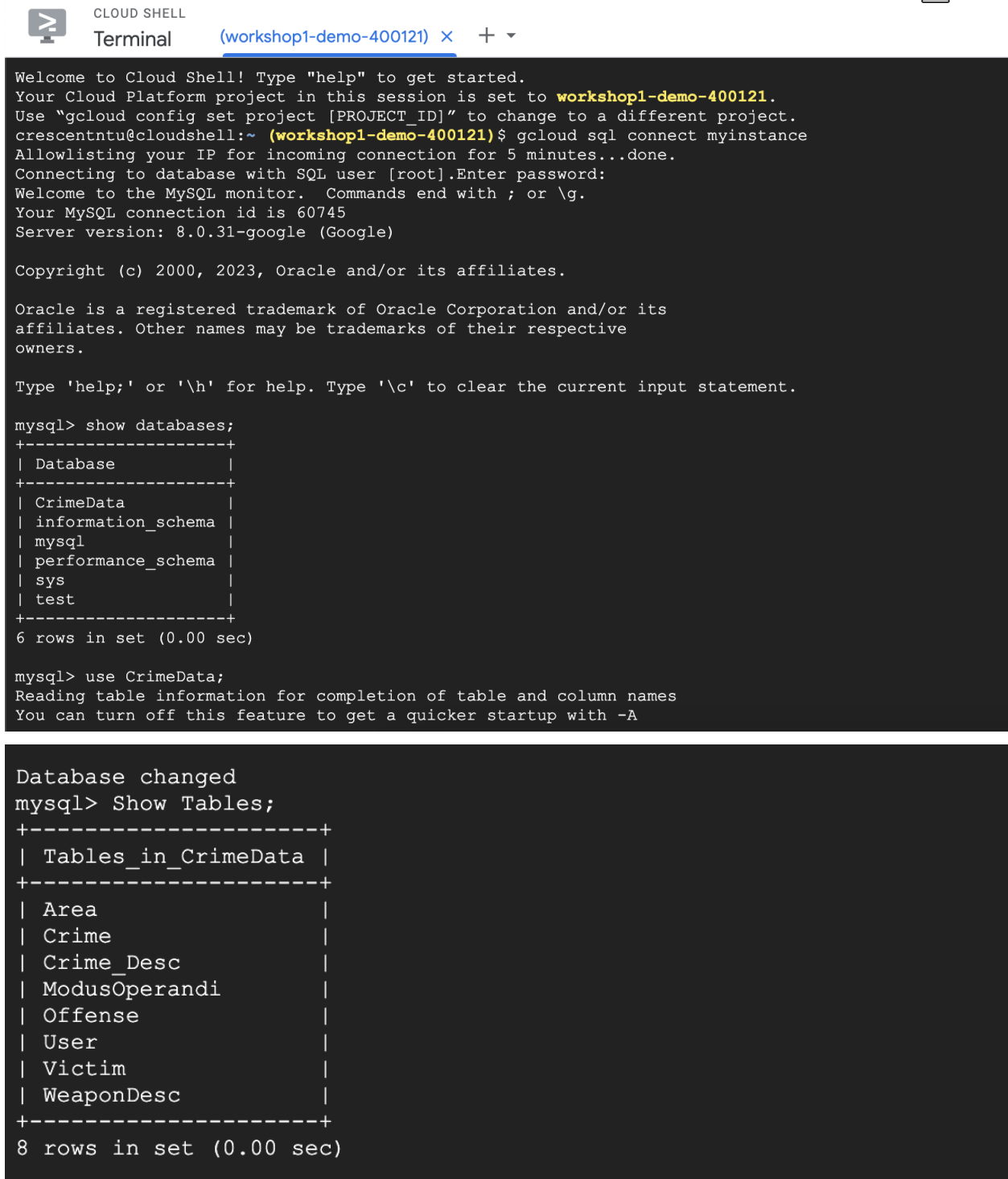


CS411 Stage3 Report

Implementing the database tables on GCP.

Here is the provided screenshot of the connection:



The screenshot shows a Cloud Shell terminal window with the following content:

```
CLOUD SHELL
Terminal (workshop1-demo-400121) x + v

Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to workshop1-demo-400121.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
crescentntu@cloudshell:~ (workshop1-demo-400121)$ gcloud sql connect myinstance
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 60745
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> show databases;
+-----+
| Database |
+-----+
| CrimeData |
| information_schema |
| mysql |
| performance_schema |
| sys |
| test |
+-----+
6 rows in set (0.00 sec)

mysql> use CrimeData;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> Show Tables;
+-----+
| Tables_in_CrimeData |
+-----+
| Area |
| Crime |
| Crime_Desc |
| ModusOperandi |
| Offense |
| User |
| Victim |
| WeaponDesc |
+-----+
8 rows in set (0.00 sec)
```

Here, we provide the DDL commands:

```
CREATE TABLE Area (  
    LON float NOT NULL,  
    LAT float NOT NULL,  
    AREA varchar(255) DEFAULT NULL,  
    Area_NAME varchar(255) DEFAULT NULL,  
    Location varchar(255) DEFAULT NULL,  
    CrossStreet varchar(255) DEFAULT NULL,  
    Rpt_Dist_No varchar(255) DEFAULT NULL,  
    PRIMARY KEY (LON,LAT)  
);
```

```
CREATE TABLE Crime (  
    Dr_ID varchar(255) NOT NULL,  
    Date_Rptd varchar(255) DEFAULT NULL,  
    Date_OCC varchar(255) DEFAULT NULL,  
    Time_OCC varchar(255) DEFAULT NULL,  
    Part1_2 varchar(255) DEFAULT NULL,  
    Crm_cd varchar(255) DEFAULT NULL,  
    PremisDesc varchar(255) DEFAULT NULL,  
    Weapon_Used_Cd varchar(255) DEFAULT NULL,  
    Judge_Status_desc` varchar(255) DEFAULT NULL,  
    Crm_cd1 varchar(255) DEFAULT NULL,  
    Crm_cd2 varchar(255) DEFAULT NULL,  
    Crm_cd3 varchar(255) DEFAULT NULL,  
    Crm_cd4 varchar(255) DEFAULT NULL,  
    LAT float DEFAULT NULL,  
    LON float DEFAULT NULL,  
    Mocode1 varchar(255) DEFAULT NULL,  
    Mocode2 varchar(255) DEFAULT NULL,  
    Mocode3 varchar(255) DEFAULT NULL,  
    UserId` varchar(255) DEFAULT NULL,  
    PRIMARY KEY (Dr_ID),  
    FOREIGN KEY (Mocode1) REFERENCES ModusOperandi (Mocodes),  
    FOREIGN KEY (Mocode2) REFERENCES ModusOperandi (Mocodes),  
    FOREIGN KEY (Mocode3) REFERENCES ModusOperandi (Mocodes),  
    FOREIGN KEY (Crm_cd) REFERENCES Crime_Desc (Crm_cd),
```

```
FOREIGN KEY (UserId) REFERENCES User (UserId),  
FOREIGN KEY (Weapon_Used_Cd) REFERENCES WeaponDesc  
(WeaponUsedCd),  
FOREIGN KEY (LON) REFERENCES Area (LON),  
FOREIGN KEY (LAT) REFERENCES Area (LAT)  
);
```

```
CREATE TABLE Crime_Desc (  
  Crm_cd varchar(255) NOT NULL,  
  Crm_cd_desc varchar(255) DEFAULT NULL,  
  PRIMARY KEY (Crm_cd)  
);
```

```
CREATE TABLE ModusOperandi (  
  Mocodes varchar(255) NOT NULL,  
  Mocodes_Desc varchar(255) DEFAULT NULL,  
  PRIMARY KEY (Mocodes)  
);
```

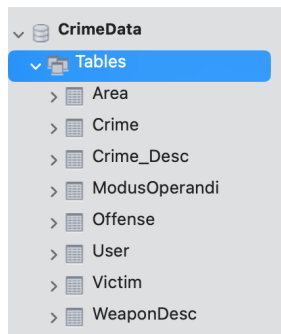
```
CREATE TABLE Offense (  
  Dr_ID varchar(255) NOT NULL,  
  VictimID varchar(255) NOT NULL,  
  VictimID2 varchar(255) DEFAULT NULL,  
  PRIMARY KEY (Dr_ID,VictimID),  
  FOREIGN KEY (Dr_ID) REFERENCES Crime (Dr_ID),  
  FOREIGN KEY (VictimID) REFERENCES Victim (VictimID),  
  FOREIGN KEY (VictimID2) REFERENCES Victim (VictimID));
```

```
CREATE TABLE User (  
  UserID varchar(20) NOT NULL,  
  UserName varchar(20) DEFAULT NULL,  
  Password varchar(20) DEFAULT NULL,  
  PRIMARY KEY (UserID)  
);
```

```
CREATE TABLE Victim (
  VictimID varchar(255) NOT NULL,
  Vict_Age int DEFAULT NULL,
  Vict_Sex varchar(255) DEFAULT NULL,
  Vict_Descent varchar(255) DEFAULT NULL,
  PRIMARY KEY (VictimID)
);
```

```
CREATE TABLE WeaponDesc (
  WeaponUsedCd varchar(255) NOT NULL,
  Weapon_Desc varchar(255) DEFAULT NULL,
  PRIMARY KEY (WeaponUsedCd)
);
```

We have 3+ tables containing more than 10K rows.



1 • `select count(*) From Crime;` 1 • `select count(*) From Offense;` 1 • `select count(*) From Victim;`

100% ▾ 27:1	100% ▾ 29:1	100% ▾ 28:1
Result Grid Filter Rows: <input type="text" value="Search"/>	Result Grid Filter Rows: <input type="text" value="Search"/>	Result Grid Filter Rows: <input type="text" value="Search"/>
count(*)	count(*)	count(*)
11966	11967	11966

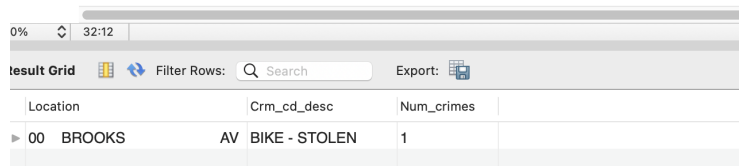
Action Output ▾					
	Time	Action	Response	Duration / Fetch Time	
✓ 1	21:25:59	show create table Crime	1 row(s) returned	0.029 sec / 0.000024...	
✓ 2	21:56:12	select * From Crime	11966 row(s) returned	0.097 sec / 0.234 sec	
✓ 3	21:57:36	select * From ModusOperandi	777 row(s) returned	0.032 sec / 0.00041 s...	
✓ 4	21:57:49	select * From Area	7806 row(s) returned	0.031 sec / 0.020 sec	
✓ 5	21:58:14	select * From Offense	11967 row(s) returned	0.030 sec / 0.0098 sec	
✓ 6	21:58:42	select * From Victim	11966 row(s) returned	0.030 sec / 0.014 sec	

Index

Query 1

```
1 • EXPLAIN ANALYZE
2 SELECT
3     A.Location,
4     CD.Crm_cd,
5     COUNT(C.Dr_ID) AS Num_Crimes
6 FROM
7     Crime C
8 JOIN
9     Area A ON C.LON = A.LON AND C.LAT = A.LAT
10 JOIN
11     Crime_Desc CD ON C.Crm_cd = CD.Crm_cd
12 Where A.Location = '00    BROOKS    AV'
13 GROUP BY
14     A.Location, CD.Crm_cd
15 ORDER BY
16     A.Location ASC, Num_Crimes ASC;
```

```
1 select
2     A.Location, CD.Crm_cd_desc, Count(C.Dr_ID) As Num_crimes
3 from Crime C
4 Join
5     Area A on C.LON = A.LON and C.LAT = A.LAT
6 Join
7     Crime_Desc CD on C.Crm_cd = CD.Crm_cd
8 where A.Location = '00    BROOKS    AV'
9 Group by
10     A.Location, CD.Crm_cd
11 order by
12     A.Location Asc,Num_Crimes Asc;
```



0% 32:12

result Grid Filter Rows: Search Export:

Location	Crm_cd_desc	Num_crimes
00 BROOKS AV	BIKE - STOLEN	1

The query itself is structured to collate the number of distinct crime types at a specific location, '00 BROOKS AV', by joining related data from the Crime, Area, and Crime_Desc tables. It groups the results by location and crime code, and then orders them to present a clear picture of crime distribution.

Initial Query Performance Without Indexes

The initial execution of the query was sluggish, primarily due to the absence of indexes, which necessitated full table scans and inefficient nested loop joins.

Enhanced Performance with Index on Area.LocationT

The implementation of an index on the LocationT column within the Area table was a game-changer, significantly accelerating the query by enabling rapid location-based filtering through index lookups.

Further Improvement with Index on Crime(lon, lat)

Subsequent optimization was achieved with a composite index on the lon and lat columns in the Crime table. This index was instrumental in speeding up the join operation with the Area table, allowing for quick matching of rows based on geographical coordinates.

Conclusion and Selected Index Design

Our final index design comprises:

An index on the Area.LocationT column to facilitate fast retrieval of area-specific records.

A composite index on the Crime.lon and Crime.lat columns to optimize the join process.

This configuration was selected due to the dramatic improvements in query execution times, which were evident in the metrics provided by the EXPLAIN ANALYZE command. The execution time was reduced from 57.058ms to 9.899ms after the first indexing, and further to 4.606ms with the second indexing.

If no improvement had been observed, it could be attributed to factors such as low distinctiveness of data (low cardinality), a high percentage of data retrieval (low selectivity), caching mechanisms, or non-ideal index utilization by the database engine.

In essence, the strategic indexing of the database has significantly optimized the query's performance, highlighting the pivotal role of indexes in database query efficiency.

Index on query

Query 1

```
SELECT
    A.Location,
    CD.Crm_cd,
    COUNT(C.Dr_ID) AS Num_Crimes
FROM
    Crime C
JOIN
    Area A ON C.LON = A.LON AND C.LAT = A.LAT
JOIN
    Crime_Desc CD ON C.Crm_cd = CD.Crm_cd
Where A.Location = '00    BROOKS                AV'
```

GROUP BY

A.Location, CD.Crm_cd

ORDER BY

A.Location ASC, Num_Crimes ASC;

Our query is selecting the number of each type of crime that happened at location '00 BROOKS AV'. The cost of the query without any index is

Before add index

- > Sort: Num_Crimes (actual time=57.058..57.058 rows=1 loops=1)
- > Table scan on <temporary> (actual time=56.994..56.995 rows=1 loops=1)
- > Aggregate using temporary table (actual time=56.990..56.990 rows=1 loops=1)
- > Nested loop inner join (cost=7930.65 rows=984) (actual time=35.113..56.919 rows=1 loops=1)
 - > Nested loop inner join (cost=4485.60 rows=9843) (actual time=0.100..27.650 rows=11966 loops=1)
 - > Filter: ((C.Crm_cd is not null) and (C.LON is not null) and (C.LAT is not null)) (cost=1040.55 rows=9843) (actual time=0.078..10.621 rows=11966 loops=1)
 - > Table scan on C (cost=1040.55 rows=9843) (actual time=0.076..8.287 rows=11966 loops=1)
 - > Single-row covering index lookup on CD using PRIMARY (Crm_cd=C.Crm_cd) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=11966)
 - > Filter: (A.Location = '00 BROOKS AV') (cost=0.25 rows=0.1) (actual time=0.002..0.002 rows=0 loops=11966)
 - > Single-row index lookup on A using PRIMARY (LON=C.LON, LAT=C.LAT) (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=11966)

After create index on (LocationT) Area

- > Sort: Num_Crimes (actual time=9.899..9.899 rows=1 loops=1)
- > Table scan on <temporary> (actual time=9.879..9.879 rows=1 loops=1)
- > Aggregate using temporary table (actual time=9.876..9.876 rows=1 loops=1)
- > Nested loop inner join (cost=92.09 rows=1) (actual time=6.637..9.846 rows=1 loops=1)
 - > Inner hash join (C.LAT = A.LAT), (C.LON = A.LON) (cost=67.38 rows=1) (actual time=6.609..9.818 rows=1 loops=1)
 - > Filter: (C.Crm_cd is not null) (cost=66.31 rows=98) (actual time=0.028..8.069 rows=11966 loops=1)

-> Table scan on C (cost=66.31 rows=9843) (actual time=0.027..6.885 rows=11966 loops=1)
 -> Hash
 -> Covering index lookup on A using idx_location (Location='00 BROOKS AV') (cost=1.07 rows=1) (actual time=0.036..0.040 rows=1 loops=1)
 -> Single-row covering index lookup on CD using PRIMARY (Crm_cd=C.Crm_cd) (cost=0.25 rows=1) (actual time=0.025..0.025 rows=1 loops=1)

Index on (lot, lat) Crime

-> Sort: Num_Crimes (actual time=4.606..4.606 rows=1 loops=1)
 -> Table scan on <temporary> (actual time=4.583..4.583 rows=1 loops=1)
 -> Aggregate using temporary table (actual time=4.580..4.580 rows=1 loops=1)
 -> Nested loop inner join (cost=1519.94 rows=1002) (actual time=0.818..4.537 rows=1 loops=1)
 -> Nested loop inner join (cost=1169.34 rows=1002) (actual time=0.801..4.519 rows=1 loops=1)
 -> Filter: (A.Location = '00 BROOKS AV') (cost=818.75 rows=795) (actual time=0.758..4.474 rows=1 loops=1)
 -> Table scan on A (cost=818.75 rows=7945) (actual time=0.151..3.343 rows=7806 loops=1)
 -> Filter: (C.Crm_cd is not null) (cost=0.32 rows=1) (actual time=0.041..0.043 rows=1 loops=1)
 -> Index lookup on C using idx_lon_lat (LON=A.LON, LAT=A.LAT) (cost=0.32 rows=1) (actual time=0.039..0.041 rows=1 loops=1)
 -> Single-row covering index lookup on CD using PRIMARY (Crm_cd=C.Crm_cd) (cost=0.25 rows=1) (actual time=0.016..0.016 rows=1 loops=1)

Index on (Crm_cd) Crime_desc

CREATE INDEX idx_dr_id_crm_cd ON Crime_Desc(Crm_cd);

-> Sort: Num_Crimes (actual time=51.416..51.416 rows=1 loops=1)
 -> Table scan on <temporary> (actual time=51.393..51.393 rows=1 loops=1)
 -> Aggregate using temporary table (actual time=51.387..51.387 rows=1 loops=1)
 -> Nested loop inner join (cost=7930.65 rows=984) (actual time=33.666..51.353 rows=1 loops=1)
 -> Nested loop inner join (cost=4485.60 rows=9843) (actual time=0.067..25.015 rows=11966 loops=1)
 -> Filter: ((C.Crm_cd is not null) and (C.LON is not null) and (C.LAT is not null)) (cost=1040.55 rows=9843) (actual time=0.048..9.544 rows=11966 loops=1)
 -> Table scan on C (cost=1040.55 rows=9843) (actual time=0.046..7.292 rows=11966 loops=1)

-> Single-row covering index lookup on CD using PRIMARY (Crm_cd=C.Crm_cd)
(cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=11966)
-> Filter: (A.Location = '00 BROOKS AV') (cost=0.25 rows=0.1)
(actual time=0.002..0.002 rows=0 loops=11966)
-> Single-row index lookup on A using PRIMARY (LON=C.LON, LAT=C.LAT)
(cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=11966)

Index on both Crime(LON, LAT) and Area(Location);

-> Sort: Num_Crimes (actual time=0.100..0.100 rows=1 loops=1)
-> Table scan on <temporary> (actual time=0.087..0.088 rows=1 loops=1)
-> Aggregate using temporary table (actual time=0.086..0.086 rows=1 loops=1)
-> Nested loop inner join (cost=1.95 rows=1) (actual time=0.058..0.062 rows=1
loops=1)
-> Nested loop inner join (cost=1.51 rows=1) (actual time=0.049..0.052 rows=1
loops=1)
-> Covering index lookup on A using idx_location (Location='00 BROOKS
AV') (cost=1.07 rows=1) (actual time=0.020..0.022 rows=1 loops=1)
-> Filter: (C.Crm_cd is not null) (cost=0.44 rows=1) (actual time=0.027..0.029
rows=1 loops=1)
-> Index lookup on C using idx_lon_lat (LON=A.LON, LAT=A.LAT) (cost=0.44
rows=1) (actual time=0.026..0.028 rows=1 loops=1)
-> Single-row covering index lookup on CD using PRIMARY (Crm_cd=C.Crm_cd)
(cost=0.33 rows=1) (actual time=0.009..0.009 rows=1 loops=1)

Query 2

```

• Select Count(*), Date_OCC, Judge_Status_desc From Crime c
  natural join Crime_Desc d
  Where Crm_cd_desc like "%RAPE%"
  Group by Date_OCC, Judge_Status_desc
  Order by Count(*) Desc
  LIMIT 15;

```



```

| -> Limit: 15 row(s) (actual time=0.608..0.611 rows=15 loops=1)
    -> Sort: Count(*) DESC, limit input to 15 row(s) per chunk (actual time=0.608..0.609 rows=15
loops=1)
        -> Table scan on <temporary> (actual time=0.575..0.581 rows=31 loops=1)
            -> Aggregate using temporary table (actual time=0.573..0.573 rows=31 loops=1)
                -> Nested loop inner join (cost=617.27 rows=1725) (actual time=0.266..0.482
rows=44 loops=1)
                    -> Filter: (d.Crm_cd_desc like '%RAPE%') (cost=13.45 rows=15) (actual
time=0.077..0.175 rows=2 loops=1)
                        -> Table scan on d (cost=13.45 rows=132) (actual time=0.063..0.102 rows=132
loops=1)
                            -> Index lookup on c using Crm_cd (Crm_cd=d.Crm_cd) (cost=30.21 rows=118)
(actual time=0.098..0.150 rows=22 loops=2)
|
+-----+
|
|
|
|
|
|
|
|
+-----+
1 row in set (0.01 sec)

```

After adding index on Crm_cd_desc
'mysql> Create index crmdesc_idx on Crime_Desc(Crm_cd_desc);
Query OK, 0 rows affected (0.04 sec)
Records: 0 Duplicates: 0 Warnings: 0'
The cost is reduced.

```

mysql> Explain Analyze Select Count(*), Date_OCC, Judge_Status_desc From Crime c natural
join Crime_Desc d Where Crm_cd_desc like "%RAPE%" Group by Date_OCC,
Judge_Status_desc Order by Count(*) Desc LIMIT 15;
+-----+
|
|
|
|
|
|
|
|
+-----+
| EXPLAIN
|

```

```

+-----+
| -> Limit: 15 row(s) (actual time=0.501..0.504 rows=15 loops=1)
  -> Sort: Count(*) DESC, limit input to 15 row(s) per chunk (actual time=0.500..0.502 rows=15
loops=1)
    -> Table scan on <temporary> (actual time=0.447..0.454 rows=31 loops=1)
      -> Aggregate using temporary table (actual time=0.446..0.446 rows=31 loops=1)
        -> Nested loop inner join (cost=617.27 rows=1725) (actual time=0.159..0.370
rows=44 loops=1)
          -> Filter: (d.Crm_cd_desc like '%RAPE%') (cost=13.45 rows=15) (actual
time=0.114..0.145 rows=2 loops=1)
            -> Covering index scan on d using crmdesc_idx (cost=13.45 rows=132) (actual
time=0.046..0.078 rows=132 loops=1)
              -> Index lookup on c using Crm_cd (Crm_cd=d.Crm_cd) (cost=30.21 rows=118)
(actual time=0.058..0.109 rows=22 loops=2)
|
+-----+
1 row in set (0.01 sec)

```

After dropping index on Crm_cd_desc and adding index on Judge_Status:

```
mysql> drop index crmdesc_idx on Crime_Desc;
```

Query OK, 0 rows affected (0.03 sec)

Records: 0 Duplicates: 0 Warnings: 0

```
mysql> Create index status_idx on Crime(Judge_Status_desc);
```

Query OK, 0 rows affected (0.25 sec)

Records: 0 Duplicates: 0 Warnings: 0

The cost is further reduced, indicating that creating an index on Judge_Status_desc is a better choice than on Crm_cd_desc.

[illegible][illegible]

- > Table scan on <temporary> (actual time=0.447..0.456 rows=31 loops=1)
- > Aggregate using temporary table (actual time=0.446..0.446 rows=31 loops=1)
- > Nested loop inner join (cost=617.27 rows=1725) (actual time=0.165..0.370 rows=44 loops=1)

```
-> Table scan on d (cost=13.45 rows=132) (actual time=0.055..0.090 rows=132 loops=1)
```

[illegible]

Let's now create a composite index on Crime:

```
CREATE INDEX idx_composite ON Crime (Date_OCC, Judge_Status_desc);
Query OK, 0 rows affected (0.27 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

The performance is not getting better, possible reason could be the index is somewhat larger and increases I/O and memory consumption, which can slow down the query performance.

```
mysql> Explain Analyze Select Count(*), Date_OCC, Judge_Status_desc From Crime c natural
join Crime_Desc d Where Crm_cd_desc like "%RAPE%" Group by Date_OCC,
Judge_Status_desc Order by Count(*) Desc LIMIT 15;
```

```
+-----+
|
|
+-----+
| EXPLAIN
|
+-----+
|
+-----+
| -> Limit: 15 row(s) (actual time=0.538..0.541 rows=15 loops=1)
  -> Sort: Count(*) DESC, limit input to 15 row(s) per chunk (actual time=0.537..0.539 rows=15
loops=1)
    -> Table scan on <temporary> (actual time=0.497..0.511 rows=31 loops=1)
      -> Aggregate using temporary table (actual time=0.496..0.496 rows=31 loops=1)
        -> Nested loop inner join (cost=617.27 rows=1725) (actual time=0.206..0.417
rows=44 loops=1)
          -> Filter: (d.Crm_cd_desc like '%RAPE%') (cost=13.45 rows=15) (actual
time=0.087..0.180 rows=2 loops=1)
            -> Table scan on d (cost=13.45 rows=132) (actual time=0.078..0.113 rows=132
loops=1)
              -> Index lookup on c using Crm_cd (Crm_cd=d.Crm_cd) (cost=30.21 rows=118)
(actual time=0.063..0.115 rows=22 loops=2)
|
+-----+
```

1 row in set (0.00 sec)

I choose indices on Crime_Desc(Crm_cd_desc), Crime(Judge_Status_desc), Crime(Date_OCC, Judge_Status_desc) because these attributes appeared in the Where clause or Group By Clause. And I want to find if composite two attributes together as an index can help to boost the performance of query execution.

The reason why adding indices to the query does not lead to significant performance improvements may be due to the fact that the selectivity of the indexed column matters. If the filtering condition does not significantly reduce the number of rows, the index may not provide substantial benefits. Because the condition using Crm_cd_desc like "%RAPE%" matches a large portion of the table, the index may not be very selective.

Also Crm_cd_desc is in the table of Crime_Desc and it is relatively a small table. Adding indexes to small tables may not lead to significant performance improvements in terms of query execution time, especially for tables that are already small and easily fit into memory. In some cases, the overhead of maintaining indexes on small tables may even slightly degrade performance.

And for the indices on the large table attributes, although the running time is a bit faster, I cannot see their usage in the "explain analyze", indicating that the optimizer might not choose to use that index at all.