CS434 – Data Base Theory and Design

Project #5

Team Database Application (TDA): Part 5 – Querying and Schema Tuning

Team

Lipika Baniya | 800794205 | lbaniya@siue.edu

The domain I would like to manage with the TDA is **Washington DC Crime Datasets 2024** by the District of Columbia Metropolitan Police Department (MPD).

General Nature of application

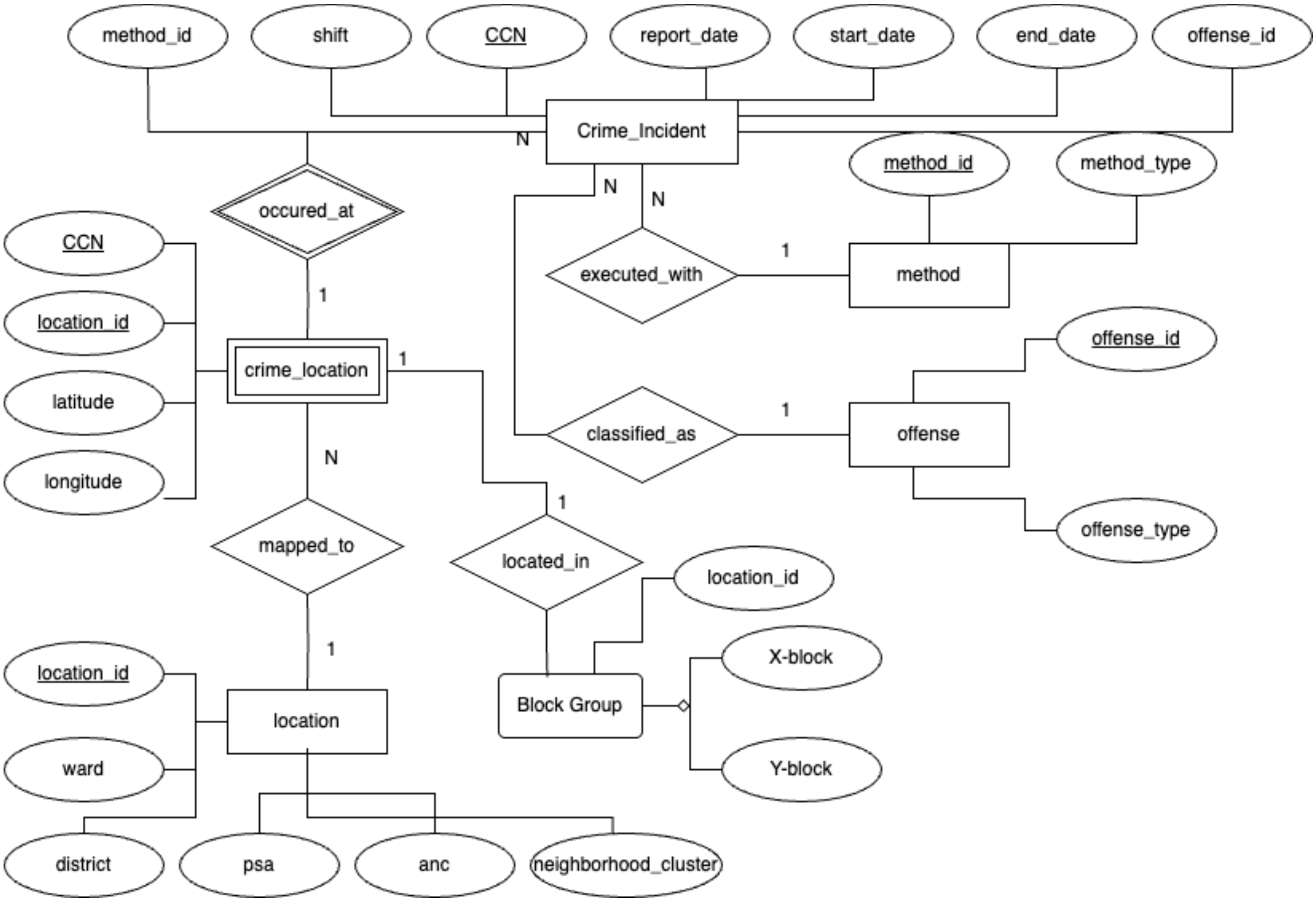
The main goal of an Entity Relationship Diagram (ER Diagram) is to explain the relationship between entities; it is a structural design of the database. Through the help of specialized symbols, it helps to define the relationship between entities. It is based on three main principles entities, attributes and relationships, these help to design the database that would be required before implementing the database. It is a systematic process to design a database as it would require analyzing all requirements.

About Data

Washington, D.C. has been facing significant challenges in ensuring public safety due to the varying and growing crime rates in different neighborhoods and time periods. It is important for law enforcement agencies to understand when and where crimes occur so that it can respond efficiently and allocate limited resources wisely. Imagine a robust database system that is designed to handle this task effectively, because without a data-driven approach and structured database, policing efforts may remain reactive, which would result in delays or gaps in coverage in high-risk areas. This database includes various entities, each representing a key component of crime data management.

In the previous project, I created the schema and inserted data into PostgreSQL based on the ER diagram. I used PostgreSQL to CREATE TABLE command and inserted data values and types for each entity's attributes. I have attached screenshots of the entire dataset and separate files for each table relation which were included in Project 4.

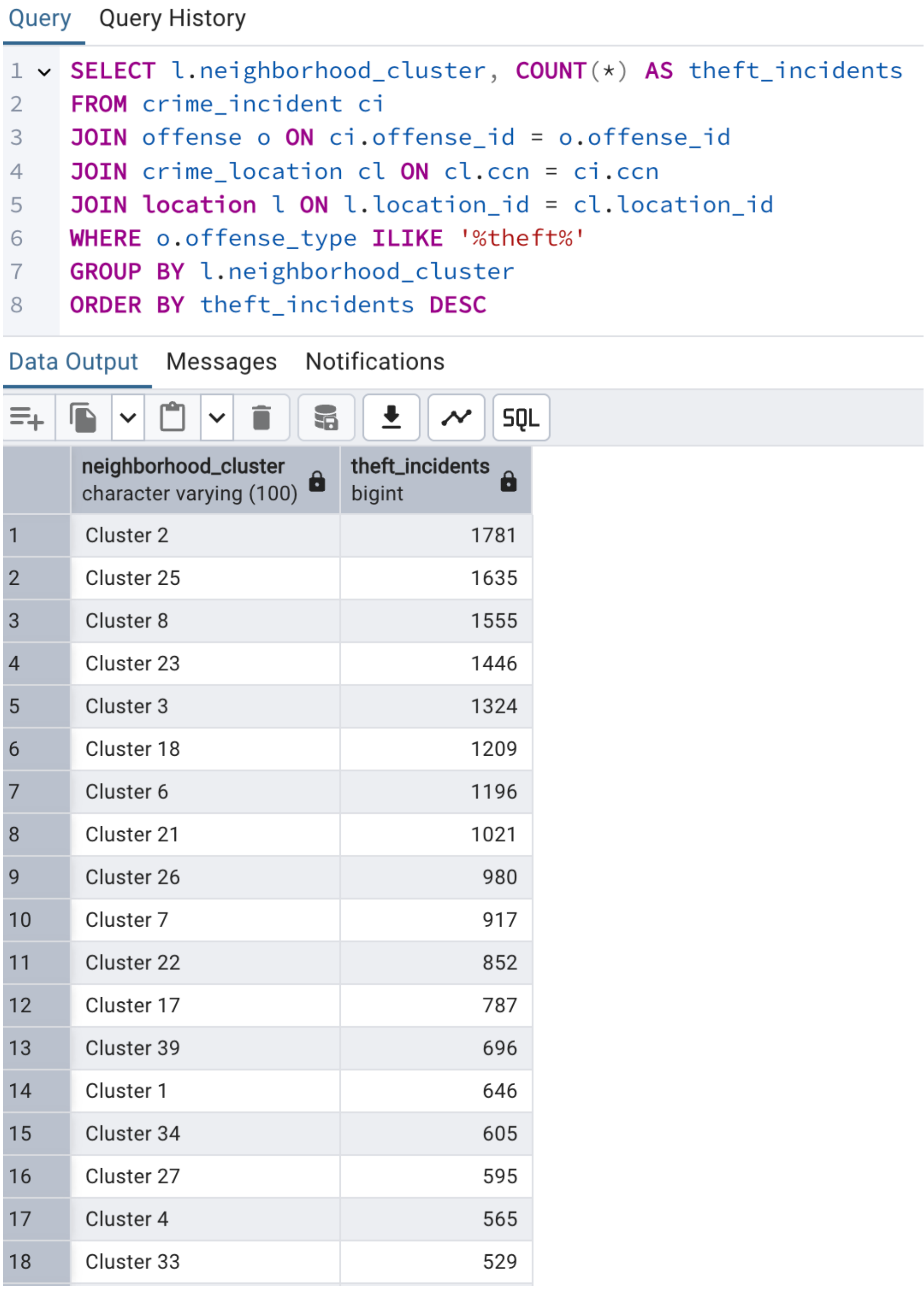
ER Diagram


method_id
shift
CCN
report_date
start_date
end_date
offense_id
Crime_Incident
N
N
N
occured_at
1
method_id
method_type
executed_with
1
method
CCN
location_id
latitude
longitude
crime_location
1
offense_id
classified_as
1
offense
offense_type
N
mapped_to
1
located_in
location_id
1
location_id
location
Block Group
X-block
Y-block
ward
district
psa
anc
neighborhood_cluster

1. **Queries**

1.1. Neighborhood Clusters with highest number of reported theft incidents

Question: Which neighborhood clusters have the highest number of reported theft incidents?



```sql
SELECT l.neighborhood_cluster, COUNT(*) AS theft_incidents
FROM crime_incident ci
JOIN offense o ON ci.offense_id = o.offense_id
JOIN crime_location cl ON cl.ccn = ci.ccn
JOIN location l ON l.location_id = cl.location_id
WHERE o.offense_type ILIKE '%theft%'
GROUP BY l.neighborhood_cluster
ORDER BY theft_incidents DESC
```

	neighborhood_cluster character varying (100)	theft_incidents bigint
1	Cluster 2	1781
2	Cluster 25	1635
3	Cluster 8	1555
4	Cluster 23	1446
5	Cluster 3	1324
6	Cluster 18	1209
7	Cluster 6	1196
8	Cluster 21	1021
9	Cluster 26	980
10	Cluster 7	917
11	Cluster 22	852
12	Cluster 17	787
13	Cluster 39	696
14	Cluster 1	646
15	Cluster 34	605
16	Cluster 27	595
17	Cluster 4	565
18	Cluster 33	529

Figure 1: Neighborhood Clusters with highest number of reported theft incidents

Figure 1 shows the result of the query, which lists neighborhood clusters along with the number of theft-related incidents.

Explanation

To identify which neighborhood clusters experienced the most theft incidents, I used the following tables

- Crime_incident: this table is logbook of all crimes, including the offense_id
- Offense: provides the offense type (e.g. theft, assault)

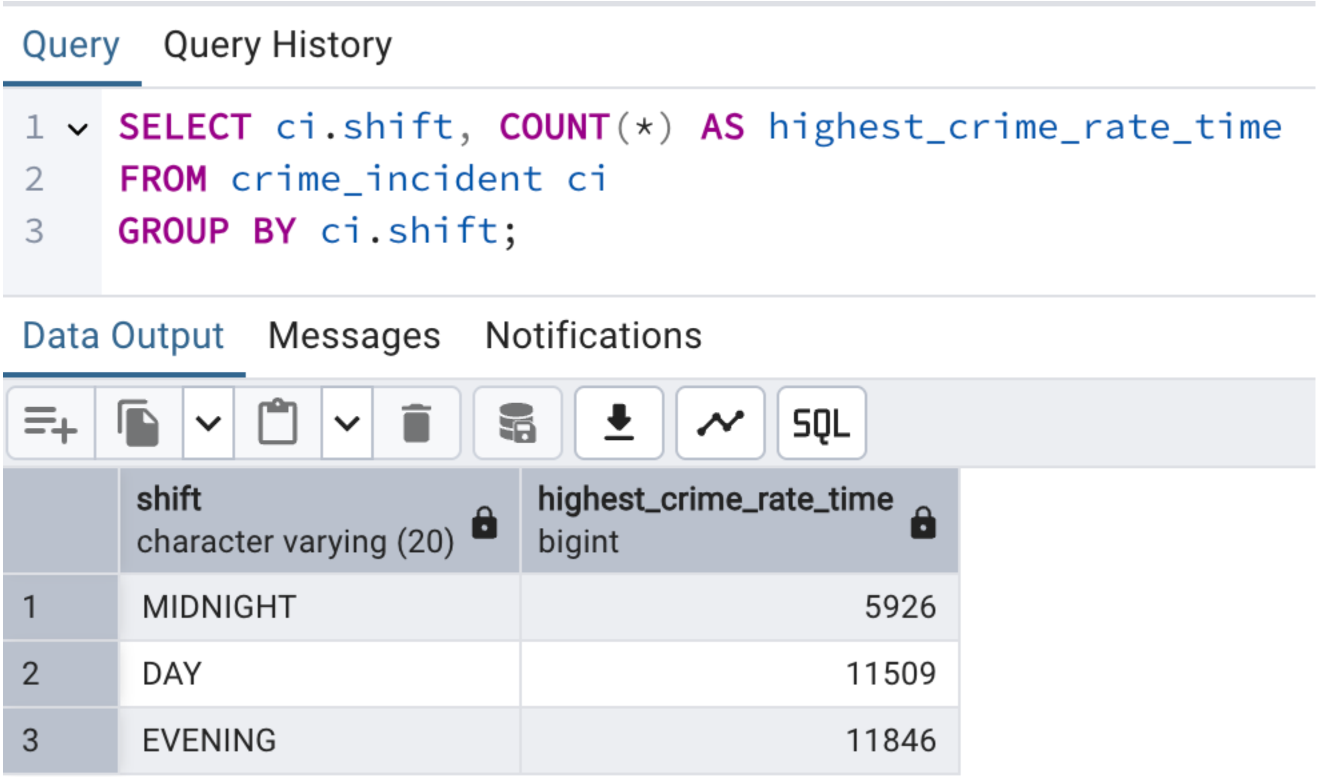
- Crime_location links each incident to the neighborhood cluster
- Location: includes the detailed geographic attributes, including neighborhood cluster

The query:

- WHERE clause with LIKE '%theft%' filters only those crimes that are theft related.
- Neighborhood_cluster is grouped to count the number of thefts in each.
- Orders the result in descending order to show the clusters with the highest theft counts first.

1.2. Major time for crime to occur

Question: At what time during the day does the crime occur the most?



```sql
SELECT ci.shift, COUNT(*) AS highest_crime_rate_time
FROM crime_incident ci
GROUP BY ci.shift;
```

	shift character varying (20)	highest_crime_rate_time bigint
1	MIDNIGHT	5926
2	DAY	11509
3	EVENING	11846

Figure 2: Major time for crime to occur

Figure 2 shows the result of the query, which displays the major time during the day does the crime occur the most.

Explanation

To find the major time for the crime to occur, I used the following tables

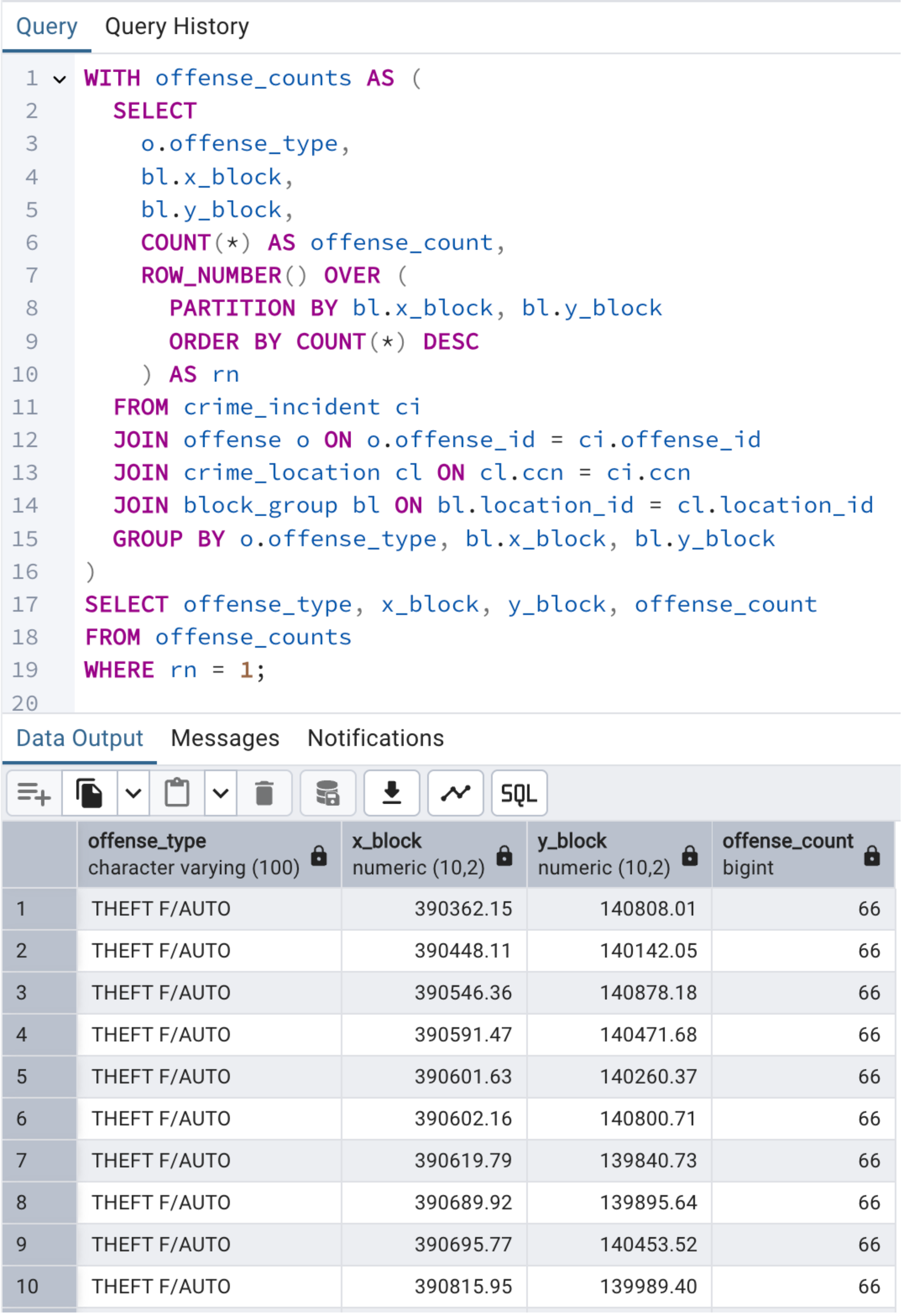
- Crime_incident: this table includes all crime details, including shift.

The query:

- The shift is grouped to show at what time the crime occurs the most.

1.3. Top Crime Type in Each Block Group

Question: What is the major crime in each block group?



```sql
WITH offense_counts AS (
  SELECT
    o.offense_type,
    bl.x_block,
    bl.y_block,
    COUNT(*) AS offense_count,
    ROW_NUMBER() OVER (
      PARTITION BY bl.x_block, bl.y_block
      ORDER BY COUNT(*) DESC
    ) AS rn
  FROM crime_incident ci
  JOIN offense o ON o.offense_id = ci.offense_id
  JOIN crime_location cl ON cl.ccn = ci.ccn
  JOIN block_group bl ON bl.location_id = cl.location_id
  GROUP BY o.offense_type, bl.x_block, bl.y_block
)
SELECT offense_type, x_block, y_block, offense_count
FROM offense_counts
WHERE rn = 1;
```

	offense_type character varying (100)	x_block numeric (10,2)	y_block numeric (10,2)	offense_count bigint
1	THEFT F/AUTO	390362.15	140808.01	66
2	THEFT F/AUTO	390448.11	140142.05	66
3	THEFT F/AUTO	390546.36	140878.18	66
4	THEFT F/AUTO	390591.47	140471.68	66
5	THEFT F/AUTO	390601.63	140260.37	66
6	THEFT F/AUTO	390602.16	140800.71	66
7	THEFT F/AUTO	390619.79	139840.73	66
8	THEFT F/AUTO	390689.92	139895.64	66
9	THEFT F/AUTO	390695.77	140453.52	66
10	THEFT F/AUTO	390815.95	139989.40	66

Figure 3: Top Crime Type in Each Block Group

Figure 3 displays the major crime in each block group.

Explanation

To find the major crime in each block group, the following tables were used:

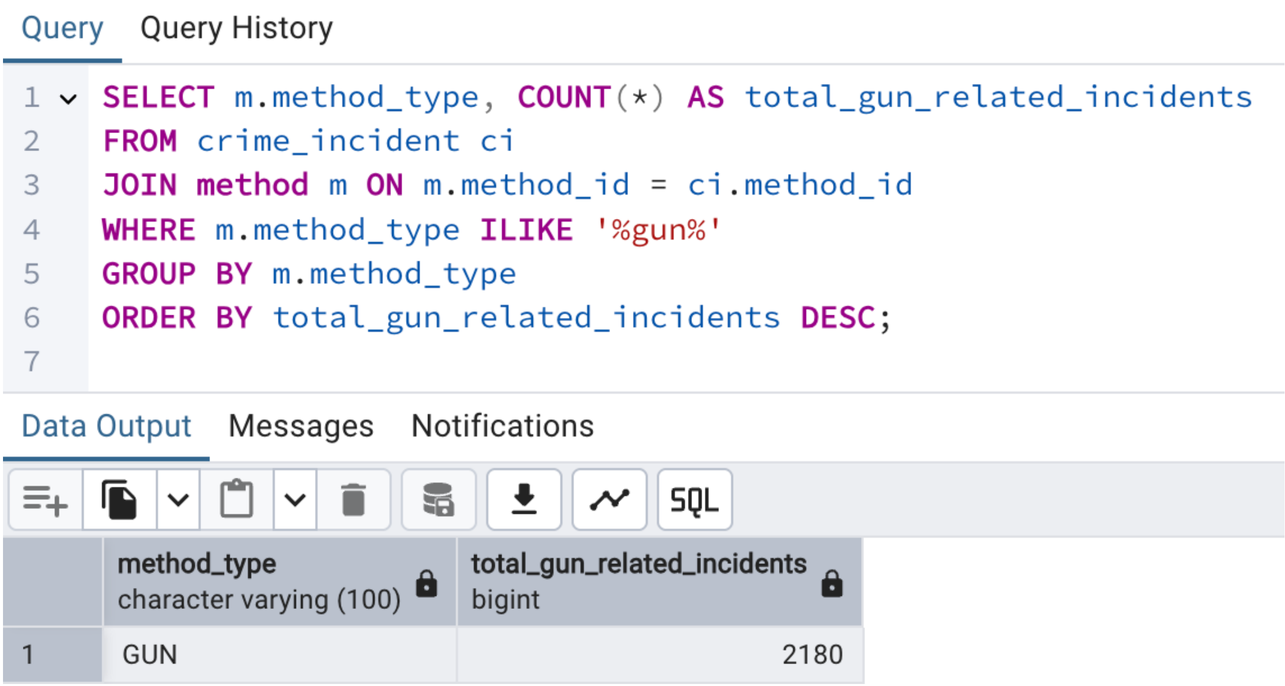
- Crime_incident: the main record of incidents
- Offense: maps offense_id to a name like “THEFT F/AUTO”.
- Crime_location: links crimes to a location.
- Block_group: gives geographic blocks via x_block, y_block.

The query:

- WITH offense_counts AS is used to create Common Table Expression (CTE) to get temporary result named offense_counts.
- SELECT is used to find the Type of offense, X and Y coordinates of block_group and count of how many such offenses occurred.
- Row_number() for each (x_block, y_block) pair, ranks the most frequent offense in each block.
- Group_by is used to group offense_type and block_group to count number of times each offense happens in each block.
- The final selects query filters only the top-ranked offense type per block group.

1.4.Incidents with “Gun” involved

Question: How many incidents involved a "Gun" as the method?



```sql
SELECT m.method_type, COUNT(*) AS total_gun_related_incidents
FROM crime_incident ci
JOIN method m ON m.method_id = ci.method_id
WHERE m.method_type ILIKE '%gun%'
GROUP BY m.method_type
ORDER BY total_gun_related_incidents DESC;
```

	method_type character varying (100)	total_gun_related_incidents bigint
1	GUN	2180

Figure 4: Incidents with "Gun" involved

Figure 4 shows the number of incidents that involved “gun” as the method.

Explanation

To find incidents that involved “Gun” as the method, I used the following the tables:

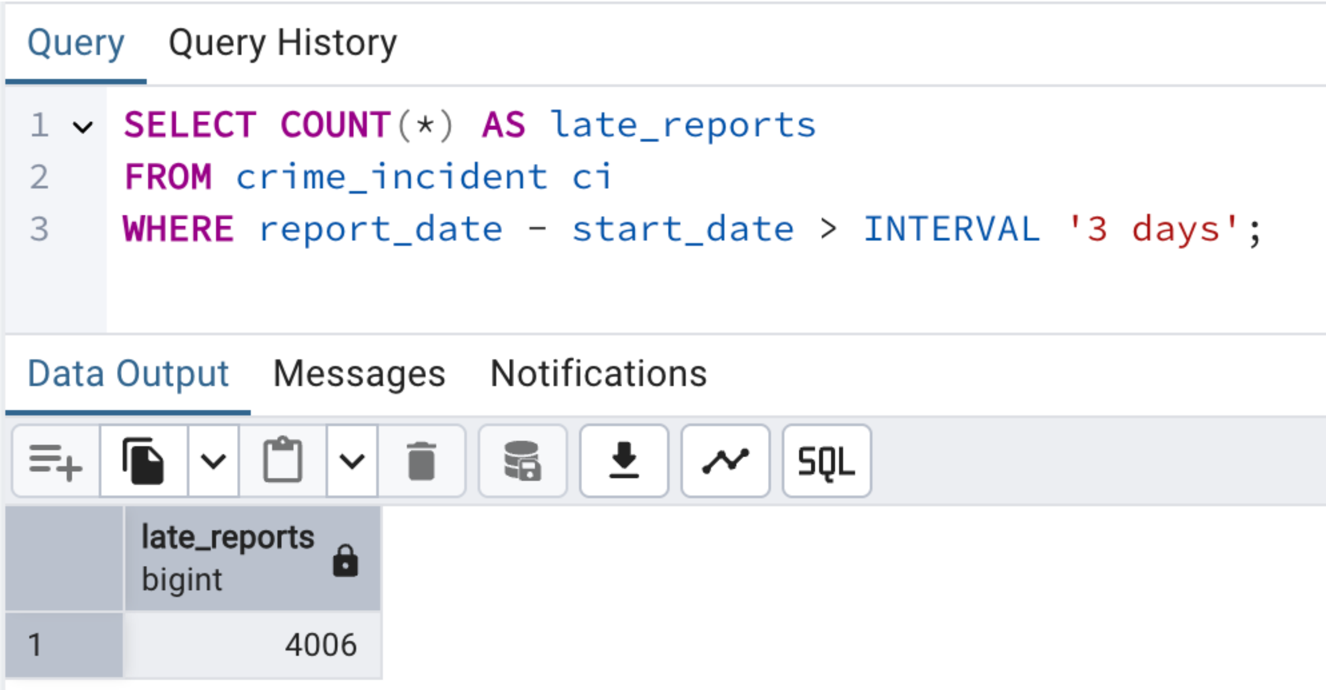
- Crime_incident: this table includes all crime details, including shift.
- Method: this table includes method details, including method type (e.g. “KNIFE”, “GUN”)

The query:

- JOIN is used to join crime_incident to the method table using method_id.
- WHERE ILIKE “%gun%” is used to find methods that use gun.
- GROUP BY is used to count the number of incidents that involved gun.
- ORDER BY is used to display in descending order.

1.5. Crime Reported Late

Question: How many crimes were reported more than 3 days after they occurred?



```
SELECT COUNT(*) AS late_reports
FROM crime_incident ci
WHERE report_date - start_date > INTERVAL '3 days';
```

	late_reports bigint
1	4006

Figure 5: crimes that were reported late

Figure 5 shows the number of crimes reported after 3 days than they occurred.

Explanation

To find the number of reports that were reported late, I used the queries:

- Crime_incident: this table includes all crime details, including report_date, start_date.

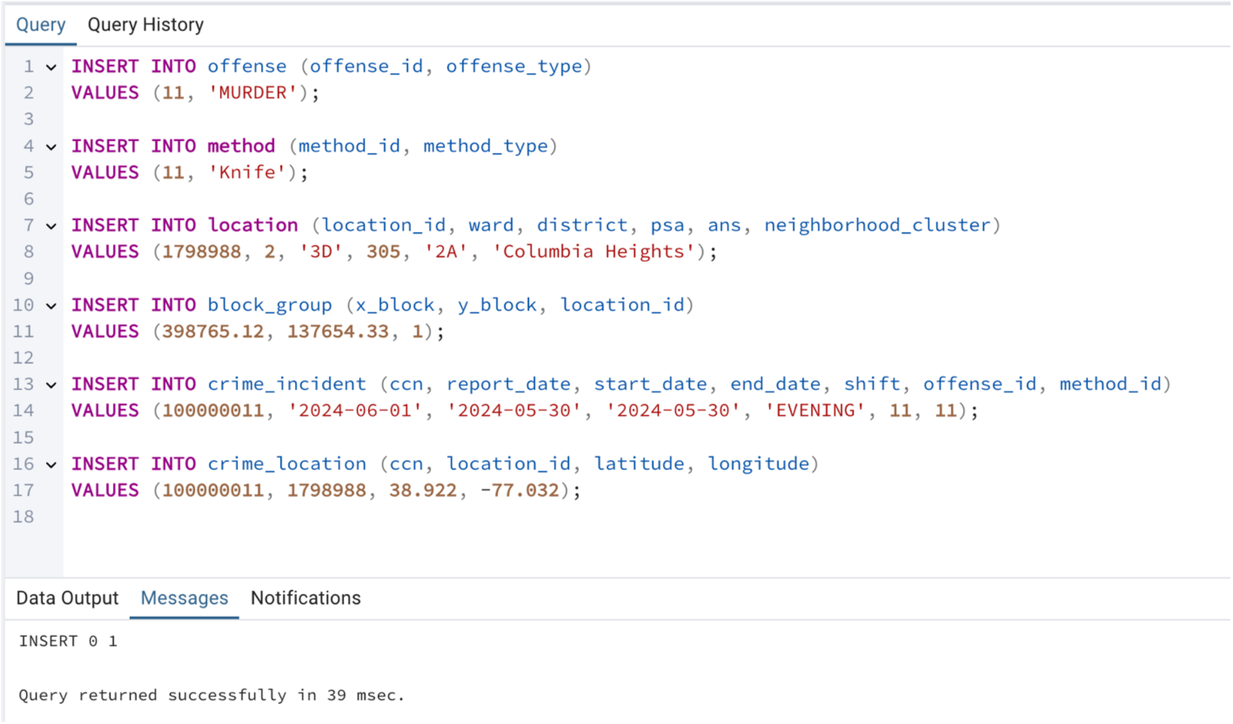
The query:

- WHERE is used to subtract report_date – start_date to find interval of 3 days.

2. Data Modification Queries

2.1. Insert

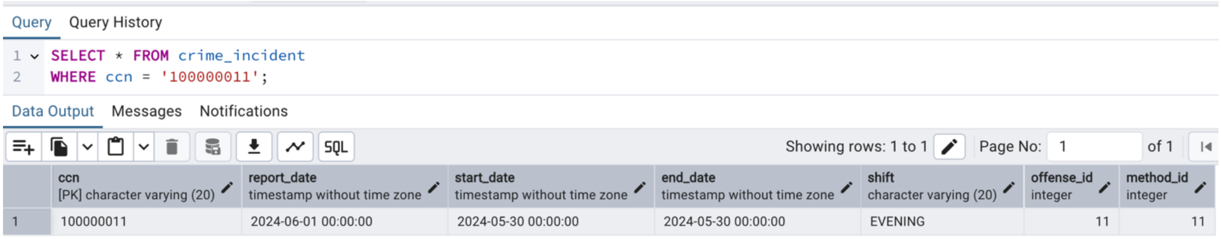
For the insert statement we are inputting into all the six table using dummy data (will be deleted during the delete command operation).



```
INSERT INTO offense (offense_id, offense_type)
VALUES (11, 'MURDER');

INSERT INTO method (method_id, method_type)
VALUES (11, 'Knife');

INSERT INTO location (location_id, ward, district, psa, ans, neighborhood_cluster)
VALUES (1798988, 2, '3D', 305, '2A', 'Columbia Heights');

INSERT INTO block_group (x_block, y_block, location_id)
VALUES (398765.12, 137654.33, 1);

INSERT INTO crime_incident (ccn, report_date, start_date, end_date, shift, offense_id, method_id)
VALUES (100000011, '2024-06-01', '2024-05-30', '2024-05-30', 'EVENING', 11, 11);

INSERT INTO crime_location (ccn, location_id, latitude, longitude)
VALUES (100000011, 1798988, 38.922, -77.032);
```

```
INSERT 0 1

Query returned successfully in 39 msec.
```

Figure 6: Insert Query

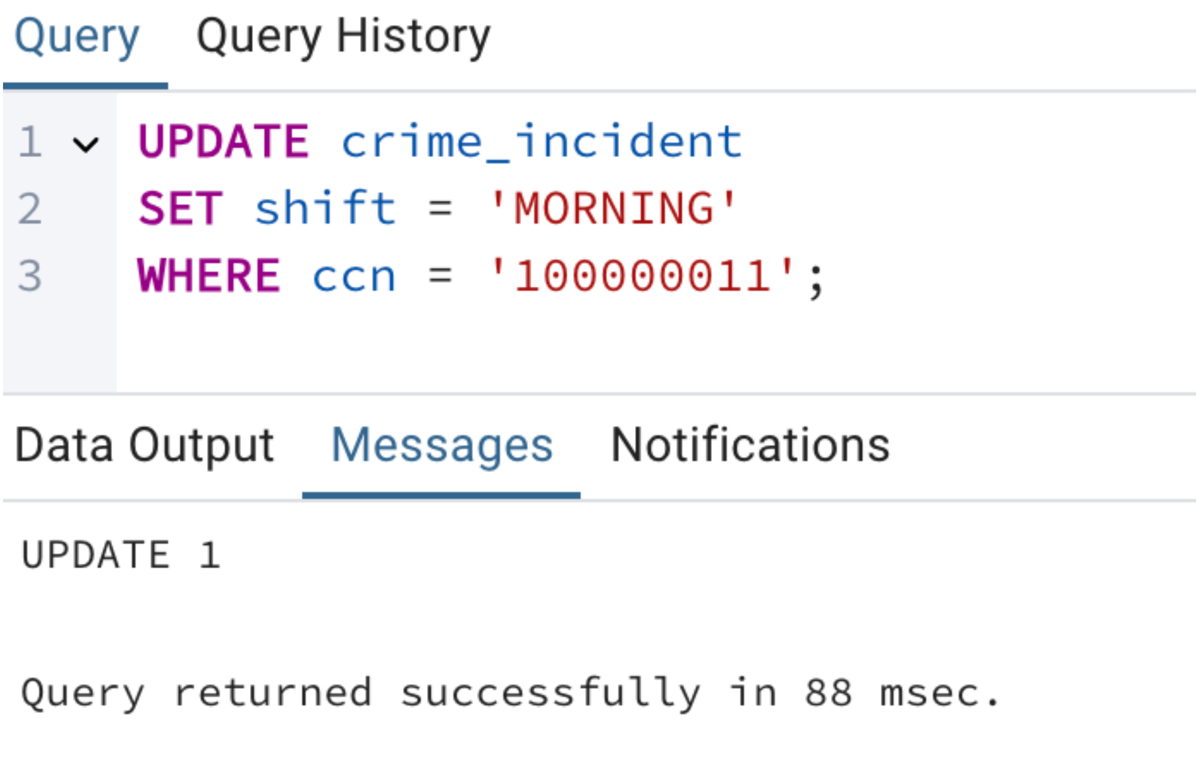


```
SELECT * FROM crime_incident
WHERE ccn = '100000011';
```

	ccn [PK] character varying (20)	report_date timestamp without time zone	start_date timestamp without time zone	end_date timestamp without time zone	shift character varying (20)	offense_id integer	method_id integer
1	100000011	2024-06-01 00:00:00	2024-05-30 00:00:00	2024-05-30 00:00:00	EVENING	11	11

Figure 7: New tuple added to the DB

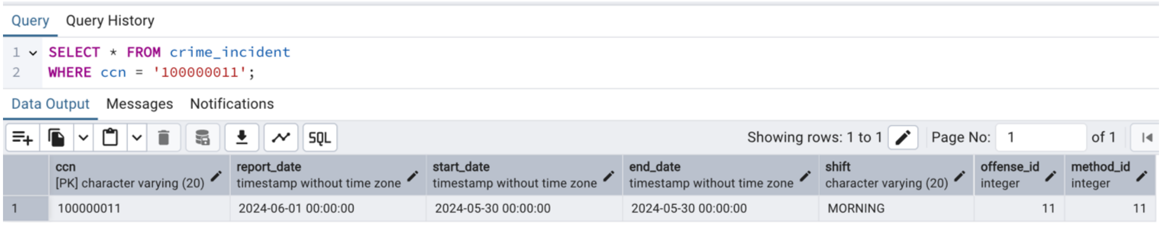
2.2.Update: Updating Shift



```sql
UPDATE crime_incident
SET shift = 'MORNING'
WHERE ccn = '100000011';
```

```
UPDATE 1

Query returned successfully in 88 msec.
```

Figure 8: Updating shift



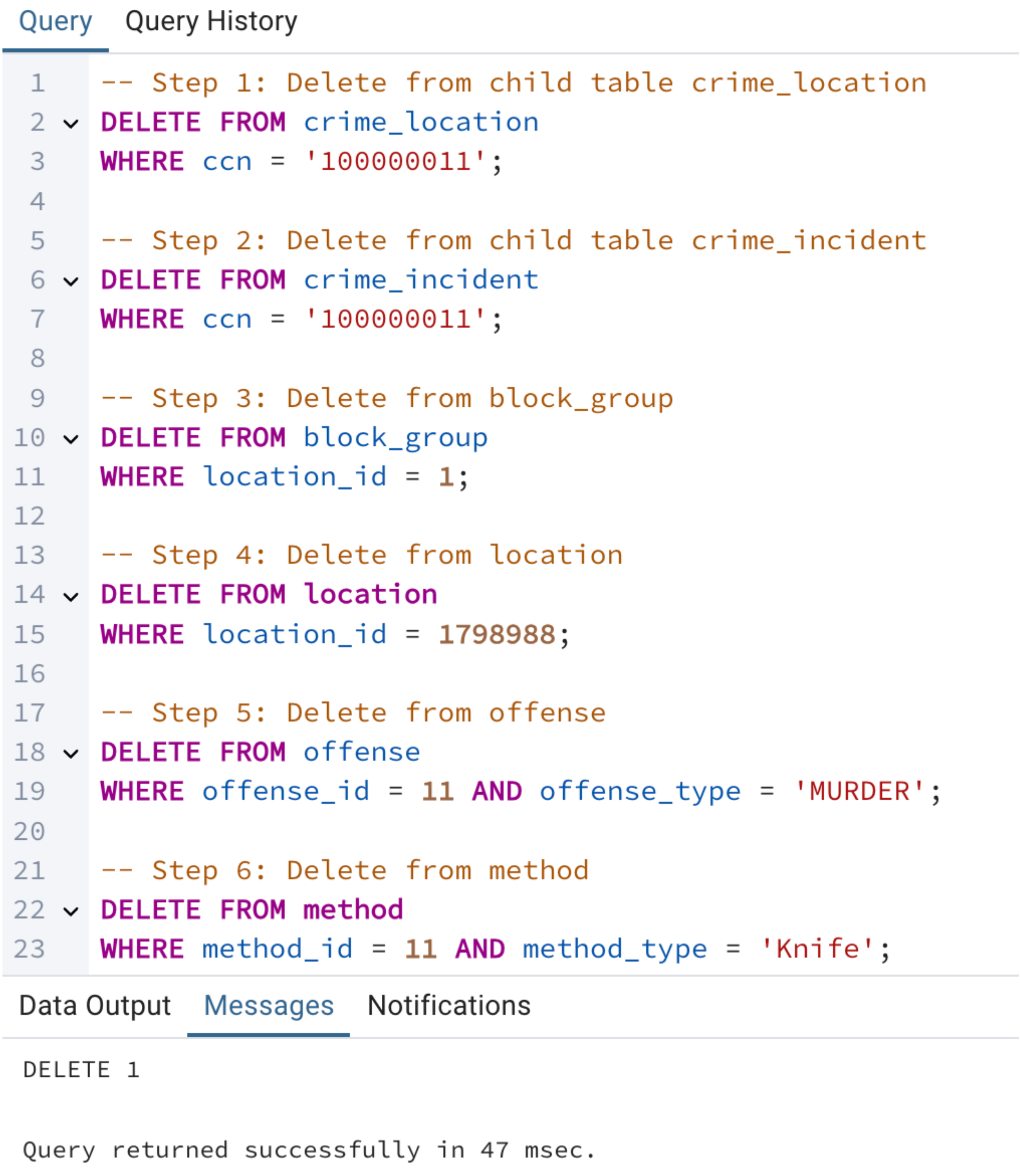
```sql
SELECT * FROM crime_incident
WHERE ccn = '100000011';
```

	ccn [PK] character varying (20)	report_date timestamp without time zone	start_date timestamp without time zone	end_date timestamp without time zone	shift character varying (20)	offense_id integer	method_id integer
1	100000011	2024-06-01 00:00:00	2024-05-30 00:00:00	2024-05-30 00:00:00	MORNING	11	11

Figure 9: Updated record

2.3.Delete

Deleting the recently added dummy record from all the tables.



```sql
-- Step 1: Delete from child table crime_location
DELETE FROM crime_location
WHERE ccn = '100000011';

-- Step 2: Delete from child table crime_incident
DELETE FROM crime_incident
WHERE ccn = '100000011';

-- Step 3: Delete from block_group
DELETE FROM block_group
WHERE location_id = 1;

-- Step 4: Delete from location
DELETE FROM location
WHERE location_id = 1798988;

-- Step 5: Delete from offense
DELETE FROM offense
WHERE offense_id = 11 AND offense_type = 'MURDER';

-- Step 6: Delete from method
DELETE FROM method
WHERE method_id = 11 AND method_type = 'Knife';
```

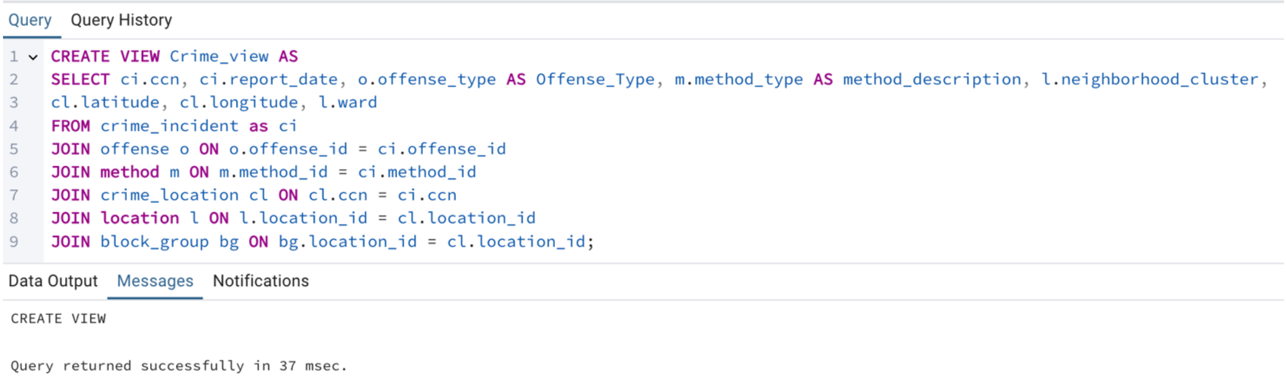
```
DELETE 1


Query returned successfully in 47 msec.
```

Figure 10: Delete the records

3. Create View

3.1.Creating the view table

A virtual table using existing schema has been created using create view as follows:

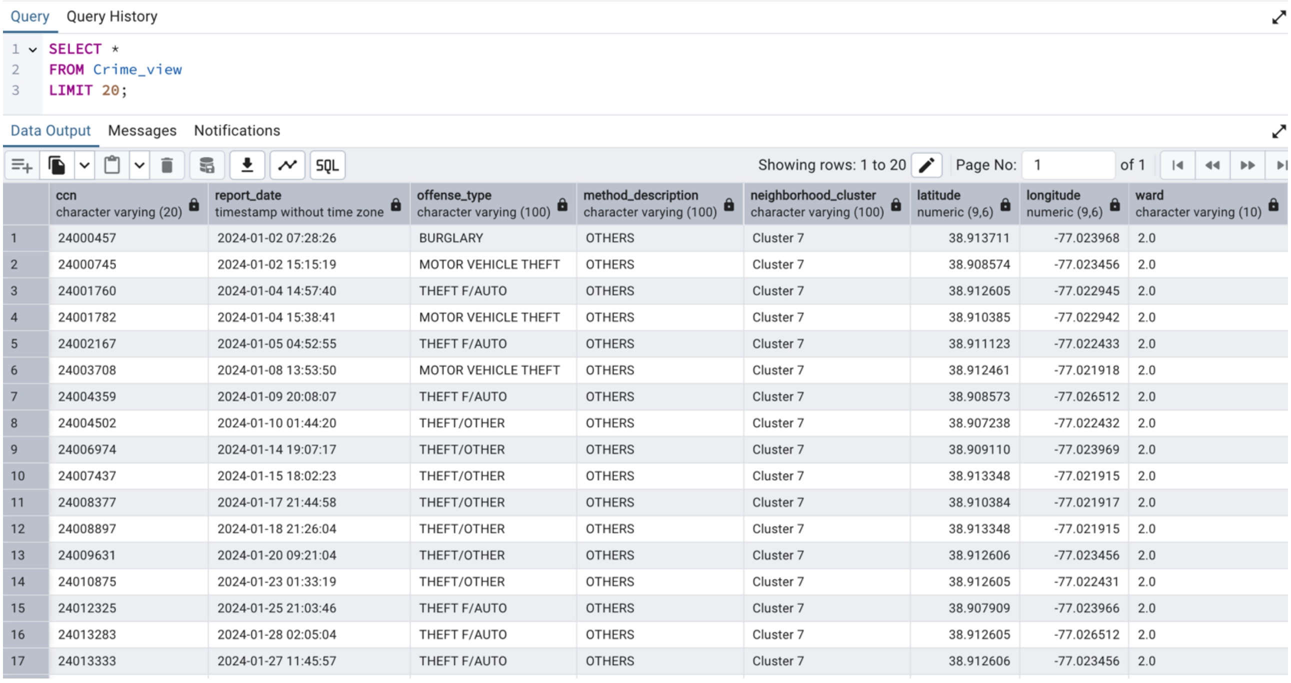


```sql
CREATE VIEW Crime_view AS
SELECT ci.ccn, ci.report_date, o.offense_type AS Offense_Type, m.method_type AS method_description, l.neighborhood_cluster,
cl.latitude, cl.longitude, l.ward
FROM crime_incident as ci
JOIN offense o ON o.offense_id = ci.offense_id
JOIN method m ON m.method_id = ci.method_id
JOIN crime_location cl ON cl.ccn = ci.ccn
JOIN location l ON l.location_id = cl.location_id
JOIN block_group bg ON bg.location_id = cl.location_id;
```

```
CREATE VIEW

Query returned successfully in 37 msec.
```

Figure 11: Query for CREATE VIEW

3.2.The Crime View Table

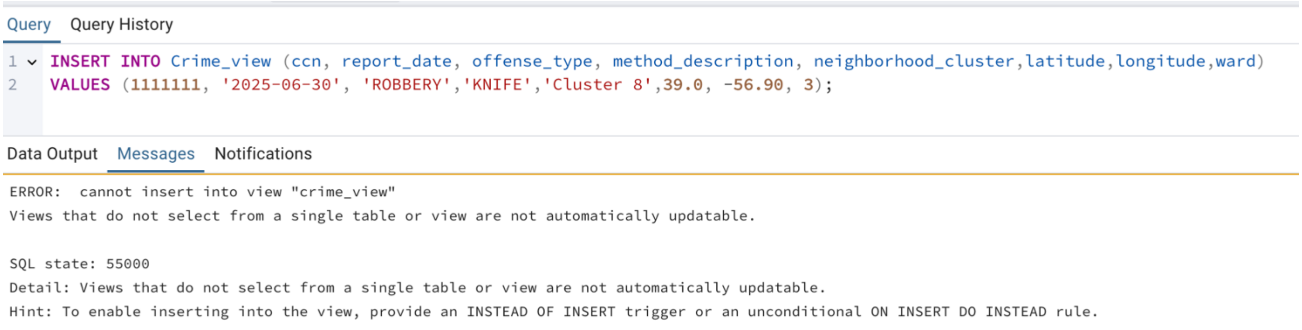


```sql
SELECT *
FROM Crime_view
LIMIT 20;
```

	ccn character varying (20)	report_date timestamp without time zone	offense_type character varying (100)	method_description character varying (100)	neighborhood_cluster character varying (100)	latitude numeric (9,6)	longitude numeric (9,6)	ward character varying (10)
1	24000457	2024-01-02 07:28:26	BURGLARY	OTHERS	Cluster 7	38.913711	-77.023968	2.0
2	24000745	2024-01-02 15:15:19	MOTOR VEHICLE THEFT	OTHERS	Cluster 7	38.908574	-77.023456	2.0
3	24001760	2024-01-04 14:57:40	THEFT F/AUTO	OTHERS	Cluster 7	38.912605	-77.022945	2.0
4	24001782	2024-01-04 15:38:41	MOTOR VEHICLE THEFT	OTHERS	Cluster 7	38.910385	-77.022942	2.0
5	24002167	2024-01-05 04:52:55	THEFT F/AUTO	OTHERS	Cluster 7	38.911123	-77.022433	2.0
6	24003708	2024-01-08 13:53:50	MOTOR VEHICLE THEFT	OTHERS	Cluster 7	38.912461	-77.021918	2.0
7	24004359	2024-01-09 20:08:07	THEFT F/AUTO	OTHERS	Cluster 7	38.908573	-77.026512	2.0
8	24004502	2024-01-10 01:44:20	THEFT/OTHER	OTHERS	Cluster 7	38.907238	-77.022432	2.0
9	24006974	2024-01-14 19:07:17	THEFT/OTHER	OTHERS	Cluster 7	38.909110	-77.023969	2.0
10	24007437	2024-01-15 18:02:23	THEFT/OTHER	OTHERS	Cluster 7	38.913348	-77.021915	2.0
11	24008377	2024-01-17 21:44:58	THEFT/OTHER	OTHERS	Cluster 7	38.910384	-77.021917	2.0
12	24008897	2024-01-18 21:26:04	THEFT/OTHER	OTHERS	Cluster 7	38.913348	-77.021915	2.0
13	24009631	2024-01-20 09:21:04	THEFT/OTHER	OTHERS	Cluster 7	38.912606	-77.023456	2.0
14	24010875	2024-01-23 01:33:19	THEFT/OTHER	OTHERS	Cluster 7	38.912605	-77.022431	2.0
15	24012325	2024-01-25 21:03:46	THEFT F/AUTO	OTHERS	Cluster 7	38.907909	-77.023966	2.0
16	24013283	2024-01-28 02:05:04	THEFT F/AUTO	OTHERS	Cluster 7	38.912605	-77.026512	2.0
17	24013333	2024-01-27 11:45:57	THEFT F/AUTO	OTHERS	Cluster 7	38.912606	-77.023456	2.0

Figure 12: Glimpse of the new Crime_view Table

3.3.Updating the View Table



```sql
INSERT INTO Crime_view (ccn, report_date, offense_type, method_description, neighborhood_cluster,latitude,longitude,ward)
VALUES (1111111, '2025-06-30', 'ROBBERY','KNIFE','Cluster 8',39.0, -56.90, 3);
```

```
ERROR:  cannot insert into view "crime_view"
Views that do not select from a single table or view are not automatically updatable.

SQL state: 55000
Detail: Views that do not select from a single table or view are not automatically updatable.
Hint: To enable inserting into the view, provide an INSTEAD OF INSERT trigger or an unconditional ON INSERT DO INSTEAD rule.
```

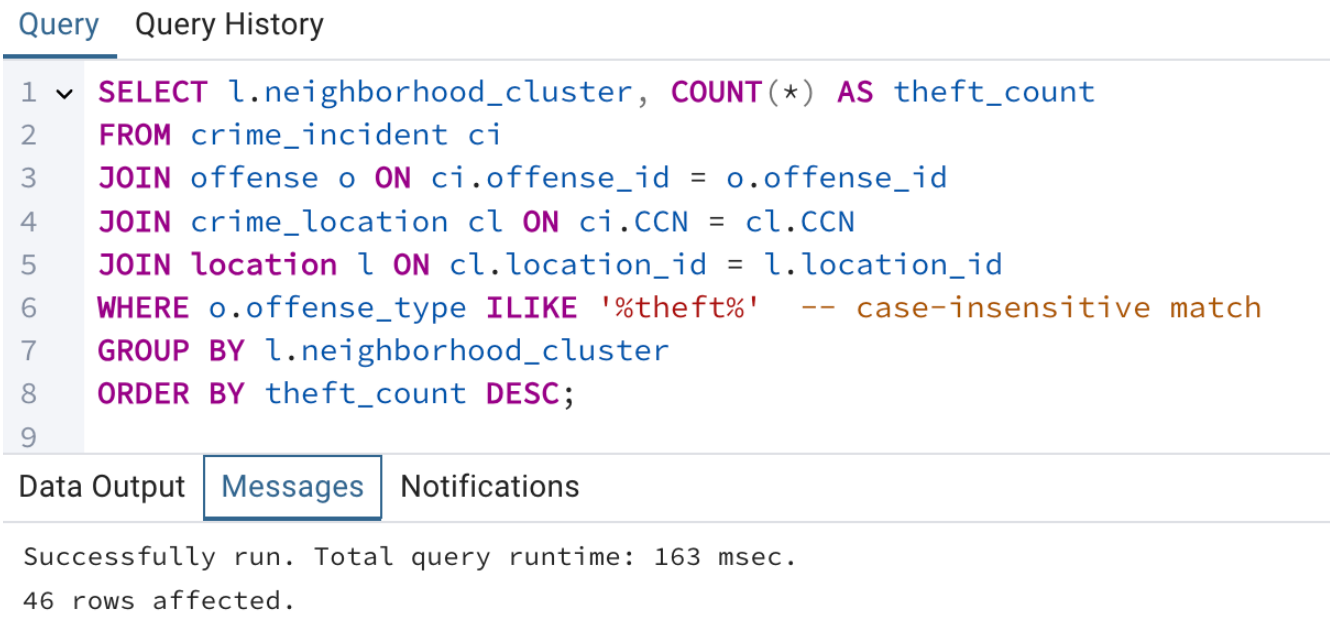
Figure 13: Updating View Table

The update view table query does not work and returned error; This is because the updated view cannot be inserted as it involves joins, and these are typically not updatable because the database cannot determine how to properly distribute new data across the base tables.

4. Indexing

4.1. Without indexing

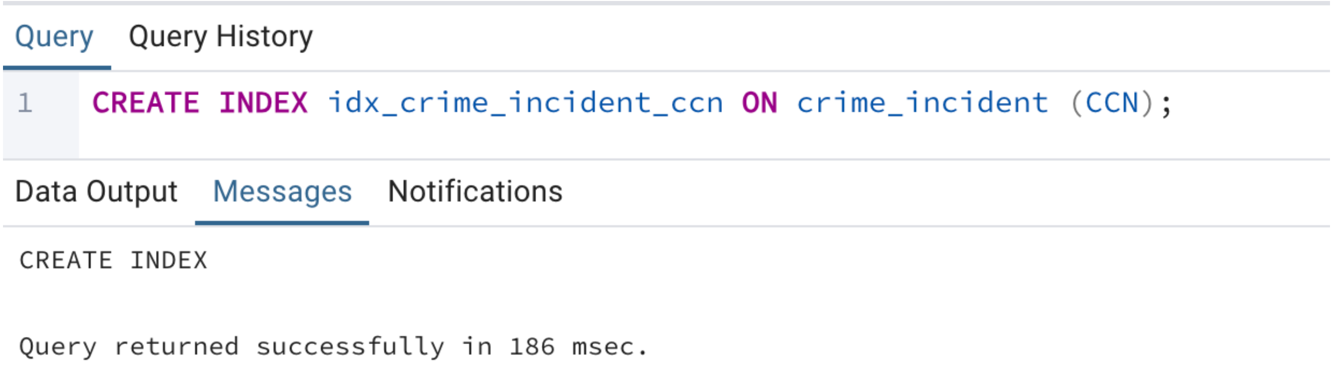
For the below query, the query time was 163 msec.



```sql
SELECT l.neighborhood_cluster, COUNT(*) AS theft_count
FROM crime_incident ci
JOIN offense o ON ci.offense_id = o.offense_id
JOIN crime_location cl ON ci.CCN = cl.CCN
JOIN location l ON cl.location_id = l.location_id
WHERE o.offense_type ILIKE '%theft%'  -- case-insensitive match
GROUP BY l.neighborhood_cluster
ORDER BY theft_count DESC;

```

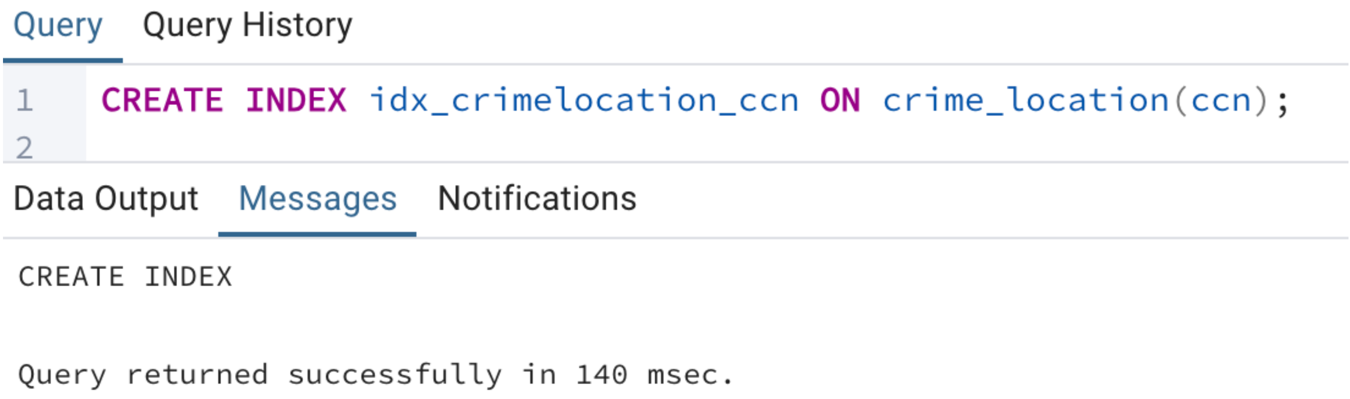
```
Successfully run. Total query runtime: 163 msec.
46 rows affected.
```

4.2.Creating index



```sql
CREATE INDEX idx_crime_incident_ccn ON crime_incident (CCN);
```

```
CREATE INDEX

Query returned successfully in 186 msec.
```

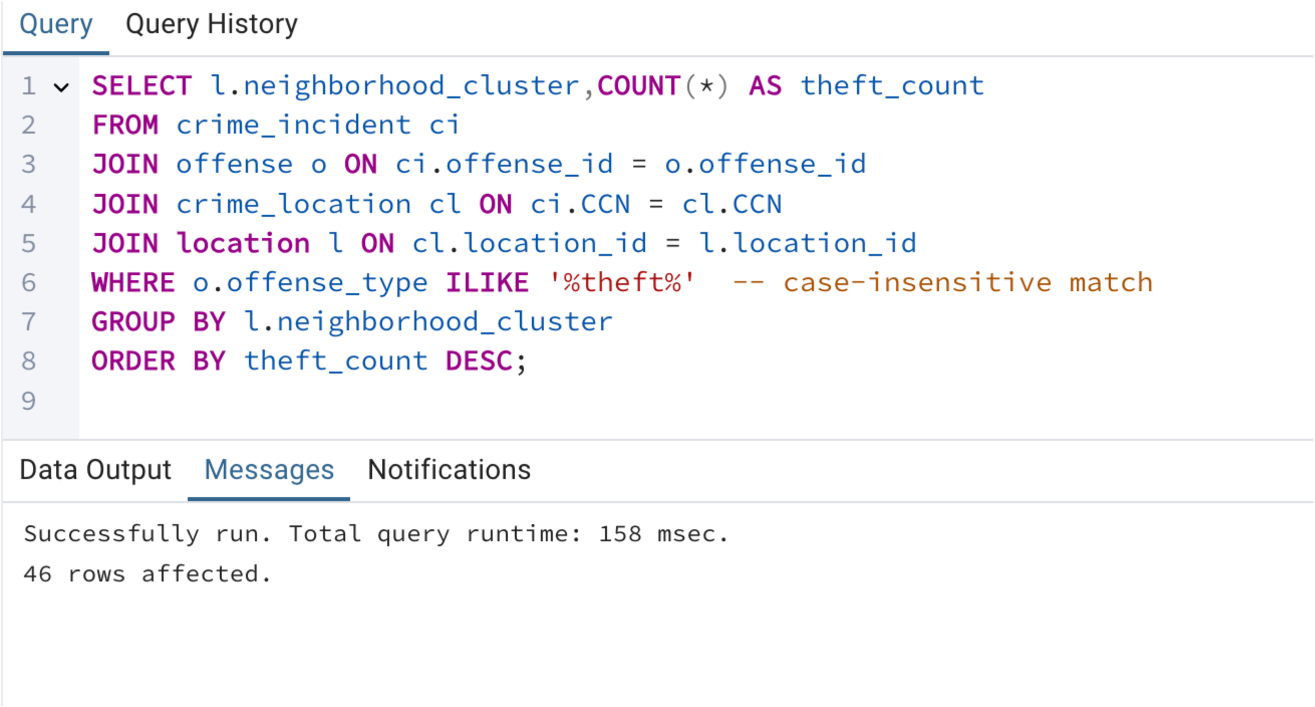


```sql
CREATE INDEX idx_crimelocation_ccn ON crime_location(ccn);

```

```
CREATE INDEX

Query returned successfully in 140 msec.
```

4.3.With indexing

With indexing, the query time dropped to 158 msec



```sql
SELECT l.neighborhood_cluster,COUNT(*) AS theft_count
FROM crime_incident ci
JOIN offense o ON ci.offense_id = o.offense_id
JOIN crime_location cl ON ci.CCN = cl.CCN
JOIN location l ON cl.location_id = l.location_id
WHERE o.offense_type ILIKE '%theft%'  -- case-insensitive match
GROUP BY l.neighborhood_cluster
ORDER BY theft_count DESC;

```

```
Successfully run. Total query runtime: 158 msec.
46 rows affected.
```