Checkpoint 1: The Spectacular Sailors

By: Pengyi Shi, Milan McGraw, and Justin Chae

Project Title: Network Graphs of Arresting Officers

Date: 6 October 2020

Note: The code provided in this PDF is maintained in our GitHub repository at https://github.com/Northwestern-Data-Sci-Seminar/Invisible-Institute-Chicago-Reporter-Collaboration-Public/tree/master/The%20Spectacular%20Sailors/Checkpoint_1

THEME

As provided by class, police are known to work in “crews” which are defined as "tight-knit community of officers involved in high levels of egregious misconduct and criminal activity." Four categories help define the four characteristics of a crew to include frequency, exclusivity, severity, and cohesion. A common theme among the four crew characteristics is repeated misconduct. In other words, bad cops do bad things together frequently.

The theme of this project is to improve how crew membership is defined by analyzing complaint data and performing network analysis. At a high-level, the tasks include (1) Relational Analytics: analyze which officers are co-accused in complaints, (2) Relational Analytics: implement network analysis to confirm clusters and represent network memberships in SQL, (3) Visualization: determine whether geospatial references in complaints and beats can help identify clusters.

Summary of Checkpoint 1

For Checkpoint 1, our team explores the dataset with SQL in three general areas. First, we craft a query that attempts to identify crew membership by combining data allegations and officers that are co-accused. Second, we learn technical aspects of how to create and represent network graphs in SQL tables and queries. Third, and lastly, we explore how to query and join tables with address or geospatial data to identify where crews operate.

Questions and Queries for Checkpoint 1

1. What officers, by officer_id, are disciplined for the same complaint, where “same complaint” is defined as having the same beat, on the same incident date, and same CRID? Within similar criteria, what officers are not disciplined?

```
SELECT distinct a.id, c.crid, count(c.crid) over (partition by crid) as
team_count, c.beat_id, d.name as location_area, c.incident_date,
a.allegation_count,c.coaccused_count
FROM data_officer a
LEFT JOIN data_officerallegation b
ON a.id = b.officer_id
LEFT JOIN data_allegation c
ON b.allegation_id = c.crid
LEFT JOIN data_area d
ON d.id = c.beat_id
WHERE b.disciplined ='true'
group by a.id, c.crid, d.name, a.allegation_count
order by crid asc

DROP TABLE IF EXISTS officers_grouped_allegations
CREATE TEMP TABLE officers_grouped_allegations
AS (

SELECT distinct a.id, c.crid, count(c.crid) over (partition by crid) as
team_count, c.beat_id, d.name as location_area, c.incident_date,
a.allegation_count,c.coaccused_count
FROM data_officer a
LEFT JOIN data_officerallegation b
ON a.id = b.officer_id
LEFT JOIN data_allegation c
ON b.allegation_id = c.crid
LEFT JOIN data_area d
ON d.id = c.beat_id
WHERE b.disciplined = 'true'
group by a.id, c.crid, d.name, a.allegation_count
order by crid asc);
```

The initial results of the query provide some insight into how officer teams are disciplined together. The query yields officer ID, crid (for the specific record used to discipline officers), team_count (how many officers were disciplined using the same crid or incident report), then the query also evaluates if the beat_id, location and incident_date all match, lastly the query provides some additional information on how many past allegation_counts’s and coaccused_counts these officers have received in the past to provide some additional insight into officer behavior.

Example output from the crews and teams query.

#	id	crid	team_count	beat_id	location_area	incident_date	allegation_count	coaccused_count
3361	485	1089358	1	72	1831	2018-05-04 00:00:00+00	3	2
3366	33015	1089814	1	61	1213	2018-06-10 00:00:00+00	1	1
3365	26808	1089740	1	231	834	2018-06-04 00:00:00+00	23	1
3364	22423	1089618	1	82	114	2018-05-27 00:00:00+00	18	1
3363	18732	1089610	1	135	1131	2018-05-26 00:00:00+00	5	1
3362	14674	1089386	1	130	1231	2018-05-02 00:00:00+00	11	1
3360	13256	1089237	1	201	725	2018-04-25 00:00:00+00	24	1
3358	510	1089065	1	106	222	2018-04-11 00:00:00+00	4	1

2. Within similar criteria, what officers are not disciplined?

```
SELECT distinct a.id, c.crid, count(c.crid) over (partition by crid) as
team_count, c.beat_id, d.name as location_area, c.incident_date,
a.allegation_count,c.coaccused_count
FROM data_officer a
LEFT JOIN data_officerallegation b
ON a.id = b.officer_id
LEFT JOIN data_allegation c
ON b.allegation_id = c.crid
LEFT JOIN data_area d
ON d.id = c.beat_id
WHERE b.disciplined <> 'true'
group by a.id, c.crid, d.name, a.allegation_count
order by crid asc

DROP TABLE IF EXISTS officers_grouped_allegations
CREATE TEMP TABLE officers_grouped_allegations
```

```
AS (

SELECT distinct a.id, c.crid, count(c.crid) over (partition by crid) as
team_count, c.beat_id, d.name as location_area, c.incident_date,
a.allegation_count,c.coaccused_count
FROM data_officer a
LEFT JOIN data_officerallegation b
ON a.id = b.officer_id
LEFT JOIN data_allegation c
ON b.allegation_id = c.crid
LEFT JOIN data_area d
ON d.id = c.beat_id
WHERE b.disciplined <> 'true'
group by a.id, c.crid, d.name, a.allegation_count
order by crid asc);
```

3. If given a list of officer_ids that are known or suspected to be associated with each other, how can the relationships between associated officer_ids be represented in SQL as queries and tables?

Summary of Query 2: A technical challenge for this project's theme is to represent crews as network graphs in SQL tables and queries. As part of Checkpoint 1, our team explores how to build queries to accomplish the technical challenge.

For example, if given a list of crew members from our first set of queries, the following code attempts to build a working knowledge of how to create tables of nodes and edges that can be queried for network membership and later, to be represented as a visualization of networks. A stretch goal for this sub-project is to potentially identify the interrelationships between crews that may not be readily apparent in identifying suspected crews.

As our team has no prior experience in creating network graphs in SQL, we leverage SQL code from two sources to build our first attempt. References are cited in SQL code comments below.

```
-- For Query 2: A technical exploration to represent "crews" as network graphs
of officers in SQL tables
-- Run queries, in order, between commented sections

-- start from a sample query from Query 1

DROP TABLE IF EXISTS officers_grouped_allegations
CREATE TEMP TABLE officers_grouped_allegations
AS (
SELECT a.id, b.disciplined, b.allegation_category_id, c.crid, c.beat_id,
d.name, c.incident_date
```

```
FROM data_officer a
LEFT JOIN data_officerallegation b
ON a.id = b.officer_id
LEFT JOIN data_allegation c
ON b.allegation_id = c.crid
LEFT JOIN data_area d
ON d.id = c.beat_id
WHERE disciplined = 'true');

-- view the officers_grouped_allegations table
SELECT * FROM officers_grouped_allegations;

-- filter officers_grouped_allegations from Query 1 to identify a sample group
of officers (for demo purposes only)
-- backlog: Filter and group by officers having the same CRID
DROP TABLE IF EXISTS officer_samples;
CREATE TEMP TABLE officer_samples
AS (
   SELECT id, crid, allegation_category_id
   FROM officers_grouped_allegations
   WHERE crid = 'C260174');

-- view officer_samples
SELECT * FROM officer_samples;

-- create a temp table structure to capture officer associations as graphs
-- Reference 1:
https://inviqa.com/blog/storing-graphs-database-sql-meets-social-network

-- create an empty nodes_temp table
DROP TABLE IF EXISTS nodes_temp CASCADE
CREATE TEMP TABLE nodes_temp (
   id INTEGER PRIMARY KEY,
   crid_id VARCHAR(10) NOT NULL,
   allegation_category_id VARCHAR(10));

-- create an empty edges_temp table
DROP TABLE IF EXISTS edges_temp;
CREATE TEMP TABLE edges_temp
(
   a INTEGER NOT NULL REFERENCES nodes_temp (id) ON UPDATE CASCADE ON DELETE
CASCADE,
   b INTEGER NOT NULL REFERENCES nodes_temp (id) ON UPDATE CASCADE ON DELETE
CASCADE,
   PRIMARY KEY (a, b)
);

DROP INDEX IF EXISTS a_idx;
DROP INDEX IF EXISTS b_idx;
```

```sql
-- create indexes for edges
CREATE INDEX a_idx ON edges_temp (a);
CREATE INDEX b_idx ON edges_temp (b);

-- view an empty edges_temp table
SELECT * FROM edges_temp;

-- view an empty nodes_temp table
SELECT * FROM nodes_temp;

-- populate graph table from a table that contains officers in the same crew
-- the officer_sample table is used for this checkpoint
INSERT INTO nodes_temp (id, crid_id, allegation_category_id)
SELECT id, crid, allegation_category_id
FROM officer_samples;

-- view a populated nodes_temp table
SELECT * FROM nodes_temp;

-- Initial attempt to create pairs of officers as edges
-- Reference 2:
https://stackoverflow.com/questions/36694641/sql-server-select-pairs-of-values
-from-one-column/36696098
INSERT INTO edges_temp (a, b)
WITH cte AS(
SELECT
   id as a,
   LEAD(id, 1, NULL) OVER (ORDER BY id) AS b
FROM officer_samples)
SELECT a, b
FROM cte
WHERE b IS NOT NULL
ORDER BY a;

-- view the populated edges_temp table
SELECT * FROM nodes_temp;

-- Traverse and view graph (Reference 1)
-- Backlog: address the null pair
SELECT *
FROM nodes_temp n
LEFT JOIN edges_temp e ON n.id = e.b;

-- TODO
-- Create a reliable query to create nodes and edges from a list of
officer_ids suspected to be in the same 'crew'
-- Adjust the node identity columns to contain relevant data
-- Integrate a visualization of the table graphs
```

Example output of network graph queries:

#	id	crid_id	allegation_category_id	a	b
1	27740	C260174	208	9517	27740
2	30737	C260174	208	27740	30737
3	32294	C260174	208	30737	32294
4	9517	C260174	208	NULL	NULL

4. What officers work in what beat with the highest number of complaints by year?

Summary of Query 4: We want to get the specific number of complaints and their corresponding officer ids by area and by time. As a result, we merge data_officer to get officers' information, and join the rest of the tables to connect useful geographical information related to each allegation.

The table from the query is ordered by beat_id and incident_date. In beat_id, we find a policing term which identifies the patrol area, and incident_date is the date when the incident happened.

Below is a screenshot from the query. We can see from the table that although we cannot have the direct geographical location of the corresponding beat_id, we do have the street address for the later analysis for a geospatial representation. With the street address, we can even cluster our data of complaints to different zip codes. We may be able to use the incident_date, to quantify and group the number of complaints in each beat area per time period. However, as you may notice, there are some addresses with not NULL and no empty values, so it raises a future question about data cleaning, i.e. whether to drop the data or derive other data to fill.

```
SELECT a.id, c.crid, c.beat_id, d.name, c.incident_date, c.add1, c.add2,
c.old_complaint_address
```

```
FROM data_officer a
LEFT JOIN data_officerallegation b
ON a.id = b.officer_id
LEFT JOIN data_allegation c
ON b.allegation_id = c.crid
LEFT JOIN data_area d
ON d.id = c.beat_id
WHERE c.add2 IS NOT NULL or c.old_complaint_address IS NOT NULL and c.beat_id
is not NULL
ORDER BY c.beat_id, c.incident_date
```

Example output of geospatial focus queries:

#	id	crid	beat_id	name	incident_date	add1	add2	old_complaint_address
120375	5144	C184507			1919-05-25 00:00:00+00			22XX N LOREL
120376	6307	C184507			1919-05-25 00:00:00+00			22XX N LOREL
120377	8860	C184507			1919-05-25 00:00:00+00			22XX N LOREL
120378	24344	C184507			1919-05-25 00:00:00+00			22XX N LOREL
120379	12612	C184507			1919-05-25 00:00:00+00			22XX N LOREL
120380	3331	C184507			1919-05-25 00:00:00+00			22XX N LOREL
120381	30848	C187656			1919-09-26 00:00:00+00			** E WASHINGTON
120382	6906	C188797			1919-11-16 00:00:00+00			91XX S LUELLA
120383	267	C202088			1930-02-01 00:00:00+00			017TH DIST.