STAGE 3

I. DDL and Insert Commands Used to Create the Tables

We created a table to hold all the Crime Data. We inserted into this table, all values from the Crime in Los Angeles Data from 2020 to Present dataset. We then used these real values for our tables.

```
CREATE TABLE Complete_Table
(dr_no INT NOT NULL,
date_rptd VARCHAR(100),
date_occ VARCHAR(100),
time_occ INT,
area_code INT,
area_name VARCHAR(50),
rpt_dist_no INT,
part INT,
crm_cd INT,
crm_cd_desc VARCHAR(100),
mocodes CHAR(40),
vict_age INT,
vict_sex CHAR(1),
vict_descent CHAR(1),
premis_cd INT,
premis_desc CHAR(100),
weapon_used_cd INT,
weapon_desc VARCHAR(100),
status VARCHAR(2),
status_desc CHAR(50),
crime_cd_1 INT,
crime_cd_2 INT,
crime_cd_3 INT,
crime_cd_4 INT,
location VARCHAR(100),
cross_street CHAR(50),
latitude REAL,
longitude REAL
);
```

The 4 main tables we created are : Crime, Location, Weapon, Premise. We also created a table for the many to many relation Use between crime and weapon called Use_table. The many to one relation “where” between Crime and Location is dealt with by adding the primary keys of Location to Crime. We used real values from the Crime in Los Angeles Data from 2020 to Present dataset to insert into our tables.
Below are the DDL and insertion commands for each of these tables.

```
CREATE TABLE Crime
(
     record_no INT NOT NULL PRIMARY KEY,
     crime_code INT,
     crime_code_description VARCHAR(100),
     crime_code_1 INT,
     crime_code_2 INT,
     crime_code_3 INT,
     crime_code_4 INT,
     mocodes CHAR(40),
     longitude REAL,
     latitude REAL,
     rpt_dist_no INT,
     address VARCHAR(100),
     FOREIGN KEY (longitude, latitude, rpt_dist_no, address)
REFERENCES Location(longitude, latitude, rpt_dist_no, address)
ON
     DELETE SET NULL ON UPDATE CASCADE
);

INSERT INTO Crime
SELECT DISTINCT dr_no, crm_cd, crm_cd_desc, crime_cd_1,
crime_cd_2, crime_cd_3, crime_cd_4, mocodes, longitude,
latitude, rpt_dist_no, location
FROM Complete_Table
WHERE premis_cd <> 0;
```

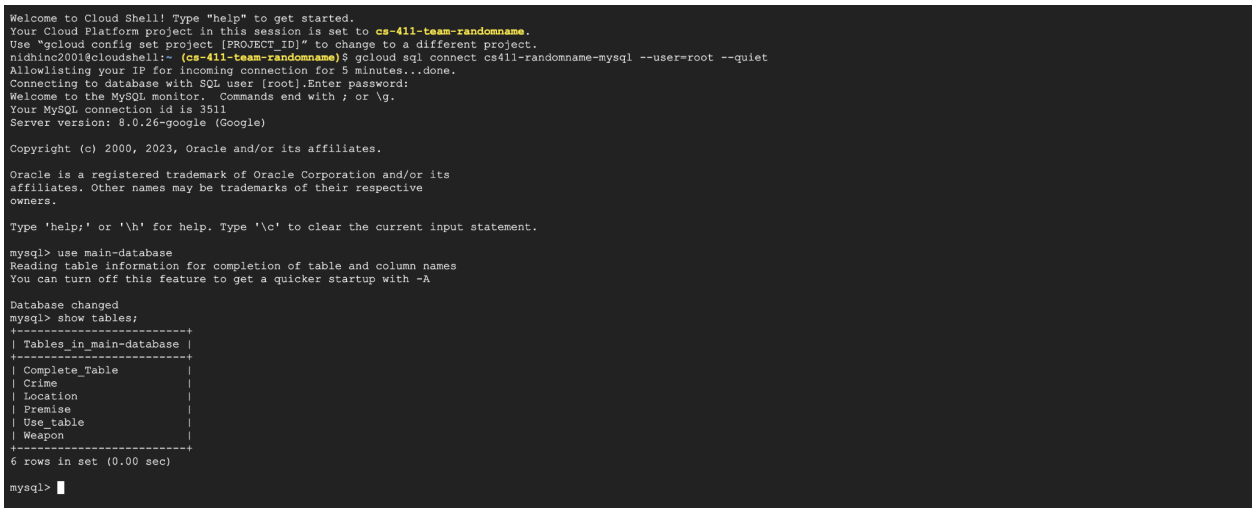
For table Location, we realized that latitude and longitude do not suffice as primary keys alone and so we added rpt_dist_no and address as primary keys.

```
CREATE TABLE Location
(
     longitude REAL NOT NULL,
     latitude REAL NOT NULL,
```

```
     area_code INT,
     area_name VARCHAR(50),
     rpt_dist_no INT,
     address VARCHAR(100),
     PRIMARY KEY (longitude, latitude, rpt_dist_no, address)
);

INSERT into Location
SELECT Distinct longitude, latitude, area_code, area_name,
rpt_dist_no, location from Complete_Table
WHERE premis_cd <> 0;

CREATE TABLE Weapon
(
     code INT NOT NULL,
     description VARCHAR(100),
     PRIMARY KEY (code, description)
);

INSERT INTO Weapon
SELECT DISTINCT weapon_used_cd, weapon_desc
FROM Complete_Table
WHERE premis_cd <> 0;

CREATE TABLE Use_table
(
     record_no INT NOT NULL,
     code INT NOT NULL,
     weapon_desc VARCHAR(100),
     FOREIGN KEY (record_no) REFERENCES Crime(record_no),
     FOREIGN KEY (code, weapon_desc) REFERENCES Weapon(code,
description),
     PRIMARY KEY (record_no, code, weapon_desc)
);

INSERT INTO Use_table
SELECT DISTINCT dr_no, weapon_used_cd, weapon_desc
FROM Complete_Table
WHERE premis_cd <> 0;
```

```
CREATE TABLE Premise(code INT NOT NULL AUTO_INCREMENT,
description CHAR(100), PRIMARY KEY(code));


INSERT INTO Premise
SELECT DISTINCT premis_cd, premis_desc
FROM Complete_Table;
```

II. Establishing Connection with the Database and Proof of Tables



Tables Crime, Location, Use_table have at least 1000 rows.

```
mysql> select count(*) from Crime;
+----------+
| count(*) |
+----------+
|   317849 |
+----------+
1 row in set (0.03 sec)

mysql> select count(*) from Location;
+----------+
| count(*) |
+----------+
|   103336 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from Use_table;
+----------+
| count(*) |
+----------+
|   317849 |
+----------+
1 row in set (0.02 sec)
```

III. Advanced Query

1) Find the number of crimes that have happened till date at all locations:

```
SELECT longitude, latitude, Count(*)
FROM Crime NATURAL JOIN Location
GROUP BY longitude, latitude;
```

```
mysql> SELECT longitude, latitude, Count(*)  FROM Crime NATURAL JOIN Location GROUP BY longitude, latitude LIMIT 15;
+-----------+----------+----------+
| longitude | latitude | Count(*) |
+-----------+----------+----------+
| -118.6676 |    34.18 |        1 |
| -118.6673 |  34.1775 |        2 |
| -118.6672 |  34.1824 |        1 |
| -118.6665 |  34.1839 |        1 |
| -118.6661 |  34.1824 |        3 |
| -118.6652 |  34.1774 |        3 |
| -118.6634 |  34.1801 |        1 |
| -118.6629 |  34.2002 |        3 |
| -118.6626 |  34.1815 |        1 |
| -118.6625 |  34.1776 |        1 |
| -118.6623 |  34.1796 |        1 |
| -118.6623 |  34.1859 |        1 |
| -118.6616 |  34.2017 |        1 |
| -118.6612 |  34.1815 |        3 |
| -118.6611 |  34.1982 |        1 |
+-----------+----------+----------+
15 rows in set (0.00 sec)
```

2) Find all the crimes that used weapon Strong-Arm (Weapon Code 400) AND weapon Rock/Object Thrown (Weapon Code 306)

```
(SELECT * FROM Crime WHERE EXISTS(SELECT * FROM Use_table
WHERE code = 306 AND record_no = Crime.record_no)
UNION
SELECT * FROM Crime WHERE EXISTS(SELECT * FROM Use_table
WHERE code = 400 AND record_no = Crime.record_no));
```

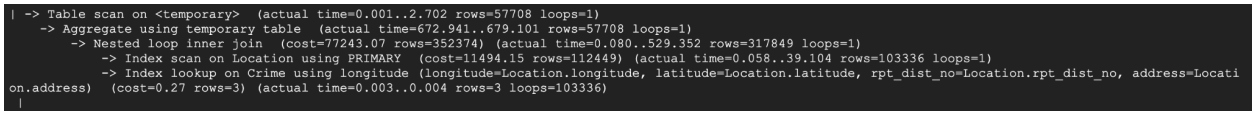
```
+-----------+----------+-------------+----------------------------------------+
| record_no | crime_code | crime_code_description                              | crime_code_1 | crime_code_2 | crime_code_3 | crime_code_4 | mocodes
| longitude | latitude | rpt_dist_no | address                                |
+-----------+----------+-------------+----------------------------------------+
| 200100509 |      330 | BURGLARY FROM VEHICLE                               |          330 |            0 |            0 |            0 | 1822 1414 0344 1307
| -118.2648 |  34.0359 |         192 | 15TH                                   |
| 200100786 |      930 | CRIMINAL THREATS - NO WEAPON DISPLAYED              |          930 |          998 |            0 |            0 | 0421 1822 1402 1414
| -118.2507 |  34.0481 |         153 | 500 S  BROADWAY                        |
| 200100799 |      310 | BURGLARY                                            |          310 |          998 |            0 |            0 | 0906 0329 1402 0345 1420 2004 1609
| -118.2335 |  34.0387 |         159 | 2000 E  7TH                         ST |
| 200100811 |      647 | THROWING OBJECT AT MOVING VEHICLE                   |          647 |          998 |            0 |            0 | 1414 1307 0447 1300 1310 1402
| -118.2585 |  34.0509 |         151 | 6TH                                    |
| 200101078 |      946 | OTHER MISCELLANEOUS CRIME                           |          946 |          998 |            0 |            0 | 1501 2004 1402 0417 0400 1822 0329
| -118.2426 |  34.0549 |         123 | 200 W  TEMPLE                       ST |
| 200101132 |      740 | VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS) |          740 |            0 |            0 |            0 | 1414 0329 1822 1609
| -118.2425 |  34.0532 |         124 | 200 N  MAIN                         ST |
| 200104336 |      647 | THROWING OBJECT AT MOVING VEHICLE                   |          647 |            0 |            0 |            0 | 1300 0447 0329 1822
| -118.2621 |  34.0377 |         185 | HILL                                   |
| 200105409 |      210 | ROBBERY                                             |          210 |            0 |            0 |            0 | 0416 2004 1822 1414 0945 1266 0344
| -118.2447 |  34.0464 |         138 | WALL                             ST    |
| 200106083 |      626 | INTIMATE PARTNER - SIMPLE ASSAULT                   |          626 |            0 |            0 |            0 | 2000 1813 0447
| -118.2507 |  34.0446 |         164 | 100 W  7TH                          ST |
| 200106345 |      230 | ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT      |          230 |            0 |            0 |            0 | 1414 1822 2004 1310 1307 0329 0447
| -118.2437 |   34.057 |         112 | HOPE                                   |
| 200106909 |      740 | VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS) |          740 |          998 |            0 |            0 | 0329 1402 2004
|  -118.244 |  34.0416 |         157 | CENTRAL                          AV    |
| 200107466 |      230 | ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT      |          230 |            0 |            0 |            0 | 2004 0447 1822 0355 1414
| -118.2383 |  34.0623 |         111 | 800 N  BROADWAY                        |
| 200109022 |      624 | BATTERY - SIMPLE ASSAULT                            |          624 |            0 |            0 |            0 | 0447 0416 1822 1307
| -118.2422 |  34.0571 |         111 | ARCADIA                          ST    |
| 200109702 |      210 | ROBBERY                                             |          210 |            0 |            0 |            0 | 2003 1414 1822 2004 0316
| -118.2627 |   34.038 |         182 | PICO                             BL    |
| 200109854 |      310 | BURGLARY                                            |          310 |            0 |            0 |            0 | 1609 0344 1822 0324 0329 1414
| -118.2391 |  34.0492 |         128 | 100 S  CENTRAL                      AV |
+-----------+----------+-------------+----------------------------------------+
15 rows in set (0.00 sec)
```

IV. Indexing Analysis

```
EXPLAIN ANALYZE
SELECT longitude, latitude, Count(*)
FROM Crime NATURAL JOIN Location
```

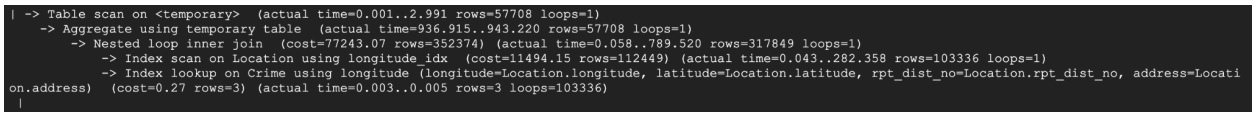
```
GROUP BY longitude, latitude;
```

Before Indexing



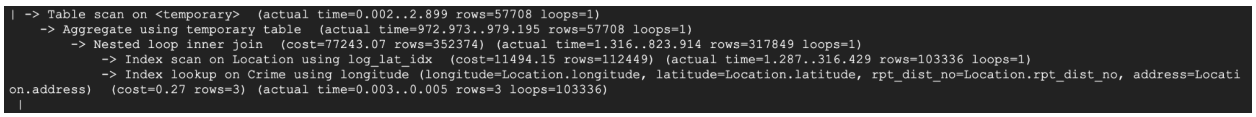
```
| -> Table scan on <temporary>  (actual time=0.001..2.702 rows=57708 loops=1)
    -> Aggregate using temporary table  (actual time=672.941..679.101 rows=57708 loops=1)
        -> Nested loop inner join  (cost=77243.07 rows=352374) (actual time=0.080..529.352 rows=317849 loops=1)
            -> Index scan on Location using PRIMARY  (cost=11494.15 rows=112449) (actual time=0.058..39.104 rows=103336 loops=1)
            -> Index lookup on Crime using longitude (longitude=Location.longitude, latitude=Location.latitude, rpt_dist_no=Location.rpt_dist_no, address=Locati
on.address)  (cost=0.27 rows=3) (actual time=0.003..0.004 rows=3 loops=103336)
 |
```

After Indexing longitude



```
| -> Table scan on <temporary>  (actual time=0.001..2.991 rows=57708 loops=1)
    -> Aggregate using temporary table  (actual time=936.915..943.220 rows=57708 loops=1)
        -> Nested loop inner join  (cost=77243.07 rows=352374) (actual time=0.058..789.520 rows=317849 loops=1)
            -> Index scan on Location using longitude_idx  (cost=11494.15 rows=112449) (actual time=0.043..282.358 rows=103336 loops=1)
            -> Index lookup on Crime using longitude (longitude=Location.longitude, latitude=Location.latitude, rpt_dist_no=Location.rpt_dist_no, address=Locati
on.address)  (cost=0.27 rows=3) (actual time=0.003..0.005 rows=3 loops=103336)
 |
```

After Indexing (latitude, longitude)



```
| -> Table scan on <temporary>  (actual time=0.002..2.899 rows=57708 loops=1)
    -> Aggregate using temporary table  (actual time=972.973..979.195 rows=57708 loops=1)
        -> Nested loop inner join  (cost=77243.07 rows=352374) (actual time=1.316..823.914 rows=317849 loops=1)
            -> Index scan on Location using log_lat_idx  (cost=11494.15 rows=112449) (actual time=1.287..316.429 rows=103336 loops=1)
            -> Index lookup on Crime using longitude (longitude=Location.longitude, latitude=Location.latitude, rpt_dist_no=Location.rpt_dist_no, address=Locati
on.address)  (cost=0.27 rows=3) (actual time=0.003..0.005 rows=3 loops=103336)
 |
```

Index Selection Report

We did not choose any index design for this advanced query because none of them improve the performance of our query. We think that indexing did not improve this query because for `Count`, the program had to scan through each row in the join table to get the aggregated count. Thus, using this query without indexing is sufficient to achieve an optimal query performance.

```
(SELECT * FROM Crime WHERE EXISTS(SELECT * FROM Use_table WHERE
code = 306 AND record_no = Crime.record_no) UNION
SELECT * FROM Crime WHERE EXISTS(SELECT * FROM Use_table WHERE
code = 400 AND record_no = Crime.record_no));
```

Before Indexing

```
+------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Table scan on <union temporary>  (cost=0.01..1561.31 rows=124705) (actual time=0.002..12.747 rows=63220 loops=1)
    -> Union materialize with deduplication  (cost=53673.66..55234.96 rows=124705) (actual time=313.060..329.425 rows=63220 loops=1)
        -> Nested loop inner join  (cost=364.72 rows=1095) (actual time=0.513..4.647 rows=1095 loops=1)
            -> Table scan on <subquery2>  (cost=0.01..16.19 rows=1095) (actual time=0.001..0.092 rows=1095 loops=1)
                -> Materialize with deduplication  (cost=238.80..254.97 rows=1095) (actual time=0.498..0.658 rows=1095 loops=1)
                    -> Index lookup on Use_table using code (code=306)  (cost=129.29 rows=1095) (actual time=0.038..0.283 rows=1095 loops=1)
            -> Single-row index lookup on Crime using PRIMARY (record_no=`<subquery2>`.record_no)  (cost=0.35 rows=1) (actual time=0.003..0.003 rows=1 loops=1095)
        -> Nested loop inner join  (cost=40838.43 rows=123610) (actual time=36.198..158.330 rows=62125 loops=1)
            -> Table scan on <subquery4>  (cost=0.01..1547.62 rows=123610) (actual time=0.002..5.911 rows=62125 loops=1)
                -> Materialize with deduplication  (cost=26929.57..28477.18 rows=123610) (actual time=36.169..46.405 rows=62125 loops=1)
                    -> Index lookup on Use_table using code (code=400)  (cost=14568.55 rows=123610) (actual time=0.043..16.570 rows=62125 loops=1)
            -> Single-row index lookup on Crime using PRIMARY (record_no=`<subquery4>`.record_no)  (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=62125)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------
```

After Indexing Code

```
-------------------------------------------------------------------------------------+
| -> Table scan on <union temporary>  (cost=0.01..1533.84 rows=122507) (actual time=0.003..12.964 rows=63220 loops=1)
    -> Union materialize with deduplication  (cost=72467.58..74001.41 rows=122507) (actual time=276.404..293.069 rows=63220 loops=1)
        -> Nested loop inner join  (cost=534.27 rows=1095) (actual time=0.055..2.884 rows=1095 loops=1)
            -> Remove duplicates from input sorted on code_idx  (cost=151.02 rows=1095) (actual time=0.042..0.449 rows=1095 loops=1)
                -> Index lookup on Use_table using code_idx (code=306)  (cost=151.02 rows=1095) (actual time=0.040..0.337 rows=1095 loops=1)
            -> Single-row index lookup on Crime using PRIMARY (record_no=Use_table.record_no)  (cost=273.85 rows=1) (actual time=0.002..0.002 rows=1 loops=1095)
        -> Nested loop inner join  (cost=59682.60 rows=121412) (actual time=0.051..129.333 rows=62125 loops=1)
            -> Remove duplicates from input sorted on code_idx  (cost=16638.90 rows=121412) (actual time=0.042..26.010 rows=62125 loops=1)
                -> Index lookup on Use_table using code_idx (code=400)  (cost=16638.90 rows=121412) (actual time=0.042..19.557 rows=62125 loops=1)
            -> Single-row index lookup on Crime using PRIMARY (record_no=Use_table.record_no)  (cost=30902.60 rows=1) (actual time=0.001..0.001 rows=1 loops=62125)
 |
```

We wanted to index on the field code, because we specify a condition on that field in every single one of our subqueries, and if that time could be cut down (even a little bit), then we would expect to see massive performance gains on our advanced query. However, no significant difference was found before and after creating an index on the code of the weapon. Upon looking further into this, we realized that there was already a BTree index on the code column of the weapon table. This was because the code is a part of the primary key in the table, so the database management system automatically created an index for queries to run fast on this field.