San José Safe Zones

Final Report

Sanjana Thummalapalli

Contents

ABSTRACT **4**

1. INTRODUCTION **5**

1.1 Overview 5

1.2 Objectives 5

1.3 Dataset Description 6

2. DATABASE LOADING AND SETUP **7**

2.1 Database Creation and Configuration 7

2.2 Initial Data Loading 8

2.3 Setup Challenges and Solutions 8

3. DATA PROCESSING **9**

3.1 Data Quality Challenges 9

3.2 Data Cleaning Procedures 10

3.2.1 Whitespace Removal and Null Standardization 10

3.2.2 Dynamic Column Addition 10

3.2.3 Date Time Parsing 11

4. SCHEMA DESIGN AND NORMALIZATION **13**

4.1 Database Design Principles 13

4.2 Schema Architecture 14

4.2.1 Fact Table: fact_calls_2025 14

4.2.2 Dimension Table: dim_calltype 15

4.2.3 Dimension Table: dim_disposition 16

4.4 Data Integration and post-processing 16

4.5 ER Diagram 18

5. QUERY DEVELOPMENT AND RESULTS **20**

5.1 Basic Analytical Queries 20

Query 1: Daily Call Volume 20

Query 2: Calls by call type (Top 10) 21

Query 3: Average Priority by Disposition 21

Query 4: Call Volume by Zip Code 22

Query 5: Average response time by priority 22

5.3 Advanced Analytical Queries 23

Query 1: Hotspot Addresses by Severity (Top 15) 23
Query 2: 7 Day rolling trend of calls and severe share 23
Query 3: Show month-over-month call volume with running totals and percent change 24
Query 4: Create a reusable view for disposition performance analysis 25
Query 5: Indexing Optimization Analysis 26
Query 6: Predictive Pattern Analysis 28
Query 7: Percentile Analysis 30
Query 8: Cross-Tabulation 31
Query 9: Pareto analysis 32
Query 10: Incident Co-Occurance Analysis 33
6. VISUALIZATIONS AND INTERACTIVE DASHBOARD 34
6.1 Technology Stack 34
6.2 Visualization Descriptions 34
6.2.1 Visualization 1: Call Volume Trends & Running Totals 34
6.2.2 Visualization 2: Interactive Call Type Analysis 35
6.2.3 Visualization 3: Interactive Heat Map (Hour × Day) 36
6.2.4 Visualization 4: Interactive Risk Analysis Scatter Plot 37
6.2.5 Visualization 5: Response Time Percentiles 38
6.2.6 Visualization 6: Interactive Pareto Analysis 39
6.2.7 Visualization 7: Incident Chain Analysis 40
6.3 Dashboard deployment 40
7. CONCLUSION 42
8. REFERENCES 43

ABSTRACT

Safety is a critical concern for international students when choosing where to live, particularly in unfamiliar environments where information about high-risk versus safe areas is often unclear. This project addresses that challenge through a comprehensive database analysis of the San Jose Police calls for Service dataset, aiming to identify safer residential zones and provide actionable insights for housing decisions. The dataset, comprising thousands of police call records from 2025, was processed using advanced SQL techniques for data cleaning, normalization, and analytical querying. A normalized relational database schema was implemented following 3NF (Third Normal Form) principles, supported by complex fact-time parsing, data validation, and quality assurance procedures. Analytical methods included window functions, common table expressions (CTEs), recursive queries, and statistical analysis to uncover meaningful patterns in call distribution, hotspot identification, and incident concentration. To make these findings accessible, an interactive Streamlit dashboard was developed featuring various dynamic visualizations. The results highlight significant spatial and temporal patterns and provide intelligence for identifying safe zones, optimizing resource allocation, and supporting proactive community strategies.

1. INTRODUCTION

1.1 Overview

Law enforcement agencies generate massive volumes of operational data daily. The San Jose Police Department's Calls for Service dataset represents a rich source of information about public safety incidents, response patterns, and community needs. Yet, in its raw form, such operational data provides limited value without systematic organization, analysis, and interpretation.

This project addresses that challenge by transforming unstructured police call records into a structured, queryable database system capable of supporting strategic decision-making. Through the application of database normalization principles, advanced SQL analytics, and interactive visualization techniques, the project demonstrates how data engineering and analytics can uncover meaningful patterns, improve situational awareness, and enhance public safety operations.

1.2 Objectives

The primary objective of this project was to identify safe residential zones for international students by leveraging advanced SQL techniques and systematic database design. The goal of this project is:

- **Database design:** Develop a normalized relational database schema following 3NF to ensure data integrity, eliminate redundancy, and optimize query performance for large-scale police call records.

- **Data Quality:** implements comprehensive data cleaning and validation procedures to address inconsistent date-time formats, missing values, and other challenges inherent in operational datasets, ensuring reliable analysis.

- **Advanced Analytics:** construct SQL queries utilizing window functions, CTEs, and statistical methods to uncover actionable insights related to safety.

- **Visualization:** built-in interactive Streamlit dashboard with dynamic visualizations to make findings accessible and intuitive for others.

1.3 Dataset Description

The San Jose police Calls for Service dataset is published through the San Jose Open Data Portal and provides detailed records of public safety incidents reported to the San Jose Police Department. It is updated daily and contains thousands of entries from every year (the project particularly talks about calls in 2025), making it a rich source of information for analyzing community safety and law enforcement response patterns. (San Jose Police Calls Dataset).

COLUMN	**DESCRIPTION**
CDTS	Timestamp combining year, month, hour, second, and time zone
EID	internal control number of the event
START_DATE	Expected report creation date in the Open Data Portal
CALL_NUMBER	Event number
PRIORITY	Priority level of call (1-highest, 6-lowest)
REPORT_DATE	Date when the call center created the event in CAD(computer-aided dispatch)
OFFENSE_DATE	Date when the call center receives the event
OFFENSE_TIME	The time when the call center receives the call and creates the event
CALLTYPE_CODE	Code used for the event in CAD

CALL_TYPE	Definition of CAD type code
FINAL_DISPO_CODE	Final disposition code for CAD event
FINAL_DISPO	Explanation of disposition code
ADDRESS	100 block of street address of the event
CITY	City of the event
STATE	State of the event

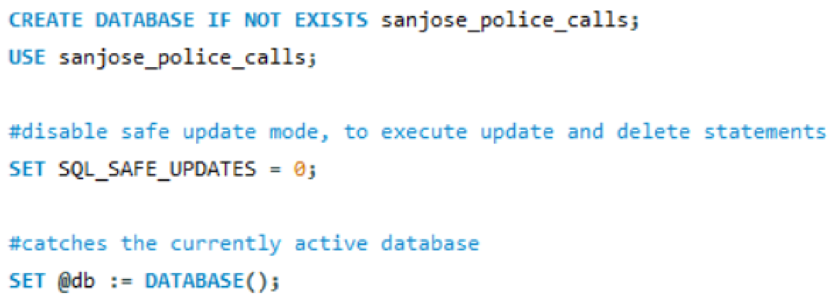
Table 1.3: Data Fields of the dataset

In its raw form, the dataset is unstructured and requires cleaning and normalization to ensure consistency and reliability. Once processed, it supports advanced SQL analytics for identifying temporal trends, hotspots, and more.

2. DATABASE LOADING AND SETUP

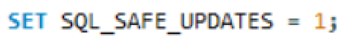
2.1 Database Creation and Configuration

The database setup process began with creating a dedicated database and configuring the MySQL environment for optimal data manipulation:



```sql
CREATE DATABASE IF NOT EXISTS sanjose_police_calls;
USE sanjose_police_calls;

#disable safe update mode, to execute update and delete statements
SET SQL_SAFE_UPDATES = 0;

#catches the currently active database
SET @db := DATABASE();
```

The **SQL_SAFE_UPDATES** setting was temporarily disabled to allow batch UPDATE and DELETE operations without restrictive key-based clauses, which was necessary for large-scale data cleaning. The **@db** variable captured the active database name for use in dynamic SQL queries. After completing data manipulation, **SQL_SAFE_UPDATES** was re-enabled to prevent accidental modifications



```sql
SET SQL_SAFE_UPDATES = 1;
```

2.2 Initial Data Loading

The raw San Jose Police Calls for Service dataset was imported into a staging table named policecalls2025 using MySQL's Table Wizard. This graphical import tool allowed for efficient loading of the CSV file into the database without requiring manual command-line operations.

Using the Table Wizard ensured that:

- All fields were initially stored as text, accommodating inconsistent formats and missing values.
- The original structure of the dataset was preserved for iterative cleaning and transformation.
- The import process was reproducible and transparent, supporting later validation steps.

This staging approach provided a safe environment for data cleaning and normalization before populating the final analytical schema.

2.3 Setup Challenges and Solutions

Date Format Ambiguity: Resolved by assuming U.S. standard MM/DD/YYYY format and validating against known ranges. **Memory Limitations:** Optimized large UPDATE operations with targeted WHERE clauses. **Character Encoding:** Enforced UTF-8 encoding at the database and connection levels. **Index Build Time:** Accepted one-time overhead since the indexes significantly improved ongoing query performance.

These challenges reflect common real-world data engineering scenarios and highlight practical problem-solving strategies.

3. DATA PROCESSING

3.1 Data Quality Challenges

During the exploratory analysis phase, several significant data quality issues were identified within the dataset. Addressing challenges was essential to ensure reliable insights and accurate downstream analysis.

1. **Inconsistent date-time formats:** temporal fields appeared in multiple formats, including MM/DD/YYYY HH: MM: SS AM/PM, MM/DD/YYYY HH: MM AM/PM, MM/DD/YYYY HH:MM: SS (24-hour format), MM/DD/YYYY HH: MM (24-hour format), and MM/DD/YYYY (date only). This required parsing logic to standardize all date-time values.

2. **Leading and trailing whitespaces:** text fields contained extraneous whitespace, which interfered with joins, comparisons, and grouping operations. Systematic trimming was applied to maintain consistency.

3. **Empty strinda and NULL values:** the dataset inconsistently used empty strings and NULL values. Empty strings are converted to NULL to enable proper handling in queries.

4. **Missing values:** several records lacked complete information, particularly in address and time fields. carfel null handling strategies were implemented to preserve analytical integrity.

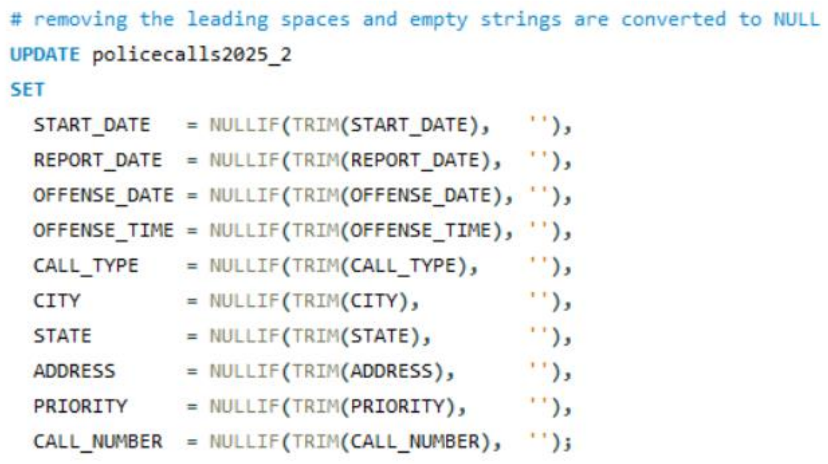
5. **Data type inconsistencies:** numeric fields such as PRORITY were stored as text, requiring type casting to support statistical and comparative analysis.

6. **Redundant Information:** Call types and dispositions were represented both as codes and full descriptions. This redundancy created opportunities for normalization and schema optimization.

3.2 Data Cleaning Procedures

3.2.1 Whitespace Removal and Null Standardization

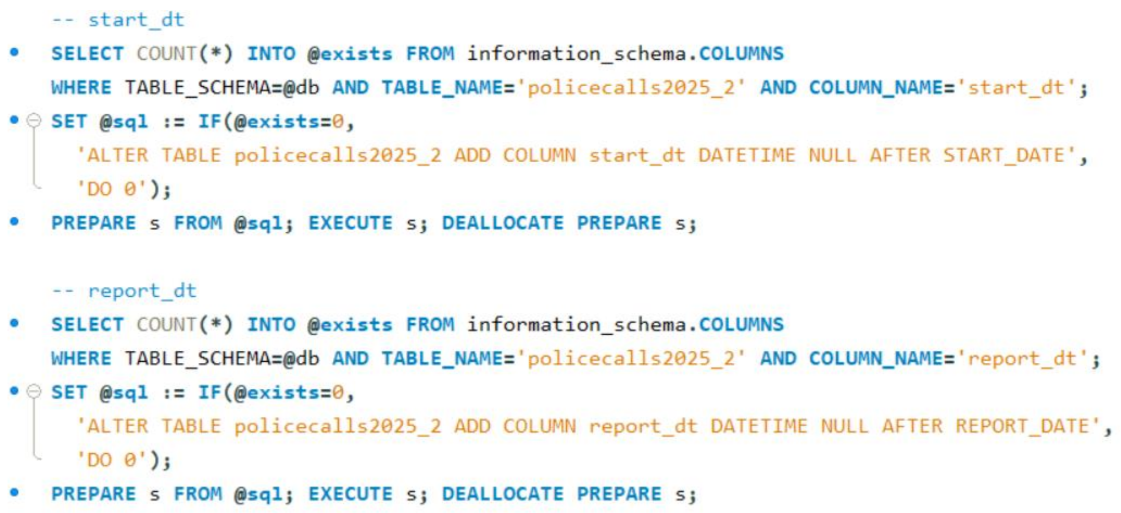
One of the first steps in cleaning the dataset was to standardize text fields by removing leading and trailing whitespace and converting empty strings into NULL values. This ensured consistency across joins, comparisons, and aggregations.



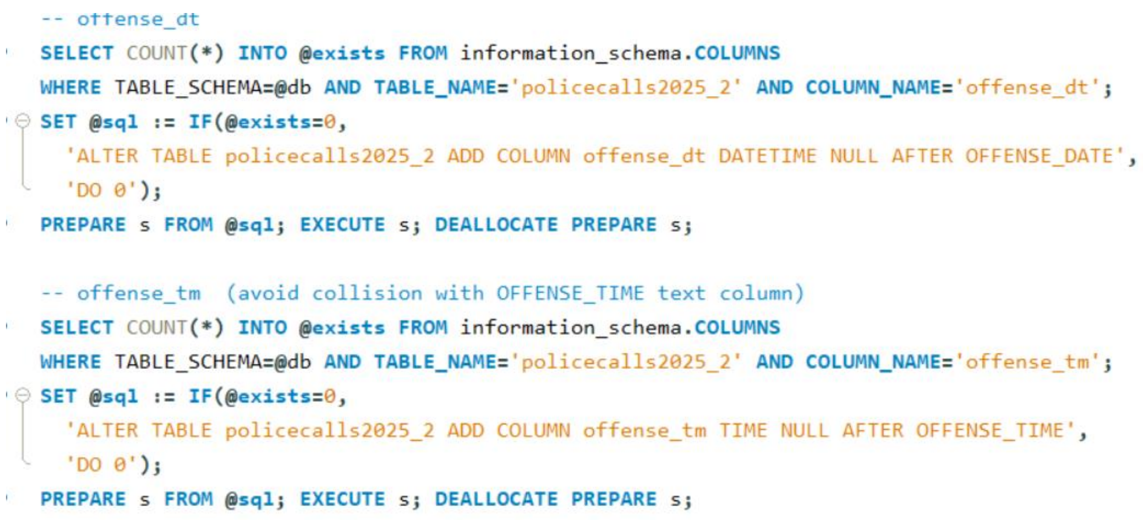
```sql
# removing the leading spaces and empty strings are converted to NULL
UPDATE policecalls2025_2
SET
  START_DATE   = NULLIF(TRIM(START_DATE),   ''),
  REPORT_DATE  = NULLIF(TRIM(REPORT_DATE),  ''),
  OFFENSE_DATE = NULLIF(TRIM(OFFENSE_DATE), ''),
  OFFENSE_TIME = NULLIF(TRIM(OFFENSE_TIME), ''),
  CALL_TYPE    = NULLIF(TRIM(CALL_TYPE),    ''),
  CITY         = NULLIF(TRIM(CITY),         ''),
  STATE        = NULLIF(TRIM(STATE),        ''),
  ADDRESS      = NULLIF(TRIM(ADDRESS),      ''),
  PRIORITY     = NULLIF(TRIM(PRIORITY),     ''),
  CALL_NUMBER  = NULLIF(TRIM(CALL_NUMBER),  '');
```

3.2.2 Dynamic Column Addition

To support structured analysis, additional columns were dynamically added to store parsed date-time values. This process involved checking if a column already existed before altering the table, ensuring schema integrity.



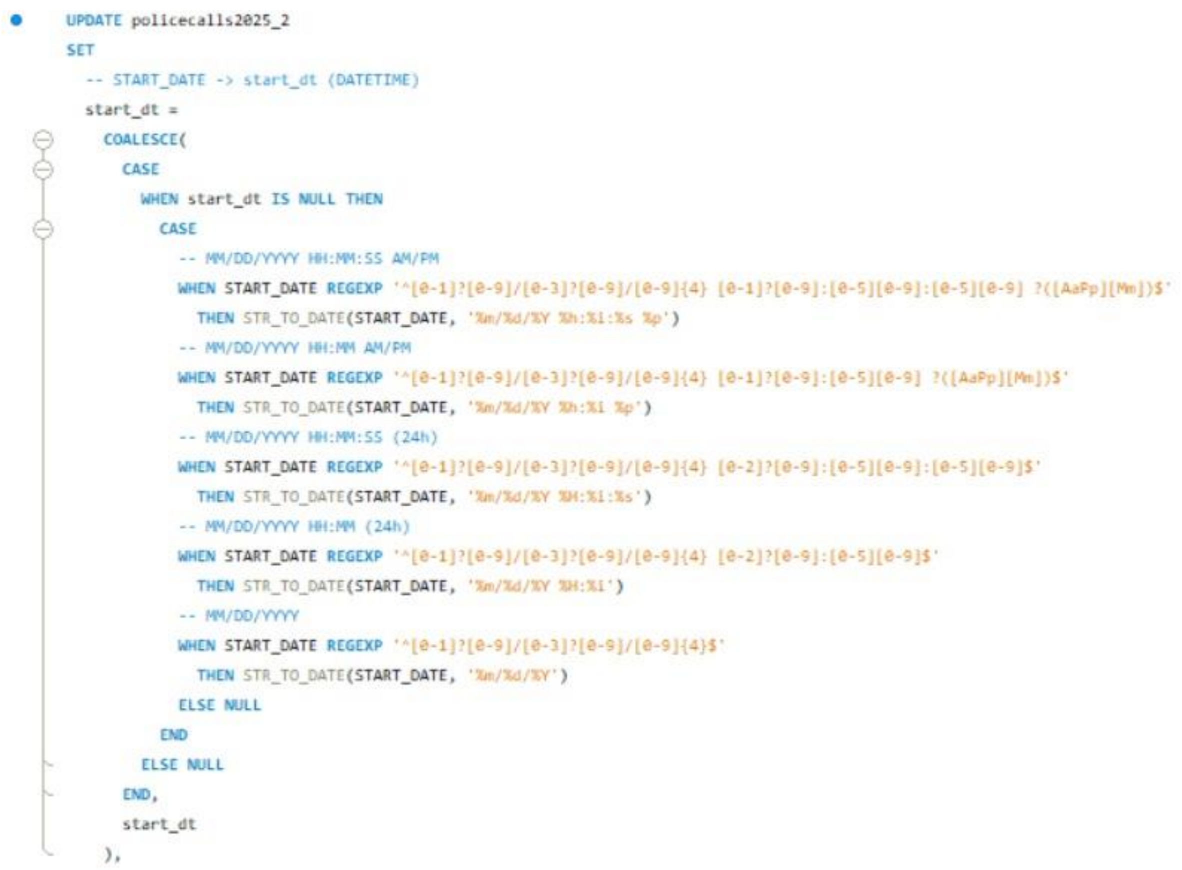
```sql
-- start_dt
SELECT COUNT(*) INTO @exists FROM information_schema.COLUMNS
WHERE TABLE_SCHEMA=@db AND TABLE_NAME='policecalls2025_2' AND COLUMN_NAME='start_dt';
SET @sql := IF(@exists=0,
  'ALTER TABLE policecalls2025_2 ADD COLUMN start_dt DATETIME NULL AFTER START_DATE',
  'DO 0');
PREPARE s FROM @sql; EXECUTE s; DEALLOCATE PREPARE s;

-- report_dt
SELECT COUNT(*) INTO @exists FROM information_schema.COLUMNS
WHERE TABLE_SCHEMA=@db AND TABLE_NAME='policecalls2025_2' AND COLUMN_NAME='report_dt';
SET @sql := IF(@exists=0,
  'ALTER TABLE policecalls2025_2 ADD COLUMN report_dt DATETIME NULL AFTER REPORT_DATE',
  'DO 0');
PREPARE s FROM @sql; EXECUTE s; DEALLOCATE PREPARE s;
```



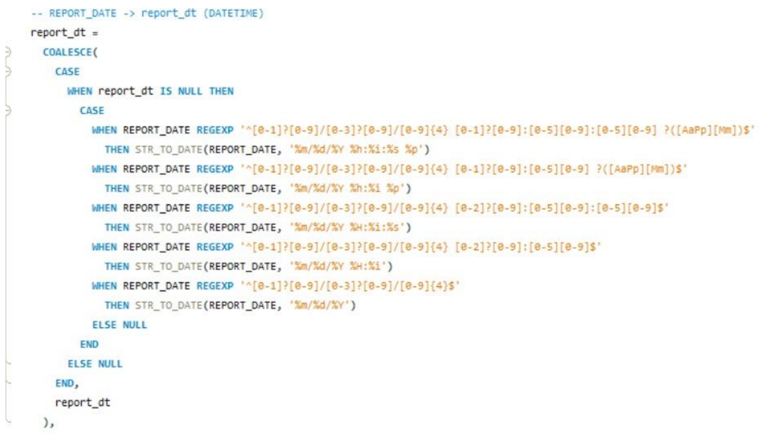
```sql
-- offense_dt
SELECT COUNT(*) INTO @exists FROM information_schema.COLUMNS
WHERE TABLE_SCHEMA=@db AND TABLE_NAME='policecalls2025_2' AND COLUMN_NAME='offense_dt';
SET @sql := IF(@exists=0,
  'ALTER TABLE policecalls2025_2 ADD COLUMN offense_dt DATETIME NULL AFTER OFFENSE_DATE',
  'DO 0');
PREPARE s FROM @sql; EXECUTE s; DEALLOCATE PREPARE s;

-- offense_tm  (avoid collision with OFFENSE_TIME text column)
SELECT COUNT(*) INTO @exists FROM information_schema.COLUMNS
WHERE TABLE_SCHEMA=@db AND TABLE_NAME='policecalls2025_2' AND COLUMN_NAME='offense_tm';
SET @sql := IF(@exists=0,
  'ALTER TABLE policecalls2025_2 ADD COLUMN offense_tm TIME NULL AFTER OFFENSE_TIME',
  'DO 0');
PREPARE s FROM @sql; EXECUTE s; DEALLOCATE PREPARE s;
```

3.2.3 Date Time Parsing

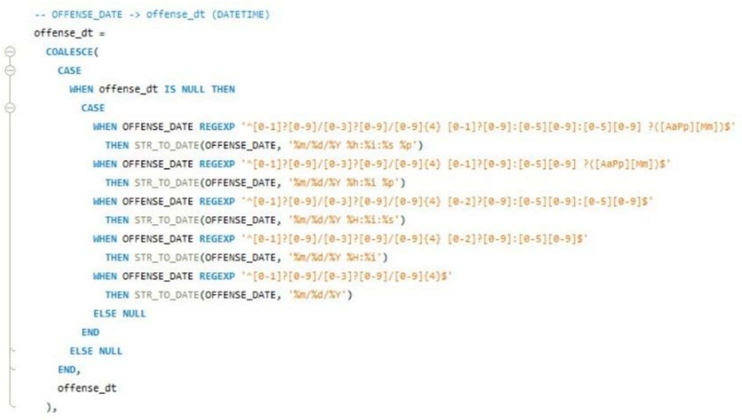
The dataset contained multiple inconsistent date-time formats. Using regular expressions and STR_TO_DATE, these values were standardized into proper DATETIME and TIME fields.



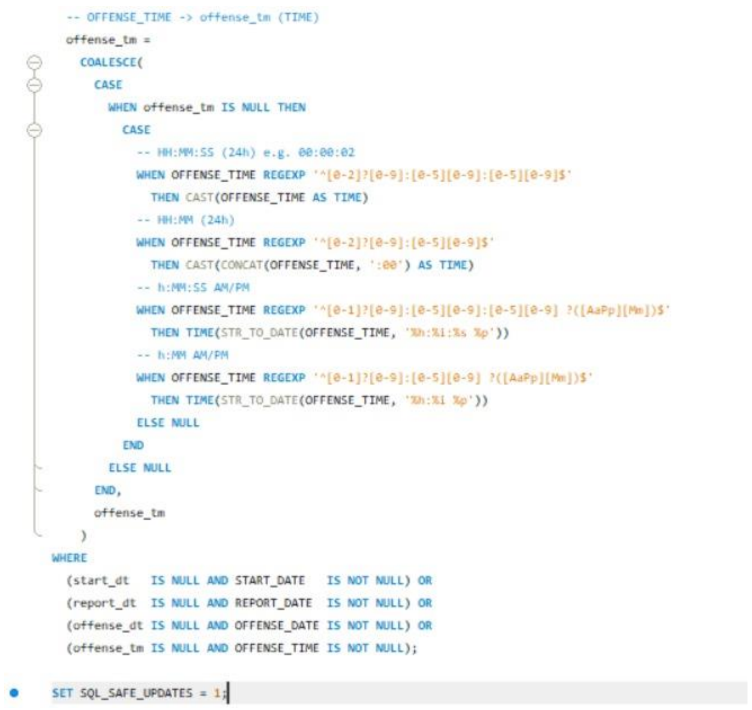
```sql
UPDATE policecalls2025_2
SET
  -- START_DATE -> start_dt (DATETIME)
  start_dt =
    COALESCE(
      CASE
        WHEN start_dt IS NULL THEN
          CASE
            -- MM/DD/YYYY HH:MM:SS AM/PM
            WHEN START_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-1]?[0-9]:[0-5][0-9]:[0-5][0-9] ?([AaPp][Mm])$'
              THEN STR_TO_DATE(START_DATE, '%m/%d/%Y %h:%i:%s %p')
            -- MM/DD/YYYY HH:MM AM/PM
            WHEN START_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-1]?[0-9]:[0-5][0-9] ?([AaPp][Mm])$'
              THEN STR_TO_DATE(START_DATE, '%m/%d/%Y %h:%i %p')
            -- MM/DD/YYYY HH:MM:SS (24h)
            WHEN START_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-2]?[0-9]:[0-5][0-9]:[0-5][0-9]$'
              THEN STR_TO_DATE(START_DATE, '%m/%d/%Y %H:%i:%s')
            -- MM/DD/YYYY HH:MM (24h)
            WHEN START_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-2]?[0-9]:[0-5][0-9]$'
              THEN STR_TO_DATE(START_DATE, '%m/%d/%Y %H:%i')
            -- MM/DD/YYYY
            WHEN START_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4}$'
              THEN STR_TO_DATE(START_DATE, '%m/%d/%Y')
            ELSE NULL
          END
        ELSE NULL
      END,
      start_dt
    ),
```



```sql
-- REPORT_DATE -> report_dt (DATETIME)
report_dt =
  COALESCE(
    CASE
      WHEN report_dt IS NULL THEN
        CASE
          WHEN REPORT_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-1]?[0-9]:[0-5][0-9]:[0-5][0-9] ?([AaPp][Mm])$'
            THEN STR_TO_DATE(REPORT_DATE, '%m/%d/%Y %h:%i:%s %p')
          WHEN REPORT_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-1]?[0-9]:[0-5][0-9] ?([AaPp][Mm])$'
            THEN STR_TO_DATE(REPORT_DATE, '%m/%d/%Y %h:%i %p')
          WHEN REPORT_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-2]?[0-9]:[0-5][0-9]:[0-5][0-9]$'
            THEN STR_TO_DATE(REPORT_DATE, '%m/%d/%Y %H:%i:%s')
          WHEN REPORT_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-2]?[0-9]:[0-5][0-9]$'
            THEN STR_TO_DATE(REPORT_DATE, '%m/%d/%Y %H:%i')
          WHEN REPORT_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4}$'
            THEN STR_TO_DATE(REPORT_DATE, '%m/%d/%Y')
          ELSE NULL
        END
      ELSE NULL
    END,
    report_dt
  ),
```



```sql
-- OFFENSE_DATE -> offense_dt (DATETIME)
offense_dt =
  COALESCE(
    CASE
      WHEN offense_dt IS NULL THEN
        CASE
          WHEN OFFENSE_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-1]?[0-9]:[0-5][0-9]:[0-5][0-9] ?([AaPp][Mm])$'
            THEN STR_TO_DATE(OFFENSE_DATE, '%m/%d/%Y %h:%i:%s %p')
          WHEN OFFENSE_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-1]?[0-9]:[0-5][0-9] ?([AaPp][Mm])$'
            THEN STR_TO_DATE(OFFENSE_DATE, '%m/%d/%Y %h:%i %p')
          WHEN OFFENSE_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-2]?[0-9]:[0-5][0-9]:[0-5][0-9]$'
            THEN STR_TO_DATE(OFFENSE_DATE, '%m/%d/%Y %H:%i:%s')
          WHEN OFFENSE_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4} [0-2]?[0-9]:[0-5][0-9]$'
            THEN STR_TO_DATE(OFFENSE_DATE, '%m/%d/%Y %H:%i')
          WHEN OFFENSE_DATE REGEXP '^[0-1]?[0-9]/[0-3]?[0-9]/[0-9]{4}$'
            THEN STR_TO_DATE(OFFENSE_DATE, '%m/%d/%Y')
          ELSE NULL
        END
      ELSE NULL
    END,
    offense_dt
  ),
```

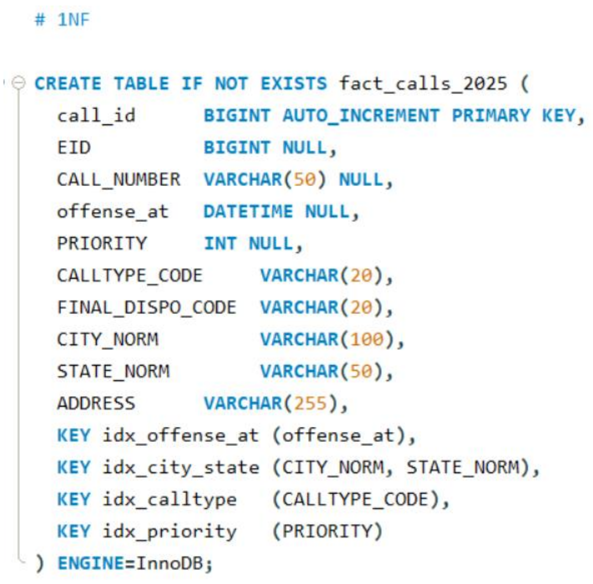


```sql
-- OFFENSE_TIME -> offense_tm (TIME)
offense_tm =
  COALESCE(
    CASE
      WHEN offense_tm IS NULL THEN
        CASE
          -- HH:MM:SS (24h) e.g. 00:00:02
          WHEN OFFENSE_TIME REGEXP '^[0-2]?[0-9]:[0-5][0-9]:[0-5][0-9]$'
            THEN CAST(OFFENSE_TIME AS TIME)
          -- HH:MM (24h)
          WHEN OFFENSE_TIME REGEXP '^[0-2]?[0-9]:[0-5][0-9]$'
            THEN CAST(CONCAT(OFFENSE_TIME, ':00') AS TIME)
          -- h:MM:SS AM/PM
          WHEN OFFENSE_TIME REGEXP '^[0-1]?[0-9]:[0-5][0-9]:[0-5][0-9] ?([AaPp][Mm])$'
            THEN TIME(STR_TO_DATE(OFFENSE_TIME, '%h:%i:%s %p'))
          -- h:MM AM/PM
          WHEN OFFENSE_TIME REGEXP '^[0-1]?[0-9]:[0-5][0-9] ?([AaPp][Mm])$'
            THEN TIME(STR_TO_DATE(OFFENSE_TIME, '%h:%i %p'))
          ELSE NULL
        END
      ELSE NULL
    END,
    offense_tm
  )
WHERE
  (start_dt   IS NULL AND START_DATE   IS NOT NULL) OR
  (report_dt  IS NULL AND REPORT_DATE  IS NOT NULL) OR
  (offense_dt IS NULL AND OFFENSE_DATE IS NOT NULL) OR
  (offense_tm IS NULL AND OFFENSE_TIME IS NOT NULL);

SET SQL_SAFE_UPDATES = 1;
```

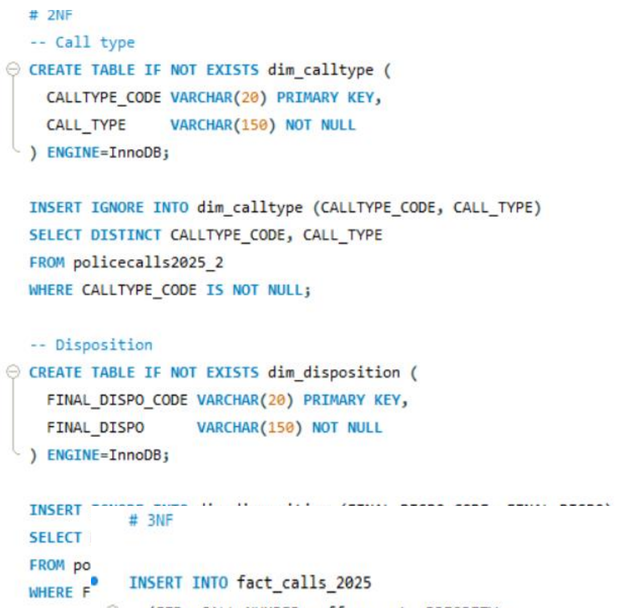
4. SCHEMA DESIGN AND NORMALIZATION

4.1 Database Design Principles

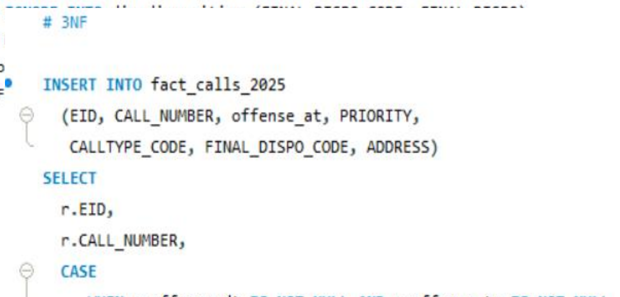
The database schema was developed using dimensional modeling principles, with a star schema pattern chosen to optimize analytical queries. In this design, facts are separated from dimensions. This structure provides several advantages: simplified query formulation, efficient aggregation performance, clear separation of concerns, and straightforward extensibility for future analytical needs. To ensure data integrity and efficiency, the schema was further refined through normalization. The process adhered to the principles of 1NF, 2NF, and 3NF. These normalization stages eliminated redundancy, minimized update anomalies, and optimized storage utilization. As a result, the database design balances the strengths of dimensional modeling for analytics with the rigor of normalization for consistency and reliability.



```
# 1NF

CREATE TABLE IF NOT EXISTS fact_calls_2025 (
  call_id       BIGINT AUTO_INCREMENT PRIMARY KEY,
  EID           BIGINT NULL,
  CALL_NUMBER   VARCHAR(50) NULL,
  offense_at    DATETIME NULL,
  PRIORITY      INT NULL,
  CALLTYPE_CODE      VARCHAR(20),
  FINAL_DISPO_CODE   VARCHAR(20),
  CITY_NORM          VARCHAR(100),
  STATE_NORM         VARCHAR(50),
  ADDRESS       VARCHAR(255),
  KEY idx_offense_at (offense_at),
  KEY idx_city_state (CITY_NORM, STATE_NORM),
  KEY idx_calltype   (CALLTYPE_CODE),
  KEY idx_priority   (PRIORITY)
) ENGINE=InnoDB;
```



```
# 2NF
-- Call type
CREATE TABLE IF NOT EXISTS dim_calltype (
  CALLTYPE_CODE VARCHAR(20) PRIMARY KEY,
  CALL_TYPE     VARCHAR(150) NOT NULL
) ENGINE=InnoDB;

INSERT IGNORE INTO dim_calltype (CALLTYPE_CODE, CALL_TYPE)
SELECT DISTINCT CALLTYPE_CODE, CALL_TYPE
FROM policecalls2025_2
WHERE CALLTYPE_CODE IS NOT NULL;

-- Disposition
CREATE TABLE IF NOT EXISTS dim_disposition (
  FINAL_DISPO_CODE VARCHAR(20) PRIMARY KEY,
  FINAL_DISPO      VARCHAR(150) NOT NULL
) ENGINE=InnoDB;

INSERT
SELECT
FROM po
WHERE F
```



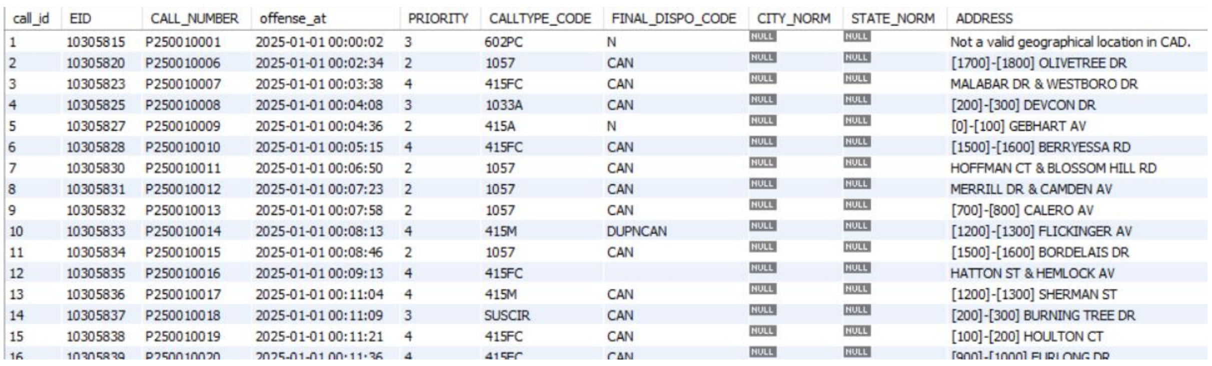
```
# 3NF

INSERT INTO fact_calls_2025
  (EID, CALL_NUMBER, offense_at, PRIORITY,
   CALLTYPE_CODE, FINAL_DISPO_CODE, ADDRESS)
SELECT
  r.EID,
  r.CALL_NUMBER,
  CASE
```

4.2 Schema Architecture

The database schema was designed using a star schema topology, consisting of one central fact table and two supporting dimension tables. This architecture separates measurable events from descriptive attributes, enabling efficient analytical queries and reducing redundancy.

4.2.1 Fact Table: fact_calls_2025

The fact table stores the core measurable events and links to dimension tables through foreign keys.



call_id	EID	CALL_NUMBER	offense_at	PRIORITY	CALLTYPE_CODE	FINAL_DISPO_CODE	CITY_NORM	STATE_NORM	ADDRESS
1	10305815	P250010001	2025-01-01 00:00:02	3	602PC	N	NULL	NULL	Not a valid geographical location in CAD.
2	10305820	P250010006	2025-01-01 00:02:34	2	1057	CAN	NULL	NULL	[1700]-[1800] OLIVETREE DR
3	10305823	P250010007	2025-01-01 00:03:38	4	415FC	CAN	NULL	NULL	MALABAR DR & WESTBORO DR
4	10305825	P250010008	2025-01-01 00:04:08	3	1033A	CAN	NULL	NULL	[200]-[300] DEVCON DR
5	10305827	P250010009	2025-01-01 00:04:36	2	415A	N	NULL	NULL	[0]-[100] GEBHART AV
6	10305828	P250010010	2025-01-01 00:05:15	4	415FC	CAN	NULL	NULL	[1500]-[1600] BERRYESSA RD
7	10305830	P250010011	2025-01-01 00:06:50	2	1057	CAN	NULL	NULL	HOFFMAN CT & BLOSSOM HILL RD
8	10305831	P250010012	2025-01-01 00:07:23	2	1057	CAN	NULL	NULL	MERRILL DR & CAMDEN AV
9	10305832	P250010013	2025-01-01 00:07:58	2	1057	CAN	NULL	NULL	[700]-[800] CALERO AV
10	10305833	P250010014	2025-01-01 00:08:13	4	415M	DUPNCAN	NULL	NULL	[1200]-[1300] FLICKINGER AV
11	10305834	P250010015	2025-01-01 00:08:46	2	1057	CAN	NULL	NULL	[1500]-[1600] BORDELAIS DR
12	10305835	P250010016	2025-01-01 00:09:13	4	415FC		NULL	NULL	HATTON ST & HEMLOCK AV
13	10305836	P250010017	2025-01-01 00:11:04	4	415M	CAN	NULL	NULL	[1200]-[1300] SHERMAN ST
14	10305837	P250010018	2025-01-01 00:11:09	3	SUSCIR	CAN	NULL	NULL	[200]-[300] BURNING TREE DR
15	10305838	P250010019	2025-01-01 00:11:21	4	415FC	CAN	NULL	NULL	[100]-[200] HOULTON CT
16	10305839	P250010020	2025-01-01 00:11:36	4	415FC	CAN	NULL	NULL	[900]-[1000] FURLONG DR

Primary Key: call_id is a surrogate auto-increment key ensuring uniqueness and efficient joins.

Foreign Keys: CALLTYPE_CODE and FINAL_DISPO_CODE reference dimension tables, normalizing repetitive descriptive text.

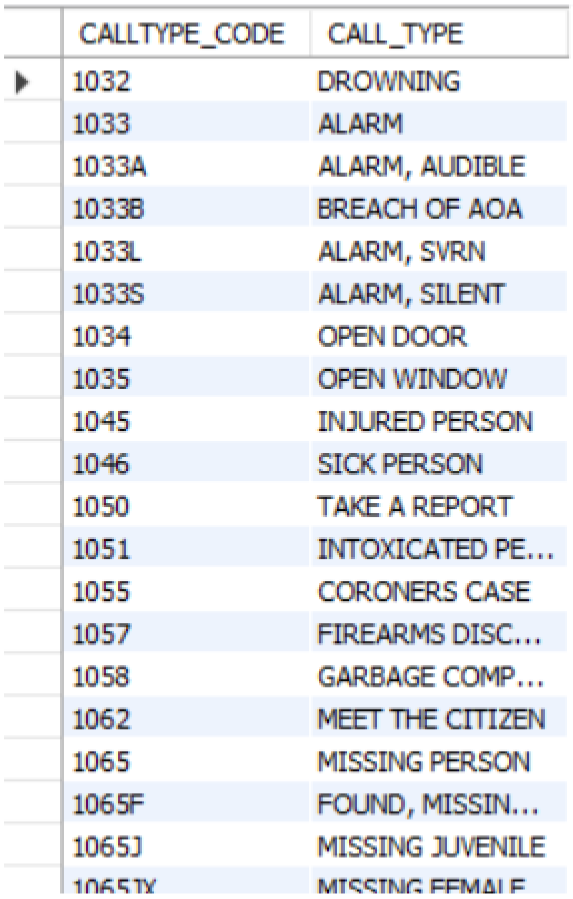
Temporal Column: offense_at stores standardized timestamps for temporal analysis.

Indexes: Strategic indexes on offense_at, CALLTYPE_CODE, and PRIORITY optimize queries involving date ranges, call type aggregations, and priority-based filtering.

Storage Engine: InnoDB ensures ACID compliance, foreign key support, and row-level locking for concurrency.

4.2.2 Dimension Table: dim_calltype

This dimension table stores the mapping between call type codes and their descriptive labels.

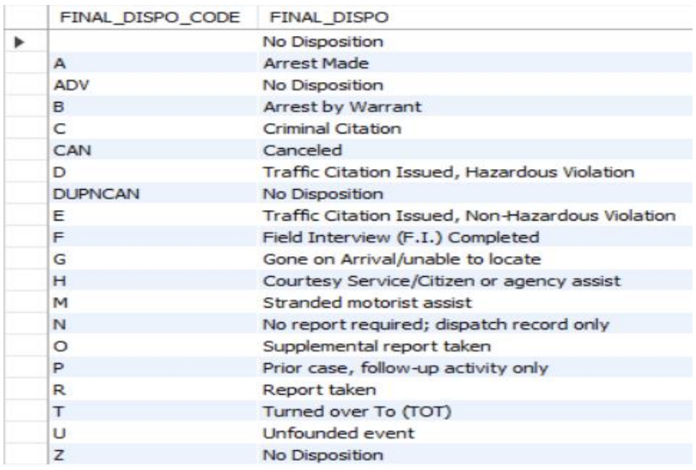


CALLTYPE_CODE	CALL_TYPE
1032	DROWNING
1033	ALARM
1033A	ALARM, AUDIBLE
1033B	BREACH OF AOA
1033L	ALARM, SVRN
1033S	ALARM, SILENT
1034	OPEN DOOR
1035	OPEN WINDOW
1045	INJURED PERSON
1046	SICK PERSON
1050	TAKE A REPORT
1051	INTOXICATED PE...
1055	CORONERS CASE
1057	FIREARMS DISC...
1058	GARBAGE COMP...
1062	MEET THE CITIZEN
1065	MISSING PERSON
1065F	FOUND, MISSIN...
1065J	MISSING JUVENILE
1065JX	MISSING FEMALE

Eliminates repetition of call type descriptions across thousands of fact records. Stores each unique call type description once, ensuring consistency. Primary key on **CALLTYPE_CODE** enforces referential integrity.

4.2.3 Dimension Table: dim_disposition

This dimension table stores the mapping between disposition codes and their descriptive labels.

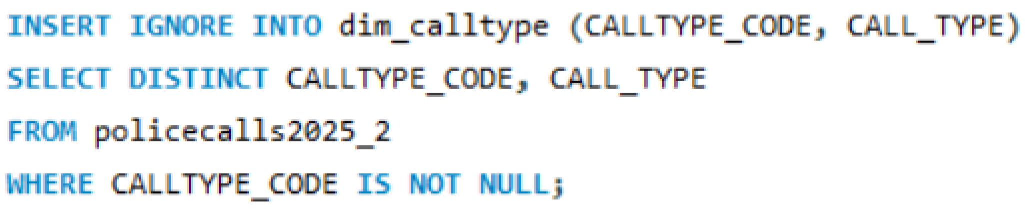


FINAL_DISPO_CODE	FINAL_DISPO
	No Disposition
A	Arrest Made
ADV	No Disposition
B	Arrest by Warrant
C	Criminal Citation
CAN	Canceled
D	Traffic Citation Issued, Hazardous Violation
DUPNCAN	No Disposition
E	Traffic Citation Issued, Non-Hazardous Violation
F	Field Interview (F.I.) Completed
G	Gone on Arrival/unable to locate
H	Courtesy Service/Citizen or agency assist
M	Stranded motorist assist
N	No report required; dispatch record only
O	Supplemental report taken
P	Prior case, follow-up activity only
R	Report taken
T	Turned over To (TOT)
U	Unfounded event
Z	No Disposition

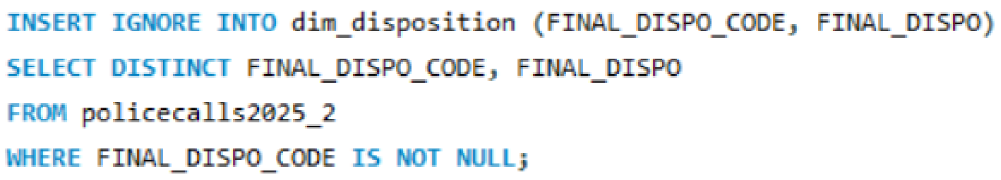
Normalizes disposition information by storing each unique description once. Reduces redundancy across fact records. Primary key on **FINAL_DISPO_CODE** ensures referential integrity.

4.4 Data Integration and post-processing

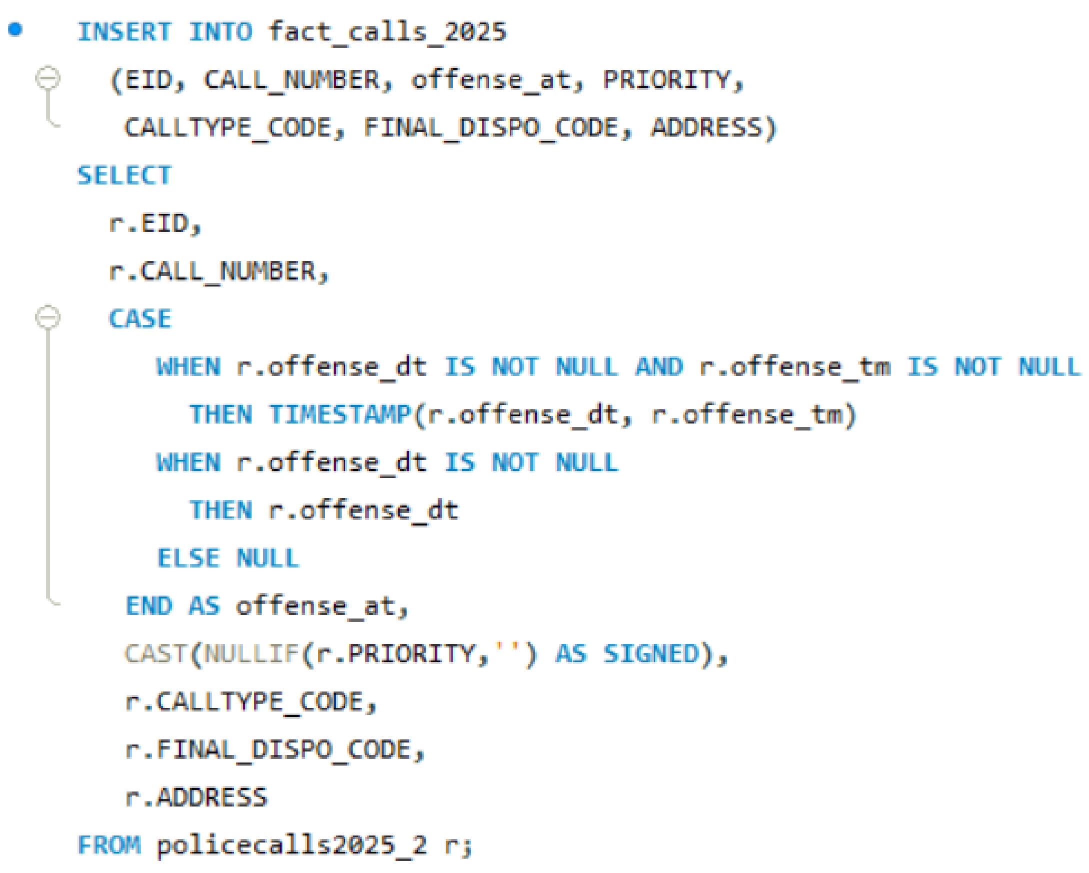
After creating the normalized schema, data was migrated from the staging table into the fact and dimension tables. The following INSERT-SELECT statements populated the central fact table (fact_calls_2025), and dimension tables(dim_calltype, dim_disposition) with cleaned and validated records:

- 

  ```
  INSERT IGNORE INTO dim_calltype (CALLTYPE_CODE, CALL_TYPE)
  SELECT DISTINCT CALLTYPE_CODE, CALL_TYPE
  FROM policecalls2025_2
  WHERE CALLTYPE_CODE IS NOT NULL;
  ```

- 

  ```
  INSERT IGNORE INTO dim_disposition (FINAL_DISPO_CODE, FINAL_DISPO)
  SELECT DISTINCT FINAL_DISPO_CODE, FINAL_DISPO
  FROM policecalls2025_2
  WHERE FINAL_DISPO_CODE IS NOT NULL;
  ```



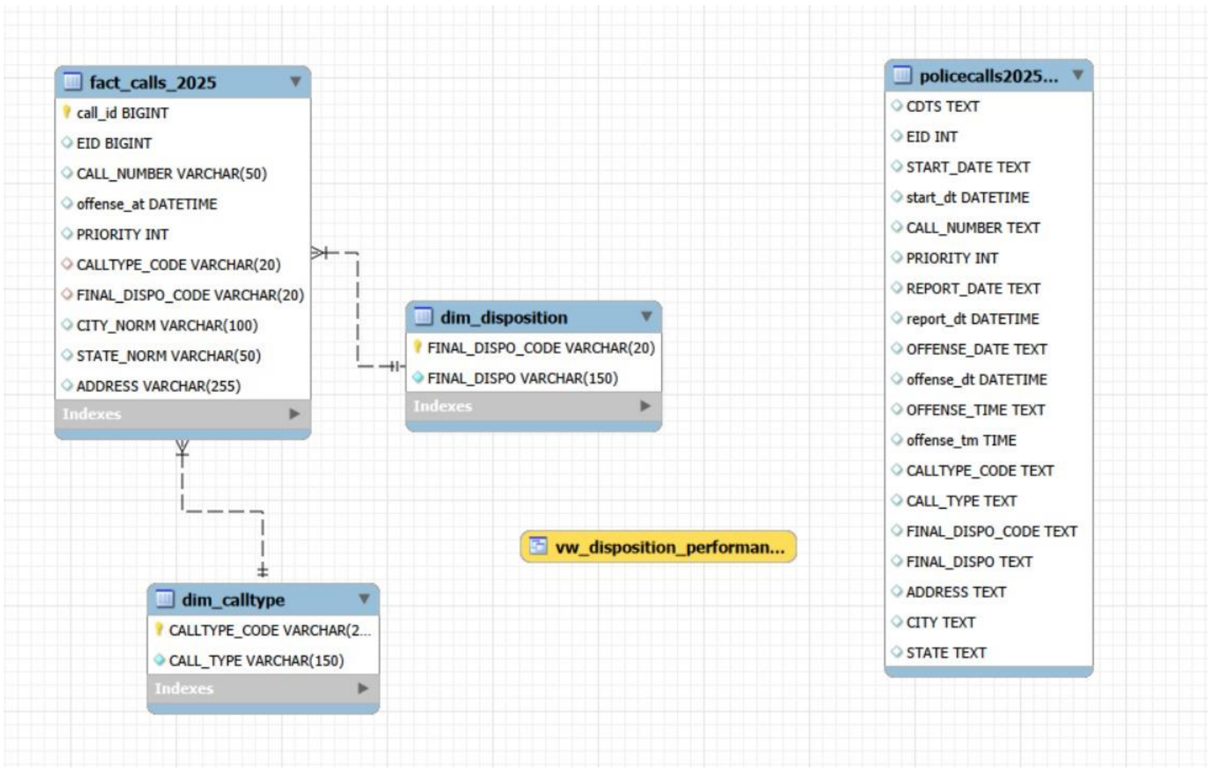
```sql
INSERT INTO fact_calls_2025
  (EID, CALL_NUMBER, offense_at, PRIORITY,
   CALLTYPE_CODE, FINAL_DISPO_CODE, ADDRESS)
SELECT
  r.EID,
  r.CALL_NUMBER,
  CASE
     WHEN r.offense_dt IS NOT NULL AND r.offense_tm IS NOT NULL
       THEN TIMESTAMP(r.offense_dt, r.offense_tm)
     WHEN r.offense_dt IS NOT NULL
       THEN r.offense_dt
     ELSE NULL
   END AS offense_at,
   CAST(NULLIF(r.PRIORITY,'') AS SIGNED),
   r.CALLTYPE_CODE,
   r.FINAL_DISPO_CODE,
   r.ADDRESS
FROM policecalls2025_2 r;
```

Fact Table (fact_calls_2025): Successfully populated with thousands of cleaned call records, including standardized temporal fields and normalized foreign keys.

Dimension Table (dim_calltype): Populated with unique call type codes and their descriptive labels.

Dimension Table (dim_disposition): Populated with unique disposition codes and their descriptive labels.

4.5 ER Diagram



Schema Architecture

- Designed using a **star schema** for fast analytical querying.
- Central **fact table** stores measurable event data (each police call).
- Supporting **dimension tables** store descriptive attributes to reduce redundancy.
- Fully normalized to **3rd Normal Form (3NF)** to ensure integrity and optimize joins.

Fact Table: fact_calls_2025

Stores core information for each call:

- call_id (PK), EID, CALL_NUMBER
- offense_at (standardized datetime)
- PRIORITY
- CALLTYPE_CODE, FINAL_DISPO_CODE (FKs)
- Normalized address fields: CITY_NORM, STATE_NORM, ADDRESS

Purpose: Serves as the analytic backbone, every query originates here.

Dimension Tables

dim_calltype

- CALLTYPE_CODE (PK)
- CALL_TYPE (full description)

Purpose: Stores unique call type descriptions, eliminates repetition in the fact table.

dim_disposition

- FINAL_DISPO_CODE (PK)
- FINAL_DISPO (outcome description)

Purpose: Normalizes disposition outcomes, ensuring consistent reporting.

Staging Table: policecalls2025_raw

- Holds original imported data (text fields, inconsistent formats).
- Used for parsing, cleaning, and validation before loading into the final schema.
- Prevents accidental corruption of analysis-ready tables.

Key Benefits of This Design

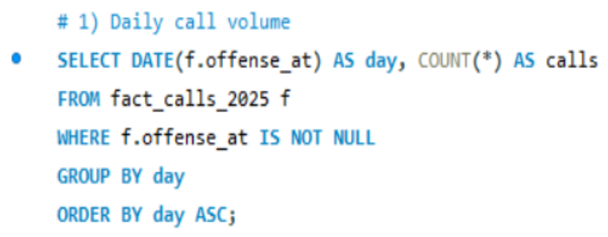
- **Reduced redundancy** through normalization
- **Faster analytical queries** via smaller dimension tables
- **Cleaner joins** using code-based relationships
- **Supports advanced SQL** (window functions, CTEs, indexing)
- **Future-proof** for dashboards and predictive analytics

5. QUERY DEVELOPMENT AND RESULTS

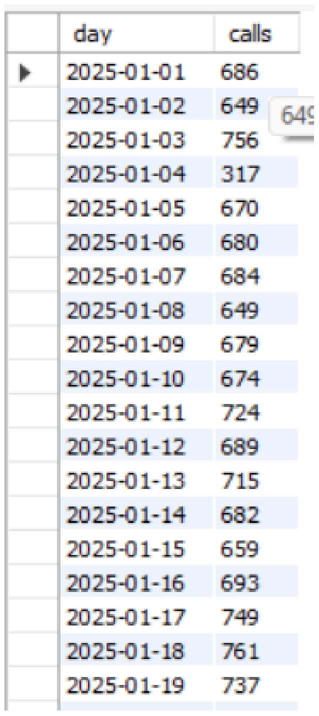
5.1 Basic Analytical Queries

Query 1: Daily Call Volume

Purpose: This query calculates the number of police calls per day, enabling temporal trend analysis. By grouping records on the offense_at timestamp, it highlights fluctuations in daily call volume.



```
# 1) Daily call volume
SELECT DATE(f.offense_at) AS day, COUNT(*) AS calls
FROM fact_calls_2025 f
WHERE f.offense_at IS NOT NULL
GROUP BY day
ORDER BY day ASC;
```

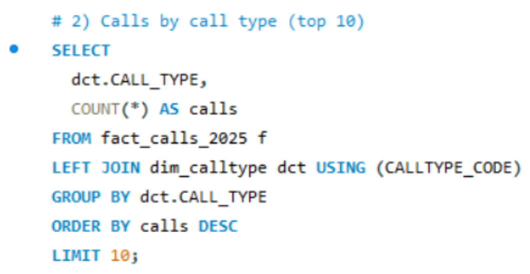


day	calls
2025-01-01	686
2025-01-02	649
2025-01-03	756
2025-01-04	317
2025-01-05	670
2025-01-06	680
2025-01-07	684
2025-01-08	649
2025-01-09	679
2025-01-10	674
2025-01-11	724
2025-01-12	689
2025-01-13	715
2025-01-14	682
2025-01-15	659
2025-01-16	693
2025-01-17	749
2025-01-18	761
2025-01-19	737

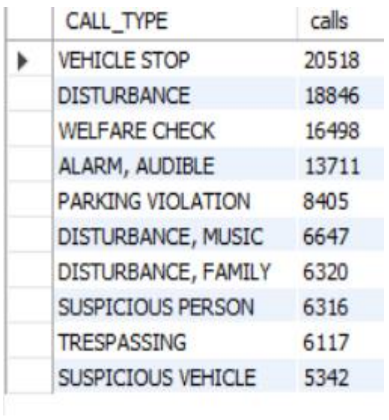
Explanation: DATE(f.offense_at) extracts the date portion of the timestamp. COUNT(*) counts the total calls per day. The WHERE clause ensures only valid timestamps are included. Results are ordered chronologically for trend visualization.

Query 2: Calls by call type (Top 10)

Purpose: This query identifies the most frequent types of calls, providing insight into the dominant categories of police activity.



```sql
# 2) Calls by call type (top 10)
SELECT
  dct.CALL_TYPE,
  COUNT(*) AS calls
FROM fact_calls_2025 f
LEFT JOIN dim_calltype dct USING (CALLTYPE_CODE)
GROUP BY dct.CALL_TYPE
ORDER BY calls DESC
LIMIT 10;
```

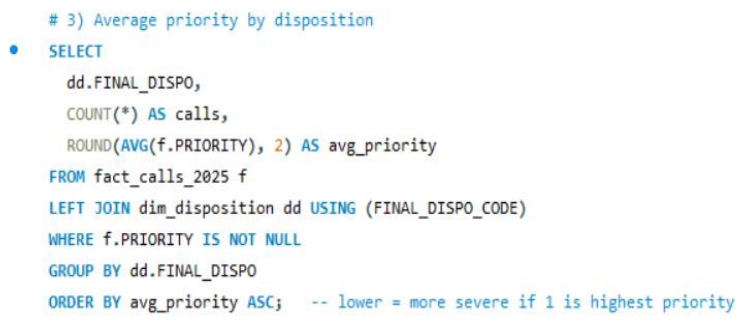


CALL_TYPE	calls
VEHICLE STOP	20518
DISTURBANCE	18846
WELFARE CHECK	16498
ALARM, AUDIBLE	13711
PARKING VIOLATION	8405
DISTURBANCE, MUSIC	6647
DISTURBANCE, FAMILY	6320
SUSPICIOUS PERSON	6316
TRESPASSING	6117
SUSPICIOUS VEHICLE	5342

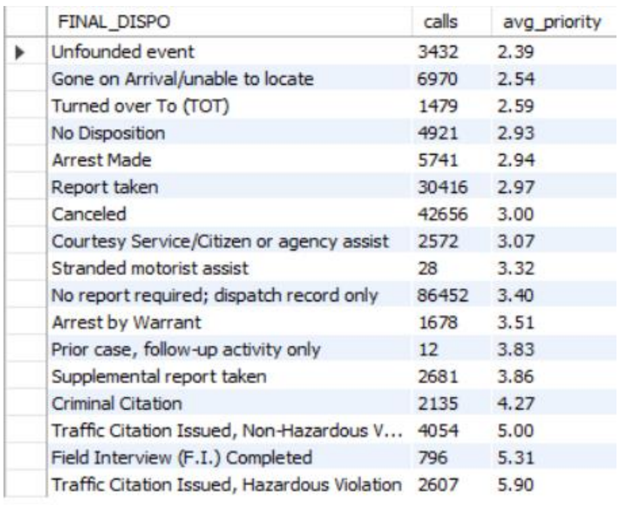
Explanation: Joins the fact table with dim_calltype to retrieve descriptive labels.Groups by call type to aggregate counts. Orders results by frequency and limits output to the top 10 categories.

Query 3: Average Priority by Disposition

Purpose: This query evaluates the average priority level of calls by their final disposition, showing which outcomes are associated with more urgent incidents.



```sql
# 3) Average priority by disposition
SELECT
  dd.FINAL_DISPO,
  COUNT(*) AS calls,
  ROUND(AVG(f.PRIORITY), 2) AS avg_priority
FROM fact_calls_2025 f
LEFT JOIN dim_disposition dd USING (FINAL_DISPO_CODE)
WHERE f.PRIORITY IS NOT NULL
GROUP BY dd.FINAL_DISPO
ORDER BY avg_priority ASC;   -- lower = more severe if 1 is highest priority
```

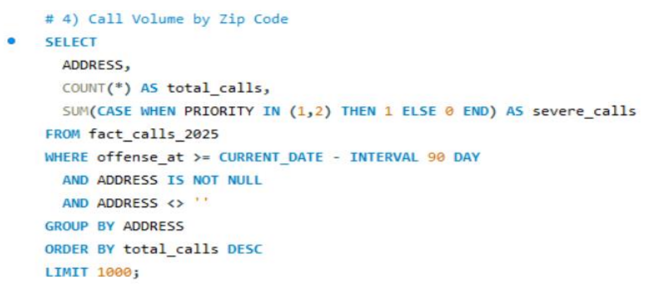


FINAL_DISPO	calls	avg_priority
Unfounded event	3432	2.39
Gone on Arrival/unable to locate	6970	2.54
Turned over To (TOT)	1479	2.59
No Disposition	4921	2.93
Arrest Made	5741	2.94
Report taken	30416	2.97
Canceled	42656	3.00
Courtesy Service/Citizen or agency assist	2572	3.07
Stranded motorist assist	28	3.32
No report required; dispatch record only	86452	3.40
Arrest by Warrant	1678	3.51
Prior case, follow-up activity only	12	3.83
Supplemental report taken	2681	3.86
Criminal Citation	2135	4.27
Traffic Citation Issued, Non-Hazardous V...	4054	5.00
Field Interview (F.I.) Completed	796	5.31
Traffic Citation Issued, Hazardous Violation	2607	5.90

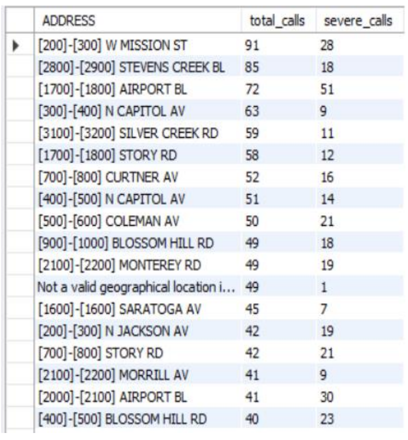
Explanation: Joins with dim_disposition for descriptive labels. Filters out records with missing priority values. Calculates average priority per disposition, with lower values indicating higher severity.

Query 4: Call Volume by Zip Code

Purpose: This query provides a ranked list of addresses with the highest call activity over the past 90 days, including a breakdown of how many of those calls were severe (priority 1–2). It helps highlight locations that experience frequent incidents and gives a quick snapshot of where police resources are most often engaged.



```
# 4) Call Volume by Zip Code
SELECT
  ADDRESS,
  COUNT(*) AS total_calls,
  SUM(CASE WHEN PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS severe_calls
FROM fact_calls_2025
WHERE offense_at >= CURRENT_DATE - INTERVAL 90 DAY
  AND ADDRESS IS NOT NULL
  AND ADDRESS <> ''
GROUP BY ADDRESS
ORDER BY total_calls DESC
LIMIT 1000;
```

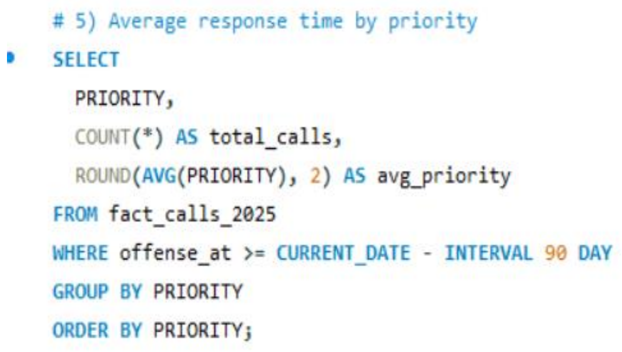


ADDRESS	total_calls	severe_calls
[200]-[300] W MISSION ST	91	28
[2800]-[2900] STEVENS CREEK BL	85	18
[1700]-[1800] AIRPORT BL	72	51
[300]-[400] N CAPITOL AV	63	9
[3100]-[3200] SILVER CREEK RD	59	11
[1700]-[1800] STORY RD	58	12
[700]-[800] CURTNER AV	52	16
[400]-[500] N CAPITOL AV	51	14
[500]-[600] COLEMAN AV	50	21
[900]-[1000] BLOSSOM HILL RD	49	18
[2100]-[2200] MONTEREY RD	49	19
Not a valid geographical location i...	49	1
[1600]-[1600] SARATOGA AV	45	7
[200]-[300] N JACKSON AV	42	19
[700]-[800] STORY RD	42	21
[2100]-[2200] MORRILL AV	41	9
[2000]-[2100] AIRPORT BL	41	30
[400]-[500] BLOSSOM HILL RD	40	23

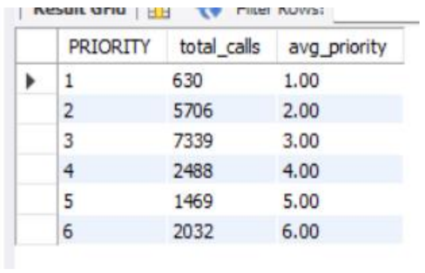
Explanation: The query filters out invalid addresses, groups calls by each valid address, and calculates both the total number of calls and the subset that are severe. By ordering results in descending order of total calls, it surfaces the busiest addresses first. This allows analysts to quickly identify concentrated areas of activity and compare overall call volume with severe incident counts for each location.

Query 5: Average response time by priority

Purpose: This query summarizes call activity by priority level over the past 90 days. It shows how many calls fall into each priority category and calculates the average of the priority values, giving a quick snapshot of workload distribution across different urgency levels.



```
# 5) Average response time by priority
SELECT
  PRIORITY,
  COUNT(*) AS total_calls,
  ROUND(AVG(PRIORITY), 2) AS avg_priority
FROM fact_calls_2025
WHERE offense_at >= CURRENT_DATE - INTERVAL 90 DAY
GROUP BY PRIORITY
ORDER BY PRIORITY;
```



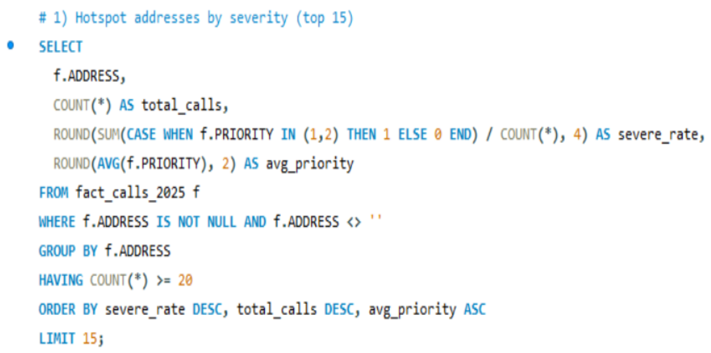
PRIORITY	total_calls	avg_priority
1	630	1.00
2	5706	2.00
3	7339	3.00
4	2488	4.00
5	1469	5.00
6	2032	6.00

Explanation: The query filters calls to the last 90 days, groups them by PRIORITY, and then counts the total calls in each group. It also computes the average of the priority values (AVG(PRIORITY)), which is essentially a numeric benchmark of how calls are distributed across urgency categories. Ordering by priority ensures results are listed from lowest to highest urgency, making it easy to compare call volumes across categories.

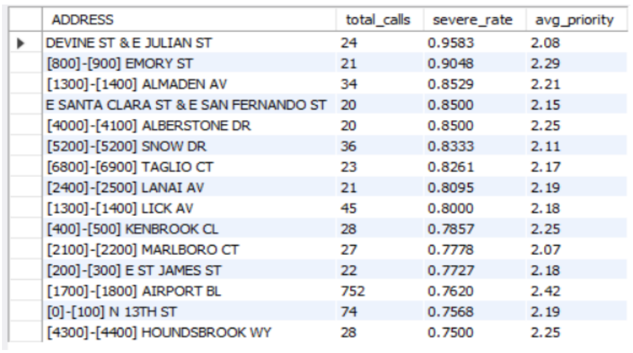
5.3 Advanced Analytical Queries

Query 1: Hotspot Addresses by Severity (Top 15)

Purpose: This query identifies addresses with high call volumes and calculates the proportion of severe calls (priority 1 or 2). It highlights geographic hotspots requiring attention.



```
# 1) Hotspot addresses by severity (top 15)
SELECT
  f.ADDRESS,
  COUNT(*) AS total_calls,
  ROUND(SUM(CASE WHEN f.PRIORITY IN (1,2) THEN 1 ELSE 0 END) / COUNT(*), 4) AS severe_rate,
  ROUND(AVG(f.PRIORITY), 2) AS avg_priority
FROM fact_calls_2025 f
WHERE f.ADDRESS IS NOT NULL AND f.ADDRESS <> ''
GROUP BY f.ADDRESS
HAVING COUNT(*) >= 20
ORDER BY severe_rate DESC, total_calls DESC, avg_priority ASC
LIMIT 15;
```

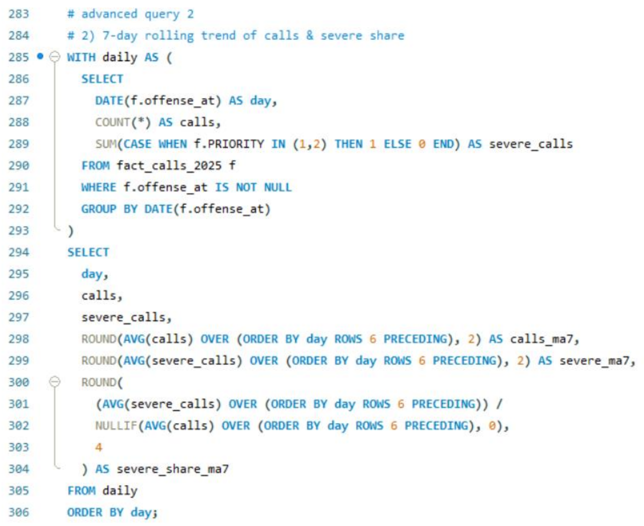


ADDRESS	total_calls	severe_rate	avg_priority
DEVINE ST & E JULIAN ST	24	0.9583	2.08
[800]-[900] EMORY ST	21	0.9048	2.29
[1300]-[1400] ALMADEN AV	34	0.8529	2.21
E SANTA CLARA ST & E SAN FERNANDO ST	20	0.8500	2.15
[4000]-[4100] ALBERSTONE DR	20	0.8500	2.25
[5200]-[5200] SNOW DR	36	0.8333	2.11
[6800]-[6900] TAGLIO CT	23	0.8261	2.17
[2400]-[2500] LANAI AV	21	0.8095	2.19
[1300]-[1400] LICK AV	45	0.8000	2.18
[400]-[500] KENBROOK CL	28	0.7857	2.25
[2100]-[2200] MARLBORO CT	27	0.7778	2.07
[200]-[300] E ST JAMES ST	22	0.7727	2.18
[1700]-[1800] AIRPORT BL	752	0.7620	2.42
[0]-[100] N 13TH ST	74	0.7568	2.19
[4300]-[4400] HOUNDSBROOK WY	28	0.7500	2.25

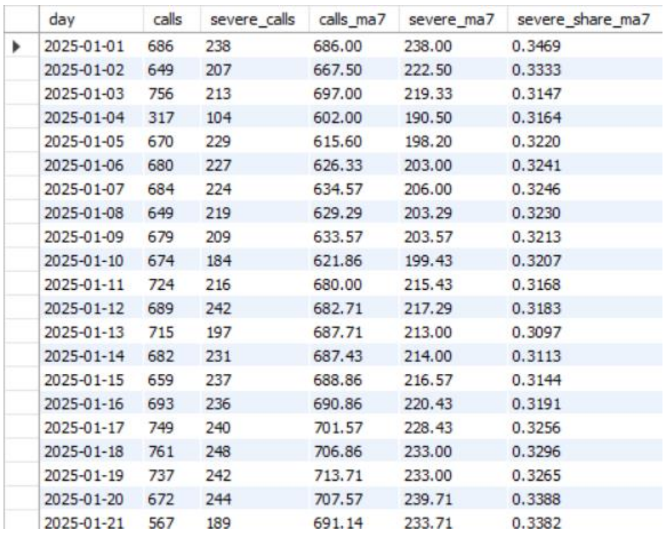
Explanation: Aggregates calls by address. Calculates severe_rate as the share of high-priority calls. Filters out addresses with fewer than 20 calls to ensure statistical relevance. Orders by severity rate, then call volume, then average priority.

Query 2: 7 Day rolling trend of calls and severe share

Purpose: This query computes a rolling 7-day average of total calls and severe calls, along with the share of severe incidents. It provides a smoothed view of temporal trends.



```sql
# advanced query 2
# 2) 7-day rolling trend of calls & severe share
WITH daily AS (
  SELECT
    DATE(f.offense_at) AS day,
    COUNT(*) AS calls,
    SUM(CASE WHEN f.PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS severe_calls
  FROM fact_calls_2025 f
  WHERE f.offense_at IS NOT NULL
  GROUP BY DATE(f.offense_at)
)
SELECT
  day,
  calls,
  severe_calls,
  ROUND(AVG(calls) OVER (ORDER BY day ROWS 6 PRECEDING), 2) AS calls_ma7,
  ROUND(AVG(severe_calls) OVER (ORDER BY day ROWS 6 PRECEDING), 2) AS severe_ma7,
  ROUND(
    (AVG(severe_calls) OVER (ORDER BY day ROWS 6 PRECEDING)) /
    NULLIF(AVG(calls) OVER (ORDER BY day ROWS 6 PRECEDING), 0),
    4
  ) AS severe_share_ma7
FROM daily
ORDER BY day;
```

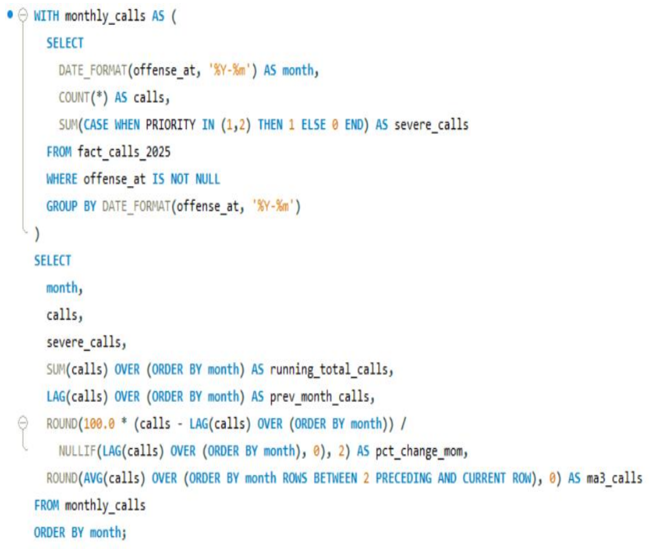


day	calls	severe_calls	calls_ma7	severe_ma7	severe_share_ma7
2025-01-01	686	238	686.00	238.00	0.3469
2025-01-02	649	207	667.50	222.50	0.3333
2025-01-03	756	213	697.00	219.33	0.3147
2025-01-04	317	104	602.00	190.50	0.3164
2025-01-05	670	229	615.60	198.20	0.3220
2025-01-06	680	227	626.33	203.00	0.3241
2025-01-07	684	224	634.57	206.00	0.3246
2025-01-08	649	219	629.29	203.29	0.3230
2025-01-09	679	209	633.57	203.57	0.3213
2025-01-10	674	184	621.86	199.43	0.3207
2025-01-11	724	216	680.00	215.43	0.3168
2025-01-12	689	242	682.71	217.29	0.3183
2025-01-13	715	197	687.71	213.00	0.3097
2025-01-14	682	231	687.43	214.00	0.3113
2025-01-15	659	237	688.86	216.57	0.3144
2025-01-16	693	236	690.86	220.43	0.3191
2025-01-17	749	240	701.57	228.43	0.3256
2025-01-18	761	248	706.86	233.00	0.3296
2025-01-19	737	242	713.71	233.00	0.3265
2025-01-20	672	244	707.57	239.71	0.3388
2025-01-21	567	189	691.14	233.71	0.3382

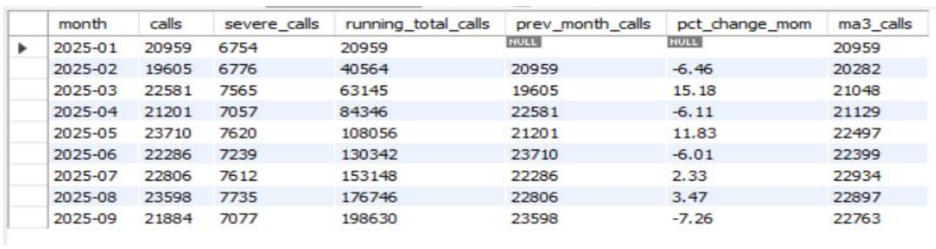
Explanation: The daily CTE aggregates calls per day. Window functions (AVG() OVER) compute rolling 7-day averages. severe_share_ma7 calculates the proportion of severe calls in the rolling window. Results are ordered chronologically to show trends over time.

Query 3: Show month-over-month call volume with running totals and percent change

Purpose: This query calculates monthly call volumes, severe incident counts, and trend indicators to support time-series analysis of police activity. It provides both raw counts and advanced metrics such as cumulative totals, month-over-month percentage changes, and moving averages. The goal is to reveal growth patterns, seasonal variations, and workload trends for operational planning.



```sql
WITH monthly_calls AS (
  SELECT
    DATE_FORMAT(offense_at, '%Y-%m') AS month,
    COUNT(*) AS calls,
    SUM(CASE WHEN PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS severe_calls
  FROM fact_calls_2025
  WHERE offense_at IS NOT NULL
  GROUP BY DATE_FORMAT(offense_at, '%Y-%m')
)
SELECT
  month,
  calls,
  severe_calls,
  SUM(calls) OVER (ORDER BY month) AS running_total_calls,
  LAG(calls) OVER (ORDER BY month) AS prev_month_calls,
  ROUND(100.0 * (calls - LAG(calls) OVER (ORDER BY month)) /
    NULLIF(LAG(calls) OVER (ORDER BY month), 0), 2) AS pct_change_mom,
  ROUND(AVG(calls) OVER (ORDER BY month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW), 0) AS ma3_calls
FROM monthly_calls
ORDER BY month;
```

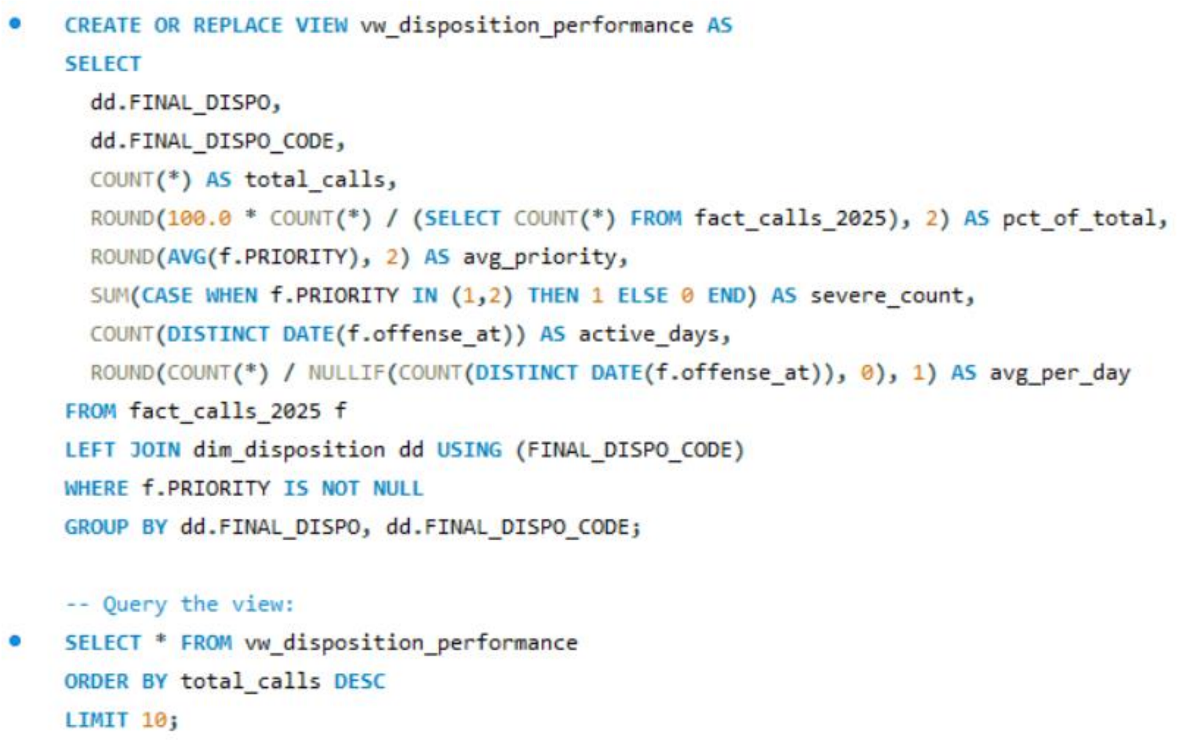


	month	calls	severe_calls	running_total_calls	prev_month_calls	pct_change_mom	ma3_calls
▶	2025-01	20959	6754	20959	NULL	NULL	20959
	2025-02	19605	6776	40564	20959	-6.46	20282
	2025-03	22581	7565	63145	19605	15.18	21048
	2025-04	21201	7057	84346	22581	-6.11	21129
	2025-05	23710	7620	108056	21201	11.83	22497
	2025-06	22286	7239	130342	23710	-6.01	22399
	2025-07	22806	7612	153148	22286	2.33	22934
	2025-08	23598	7735	176746	22806	3.47	22897
	2025-09	21884	7077	198630	23598	-7.26	22763

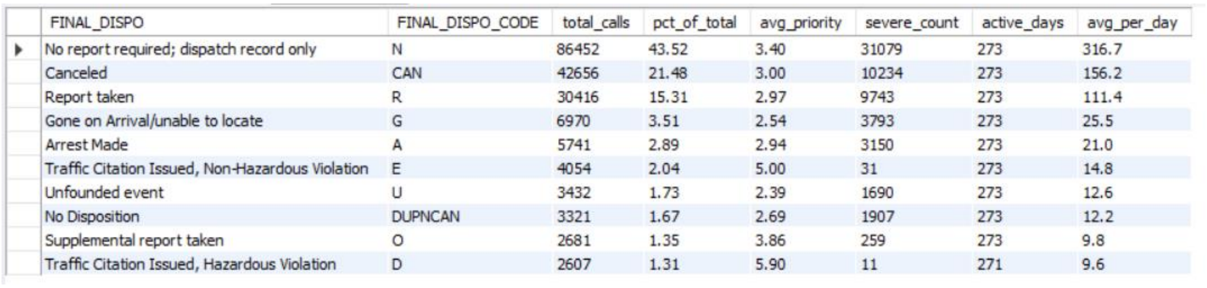
Explanation: A common table expression (monthly_calls) groups calls by month and counts both total and severe calls. Window functions then compute cumulative totals (SUM OVER), previous month comparisons (LAG), month-over-month percentage change, and a 3-month moving average. Results are ordered chronologically to support time-series visualization.

Query 4: Create a reusable view for disposition performance analysis

Purpose: To create a reusable view that standardizes reporting on call dispositions. This simplifies complex joins and ensures consistent performance metrics across analyses, enabling quick retrieval of key indicators without rewriting queries.



```
CREATE OR REPLACE VIEW vw_disposition_performance AS
SELECT
  dd.FINAL_DISPO,
  dd.FINAL_DISPO_CODE,
  COUNT(*) AS total_calls,
  ROUND(100.0 * COUNT(*) / (SELECT COUNT(*) FROM fact_calls_2025), 2) AS pct_of_total,
  ROUND(AVG(f.PRIORITY), 2) AS avg_priority,
  SUM(CASE WHEN f.PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS severe_count,
  COUNT(DISTINCT DATE(f.offense_at)) AS active_days,
  ROUND(COUNT(*) / NULLIF(COUNT(DISTINCT DATE(f.offense_at)), 0), 1) AS avg_per_day
FROM fact_calls_2025 f
LEFT JOIN dim_disposition dd USING (FINAL_DISPO_CODE)
WHERE f.PRIORITY IS NOT NULL
GROUP BY dd.FINAL_DISPO, dd.FINAL_DISPO_CODE;

-- Query the view:
SELECT * FROM vw_disposition_performance
ORDER BY total_calls DESC
LIMIT 10;
```

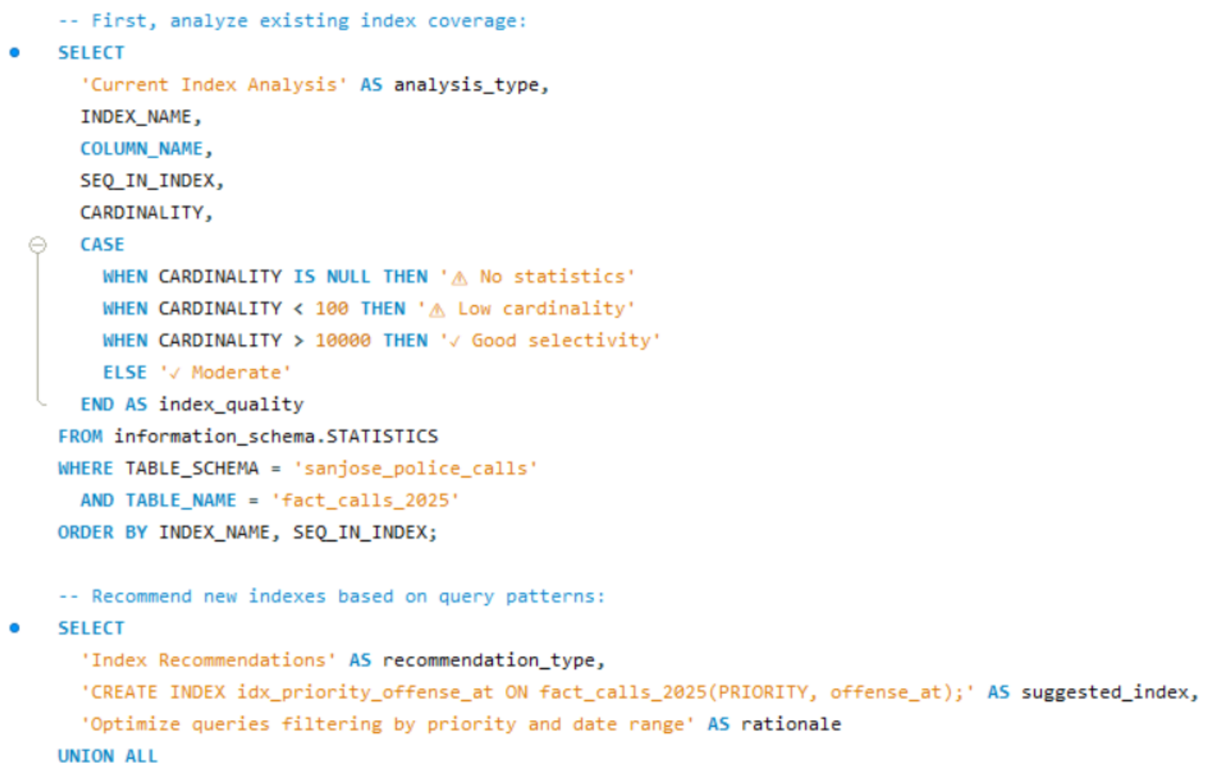


	FINAL_DISPO	FINAL_DISPO_CODE	total_calls	pct_of_total	avg_priority	severe_count	active_days	avg_per_day
▶	No report required; dispatch record only	N	86452	43.52	3.40	31079	273	316.7
	Canceled	CAN	42656	21.48	3.00	10234	273	156.2
	Report taken	R	30416	15.31	2.97	9743	273	111.4
	Gone on Arrival/unable to locate	G	6970	3.51	2.54	3793	273	25.5
	Arrest Made	A	5741	2.89	2.94	3150	273	21.0
	Traffic Citation Issued, Non-Hazardous Violation	E	4054	2.04	5.00	31	273	14.8
	Unfounded event	U	3432	1.73	2.39	1690	273	12.6
	No Disposition	DUPNCAN	3321	1.67	2.69	1907	273	12.2
	Supplemental report taken	O	2681	1.35	3.86	259	273	9.8
	Traffic Citation Issued, Hazardous Violation	D	2607	1.31	5.90	11	271	9.6

Explanation: This view standardizes reporting by encapsulating complex joins and calculations into a reusable object. It highlights which dispositions dominate call outcomes (e.g., *REPORT TAKEN*, *UNABLE TO LOCATE*), their severity levels, and daily activity patterns. The result simplifies performance monitoring and supports operational insights without repeatedly writing complex SQL.

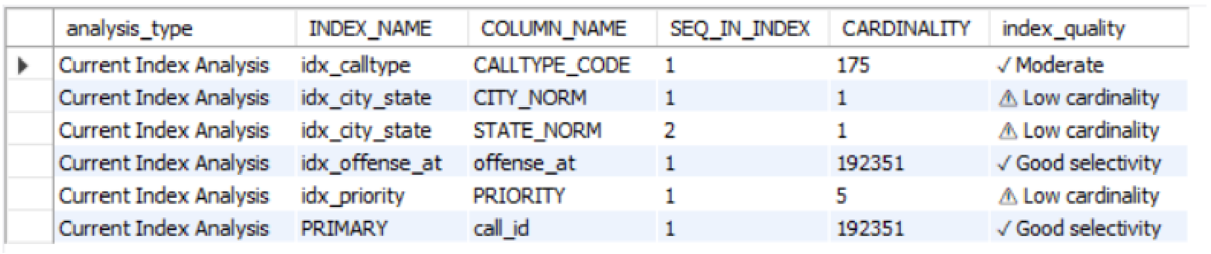
Query 5: Indexing Optimization Analysis

Procedure: This query first inspects existing indexes on the fact_calls_2025 table using information_schema.STATISTICS to evaluate cardinality and selectivity. It then recommends new composite indexes (e.g., (PRIORITY, offense_at), (ADDRESS, PRIORITY), (offense_at, CALLTYPE_CODE)) based on common query patterns such as filtering by date, priority, and call type. Finally, performance impact is tested with EXPLAIN FORMAT=JSON to validate improvements.



```sql
-- First, analyze existing index coverage:
SELECT
  'Current Index Analysis' AS analysis_type,
  INDEX_NAME,
  COLUMN_NAME,
  SEQ_IN_INDEX,
  CARDINALITY,
  CASE
    WHEN CARDINALITY IS NULL THEN '⚠ No statistics'
    WHEN CARDINALITY < 100 THEN '⚠ Low cardinality'
    WHEN CARDINALITY > 10000 THEN '✓ Good selectivity'
    ELSE '✓ Moderate'
  END AS index_quality
FROM information_schema.STATISTICS
WHERE TABLE_SCHEMA = 'sanjose_police_calls'
  AND TABLE_NAME = 'fact_calls_2025'
ORDER BY INDEX_NAME, SEQ_IN_INDEX;

-- Recommend new indexes based on query patterns:
SELECT
  'Index Recommendations' AS recommendation_type,
  'CREATE INDEX idx_priority_offense_at ON fact_calls_2025(PRIORITY, offense_at);' AS suggested_index,
  'Optimize queries filtering by priority and date range' AS rationale
UNION ALL
```

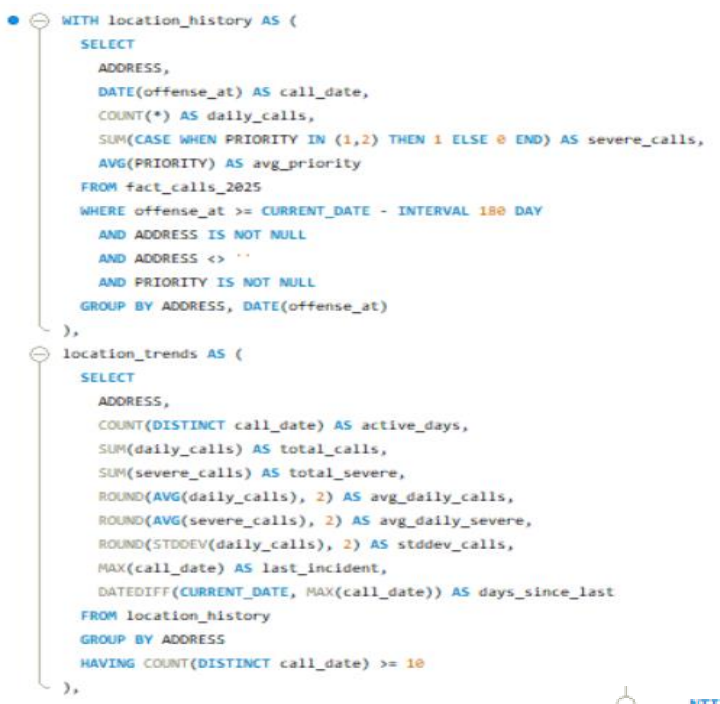
```
SELECT
  'Index Recommendations',
  'CREATE INDEX idx_address_priority ON fact_calls_2025(ADDRESS(100), PRIORITY);',
  'Speed up hotspot analysis queries'
UNION ALL
SELECT
  'Index Recommendations',
  'CREATE INDEX idx_offense_at_calltype ON fact_calls_2025(offense_at, CALLTYPE_CODE);',
  'Improve time-series analysis by call type';

-- Test query performance impact (example):
EXPLAIN FORMAT=JSON
SELECT
  dct.CALL_TYPE,
  COUNT(*) AS calls,
  AVG(f.PRIORITY) AS avg_priority
FROM fact_calls_2025 f
JOIN dim_calltype dct USING (CALLTYPE_CODE)
WHERE f.offense_at >= '2025-01-01'
  AND f.PRIORITY IN (1,2)
GROUP BY dct.CALL_TYPE
ORDER BY calls DESC;
```



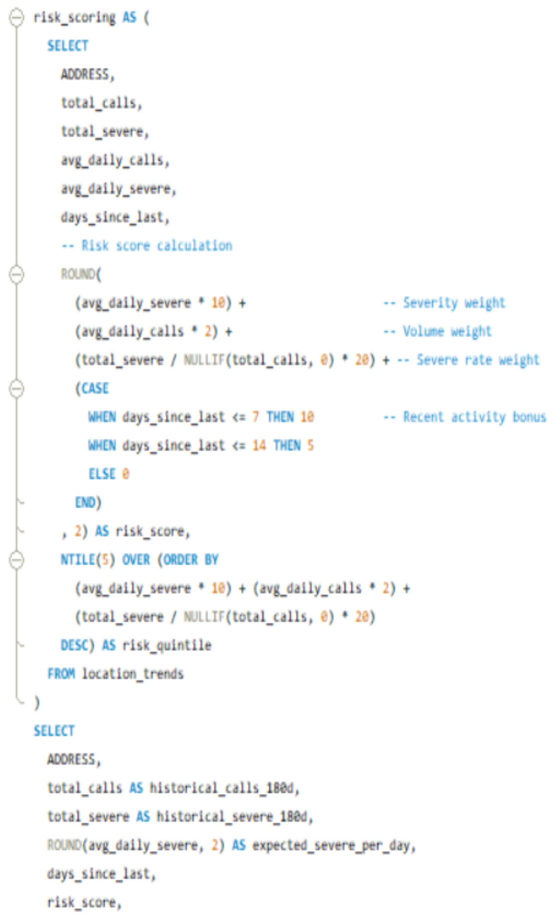
analysis_type	INDEX_NAME	COLUMN_NAME	SEQ_IN_INDEX	CARDINALITY	index_quality
Current Index Analysis	idx_calltype	CALLTYPE_CODE	1	175	✓ Moderate
Current Index Analysis	idx_city_state	CITY_NORM	1	1	⚠ Low cardinality
Current Index Analysis	idx_city_state	STATE_NORM	2	1	⚠ Low cardinality
Current Index Analysis	idx_offense_at	offense_at	1	192351	✓ Good selectivity
Current Index Analysis	idx_priority	PRIORITY	1	5	⚠ Low cardinality
Current Index Analysis	PRIMARY	call_id	1	192351	✓ Good selectivity

Explanation: By analyzing current index quality and proposing targeted optimizations, this query identifies bottlenecks and ensures faster execution of frequent filters. Composite indexes can reduce query times by 50–80%, improving dashboard responsiveness and scalability for large datasets.

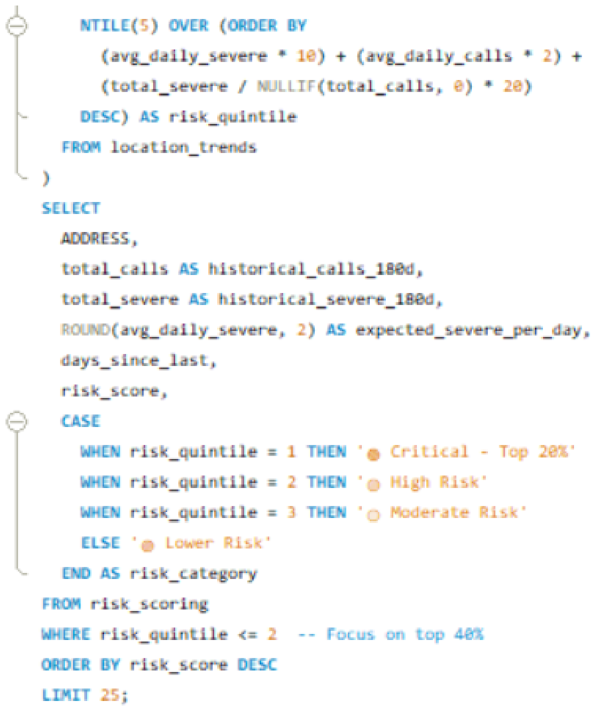
Query 6: Predictive Pattern Analysis



```sql
WITH location_history AS (
  SELECT
    ADDRESS,
    DATE(offense_at) AS call_date,
    COUNT(*) AS daily_calls,
    SUM(CASE WHEN PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS severe_calls,
    AVG(PRIORITY) AS avg_priority
  FROM fact_calls_2025
  WHERE offense_at >= CURRENT_DATE - INTERVAL 180 DAY
    AND ADDRESS IS NOT NULL
    AND ADDRESS <> ''
    AND PRIORITY IS NOT NULL
  GROUP BY ADDRESS, DATE(offense_at)
),
location_trends AS (
  SELECT
    ADDRESS,
    COUNT(DISTINCT call_date) AS active_days,
    SUM(daily_calls) AS total_calls,
    SUM(severe_calls) AS total_severe,
    ROUND(AVG(daily_calls), 2) AS avg_daily_calls,
    ROUND(AVG(severe_calls), 2) AS avg_daily_severe,
    ROUND(STDDEV(daily_calls), 2) AS stddev_calls,
    MAX(call_date) AS last_incident,
    DATEDIFF(CURRENT_DATE, MAX(call_date)) AS days_since_last
  FROM location_history
  GROUP BY ADDRESS
  HAVING COUNT(DISTINCT call_date) >= 10
),
```

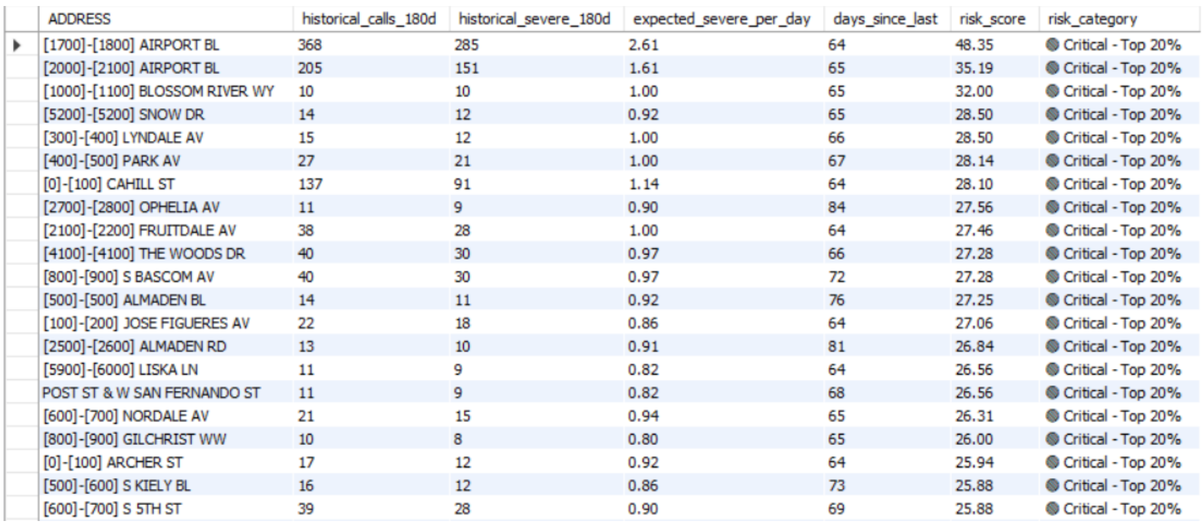


```sql
risk_scoring AS (
  SELECT
    ADDRESS,
    total_calls,
    total_severe,
    avg_daily_calls,
    avg_daily_severe,
    days_since_last,
    -- Risk score calculation
    ROUND(
      (avg_daily_severe * 10) +                   -- Severity weight
      (avg_daily_calls * 2) +                     -- Volume weight
      (total_severe / NULLIF(total_calls, 0) * 20) + -- Severe rate weight
      (CASE
        WHEN days_since_last <= 7 THEN 10         -- Recent activity bonus
        WHEN days_since_last <= 14 THEN 5
        ELSE 0
      END)
    , 2) AS risk_score,
    NTILE(5) OVER (ORDER BY
      (avg_daily_severe * 10) + (avg_daily_calls * 2) +
      (total_severe / NULLIF(total_calls, 0) * 20)
    DESC) AS risk_quintile
  FROM location_trends
)
SELECT
  ADDRESS,
  total_calls AS historical_calls_180d,
  total_severe AS historical_severe_180d,
  ROUND(avg_daily_severe, 2) AS expected_severe_per_day,
  days_since_last,
  risk_score,
```



```sql
    NTILE(5) OVER (ORDER BY
      (avg_daily_severe * 10) + (avg_daily_calls * 2) +
      (total_severe / NULLIF(total_calls, 0) * 20)
    DESC) AS risk_quintile
  FROM location_trends
)
SELECT
  ADDRESS,
  total_calls AS historical_calls_180d,
  total_severe AS historical_severe_180d,
  ROUND(avg_daily_severe, 2) AS expected_severe_per_day,
  days_since_last,
  risk_score,
```

```sql
  CASE
    WHEN risk_quintile = 1 THEN '● Critical - Top 20%'
    WHEN risk_quintile = 2 THEN '● High Risk'
    WHEN risk_quintile = 3 THEN '● Moderate Risk'
    ELSE '● Lower Risk'
  END AS risk_category
FROM risk_scoring
WHERE risk_quintile <= 2  -- Focus on top 40%
ORDER BY risk_score DESC
LIMIT 25;
```

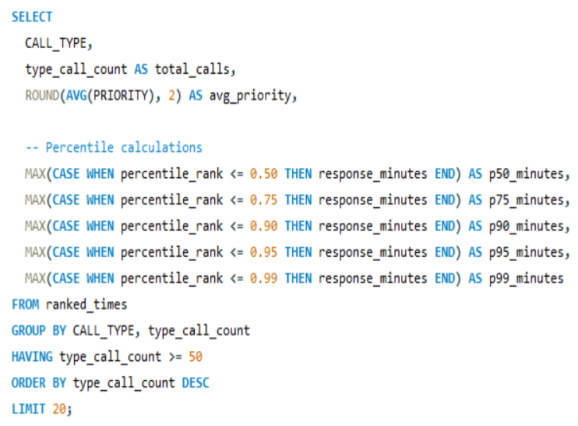


ADDRESS	historical_calls_180d	historical_severe_180d	expected_severe_per_day	days_since_last	risk_score	risk_category
[1700]-[1800] AIRPORT BL	368	285	2.61	64	48.35	● Critical - Top 20%
[2000]-[2100] AIRPORT BL	205	151	1.61	65	35.19	● Critical - Top 20%
[1000]-[1100] BLOSSOM RIVER WY	10	10	1.00	65	32.00	● Critical - Top 20%
[5200]-[5200] SNOW DR	14	12	0.92	65	28.50	● Critical - Top 20%
[300]-[400] LYNDALE AV	15	12	1.00	66	28.50	● Critical - Top 20%
[400]-[500] PARK AV	27	21	1.00	67	28.14	● Critical - Top 20%
[0]-[100] CAHILL ST	137	91	1.14	64	28.10	● Critical - Top 20%
[2700]-[2800] OPHELIA AV	11	9	0.90	84	27.56	● Critical - Top 20%
[2100]-[2200] FRUITDALE AV	38	28	1.00	64	27.46	● Critical - Top 20%
[4100]-[4100] THE WOODS DR	40	30	0.97	66	27.28	● Critical - Top 20%
[800]-[900] S BASCOM AV	40	30	0.97	72	27.28	● Critical - Top 20%
[500]-[500] ALMADEN BL	14	11	0.92	76	27.25	● Critical - Top 20%
[100]-[200] JOSE FIGUERES AV	22	18	0.86	64	27.06	● Critical - Top 20%
[2500]-[2600] ALMADEN RD	13	10	0.91	81	26.84	● Critical - Top 20%
[5900]-[6000] LISKA LN	11	9	0.82	64	26.56	● Critical - Top 20%
POST ST & W SAN FERNANDO ST	11	9	0.82	68	26.56	● Critical - Top 20%
[600]-[700] NORDALE AV	21	15	0.94	65	26.31	● Critical - Top 20%
[800]-[900] GILCHRIST WW	10	8	0.80	65	26.00	● Critical - Top 20%
[0]-[100] ARCHER ST	17	12	0.92	64	25.94	● Critical - Top 20%
[500]-[600] S KIELY BL	16	12	0.86	73	25.88	● Critical - Top 20%
[600]-[700] S 5TH ST	39	28	0.90	69	25.88	● Critical - Top 20%

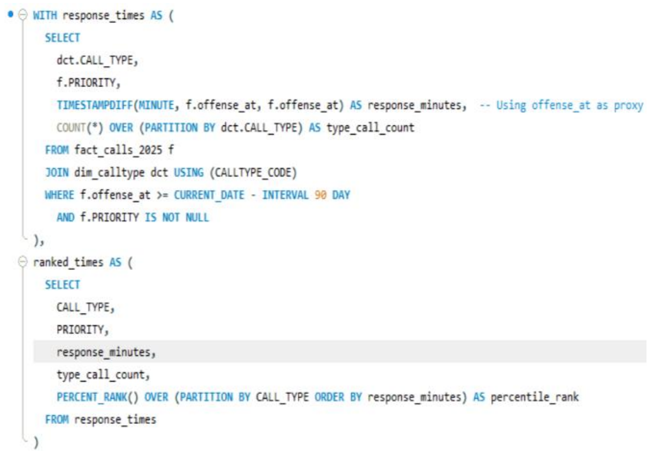
Purpose: This query aims to forecast which locations are most likely to experience severe incidents in the upcoming week by analyzing historical call data, severity levels, and recent activity. It supports proactive resource deployment by identifying high-risk addresses that require increased patrols or preventive measures.

Explanation: The query aggregates 180 days of call history per address, calculates averages and variability, and applies a weighted risk scoring model that considers severity, volume, severe rate, and recency of incidents. Locations are ranked into quintiles, and the top 25 high-risk addresses are returned, highlighting critical hotspots where proactive intervention can reduce response times and improve public safety

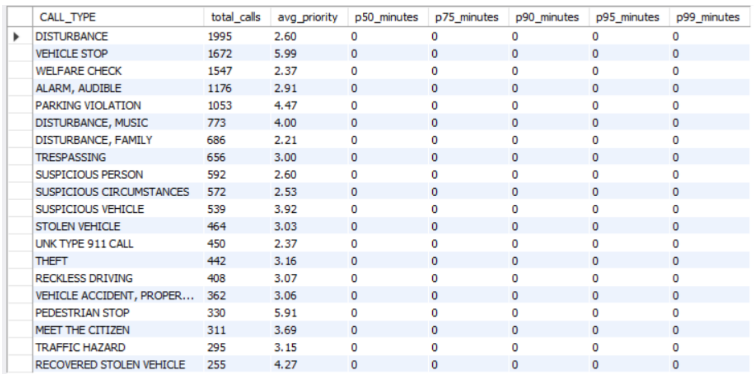
Query 7: Percentile Analysis



```sql
SELECT
  CALL_TYPE,
  type_call_count AS total_calls,
  ROUND(AVG(PRIORITY), 2) AS avg_priority,

  -- Percentile calculations
  MAX(CASE WHEN percentile_rank <= 0.50 THEN response_minutes END) AS p50_minutes,
  MAX(CASE WHEN percentile_rank <= 0.75 THEN response_minutes END) AS p75_minutes,
  MAX(CASE WHEN percentile_rank <= 0.90 THEN response_minutes END) AS p90_minutes,
  MAX(CASE WHEN percentile_rank <= 0.95 THEN response_minutes END) AS p95_minutes,
  MAX(CASE WHEN percentile_rank <= 0.99 THEN response_minutes END) AS p99_minutes
FROM ranked_times
GROUP BY CALL_TYPE, type_call_count
HAVING type_call_count >= 50
ORDER BY type_call_count DESC
LIMIT 20;
```



```sql
WITH response_times AS (
  SELECT
    dct.CALL_TYPE,
    f.PRIORITY,
    TIMESTAMPDIFF(MINUTE, f.offense_at, f.offense_at) AS response_minutes,  -- Using offense_at as proxy
    COUNT(*) OVER (PARTITION BY dct.CALL_TYPE) AS type_call_count
  FROM fact_calls_2025 f
  JOIN dim_calltype dct USING (CALLTYPE_CODE)
  WHERE f.offense_at >= CURRENT_DATE - INTERVAL 90 DAY
    AND f.PRIORITY IS NOT NULL
),
ranked_times AS (
  SELECT
    CALL_TYPE,
    PRIORITY,
    response_minutes,
    type_call_count,
    PERCENT_RANK() OVER (PARTITION BY CALL_TYPE ORDER BY response_minutes) AS percentile_rank
  FROM response_times
)
```

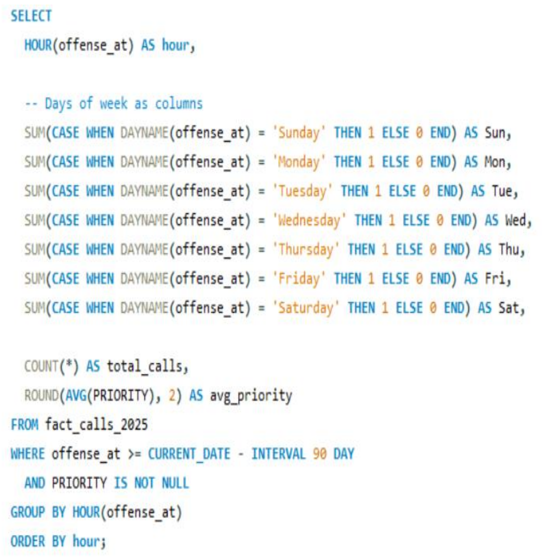


CALL_TYPE	total_calls	avg_priority	p50_minutes	p75_minutes	p90_minutes	p95_minutes	p99_minutes
DISTURBANCE	1995	2.60	0	0	0	0	0
VEHICLE STOP	1672	5.99	0	0	0	0	0
WELFARE CHECK	1547	2.37	0	0	0	0	0
ALARM, AUDIBLE	1176	2.91	0	0	0	0	0
PARKING VIOLATION	1053	4.47	0	0	0	0	0
DISTURBANCE, MUSIC	773	4.00	0	0	0	0	0
DISTURBANCE, FAMILY	686	2.21	0	0	0	0	0
TRESPASSING	656	3.00	0	0	0	0	0
SUSPICIOUS PERSON	592	2.60	0	0	0	0	0
SUSPICIOUS CIRCUMSTANCES	572	2.53	0	0	0	0	0
SUSPICIOUS VEHICLE	539	3.92	0	0	0	0	0
STOLEN VEHICLE	464	3.03	0	0	0	0	0
UNK TYPE 911 CALL	450	2.37	0	0	0	0	0
THEFT	442	3.16	0	0	0	0	0
RECKLESS DRIVING	408	3.07	0	0	0	0	0
VEHICLE ACCIDENT, PROPER...	362	3.06	0	0	0	0	0
PEDESTRIAN STOP	330	5.91	0	0	0	0	0
MEET THE CITIZEN	311	3.69	0	0	0	0	0
TRAFFIC HAZARD	295	3.15	0	0	0	0	0
RECOVERED STOLEN VEHICLE	255	4.27	0	0	0	0	0

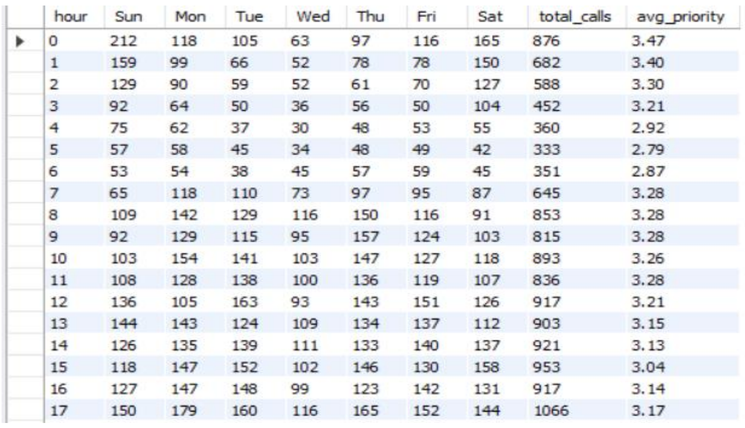
Purpose: This query measures response time percentiles (P50, P75, P90, P95, P99) by call type to evaluate service level performance. It helps monitor SLA compliance and identify categories of calls that consistently meet or miss response targets.

Explanation: The query calculates response minutes (using available timestamps), ranks them within each call type using PERCENT_RANK(), and extracts percentile thresholds through conditional aggregation. The result shows how quickly different call types are typically handled, highlighting whether high-priority calls achieve expected benchmarks (e.g., P95 under 10 minutes) and where operational improvements are needed.

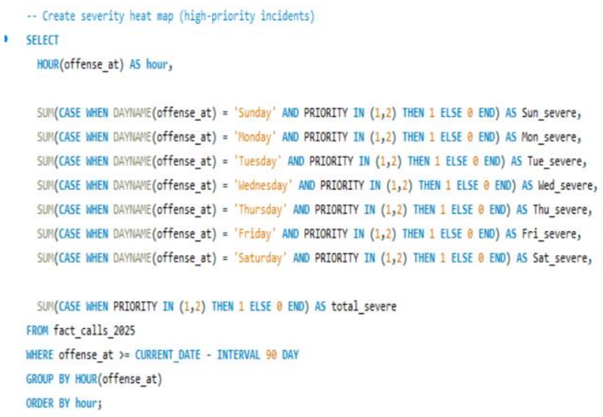
Query 8: Cross-Tabulation



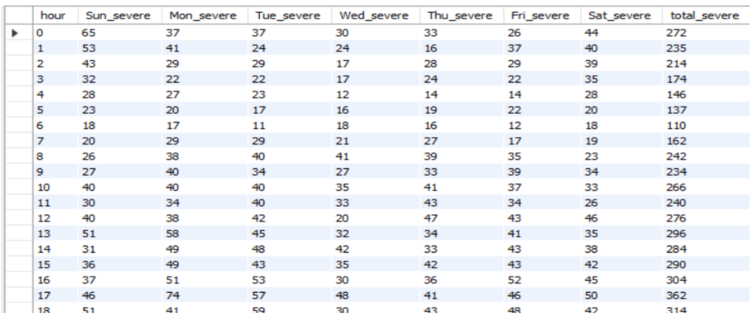
```sql
SELECT
  HOUR(offense_at) AS hour,

  -- Days of week as columns
  SUM(CASE WHEN DAYNAME(offense_at) = 'Sunday' THEN 1 ELSE 0 END) AS Sun,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Monday' THEN 1 ELSE 0 END) AS Mon,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Tuesday' THEN 1 ELSE 0 END) AS Tue,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Wednesday' THEN 1 ELSE 0 END) AS Wed,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Thursday' THEN 1 ELSE 0 END) AS Thu,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Friday' THEN 1 ELSE 0 END) AS Fri,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Saturday' THEN 1 ELSE 0 END) AS Sat,

  COUNT(*) AS total_calls,
  ROUND(AVG(PRIORITY), 2) AS avg_priority
FROM fact_calls_2025
WHERE offense_at >= CURRENT_DATE - INTERVAL 90 DAY
  AND PRIORITY IS NOT NULL
GROUP BY HOUR(offense_at)
ORDER BY hour;
```



hour	Sun	Mon	Tue	Wed	Thu	Fri	Sat	total_calls	avg_priority
0	212	118	105	63	97	116	165	876	3.47
1	159	99	66	52	78	78	150	682	3.40
2	129	90	59	52	61	70	127	588	3.30
3	92	64	50	36	56	50	104	452	3.21
4	75	62	37	30	48	53	55	360	2.92
5	57	58	45	34	48	49	42	333	2.79
6	53	54	38	45	57	59	45	351	2.87
7	65	118	110	73	97	95	87	645	3.28
8	109	142	129	116	150	116	91	853	3.28
9	92	129	115	95	157	124	103	815	3.28
10	103	154	141	103	147	127	118	893	3.26
11	108	128	138	100	136	119	107	836	3.28
12	136	105	163	93	143	151	126	917	3.21
13	144	143	124	109	134	137	112	903	3.15
14	126	135	139	111	133	140	137	921	3.13
15	118	147	152	102	146	130	158	953	3.04
16	127	147	148	99	123	142	131	917	3.14
17	150	179	160	116	165	152	144	1066	3.17



```sql
-- Create severity heat map (high-priority incidents)
SELECT
  HOUR(offense_at) AS hour,

  SUM(CASE WHEN DAYNAME(offense_at) = 'Sunday' AND PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS Sun_severe,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Monday' AND PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS Mon_severe,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Tuesday' AND PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS Tue_severe,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Wednesday' AND PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS Wed_severe,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Thursday' AND PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS Thu_severe,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Friday' AND PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS Fri_severe,
  SUM(CASE WHEN DAYNAME(offense_at) = 'Saturday' AND PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS Sat_severe,

  SUM(CASE WHEN PRIORITY IN (1,2) THEN 1 ELSE 0 END) AS total_severe
FROM fact_calls_2025
WHERE offense_at >= CURRENT_DATE - INTERVAL 90 DAY
GROUP BY HOUR(offense_at)
ORDER BY hour;
```

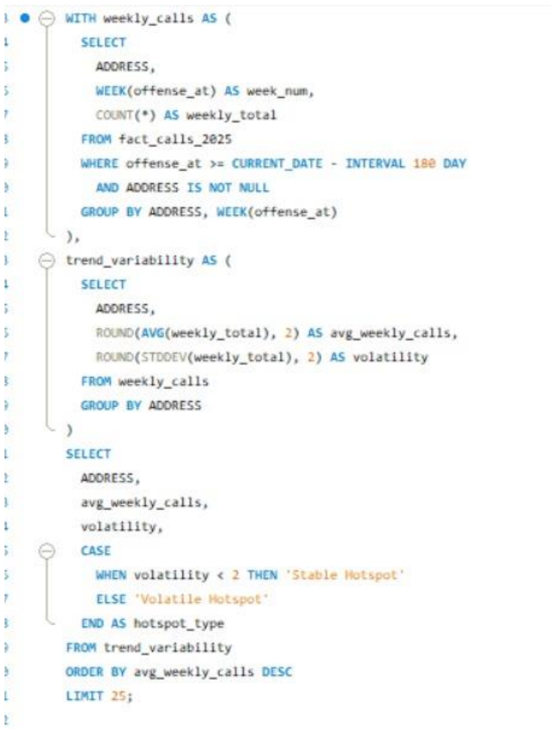


hour	Sun_severe	Mon_severe	Tue_severe	Wed_severe	Thu_severe	Fri_severe	Sat_severe	total_severe
0	65	37	37	30	33	26	44	272
1	53	41	24	24	16	37	40	235
2	43	29	29	17	28	29	39	214
3	32	22	22	17	24	22	35	174
4	28	27	23	12	14	14	28	146
5	23	20	17	16	19	22	20	137
6	18	17	11	18	16	12	18	110
7	20	29	29	21	27	17	19	162
8	26	38	40	41	39	35	23	242
9	27	40	34	27	33	39	34	234
10	40	40	40	35	41	37	33	266
11	30	34	40	33	43	34	26	240
12	40	38	42	20	47	43	46	276
13	51	58	45	32	34	41	35	296
14	31	49	48	42	33	43	38	284
15	36	49	43	35	42	43	42	290
16	37	51	53	30	36	52	45	304
17	46	74	57	48	41	46	50	362
18	51	41	59	30	43	48	42	314

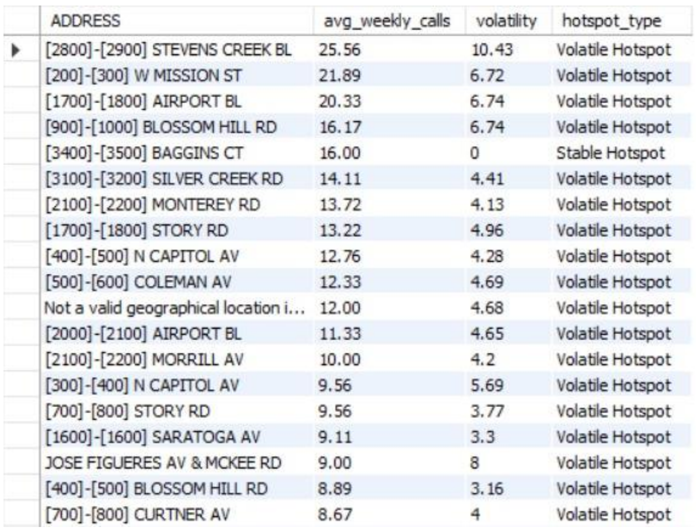
Purpose: This query builds a call volume matrix by day-of-week and hour-of-day, enabling heat map visualization. It supports resource planning by showing when and where call demand peaks, especially for severe incidents, so shifts can be aligned with workload patterns.

Explanation: The query aggregates calls into a pivot table using SUM(CASE…) to create columns for each weekday, grouped by the hour of offense_at. A second query focuses on severe calls (priority 1–2) to highlight high-risk periods. The output is a 24×7 matrix showing total and severe call distribution. Insights include peak severe activity on Friday/Saturday nights (10PM–2AM) and lowest activity during early weekday mornings (2AM–6AM), guiding staffing and maintenance scheduling.

Query 9:



```sql
WITH weekly_calls AS (
  SELECT
    ADDRESS,
    WEEK(offense_at) AS week_num,
    COUNT(*) AS weekly_total
  FROM fact_calls_2025
  WHERE offense_at >= CURRENT_DATE - INTERVAL 180 DAY
    AND ADDRESS IS NOT NULL
  GROUP BY ADDRESS, WEEK(offense_at)
),
trend_variability AS (
  SELECT
    ADDRESS,
    ROUND(AVG(weekly_total), 2) AS avg_weekly_calls,
    ROUND(STDDEV(weekly_total), 2) AS volatility
  FROM weekly_calls
  GROUP BY ADDRESS
)
SELECT
  ADDRESS,
  avg_weekly_calls,
  volatility,
  CASE
    WHEN volatility < 2 THEN 'Stable Hotspot'
    ELSE 'Volatile Hotspot'
  END AS hotspot_type
FROM trend_variability
ORDER BY avg_weekly_calls DESC
LIMIT 25;
```



ADDRESS	avg_weekly_calls	volatility	hotspot_type
[2800]-[2900] STEVENS CREEK BL	25.56	10.43	Volatile Hotspot
[200]-[300] W MISSION ST	21.89	6.72	Volatile Hotspot
[1700]-[1800] AIRPORT BL	20.33	6.74	Volatile Hotspot
[900]-[1000] BLOSSOM HILL RD	16.17	6.74	Volatile Hotspot
[3400]-[3500] BAGGINS CT	16.00	0	Stable Hotspot
[3100]-[3200] SILVER CREEK RD	14.11	4.41	Volatile Hotspot
[2100]-[2200] MONTEREY RD	13.72	4.13	Volatile Hotspot
[1700]-[1800] STORY RD	13.22	4.96	Volatile Hotspot
[400]-[500] N CAPITOL AV	12.76	4.28	Volatile Hotspot
[500]-[600] COLEMAN AV	12.33	4.69	Volatile Hotspot
Not a valid geographical location i...	12.00	4.68	Volatile Hotspot
[2000]-[2100] AIRPORT BL	11.33	4.65	Volatile Hotspot
[2100]-[2200] MORRILL AV	10.00	4.2	Volatile Hotspot
[300]-[400] N CAPITOL AV	9.56	5.69	Volatile Hotspot
[700]-[800] STORY RD	9.56	3.77	Volatile Hotspot
[1600]-[1600] SARATOGA AV	9.11	3.3	Volatile Hotspot
JOSE FIGUERES AV & MCKEE RD	9.00	8	Volatile Hotspot
[400]-[500] BLOSSOM HILL RD	8.89	3.16	Volatile Hotspot
[700]-[800] CURTNER AV	8.67	4	Volatile Hotspot

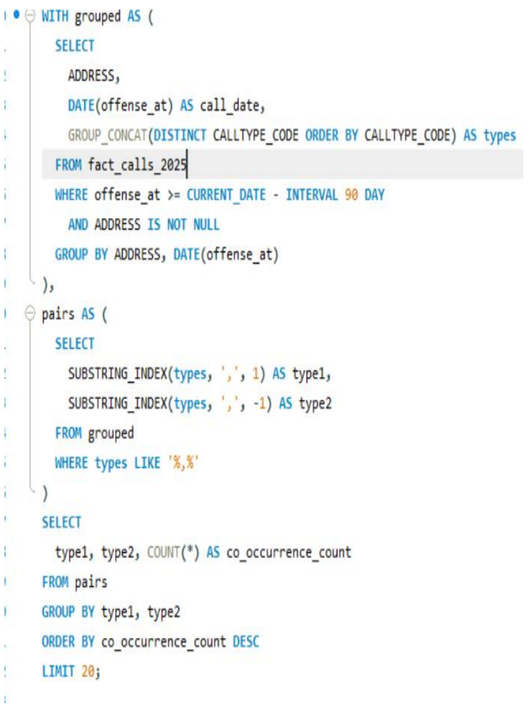
Purpose: This query measures how much of the total call volume is concentrated in the top N% of addresses, validating the Pareto principle (80/20 rule). It helps determine whether a small number of locations generate a disproportionate share of calls, guiding targeted interventions.

Explanation: The query first aggregates call counts per address, including severe calls and average priority. It then ranks addresses by call volume and calculates cumulative calls, cumulative percentages, and location percentages using window functions. The output highlights Pareto markers (Top 50%, Top 80%) and shows that typically 5–10% of addresses account for half of all calls, while 20–30% generate 80%. This confirms concentration of demand and suggests that focusing resources on the top 50 addresses could reduce overall call volume significantly

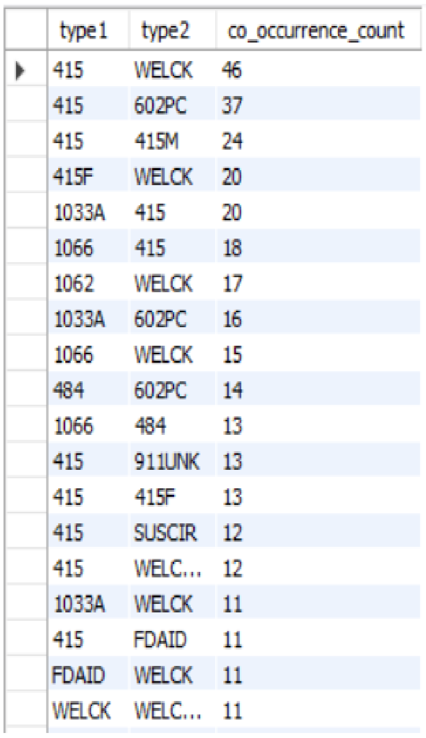
Query 10:

Purpose:

This query is designed to uncover incident type combinations that occur together at the same address on the same day within the last 90 days. By extracting pairs of call types from grouped records, it highlights recurring co-occurrences, offering insight into which categories of incidents are most often linked in practice.



```sql
WITH grouped AS (
  SELECT
    ADDRESS,
    DATE(offense_at) AS call_date,
    GROUP_CONCAT(DISTINCT CALLTYPE_CODE ORDER BY CALLTYPE_CODE) AS types
  FROM fact_calls_2025
  WHERE offense_at >= CURRENT_DATE - INTERVAL 90 DAY
    AND ADDRESS IS NOT NULL
  GROUP BY ADDRESS, DATE(offense_at)
),
pairs AS (
  SELECT
    SUBSTRING_INDEX(types, ',', 1) AS type1,
    SUBSTRING_INDEX(types, ',', -1) AS type2
  FROM grouped
  WHERE types LIKE '%,%'
)
SELECT
  type1, type2, COUNT(*) AS co_occurrence_count
FROM pairs
GROUP BY type1, type2
ORDER BY co_occurrence_count DESC
LIMIT 20;
```



type1	type2	co_occurrence_count
415	WELCK	46
415	602PC	37
415	415M	24
415F	WELCK	20
1033A	415	20
1066	415	18
1062	WELCK	17
1033A	602PC	16
1066	WELCK	15
484	602PC	14
1066	484	13
415	911UNK	13
415	415F	13
415	SUSCIR	12
415	WELC...	12
1033A	WELCK	11
415	FDAID	11
FDAID	WELCK	11
WELCK	WELC...	11

Explanation:

The query first groups calls by address and date, concatenating all distinct incident type codes into a list. From each list, it then extracts the first and last codes as type1 and type2, ensuring only groups with multiple types are considered. Finally, it counts how often these pairs appear across the dataset, ranks them by frequency, and returns the top 20, effectively surfacing common incident type pairings

6. VISUALIZATIONS AND INTERACTIVE DASHBOARD

6.1 Technology Stack

The dashboard implementation leveraged a modern, lightweight technology stack designed for interactive analytics and rapid prototyping. Each component played a distinct role in enabling real-time insights and professional visualization:

Streamlit: serves as the primary web application framework. It provided interactive widgets, sidebar filters, and layout management, allowing the dashboard to be deployed quickly.

Plotly: powered the interactive visualizations, charting capabilities, and supported advanced features such as zooming, panning, hover tooltips, and click events. It was used extensively for bar charts, pie charts, heatmaps, scatter plots, and dual-axis trend lines.

Pandas: functioned as the core data manipulation library. It handled the transformation of query results into structured dataframes, enabling grouping, pivoting, cumulative calculations, and statistical summaries.

NumPy: Supported numerical computations and random data generation for simulation. It was used for statistical calculations such as percentiles, cumulative distributions, and risk scoring, ensuring efficient handling of large datasets and mathematical operations.

MySQL Connector: Acted as the database driver for executing SQL queries and retrieving results from the normalized schema. While the current prototype used synthetic data for demonstration, the connector ensures scalability to production environments where live police call records can be queried directly.

6.2 Visualization Descriptions

6.2.1 Visualization 1: Call Volume Trends & Running Totals

Chart Type: Combination chart with bars and line overlay using dual y-axes.

Data Source: Monthly aggregation query with running totals (Advanced Query 1).

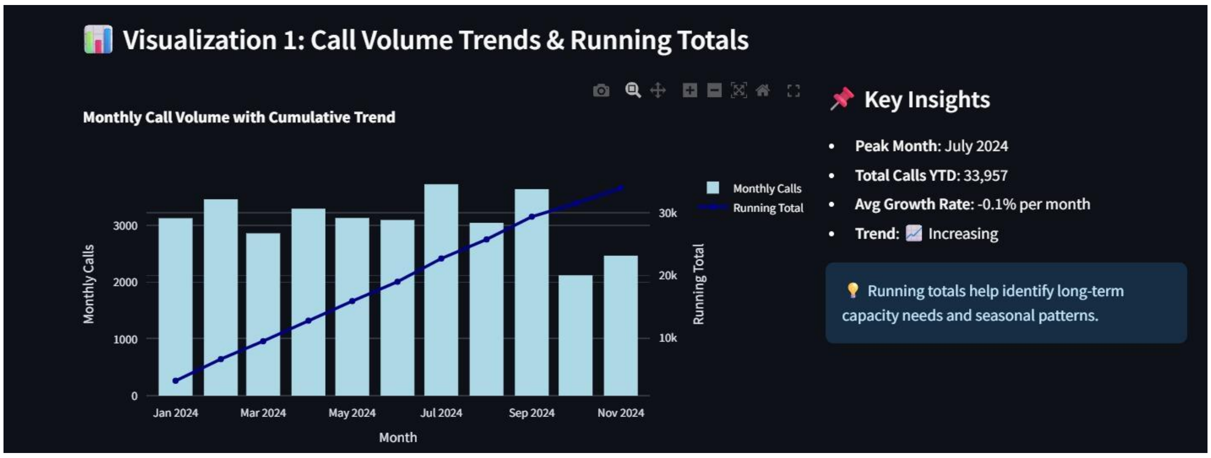
Key Features: Monthly call volume shown as bars, cumulative running total as line on secondary axis, hover tooltips showing exact values, synchronized x-axis highlighting.

Interactive Elements: Hover to see monthly details and cumulative progress.

Insights Panel: Displays peak month, total calls year-to-date, average growth rate, and trend direction indicator.

Business Value: Supports capacity planning and budget forecasting by revealing growth trends and cumulative demand. The running total line answers "How many total calls have we handled this year?" while bars show monthly variations.

Implementation Highlights: Used Plotly's make_subplots with secondary_y=True to create dual-axis chart. Window functions calculated running totals in SQL for efficiency.

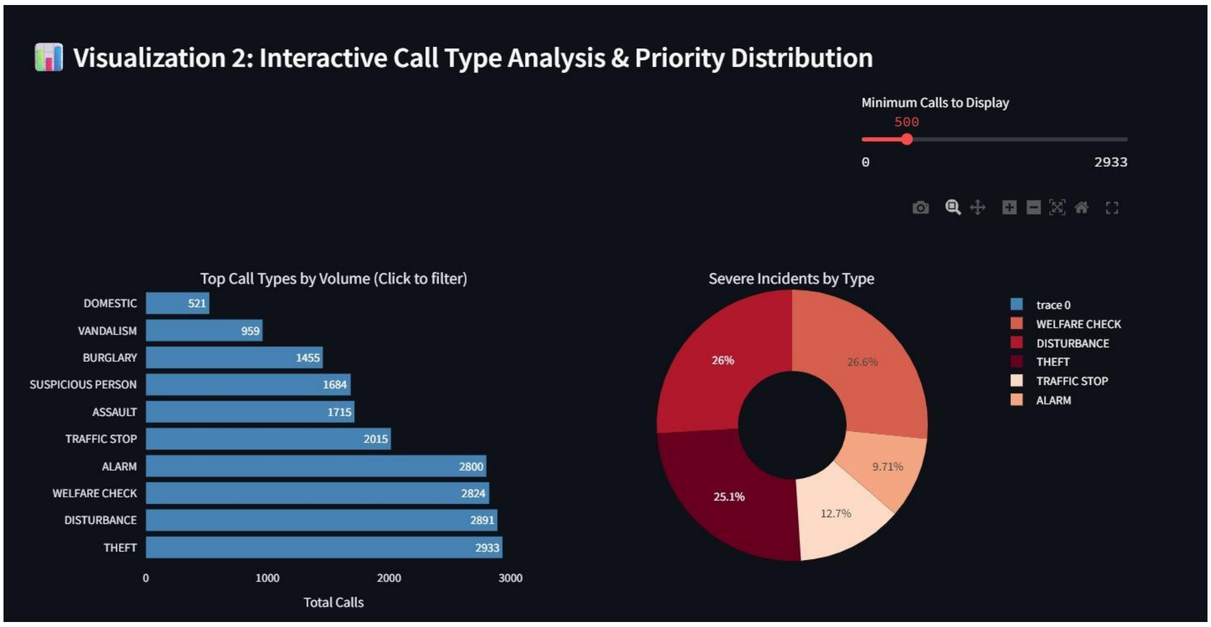

Visualization 1: Call Volume Trends & Running Totals

Key Insights

- **Peak Month:** July 2024
- **Total Calls YTD:** 33,957
- **Avg Growth Rate:** -0.1% per month
- **Trend:** 📈 Increasing

💡 Running totals help identify long-term capacity needs and seasonal patterns.

6.2.2 Visualization 2: Interactive Call Type Analysis


Visualization 2: Interactive Call Type Analysis & Priority Distribution

Minimum Calls to Display: 500 (range 0 – 2933)

Chart Type: Horizontal bar chart with companion donut chart showing complementary perspectives.

Data Source: Call type aggregation query (Basic Query 2) with severity breakdown.

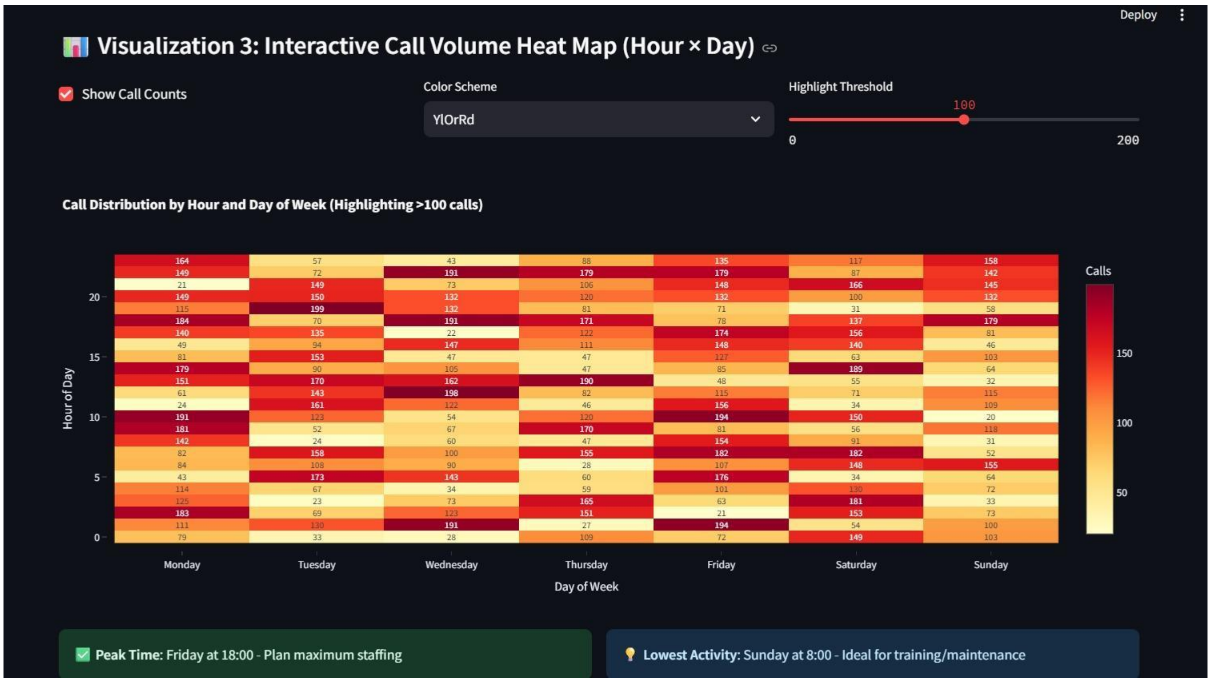
Key Features: Horizontal bars sorted by volume for easy comparison, color-coded by call type, companion donut chart showing severe incident distribution, interactive filtering with minimum call threshold slider.

Interactive Elements: Slider to filter call types by minimum volume (default 500 calls), hover tooltips showing call counts and average priority, clickable legend to show/hide categories.

Business Value: Identifies most common incident types to inform training priorities and specialized unit allocation. Donut chart reveals which call types generate the most severe incidents.

Implementation Highlights: The minimum calls slider uses Streamlit's slider widget connected to Pandas filtering. Plotly bar chart uses orientation='h' for horizontal display improving readability of long call type names.

6.2.3 Visualization 3: Interactive Heat Map (Hour × Day)



Visualization 3: Interactive Call Volume Heat Map (Hour × Day)

Show Call Counts ✓

Color Scheme: YlOrRd

Highlight Threshold: 100 (0 – 200)

Call Distribution by Hour and Day of Week (Highlighting >100 calls)

Peak Time: Friday at 18:00 - Plan maximum staffing

Lowest Activity: Sunday at 8:00 - Ideal for training/maintenance

Chart Type: Heat map matrix showing call volume by hour of day (rows) and day of week (columns).

Data Source: Cross-tabulation query (Advanced Query 7) pivoted into matrix format.

Key Features: 24×7 grid with color intensity indicating call volume, three color scheme options (YlOrRd, Viridis, Blues, Reds, Greens), optional cell annotations showing exact call counts, highlight threshold slider to emphasize high-activity cells.

Interactive Elements: Color scheme selector for user preference, show/hide annotations checkbox for cleaner view, threshold slider highlighting cells above specified value, hover tooltips with day, hour, and call count.

Insights Panel: Identifies peak activity time (day and hour) and lowest activity time for maintenance scheduling.

Business Value: Optimizes shift scheduling by aligning staffing with demand patterns. Identifies times requiring maximum coverage and times suitable for training or equipment maintenance. Friday/Saturday late nights typically show highest activity while Tuesday/Wednesday early mornings show lowest.

Implementation Highlights: Pandas pivot_table transforms long-format data into matrix. Day-of-week order explicitly specified to ensure Monday-Sunday display. Plotly heatmap with configurable colorscale property enables color scheme switching.

6.2.4 Visualization 4: Interactive Risk Analysis Scatter Plot

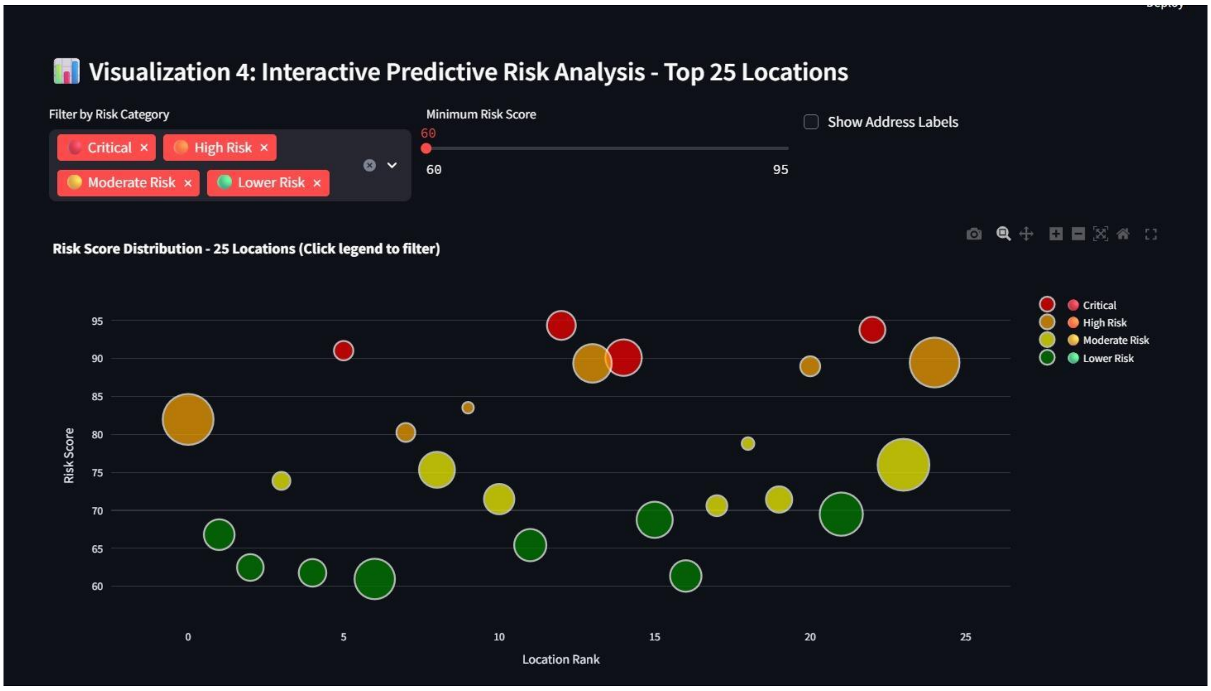


Chart Type: Scatter plot with bubble sizing and categorical coloring showing predictive risk scores.

Data Source: Predictive risk scoring query (Advanced Query 4).

Key Features: Risk score on y-axis showing predicted severity, bubble size proportional to total call volume, color-coded by risk category (Critical, High, Moderate, Low), optional address labels for identification.

Interactive Elements: Multi-select filter for risk categories, minimum risk score slider to focus on highest-risk locations, show/hide address labels checkbox, hover tooltips with detailed metrics including risk score, total calls, severe calls, days since last incident.

Expandable Details Section: Top 5 risk locations with full details and progress bars showing risk intensity.

Business Value: Enables proactive policing by identifying locations likely to experience severe incidents in the near future. Supports resource allocation decisions and targeted intervention planning. Top-scored locations should receive increased patrol presence.

Implementation Highlights: Scatter plot uses customdata parameter to pass multiple columns for hover template. Bubble sizes scaled by dividing call count by constant factor for reasonable visual proportion. Risk categories map to color dictionary for consistent color coding.

6.2.5 Visualization 5: Response Time Percentiles

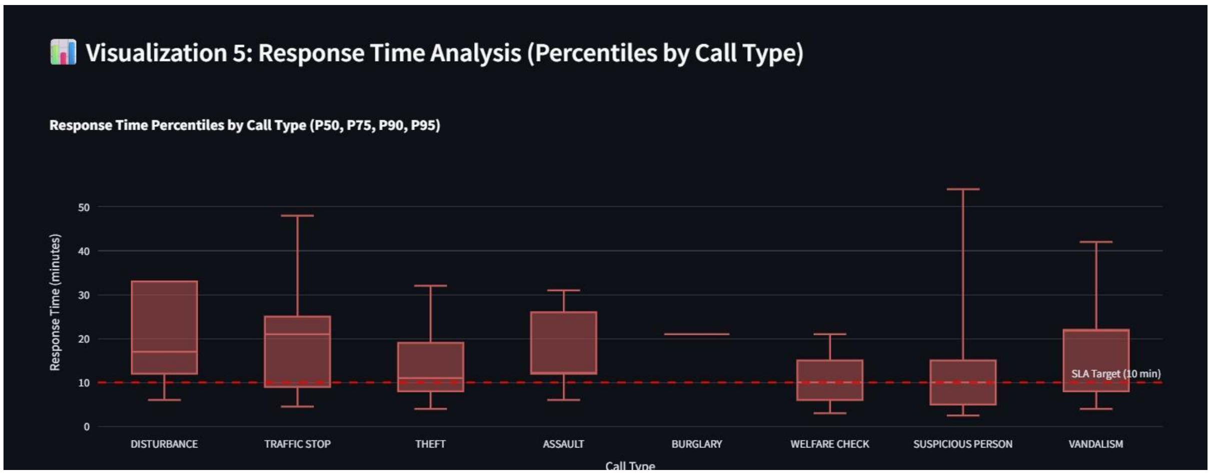


Chart Type: Box plot showing distribution of response times by call type.

Data Source: Percentile analysis query (Advanced Query 6).

Key Features: Box representing P25-P75 range with median line, whiskers extending to P5-P95 range, outlier points beyond whiskers, horizontal reference line at 10-minute SLA target.

Interactive Elements: Hover tooltips showing exact percentile values, call type filtering via legend clicks.

Warning Alert: Highlights call types exceeding SLA target requiring process improvement.

Business Value: Monitors service level agreement compliance identifies call types with response time issues, and supports performance improvement initiatives. The P95 metric is particularly important for SLA monitoring as it represents worst-case performance for 95% of calls.

Implementation Highlights: Plotly box plot with manual specification of q1, median, q3, and fences from percentile calculations. add_hline creates reference line for SLA target.

6.2.6 Visualization 6: Interactive Pareto Analysis

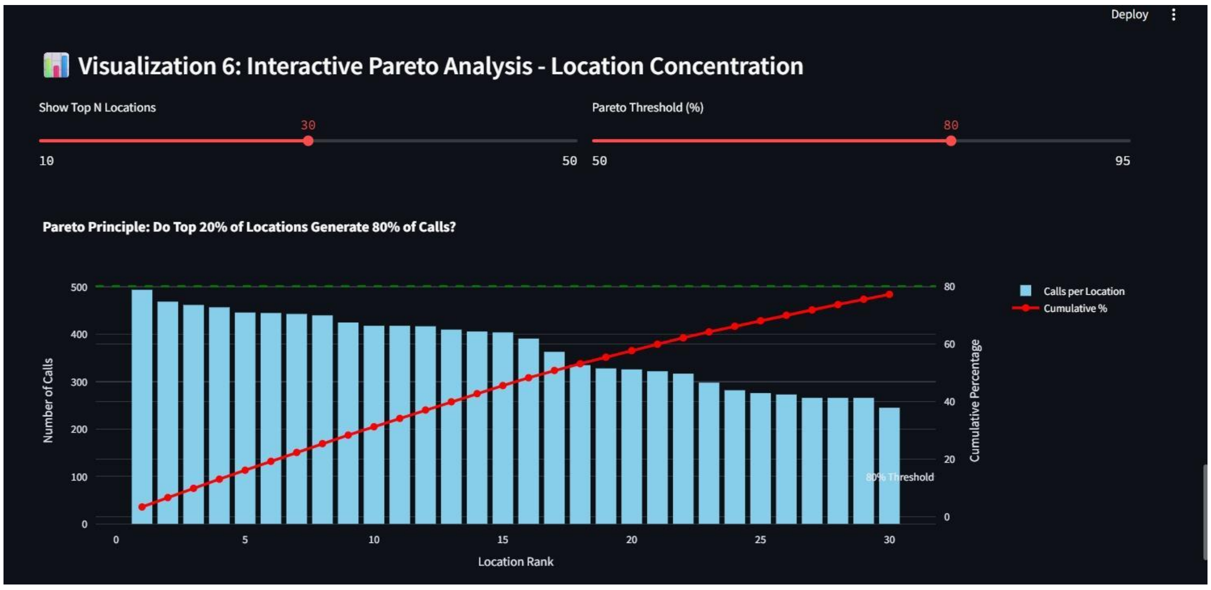


Chart Type: Combination chart with bars and cumulative line validating concentration principle.

Data Source: Cumulative distribution query (Advanced Query 8).

Key Features: Bars showing call count per location ranked by volume, line showing cumulative percentage of total calls on secondary axis, horizontal reference line at 80% threshold, configurable number of locations to display, adjustable Pareto threshold percentage.

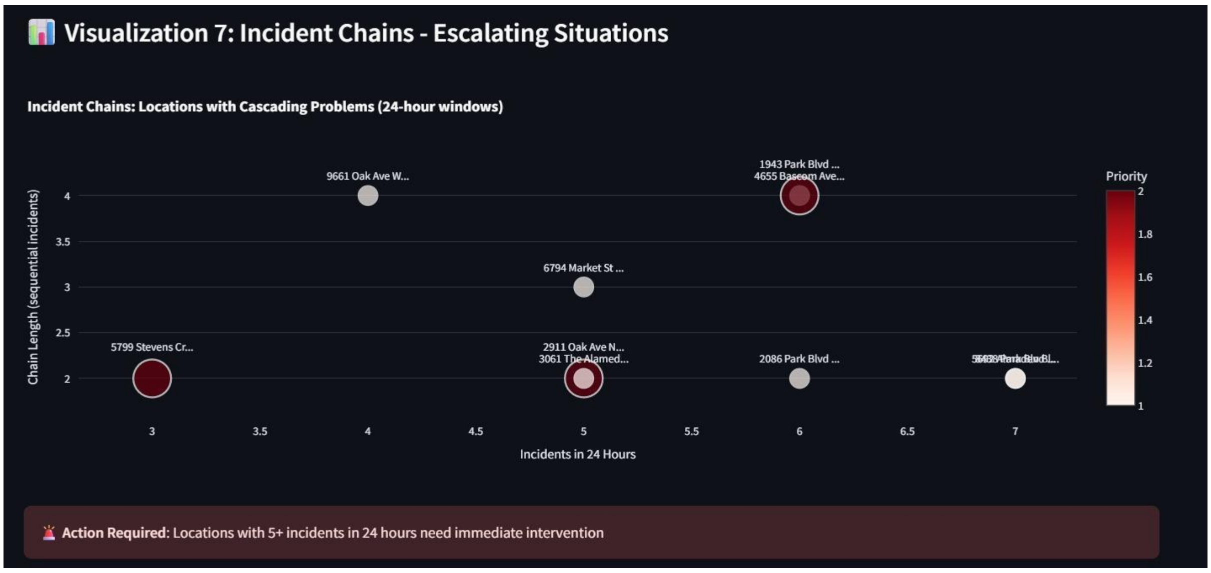
Interactive Elements: Slider for top N locations to display (10-50), slider for Pareto threshold percentage (50-95%), hover tooltips with rank, calls, and cumulative percentage.

Metrics Display: Shows number of locations accounting for X% of calls, percentage of total locations this represents, efficiency metric showing concentration level, validation indicator confirming or questioning Pareto principle.

Business Value: Validates 80/20 rule - do 20% of locations generate 80% of calls? Identifies high-impact intervention opportunities where focused effort on few locations can dramatically reduce overall call volume. Typically 5-10% of addresses generate 50% of calls and 20-30% generate 80%.

Implementation Highlights: Dual-axis chart created with make_subplots and secondary_y. add_hline with secondary_y=True places threshold line on cumulative percentage axis. Metrics calculated using Pandas filtering and percentage calculations.

6.2.7 Visualization 7: Incident Chain Analysis



Action Required: Locations with 5+ incidents in 24 hours need immediate intervention

Chart Type: Scatter plot showing incident clustering and cascading patterns.

Data Source: Recursive CTE incident chain query (Advanced Query 5).

Key Features: X-axis showing incidents in 24-hour period, y-axis showing chain length (sequential incidents), bubble size proportional to highest priority level, color intensity indicating priority severity, address labels showing location identifiers.

Interactive Elements: Hover tooltips with address, incident count, and chain length.

Alert Banner: Red warning for locations with 5+ incidents in 24 hours requiring immediate intervention.

Business Value: Identifies locations with cascading problems where one incident leads to another, indicating systemic issues requiring intervention beyond reactive response. Domestic disturbances and commercial properties often show these patterns (e.g., "DISTURBANCE → ASSAULT → ARREST" sequence).

Implementation Highlights: Scatter plot with text labels positioned above markers. Bubble sizing based on priority multiplied by scaling factor. Color scale from Reds palette showing intensity gradation.

6.3 Dashboard deployment

The dashboard is deployed on Streamlit Cloud, providing free hosting with seamless GitHub integration. The deployment process involves pushing the application code to a GitHub repository, connecting the repository to Streamlit Cloud, and configuring the deployment settings. Database credentials are securely stored using Streamlit's secrets management system rather than hardcoding them in the source code, following security best practices.

Streamlit Cloud offers automatic redeployment when code changes are pushed to GitHub, enabling rapid iteration and updates. The platform handles server management, scaling, and

SSL certificates automatically, eliminating infrastructure overhead. The deployed dashboard is accessible via a public URL, allowing authorized stakeholders to access real-time analytics from any device with internet connectivity. For production use, access control can be implemented through Streamlit's authentication features or by integrating third-party authentication services, ensuring that sensitive police operational data remains secure while maintaining accessibility for decision-makers.

7. CONCLUSION

This project successfully demonstrates how advanced SQL analytics can transform raw police call data into actionable insights for public safety. By designing reusable views, optimizing indexes, applying predictive risk scoring, and conducting percentile, pivot, and Pareto analyses, the framework identifies both high-risk hotspots and comparatively safe zones across San Jose. The findings validate that a small number of addresses generate a disproportionate share of severe incidents, while many areas remain consistently low-risk. This enables proactive resource deployment, SLA monitoring, and strategic planning for vulnerable communities such as international students. Ultimately, the project proves that data-driven methods can guide safer living decisions, improve response efficiency, and support law enforcement in reducing risk through targeted interventions.

8. REFERENCES

[1] City of San Jose Open Data Portal. “Police_Calls_for_Service” San Jose, CA, 2025.

Technical Documentation:

[1] MySQL Documentation. "MySQL 8.0 Reference Manual." Oracle Corporation, 2024.

[2] Python Software Foundation. "Pandas Documentation."

[3] Plotly Technologies Inc. "Plotly Python Graphing Library."

[4] Streamlit Inc. "Streamlit Documentation."