Zillow Database

Team composition and responsibilities

1. Kirill Nartov - Responsible for data preparation and DDL/DML Statements, query scenario creation
2. Aloke Aggarwal - Responsible for the final report and ER Diagram, query scenario creation, performance recording, experiment design
3. Anish Kolaparthi - Responsible for SQL query execution and Experiment execution, performance indexing

Project Goal

The goal of this project is to test the complexity of queries that can be run in PostgreSQL 13 against a database made up of a number of smaller datasets from the real estate company Zillow.

This project will give us the opportunity to test the following concepts learned in class:

- Ability to design a conceptual model (ER diagram) for a real-world data set
- Ability to translate a conceptual model into a SQL schema
- Ability to create a PostgreSQL Database
- Ability to write proper SQL queries
- Ability to show the analysis of SQL queries

Attached Files

- Raw Zillow datasets (csv format): (5 files in original data folder in csv format)
- Data transformation/cleaning: cis556_data_transform.ipynb (.py also available)
- Transformed dataset (csv format): (5 files in transformed data folder in csv format)
- To load the data make sure to have t
- DDL statements: ddl_schema.sql
- DML statements: dml.sql - note that you must replace the user directory
- SQL queries + code for experiments: queries.sql

Dataset

We downloaded the Zillow datasets from https://www.zillow.com/research/data/ .
The dataset consists of 5 files from the website:

- **metro_forecast.csv** - contains the forecast for one month, 3 months, and 1 year to estimate the expected change in Zillow's Home value index (ZHVI) which tracks changes in values over time.
- **metro_median_price.csv** - monthly data on the median list price of homes in each Region of the US
- **metro_median_sale.csv** - monthly data on the median sale price of homes in each Region of the US
- **metro_share_listing_price_cut.csv** - monthly data on the percentage of listings that have had a list price cust in each region of the US
- **metro_zori.csv** - monthly data for rental prices going back 8 years.

Below is a snapshot of the data from relation metro_forecast.csv

RegionID	SizeRank	RegionName	RegionType	StateName	BaseDate	2023-11-30	2024-01-31	2024-10-31
102001	0	United States	country		2023-10-31	0.2	0.2	-0.1
394913	1	New York, NY	msa	NY	2023-10-31	0.2	-0.3	-3
753899	2	Los Angeles, CA	msa	CA	2023-10-31	0.7	0.4	-1.7
394463	3	Chicago, IL	msa	IL	2023-10-31	0.2	-0.1	-2.4
394514	4	Dallas, TX	msa	TX	2023-10-31	-0.1	-0.7	-1.1
394692	5	Houston, TX	msa	TX	2023-10-31	-0.2	-0.9	-2.1
395209	6	Washington, DC	msa	VA	2023-10-31	0	-0.5	-2.8
394974	7	Philadelphia, PA	msa	PA	2023-10-31	0.2	0	-0.9
394856	8	Miami, FL	msa	FL	2023-10-31	0.4	0.6	1.8
394347	9	Atlanta, GA	msa	GA	2023-10-31	0.1	0	1

Dataset Transformation

In the raw data set NULL values are encoded as blank.

No actual transformation was done other than to make the date variables not begin with numbers so that we could write the DDL statements.

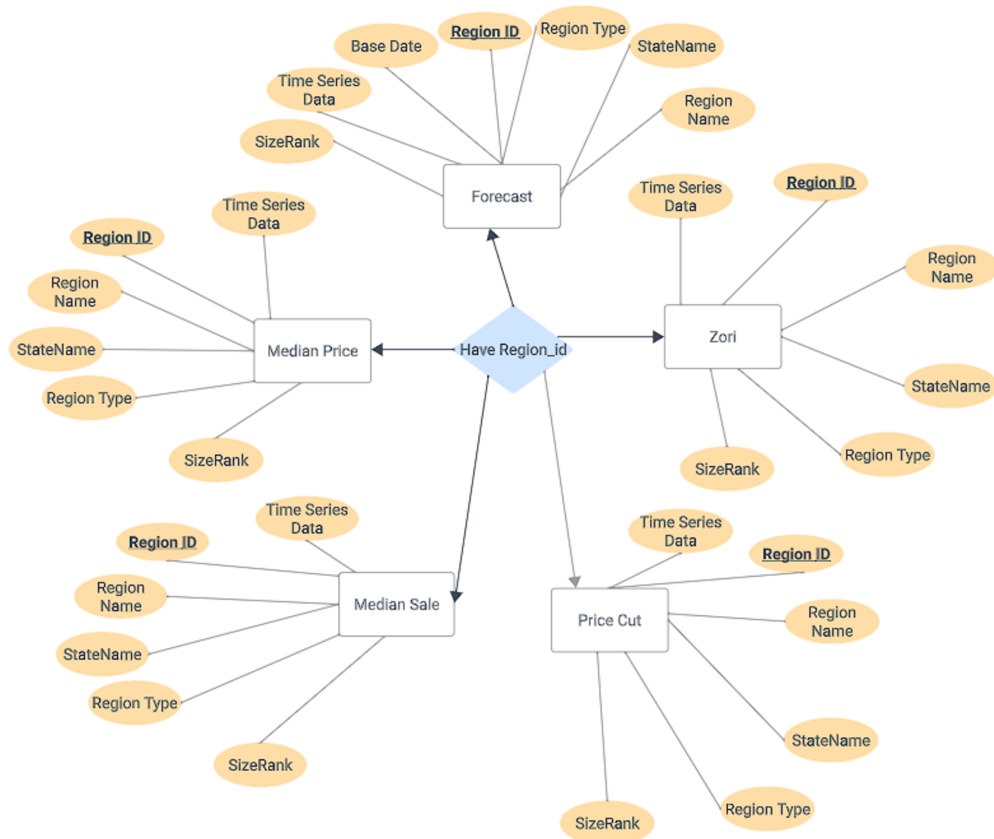
All the code for getting the column titles is in "cis556_data_transform.py"

Database Design

Because our data has useful attributes already stored in each file for aggregation, we opted to keep the file format as is. (See our conclusions for why this might not have been the best choice)

Conceptual Design

The ER diagram below provides the design we came up with for our database based on the datasets and their relationship to one another.



Database Schema

- **forecast**(<u>RegionID</u>, SizeRank, RegionName, RegionType, StateName, BaseDate, Date_11_30_2023, Date_1_31_2024, Date_10_31_2024)
- **median_Price**(<u>RegionID</u>, SizeRank, RegionName, RegionType, StateName, Date_2018_03_31, … , date_2023_10_31)
- **median_Sale**(<u>RegionID</u>, SizeRank, RegionName, RegionType, StateName, Date_2018_03_31, … , date_2023_10_31)
- **price_cut**(<u>RegionID</u>, SizeRank, RegionName, RegionType, StateName, Date_2018_03_31, … , date_2023_10_31)
- **zori**(<u>RegionID</u>, SizeRank, RegionName, RegionType, StateName, Date_2018_03_31, … , date_2023_10_31)

We converted the above conceptual design into the following SQL schema:

DROP TABLE IF EXISTS forecast CASCADE;
DROP TABLE IF EXISTS median_price CASCADE;
DROP TABLE IF EXISTS median_sale CASCADE;
DROP TABLE IF EXISTS price_cut CASCADE;
DROP TABLE IF EXISTS zori CASCADE;

–Forecast Table
CREATE TABLE forecast (
RegionID INT PRIMARY KEY,
SizeRank INT NOT NULL,
RegionName VARCHAR(40) NOT NULL, RegionType VARCHAR(10) NOT NULL, StateName VARCHAR(2),
BaseDate VARCHAR(20), Date_11_30_2023 FLOAT, Date_1_31_2024_ FLOAT, Date_10_31_2024_ FLOAT);

–Median Price Table
CREATE TABLE median_price (
RowIndex INT,
RegionID INT PRIMARY KEY,
SizeRank INT NOT NULL,
RegionName VARCHAR(40) NOT NULL, RegionType VARCHAR(10) NOT NULL, StateName VARCHAR(2),
date_2018_03_31 FLOAT, date_2018_04_30 FLOAT, date_2018_05_31 FLOAT, date_2018_06_30 FLOAT,
date_2018_07_31 FLOAT, date_2018_08_31 FLOAT, date_2018_09_30 FLOAT, date_2018_10_31 FLOAT,
date_2018_11_30 FLOAT, date_2018_12_31 FLOAT, date_2019_01_31 FLOAT, date_2019_02_28 FLOAT,
date_2019_03_31 FLOAT, date_2019_04_30 FLOAT, date_2019_05_31 FLOAT, date_2019_06_30 FLOAT,
date_2019_07_31 FLOAT, date_2019_08_31 FLOAT, date_2019_09_30 FLOAT, date_2019_10_31 FLOAT,
date_2019_11_30 FLOAT, date_2019_12_31 FLOAT, date_2020_01_31 FLOAT, date_2020_02_29 FLOAT,
date_2020_03_31 FLOAT, date_2020_04_30 FLOAT, date_2020_05_31 FLOAT, date_2020_06_30 FLOAT,
date_2020_07_31 FLOAT, date_2020_08_31 FLOAT, date_2020_09_30 FLOAT, date_2020_10_31 FLOAT,
date_2020_11_30 FLOAT, date_2020_12_31 FLOAT, date_2021_01_31 FLOAT, date_2021_02_28 FLOAT,
date_2021_03_31 FLOAT, date_2021_04_30 FLOAT, date_2021_05_31 FLOAT, date_2021_06_30 FLOAT,
date_2021_07_31 FLOAT, date_2021_08_31 FLOAT, date_2021_09_30 FLOAT, date_2021_10_31 FLOAT,
date_2021_11_30 FLOAT, date_2021_12_31 FLOAT, date_2022_01_31 FLOAT, date_2022_02_28 FLOAT,
date_2022_03_31 FLOAT, date_2022_04_30 FLOAT, date_2022_05_31 FLOAT, date_2022_06_30 FLOAT,
date_2022_07_31 FLOAT, date_2022_08_31 FLOAT, date_2022_09_30 FLOAT, date_2022_10_31 FLOAT,
date_2022_11_30 FLOAT, date_2022_12_31 FLOAT, date_2023_01_31 FLOAT, date_2023_02_28 FLOAT,
date_2023_03_31 FLOAT, date_2023_04_30 FLOAT, date_2023_05_31 FLOAT, date_2023_06_30 FLOAT,
date_2023_07_31 FLOAT, date_2023_08_31 FLOAT, date_2023_09_30 FLOAT, date_2023_10_31 FLOAT
);

–Median Sale Table
CREATE TABLE median_sale (
RowIndex INT,
RegionID INT PRIMARY KEY,
SizeRank INT NOT NULL,
RegionName VARCHAR(40) NOT NULL, RegionType VARCHAR(10) NOT NULL, StateName VARCHAR(2),
date_2018_03_31 FLOAT, date_2018_04_30 FLOAT, date_2018_05_31 FLOAT, date_2018_06_30 FLOAT,
date_2018_07_31 FLOAT, date_2018_08_31 FLOAT, date_2018_09_30 FLOAT, date_2018_10_31 FLOAT,
date_2018_11_30 FLOAT, date_2018_12_31 FLOAT, date_2019_01_31 FLOAT, date_2019_02_28 FLOAT,
date_2019_03_31 FLOAT, date_2019_04_30 FLOAT, date_2019_05_31 FLOAT, date_2019_06_30 FLOAT,
date_2019_07_31 FLOAT, date_2019_08_31 FLOAT, date_2019_09_30 FLOAT, date_2019_10_31 FLOAT,
date_2019_11_30 FLOAT, date_2019_12_31 FLOAT, date_2020_01_31 FLOAT, date_2020_02_29 FLOAT,
date_2020_03_31 FLOAT, date_2020_04_30 FLOAT, date_2020_05_31 FLOAT, date_2020_06_30 FLOAT,
date_2020_07_31 FLOAT, date_2020_08_31 FLOAT, date_2020_09_30 FLOAT, date_2020_10_31 FLOAT,
date_2020_11_30 FLOAT, date_2020_12_31 FLOAT, date_2021_01_31 FLOAT, date_2021_02_28 FLOAT,

date_2021_03_31 FLOAT, date_2021_04_30 FLOAT, date_2021_05_31 FLOAT, date_2021_06_30 FLOAT, date_2021_07_31 FLOAT, date_2021_08_31 FLOAT, date_2021_09_30 FLOAT, date_2021_10_31 FLOAT, date_2021_11_30 FLOAT, date_2021_12_31 FLOAT, date_2022_01_31 FLOAT, date_2022_02_28 FLOAT, date_2022_03_31 FLOAT, date_2022_04_30 FLOAT, date_2022_05_31 FLOAT, date_2022_06_30 FLOAT, date_2022_07_31 FLOAT, date_2022_08_31 FLOAT, date_2022_09_30 FLOAT, date_2022_10_31 FLOAT, date_2022_11_30 FLOAT, date_2022_12_31 FLOAT, date_2023_01_31 FLOAT, date_2023_02_28 FLOAT, date_2023_03_31 FLOAT, date_2023_04_30 FLOAT, date_2023_05_31 FLOAT, date_2023_06_30 FLOAT, date_2023_07_31 FLOAT, date_2023_08_31 FLOAT, date_2023_09_30 FLOAT);

–Price Cut Table

CREATE TABLE price_cut (
RowIndex INT,
RegionID INT PRIMARY KEY,
SizeRank INT NOT NULL,
RegionName VARCHAR(40) NOT NULL, RegionType VARCHAR(10) NOT NULL, StateName VARCHAR(2),
date_2018_03_31 FLOAT, date_2018_04_30 FLOAT, date_2018_05_31 FLOAT, date_2018_06_30 FLOAT, date_2018_07_31 FLOAT, date_2018_08_31 FLOAT, date_2018_09_30 FLOAT, date_2018_10_31 FLOAT, date_2018_11_30 FLOAT, date_2018_12_31 FLOAT, date_2019_01_31 FLOAT, date_2019_02_28 FLOAT, date_2019_03_31 FLOAT, date_2019_04_30 FLOAT, date_2019_05_31 FLOAT, date_2019_06_30 FLOAT, date_2019_07_31 FLOAT, date_2019_08_31 FLOAT, date_2019_09_30 FLOAT, date_2019_10_31 FLOAT, date_2019_11_30 FLOAT, date_2019_12_31 FLOAT, date_2020_01_31 FLOAT, date_2020_02_29 FLOAT, date_2020_03_31 FLOAT, date_2020_04_30 FLOAT, date_2020_05_31 FLOAT, date_2020_06_30 FLOAT, date_2020_07_31 FLOAT, date_2020_08_31 FLOAT, date_2020_09_30 FLOAT, date_2020_10_31 FLOAT, date_2020_11_30 FLOAT, date_2020_12_31 FLOAT, date_2021_01_31 FLOAT, date_2021_02_28 FLOAT, date_2021_03_31 FLOAT, date_2021_04_30 FLOAT, date_2021_05_31 FLOAT, date_2021_06_30 FLOAT, date_2021_07_31 FLOAT, date_2021_08_31 FLOAT, date_2021_09_30 FLOAT, date_2021_10_31 FLOAT, date_2021_11_30 FLOAT, date_2021_12_31 FLOAT, date_2022_01_31 FLOAT, date_2022_02_28 FLOAT, date_2022_03_31 FLOAT, date_2022_04_30 FLOAT, date_2022_05_31 FLOAT, date_2022_06_30 FLOAT, date_2022_07_31 FLOAT, date_2022_08_31 FLOAT, date_2022_09_30 FLOAT, date_2022_10_31 FLOAT, date_2022_11_30 FLOAT, date_2022_12_31 FLOAT, date_2023_01_31 FLOAT, date_2023_02_28 FLOAT, date_2023_03_31 FLOAT, date_2023_04_30 FLOAT, date_2023_05_31 FLOAT, date_2023_06_30 FLOAT, date_2023_07_31 FLOAT, date_2023_08_31 FLOAT, date_2023_09_30 FLOAT, date_2023_10_31 FLOAT
);

—Zori Table

CREATE TABLE zori (
RowIndex INT,
RegionID INT PRIMARY KEY,
SizeRank INT NOT NULL,
RegionName VARCHAR(40) NOT NULL, RegionType VARCHAR(10) NOT NULL, StateName VARCHAR(2),
date_2018_03_31 FLOAT, date_2018_04_30 FLOAT, date_2018_05_31 FLOAT, date_2018_06_30 FLOAT, date_2018_07_31 FLOAT, date_2018_08_31 FLOAT, date_2018_09_30 FLOAT, date_2018_10_31 FLOAT, date_2018_11_30 FLOAT, date_2018_12_31 FLOAT, date_2019_01_31 FLOAT, date_2019_02_28 FLOAT, date_2019_03_31 FLOAT, date_2019_04_30 FLOAT, date_2019_05_31 FLOAT, date_2019_06_30 FLOAT, date_2019_07_31 FLOAT, date_2019_08_31 FLOAT, date_2019_09_30 FLOAT, date_2019_10_31 FLOAT, date_2019_11_30 FLOAT, date_2019_12_31 FLOAT, date_2020_01_31 FLOAT, date_2020_02_29 FLOAT, date_2020_03_31 FLOAT, date_2020_04_30 FLOAT, date_2020_05_31 FLOAT, date_2020_06_30 FLOAT, date_2020_07_31 FLOAT, date_2020_08_31 FLOAT, date_2020_09_30 FLOAT, date_2020_10_31 FLOAT, date_2020_11_30 FLOAT, date_2020_12_31 FLOAT, date_2021_01_31 FLOAT, date_2021_02_28 FLOAT, date_2021_03_31 FLOAT, date_2021_04_30 FLOAT, date_2021_05_31 FLOAT, date_2021_06_30 FLOAT, date_2021_07_31 FLOAT, date_2021_08_31 FLOAT, date_2021_09_30 FLOAT, date_2021_10_31 FLOAT, date_2021_11_30 FLOAT, date_2021_12_31 FLOAT, date_2022_01_31 FLOAT, date_2022_02_28 FLOAT, date_2022_03_31 FLOAT, date_2022_04_30 FLOAT, date_2022_05_31 FLOAT, date_2022_06_30 FLOAT, date_2022_07_31 FLOAT, date_2022_08_31 FLOAT, date_2022_09_30 FLOAT, date_2022_10_31 FLOAT,

date_2022_11_30 FLOAT, date_2022_12_31 FLOAT, date_2023_01_31 FLOAT, date_2023_02_28 FLOAT, date_2023_03_31 FLOAT, date_2023_04_30 FLOAT, date_2023_05_31 FLOAT, date_2023_06_30 FLOAT, date_2023_07_31 FLOAT, date_2023_08_31 FLOAT, date_2023_09_30 FLOAT, date_2023_10_31 FLOAT);

DML Statements

We populated our schema with the following DML statements:

\copy forecast FROM '**/Users/kirillnartov/Downloads**/metro_forecast.csv' WITH DELIMITER ',' CSV HEADER;
\copy median_price FROM '**/Users/kirillnartov/Downloads**/metro_median_price_tranformed.csv' WITH DELIMITER ',' CSV HEADER;
\copy median_sale FROM '**/Users/kirillnartov/Downloads**/metro_median_sale_transformed.csv' WITH DELIMITER ',' CSV HEADER;
\copy price_cut FROM '**/Users/kirillnartov/Downloads**/metro_share_listings_price_cut_transformed.csv' WITH DELIMITER ',' CSV HEADER;
\copy zori FROM '**/Users/kirillnartov/Downloads**/metro_zori_transformed.csv' WITH DELIMITER ',' CSV HEADER;

Replace "'/Users/kirillnartov/Downloads/" with your own user directory.

Scenarios, Queries, and Methodology

We proposed a series of questions based on what we think are some real world questions that people would have regarding real estate.

Scenario 1

Ada works a remote job and is looking to downsize, she is looking for a home at the median price to buy in 3 months and she can pay the median home price of $360,000 today. What regions in the US do we think will have homes with a list within 10% of our price target in 3 months?

Query -

```
SELECT mp.RegionName, (mp.date_2023_10_31 * (fc.date_1_31_2024_ + 100) / 100) AS target_price
FROM median_price mp
INNER JOIN forecast fc ON mp.RegionId = fc.RegionID
WHERE (mp.date_2023_10_31 * (fc.date_1_31_2024_ + 100) / 100) BETWEEN 324000 AND 396000;
```

- What Techniques were needed in this query? Joins and where clauses
- How many tables were needed? 2
- What was the result? Ada has several options to buy across the country (+300).

Scenario 2

What was the average change in listing prices from before the public health emergency (date_2020_02_29) was declared to after it ended for Covid (Date_2023_05_31)?

Query -

```
select
avg(date_2023_05_31 - date_2020_02_29) as AVERAGECHANGE
from
median_price
where
date_2020_02_29 is not NULL
AND date_2023_05_31 is not NULL;
```

- What Technique is demonstrated? Aggregate function use in one table with a handling of Nulls.
- How many datasets were needed? 1
- What was the result? We determined that the average price change during covid was an increase in prices for homes by $82,000 dollars.

Scenario 3

How many regions in the US have more than 10% of listings with a price drop since the Federal Reserve began raising interest rates (Date_2023_03_31)?

Query -

```
Select
Count(*) as numberofregions
From
Price_cut
Where
Date_2023_03_31 is not null
```

- What Techniques were needed in this query? Counting
- How many tables were needed? 1
- What was the result? 248 regions have had price drops since the Federal reserve began raising rates.

Scenario 4

Bob is finalizing the sale of his home in Los Angeles, CA at the median price. With the money he is expected to receive, he saves 3% of it for rent in the next year. In what regions of California can he expect to be able to rent.

Query -

Select Rent for today for all regions in CA and be less than (subquery for median price today)*.35/12)

median price, ZIro

```
select
z.regionID, z.regionname, z.date_2023_10_31
from zori z
join median_sale ms
```

```
on ms.regionID = z.regionID
where
z.statename = 'CA'
and
z.date_2023_10_31 <
(select ((ms.date_2023_09_30*0.03)/12) as Median
 from median_sale ms
where ms.regionID = 753899);
```

- What Techniques were needed in this query? Sub-query
- How many datasets were needed? 2.
- What was the result? Bob has several options to choose from see below for a snapshot

Data Output Explain Messages Notifications

	regionid [PK] integer	regionname character varying (40)	date_2023_10_31 double precision
1	394619	Fresno, CA	1937.942262167
2	394357	Bakersfield, CA	1742.48902977947
3	394871	Modesto, CA	2049.29968287526
4	395195	Visalia, CA	1587.50569183903
5	394472	Clearlake, CA	1677.38888888889
6	394851	Merced, CA	1993.02238095238
7	395016	Red Bluff, CA	1530
8	394464	Chico, CA	1559.62
9	395018	Redding, CA	1401.0052546826
10	394559	El Centro, CA	1585

Scenario 5

The Federal Reserve is monitoring rent prices across the country and wants to see what the average rent is in each state for the past 3 years. We should also keep in mind the number of regions being observed.

```
select
statename,count(distinct regionID), Avg(z.date_2023_10_31) as Avg_2023, Avg(z.date_2022_10_31) as
Avg_2022, avg(z.date_2021_10_31) as Avg_2021
from zori z
where statename is not null
group by statename
order by statename;
```

- What Techniques were needed in this query? Distinct count and group by with null handling.
- How many datasets were needed? 1.
- What was the result? The fed is able to look at price changes across the past 3 years. See below for a snapshot.

	statename character varying (2)	count bigint	avg_2023 double precision	avg_2022 double precision	avg_2021 double precision
1	AK	2	1725.49776885319	1626.67472177178	1515.70179038569
2	AL	16	1319.85928666117	1324.76258631558	1240.44008886355
3	AR	8	1114.98780532743	1163.41959096533	1077.9413644516
4	AZ	7	1751.40268673658	1698.15796194509	1701.41338090878
5	CA	32	2288.03307199518	2324.3983170743	2246.44725746972
6	CO	14	2473.84293512298	2092.26059465866	1889.10652579764
7	CT	5	2047.99385053843	1892.28948556928	1717.66451101273
8	DE	2	1718.59682030932	1656.78763263268	1549.85293170192
9	FL	25	2040.53407081615	2009.17178524081	1797.81464456081
10	GA	22	1399.66964741268	1445.7557366334	1344.73905400608
11	HI	3	2700.85700089033	2550.44810504994	2296.72407815375

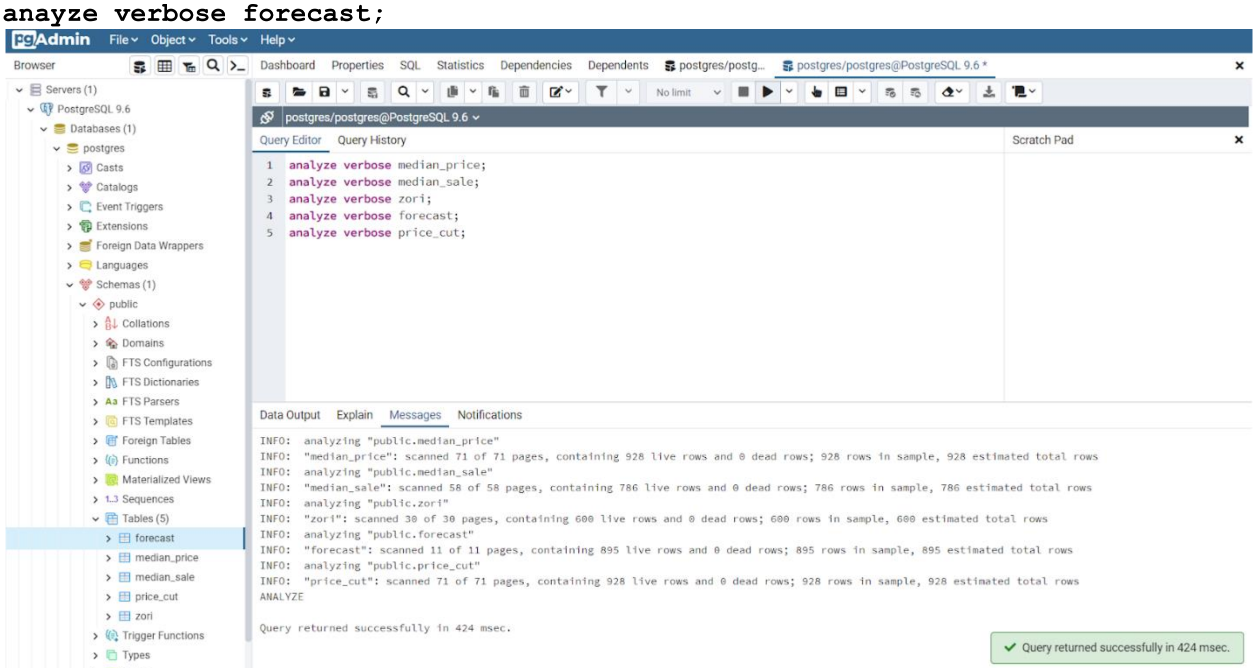
Experiment

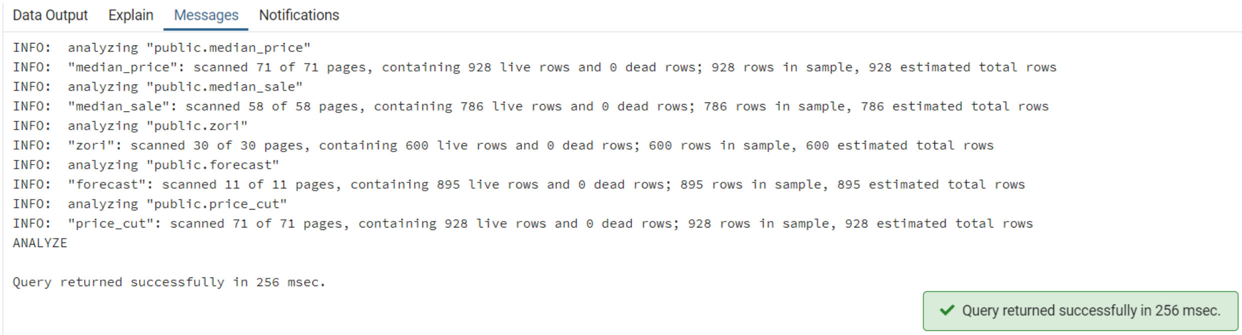
We used the command `EXPLAIN ANALYZE <query>.` to test each query using in the following scenarios

1. With statistics but without indexes
2. With both statistic and indexes

We used the following commands to collect the table statistics:

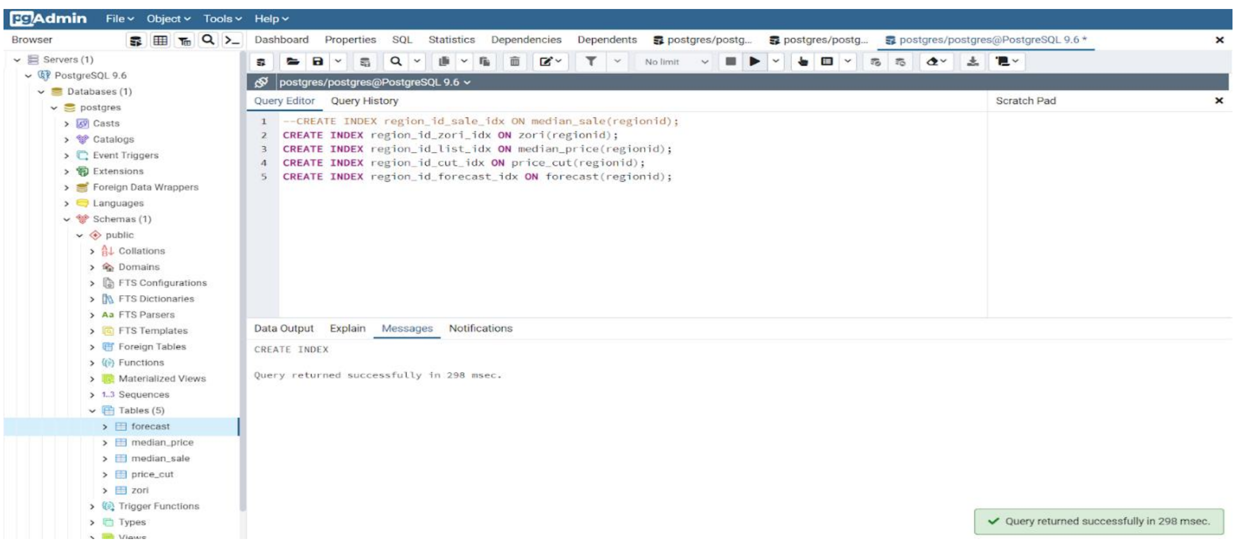
```
analyze verbose median_price;
analyze verbose median_sale;
analyze verbose zori;
analyze verbose price_cut;
anayze verbose forecast;
```





We used the following indexing scheme (Note we executed in our SQL query window and did not save this command separately):

CREATE INDEX region_id_sale_idx ON median_sale(regionid);
CREATE INDEX region_id_zori_idx ON zori(regionid);
CREATE INDEX region_id_list_idx ON median_price(regionid);
CREATE INDEX region_id_cut_idx ON price_cut(regionid);
CREATE INDEX region_id_forecast_idx ON forecast(regionid);



DROP INDEX region_id_sale_idx;
DROP INDEX region_id_zori_idx;
DROP INDEX region_id_list_idx;
DROP INDEX region_id_cut_idx;
DROP INDEX region_id_forecast_idx;

Benchmarks

Below are the results of 3 runs each of each query before and after indexing

Time (in s)	t1	t2	t3	Stat Avg	it1	it2	it3	Index Avg
Query 1	184	141	99	141	89	102	108	100
Query 2	108	88	83	93	127	71	69	89
Query 3	234	304	180	239	102	95	90	96
Query 4	105	247	124	159	163	73	108	115
Query 5	98	81	75	85	117	125	93	112

Below are screenshots of the query plans generated for one of each type of query before/after indexing

Query 1 Run 3 Example

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=178.63..178.87 rows=96 width=29) (actual time=2.328..2.346 rows=96 loops=1)
2	[...] Sort Key: mp.date_2023_06_30
3	[...] Sort Method: quicksort Memory: 32kB
4	[...] -> Hash Join (cost=88.44..175.47 rows=96 width=29) (actual time=1.025..2.236 rows=96 loops=1)
5	[...] Hash Cond: (pc.regionid = mp.regionid)
6	[...] -> Seq Scan on price_cut pc (cost=0.00..82.60 rows=925 width=12) (actual time=0.017..1.044 rows=925 loops=1)
7	[...] Filter: (date_2023_06_30 < '30'::double precision)
8	[...] Rows Removed by Filter: 3
9	[...] -> Hash (cost=87.24..87.24 rows=96 width=25) (actual time=0.984..0.984 rows=96 loops=1)
10	[...] Buckets: 1024 Batches: 1 Memory Usage: 14kB
11	[...] -> Seq Scan on median_price mp (cost=0.00..87.24 rows=96 width=25) (actual time=0.017..0.937 rows=96 loops=1)
12	[...] Filter: ((date_2023_06_30 >= '324000'::double precision) AND (date_2023_06_30 <= '396000'::double precision) AND (date_2022_06_30 < '360000'::double precision))
13	[...] Rows Removed by Filter: 832
14	Planning time: 0.334 ms
15	Execution time: 2.453 ms

Query 1 Indexed Run 3 Example

Data Output

	QUERY PLAN text
1	Hash Join (cost=31.14..142.49 rows=99 width=21) (actual time=0.555..2.642 rows=124 loops=1)
2	[...] Hash Cond: (mp.regionid = fc.regionid)
3	[...] Join Filter: ((((mp.date_2023_10_31 * (fc.date_1_31_2024_ + '100'::double precision)) / '100'::double precision) >= '324000'::double precision) AND (((mp.date_2023_10_31 * (fc.date_1_31_2024_ + '100'::double precision)) /
4	[...] Rows Removed by Join Filter: 771
5	[...] -> Seq Scan on median_price mp (cost=0.00..80.28 rows=928 width=25) (actual time=0.020..0.258 rows=928 loops=1)
6	[...] -> Hash (cost=19.95..19.95 rows=895 width=12) (actual time=0.505..0.505 rows=895 loops=1)
7	[...] Buckets: 1024 Batches: 1 Memory Usage: 47kB
8	[...] -> Seq Scan on forecast fc (cost=0.00..19.95 rows=895 width=12) (actual time=0.013..0.270 rows=895 loops=1)
9	Planning time: 0.454 ms
10	Execution time: 2.749 ms

Query 2 Run 3 example

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Aggregate (cost=84.72..84.73 rows=1 width=8) (actual time=1.112..1.113 rows=1 loops=1)
2	[...] -> Seq Scan on median_price (cost=0.00..80.28 rows=888 width=16) (actual time=0.023..0.879 rows=891 loops=1)
3	[...] Filter: ((date_2020_02_29 IS NOT NULL) AND (date_2023_05_31 IS NOT NULL))
4	[...] Rows Removed by Filter: 37
5	Planning time: 0.110 ms
6	Execution time: 1.195 ms

Query 2 Indexed Run 3 Example

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Aggregate (cost=84.72..84.73 rows=1 width=8) (actual time=0.772..0.772 rows=1 loops=1)
2	[...] -> Seq Scan on median_price (cost=0.00..80.28 rows=888 width=16) (actual time=0.022..0.617 rows=891 loops=1)
3	[...] Filter: ((date_2020_02_29 IS NOT NULL) AND (date_2023_05_31 IS NOT NULL))
4	[...] Rows Removed by Filter: 37
5	Planning time: 0.094 ms
6	Execution time: 0.823 ms

Query 3 Run 3 Example

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Aggregate (cost=82.59..82.60 rows=1 width=8) (actual time=0.995..0.995 rows=1 loops=1)
2	[...] -> Seq Scan on price_cut (cost=0.00..80.28 rows=924 width=0) (actual time=0.022..0.893 rows=924 loops=1)
3	[...] Filter: (date_2023_03_31 IS NOT NULL)
4	[...] Rows Removed by Filter: 4
5	Planning time: 0.096 ms
6	Execution time: 1.050 ms

Query 3 Indexed Run 3 Example

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Aggregate (cost=82.59..82.60 rows=1 width=8) (actual time=0.842..0.843 rows=1 loops=1)
2	[...] -> Seq Scan on price_cut (cost=0.00..80.28 rows=924 width=0) (actual time=0.018..0.754 rows=924 loops=1)
3	[...] Filter: (date_2023_03_31 IS NOT NULL)
4	[...] Rows Removed by Filter: 4
5	Planning time: 0.151 ms
6	Execution time: 0.898 ms

Query 4 Run 3 Example

	QUERY PLAN text
1	Nested Loop (cost=8.57..114.63 rows=11 width=25) (actual time=0.126..0.669 rows=16 loops=1)
2	[...] InitPlan 1 (returns $0)
3	[...] -> Index Scan using median_sale_pkey on median_sale ms_1 (cost=0.28..8.30 rows=1 width=8) (actual time=0.011..0.012 rows=1 loops=1)
4	[...] Index Cond: (regionid = 753899)
5	[...] -> Seq Scan on zori z (cost=0.00..39.00 rows=11 width=25) (actual time=0.106..0.585 rows=16 loops=1)
6	[...] Filter: ((date_2023_10_31 < $0) AND ((statename)::text = 'CA'::text))
7	[...] Rows Removed by Filter: 584
8	[...] -> Index Only Scan using median_sale_pkey on median_sale ms (cost=0.28..6.11 rows=1 width=4) (actual time=0.003..0.004 rows=1 loops=16)
9	[...] Index Cond: (regionid = z.regionid)
10	[...] Heap Fetches: 16
11	Planning time: 0.447 ms
12	Execution time: 0.778 ms

Query 4 Indexed Run 3 Example

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Nested Loop (cost=8.57..114.63 rows=11 width=25) (actual time=0.105..0.542 rows=16 loops=1)
2	[...] InitPlan 1 (returns $0)
3	[...] -> Index Scan using region_id_sale_idx on median_sale ms_1 (cost=0.28..8.30 rows=1 width=8) (actual time=0.014..0.014 rows=1 loops=1)
4	[...] Index Cond: (regionid = 753899)
5	[...] -> Seq Scan on zori z (cost=0.00..39.00 rows=11 width=25) (actual time=0.091..0.470 rows=16 loops=1)
6	[...] Filter: ((date_2023_10_31 < $0) AND ((statename)::text = 'CA'::text))
7	[...] Rows Removed by Filter: 584
8	[...] -> Index Only Scan using region_id_sale_idx on median_sale ms (cost=0.28..6.11 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=16)
9	[...] Index Cond: (regionid = z.regionid)
10	[...] Heap Fetches: 16
11	Planning time: 0.648 ms
12	Execution time: 0.629 ms

Query 5 Run 3
Example

Data Output Explain Messages Notifications

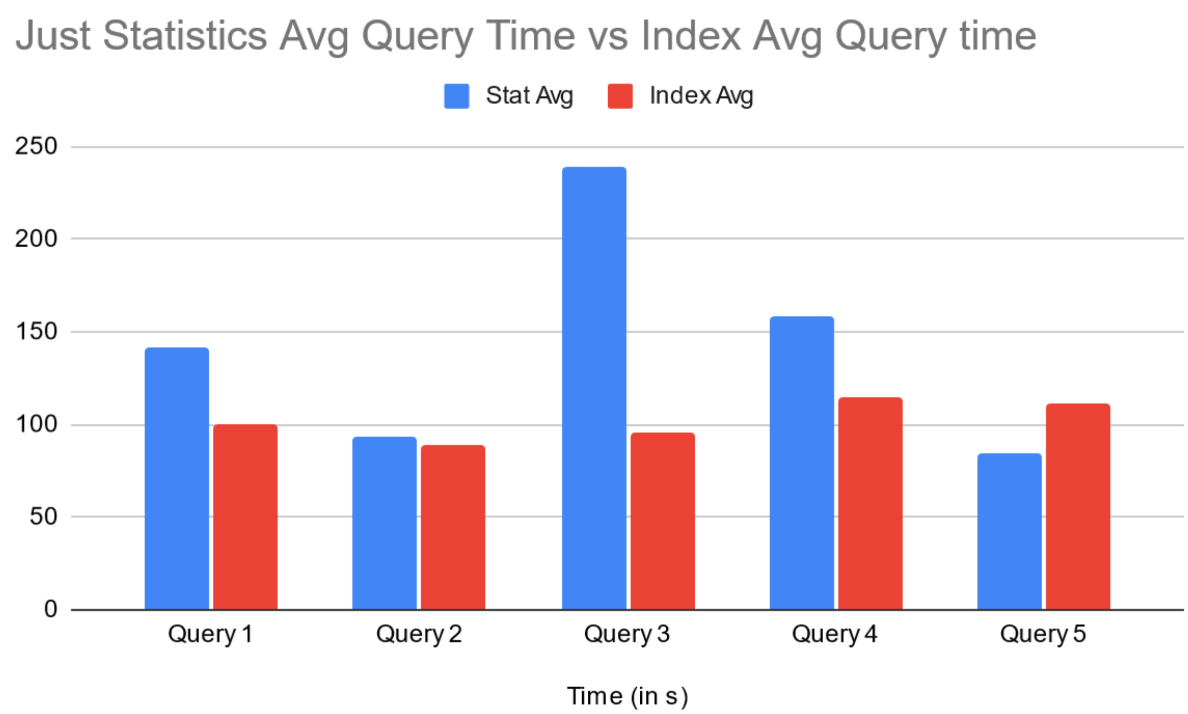
	QUERY PLAN text
1	GroupAggregate (cost=63.63..73.49 rows=50 width=35) (actual time=1.577..2.085 rows=50 loops=1)
2	[...] Group Key: statename
3	[...] -> Sort (cost=63.63..65.13 rows=599 width=31) (actual time=1.541..1.607 rows=599 loops=1)
4	[...] Sort Key: statename
5	[...] Sort Method: quicksort Memory: 66kB
6	[...] -> Seq Scan on zori z (cost=0.00..36.00 rows=599 width=31) (actual time=0.018..0.422 rows=599 loops=1)
7	[...] Filter: (statename IS NOT NULL)
8	[...] Rows Removed by Filter: 1
9	Planning time: 0.109 ms
10	Execution time: 2.151 ms

Query 5 Indexed Run 3 Example -

Data Output Explain Messages Notifications

	QUERY PLAN text
1	Sort (cost=45.77..45.90 rows=50 width=35) (actual time=0.538..0.541 rows=50 loops=1)
2	[...] Sort Key: statename
3	[...] Sort Method: quicksort Memory: 28kB
4	[...] -> HashAggregate (cost=43.49..44.36 rows=50 width=35) (actual time=0.435..0.445 rows=50 loops=1)
5	[...] Group Key: statename
6	[...] -> Seq Scan on zori z (cost=0.00..36.00 rows=599 width=31) (actual time=0.017..0.100 rows=599 loops=1)
7	[...] Filter: (statename IS NOT NULL)
8	[...] Rows Removed by Filter: 1
9	Planning time: 0.104 ms
10	Execution time: 0.629 ms

Here is a plot to summarize our findings



Instructions for reproducing the experiments

- The experiments were simply run by first collecting the statistics as indicated above.
- Next we added explain analyze before each query and recorded the time observed.
- Next we added in the indexes and then ran the same queries again.

CONCLUSION

- Indexing did save time on queries but using the distinct count in query 5 seems to have hurt indexing.
- It would be interesting to see how it performs on a large database.
- Looking at the analysis on the query optimizer it looks like the main difference is the use of a hash function. Although the optimizer expected its plan to be faster, it turned out to not be more efficient than ignoring the index.
- Overall, in this project we were able to create a database that we could run queries against using real world data.
- One thing about our data is that it already had many attributes of the data joined to its datasets.
- In a real-world database, these attributes would be in a separate attribute table and just be connected via region ID.

- Due to how we created our database, our ER diagram looks quite cluttered and messy. Below is a more maintainable database design.
- To update the database below you could add region ids when necessary. For new data you would have to add new columns in each table separately and then update the values by regionid.