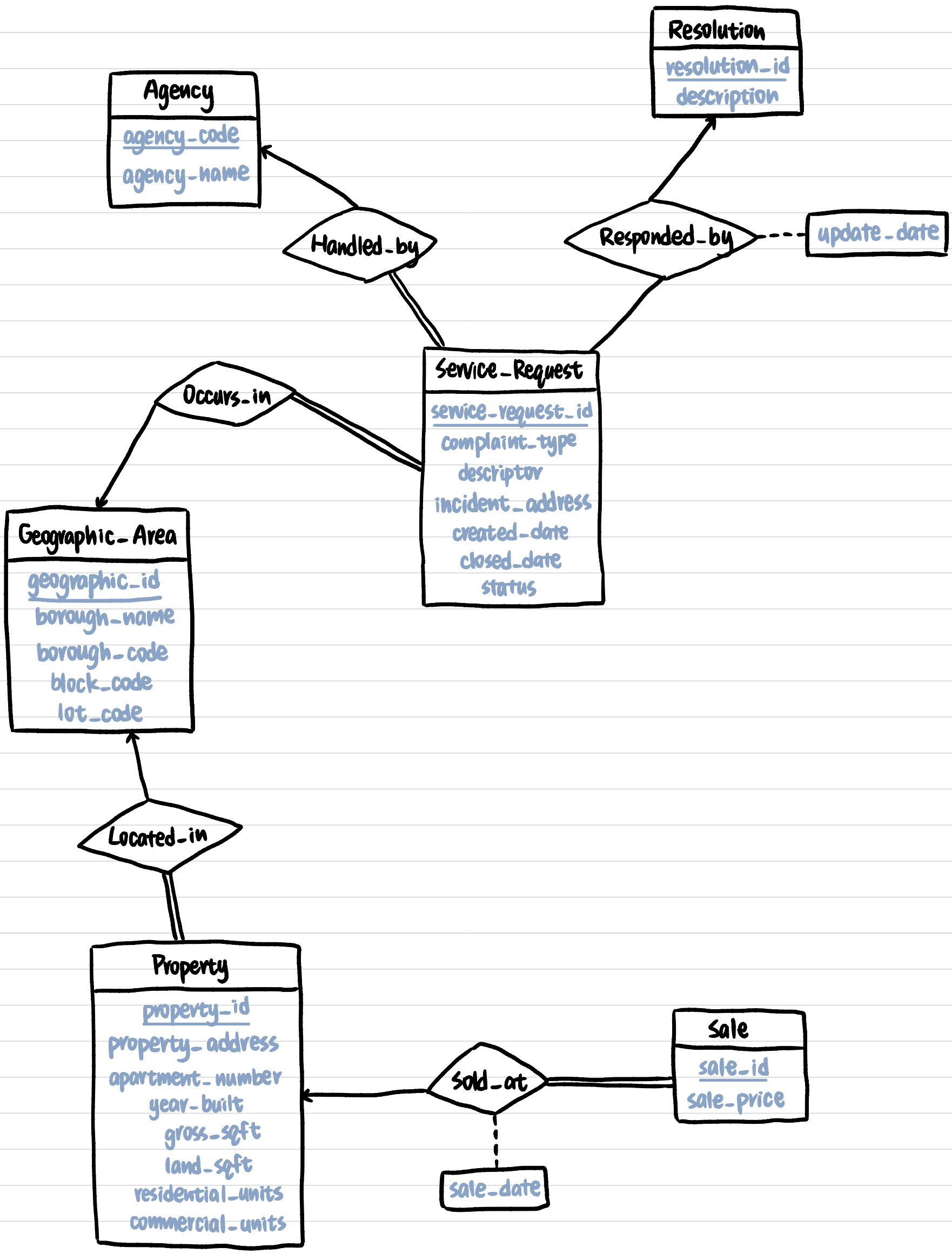
Application
NYC 311 Service Requests & Property Sale Price
311 Service Requests from 2010 to Present | NYC Open Data
Rolling Sales Data

Description
We'll analyze how 311-reported incidents (noise, sanitation, etc.) relate to property sale prices at the neighborhood level. The intuition is that neighborhoods with more nuisances or unresolved service requests are less desirable, hence having lower prices on property. Directly combining 311 service requests with property sales to study their relationship and summarize patterns has not been done in a straightforward way before, which makes our project interesting. The challenge is that incidents are not always tied to the exact property that appears in the sales data. However, we resolve this issue by matching both datasets to their common geographic unit, the Borough-Block-Lot (BBL), so that service requests and property sales can be analyzed consistently at the geographic level. Our database includes six entity sets: ***Service_Request*** incidents reported to 311 with details such as complaint type, dates, status, ***Agency*** represents the city department responsible for handling the requests, ***Resolution*** describes the action taken on service requests by the responsible agency, ***Geographic_Area*** captures location using borough, block, and lot codes, ***Property*** stores the details of the buildings sold, ***Sale*** records the property transactions. The five relationships are as follow: ***Handled_by*** (*Service_Request* → *Agency*; many-to-one; total on *Service_Request*: each request is handled by exactly one agency), ***Responded_by*** (*Service_Request* → *Resolution*; many-to-one: a request might not be linked to one resolution yet when created, but will be linked to at most one resolution by the time it is closed), ***Occurs_in*** (*Service_Request* → *Geographic_Area*; many-to-one; total on *Service_Request*: each request occurs in exactly one geographic area), ***Located_in*** (*Property* → *Geographic_Area*; many-to-one; total on *Property*: each property belongs to exactly one geographic area), ***Sold_at*** (*Sale* → *Property*; many-to-one, total on *Sale*: each sale refers to exactly one property). We will follow the Web Front-End Option.

Constraints not captured by E/R syntax include the following:

1. closed_date cannot precede created_date
2. (borough_code, block_code, lot_code) uniquely identifies a geographic_area
3. borough_name must be one of {Manhattan, Bronx, Brooklyn, Queens, Staten Island}
4. sale_price must be positive
5. If provided, gross_sqft and land_sqft are > 0; residential_units and commercial_units are ≥ 0
6. If provided, year_built should be after 1700 years and cannot be in the future



Notes on total / partial participation on Service_Request

They are the ones that depend on our design choice. The others are clear. We provide the reasoning:

(1) Service_Request (total) - Occurs_in → Geographic_Area: Every request must happen somewhere

(2) Service_Request (total) - Handled_by → Agency: Every request must be assigned when created

(3) Service_Request (partial) - Responded_by → Resolution: Not every request has resolution yet at creation time

Data Plan
The database will be populated using two data sources. The NYC Open Data 311 Service Requests dataset provides complaint-level records, which can be linked to geographic areas through Borough-Block-Lot identifiers. The NYC property sales dataset provides information on property characteristics and transaction details, and is also keyed by the same BBL identifiers. Using this common geographic unit allows us to integrate service requests and property sales at the geographic level for consistent analysis.

User Interaction Plan
We'll have a single search box that accepts either an address or a BBL. Addresses can be converted to BBL using a precomputed address → BBL lookup generated during ETL. Users can also select a time period to filter results. The results page will display an analytics dashboard summarizing service requests and property sales at the BBL level. Tables and visualizations will include requests by type, average property sale price, and more relevant metrics. Users can compare two areas side-by-side, bookmark areas, and export CSV for any filtered table. Each data summary also includes a "Trends" toggle to switch from a snapshot to a time-series view over the selected period.

Contingency Plan
The simplified design will include four entities: *Service_Request*, *Agency*, *Resolution*, *Geographic_Area*, and three relationships: *Handled_by*, *Responded_by*, *Occurs_in*. This database only relies on one data source, the NYC 311 Service Requests dataset. The front end retains the core functions: a single search box, a time filter, and an analytics dashboard with tables and visualizations**.** We will omit compare mode, bookmarks, and CSV export.

```
create table Agency (
    agency_code Integer Primary Key,
    agency_name Varchar(50) not null
);

create table Resolution (
    resolution_id Integer Primary Key,
    description Varchar(100)
);

create table Geographic_Area (
            geographic_id Integer Primary Key,
    borough_name Varchar(20) not null,
    borough_code Integer not null,
    block_code Integer not null,
    lot_code Integer not null,
    UNIQUE (borough_code, block_code, lot_code),
    Check (borough_name IN ('Manhattan','Bronx','Brooklyn','Queens','Staten Island'))
);

create table Property (
    property_id Integer Primary Key,
    geographic_id Integer not null,
    property_address Varchar(200) not null,
    apartment_number Integer,
    year_built Integer,
    gross_sqft numeric(10, 2),
    land_sqft numeric(10, 2),
    residential_units Integer,
    commercial_units Integer,
    Foreign Key (geographic_id) References Geographic_Area,
    Check (gross_sqft > 0),
    Check (land_sqft > 0),
    Check (residential_units >= 0),
    Check (commercial_units >= 0),
    Check (year_built between 1700 and extract(year from current_date))
);

create table Sale (
    sale_id Integer Primary Key,
    property_id Integer not null,
    sale_price numeric(12, 2) not null,
    sale_date Date not null,
    Foreign Key (property_id) References Property,
```

```
    Check (sale_price > 0)
);

create table Service_Request (
    service_request_id Integer Primary Key,
    geographic_id Integer not null,
    resolution_id Integer,
    agency_code Integer not null,
    complaint_type Varchar(10),
    descriptor Varchar(100),
    incident_address Varchar(200),
    created_date Date not null,
    closed_date Date,
    status Varchar(10),
    update_date Date,
    Foreign Key (agency_code) References Agency,
    Foreign Key (geographic_id) References Geographic_Area,
    Foreign Key (resolution_id) References Resolution,
    Check (closed_date >= created_date)
)
```