DBMS Final Project

Spine Failure

Group Members and CWIDs:

Thomas McInnes (10901477),

Thomas Dowd (10900152),

Andrew Nicola (10899694),

Dmitry Weakly (10915244),

Hannah Clark (10875959)

Our Database – Denver Crime

The dataset we selected is a collection of Denver crimes and their metadata, including information about the neighborhood the crime occurred in, the date it was recorded, and so on. Specific crimes shown in this set vary from disturbing the peace to motor vehicle theft.

This dataset appeared interesting to us because it has the potential to give an insightful look into the crime and safety of a very local area, making it relevant to us as Mines students. The dataset is also simple to understand, since most people will be familiar with the crimes shown within the data, so clarifying its significance will be made simpler without having to give long-winded explanations in our report. However, due to the abundance of information within the dataset itself, there's a lot of potential for interesting analysis.

The data was obtained from the Denver Open Data Catalogue. This catalogue has a vast amount of data available, from data on transportation to data on Denver's environment. It is available under a Creative Commons Attribution 3.0 license, which states that the data is free to be shared and adapted so long as proper attribution is given. The citation for our dataset is provided on the works cited page at the end of the document.

To work with this data in SQL, we created three tables; incident, containing general incident information, offense, containing general offense (crime) information, and location, containing information about the various crime locations. The tables contain the following columns:

Incident:

incident_id: Unique identifier to track each incident report.

reported_date: Date and time when incident was reported.

incident_address: The address of where the crime occurred.

is_traffic: A boolean value for if it is a traffic violation or not.

victim_count: The number of victims of the crime.

Offense:

offense_id: Unique identifier tracking reported offenses.

offense_code: Numeric identifier for specific crime types.

offense_code_extension: Unspecified relation to seemingly clarify offense codes.

offense_type_id: Specific descriptor for incident.

offense_category_id: General descriptor for incident.

first_occurrence_date: Date and time this offense was noted to have first occurred.

last_occurrence_date: Date and time this offense was noted to have last occurred.

Location:

location_id: Unique value for each location where an incident occurred.

geo_lon: The longitudinal location of where the crime occurred.

geo_lat: The latitudinal location of where the crime occurred.

district_id: Categorical identifier based on district where crime occurred.

precinct_id: Categorical identifier of police precinct that dealt with incident.

neighborhood_id: The identifier for neighborhood for the incident.

x: x-coordinate for incident location.

y: y-coordinate for incident location

Below is an ERD created to represent these tables and their relationships:

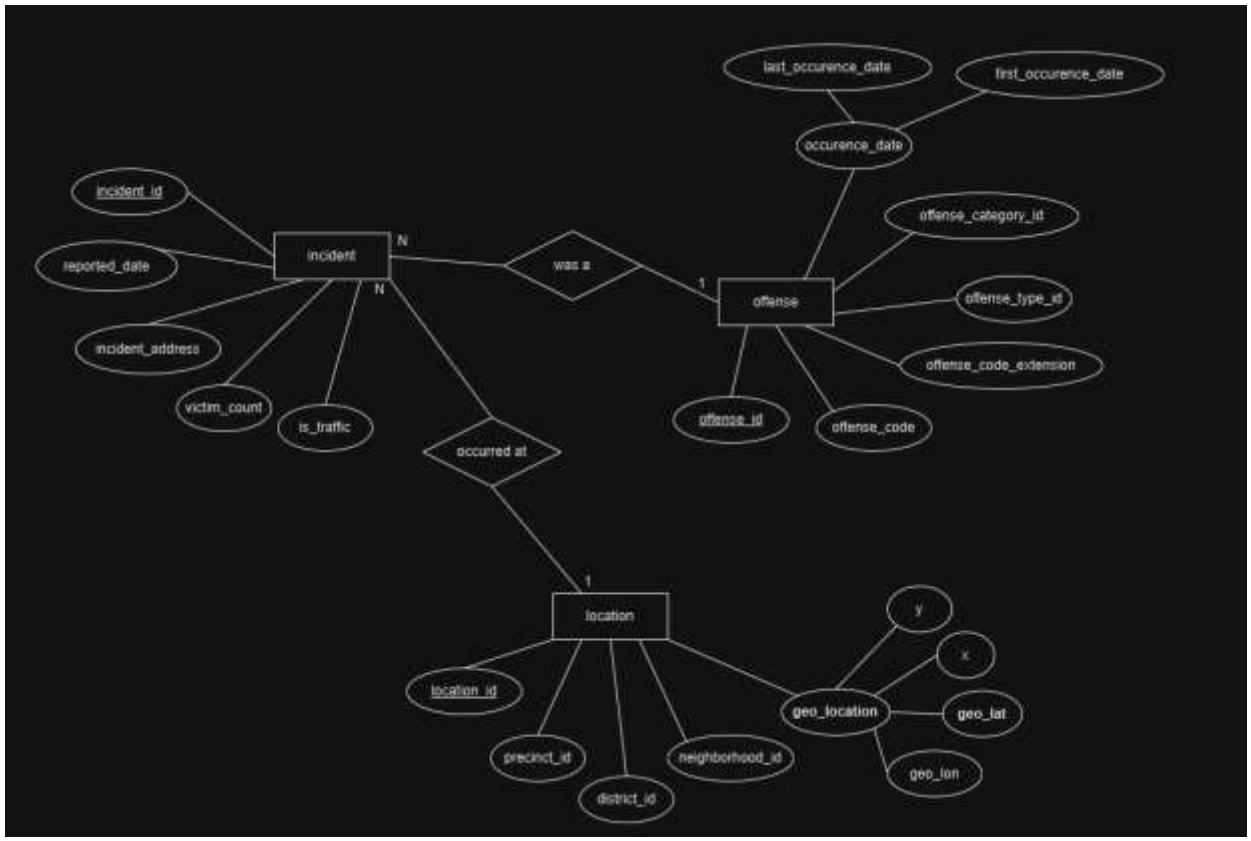

Figure 1*: ERD describing splits of initial dataset*

Of course, before we created any of these tables, we ensured the data was cleaned up. We did this by removing any empty rows or rows with missing values. We then had to transform the data from the way it was provided, as all of the described attributes were

present in the form of one giant CSV. To do this, we split the data into 3 different CSV files, each corresponding to a different table from the above ERD. This is how we cleaned and prepared our data for analysis.

For the sake of testing and some later analysis, we also created one extra table, called full_crime_data, which included all the previously mentioned columns in one table. This was only done to support the first question we had already written and analyzed before the project evolved into having multiple tables.

Analysis

Question 1

The first question we explored via our database was which neighborhoods had the highest frequency of non-business-related robberies. The correlation between Denver neighborhoods and non-business-related robberies interested us as students that all live near and frequent the Denver area. To answer this question, we wrote a query that first filtered through all crime incidents and strained out only incidents that involved a robbery or burglary not committed on a business and where the neighborhood was not a null or blank value. This filtered data was made into its own table, to be used for the following queries. The code to create this new table is featured here:

```
CREATE TABLE relevant_crimes_rq1 AS SELECT *

FROM full_crime_data WHERE

 offense_type_id ILIKE 'burglary-residence-%'

OR offense_type_id ILIKE 'robbery-street-%'

OR offense_type_id ILIKE 'robbery-residence'

OR offense_type_id ILIKE 'burglary-other'

AND neighborhood_id IS NOT NULL

AND neighborhood_id != '';
```

With this new table, we wrote a query that performed a union between the top five burglarized neighborhoods and the bottom five while also tallying any non-business-related robberies recorded in each. The result being a singular table featuring the top and bottom burglarized neighborhoods that yielded Five Points as being the neighborhood with the highest frequency of non-business-related robberies. This is not a surprising finding; however, it was still interesting to see the relationship between each neighborhood and this crime type. The table and graph of our results is below:

Table 1: Tally of Crime per Denver Neighborhood

neighborhood id	total crimes
five-points	848
central-park	664
capitol-hill	575
west-colfax	442
east-colfax	430
	10
auraria	16
wellshire	19
sun-valley	24
indian-creek	31

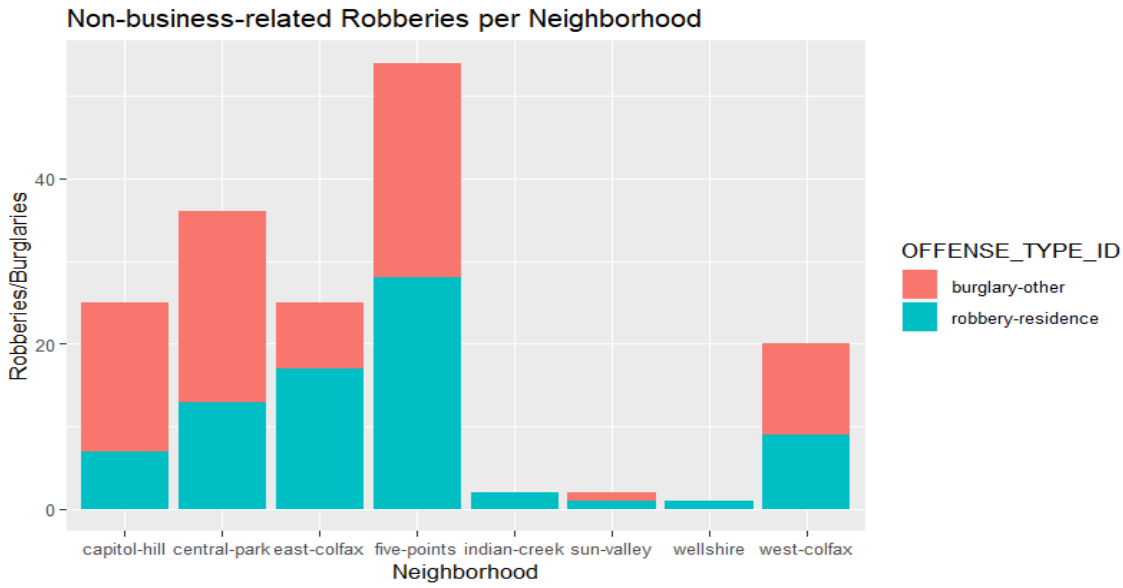

***Figure 2**: Stacked bar chart displaying the count of burglaries and robberies by neighborhood*

Question 2

Our next question revolved around discovering the proportions of each type of crime across all neighborhoods. Our code to answer this question is as follows:

```
SELECT offense_type_id,

COUNT(offense_type_id) * 1.0 / (SELECT COUNT(*) FROM offense) AS proportion

FROM offense

GROUP BY offense_type_id
```

```
ORDER BY proportion DESC limit 15;
```

As featured in our code, we calculated the proportion of each crime by retrieving the count of that specific crime across all neighborhoods and dividing it by the total number of all crimes. These individual proportions were grouped by their associated crime and printed out as a table in descending order. The resulting table is below:

Table 2: Proportion of Crimes across Denver Neighborhoods

offense type id	proportion
theft-of-motor-vehicle	0.15013302734941430139
theft-items-from-vehicle	0.10307539057081972588
criminal-mischief-mtr-veh	0.07027626817304903729
theft-parts-from-vehicle	0.06518214029026441550
theft-other	0.06001740073385703658
criminal-mischief-other	0.04191307198607941291
criminal-trespassing	0.03705599757902833293
assault-simple	0.03692990530470198091
theft-shoplift	0.03450893363763602204
theft-bicycle	0.02396257581297993872
burglary-business-by-force	0.02337498581461913829
weapon-unlawful-discharge-of	0.02161725951050979107
theft-from-bldg	0.02157186629175230434
burglary-residence-no-force	0.02155925706431966913
aggravated-assault	0.01826320501342882722

This helped us to find that theft of motor vehicles is the highest crime proportionally. It also yielded the proportions of the other 15 top crimes. It is a known fact that Denver is known for vehicle theft; however, learning that it is the highest crime proportionally was surprising. This table also yielded that theft, of multiple types, and criminal mischief are the top crimes when put in proportion to all other crimes committed in the area. Although it cannot necessarily be said that either discovery is surprising, it is interesting to see how much more likely one crime is to happen across some Denver neighborhoods compared to another. For example, criminal trespassing is almost twice as likely to occur compared to aggravated assault across Denver neighborhoods.

Question 3

Our next questions are focused on looking into the existence of time-based trends. Firstly, are there any monthly trends to the amount of crime throughout the year?

The query to answer this first question and a visualization is as follows:

```
SELECT TO_CHAR(reported_date, 'MM') AS month, COUNT(*) AS count
```

```
FROM incident

GROUP BY TO_CHAR(reported_date, 'MM')

ORDER BY month ASC;
```

***Figure 3**: Bar chart displaying total crimes by month*

This data suggests that there is a clear influx of crime during summer with a steep drop-off in winter and during the holidays. The peaks during summer may be due to more people being outdoors and unsupervised youths being on summer break. We also see that January has a surprisingly high number of crimes compared to the months around it, this could be caused by additional financial strain on individuals after the holiday season.

Question 4

Our second question regarding time-related trends: which crimes are most common in different months? The queries used to generate this plot are included in the appendix and the data is plotted below:

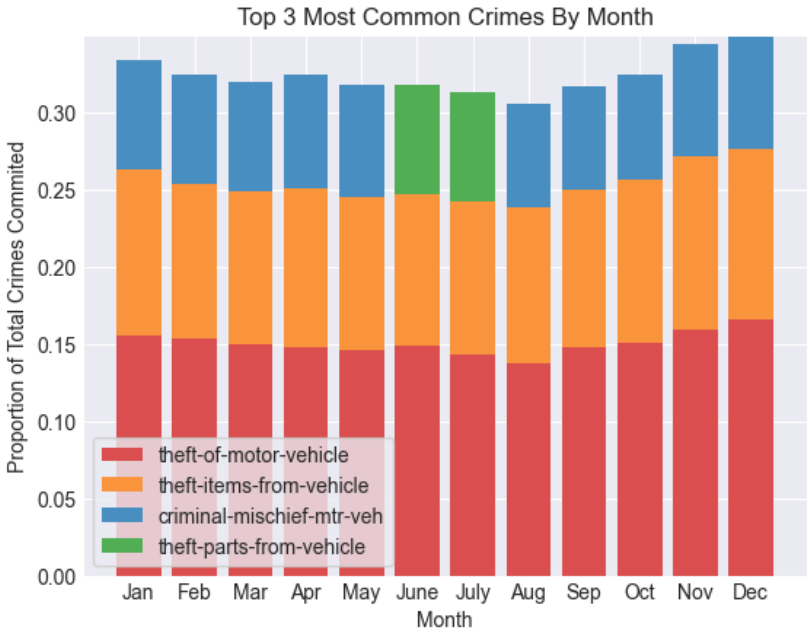

***Figure 4**: Bar chart displaying proportion of the 3 most common crimes by month*

This query closely matches the results we got earlier, reinforcing that generally theft regarding motor vehicles is the most common crime in the Denver area. One interesting thing to note is that there is a greater proportion of vehicle part theft during the summer than any other time. This may be due to warmer weather or people parking outside more often. We also see that in colder months, the total proportion of these vehicle-related crimes is higher, conversely indicating more varied crimes occur during the summer than during the winter.

Technical Challenges

- Obtaining, manipulating, loading the dataset
    - The data was fortunately quite simple to obtain, as the Denver Open Data Catalogue has a fairly open license and directly gave the ability to download the (uncleaned) CSV data. Manipulating the data was somewhat trickier; we chose to split the bulky CSV file into a cleaned version with many empty rows removed, then split that cleaned CSV into 3 smaller CSVs, one for each table in our ERD. The last problem we faced with loading the data was getting the location's ID column into the database. The data given on the website had no ID unique ID for each location, so we needed to create a new serial ID column

in the location table. This was done by loading the given data into the table with \COPY, then adding the serial ID column using ALTER TABLE. From there, all the rest of the data could be loaded using \COPY.

- The Web Visualization Application
    - The Python Flask library and pg8000 library were used to construct a web application to help filter out queries with a simple form. The main python script for the application is in Appendix B.
    - Some of the difficulty in the application came from issues with parsing form results to create good joins in our queries and we ended up using the full table for the web application since it was essentially constructing it each time anyways.
    - There was some difficulty with debugging issues with how the web library we used output data. This was resolved with learning the new logger that is a part of the flask library.
    - There were plans to fix and build out a graph creation tool (using an R interface library) by using more group by functions. These should be simpler changes that can be made in the future, but what we currently have is a good visualization tool that helps break down and select different parts of the query through the crime data.
    - As evidenced in the Appendix B code, there were repeated rewritings of different segments of the server script to have it function properly and parse query results as the tables have changed multiple times throughout its development.
- Visualizations and Tools Used
    - RStudio was used to produce some of the graphical representations correlating to our questions. RStudio is a good resource that proves useful for large sets of data. The only issue that occurred with using RStudio was determining the correct pipelines to filter out the desired data.
    - Additionally, Python's matplotlib library was used to generate some of the graphics. We faced similar challenges regarding parsing the outputs of our queries and displaying that, primarily relating to Analysis Question 4 as the categories of crimes were not consistent between months.
    - LaTeX was used for some of the tabular models. Title names including underscores and spacing were the biggest challenges we faced using LaTeX as it was not a tool any of us had much experience with; however, it is a tool that building experience with appears useful for its versatility.
- Data Wrangling
    - Data wrangling proved to be a bit of a challenge due to the size of the dataset. However, with a clever bit of work around we were able to complete the data

cleaning. We opted to further breakdown the massive full dataset into 3 smaller and cleaned .csv files. These later got turned into the 3 tables shown in the ERD.

- ERD to Relational Database
    - Since our ERD was not extremely complicated, it was mostly a matter of incorporating the relational foreign keys where needed within the incident table for the location and offense tables. Other than that, creating the schema from our ERD was trivial.

Appendix A

A1. Queries for Analysis Question 4:

```
CREATE TABLE monthly_crime_stats AS (
SELECT
        EXTRACT(MONTH FROM i.reported_date) AS month,
        o.offense_type_id,
        COUNT(i.incident_id) AS total_incidents
        FROM incident AS i
        JOIN offense AS o ON i.incident_offense_id = o.offense_id
        GROUP BY month, o.offense_type_id
);
CREATE TABLE monthly_total_incidents AS (
SELECT
        EXTRACT(MONTH FROM i.reported_date) AS month,
        COUNT(i.incident_id) AS total_count
        FROM incident AS i
        GROUP BY month
);
CREATE TABLE monthly_proportions AS (
SELECT
        mcs.month,
        mcs.offense_type_id,
        mcs.total_incidents,
        (mcs.total_incidents * 1.0 / mti.total_count) AS offense_proportion
        FROM monthly_crime_stats AS mcs
        JOIN monthly_total_incidents AS mti ON mcs.month = mti.month );
```

```
CREATE TABLE ranked_crimes AS (
SELECT
        month,
        offense_type_id,
        offense_proportion,
        ROW_NUMBER() OVER (PARTITION BY month ORDER BY offense_proportion DESC) AS rank
        FROM monthly_proportions
);
SELECT month, offense_type_id, offense_proportion
FROM ranked_crimes
WHERE rank <= 3
ORDER BY month, rank
```

Appendix B

This is the python flask web application code, html omitted.

```python
from flask import Flask, render_template, request, redirect, url_for, session
import pg8000
import uuid
import logging

app = Flask(__name__, template_folder='src')

# Set up basic logging
logging.basicConfig(level=logging.DEBUG)

# Global variables
global db
global signedIn
global query_results
query_results = {}
signedIn = False  # Default



@app.route('/', methods=['GET', 'POST'])
def home():
    logging.debug("Home route hit!")

    if request.method == 'POST':
        global signedIn
        global db
```

```
        username = request.form.get('username')
        password = request.form.get('password')

        logging.debug(f"Received username: {username}, password: {'*' *
len(password)}")

        try:
            db = pg8000.connect(user=username, password=password,
host='codd.mines.edu', port=5433, database='csci403')
            signedIn = True
            logging.debug("Successfully signed in.")
            return redirect(url_for('form'))
        except Exception as e:
            signedIn = False
            logging.error(f"Error connecting to the database: {e}")
            return redirect(url_for('error'))

    return render_template('index.html')


@app.route('/form', methods=['GET', 'POST'])
def form():
    if not signedIn:
        logging.warning("User not signed in, redirecting to home.")
        return redirect(url_for('home'))

    if request.method == 'GET':
        logging.debug("Rendering form.")
        return render_template('form.html')

    # Process form on POST request
    logging.debug("Processing POST request from form.")

    global db
    cursor = db.cursor()

    # Collect form inputs
    incident_id = request.form.get('incident_id')
    reported_date = request.form.get('reported_date')
    incident_address = request.form.get('incident_address')
    victim_count = request.form.get('victim_count')
    is_traffic = request.form.get('is_traffic')
    offense_id = request.form.get('offense_id')
```

```
    offense_code = request.form.get('offense_code')
    precinct_id = request.form.get('precinct_id')
    district_id = request.form.get('district_id')
    neighborhood_id = request.form.get('neighborhood_id')

    # Construct the query
    query = """
        SELECT
            fcd.incident_id, fcd.offense_id, fcd.incident_address,
fcd.reported_date,
            fcd.victim_count
        FROM
            full_crime_data fcd
    """
    conditions = []

    # Add conditions based on user input
    if incident_id:
        conditions.append(f"fcd.incident_id = {incident_id}")
    if reported_date:
        conditions.append(f"fcd.reported_date = '{reported_date}'")
    if incident_address:
        conditions.append(f"fcd.incident_address LIKE '%{incident_address}%'")
    if victim_count:
        conditions.append(f"fcd.victim_count = {victim_count}")
    if is_traffic:
        conditions.append(f"fcd.is_traffic = {'TRUE' if is_traffic == '1' else
'FALSE'}")
    if offense_id:
        conditions.append(f"fcd.offense_id = {offense_id}")
    if offense_code:
        conditions.append(f"fcd.offense_code_extension LIKE '%{offense_code}%'")
    if precinct_id:
        conditions.append(f"fcd.precinct_id LIKE '%{precinct_id}%'")
    if district_id:
        conditions.append(f"fcd.district_id = '{district_id}'")
    if neighborhood_id:
        conditions.append(f"fcd.neighborhood_id LIKE '%{neighborhood_id}%'")

    # Append WHERE clause if there are conditions
    if conditions:
        query += " WHERE " + " AND ".join(conditions)
```

```
    logging.debug(f"Constructed Query: {query}")

    try:
        cursor.execute(query)
        results = cursor.fetchall()
        column_names = [desc[0] for desc in cursor.description]

        # Store results with a unique key
        key = str(uuid.uuid4())
        query_results[key] = {'results': results, 'columns': column_names,
'query': query}

        logging.debug(f"Redirecting to results with key: {key}")
        return redirect(url_for('results', key=key))
        # This sends a GET request to /results
    except Exception as e:
        logging.error(f"Error executing query: {e}")
        return f"An error occurred: {e}"



@app.route('/results', methods=['GET'])
def results():
    key = request.args.get('key')
    logging.debug(f"Received key in results route: {key}")

    global query_results

    if key in query_results:
        logging.debug(f"Found data for key: {key}")
        data = query_results.pop(key)  # Remove data after access

        if not data['results']:
            logging.debug("No results found for the query.")
            return render_template('results.html', message="No data found for
your query.", query=data['query'])

        logging.debug(f"Displaying results for query: {data['query']}")
        return render_template('results.html', results=data['results'],
columns=data['columns'], query=data['query'])

    logging.warning(f"No data found for the provided key: {key}")
    return "No results to display."
```

```python
@app.route('/error')
def error():
    logging.error("An error occurred during database connection or query.")
    return render_template('error.html')


# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```

Works Cited

"Crime." Open Data Catalog, City and County of Denver, 22 Oct. 2019, opendata geospatialdenver.hub.arcgis.com/datasets/1e080d3ce2ae4e2698745a0d02345d4a_324/explore. Accessed 13 Nov. 2024.