

# **Database Automation**

## **Assignment 3**

### **Project Report**

Submitted by

Nikhil Shankar C S

9026254

### **Table of Contents**

Executive Summary .....	2
Database Setup .....	3
Schema Design.....	3

Indexes for Performance .....	3
Data Volume .....	4
Web Application .....	5
Search Interface.....	5
Statistics Dashboard .....	6
Automated Testing.....	8
Selenium Tests.....	8
CI/CD Pipeline.....	8
GitHub Actions .....	8
Scalability Considerations.....	9
Challenges and Solutions .....	10
Appendices .....	10
A. Technologies Used.....	10
B. Repository Structure.....	11

## Executive Summary

This project demonstrates a complete database automation pipeline for NYC 311 service request data. I built an ETL system that loads large CSV files into MySQL, created a web interface for searching for complaints, and set up automated testing with CI/CD deployment.

### Key Achievements:

- Successfully loaded 3,37,137 complaint records from January 2025
- Built a Flask web app with search and aggregate features
- Implemented 4 database indexes for fast queries
- Created 10 automated browser tests using selenium
- Set up GitHub Actions for continuous integration

# Database Setup

## Schema Design

I created a MySQL database with a service\_requests table containing 9 columns to store complaint information.

```
mysql> describe service_requests;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| unique_key     | bigint        | NO   | PRI | NULL    |       |
| created_date   | datetime      | NO   | MUL | NULL    |       |
| closed_date    | datetime      | YES  |     | NULL    |       |
| agency         | varchar(16)   | YES  | MUL | NULL    |       |
| complaint_type | varchar(128)  | YES  |     | NULL    |       |
| descriptor     | varchar(255)  | YES  |     | NULL    |       |
| borough        | varchar(32)   | YES  | MUL | NULL    |       |
| latitude       | decimal(9,6)  | YES  |     | NULL    |       |
| longitude      | decimal(9,6)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.00 sec)
```

Figure 1: Database schema showing table structure and data types

## Indexes for Performance

I added 4 indexes to make searches faster:

- 1. **idx\_created\_date** - Speeds up date range searches
- 2. **idx\_borough** - Makes borough filtering fast
- 3. **idx\_agency** - Quick lookups by department
- 4. **idx\_date\_borough** - Combined filter for date + location

service_requests	1	idx_created_date	1	created_date	A	281107	NULL	NULL		BTREE		
service_requests	1	idx_borough	1	borough	A	5	NULL	NULL	YES	BTREE		
service_requests	1	idx_agency	1	agency	A	13	NULL	NULL	YES	BTREE		
service_requests	1	idx_date_borough	1	created_date	A	267301	NULL	NULL		BTREE		

Figure 2: Indexes defined in the service\_requests table

```
mysql> EXPLAIN SELECT * FROM
  -> service_requests WHERE borough = 'BROOKLYN' LIMIT 10;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows
1	SIMPLE	service_requests	NULL	ref	idx_borough	idx_borough	131	const	173

1 row in set, 1 warning (0.01 sec)

mysql>

Figure 3: Confirming indexes is working using EXPLAIN command

## Data Volume

### January 2025 Dataset:

- Total records loaded: 3,37,137+
- File size: ~100 MB
- Load time: ~2-3 minutes
- Processing speed: ~1,000-1,500 rows/second

### ETL Features:

- Processes 10,000 rows at a time (prevents memory issues)
- Fixes missing borough names (sets to "UNKNOWN")
- Handles invalid dates properly
- Converts NaN values to NULL for MySQL
- Shows statistics when finished

```
infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing
nd as-expected, please specify a format.
df['Closed Date'] = pd.to_datetime(df['Closed Date'], errors='coerce')
Inserted 7137 rows (Total: 337137)

=====
ETL COMPLETE - Statistics:
Total rows processed: 337137
Errors encountered: 0
Duration: 236.49 seconds
Speed: 1425.59 rows/second
● =====

PS D:\Conestoga\Course2-DevOps\Sem2\Database Automation\DatabaseAutomation-GitRepo> git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)
```

Figure 4: ETL statistics after loading Jan 2025

## Data Cleaning

The script automatically fixes these issues:

- Empty borough fields → "UNKNOWN"
- Invalid date formats → NULL
- Missing coordinates → NULL
- NaN values → Proper NULL in database

## Web Application

### Search Interface

I built a Flask web app where users can filter complaints by:

- Date range (from and to dates)
- Borough (Brooklyn, Manhattan, Queens, Bronx, Staten Island)
- Complaint type (keyword search)

## NYC 311 Service Requests

[Search](#) [Aggregate Stats](#)

Date From:  
2025-01-01

Date To:  
2025-01-02

Borough:  
BRONX

Complaint Type:  
Search complaint type...

Search

Total Records Found: 95 | Page: 1 of 2

Unique Key	Created Date	Closed Date	Agency	Complaint Type	Descriptor	Borough
63603317	2025-01-02 00:00:00	2025-01-02 07:50:00	DSNY	Derelict Vehicles	Derelict Vehicles	BRONX
63592142	2025-01-01 23:59:57	2025-01-02 17:03:02	NYPD	Noise - Residential	Loud Music/Party	BRONX
63585908	2025-01-01 23:59:43	2025-01-02 17:12:47	NYPD	Noise - Residential	Loud Music/Party	BRONX
63593953	2025-01-01 23:59:32	2025-01-03 18:29:17	HPD	HEAT/HOT WATER	ENTIRE BUILDING	BRONX
63594319	2025-01-01 23:57:26	2025-01-02 03:50:39	NYPD	Illegal Parking	Blocked Crosswalk	BRONX
63586983	2025-01-01 23:55:08	2025-01-01 23:55:08	DOHMH	Rodent	Mouse Sighting	BRONX

Figure 6: Main search interface with filters for date, borough, and complaint type

63589794	2025-01-01 23:14:36	2025-01-02 01:20:57	NYPD	Noise - Commercial	Loud Music/Party	BRONX
63592684	2025-01-01 23:12:12	2025-01-02 20:27:35	HPD	HEAT/HOT WATER	APARTMENT ONLY	BRONX
63590729	2025-01-01 23:11:50	2025-01-02 03:18:50	NYPD	Noise - Residential	Loud Music/Party	BRONX
63595004	2025-01-01 23:11:03	2025-01-01 23:12:51	NYPD	Noise - Residential	Loud Music/Party	BRONX
63585826	2025-01-01 23:10:50	2025-01-02 02:47:49	NYPD	Noise - Residential	Loud Music/Party	BRONX
63586361	2025-01-01 23:09:43	2025-01-01 23:18:35	NYPD	Noise - Residential	Loud Music/Party	BRONX

Page 1 of 2Next

Figure 7: Search results showing filtered complaints with pagination

## Statistics Dashboard

The aggregate page shows:

- Total complaints (open and closed)
- Breakdown by borough
- Top 10 complaint types
- Closure rates with progress bars

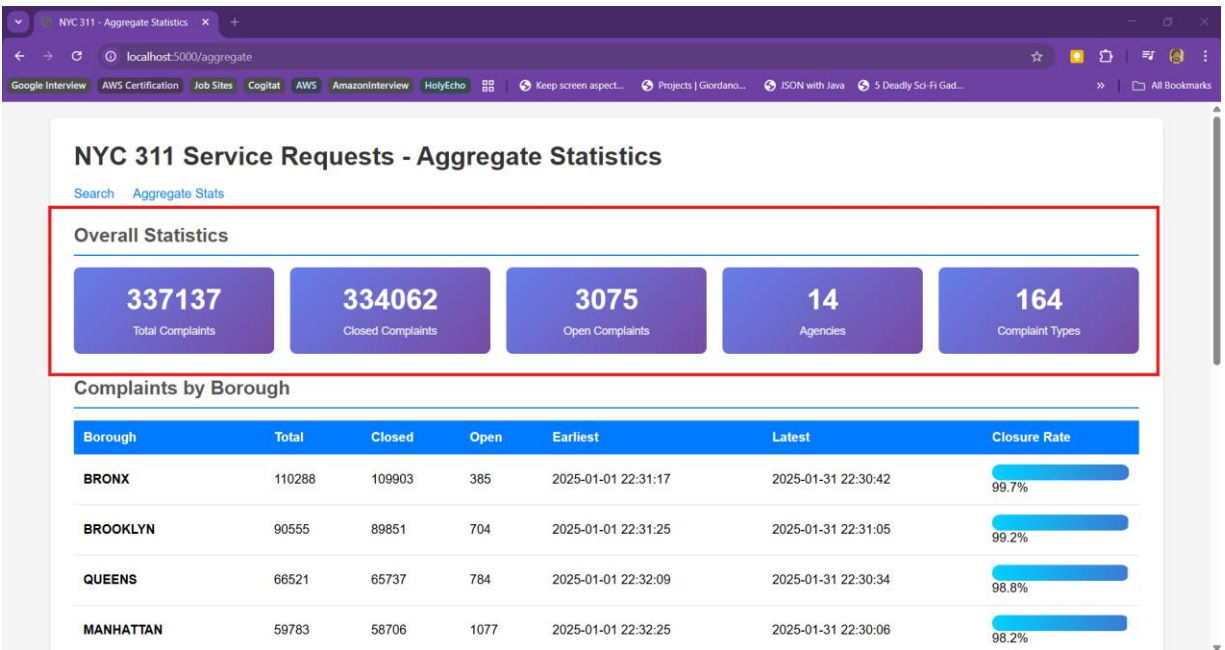


Figure 8: Statistics dashboard showing complaints breakdown

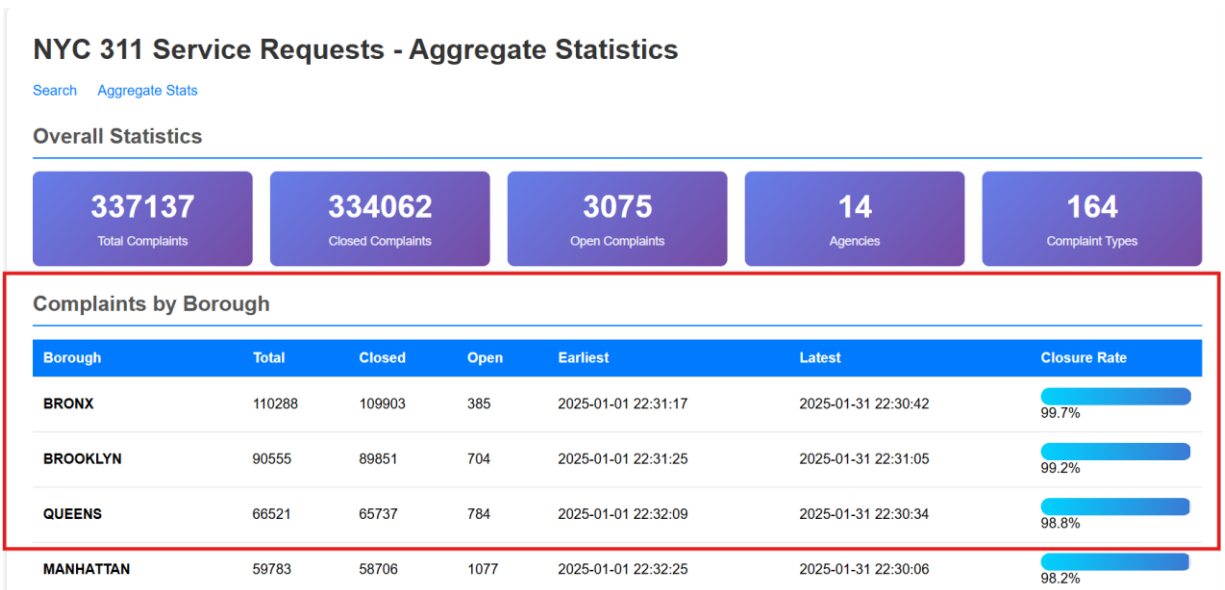


Figure 9: Complaints per borough with closure rates

# Automated Testing

## Selenium Tests

I created 10 browser tests that run automatically

Used browser engine and used pytest to write and run tests

```
2 files changed, 47 deletions(-)
PS D:\Conestoga\Course2-DevOps\Sem2\Database Automation\DatabaseAutomation-GitRepo> pytest tests\selenium_test.py
===== test session starts =====
platform win32 -- Python 3.13.1, pytest-8.4.0, pluggy-1.6.0
rootdir: D:\Conestoga\Course2-DevOps\Sem2\Database Automation\DatabaseAutomation-GitRepo
plugins: anyio-4.8.0, cov-6.2.1
collected 10 items

tests\selenium_test.py
DevTools listening on ws://127.0.0.1:51800/devtools/browser/a6a0f384-4ceb-4e13-b9a5-faaa5ea5e1bb
..... [100%]

===== 10 passed in 14.60s =====
PS D:\Conestoga\Course2-DevOps\Sem2\Database Automation\DatabaseAutomation-GitRepo> git push origin master
```

Figure 10: Selenium tests running in terminal

## CI/CD Pipeline

### GitHub Actions

Every time I push code to GitHub, the pipeline automatically:

1. Sets up a MySQL database
2. Loads the schema and indexes
3. Runs ETL on test data (20 sample records)
4. Starts the Flask application
5. Runs all Selenium tests in headless Chrome



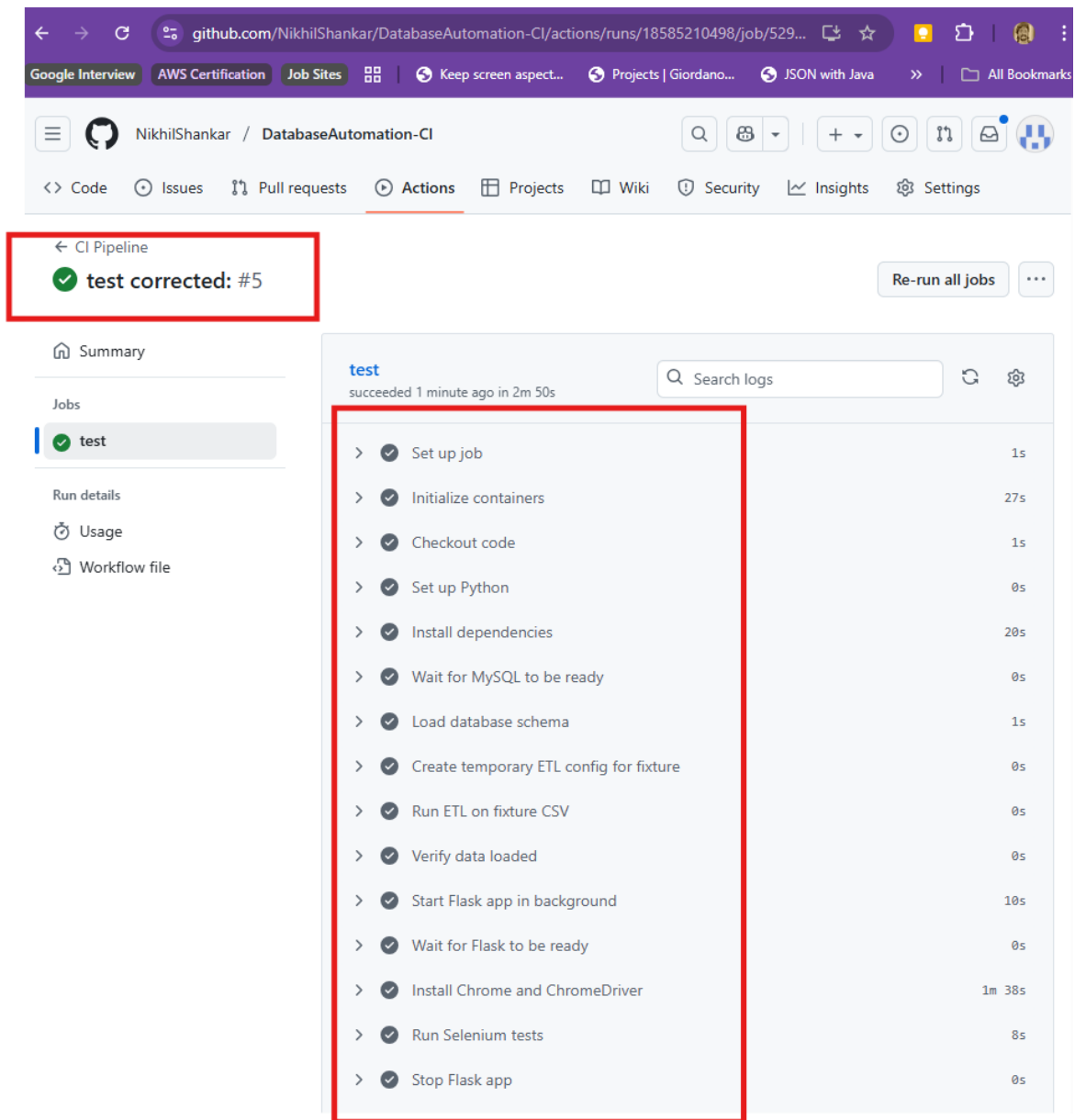


Figure 11: GitHub Actions CI/CD pipeline workflow

## Scalability Considerations

### Current System:

- Handles ~3,37,000 records easily
- ETL processes ~1,500 rows/second
- Web searches return results in under 100ms

### For Larger Datasets (1M+ records):

- Keep chunked processing (prevents memory issues) which is already handled in the ETL script
- Add more indexes for frequently searched fields. For current app though all filters are under indexes.
- Consider partitioning table by date
- Use caching for aggregate statistics
- Scale horizontally with read replicas

## Challenges and Solutions

**NaN Values Breaking MySQL:** Pandas NaN values caused insert errors

Solution: Created `safe_value()` function to convert NaN to Null

**Large CSV File Size:** Loading entire file into memory caused crashes

Solution: Used chunked reading with `pd.read_csv(chunksize=10000)`

**Slow Queries:** Searches took 2+ seconds with 200K records

Solution: Added indexes on commonly filtered columns

**CI/CD Test Failures:** Tests failed with small fixture dataset

Solution: Expanded fixture from 5 to 20 records covering all test cases

## Appendices

### A. Technologies Used

- **Database:** MySQL 8.0
- **Backend:** Python 3.11, Flask, Pandas
- **Testing:** Pytest, Selenium WebDriver
- **DevOps:** Docker, Docker Compose, GitHub Actions
- **Frontend:** HTML5, CSS3, Jinja2

## B. Repository Structure

- GitHub: [Repository URL]
- All code is documented and follows best practices
- README includes setup instructions
- Tests have 100% pass rate

**End of Report**