Database Automation

Assignment 3

Project Report

Submitted by

Nikhil Shankar C S

9026254

Table of Contents

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      |         |
         | YES     | NULL       |
| service_requests |         1 | idx_borough      |           1 | borough      | A         |          5 |     NULL |   NULL | YES  | BTREE      |         |
         | YES     | NULL       |
| service_requests |         1 | idx_agency       |           1 | agency       | A         |         13 |     NULL |   NULL | YES  | BTREE      |         |
         | YES     | NULL       |
| service_requests |         1 | idx_date_borough |           1 | created_date | A         |     267301 |     NULL |   NULL |      | BTREE      |         |
         | YES     | NULL       |
```

Figure 2: Indexes defined in the service_requests table

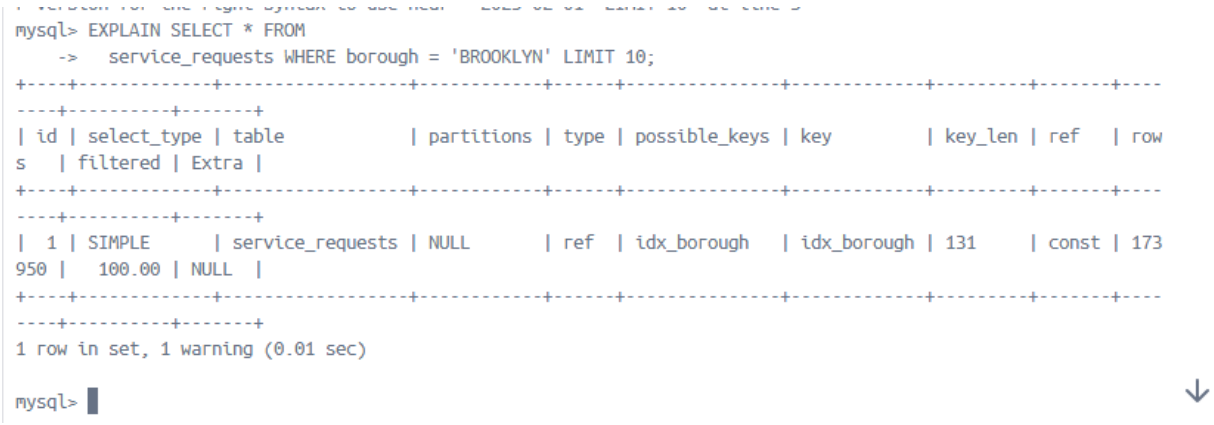

Figure 3: Confirming indexes is working using EXPLAIN command

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

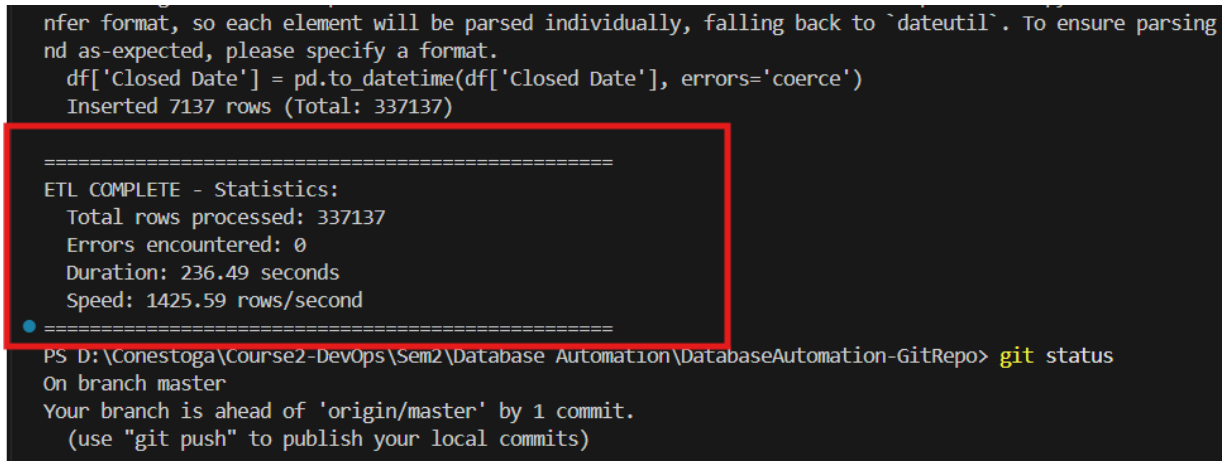

Figure 4: ETL statistics after loading Jan 2025

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)

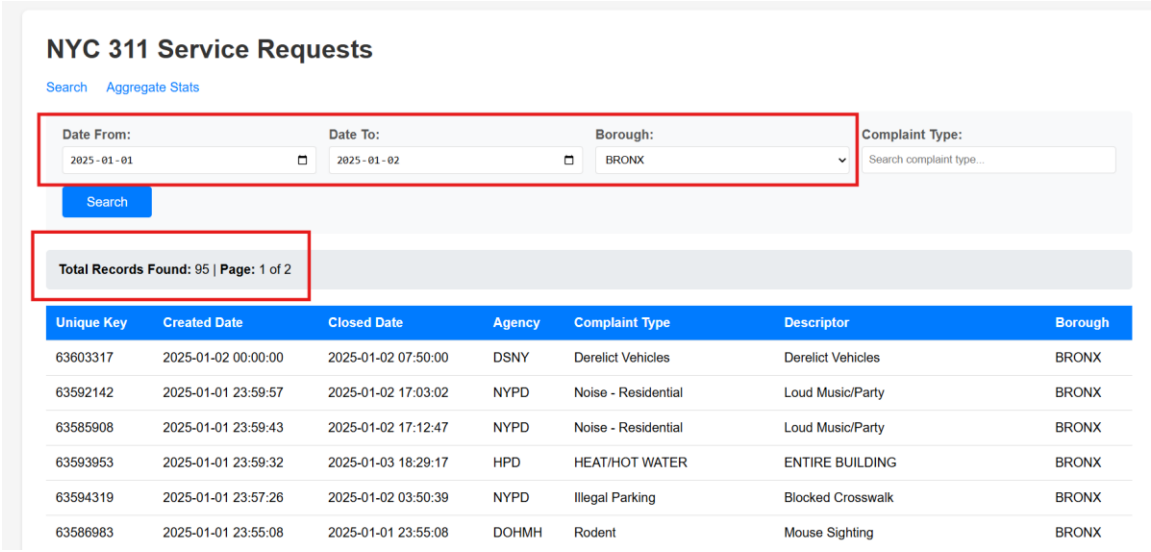

Figure 6: Main search interface with filters for date, borough, and complaint type

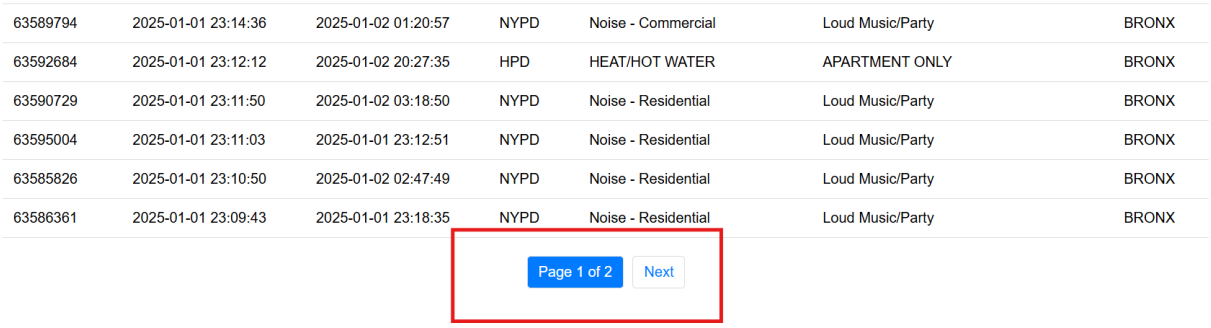

Figure 7: Search results showing filtered complaints with pagination

Statistics Dashboard

The aggregate page shows:

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

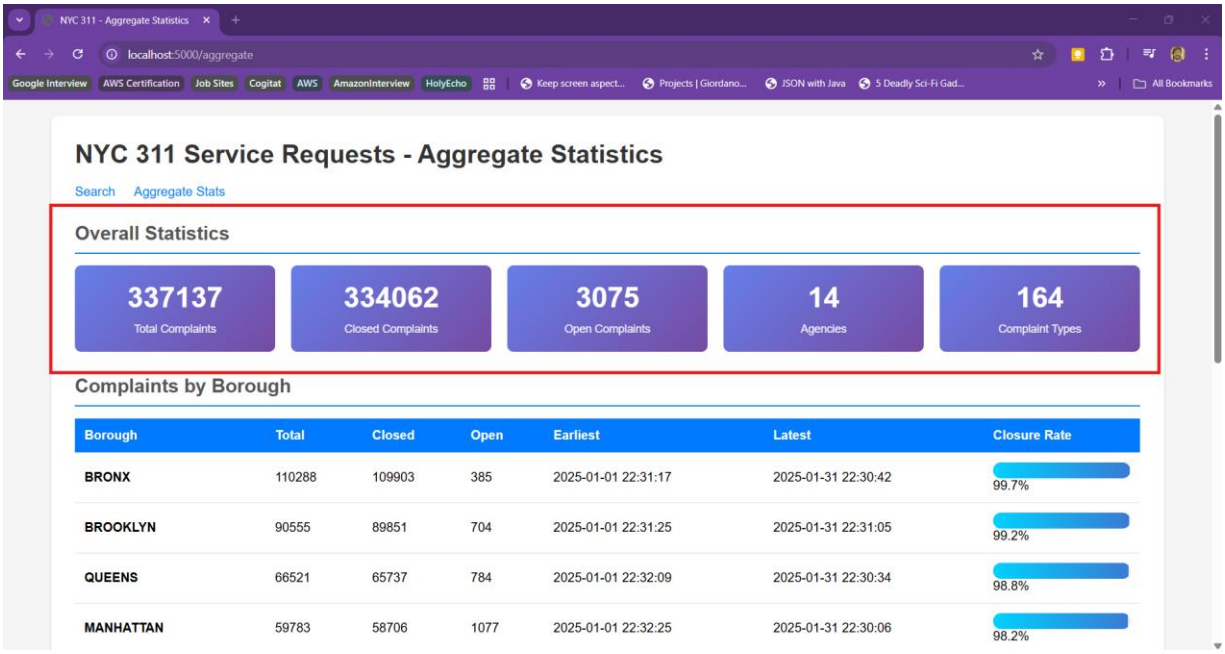

Figure 8: Statistics dashboard showing complaints breakdown

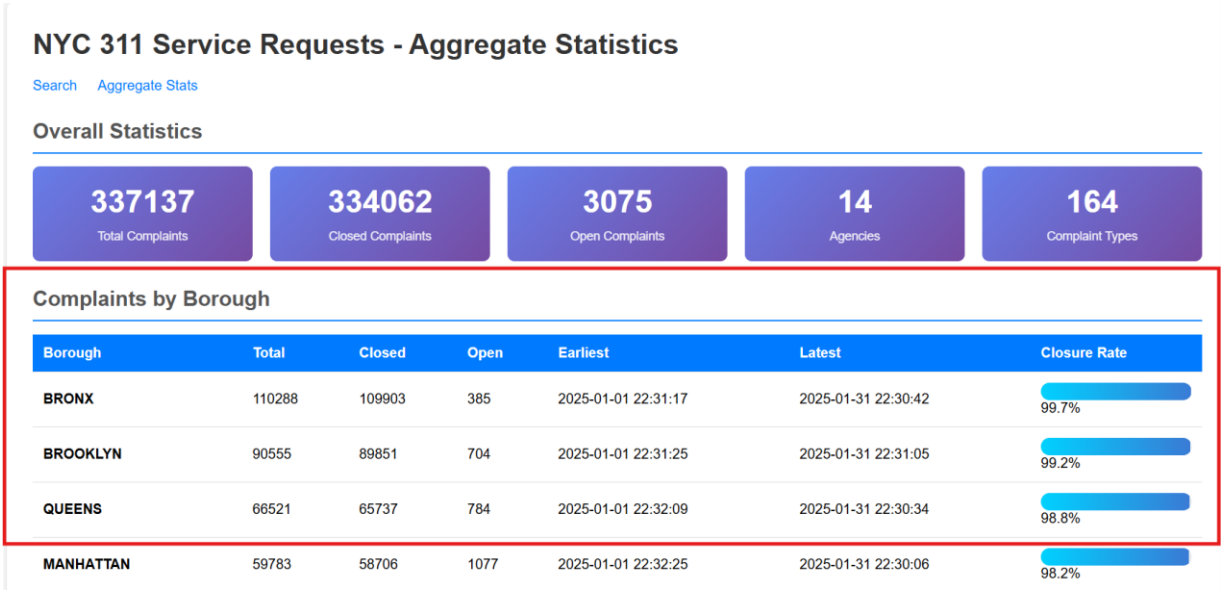

Figure 9: Complaints per borough with closure rates

Automated Testing

Selenium Tests

I created 10 browser tests that run automatically

Used browser engine and used pytest to write and run tests

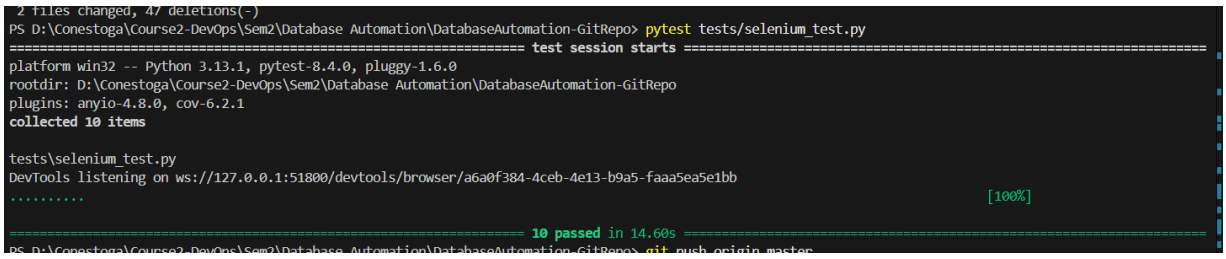

Figure 10: Selenium tests running in terminal

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

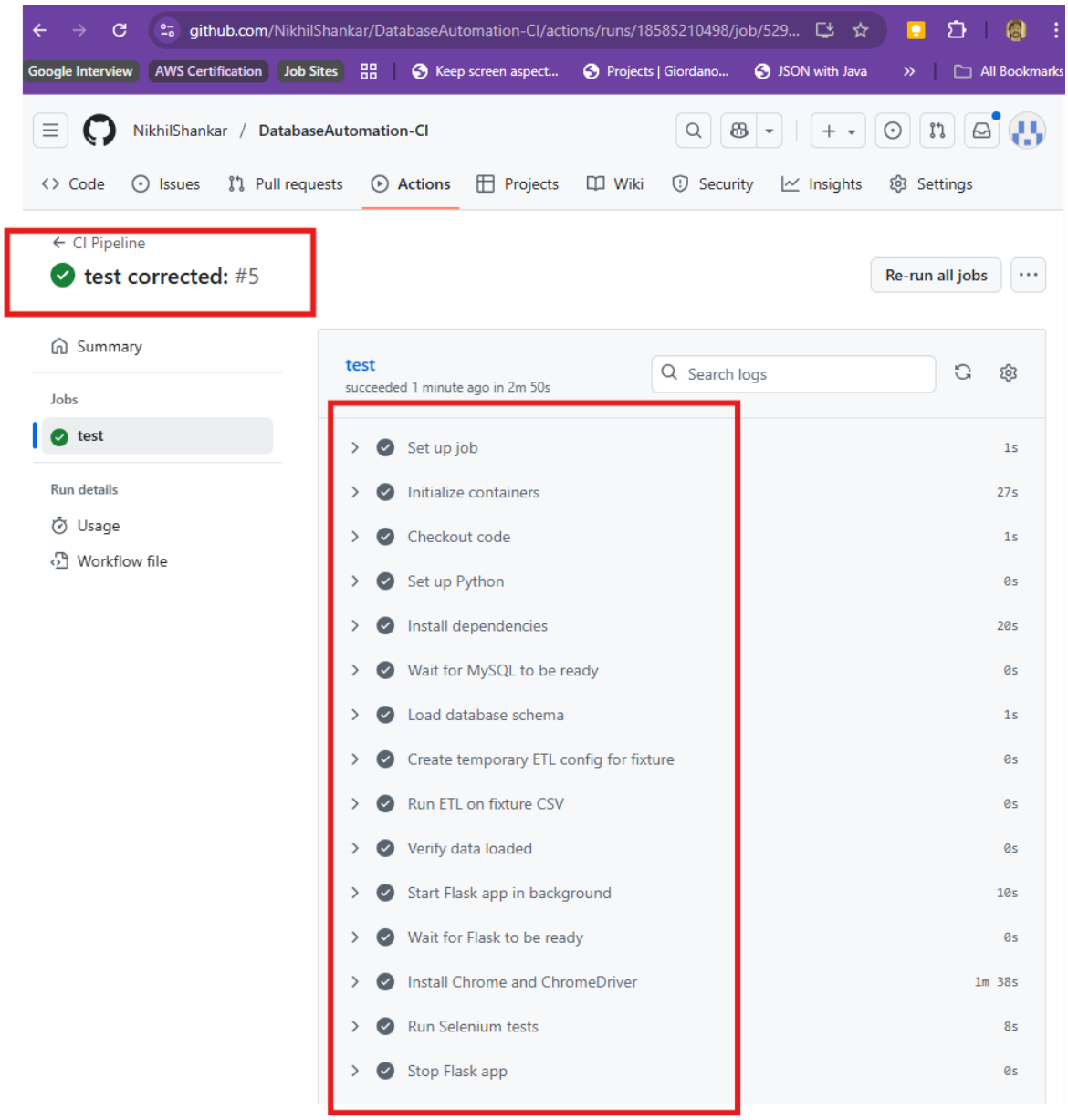

Figure 11: GitHub Actions CI/CD pipeline workflow

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