CSC-315: Database Systems

# Database System Report: Housing Affordability in New York City

Cole Ding, Jake Dingley, Justin Farnsworth, Anthony Kim, Quinn Wasko

## Abstract

The following application was designed to address the social issue of affordable housing through the implementation of a relational database. This article will go into detail about the objective, purpose, design and implementation of this application as well as the various models used to create the database. The reader should develop a greater understanding of the thought process behind this application along with its intended use.

## Introduction

For millions of people, living in the city for various career opportunities is virtually an imperative. However, due to a competitive market as well as gentrification, the ability to afford housing is slowly but surely becoming increasingly difficult. Particularly in New York City, people of low income struggle to keep their homes and are forced to move out of the city.

It is also costly and time-consuming to search for other homes in New York City as there are tens of thousands of properties for sale in the market. The inconvenience of looking at every house one by one in hopes that it fits one's preferences is a heavy burden. It would be much easier if the list of options was narrowed down to include only those that satisfy the user's specifications.

To address this issue, our team decided to gather data about New York City. Specifically, on housing, crime, graduation rates, and points of interest since they all play a role on the property values. The boroughs also play a significant factor as well. For example, living in Manhattan may be more expensive than living in the Bronx. It is vital that the team considered this when working towards a solution.

## Objective

The purpose of this project was to implement and operate a database that can efficiently provide users data on housing in New York City. Queries are designed to allow users to analyze the data and determine possible housing options. Numerous variables must be considered, such as total area of the borough, living space, crime rate, graduation rate, points of interest, etc., as they may affect the housing prices.

The user may also specify parameters, such as the borough one may want to live in. For example, if the user wants to search for houses that costs less than $100,000 in any borough, the query will return every city that satisfies this condition. The users can also specify other parameters as well, such as if one wants to live in Manhattan where the graduation rate is relatively high. Such flexibility allows the users to tailor their options based on their preferences.

## Specifications

While this project is limited to New York City only, it could be applied to other cities or even to a larger area such as counties and states. It also might be possible to add a more holistic sense to the database where it incorporates average cost for living (transit, food, utilities, school, etc.) and based off of income determine whether outside of the cost of housing is it reasonable to live in this location.

To improve performance, the database will have separate tables for non-unique attributes of house entities. It was intended to limit the number of properties needed to look at through query search by separating the tables.

The main security threat to the database would likely be from injection attacks. User inputs will be properly sanitized before being fed to the DBMS to prevent unauthorized access and modification to the system. The system will have minimal personal info attached to it, so that even if data was leaked it would not be a significant loss. Logs may be used to verify if data was accessed or modified by a malicious user, and backups may be required to restore the data to an accurate state if necessary.

To ensure the data can be recovered in the event of corruption or catastrophic storage failure, the team implemented multiple server redundancies. If one server with the data malfunctions, then there is another server with the data stored on it somewhere, and just general use of redundancy within the database ensures that data doesn't only have one copy.

The following techniques and languages were needed to complete this project.

- PostgreSQL
- Ruby on Rails
- CentOS
- SQL
- Normalization
- Database structure
- Data Integrity and Consistency

The plan is, either together or separately, use different resources such as textbooks or the internet to gather information about these topics.

The following data sets were used during this project. Each data set contained vital information that was deemed necessary for the project.

- Data on Houses:
  - https://www.kaggle.com/new-york-city/nyc-property-sales
- District Graduation Rates:
  - http://lohud.nydatabases.com/database/2017-18-new-york-graduation-rates-school-district
- NYC Crime Stats:
  - https://compstat.nypdonline.org/2e5c3f4b-85c1-4635-83c6-22b27fe7c75c/view/89
- Open Data:
  - https://opendata.cityofnewyork.us/
- Points of Interest:
  - https://data.cityofnewyork.us/City-Government/Points-Of-Interest/rxuy-2muj
- Borough Graduation Rates:
  - https://data.cityofnewyork.us/Education/2005-2010-Graduation-Outcomes-By-Borough/avir-tzek

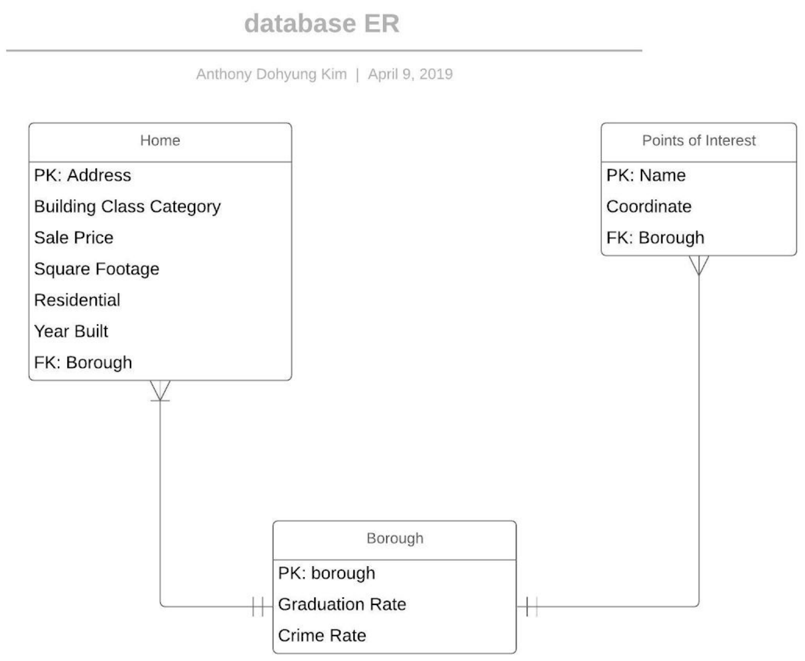
**Database Model**

The database consists of 3 tables: houses, boroughs, and points of interest. The tables and their attributes are listed as follows:

Home(**<u>Address</u>**, Category, SalePrice, SqFootage, Residential, YearBuilt, *Borough*)
Borough(**<u>Borough</u>**, GraduationRate, CrimeRate)
PointsOfInterst(**<u>XCoordinates</u>**, **<u>YCoordinates</u>**, Name, *Borough*)

It is noted that there are many homes to a borough as well as many points of interest to a borough. Conversely, there is only one borough for each house as well as only one borough for each point of interest.

The following diagram is a visual representation of the database schema:



There are approximately 84,000 housing entries expected to be pushed into the database.

There are numerous queries that will allow the user to search for houses under various parameters. Examples of such queries are listed in the following:

- Price Range Query
  - Will create a view of the available houses that fall into price ranges specified by the user.
  - Will request the database for rent expense attributes from the house table where the rent expense matches the range specified by the user.
    - SELECT Address, Sale Price
      FROM Home

WHERE Sale Price = User Selection;

- Square Footage Query
    - Creates view of available houses with an exact value or within a range specified by the user.
    - Requests database for square footage attribute from house table to find houses matching square footage specifications entered by the user.
        - SELECT Address, Square Footage
          FROM Home
          WHERE Square Footage BETWEEN User Selection - 100 AND User Selection + 100;
- Year Built Query
    - Creates view of available houses built in a certain year, or within a range of years specified by user
    - Requests database for year built attribute from house table to find houses matching year or range of years entered by the user.
        - SELECT Address
          FROM Home
          WHERE Year Built = User Selection;
- Crime Rate Query
    - Creates view of houses with a crime rate lower than the one entered by the user
    - Requests database for crime rate of each house through the borough attribute of each house in the house table where the crime rate is below a certain threshold.
        - SELECT Home.Address, Borough.CrimeRate
          FROM Home
          INNER JOIN Borough ON Home.Borough = Borough.Borough
          WHERE Borough.Crime Rate < User Selection;
- Points of Interest Query
    - Creates view of houses near point of attribute specified by the user.
    - Sends point of interest key to database to request houses in borough the primary key is in.
        - SELECT Home.Address, PointsOfInterest.Name
          FROM Home
          INNER JOIN PointsOfInterest ON Home.Borough = PointsOfInterest.Borough
          WHERE PointsOfInterest.Name = User Selection;
- Borough Query
    - Will create a view of the available houses that are in this borough.
    - Will request the database for borough attributes from house table where borough matches the borough specified by the user.
        - SELECT Home.Address, Borough.Borough
          FROM Home
          INNER JOIN Borough ON Home.Borough = Borough.Borough
          WHERE Home.Borough = User Selection;

- Graduation rating Query
    - Will return a result of the addresses with graduation ratings that have the rating specified by the user.
    - We will do a 5 star rating based off the normal distribution of graduation rates.
    - Will request database for graduation rating from borough table where school rating matches the specifications of the user and links it to the addresses from the house table.

        -
          ```
          SELECT Home.Address, Borough.GraduationRate
          FROM Home
          INNER JOIN Borough ON Home.Borough = Borough.Borough
          WHERE Borough.GraduationRate > User Selection;
          ```

- Combined Queries
    - Combined queries will ask the user for multiple criteria and return the results that match both these criteria.
        - There will be multiple of these based off the simple queries specified above and will result in "inner joins" between tables based off the keys when attributes are spread across multiple queries.
    - EXAMPLES:
        - Sale Price, Graduation Rate, & Crime Rate Query
        - Sale Price, Square Footage, Year Built, & Crime Rate Query
        - Sale Price, Square Footage, & Graduation Rate Query
        - Sale Price & Borough Query

Note that the combined queries will narrow the results even further since there are more parameters that the housing options need to satisfy.

**Design**

After a thorough review of the tables, the team determined that the tables are already in Boyce-Codd Normal Form. This is due to the fact that there are no dependencies between attributes besides the attribute's dependency on the primary key for each table. It is believed that this is the result of initial careful planning when designing the tables and understanding their relationship in the ER diagram. When creating each table, the team also ensured that the attributes for each table were related to the primary keys we specified. For example, in the borough table there are the attributes "Crime rate" and "Graduation rate". These two statistics are being sourced from the borough themselves and therefore are attributes of the borough and dependent on the borough primary key.

Since the queries rely heavily on user inputs and selections, the plan is to put in security features to ensure SQL injections won't give users unauthorized access to the database. This will be accomplished by preventing users from inputting SQL directly into the text boxes by preventing the input of SQL commands such as SELECT, FROM, WHERE, etc.

## Implementation

Our database was implemented through PostgreSQL with rails used as the framework. The MVC architecture of rails allowed us to make calls to the underlying database through ruby. The relational database was created first with the tables and attributes created through migrations and the associations reinforced through the models. Foreign keys were added to both the building and points of interest tables to facilitate this relationship. To deploy the searching functionality in the database, a Gem called simple form was used to allow us to write SQL statements directly in the framework. Various functions were created in the controller that allowed us to show the details of each house along with an index that showed all the houses. Since there were so many addresses in our database, a limit was set to the page to only show the top one hundred results. Views were created to implement the search bar and show the database search results to the end user.

## Future Work

At this point, the database will currently only ask the user to specify a price range and return results of addresses within that price range. In the future, we plan to deploy more queries that will allow users to narrow down the results further. Additionally, since the database requires user input, security against SQL injections will be added to prevent unauthorized access to our database.