CS 779 FINAL PROJECT

Weixuan Huang

Abstract

This Final Project Report lists out the steps and details of the building process of the food data warehouse, visualization using Tableau and Python

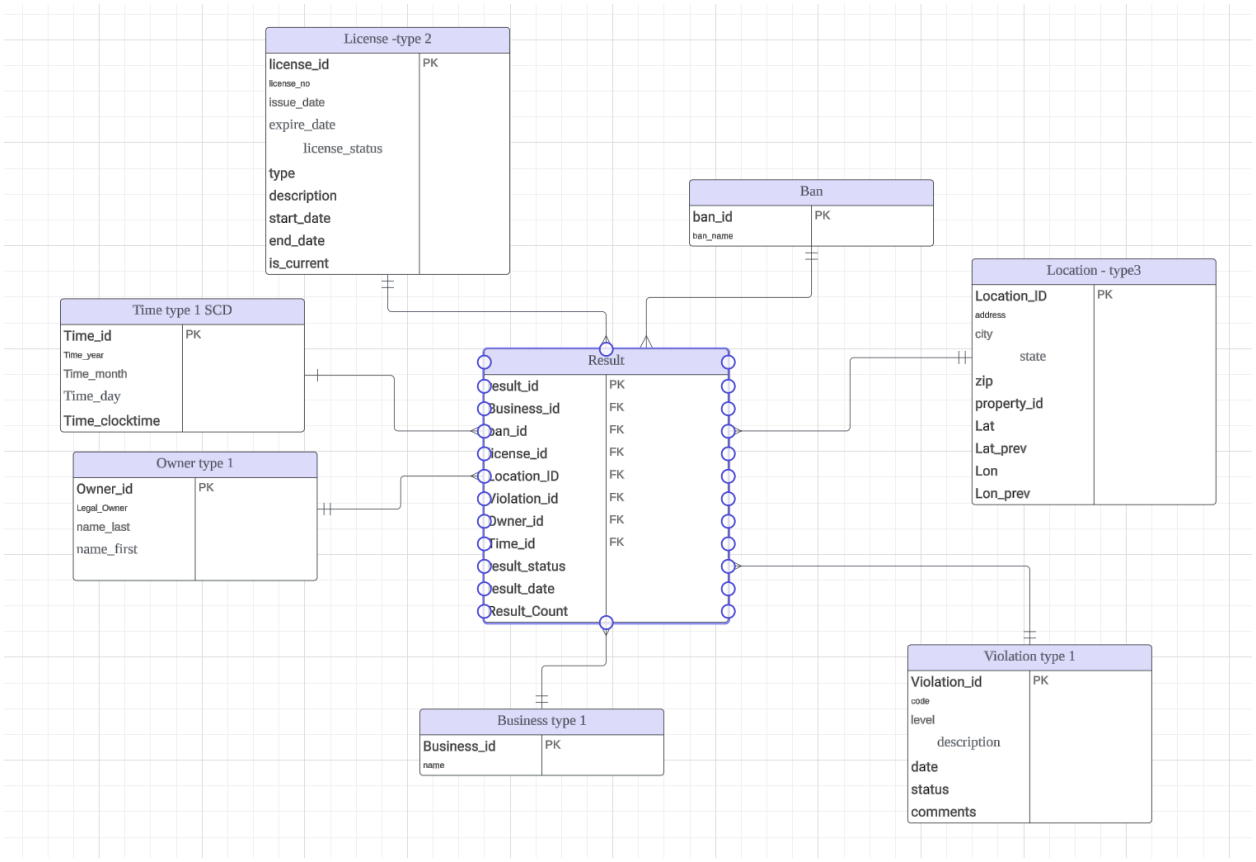
Weixuan Huang
whx@bu.edu

Table of Contents

1. Introduction

The dataset I have used is the Boston Food Establishment Inspections, which includes the inspection and related information for each food business. It specifies the detailed information for each business and each inspection for each business.

2.DImensional Schema Design



I used the star schema design for the crime data warehouse.
It includes 1 fact table: Result. 6 Dimensional tables: Ban, Location, Violation, Business, Owner, License.

1)SCD Explanation

Type 1: Overwrites old data with new data, and does not track historical data.

Type 2: Keeps historical data by adding new records with a version or effective date.

Type 3: Adds new columns to track changes and typically keeps limited historical data.

I made the Location to be type3 SCD and added two new columns lat_prev and loc_prev to keep track of the latitude and longitude changes.

I made the License to be the type2 SCD so we can see the slowly changing of the License.

The rest is type 1 SCD.

2) Table Explanation

- The load_food table holds data related to the food dataset
- The Ban, Business, Owner, Violation tables contain the ban name, business names, owner's information and Violation details
- The Location table is a type 3 SCD table.
- The License table is a type 2 SCD table.
- The time contains the result year, month, and day

3. ETL Process

Pre-processing:

Before loading data into SQL officially, I have loaded the data in Python.

```
df = pd.read_csv('/Users/weixuanhuang/Desktop/CS
779/final/food.csv')
```

Using the pandas package.

The original Location is a format like (42.35925907091849, -71.05890092520279) . To better use it, I convert it to two column and remove the brackets.

```
df[['lat', 'lon']] =
df['location'].str.extract(r'\((.*), (.*)\)')

df['lat'] = pd.to_numeric(df['lat'])

df['lon'] = pd.to_numeric(df['lon'])
```

Then drop location

```
df = df.drop(columns=['location'])
```

Also there are some NAs that I want to fill for column property_id.

```
df['property_id'] =
df['property_id'].fillna(0).astype(int)
```

Also for column zip, there are lots of incorrect formats there.

First I would like to convert the empty rows to the '00000' format which represents the invalid zip code. Other than that, there are some invalid zip codes like 2018.0 and the actual zip code is 02018, so I converted it to be the correct zip code. The other types of zip codes like 01245, 01234-39 are the correct types.

```
# some zip code are in wrong format like 02108 ->
2108.0

def format_zip(zip_code):

   if pd.isna(zip_code) or zip_code == '':

       return '00000'  # Default for missing or
empty ZIP codes

   elif '.' in zip_code:

       # Remove the decimal and format as a
five-digit number
```

```
        zip_code = f"{int(float(zip_code)):05d}"

        return zip_code

    else:

        return zip_code # keep it the same



# Apply the function to the 'zip' column

df['zip'] = df['zip'].astype(str) # convert it to
string first

df['zip'] = df['zip'].apply(format_zip)
```

And I handled the date here:

```
# Convert date strings to datetime objects

df['issdttm'] = pd.to_datetime(df['issdttm'])

df['expdttm'] = pd.to_datetime(df['expdttm'])

# and for the license start-effective-date and
end-effective-date

df['start_date'] = df['expdttm']
```

```
df['end_date'] = df['expdttm'].shift(-1) -
pd.Timedelta(days=1)
```

Then I make it to a new dataset:

```
df.to_csv('/Users/weixuanhuang/Desktop/CS
779/final/Bos_food.csv', index=False)
```

Extract Phase:

I created one staging table which is used to hold the information from the food dataset

```
CREATE TABLE load_food (
    id BIGINT Primary Key,
    business_name VARCHAR(255),
    dbanname VARCHAR(255),
    legalowner VARCHAR(255),
    namelast VARCHAR(255),
    namefirst VARCHAR(255),
    licenseNO INT,
    issdttm Date,
    expdttm Date,
    licstatus VARCHAR(30),
    licsesecat VARCHAR(10),
    descript VARCHAR(60),
    result VARCHAR(30),
    resultdttm timestamp,
    violation VARCHAR(60),
    violevels Varchar(10),
    viodesc VARCHAR(255),
    viodttm DATE,
    viostatus VARCHAR(20),
    status_date DATE,
    comments VARCHAR(5000),
    address VARCHAR(255),
    city VARCHAR(60),
    state VARCHAR(10),
    zip varchar(100),
    property_id BIGINT,
    lat float,
    lon float,
    lat_prev float,
    lon_prev float,
    start_date date,
    end_date date
);
```

After creating the 2 tables, then we can load the dataset into the 2 tables.

I used PostgreSQL for the final project, so the command lines I have used in psql query is :

\copy load_food FROM '/Users/weixuanhuang/Desktop/CS 779/final/Bos_food.csv' DELIMITER ',' CSV HEADER;

Transform Phase:

The transformations like cleaning and handling data have been done through Python as I have shown above..

Load Phase:

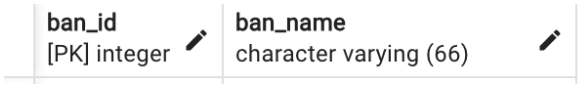
Before loading all the data into each dimensional table and fact table, I create the dimensional tables first.

License:

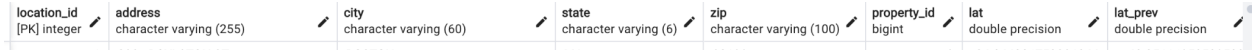
Time:

The full_time is the indexing that I have prepared for the fact table loading since redoing the extract() will slow the query down

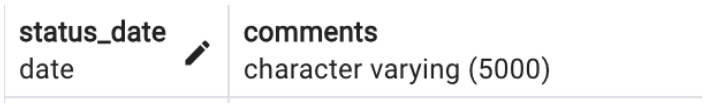
Ban:



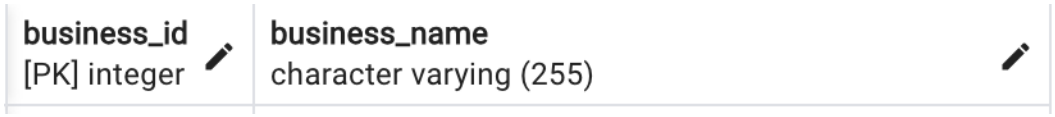
Location:



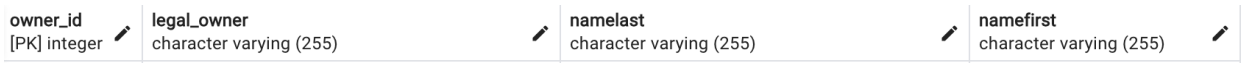
VIolation:



Business:



Owner:



And then is the fact table **Result**

Then we can load the data into these tables:

License:

The License is a type 2 SCD, so it includes the start effective date and the end effective date, is_current flag.

In order to set the start/end effective date and the is_current. I created 3 columns at the end of the table and named them in sequence.

For start_date, I set it to be the expiration date for each license. For the end_date, I set it to the next row's expiration date - 1 day. And the is_current flag is based on the end_date

EX:

issue_date	expire_date	start_date	end_date	is_current

03/22/2023	08/22/2023	08/22/2023	10/21/2023	False
05/24/2023	10/22/2023	10/22/2023	10/21/9999	Ture

As you can see above, we set the start date to be the current row's expire_date

And the end_date to be next row's expire_date - 1 day.

For is_current, since the 10/21/2023 is past so we set it to False. For the second row's is_current since the 9999 is a future year, so we set it to be True.

In order to make all the things happen and due to the reason that the dimensional table should get rid of the duplicate rows first and then insert all the information to the License table.

The reason why I don't handle them in one query is because SQL can not handle the two tasks properly at the same time.

EX:

key	issue_date	expire_date	start_date	end_date	is_current
1	03/22/2023	08/22/2023			
2	03/22/2023	08/22/2023			
3	03/22/2023	08/22/2023			
4	xx	xx			

For example, my original table looks above, the row 1, 2, 3 are the same and the start/end date for 1 and 2 will be the same, if I process them directly. And if I **select distinct** at the same query it will keep the first row instead but not the last row.

So, I copied all distinct values into a cp table first.

```
drop table cp_License;

create table cp_License(
    license_id serial primary key,
    license_no int,
    issue_date date,
    expire_date date,
    license_status varchar(50),
    cattype varchar(10),
    description varchar(60)
);
insert into cp_License (license_no, issue_date, expire_date, license_status, cattype, description)
select distinct
    licenseno,
    issdttm,
    expdttm,
    licstatus,
    licsesecat,
    descript
from load_food;
```

	license_id [PK] integer	license_no integer	issue_date date	expire_date date	license_status character varying (50)	cattype character varying (10)	description character varying (60)
1	1	308959	2016-11-15	2022-01-01	Inactive	RF	Retail Food
2	2	27964	2011-12-28	2025-01-01	Active	RF	Retail Food
3	3	137823	2015-03-16	2017-01-01	Inactive	FT	Eating & Drinking w/ Take Out
4	4	77756	2012-05-15	2016-01-01	Inactive	FS	Eating & Drinking
5	5	126083	2014-02-24	2019-01-01	Inactive	FT	Eating & Drinking w/ Take Out
6	6	81278	2013-05-24	2021-01-01	Inactive	FS	Eating & Drinking
7	7	32976	2011-02-10	2010-12-31	Inactive	FT	Eating & Drinking w/ Take Out
8	8	30120	2008-06-06	2009-01-01	Inactive	RF	Retail Food
9	9	18398	2010-05-07	2011-01-01	Inactive	FS	Eating & Drinking
10	10	130633	2016-05-05	2025-01-01	Active	FS	Eating & Drinking
11	11	20003	2011-03-08	2012-01-01	Inactive	FT	Eating & Drinking w/ Take Out
12	12	33149	2011-12-19	2015-01-01	Inactive	RF	Retail Food

And then I used the MERGE to merge the temporary data table into the actual License table.

```
MERGE INTO License l
USING (
    SELECT
        license_no,
        issue_date,
        expire_date,
        license_status,
        cattype,
        description,
        expire_date AS start_date,
        COALESCE(LEAD(expire_date) OVER (ORDER BY license_id) - INTERVAL '1 day', '9999-12-31') AS end_d
        CASE
            WHEN COALESCE(LEAD(expire_date) OVER (ORDER BY license_id) - INTERVAL '1 day', '9999-12-31'
            ELSE FALSE
        END AS is_current
    FROM cp_License
) AS t
ON t.license_no = l.license_no
WHEN MATCHED THEN
    UPDATE SET
        issue_date = t.issue_date,
        expire_date = t.expire_date,
        license_status = t.license_status,
        cattype = t.cattype,
        description = t.description,
        start_date = t.start_date,
        end_date = t.end_date,
        is_current = t.is_current
WHEN NOT MATCHED THEN
    INSERT (license_no, issue_date, expire_date, license_status, cattype, description, start_date, end_d
    VALUES (t.license_no, t.issue_date, t.expire_date, t.license_status, t.cattype, t.description, t.sta
```

Now the actual License table has the start/end date and is_current aligned with its original order.

	d ger	license_no integer	issue_date date	expire_date date	license_status character varying (50)	cattype character varying (10)	description character varying (60)	start_date date	end_date date	is_current boolean
1	1	308959	2016-11-15	2022-01-01	Inactive	RF	Retail Food	2022-01-01	2024-12-31	true
2	2	27964	2011-12-28	2025-01-01	Active	RF	Retail Food	2025-01-01	2016-12-31	false
3	3	137823	2015-03-16	2017-01-01	Inactive	FT	Eating & Drinking w/ Take Out	2017-01-01	2015-12-31	false
4	4	77756	2012-05-15	2016-01-01	Inactive	FS	Eating & Drinking	2016-01-01	2018-12-31	false
5	5	126083	2014-02-24	2019-01-01	Inactive	FT	Eating & Drinking w/ Take Out	2019-01-01	2020-12-31	false
6	6	81278	2013-05-24	2021-01-01	Inactive	FS	Eating & Drinking	2021-01-01	2010-12-30	false
7	7	32976	2011-02-10	2010-12-31	Inactive	FT	Eating & Drinking w/ Take Out	2010-12-31	2008-12-31	false
8	8	30120	2008-06-06	2009-01-01	Inactive	RF	Retail Food	2009-01-01	2010-12-31	false
9	9	18398	2010-05-07	2011-01-01	Inactive	FS	Eating & Drinking	2011-01-01	2024-12-31	true
10	10	130633	2016-05-05	2025-01-01	Active	FS	Eating & Drinking	2025-01-01	2011-12-31	false
11	11	20003	2011-03-08	2012-01-01	Inactive	FT	Eating & Drinking w/ Take Out	2012-01-01	2014-12-31	false
12	12	33149	2011-12-19	2015-01-01	Inactive	RF	Retail Food	2015-01-01	2014-12-31	false
13	13	68028	2012-06-06	2015-01-01	Inactive	FT	Eating & Drinking w/ Take Out	2015-01-01	2024-12-31	true
14	14	21833	2012-02-13	2025-01-01	Active	FS	Eating & Drinking	2025-01-01	2017-12-31	false
15	15	26936	2012-02-24	2018-01-01	Inactive	FT	Eating & Drinking w/ Take Out	2018-01-01	2007-12-31	false

Time:

For the Time table, I used the resultdttm as the date that we want to extract for.

	time_id [PK] integer	full_time timestamp without time zone	time_year integer	time_month integer	time_day integer	time_clocktime time without time zone
5	5	2019-06-07 19:49:34	2019	6	7	19:49:34
6	6	2018-11-20 15:05:38	2018	11	20	15:05:38
7	7	2012-02-22 16:59:52	2012	2	22	16:59:52
8	8	2011-09-26 18:38:30	2011	9	26	18:38:30
9	9	2011-10-17 13:58:22	2011	10	17	13:58:22
10	10	2020-09-01 19:29:21	2020	9	1	19:29:21
11	11	2011-11-07 16:31:58	2011	11	7	16:31:58
12	12	2013-07-30 14:43:47	2013	7	30	14:43:47
13	13	2010-10-05 14:06:21	2010	10	5	14:06:21
14	14	2012-04-30 15:31:56	2012	4	30	15:31:56
15	15	2020-09-11 16:18:14	2020	9	11	16:18:14
16	16	2011-11-15 19:28:46	2011	11	15	19:28:46

Total rows: 1000 of 168740 Query complete 00:00:00.100

Ban:

Ban is a type 1 and it only has one column to load.

```
248 insert into Ban(ban_name)
249 select distinct
250     dbanname
251 from load_food;
252
253 select * from Ban;
254
255
```

Data Output | Messages | Notifications

	ban_id [PK] integer	ban_name character varying (66)
1	1	[null]
2	2	Baggage Claim
3	3	Upper Level/Main Food Court
4	4	#477
5	5	1844 Inc
6	6	Devonshire Wine LLC
7	7	74 LTD.
8	8	Cha Kung Fong
9	9	[null]
10	10	Big Fish Promotions LLC
11	11	Milalilu Inc.
12	12	Shanti Boston LLC

Location:

The Location is the type3 SCD, so I have added two new columns the lat_prev and lon_prev to it which represents previous latitude and longitude separately.

Similar to what I have done for License.

I first created a temporary holder to hold the information without the previous Latitude and previous Longitude.

```
create table cp_Location(
    location_id serial primary key,
    address varchar(255),
    city varchar(60),
    state varchar(6),
    zip varchar(100),
    property_id BIGINT,
    lat float,
    lon float
);

insert into cp_Location(address, city, state, zip, property_id, lat, lon)
select distinct
address,
city,
state,
zip,
property_id,
lat,
lon
from load_food;
```

And then I used the MERGE to merge the temporary data table into the actual Location table.

```
MERGE INTO Location l
USING (
    SELECT
        address,
        city,
        state,
        zip,
        property_id,
        lat,
        COALESCE(LAG(lat) OVER (ORDER BY location_id), lat) AS lat_prev,
        lon,
        COALESCE(LAG(lon) OVER (ORDER BY location_id), lon) AS lon_prev
    FROM cp_Location
) AS t
ON l.property_id = t.property_id
WHEN MATCHED THEN
    UPDATE SET
        address = t.address,
        city = t.city,
        state = t.state,
        zip = t.zip,
        lat = t.lat,
        lat_prev = CASE
            WHEN l.lat <> t.lat THEN l.lat
            ELSE l.lat_prev
        END,
        lon = t.lon,
        lon_prev = CASE
            WHEN l.lon <> t.lon THEN l.lon
            ELSE l.lon_prev
        END
WHEN NOT MATCHED THEN
    INSERT (address, city, state, zip, property_id, lat, lat_prev, lon, lon_prev)
    VALUES (t.address, t.city, t.state, t.zip, t.property_id, t.lat, t.lat_prev, t.lon, t.lon_prev);
```

Now if we select one address from the Location, you can see the location is now unique:

select * from Location where address = '55 COURT ST';

```
select * from Location where address = '55  COURT ST';


-- violation
insert into Violation(code, level, description, vio_date, status, status_date, comments)
```

Data Output | Messages | Notifications

	location_id [PK] integer	address character varying (255)	city character varying (60)	state character varying (6)	zip character varying (100)	property_id bigint	lat double precision	lat_prev double precision	lon
1	2149	55 COURT ST	BOSTON	MA	02108	156226	42.35925907091849	42.3711954972086	-7

VIolation:

For Violation, it is a type 1 SCD.

```
-- violation
insert into Violation(code, level, description, vio_date, status, status_date, comments)
select
    violation,
    violevels,
    viodesc,
    viodttm,
    viostatus,
    status_date,
    comments
from load_food;

select * from Violation;
select * from Violation where code = '15-4-202.16';
```

Data Output | Messages | Notifications

	violation_id [PK] integer	code character varying (60)	level character varying (10)	description character varying (255)
	176411	08-3-305-307.11	*	Food Protection
	176412	14-4-202.11	*	Food Contact Surfaces Design
	176413	24-4-903.11	*	Clean Equipment & Utensils Storage
	176414	35-6-501.111/.115	**	Insects Rodents Animals
	176415	36-6-501.11-.12	*	Improper Maintenance of Floors
	176416	37-6-501.11-.12	*	Improper Maintenance of Walls/Ceilings
	176417	42-6-501.113/.114	*	Premises Maintained

Business:

Business is a type 1 SCD table too.

And it also has lots of duplicate rows so I only load the unique business names into it

```
insert into Business(business_name)
select
Distinct business_name
from load_food;

-- checking
select * from Business where business_name = '1000 Degrees Pizza';
select * from Business;
```

ata Output | Messages | Notifications

business_id [PK] integer	business_name character varying (255)
1	Mass General Hospital/Blossom Cafe
2	Element Boston Seaport Hotel Rise
3	CVS Pharmacy No. 11201
4	TD Garden L3 Pantry Rm. No. SE-325
5	Tandoor & Curry on Wheels
6	Courtyard Boston Downtown Market
7	BOSTON PIZZA & GRILL
8	J.P. LICKS INC.
9	ADAMS ST SHELL
10	Laffa Vegetarian

Owner:

Owner is also the type 1 SCD.

```
insert into Owner(legal_owner, namelast, namefirst)
select
Distinct(legalowner),
namelast,
namefirst
from load_food;
select * from Owner;

-- checking
select * from Owner where legal_owner = 'TERADYNE INC';

select * from load_food where legalowner = 'TERADYNE INC';
```

Data Output | Messages | Notifications

	owner_id [PK] integer	legal_owner character varying (255)	namelast character varying (255)	namefirst character varying (255)
1	1		JSR Enterprises	Jim Robichau
2	2	LEON VICTOR J TS	Taylor Sales Tax Manager	Jon
3	3	NEW COMMONWEALTH COMMERCIAL	MP Sports Club Boston LLC	[null]
4	4	NAHIM WILLIAM G	Ortiz	Efrain
5	5	FINOCCHIARO LUCILLE TS	Pred	Lucretia
6	6	[null]	Big Night Venue Boston LLC	[null]
7	7	[null]	Susan Cheng	[null]
8	8	[null]	Dente	Gerald
9	9	TAM HO	Veth	Vivian
10	10	[null]	Stefonopoulos Inc.	Sotiris Stefonopoulos
11	11	DELTA AIRLINES ATTN: BAR BARA GOSLIN	Boston Food Concepts	Mohammad Sadiq
12	12	SOLUTEK CORP MASS CORP	Get Steamed Inc.	[null]

Result:

The result is the fact table for the design and it includes the measure result_count.

Since the dataset is really large and I only want to present the structure for the result table, I used the inner join and limited it to a limited amount of rows to save time.

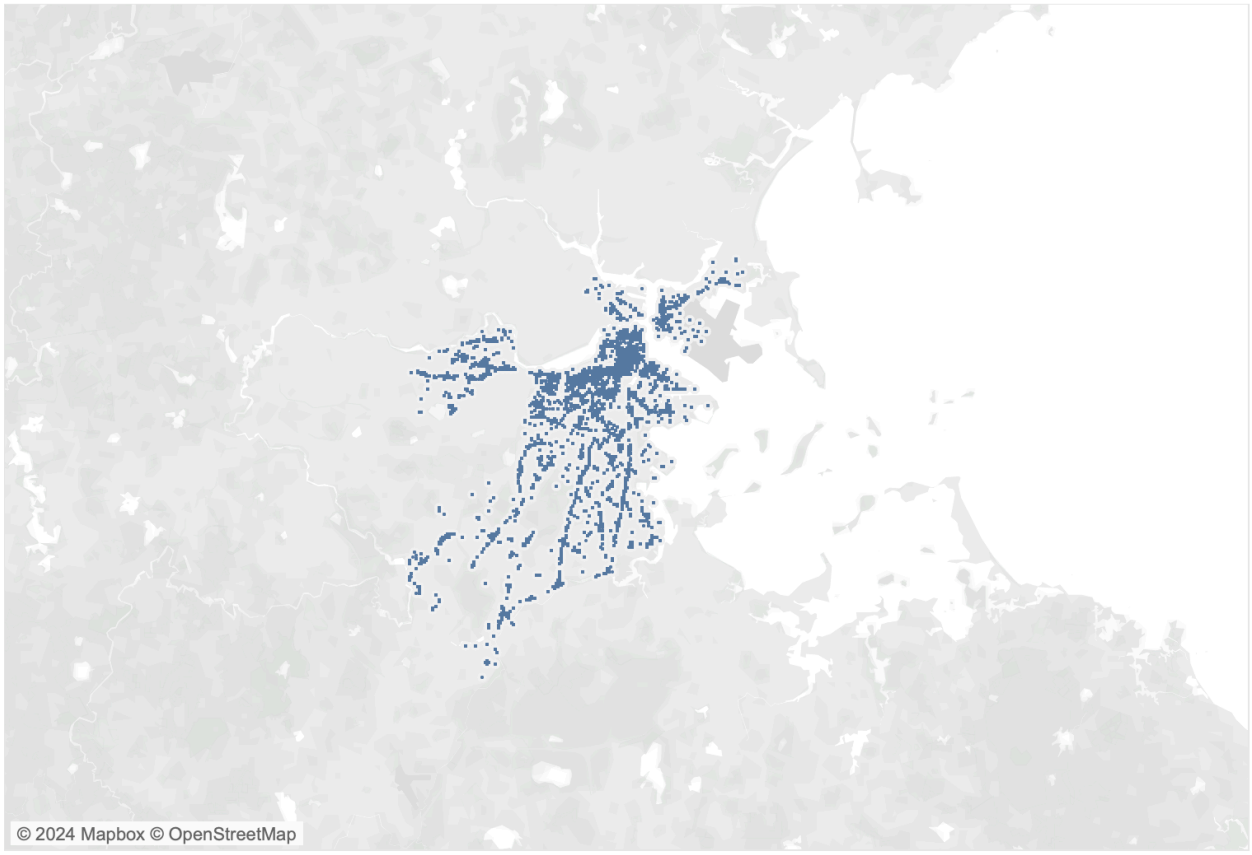
```
-- explain
INSERT INTO Result (license_id, ban_id, location_id, violation_id, business_id, owner_id, time_id, status, result_dat
select
l.license_id,
b.ban_id,
loc.location_id,
v.violation_id,
bus.business_id,
o.owner_id,
t.time_id,
lf.result,
lf.resultdttm,
COUNT(*) AS result_count -- each combination has several inspection_result
from
load_food lf
inner join License l on l.license_no = lf.licenseno
inner join Ban b on b.ban_name = lf.dbanname
inner join Location loc on loc.address = lf.address
inner join Violation v on v.code = lf.violation
inner join Business bus on bus.business_name = lf.business_name
inner join Owner o on o.namelast = lf.namelast and o.namefirst = lf.namefirst
inner join Time t on t.full_time = lf.resultdttm
group by l.license_id, b.ban_id, loc.location_id, v.violation_id, bus.business_id, o.owner_id, t.time_id, lf.result,
limit 10000;
```

	result_id [PK] integer	license_id bigint	ban_id bigint	location_id bigint	violation_id bigint	business_id bigint	owner_id bigint	time_id bigint	status character varying (60)	result_date date	result_count bigint
1	1	58	95	2151	2	6553	5477	279	HE_Filed	2008-12-02	1
2	2	58	95	2151	4	6553	5477	9381	HE_Fail	2013-03-15	1
3	3	58	95	2151	4	6553	5477	38066	HE_Pass	2013-03-21	1
4	4	58	95	2151	5	6553	5477	4326	HE_Filed	2009-04-14	1
5	5	58	95	2151	5	6553	5477	9381	HE_Fail	2013-03-15	1
6	6	58	95	2151	5	6553	5477	38066	HE_Pass	2013-03-21	1
7	7	58	95	2151	5	6553	5477	38428	HE_Fail	2008-01-29	1
8	8	58	95	2151	6	6553	5477	9381	HE_Fail	2013-03-15	1

4.Tableau

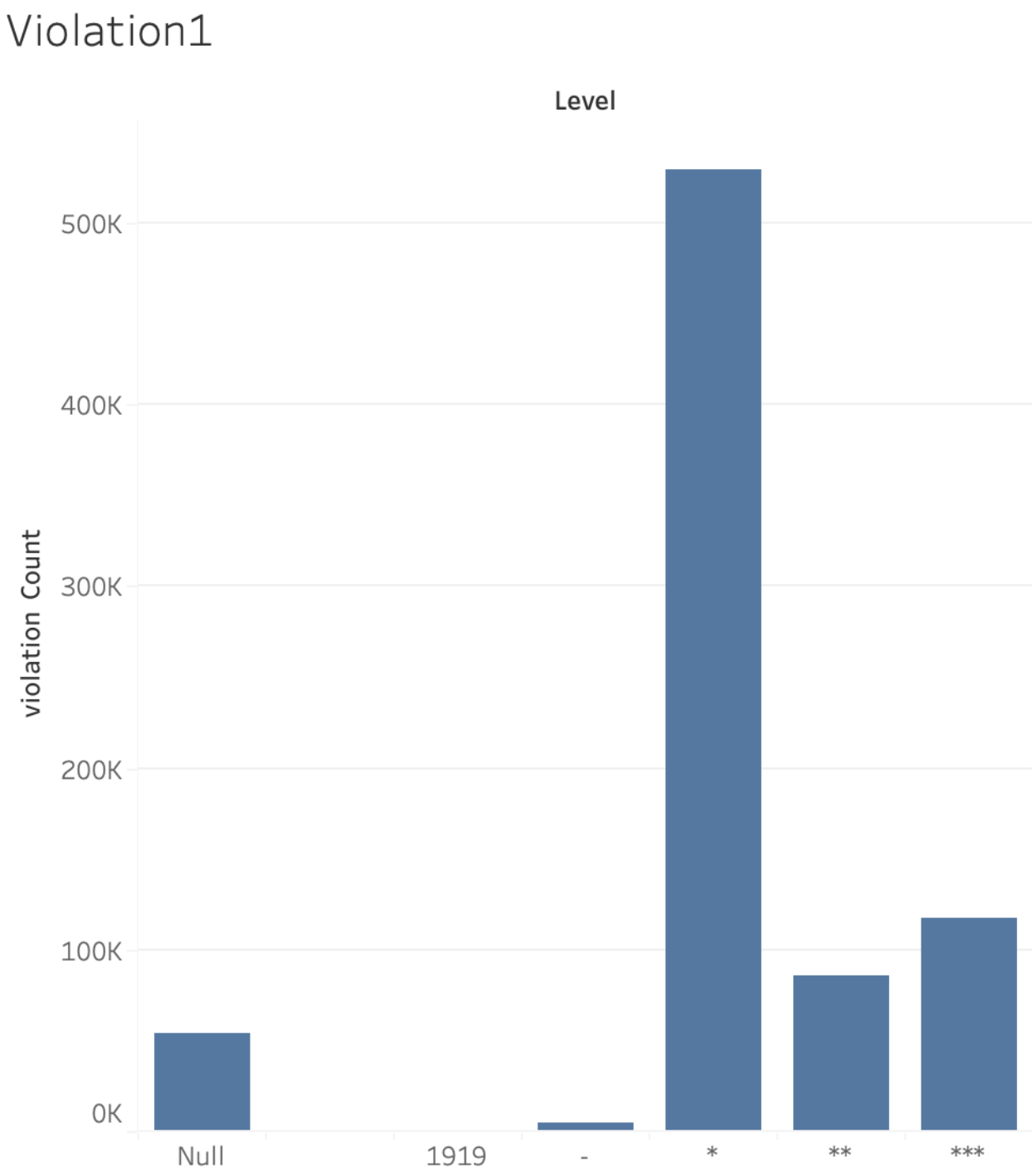
I have created several sheets based on the code.
The first one is the map. You can see the location information for it, the result, violation levels, violation description, business names, business description, and the final inspection result for it.

Map



Map based on Lon (Load Food) and Lat (Load Food). Details are shown for various dimensions. The data is filtered on Zip (Load Food), which keeps 73 of 73 members. The view is filtered on Lat (Load Food), Lon (Load Food) and Violevels. The Lat (Load Food) filter includes everything. The Lon (Load Food) filter includes everything. The Violevels filter keeps 7 of 7 members.

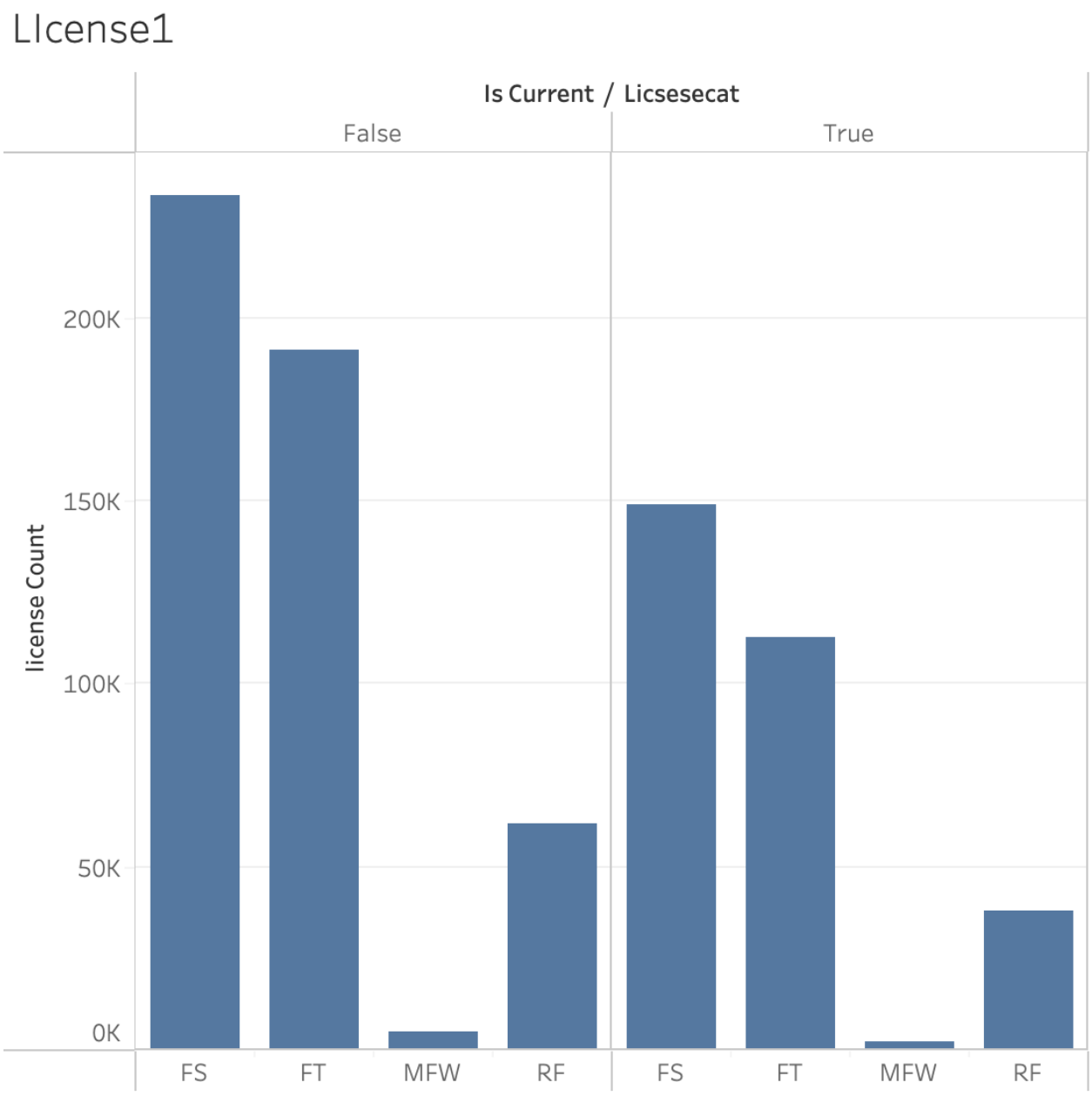
The second one is for Violation.



Count of violation for each Level. The view is filtered on Level, which keeps 7 of 7 members.

By using the filters, you can see different levels of Violations passing or failing rate.

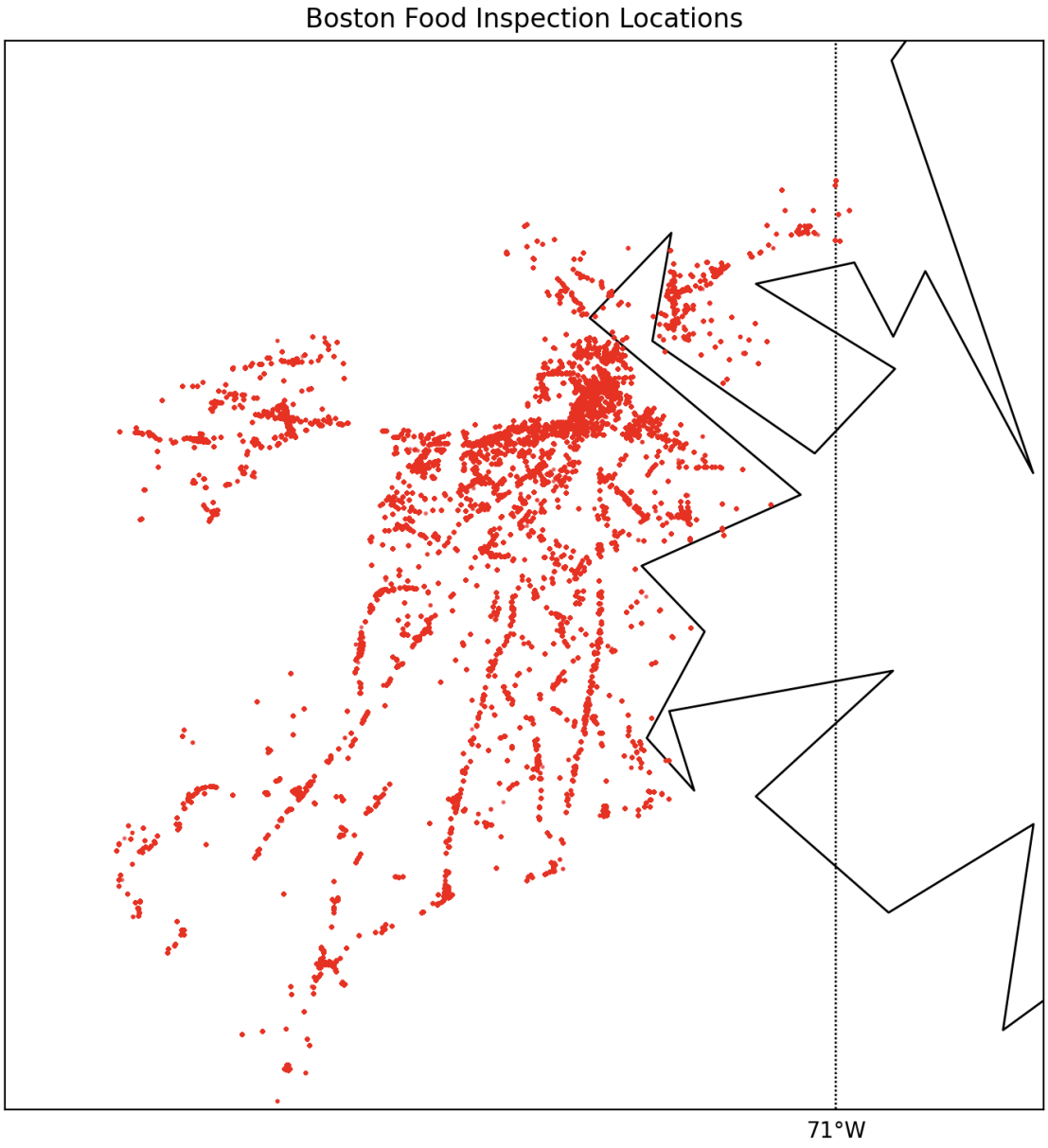
Next ons is about the License



Count of license for each Is Current.
You can see the different types of license are active or inactive for now

You can see different types of each license and the associated is_current flag for them which represent that how many/what type licenses are active or inactive for now

5.Python Map Visualization



6.Conclusion

Overall, the food data warehouse offers the users one option to store all a big set of data inside in a well structured and organized large data warehouse that the users can view and process the data easier instead of using a large Excel file. And users can extract the useful information they want by writing the queries.

7. Revision History

A history of things you added and why, not required but nice to have.

Name	Date	Version	Description
Weixuan Huang	04/27/24	1.0	Initial Document Creation
Weixuan Huang	04/28/24	1.1	Introduction
Weixuan Huang	04/28/24	1.2	Schema Design
Weixuan Huang	04/30/24	1.3	ETL Process
Weixuan Huang	04/30/24	1.4	Fixing all the insertion
Weixuan Huang	05/01/24	1.5	Finish the report
Weixuan Huang	05/01/24	1.6	Fixing Issues

Appendices

Appendix A

Bibliography:

badges, mahdi yousefimahdi yousefi 84711. “Sql - Get Previous and next Row from Rows Selected with (WHERE) Conditions - Stack Overflow.” *Stack Overflow*, https://stackoverflow.com/questions/27086377/get-previous-and-next-row-from-rows-selected-with-where-conditions. Accessed 1 May 2024.

How to Join a fact and a type 2 dimension (SCD2) table. (2021, October 30).

https://www.startdataengineering.com/post/how-to-join-fact-scd2-tables/

How to get the next row in sql. (n.d.). Stack Overflow.

https://stackoverflow.com/questions/20342045/how-to-get-the-next-row-in-sql

GeeksforGeeks. (2023, May 20). *CTE in SQL*. GeeksforGeeks.

https://www.geeksforgeeks.org/cte-in-sql/

SQL query to show time trend of a SCD Type 2 dimension table. (n.d.). Stack Overflow.

https://stackoverflow.com/questions/63296187/sql-query-to-show-time-trend-of-a-scd-type-2-dimension-table

SQL | Window Functions | LEAD() | CodeCademy. (2023, April 7). Codecademy.

https://www.codecademy.com/resources/docs/sql/window-functions/lead

Relate your data. (n.d.). Tableau.

https://help.tableau.com/current/pro/desktop/en-us/relate_tables.htm

MySQL EXTRACT() function. (n.d.). https://www.w3schools.com/sql/func_mysql_extract.asp