Rohan Bankala
Neil Bhutada

ADT Project – Part 2

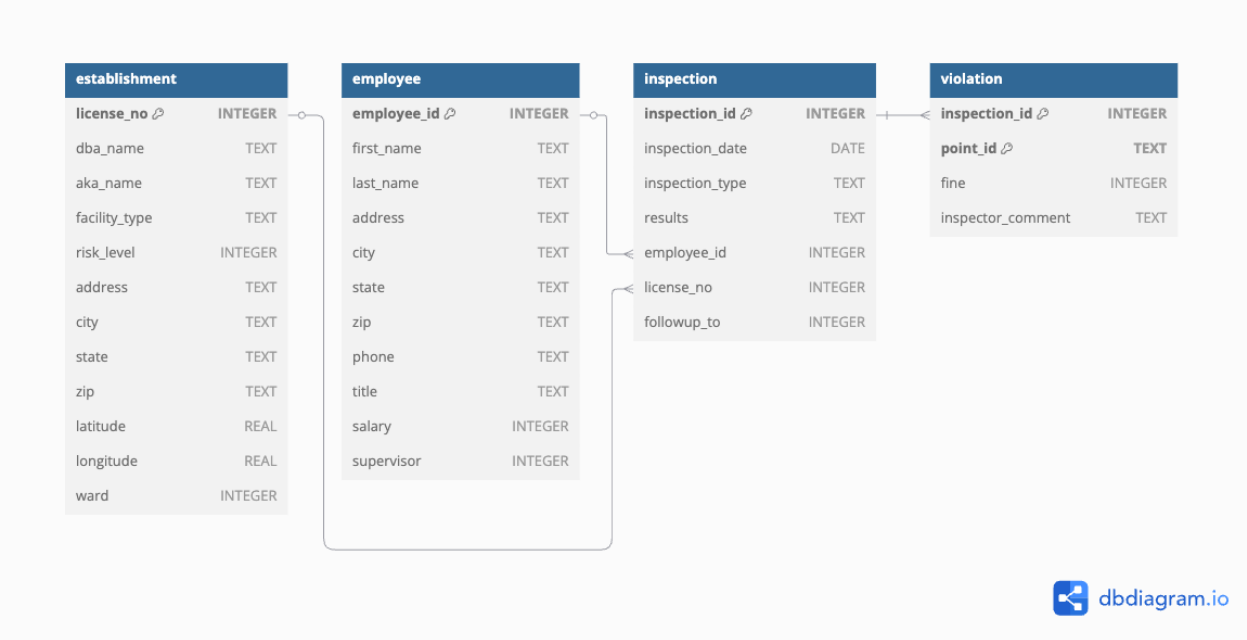
Purpose & Motivation

The purpose of this project is to design a comprehensive restaurant inspection management system that enables regulatory bodies to efficiently monitor public health compliance across food establishments. Health inspection data is often collected in unorganized systems that lack relational structure, limiting analysis of patterns in violations, inspector performance, and establishment risk levels. This project integrates a SQLite database with structured CSV datasets to provide a foundation for querying, maintaining, and analyzing inspection records. Through the addition of CRUD functionality and data visualizations, the system supports meaningful insights into food safety trends and enhances data-driven decision-making for inspection planning and public health strategy.

Data Source

The project uses a publicly available restaurant inspection dataset, composed of multiple interconnected CSV files including **establishment, employee, inspection, and violation** data. These files collectively contain critical information such as establishment demographics, inspector assignments, inspection outcomes, and violation details. The structured nature of this dataset supports relational modeling and is ideal for exploring patterns in food safety compliance, inspector workloads, and risk-level distributions across geographic and organizational segments.

Conceptual Diagram/Schema for database



Rohan Bankala
Neil Bhutada

ADT Project – Part 2

Database Constraints

Establishment Table

Constraints:

- **Primary Key Constraint:**
  - license_no is defined as an INTEGER PRIMARY KEY. This ensures uniqueness for each establishment and enforces non-nullability.
- **Nullability:**
  - All other columns such as dba_name, address, city, zip, etc. are not explicitly defined as NOT NULL, meaning they can accept null values unless restricted by the application logic.

Employee Table

Constraints:

- **Primary Key Constraint:**
  - employee_id is declared as an INTEGER PRIMARY KEY, automatically enforcing uniqueness and disallowing null values.
- **Nullability:**
  - All other fields (e.g., supervisor, salary, title, etc.) can accept NULL values since there is no explicit NOT NULL constraint.

Inspection Table

Constraints:

- **Primary Key Constraint:**
  - inspection_id is defined as the INTEGER PRIMARY KEY, ensuring each inspection record is unique and non-nullable.
- **Foreign Key Constraints:**
  - employee_id references employee(employee_id) — ensures inspections are conducted by valid employees.

Rohan Bankala
Neil Bhutada

ADT Project – Part 2

- license_no references establishment(license_no) — enforces that each inspection is linked to a registered establishment.
- **Nullability:**
  - Fields like inspection_type, results, and followup_to are nullable unless additional constraints are imposed by the application.

Violation Table

Constraints:

- **Composite Primary Key Constraint:**
  - A **composite key** is defined on (inspection_id, point_id) — ensures that the same violation cannot be entered twice for the same inspection.
- **Foreign Key Constraint:**
  - inspection_id references inspection(inspection_id) — enforces that each violation is tied to a valid inspection.
- **Nullability:**
  - Fields like fine and inspector_comment are not declared as NOT NULL, meaning they can contain null values unless restricted in logic.

Summary of Applied Constraints:

1. **Primary Keys:**
   - establishment.license_no (INTEGER, not null)
   - employee.employee_id (INTEGER, not null)
   - inspection.inspection_id (INTEGER, not null)
   - violation.(inspection_id, point_id) (Composite key)
2. **Foreign Keys:**
   - inspection.employee_id → employee.employee_id
   - inspection.license_no → establishment.license_no
   - violation.inspection_id → inspection.inspection_id
3. **Data Types & Implicit Constraints:**
   - All columns use defined data types (e.g., TEXT, INTEGER, REAL, DATE)
   - No explicit formatting or check constraints were added beyond PK/FK enforcement.
4. **Nullability:**
   - Only primary key columns are implicitly non-null.

Rohan Bankala
Neil Bhutada

ADT Project – Part 2

- All other columns can accept null values unless explicitly restricted in queries or application logic.

```
-- Neil Bhutada

-- Table: establishment
CREATE TABLE establishment (
    license_no INTEGER PRIMARY KEY,
    dba_name TEXT,
    aka_name TEXT,
    facility_type TEXT,
    risk_level INTEGER,
    address TEXT,
    city TEXT,
    state TEXT,
    zip TEXT,
    latitude REAL,
    longitude REAL,
    ward INTEGER
);

-- Table: employee
CREATE TABLE employee (
    employee_id INTEGER PRIMARY KEY,
    first_name TEXT,
    last_name TEXT,
    address TEXT,
    city TEXT,
    state TEXT,
    zip TEXT,
    phone TEXT,
    title TEXT,
    salary INTEGER,
    supervisor INTEGER
);

```

Rohan Bankala
Neil Bhutada

ADT Project – Part 2

```
38.  -- Table: inspection
39.  CREATE TABLE inspection (
40.      inspection_id INTEGER PRIMARY KEY,
41.      inspection_date DATE,
42.      inspection_type TEXT,
43.      results TEXT,
44.      employee_id INTEGER,
45.      license_no INTEGER,
46.      followup_to INTEGER,
47.      FOREIGN KEY (employee_id) REFERENCES employee(employee_id),
48.      FOREIGN KEY (license_no) REFERENCES establishment(license_no)
49.  );
50.
51.  -- Table: inspection_point
52.  CREATE TABLE inspection_point (
53.      point_id TEXT PRIMARY KEY,
54.      Description TEXT,
55.      category TEXT,
56.      code TEXT,
57.      fine INTEGER,
58.      point_level TEXT
59.  );
60.
61.  -- Table: violation
62.  CREATE TABLE violation (
63.      inspection_id INTEGER,
64.      point_id TEXT,
65.      fine INTEGER,
66.      inspector_comment TEXT,
67.      PRIMARY KEY (inspection_id, point_id),
68.      FOREIGN KEY (inspection_id) REFERENCES inspection(inspection_id)
69.  );
70.
```

Rohan Bankala
Neil Bhutada

ADT Project – Part 2

Code to create database and build queries

Attached the code at the end of the doc

Overall Contribution Summary

Name	Task	Contribution	AVR Time spent (hrs)
Rohan Bankala	Setting up database Data cleaning and ingestion Data visualizations & CRUD operations	- Set up the SQLite database structure using Jupyter Notebook - Loaded and preprocessed data from multiple CSV files - Developed CRUD operations and created multiple Matplotlib-based visualizations to explore violation trends, inspector activity, and establishment risk levels	6
Neil Bhutada	Conceptional schema Documentation Table creation	- Designed the entity-relationship diagram (ERD) and conceptual schema based on project needs - Wrote SQL table creation statements with	5

Rohan Bankala
Neil Bhutada

ADT Project – Part 2

		appropriate primary/foreign key constraints - Documented schema structure, constraints, and database design decisions	

Restaurant Inspection - ADT Project

Rohan Bankala

Neil Bhutada

In [ ]:
```
#Rohan Bankala

import pandas as pd
import sqlite3
import matplotlib.pyplot as plt

# Connects to SQLite database
conn = sqlite3.connect("restaurant_inspections.db")
cursor = conn.cursor()
```

In [ ]:
```
#Rohan Bankala

# Loading the CSV files
est_df = pd.read_csv("establishment.csv")
emp_df = pd.read_csv("employee.csv")
insp_df = pd.read_csv("inspection.csv")
viol_df = pd.read_csv("violation.csv")
#point_df = pd.read_csv("inspection_point.csv")
```

In [ ]:
```
#Neil Bhutada

with open("restaurant_inspection_schema.sql", "r") as f:
    schema_sql = f.read()
cursor.executescript(schema_sql)
print("Tables created successfully.")
```

```
Tables created successfully.
```

In [ ]:
```
#Rohan Bankala

# Data Cleaning
insp_df['inspection_date'] = pd.to_datetime(insp_df['inspection_date'], erro
insp_df.dropna(subset=['inspection_id'], inplace=True)

# Inserting data into tables
est_df.to_sql("establishment", conn, if_exists='append', index=False)
emp_df.to_sql("employee", conn, if_exists='append', index=False)
insp_df.to_sql("inspection", conn, if_exists='append', index=False)
#point_df.to_sql("inspection_point", conn, if_exists='append', index=False)
viol_df.to_sql("violation", conn, if_exists='append', index=False)
print("Data inserted into all tables.")
```

```
/var/folders/zf/340jxlt11lz6z4z1gn7gttxm0000gn/T/ipykernel_91601/2065233394.
py:2: UserWarning: Could not infer format, so each element will be parsed in
dividually, falling back to `dateutil`. To ensure parsing is consistent and
as-expected, please specify a format.
  insp_df['inspection_date'] = pd.to_datetime(insp_df['inspection_date'], er
rors='coerce').dt.date
Data inserted into all tables.
```

In [ ]:
```
#Neil Bhutada

# Checking table schema and constraints
cursor.execute("PRAGMA table_info('inspection');")
print("Inspection table schema:")
for row in cursor.fetchall():
    print(row)

cursor.execute("PRAGMA foreign_key_list('violation');")
print("\nForeign keys in violation table:")
for row in cursor.fetchall():
    print(row)
```

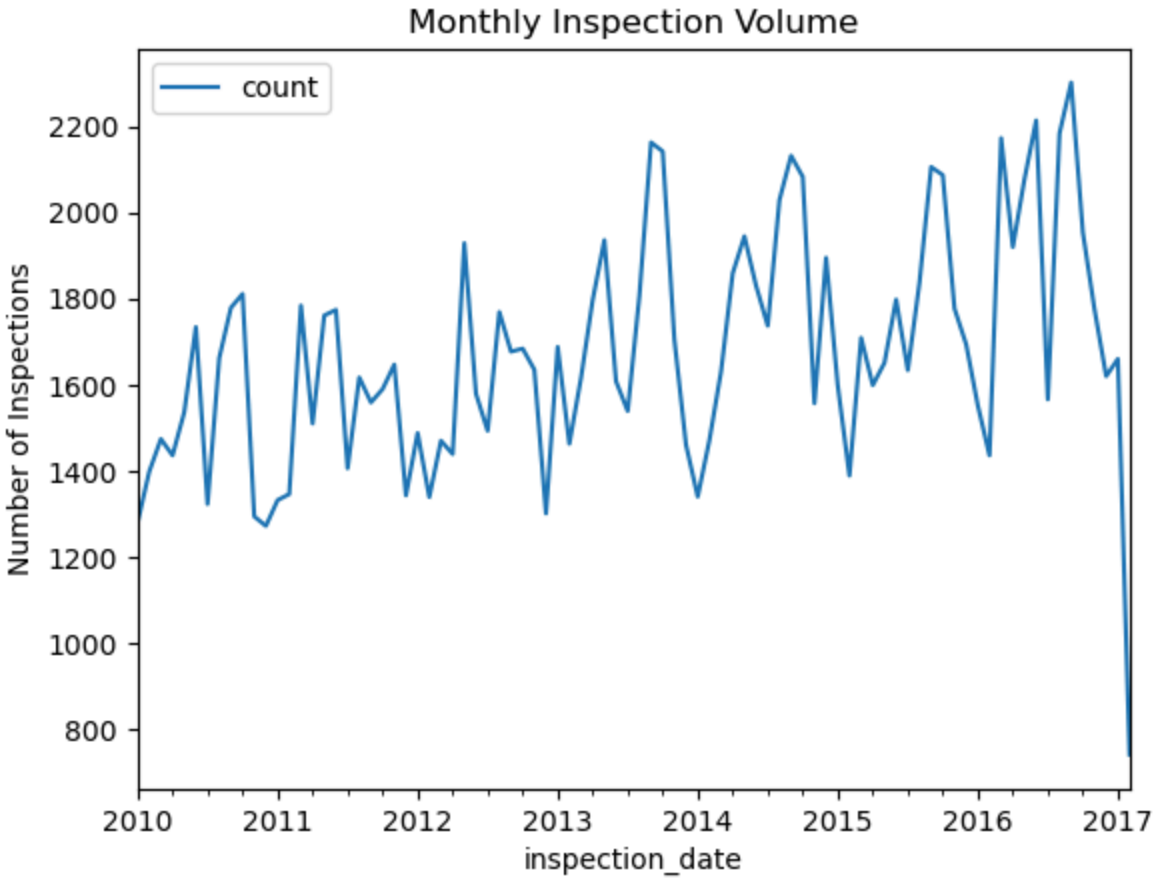
```
Inspection table schema:
(0, 'inspection_id', 'INTEGER', 0, None, 1)
(1, 'inspection_date', 'DATE', 0, None, 0)
(2, 'inspection_type', 'TEXT', 0, None, 0)
(3, 'results', 'TEXT', 0, None, 0)
(4, 'employee_id', 'INTEGER', 0, None, 0)
(5, 'license_no', 'INTEGER', 0, None, 0)
(6, 'followup_to', 'INTEGER', 0, None, 0)

Foreign keys in violation table:
(0, 0, 'inspection', 'inspection_id', 'inspection_id', 'NO ACTION', 'NO ACTI
ON', 'NONE')
```

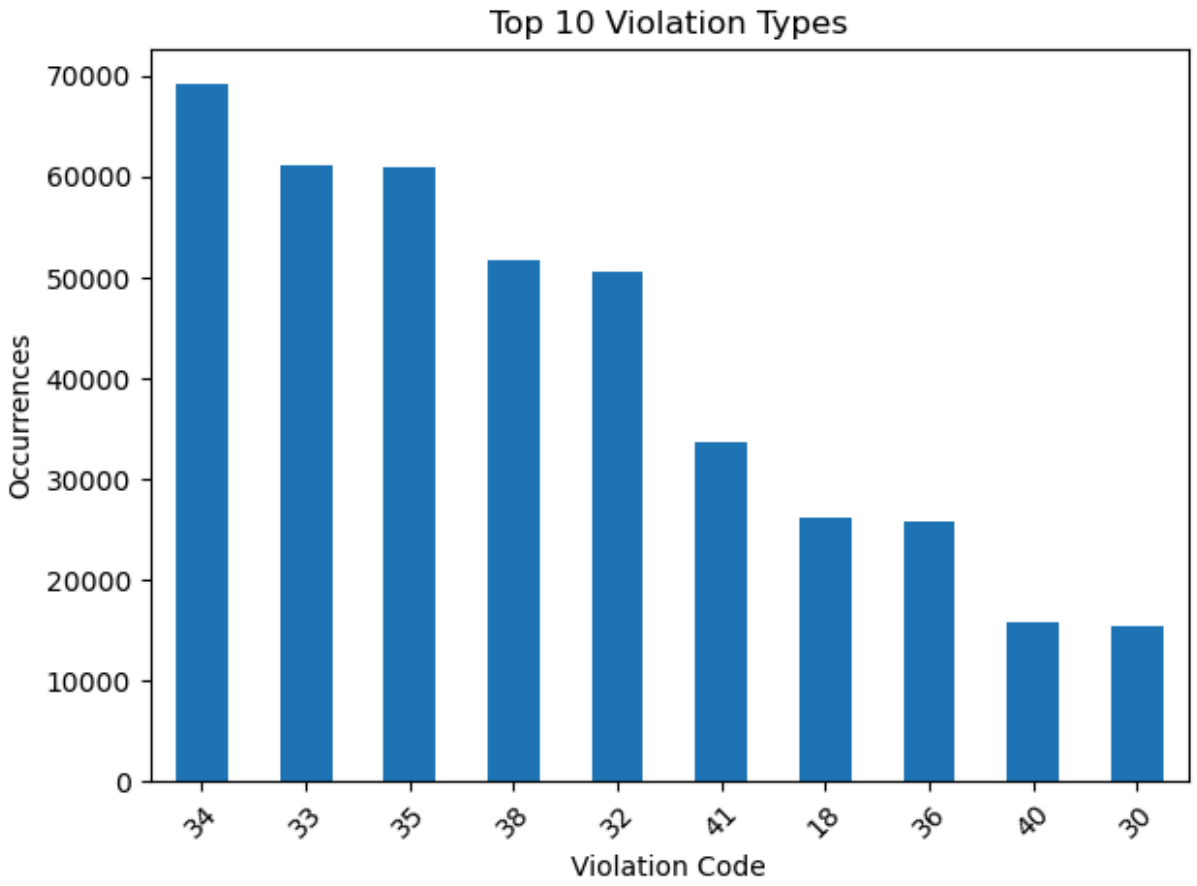
Visulaizations - Rohan Bankala

In [ ]:
```
# Inspections Over Time
query = "SELECT inspection_date, COUNT(*) as count FROM inspection GROUP BY
df = pd.read_sql_query(query, conn)
df['inspection_date'] = pd.to_datetime(df['inspection_date'])
df.set_index('inspection_date').resample('M').sum().plot(title="Monthly Insp
plt.ylabel("Number of Inspections")
plt.show()
```

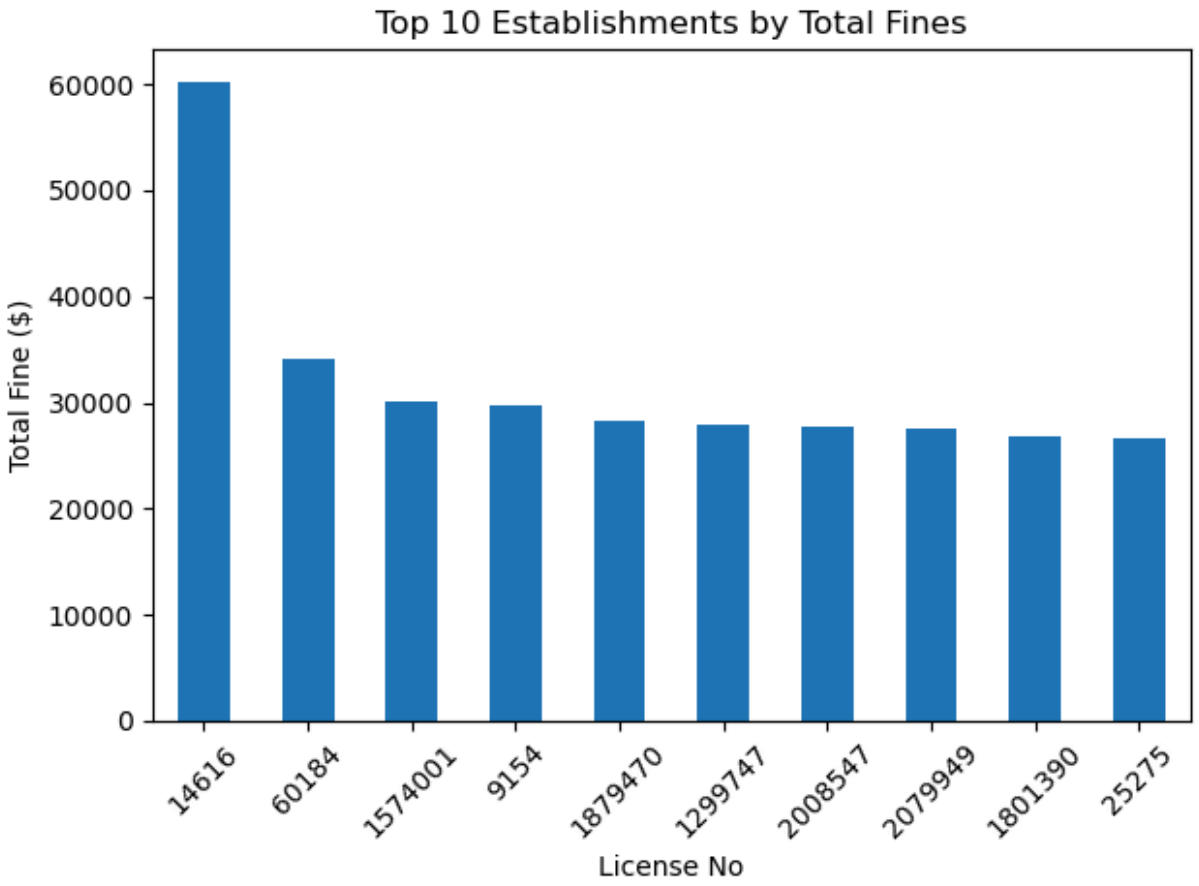
```
/var/folders/zf/340jxlt11lz6z4z1gn7gttxm0000gn/T/ipykernel_91601/1500939027.
py:5: FutureWarning: 'M' is deprecated and will be removed in a future versi
on, please use 'ME' instead.
  df.set_index('inspection_date').resample('M').sum().plot(title="Monthly In
spection Volume")
```



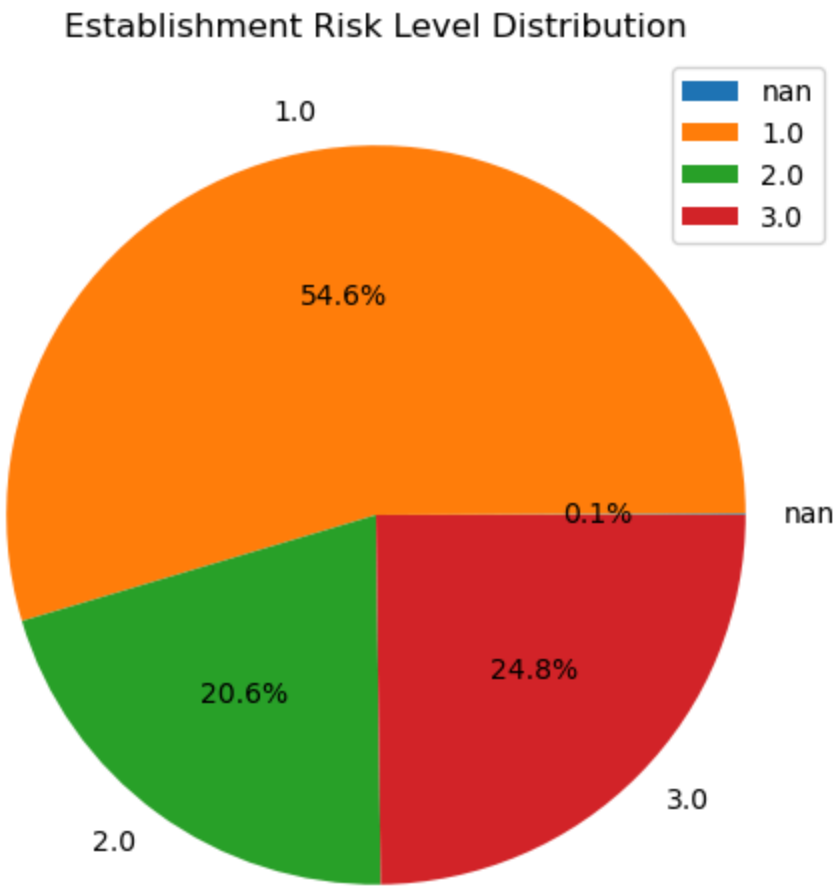
In [ ]:
```
#Top 10 violation types
query = """
SELECT point_id, COUNT(*) as count
FROM violation
GROUP BY point_id
ORDER BY count DESC
LIMIT 10
"""
df = pd.read_sql_query(query, conn)
df.plot.bar(x='point_id', y='count', title='Top 10 Violation Types', legend=
plt.xlabel("Violation Code")
plt.ylabel("Occurrences")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



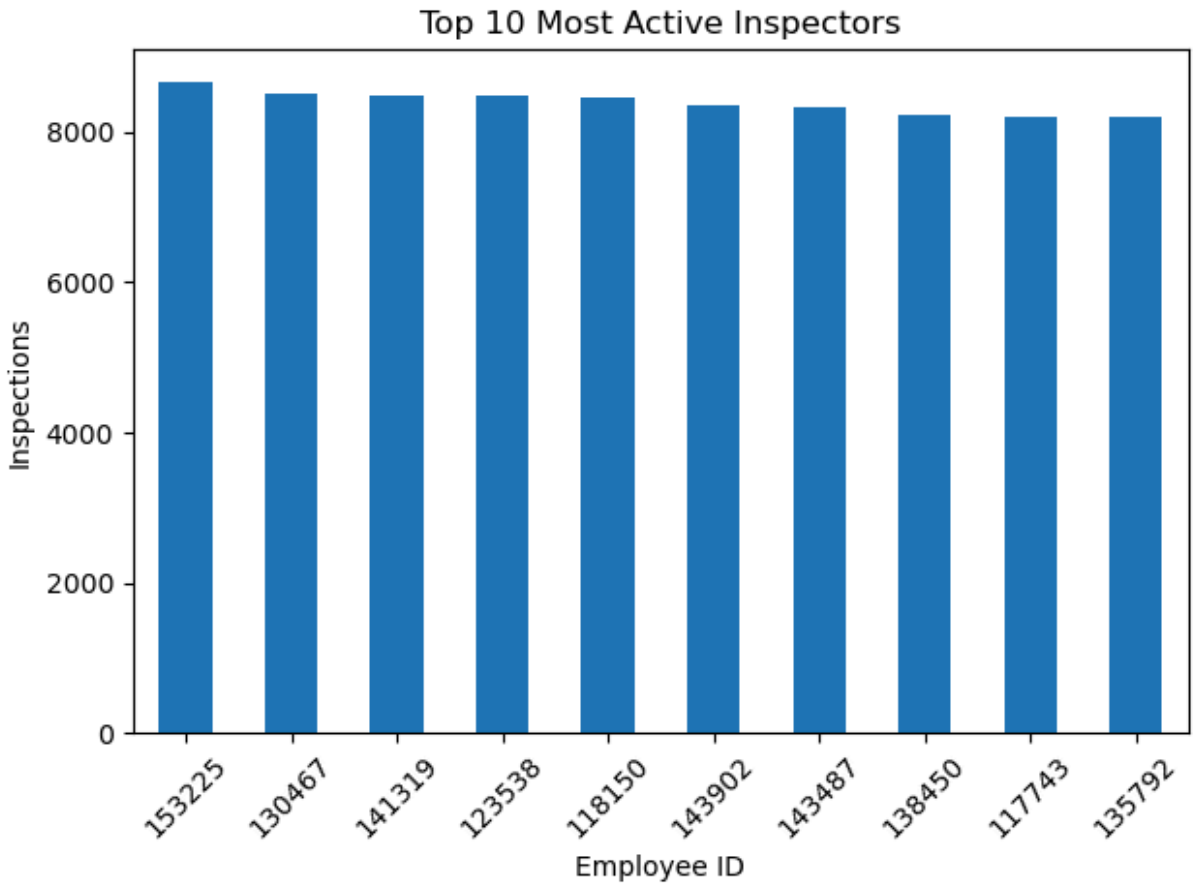
In [ ]:
```
#Top 10 Establishments by total fines
query = """
SELECT license_no, SUM(fine) as total_fine
FROM violation
JOIN inspection USING (inspection_id)
GROUP BY license_no
ORDER BY total_fine DESC
LIMIT 10
"""
df = pd.read_sql_query(query, conn)
df.plot.bar(x='license_no', y='total_fine', title='Top 10 Establishments by
plt.xlabel("License No")
plt.ylabel("Total Fine ($)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [ ]:
```
#Establishment risk level distributions
query = """
SELECT risk_level, COUNT(*) as count
FROM establishment
GROUP BY risk_level
"""
df = pd.read_sql_query(query, conn)
df.set_index('risk_level').plot.pie(y='count', autopct='%1.1f%%', title="Est
plt.ylabel('')
plt.show()
```



In [ ]:
```
#Top 10 most active health inspectors
query = """
SELECT employee_id, COUNT(*) as inspections
FROM inspection
GROUP BY employee_id
ORDER BY inspections DESC
LIMIT 10
"""
df = pd.read_sql_query(query, conn)
df.plot.bar(x='employee_id', y='inspections', title='Top 10 Most Active Insp
plt.xlabel("Employee ID")
plt.ylabel("Inspections")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```


Top 10 Most Active Inspectors
Inspections
0
2000
4000
6000
8000
153225
130467
141319
123538
118150
143902
143487
138450
117743
135792
Employee ID