ID will be routed to our backend, where we can execute the above query. For demonstrative purposes, “curr” has been replaced with 150 in the execution below – this would represent the output when a user wants to view the crime information for the school with SchoolID 150 in our application.


```
mysql> SELECT COUNT(*), WeaponUsed, CrimeDescription
    -> FROM Crimes cr JOIN CrimeCodes co
    -> ON cr.CrimeCode = co.CrimeCode
    -> WHERE cr.FileID IN
    -> (SELECT FileID
    -> FROM Suffer
    -> WHERE SchoolID = 150)
    -> GROUP BY WeaponUsed, CrimeDescription
    -> ORDER BY CrimeDescription DESC
    -> LIMIT 15;
+----------+------------------------------------------------+----------------------------------------------------------+
| COUNT(*) | WeaponUsed                                     | CrimeDescription                                         |
+----------+------------------------------------------------+----------------------------------------------------------+
|        1 | STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE) | TRESPASSING                                              |
|       17 | UNKNOWN WEAPON/OTHER WEAPON                    | TRESPASSING                                              |
|        2 | STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE) | THEFT, PERSON                                            |
|        4 | UNKNOWN WEAPON/OTHER WEAPON                    | THEFT, PERSON                                            |
|        1 | VERBAL THREAT                                  | STALKING                                                 |
|        1 | UNKNOWN WEAPON/OTHER WEAPON                    | STALKING                                                 |
|        2 | UNKNOWN FIREARM                                | SHOTS FIRED AT MOVING VEHICLE, TRAIN OR AIRCRAFT         |
|        3 | HAND GUN                                       | SHOTS FIRED AT MOVING VEHICLE, TRAIN OR AIRCRAFT         |
|        2 | UNKNOWN FIREARM                                | SHOTS FIRED AT INHABITED DWELLING                        |
|       17 | STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE) | SEX,UNLAWFUL(INC MUTUAL CONSENT, PENETRATION W/ FRGN OBJ |
|       20 | UNKNOWN WEAPON/OTHER WEAPON                    | SEX,UNLAWFUL(INC MUTUAL CONSENT, PENETRATION W/ FRGN OBJ |
|        5 | UNKNOWN WEAPON/OTHER WEAPON                    | SEX OFFENDER REGISTRANT OUT OF COMPLIANCE                |
|        2 | FOLDING KNIFE                                  | ROBBERY                                                  |
|        6 | SEMI-AUTOMATIC PISTOL                          | ROBBERY                                                  |
|        8 | VERBAL THREAT                                  | ROBBERY                                                  |
+----------+------------------------------------------------+----------------------------------------------------------+
15 rows in set (0.00 sec)
```

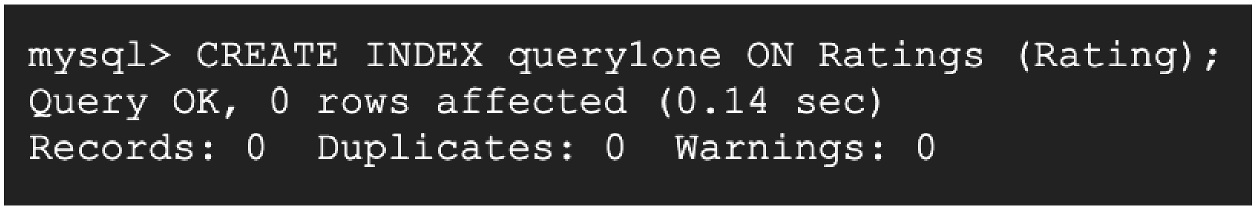
The output is sorted by CrimeDescription purely for formatting purposes in the above screenshot, and we limit the output to 15 rows.

Advanced Query #1 Indexing

When we use EXPLAIN ANALYZE on our advanced query initially, we observe the following:

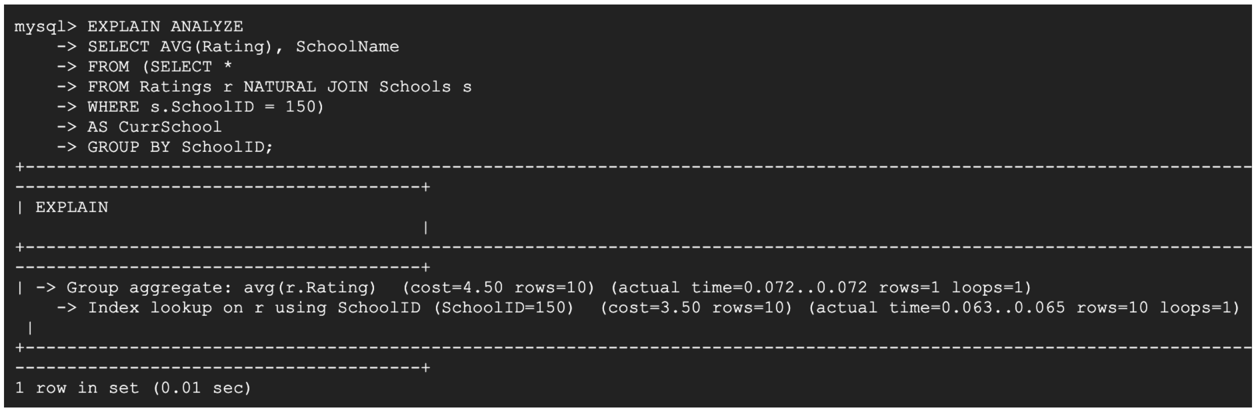

```
mysql> EXPLAIN ANALYZE
    -> SELECT AVG(Rating), SchoolName
    -> FROM (SELECT *
    -> FROM Ratings r NATURAL JOIN Schools s
    -> WHERE s.SchoolID = 150)
    -> AS CurrSchool
    -> GROUP BY SchoolID;
+------------------------------------------------------------------------------------------------------------------------
--------------------------------------+
| EXPLAIN
                                       |
+------------------------------------------------------------------------------------------------------------------------
--------------------------------------+
| -> Group aggregate: avg(r.Rating)  (cost=4.50 rows=10) (actual time=0.085..0.085 rows=1 loops=1)
    -> Index lookup on r using SchoolID (SchoolID=150)  (cost=3.50 rows=10) (actual time=0.076..0.079 rows=10 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------
--------------------------------------+
1 row in set (0.00 sec)
```

We can then add an index on the Rating attribute in our Ratings table:



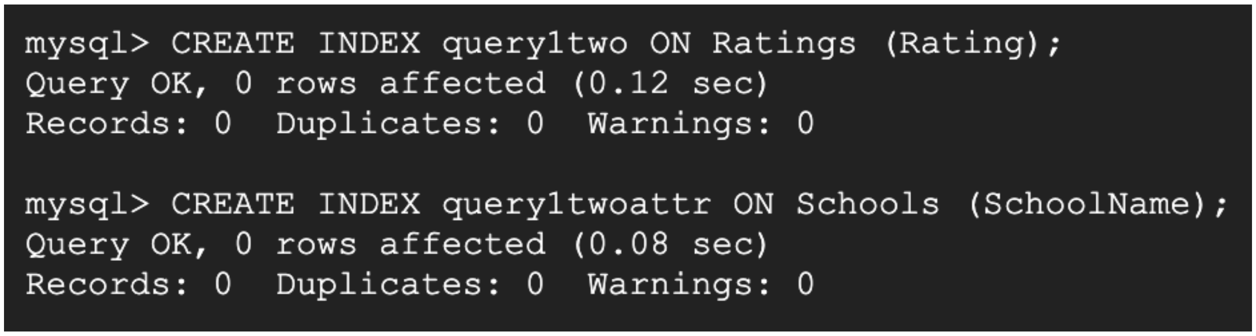
```
mysql> CREATE INDEX query1one ON Ratings (Rating);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Using EXPLAIN ANALYZE with this indexing design yields the following:



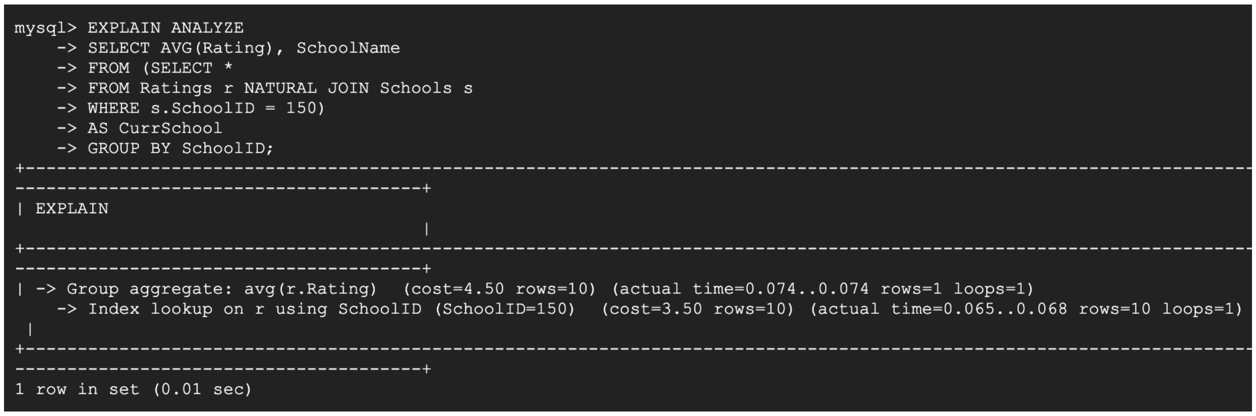
```
mysql> EXPLAIN ANALYZE
    -> SELECT AVG(Rating), SchoolName
    -> FROM (SELECT *
    -> FROM Ratings r NATURAL JOIN Schools s
    -> WHERE s.SchoolID = 150)
    -> AS CurrSchool
    -> GROUP BY SchoolID;
+-----------------------------------------------------------------------------------------------------------------------
---------------------------------------+
| EXPLAIN
                                       |
+-----------------------------------------------------------------------------------------------------------------------
---------------------------------------+
| -> Group aggregate: avg(r.Rating)  (cost=4.50 rows=10) (actual time=0.072..0.072 rows=1 loops=1)
    -> Index lookup on r using SchoolID (SchoolID=150)  (cost=3.50 rows=10) (actual time=0.063..0.065 rows=10 loops=1)
 |
+-----------------------------------------------------------------------------------------------------------------------
---------------------------------------+
1 row in set (0.01 sec)
```

Resetting our indices, we can establish an index on Rating in Ratings and an index on SchoolName in Schools as such:



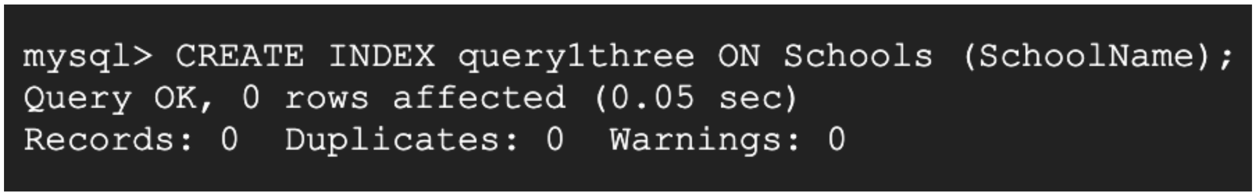
```
mysql> CREATE INDEX query1two ON Ratings (Rating);
Query OK, 0 rows affected (0.12 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX query1twoattr ON Schools (SchoolName);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Using EXPLAIN ANALYZE with this indexing design yields the following:



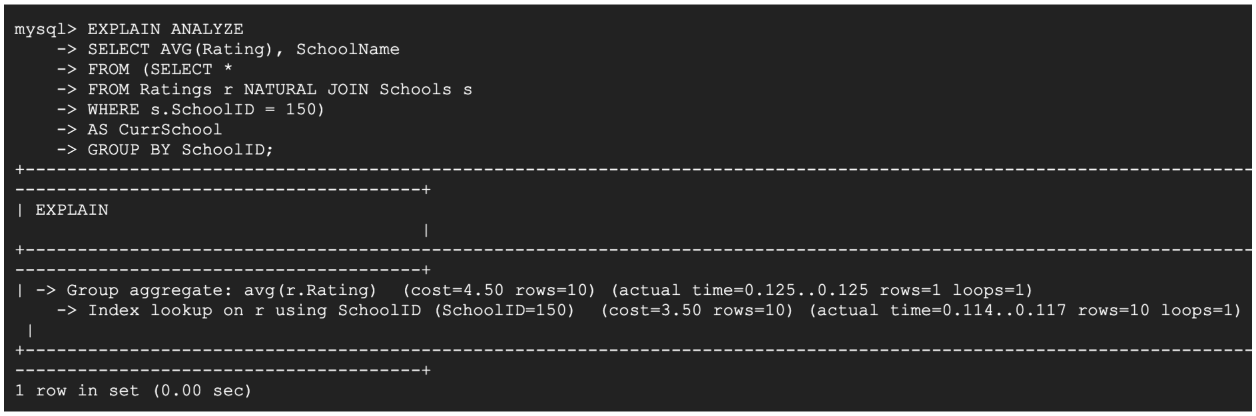
```
mysql> EXPLAIN ANALYZE
    -> SELECT AVG(Rating), SchoolName
    -> FROM (SELECT *
    -> FROM Ratings r NATURAL JOIN Schools s
    -> WHERE s.SchoolID = 150)
    -> AS CurrSchool
    -> GROUP BY SchoolID;
+----------------------------------------------------------------------------------------------------------------------
---------------------------------------+
| EXPLAIN
                                       |
+----------------------------------------------------------------------------------------------------------------------
---------------------------------------+
| -> Group aggregate: avg(r.Rating)  (cost=4.50 rows=10) (actual time=0.074..0.074 rows=1 loops=1)
    -> Index lookup on r using SchoolID (SchoolID=150)  (cost=3.50 rows=10) (actual time=0.065..0.068 rows=10 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------------
---------------------------------------+
1 row in set (0.01 sec)
```

Finally, resetting our indices again, we can establish an index on just SchoolName in Schools:



```
mysql> CREATE INDEX query1three ON Schools (SchoolName);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Using EXPLAIN ANALYZE with this indexing design yields the following:



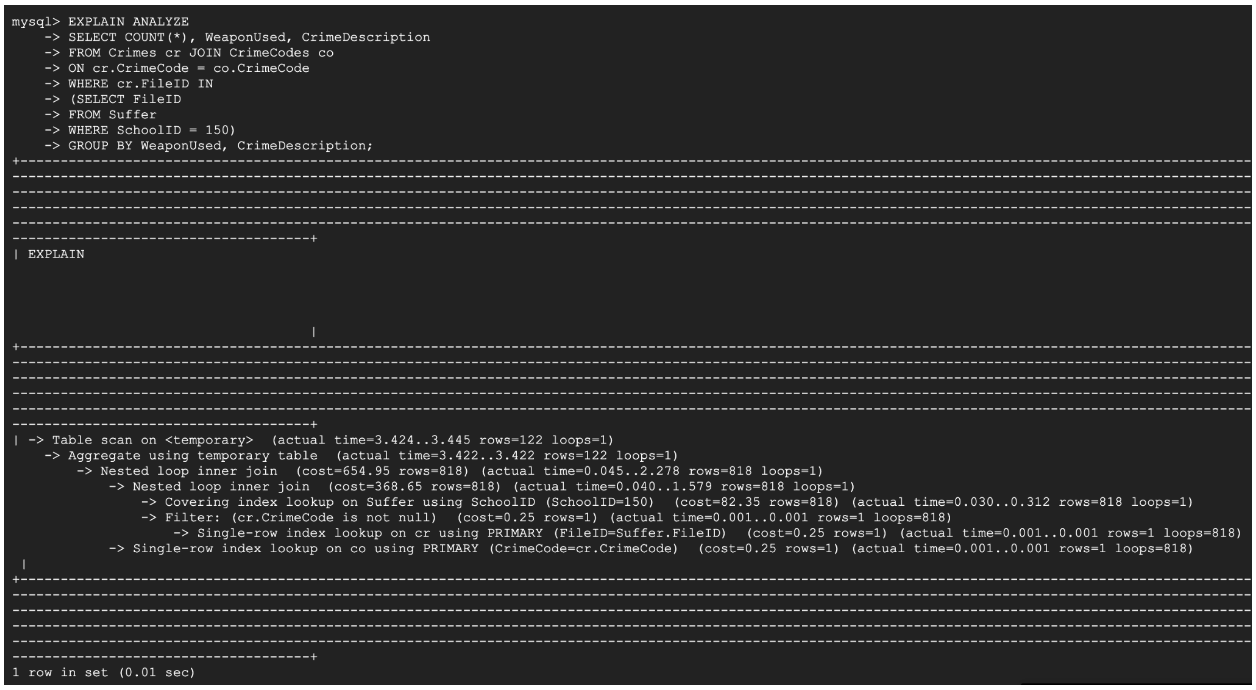
```
mysql> EXPLAIN ANALYZE
    -> SELECT AVG(Rating), SchoolName
    -> FROM (SELECT *
    -> FROM Ratings r NATURAL JOIN Schools s
    -> WHERE s.SchoolID = 150)
    -> AS CurrSchool
    -> GROUP BY SchoolID;
+----------------------------------------------------------------------------------------------------------------------
---------------------------------------+
| EXPLAIN
                                       |
+----------------------------------------------------------------------------------------------------------------------
---------------------------------------+
| -> Group aggregate: avg(r.Rating)  (cost=4.50 rows=10) (actual time=0.125..0.125 rows=1 loops=1)
    -> Index lookup on r using SchoolID (SchoolID=150)  (cost=3.50 rows=10) (actual time=0.114..0.117 rows=10 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------------
---------------------------------------+
1 row in set (0.00 sec)
```

Ultimately, all four indexing designs yield almost the same results with little to no differences. The cost for the group aggregate remained 4.5 and the cost for index lookup remained 3.5 across all four indexing designs (including the default index schema). With that in mind, and given that the actual time was largely consistent across all the indexing designs, we have decided to use the default index setup.

We believe that the changes of indices did not affect our query performance because of the nature of the information being selected in the subquery. Given that we are pulling every column from the joining of Ratings and Schools, perhaps more indexing on every column of the joint table is required to see an increase in performance. Conversely, perhaps selecting fewer columns from the joint Ratings and Schools table would see better results based on our index designs listed in this section. These are both approaches that we will likely further explore and attempt while developing Stage 4.

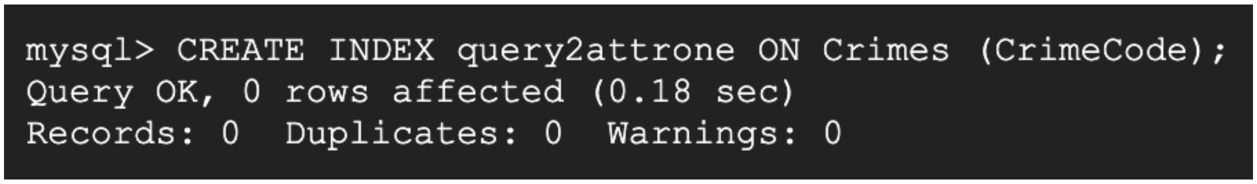
Advanced Query #2 Indexing

When we use EXPLAIN ANALYZE on our advanced query initially, we observe the following:



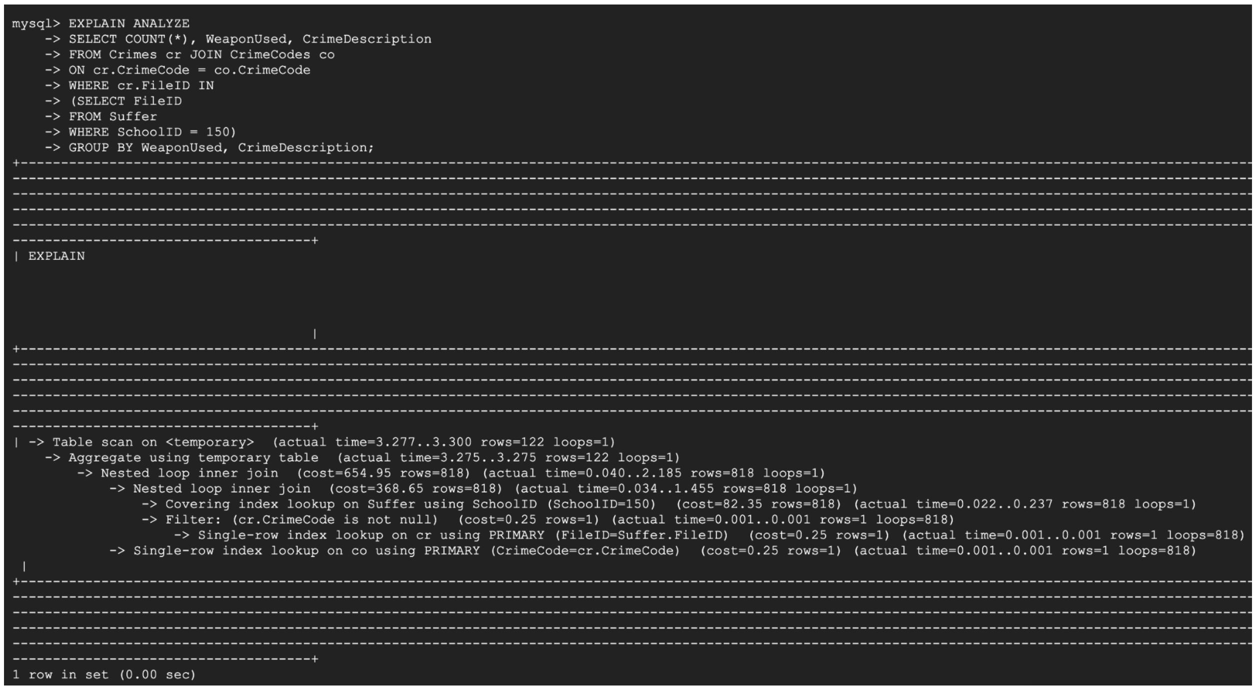
```
mysql> EXPLAIN ANALYZE
    -> SELECT COUNT(*), WeaponUsed, CrimeDescription
    -> FROM Crimes cr JOIN CrimeCodes co
    -> ON cr.CrimeCode = co.CrimeCode
    -> WHERE cr.FileID IN
    -> (SELECT FileID
    -> FROM Suffer
    -> WHERE SchoolID = 150)
    -> GROUP BY WeaponUsed, CrimeDescription;
+------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+
| EXPLAIN




                                     |
+------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+
| -> Table scan on <temporary>  (actual time=3.424..3.445 rows=122 loops=1)
    -> Aggregate using temporary table  (actual time=3.422..3.422 rows=122 loops=1)
        -> Nested loop inner join  (cost=654.95 rows=818) (actual time=0.045..2.278 rows=818 loops=1)
            -> Nested loop inner join  (cost=368.65 rows=818) (actual time=0.040..1.579 rows=818 loops=1)
                -> Covering index lookup on Suffer using SchoolID (SchoolID=150)  (cost=82.35 rows=818) (actual time=0.030..0.312 rows=818 loops=1)
                -> Filter: (cr.CrimeCode is not null)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
                    -> Single-row index lookup on cr using PRIMARY (FileID=Suffer.FileID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
            -> Single-row index lookup on co using PRIMARY (CrimeCode=cr.CrimeCode)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+
1 row in set (0.01 sec)
```

We can then add an index on the CrimeCode attribute in our Crimes table:



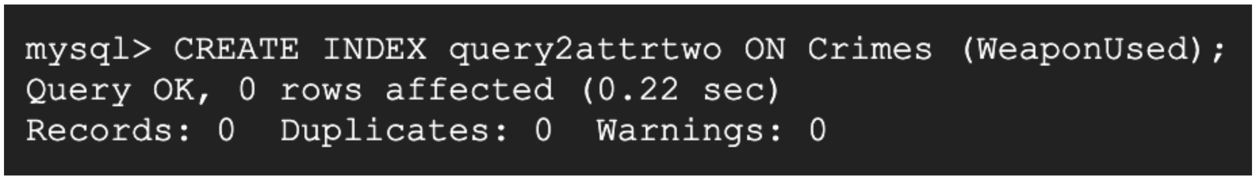
```
mysql> CREATE INDEX query2attrone ON Crimes (CrimeCode);
Query OK, 0 rows affected (0.18 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Using EXPLAIN ANALYZE with this indexing design yields the following:



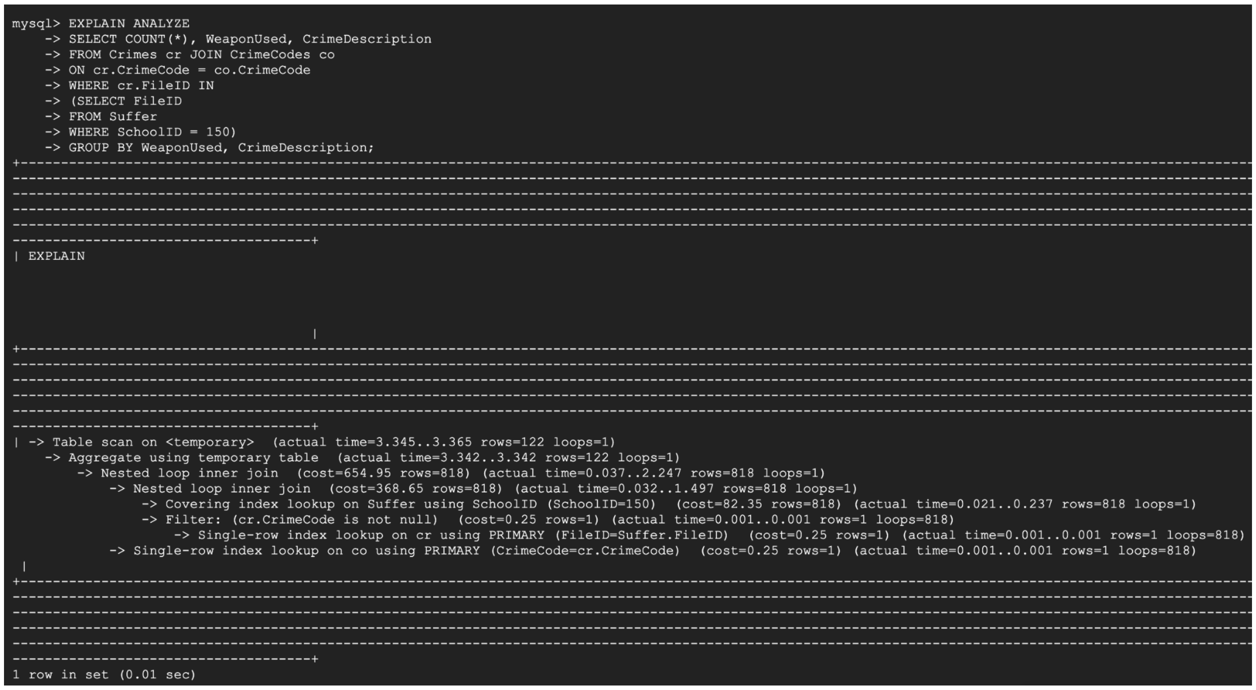
```
mysql> EXPLAIN ANALYZE
    -> SELECT COUNT(*), WeaponUsed, CrimeDescription
    -> FROM Crimes cr JOIN CrimeCodes co
    -> ON cr.CrimeCode = co.CrimeCode
    -> WHERE cr.FileID IN
    -> (SELECT FileID
    -> FROM Suffer
    -> WHERE SchoolID = 150)
    -> GROUP BY WeaponUsed, CrimeDescription;
+------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+
| EXPLAIN





                                     |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+
| -> Table scan on <temporary>  (actual time=3.277..3.300 rows=122 loops=1)
    -> Aggregate using temporary table  (actual time=3.275..3.275 rows=122 loops=1)
        -> Nested loop inner join  (cost=654.95 rows=818) (actual time=0.040..2.185 rows=818 loops=1)
            -> Nested loop inner join  (cost=368.65 rows=818) (actual time=0.034..1.455 rows=818 loops=1)
                -> Covering index lookup on Suffer using SchoolID (SchoolID=150)  (cost=82.35 rows=818) (actual time=0.022..0.237 rows=818 loops=1)
                -> Filter: (cr.CrimeCode is not null)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
                    -> Single-row index lookup on cr using PRIMARY (FileID=Suffer.FileID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
            -> Single-row index lookup on co using PRIMARY (CrimeCode=cr.CrimeCode)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+
1 row in set (0.00 sec)
```

Building off of this, we can establish another index on WeaponUsed in Crimes on top of the previous index (meaning we now have 2 additional indices on the Crimes table):



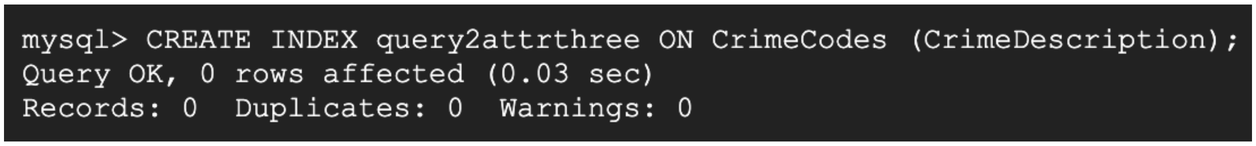
```
mysql> CREATE INDEX query2attrtwo ON Crimes (WeaponUsed);
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Using EXPLAIN ANALYZE with this indexing design yields the following:



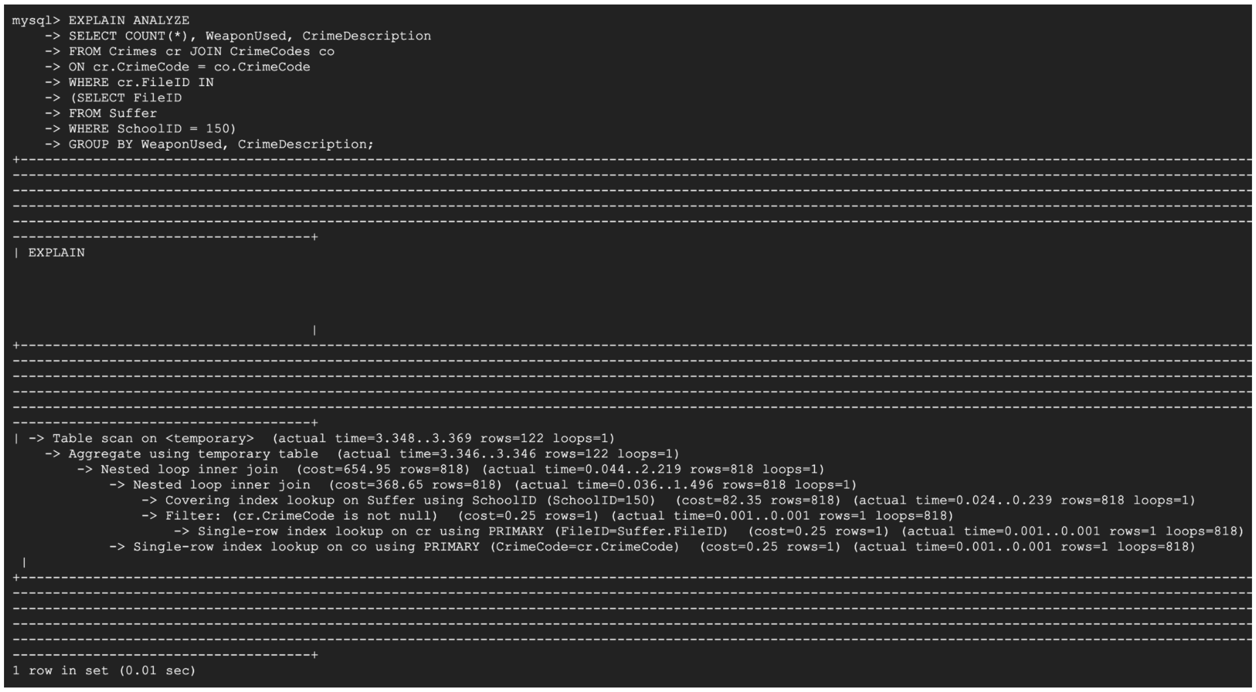
```
mysql> EXPLAIN ANALYZE
    -> SELECT COUNT(*), WeaponUsed, CrimeDescription
    -> FROM Crimes cr JOIN CrimeCodes co
    -> ON cr.CrimeCode = co.CrimeCode
    -> WHERE cr.FileID IN
    -> (SELECT FileID
    -> FROM Suffer
    -> WHERE SchoolID = 150)
    -> GROUP BY WeaponUsed, CrimeDescription;
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------+
| EXPLAIN




                                      |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------+
| -> Table scan on <temporary>  (actual time=3.345..3.365 rows=122 loops=1)
    -> Aggregate using temporary table  (actual time=3.342..3.342 rows=122 loops=1)
        -> Nested loop inner join  (cost=654.95 rows=818) (actual time=0.037..2.247 rows=818 loops=1)
            -> Nested loop inner join  (cost=368.65 rows=818) (actual time=0.032..1.497 rows=818 loops=1)
                -> Covering index lookup on Suffer using SchoolID (SchoolID=150)  (cost=82.35 rows=818) (actual time=0.021..0.237 rows=818 loops=1)
                -> Filter: (cr.CrimeCode is not null)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
                    -> Single-row index lookup on cr using PRIMARY (FileID=Suffer.FileID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
            -> Single-row index lookup on co using PRIMARY (CrimeCode=cr.CrimeCode)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------+
1 row in set (0.01 sec)
```

Finally, adding a third index, we can establish an index on CrimeDescription in CrimeCodes:



```
mysql> CREATE INDEX query2attrthree ON CrimeCodes (CrimeDescription);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Using EXPLAIN ANALYZE with this indexing design yields the following:



```
mysql> EXPLAIN ANALYZE
    -> SELECT COUNT(*), WeaponUsed, CrimeDescription
    -> FROM Crimes cr JOIN CrimeCodes co
    -> ON cr.CrimeCode = co.CrimeCode
    -> WHERE cr.FileID IN
    -> (SELECT FileID
    -> FROM Suffer
    -> WHERE SchoolID = 150)
    -> GROUP BY WeaponUsed, CrimeDescription;
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+
| EXPLAIN




                                     |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+
| -> Table scan on <temporary>  (actual time=3.348..3.369 rows=122 loops=1)
    -> Aggregate using temporary table  (actual time=3.346..3.346 rows=122 loops=1)
        -> Nested loop inner join  (cost=654.95 rows=818) (actual time=0.044..2.219 rows=818 loops=1)
            -> Nested loop inner join  (cost=368.65 rows=818) (actual time=0.036..1.496 rows=818 loops=1)
                -> Covering index lookup on Suffer using SchoolID (SchoolID=150)  (cost=82.35 rows=818) (actual time=0.024..0.239 rows=818 loops=1)
                -> Filter: (cr.CrimeCode is not null)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
                    -> Single-row index lookup on cr using PRIMARY (FileID=Suffer.FileID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
            -> Single-row index lookup on co using PRIMARY (CrimeCode=cr.CrimeCode)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=818)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------+
1 row in set (0.01 sec)
```

Overall, our indexing designs included the default index schema, an additional index on WeaponUsed, two additional indices (one one WeaponUsed and one on CrimeDescription), and finally, three additional indices (the two previous indices mentioned along with an index on CrimeCode). Ultimately, all four indexing designs yield almost the same results with little to no differences. The cost for the first nested loop inner join remained 654.95, the cost for the second nested loop inner join remained 368.65, the cost for Suffer index lookup remained 82.35, and the costs for all of filtering CrimeCode, lookup on FileID, and lookup on CrimeCode remained 0.25 across all four indexing designs (including the default index schema). With that in mind, and given that the actual time was largely consistent across all the indexing designs, we have decided to use the default index setup.

Given that our final index design involved an index on every non-primary key attribute present in our query, we expected it to have a better performance in comparison to the other designs. However, one guess we had as to why the performance remained the same is because of our COUNT(*) in the select statement. Since * involves every column, perhaps we would have needed an index on every column present in the table in order to see performance improvements. One possible extension to these indices would be to add an index to all columns in the output table to test for improvements in performance. Another possible extension would be to specifically count one single column that has an index as opposed to counting *. These are both approaches that we will likely further explore and attempt while developing Stage 4.