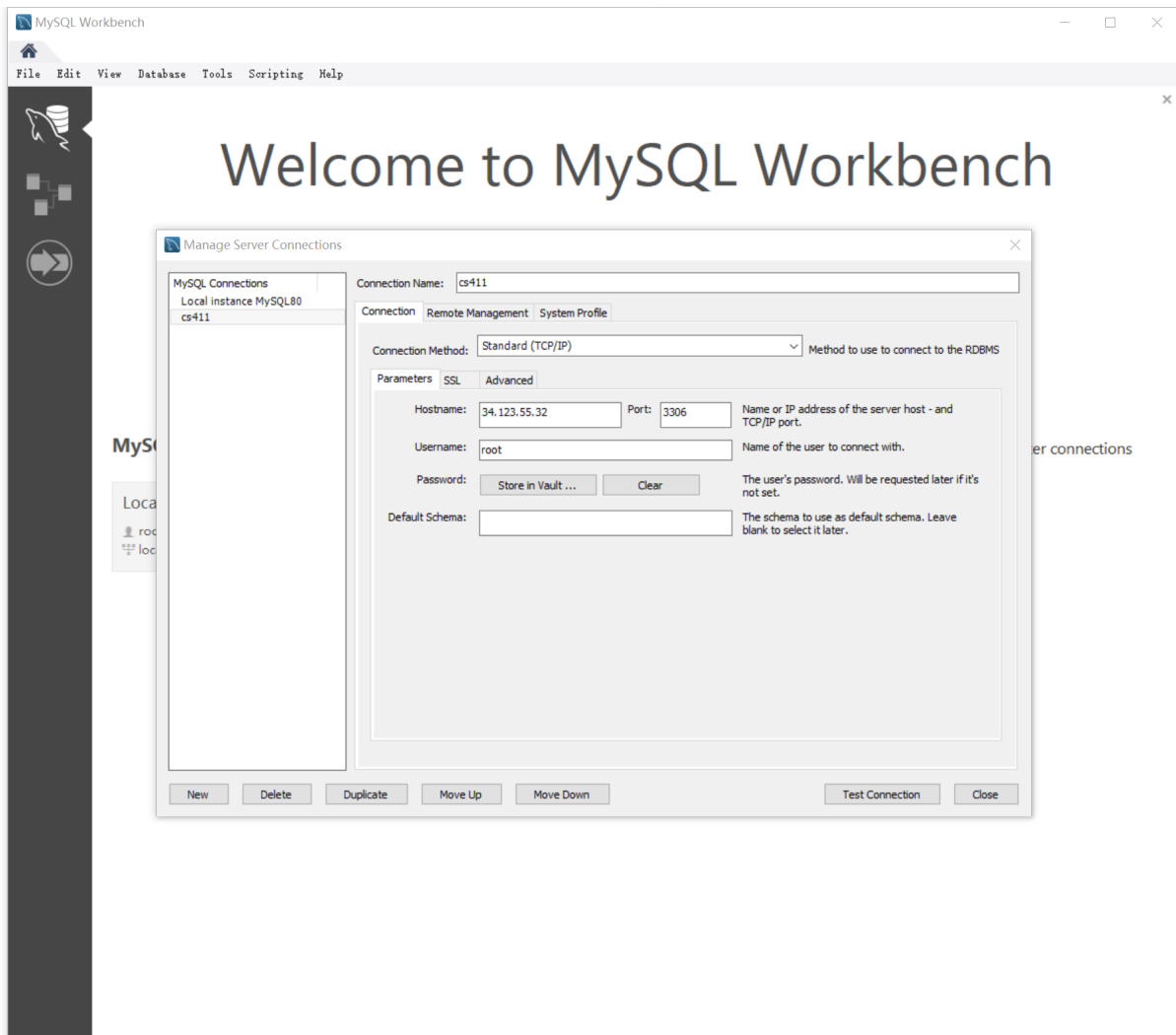
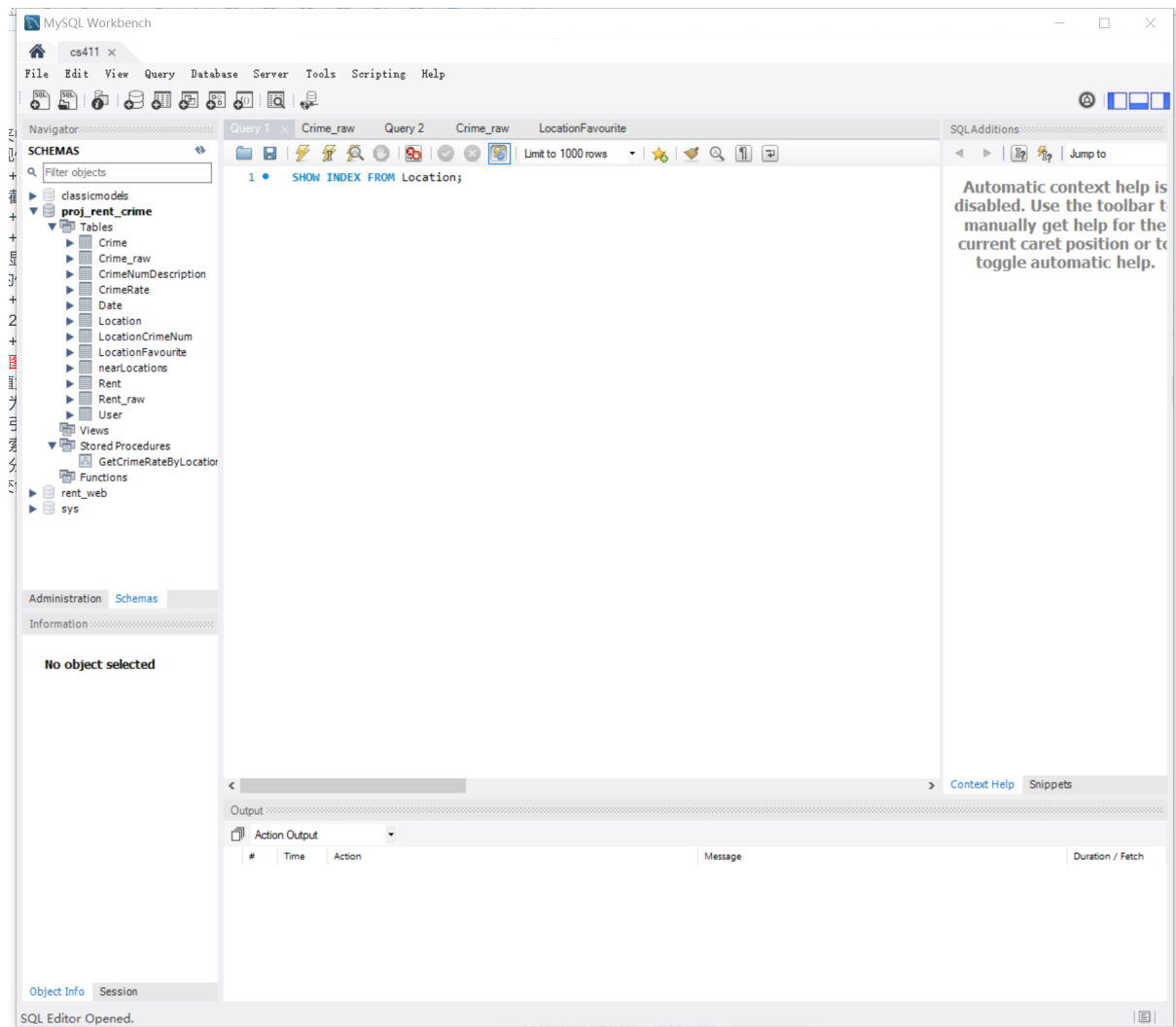


1.

+3% for implementing the database tables locally or on GCP, you should provide a screenshot of the connection (i.e. showing your terminal information)





2.

+2.5% for providing the DDL commands for your tables. (-0.5% for each mistake)

```
CREATE TABLE Crime(
    DR_NO INT PRIMARY KEY,
    Date_Rptd DATE,
    Time_OCC INT,
    AREA INT,
    AREA_Name VARCHAR(255),
    Rpt_Dist_No INT,
```

```

    Crm_Cd INT,
    Crm_Cd_Desc VARCHAR(255),
    Mocodes INT,
    Vict_Age INT,
    Vlct_Sex CHAR(1),
    Vict_Descent CHAR(1),
    Premis_Cd INT,
    Premis_Desc VARCHAR(255),
    Weapon_Used_Cd INT,
    Weapon_Desc VARCHAR(255),
    Status CHAR(2),
    Status_Desc VARCHAR(255),
    Crm_Cd1 INT,
    Crm_Cd2 INT,
    Crm_Cd3 INT,
    Crm_Cd4 INT,
    Location VARCHAR(255),
    Cross_Street VARCHAR(255),
    LAT REAL,
    LON REAL,
    FOREIGN KEY (LAT, LON) REFERENCES Location(LAT, LON)
);

```

```

CREATE TABLE Rent(
    Row_ID VARCHAR(255) PRIMARY KEY,
    Year INT,
    Amount INT,
    Tract VARCHAR(255),
    Tract_Number INT,
    Neighborhood VARCHAR(255),
    GEOID VARCHAR(255),
    LAT REAL,
    LON REAL,
    Date DATE,
    FOREIGN KEY (LAT, LON) REFERENCES Location(LAT, LON)
);

```

```

CREATE TABLE Date(
    Year INT,
    Date_OCC DATE,
    PRIMARY KEY (Year,Date_OCC)
);

```

```
CREATE TABLE User(  
    UserID INT PRIMARY KEY,  
    UnderName VARCHAR(255),  
    Password VARCHAR(255),  
    Email VARCHAR(255)  
);
```

```
CREATE TABLE CrimeRate(  
    RateNum INT PRIMARY KEY,  
    rateDescroption VARCHAR(255)  
);
```

```
CREATE TABLE Location(  
    LAT REAL,  
    LON REAL,  
    RateNum INT,  
    PRIMARY KEY (LAT,LON),  
    FOREIGN KEY (RateNum) REFERENCES CrimeRate(RateNum)  
);
```

```
CREATE TABLE LocationFavourite(  
    AddTime INT,  
    Category VARCHAR(255),  
    AddDescription VARCHAR(225),  
    UserID INT,  
    LAT REAL,  
    LON REAL,  
    PRIMARY KEY (UserID,LAT,LON),  
    FOREIGN KEY (UserID) REFERENCES User(UserID),  
    FOREIGN KEY (LAT, LON) REFERENCES Location(LAT, LON)  
);
```

```
DROP TABLE Rent_raw;  
CREATE TABLE Rent_raw(  
    Year INT,  
    Amount INT,  
    Tract VARCHAR(255),  
    Tract_Number INT,  
    Neighborhood VARCHAR(255),  
    GEOID VARCHAR(255),  
    Row_ID VARCHAR(255) PRIMARY KEY,  
    Date DATE,
```

```

        LAT REAL,
        LON REAL
    );

DROP TABLE Crime_raw;
CREATE TABLE Crime_raw(
    DR_NO INT PRIMARY KEY,
    Date_Rptd DATE,
    Time_OCC INT,
    AREA INT,
    AREA_Name VARCHAR(255),
    Rpt_Dist_No INT,
    Crm_Cd INT,
    Crm_Cd_Desc VARCHAR(255),
    Mocodes INT,
    Vict_Age INT,
    Vlct_Sex CHAR(1),
    Vict_Descent CHAR(1),
    Premis_Cd INT,
    Premis_Desc VARCHAR(255),
    Weapon_Used_Cd INT,
    Weapon_Desc VARCHAR(255),
    Status CHAR(2),
    Status_Desc VARCHAR(255),
    Crm_Cd1 INT,
    Crm_Cd2 INT,
    Crm_Cd3 INT,
    Crm_Cd4 INT,
    Location VARCHAR(255),
    Cross_Street VARCHAR(255),
    LAT REAL,
    LON REAL
);

```

```

SELECT * FROM Rent_raw LIMIT 15;
SELECT * FROM Crime_raw LIMIT 15;

```

```

GRANT FILE ON *.* TO 'root'@'34.123.55.32:3306';
FLUSH PRIVILEGES;

```

```
LOAD DATA INFILE
'F:/university/UIUC/23SUMMER/Project/su23-cs411-team003-Lazy/src/datasets/Rent
t_Price_LA_clean.csv'
INTO TABLE Rent
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(Row_ID, Year, Amount, Tract, Tract_Number, Neighborhood, GEOID, LAT, LON,
Date);
```

```
DELETE FROM Crime_raw
WHERE Date_Rptd = 0000-00-00;
```

```
INSERT INTO table2 ( name , price )
      SELECT name , price FROM table1 WHERE id=5;
```

```
ALTER TABLE Crime_raw
ADD COLUMN Year INT;
```

```
INSERT INTO CrimeRate VALUES (-1, "Undefined crime rate");
```

```
UPDATE Crime_raw
SET Year = EXTRACT(YEAR FROM Date_Rptd);
```

```
INSERT INTO Date (Year, Date_OCC)
(SELECT DISTINCT Year, Date_Rptd
FROM Crime_raw)
UNION
(SELECT DISTINCT Year, Date
FROM Rent_raw);
```

```
SELECT COUNT(*) FROM Date LIMIT 15;
```

```
TRUNCATE TABLE my_table;
```

```
ALTER TABLE Location
DROP COLUMN Tract;
-- Select * from Location;
-- AREA_Name VARCHAR(255),
--     Neighborhood VARCHAR(255),
--     Tract VARCHAR(255),
```

```
TRUNCATE TABLE Location;
INSERT INTO Location (LAT, LON, RateNum)
(SELECT DISTINCT ROUND(LAT, 3), ROUND(LON, 3), -1
FROM Crime_raw)
UNION
(SELECT DISTINCT ROUND(LAT, 3), ROUND(LON, 3), -1
FROM Rent_raw);
```

```
SELECT COUNT(*)
FROM (
    SELECT DISTINCT ROUND(LAT, 3), ROUND(LON, 3)
    FROM Crime_raw
) AS alias_table;
```

```
SELECT COUNT(*)
FROM (
    SELECT DISTINCT ROUND(LAT, 3), ROUND(LON, 3)
    FROM Rent_raw
) AS alias_table;
```

```
SELECT * FROM Location LIMIT 15;
```

```
ALTER TABLE LocationFavourite
DROP FOREIGN KEY LocationFavourite_ibfk_2;
```

```
TRUNCATE TABLE Location;
DROP TABLE Location;
```

3.

+1.5% for inserting at least 1000 rows in the tables. (You should do a count query to show this, 1% for each table) Insert data to these tables. You should insert at least 1000 rows **each in** three of the tables.

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane displays a tree view of the database structure, including tables like 'Crime_raw', 'CrimeNumDescription', 'CrimeRate', 'Date', 'Location', 'LocationCrimeNum', 'LocationFavourite', 'nearLocations', 'Rent', 'Rent_raw', 'User', 'Views', 'Stored Procedures', 'Functions', 'rent_web', and 'sys'. The main editor window shows a SQL query: `SELECT (COUNT(*)) FROM Crime_raw;`. Below the query, the 'Result Grid' shows a single row with the value 317854. The 'Output' pane at the bottom shows the execution details: 'Action Output' with a green checkmark, 'Time' 00:47:42, 'Action' `SELECT (COUNT(*)) FROM Crime_raw LIMIT 0, 1000`, 'Message' '1 row(s) returned', and 'Duration / Fetch' '0.125 sec / 0.000 sec'. A status bar at the bottom indicates 'Query Completed'.

MySQL Workbench

cs411 X

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

classicmodels

proj_rent_crime

Tables

Crime

Crime_raw

CrimeNumDescription

CrimeRate

Date

Location

LocationCrimeNum

LocationFavourite

nearLocations

Rent

Rent_raw

User

Views

Stored Procedures

GetCrimeRateByLocation

Functions

rent_web

sys

Query 1 Query 2 SQL File 8

Limit to 1000 rows

1 SELECT (COUNT(*))

2 FROM Crime_raw;

3

SQLAdditions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid

Filter Rows: (COUNT(*))

317854

Administration Schemas

Information

No object selected

Result 1

Read Only Context Help Snippets

Output

Action Output

Time Action Message Duration / Fetch

1 00:47:42 SELECT (COUNT(*)) FROM Crime_raw LIMIT 0, 1000 1 row(s) returned 0.125 sec / 0.000 sec

Object Info Session

Query Completed

MySQL Workbench

cs411 x

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHMAS

Filter objects

classicmodels

proj_rent_crime

Tables

Crime

Crime_raw

CrimeNumDescription

CrimeRate

Date

Location

LocationCrimeNum

LocationFavourite

nearLocations

Rent

Rent_raw

User

Views

Stored Procedures

GetCrimeRateByLocation

Functions

rent_web

sys

Query 1 x Query 2 SQL File 8

Limit to 1000 rows

```
1 SELECT (COUNT(*))
2 FROM Location;
3
```

SQLAdditions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid

Filter Rows:

Export:

Wrap Cell Contents:

(COUNT(*))

40181

Administration Schemas

Information

No object selected

Result 2 x

Read Only Context Help Snippets

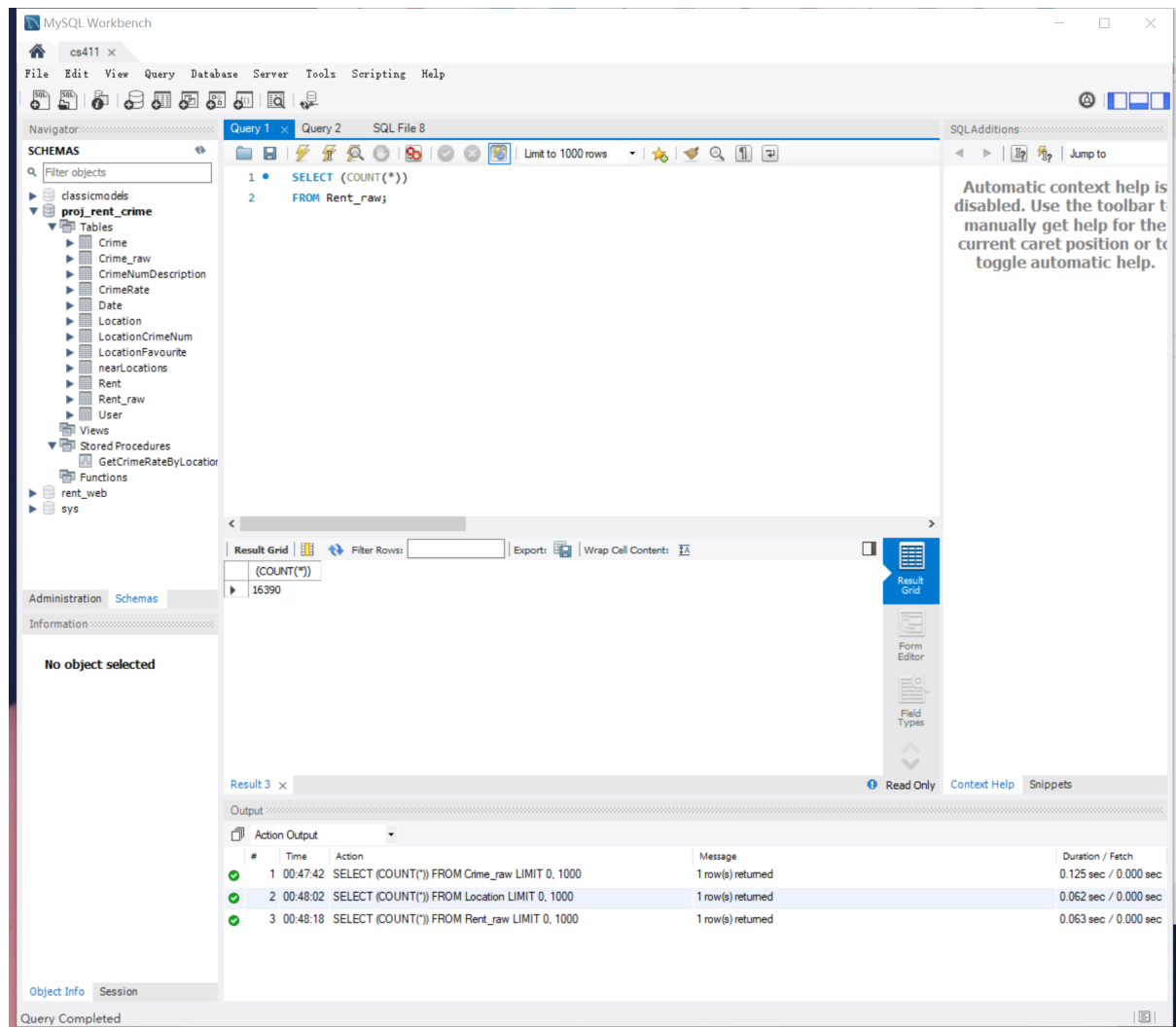
Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	00:47:42	SELECT (COUNT(*)) FROM Crime_raw LIMIT 0, 1000	1 row(s) returned	0.125 sec / 0.000 sec
2	00:48:02	SELECT (COUNT(*)) FROM Location LIMIT 0, 1000	1 row(s) returned	0.062 sec / 0.000 sec

Object Info Session

Query Completed



4. Advanced Queries are worth 7% and are graded (as a group) as follows:

+5% for developing two advanced queries (see point 4 for this stage, 2.5% each)

SQL Query1:

USE proj_rent_crime;

DROP TABLE IF EXISTS LocationCrimeNum;

DROP TABLE IF EXISTS CrimeNumDescription;

```
CREATE TABLE LocationCrimeNum (
    LAT REAL,
    LON REAL,
    Descriptions VARCHAR(255)
);
```

```
CREATE TABLE CrimeNumDescription(
    CrimeNumLow INT PRIMARY KEY,
    CrimeNumHigh INT,
    Descriptions VARCHAR(255)
);
```

```
INSERT INTO CrimeNumDescription
VALUES (0, 4, 'No Crime, Safe'),
(5, 9, 'Few Crime, Usually Safe'),
(10, 19, 'Some Crime, Should Be Safe'),
(20, 49, 'Middle Crime, Often Safe'),
(50, 99, 'More Crime, Not So Safe'),
(100, 499, 'Many Crime, Not Safe'),
(500, 10000, 'Much Crime, Very Unsafe');
```

```
INSERT INTO LocationCrimeNum
SELECT DISTINCT a.LAT, a.LON, cnd.Descriptions
    FROM (SELECT cr.LAT, cr.LON, COUNT(*) as Num
        FROM Crime_raw cr
        GROUP BY cr.LAT, cr.LON) a JOIN CrimeNumDescription cnd ON a.Num BETWEEN
cnd.CrimeNumLow AND cnd.CrimeNumHigh;
```

```
SELECT * FROM LocationCrimeNum;
```

SQL Query2:

```
USE proj_rent_crime;
SET @UserLat = 34.03;
SET @UserLon = -118.15;

SELECT @UserLat, @UserLon;

SELECT r.Row_ID, r.Amount, r.Tract, r.Year,
    SQRT(POW(r.LAT - @UserLat, 2) + POW(r.LON - @UserLon, 2)) AS Distance,
    a.AverageAmount
FROM Rent_raw r
JOIN (
    SELECT Tract, AVG(Amount) AS AverageAmount
    FROM Rent_raw
    GROUP BY Tract
) a ON r.Tract = a.Tract
ORDER BY Distance ASC, r.Amount ASC;
```

+2% for providing screenshots with the top 15 rows of the query results
(1% each)

SQL Query1:First 15 rows of output:

The screenshot shows the SQL Server Enterprise Manager interface. The 'Query1' window displays the following SQL code:

```

29 SELECT DISTINCT a.LAT, a.LON, a.Num, cnd.Descriptions
30 FROM (SELECT cr.LAT, cr.LON, COUNT(*) as Num
31 FROM Crime_raw cr
32 GROUP BY cr.LAT, cr.LON) a JOIN CrimeNumDescription cnd ON a.Num BETWEEN cnd.CrimeNumLow AND cnd.CrimeNumHigh;
33
34 * SELECT * FROM LocationCrimeNum;
35
36 SELECT DISTINCT a.LAT, a.LON, a.Num, cnd.Descriptions
37 FROM (SELECT cr.LAT, cr.LON, COUNT(*) as Num
38 FROM Crime_raw cr
39 GROUP BY cr.LAT, cr.LON) a JOIN CrimeNumDescription cnd ON a.Num BETWEEN cnd.CrimeNumLow AND cnd.CrimeNumHigh
40 LIMIT 15;
41

```

The 'Result Grid' shows the first 15 rows of the query results:

LAT	LON	Num	Descriptions
34.2367	-118.4955	2	No Crime, Safe
33.982	-118.2622	6	Few Crime, Usually Safe
34.0141	-118.2978	3	No Crime, Safe
34.0459	-118.2545	23	Middle Crime, Often Safe
34.0447	-118.2452	18	Some Crime, Should Be Safe
34.0375	-118.3508	25	Middle Crime, Often Safe
34.1865	-118.4019	4	No Crime, Safe
34.2198	-118.4468	2	No Crime, Safe
34.0617	-118.2489	34	Middle Crime, Often Safe
34.0604	-118.2504	32	Middle Crime, Often Safe
34.0482	-118.2585	182	Many Crime, Not Safe
34.0205	-118.262	14	Some Crime, Should Be Safe
34.0452	-118.2534	41	Middle Crime, Often Safe
34.0483	-118.2631	232	Many Crime, Not Safe
34.0448	-118.2474	120	Many Crime, Not Safe

This query return the location keyed by Latitude (LAT), Longitude (LON), the count of crime happened (CrimeNum), and the description of the severeness of the number (Description).

SQL Query2:First 15 Rows of this Query

The screenshot shows the MySQL Workbench interface. The 'Query1' window displays the following SQL code:

```

1 USE proj_rent_crime;
2 SET @UserLat = 34.03;
3 SET @UserLon = -118.15;
4
5 SELECT @UserLat, @UserLon;
6
7 SELECT r.Row_ID, r.Amount, r.Tract, r.Year,
8       SQRT(POW(r.LAT - @UserLat, 2) + POW(r.LON - @UserLon, 2)) AS Distance,
9       a.AverageAmount
10 FROM Rent_raw r
11 JOIN (
12     SELECT Tract, AVG(Amount) AS AverageAmount
13     FROM Rent_raw
14     GROUP BY Tract
15 ) a ON r.Tract = a.Tract
16 ORDER BY Distance ASC, r.Amount ASC;

```

The 'Result Grid' shows the first 15 rows of the query results:

Row_ID	Amount	Tract	Year	Distance	AverageAmount
Median_Rent_Price_2010_1400000US0603753_898	898	Census Tract 5303.02, Los Angeles County, Cal...	2010	0.0028716776455679938	889.5714
Median_Rent_Price_2011_1400000US0603753_914	914	Census Tract 5303.02, Los Angeles County, Cal...	2011	0.0028716776455679938	889.5714
Median_Rent_Price_2012_1400000US0603753_918	918	Census Tract 5303.02, Los Angeles County, Cal...	2012	0.0028716776455679938	889.5714
Median_Rent_Price_2013_1400000US0603753_1038	1038	Census Tract 5303.02, Los Angeles County, Cal...	2013	0.0028716776455679938	889.5714
Median_Rent_Price_2014_1400000US0603753_1046	1046	Census Tract 5303.02, Los Angeles County, Cal...	2014	0.0028716776455679938	889.5714
Median_Rent_Price_2015_1400000US0603753_1064	1064	Census Tract 5303.02, Los Angeles County, Cal...	2015	0.0028716776455679938	889.5714
Median_Rent_Price_2016_1400000US0603753_1063	1063	Census Tract 5303.02, Los Angeles County, Cal...	2016	0.0028716776455679938	889.5714
Median_Rent_Price_2011_1400000US0603753_886	886	Census Tract 5303.01, Los Angeles County, Cal...	2011	0.00486819563929866	935.4286
Median_Rent_Price_2012_1400000US0603753_884	884	Census Tract 5303.01, Los Angeles County, Cal...	2012	0.00486819563929866	935.4286
Median_Rent_Price_2010_1400000US0603753_902	902	Census Tract 5303.01, Los Angeles County, Cal...	2010	0.00486819563929866	935.4286
Median_Rent_Price_2013_1400000US0603753_946	946	Census Tract 5303.01, Los Angeles County, Cal...	2013	0.00486819563929866	935.4286
Median_Rent_Price_2014_1400000US0603753_954	954	Census Tract 5303.01, Los Angeles County, Cal...	2014	0.00486819563929866	935.4286
Median_Rent_Price_2015_1400000US0603753_960	960	Census Tract 5303.01, Los Angeles County, Cal...	2015	0.00486819563929866	935.4286
Median_Rent_Price_2016_1400000US0603753_988	988	Census Tract 5303.01, Los Angeles County, Cal...	2016	0.00486819563929866	935.4286
Median_Rent_Price_2011_1400000US0603753_1011	1011	Census Tract 5317.01, Los Angeles County, Cal...	2011	0.008574857972583844	1081.2857

This query return the Row_ID, Amount, Tract, Year, Distance (Calculated by SQRT (POW (r.LAT - @UserLat, 2) + POW (r.LON - @UserLon, 2))), and the AverageAmount of this tract in the years recorded.

5.Indexing Analysis is worth 8% and is graded (as a group) as follows:

- +3% on trying at least three different indexing designs (excluding the default index) for each advanced query.
- +4% on the indexing analysis reports.
- +1% on the accuracy and thoroughness of the analyses.

Query 1 Finding the count of crimes that happened at a location

This query return the location keyed by Latitude (LAT), Longitude (LON), the count of crime happened (CrimeNum), and the description of the severeness of the number (Description).





Explain Analyze Before Indexing

```
-> Table scan on <temporary> (cost=2.50..2.50 rows=0) (actual time=0.004..6.105 rows=57708 loops=1)
  -> Temporary table with deduplication (cost=4060.22..4060.22 rows=0) (actual time=479.404..488.984 rows=57708 loops=1)
    -> Filter: (a.Num between cnd.CrimeNumLow and cnd.CrimeNumHigh) (cost=4057.72 rows=0) (actual time=337.328..406.822 rows=57708 loops=1)
      -> Inner hash join (no condition) (cost=4057.72 rows=0) (actual time=337.321..381.092 rows=403956 loops=1)
        -> Table scan on a (cost=2.50..2.50 rows=0) (actual time=0.004..6.073 rows=57708 loops=1)
          -> Materialize (cost=2.50..2.50 rows=0) (actual time=337.245..346.668 rows=57708 loops=1)
            -> Table scan on <temporary> (actual time=0.003..2.981 rows=57708 loops=1)
              -> Aggregate using temporary table (actual time=323.430..329.846 rows=57708 loops=1)
                -> Table scan on cr (cost=33527.60 rows=315646) (actual time=0.028..148.484 rows=317854 loops=1)
                  -> Hash
```

-> Table scan on cnd (cost=0.95 rows=7) (actual time=0.039..0.043 rows=7 loops=1)

Setting LAT as an index

Indexes in Table

Visible	Key	Type	Uniq...	Columns
	 PRIMARY	BTREE	YES	DR_NO
	 idx_Crime_raw_L...	BTREE	NO	LAT

-> Table scan on <temporary> (cost=2.50..2.50 rows=0) (actual time=0.003..6.127 rows=57708 loops=1)

-> Temporary table with deduplication (cost=4060.22..4060.22 rows=0) (actual time=445.382..455.061 rows=57708 loops=1)

-> Filter: (a.Num between cnd.CrimeNumLow and cnd.CrimeNumHigh) (cost=4057.72 rows=0) (actual time=322.697..389.904 rows=57708 loops=1)

-> Inner hash join (no condition) (cost=4057.72 rows=0) (actual time=322.692..365.906 rows=403956 loops=1)

-> Table scan on a (cost=2.50..2.50 rows=0) (actual time=0.002..4.877 rows=57708 loops=1)

-> Materialize (cost=2.50..2.50 rows=0) (actual time=322.635..330.879 rows=57708 loops=1)

-> Table scan on <temporary> (actual time=0.002..2.974 rows=57708 loops=1)

-> Aggregate using temporary table (actual time=308.778..315.177 rows=57708 loops=1)



-> Table scan on cr (cost=33527.60 rows=315646) (actual time=0.072..150.816 rows=317854 loops=1)

-> Hash

-> Table scan on cnd (cost=0.95 rows=7) (actual time=0.029..0.038 rows=7 loops=1)

Setting LON as an index

Indexes in Table

Visible	Key	Type	Uniq...	Columns
<input checked="" type="checkbox"/>	 PRIMARY	BTREE	YES	DR_NO
<input checked="" type="checkbox"/>	 idx_Crime_raw_L...	BTREE	NO	LON

-> Table scan on <temporary> (cost=2.50..2.50 rows=0) (actual time=0.003..5.951 rows=57708 loops=1)

-> Temporary table with deduplication (cost=4060.22..4060.22 rows=0) (actual time=443.302..452.678 rows=57708 loops=1)

-> Filter: (a.Num between cnd.CrimeNumLow and cnd.CrimeNumHigh) (cost=4057.72 rows=0) (actual time=312.101..382.464 rows=57708 loops=1)

-> Inner hash join (no condition) (cost=4057.72 rows=0) (actual time=312.096..356.941 rows=403956 loops=1)

-> Table scan on a (cost=2.50..2.50 rows=0) (actual time=0.002..5.060 rows=57708 loops=1)

-> Materialize (cost=2.50..2.50 rows=0) (actual time=312.045..320.547 rows=57708 loops=1)

-> Table scan on <temporary> (actual time=0.002..2.861 rows=57708 loops=1)

-> Aggregate using temporary table (actual time=296.990..304.077 rows=57708 loops=1)



-> Table scan on cr (cost=33527.60 rows=315646) (actual time=0.029..143.068 rows=317854 loops=1)

-> Hash

-> Table scan on cnd (cost=0.95 rows=7) (actual time=0.029..0.033 rows=7 loops=1)

Setting Location as an index

Indexes in Table

Visible	Key	Type	Uniq...	Columns
<input checked="" type="checkbox"/>	 PRIMARY	BTREE	YES	DR_NO
<input checked="" type="checkbox"/>	 idx_Crime_raw_L...	BTREE	NO	Location

-> Table scan on <temporary> (cost=2.50..2.50 rows=0) (actual time=0.004..6.369 rows=57708 loops=1)

-> Temporary table with deduplication (cost=4060.22..4060.22 rows=0) (actual time=481.668..491.515 rows=57708 loops=1)

-> Filter: (a.Num between cnd.CrimeNumLow and cnd.CrimeNumHigh) (cost=4057.72 rows=0) (actual time=342.618..410.608 rows=57708 loops=1)

-> Inner hash join (no condition) (cost=4057.72 rows=0) (actual time=342.610..386.513 rows=403956 loops=1)

-> Table scan on a (cost=2.50..2.50 rows=0) (actual time=0.002..5.922 rows=57708 loops=1)

-> Materialize (cost=2.50..2.50 rows=0) (actual time=342.545..351.780 rows=57708 loops=1)

-> Table scan on <temporary> (actual time=0.003..2.984 rows=57708 loops=1)

-> Aggregate using temporary table (actual time=329.127..335.482 rows=57708 loops=1)

-> Table scan on cr (cost=33527.60 rows=315646) (actual time=0.033..149.878 rows=317854 loops=1)

-> Hash

-> Table scan on cnd (cost=0.95 rows=7) (actual time=0.039..0.044 rows=7 loops=1)

It seems that setting these separate columns as indexes independently does not change the performance of the query.

This is because in the advanced query, after the data is retrieved from Crime_raw, it is duplicated to a temporary table cr so that the performance of the afterward operations is not affected by the index anymore.

By not setting a temporary table cr, the effect of indexing should be realized.

```
-> Table scan on <temporary> (cost=2.50..2.50 rows=0) (actual time=0.002..5.772
rows=57708 loops=1)
  -> Temporary table with deduplication (cost=4060.22..4060.22 rows=0) (actual
time=428.591..437.753 rows=57708 loops=1)
    -> Filter: (a.Num between cnd.CrimeNumLow and cnd.CrimeNumHigh) (cost=4057.72
rows=0) (actual time=307.807..375.267 rows=57708 loops=1)
      -> Inner hash join (no condition) (cost=4057.72 rows=0) (actual
time=307.802..350.593 rows=403956 loops=1)
        -> Table scan on a (cost=2.50..2.50 rows=0) (actual time=0.002..4.438 rows=57708
loops=1)
          -> Materialize (cost=2.50..2.50 rows=0) (actual time=307.759..315.524
rows=57708 loops=1)
            -> Table scan on <temporary> (actual time=0.002..2.924 rows=57708 loops=1)
              -> Aggregate using temporary table (actual time=294.042..300.407
rows=57708 loops=1)
                -> Table scan on Crime_raw (cost=33527.60 rows=315646) (actual
time=0.022..137.384 rows=317854 loops=1)
                  -> Hash
                    -> Table scan on cnd (cost=0.95 rows=7) (actual time=0.024..0.028 rows=7
loops=1)
```

Even after deleting the temporary table duplication, the index is still applied.

This is not a mistake. The optimizer thinks that the use of these indexes are not proficient because the indexes are not unique, and some rows share the same value for LAT, LON, and LOCATION.

As a result, the optimizer does not use index scan under each circumstances and thus make the performance unchanged.

As a result, after some further tries on different index choices, the optimizer does not pick the index scan. So, the change in indexes does not bring a better effect to the query and this advanced SQL query is not combined with an index optimization.

Query 2 Finding the nearby location while there are no location on Rent

This query return the Row_ID, Amount, Tract, Year, Distance (Calculated by SQRT (POW (r.LAT - @UserLat,

2) + POW (r.LON - @UserLon, 2))), and the AverageAmount of this tract in the years recorded.

Explain Analyze Before Indexing (Duration time 0.132sec)

-> Sort: Distance, r.Amount (actual time=92.574..95.148 rows=16390 loops=1)
-> Stream results (cost=26535.60 rows=0) (actual time=40.772..79.574 rows=16390 loops=1)
-> Filter: (a.Tract = r.Tract) (cost=26535.60 rows=0) (actual time=40.743..66.747 rows=16390 loops=1)
-> Inner hash join (<hash>(a.Tract)=<hash>(r.Tract)) (cost=26535.60 rows=0) (actual time=40.737..52.068 rows=16390 loops=1)
-> Table scan on a (cost=2.50..2.50 rows=0) (actual time=0.001..0.295 rows=2344 loops=1)
-> Materialize (cost=2.50..2.50 rows=0) (actual time=24.494..24.935 rows=2344 loops=1)
-> Table scan on <temporary> (actual time=0.003..0.195 rows=2344 loops=1)
-> Aggregate using temporary table (actual time=22.769..23.180 rows=2344 loops=1)
-> Table scan on Rent_raw (cost=1659.45 rows=15712) (actual time=0.047..6.967 rows=16390 loops=1)
-> Hash
-> Table scan on r (cost=1659.45 rows=15712) (actual time=0.068..8.134 rows=16390 loops=1)

Setting Tract as index

Indexes in Table				
Visible	Key	Type	Uniq...	Columns
<input checked="" type="checkbox"/>	PRIMARY	BTREE	YES	Row_ID
<input checked="" type="checkbox"/>	idx_Rent_raw_Tr...	BTREE	NO	Tract

Explain Analyze after setting Tract as index (Duration time 0.212sec)

-> Sort: Distance, r.Amount (actual time=489.930..492.510 rows=16390 loops=1)
-> Stream results (cost=38694.64 rows=105499) (actual time=375.769..471.592 rows=16390 loops=1)
-> Nested loop inner join (cost=38694.64 rows=105499) (actual time=375.557..458.620 rows=16390 loops=1)
-> Filter: (a.Tract is not null) (cost=4801.66..1770.10 rows=15712) (actual time=375.484..376.506 rows=2344 loops=1)
-> Table scan on a (cost=0.01..198.90 rows=15712) (actual time=0.003..0.615 rows=2344 loops=1)
-> Materialize (cost=4801.86..5000.75 rows=15712) (actual time=375.478..376.264 rows=2344 loops=1)
-> Group aggregate: avg(Rent_raw.Amount) (cost=3230.65 rows=15712) (actual time=61.835..372.797 rows=2344 loops=1)
-> Index scan on Rent_raw using idx_Rent_raw_Tract (cost=1659.45 rows=15712) (actual time=61.789..357.490 rows=16390 loops=1)
-> Index lookup on r using idx_Rent_raw_Tract (Tract=a.Tract) (cost=1.68 rows=7) (actual time=0.032..0.034 rows=7 loops=2344)

Setting GEOID as index

Indexes in Table				
Visible	Key	Type	U	Columns
<input checked="" type="checkbox"/>	PRIMARY	BTREE	Y	Row_ID
<input checked="" type="checkbox"/>	idx_Rent_raw_GEOID	BTREE	N	GEOID

Explain Analyze after setting GEOID as index (Duration time 0.140sec)

-> Sort: Distance, r.Amount (actual time=102.127..104.838 rows=16390 loops=1)
-> Stream results (cost=26535.60 rows=0) (actual time=45.270..87.332 rows=16390 loops=1)
-> Filter: (a.Tract = r.Tract) (cost=26535.60 rows=0) (actual time=45.248..73.926 rows=16390 loops=1)
-> Inner hash join (<hash>(a.Tract)=<hash>(r.Tract)) (cost=26535.60 rows=0) (actual time=45.243..59.048 rows=16390 loops=1)
-> Table scan on a (cost=2.50..2.50 rows=0) (actual time=0.001..0.305 rows=2344 loops=1)
-> Materialize (cost=2.50..2.50 rows=0) (actual time=26.387..26.834 rows=2344 loops=1)
-> Table scan on <temporary> (actual time=0.002..0.201 rows=2344 loops=1)
-> Aggregate using temporary table (actual time=24.771..25.138 rows=2344 loops=1)
-> Table scan on Rent_raw (cost=1659.45 rows=15712) (actual time=0.647..6.810 rows=16390 loops=1)
-> Hash
-> Table scan on r (cost=1659.45 rows=15712) (actual time=0.053..8.033 rows=16390 loops=1)

Setting Tract and GEOID as index

Indexes in Table				
Visible	Key	Type	U	Columns
<input checked="" type="checkbox"/>	PRIMARY	BTREE	Y	Row_ID
<input checked="" type="checkbox"/>	idx_Rent_raw_Tract	BTREE	N	Tract
<input checked="" type="checkbox"/>	idx_Rent_raw_GEOID	BTREE	N	GEOID

Explain Analyze after setting Tract and GEOID as index (Duration time 0.214sec)

-> Sort: Distance, r.Amount (actual time=173.029..175.692 rows=16390 loops=1)
-> Stream results (cost=39206.60 rows=106961) (actual time=68.759..158.796 rows=16390 loops=1)
-> Nested loop inner join (cost=39206.60 rows=106961) (actual time=68.730..146.108 rows=16390 loops=1)
-> Filter: (a.Tract is not null) (cost=4801.66..1770.10 rows=15712) (actual time=68.673..69.881 rows=2344 loops=1)
-> Table scan on a (cost=0.01..198.90 rows=15712) (actual time=0.005..0.784 rows=2344 loops=1)

-> Materialize (cost=4801.86..5000.75 rows=15712) (actual time=68.669..69.643 rows=2344 loops=1)

-> Group aggregate: avg(Rent_raw.Amount) (cost=3230.65 rows=15712) (actual time=0.168..66.972 rows=2344 loops=1)

-> Index scan on Rent_raw using idx_Rent_raw_Tract (cost=1659.45 rows=15712) (actual time=0.155..52.110 rows=16390 loops=1)

-> Index lookup on r using idx_Rent_raw_Tract (Tract=a.Tract) (cost=1.70 rows=7) (actual time=0.030..0.032 rows=7 loops=2344)

Reasons why indexing varied the duration time

Because indexing replaced the Table scan by Index scan, which varies the required scanning time. Indexing also replace inner hash join by nested loop inner join, which also varies the required time. As for materialize, indexing also varies its time. Those three components influence the duration time together.

Which difference made by setting Tract as an index, and how does this difference influence the duration time?

By setting Tract as an index, MySQL is able to use an indexed nested loop inner join instead of a hash join for the query. (But this query costed more time on inner join, from actual time=40.737..52.068 to actual time=375.557..458.620)

In addition, an index scan operation is now used instead of Table scan, which also increased the required time (from actual time=0.047..6.967 to actual time=61.789..357.490)

The reason of longer nested loop inner join and index scan operation is that the Tract is not unique, Index scan on such not unique column is not necessary.

Overall, setting Tract as an index decreased the performance and slowed the processing speed.

Which difference made by setting GEOID as an index, and how does this difference influence the duration time?

By setting GEOID as an index, MySQL failed to change the operations in Explain Analyze. (No change in "replaced the Table scan by Index scan" and "replace inner hash join by nested loop inner join"). That is because GEOID is not unique (same GEOID in the different ROW_ID in different year). In addition, GEOID in the query is not in WHERE, JOIN, ORDER BY and GROUP BY clause. GEOID can not be set as useful index.

The duration time is similar to that without indexing.

Which difference made by setting Tract & GEOID as an index, and how does this difference influence the duration time?

Same as the difference made by only setting Tract as index, the longer loop inner join and the longer index scan operation decreased the performance of processing, longer the duration time.

Overall, setting an unique and in critical position (in WHERE, JOIN, ORDER BY and GROUP BY clause) column as index could better the performance, otherwise it would slow the processing speed, or failed to be set as valid index. In table rent_raw, the only column fulfill those requirements is primary key Row_ID. So any redundant indexing on this query will make the performance worse.