# Principles of Database Systems

**Aigerim (Aigo) Madakimova**
am9634@nyu.edu

## Introduction

The project objective is a website where users can add their favorite restaurants and order the food on a regular basis from their list of restaurants. Modern consumers greatly benefit from the "restaurant save list" system. It is so useful to get a list of restaurants that match consumers' preferences without much hassle, comparing, and browsing through a long list of reviews for every single restaurant.

## ER Diagram

The entity Relationship Diagram is used to design database data models.
I used Postgres software to draw an ER diagram where:
The Yellow key icons on tables are the primary key for the restaurant table
The Gray Key icons are the Foreign keys of another table.
The Dashed-lines but in the diagram below lines running across tables show the relationship of tables with each other.

- Entity:

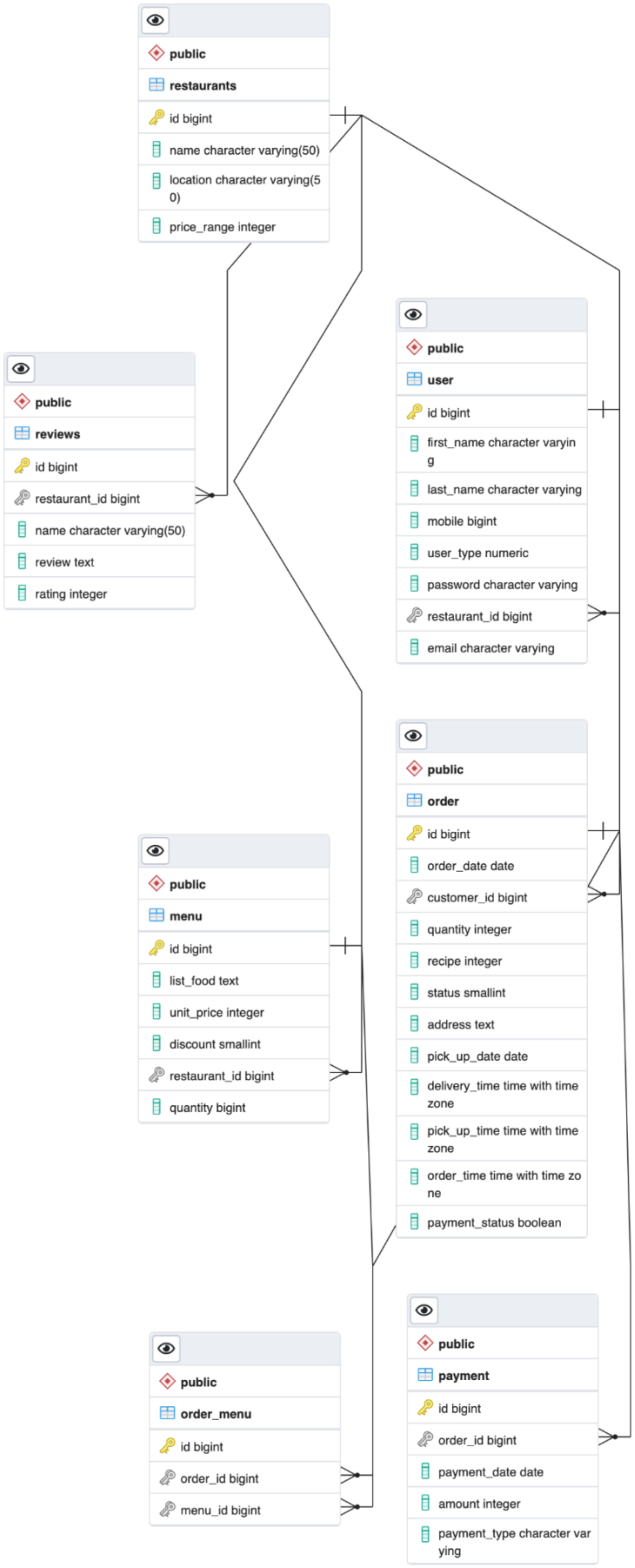
Entities represent data components within a database that could be living or non-living, real or abstract, so long as their data is stored in the database. In ER diagrams, entities are usually depicted by rectangles, with the entity name at the top.
My entities are restaurants, reviews, menu, user, order, order_menu, and payment.

- Attribute:

An attribute is a description of the properties of an entity or relationship. For example, restaurant attributes name, location, and price range. We also can see types of attributes such as characters or integers.

- Relationship:

A relationship denotes how various entities interact with each other, it can be One-to-One and One-to-Many. For example, one to many restaurants have a relationship with reviews, menus, and users, and one-to-one is payment to order.


public
restaurants
id bigint
name character varying(50)
location character varying(50)
price_range integer
public
reviews
id bigint
restaurant_id bigint
name character varying(50)
review text
rating integer
public
user
id bigint
first_name character varying
last_name character varying
mobile bigint
user_type numeric
password character varying
restaurant_id bigint
email character varying
public
order
id bigint
order_date date
customer_id bigint
quantity integer
recipe integer
status smallint
address text
pick_up_date date
delivery_time time with time zone
pick_up_time time with time zone
order_time time with time zone
payment_status boolean
public
menu
id bigint
list_food text
unit_price integer
discount smallint
restaurant_id bigint
quantity bigint
public
order_menu
id bigint
order_id bigint
menu_id bigint
public
payment
id bigint
order_id bigint
payment_date date
amount integer
payment_type character varying

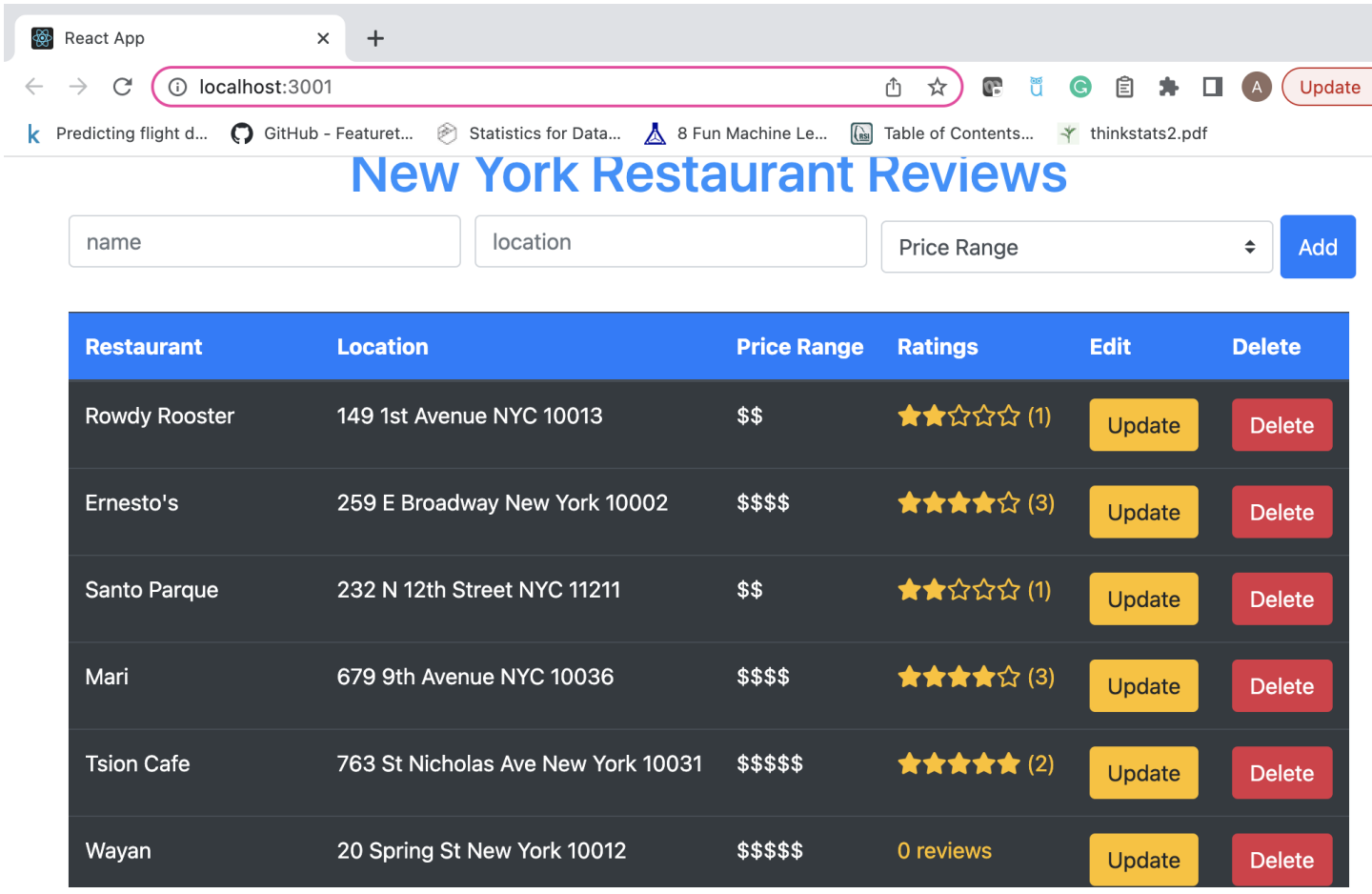
# Application Architecture Plan

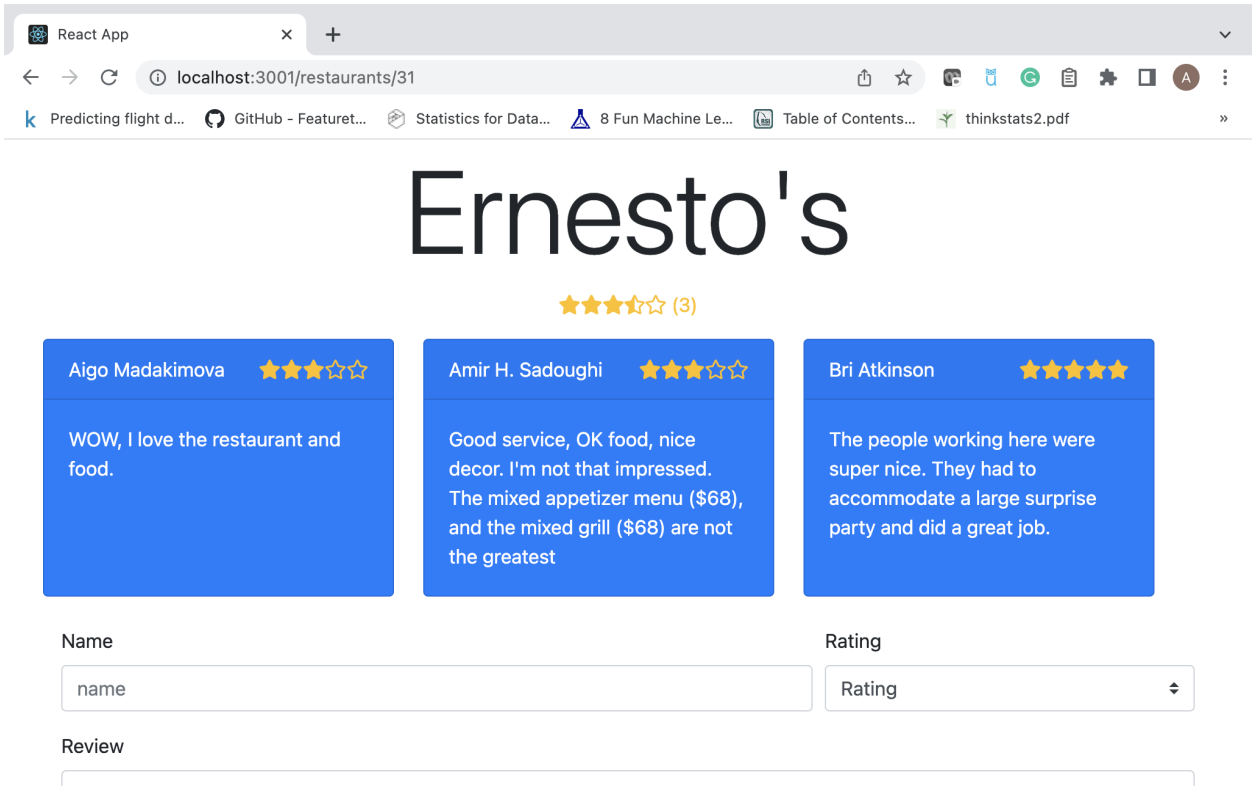
Backend: Node
Frontend: JavaScript/React, Database: Postgres

The project will utilize Node.js uses JavaScript on the server for the backend and the backend will consist of a web server that connects to a PostgreSQL database.

The front end will issue requests to the webserver and display the results, as well as send updates.

Below is a list of NYC restaurants, people can add the name of the restaurant, address, price range, and rating. It also can be updated or deleted.
A person can click on a restaurant's name and see others' comments, she/he can add a review, but the review can't be deleted, and the overall review ranking would be calculated.





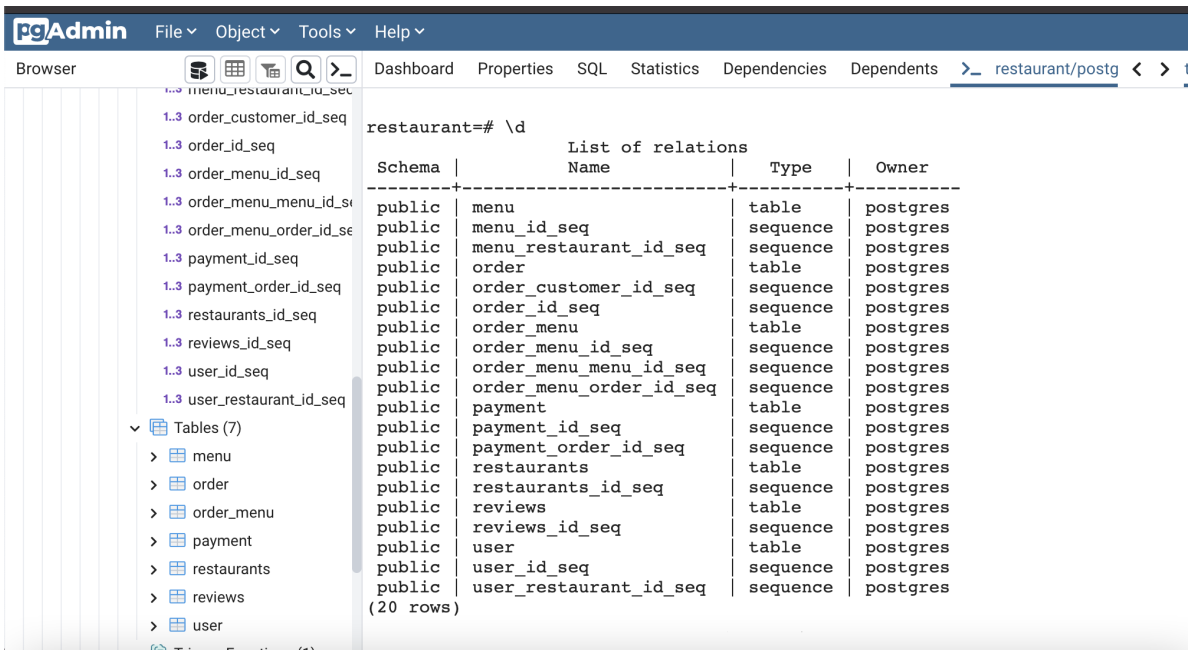

**Source code:** https://github.com/aigOffline/DATABASE_SQL/tree/master

## ● Table DDL:

Required 7 Tables, you can see in my ER 7 tables.

The DDL (Data Definition Language) commands are used to define the database.
Example: CREATE, DROP, ALTER, TRUNCATE, COMMENT, RENAME.

Using psql command '\d' I can view all tables which I created SQL commands:



- *CREATE*:

```
CREATE TABLE IF NOT EXISTS public.menu
(
    id bigint NOT NULL DEFAULT nextval('menu_id_seq'::regclass),
    list_food text COLLATE pg_catalog."default",
    unit_price integer NOT NULL,
    discount smallint,
    restaurant_id bigint NOT NULL DEFAULT nextval('menu_restaurant_id_seq'::regclass),
    quantity bigint NOT NULL,
    CONSTRAINT menu_pkey PRIMARY KEY (id),
    CONSTRAINT restaurant_fkey FOREIGN KEY (restaurant_id)
        REFERENCES public.restaurants (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)
```

- *DROP:*

```
DROP TABLE menu or DROP TABLE IF EXISTS menu;
```

- *ALTER*

```
ALTER TABLE menu ADD list_food text;
```

- *RENAME*

```
ALTER TABLE menu RENAME TO menus;
```

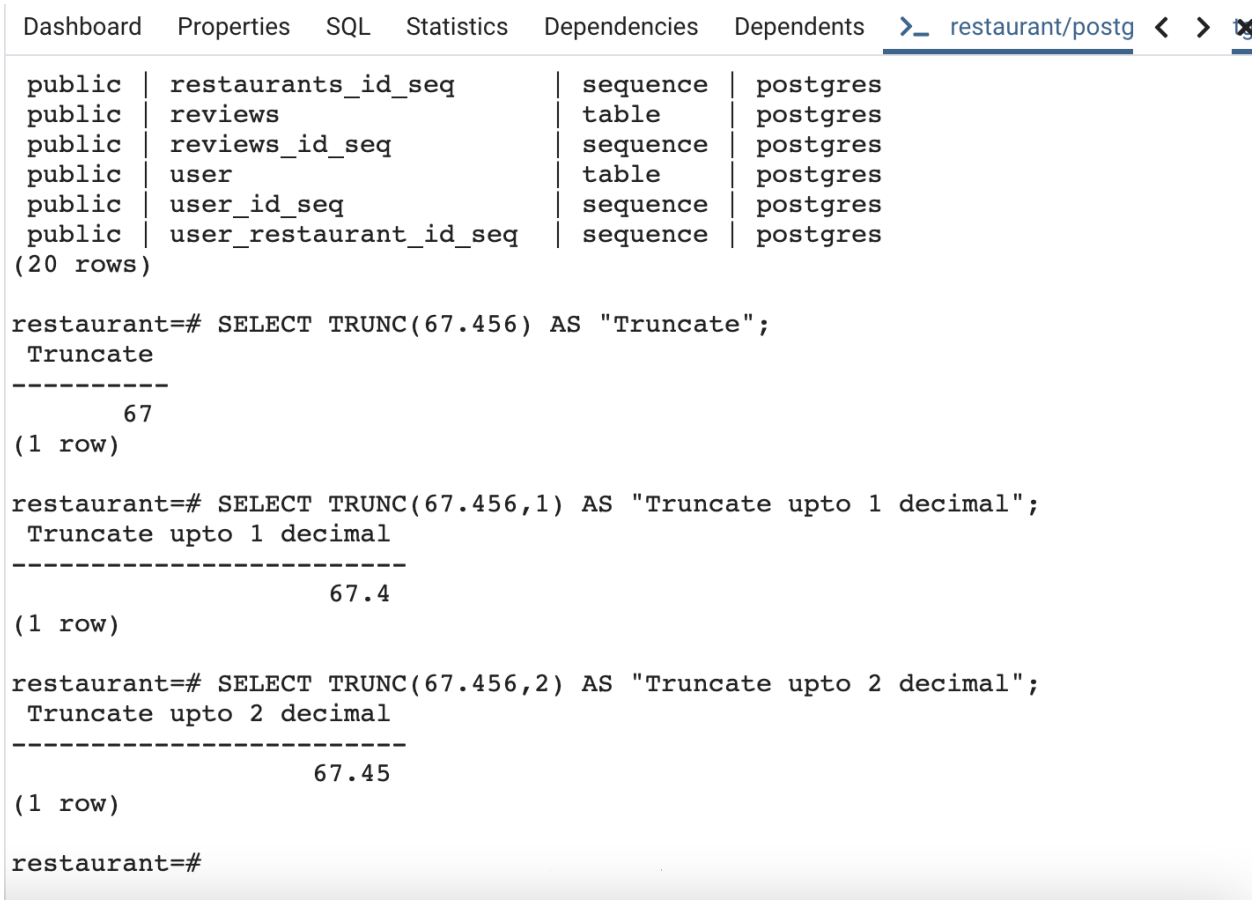
- *TRUNCATE*

The PostgreSQL trunc() function is used to truncate a number to a particular decimal place. If no decimal places are provided it truncates toward zero(0).

```
SELECT TRUNC(67.456) AS "Truncate";
```

- *COMMENT*

```
/* create the tables for the database */
```




# ● View DDL:

The DML (Data Manipulation Language) commands deal with the manipulation of data present in the database.
Example: SELECT, INSERT, UPDATE, DELETE.

- *SELECT*

SELECT id, list_food, unit_price, discount, restaurant_id, quantity
FROM public.menu;

- *INSERT*

INSERT INTO menu ( id,list_food, unit_price, discount,restaurant_id, quantity)
VALUES
( 111, Lunch special: Appetizers: Cesar Salad, Spring Salad
Entrees: Laghman
, Kurdak
,Plov
Desserts: Cheesecake
Beverages:Tea, Coffee', 10, 57, 2);

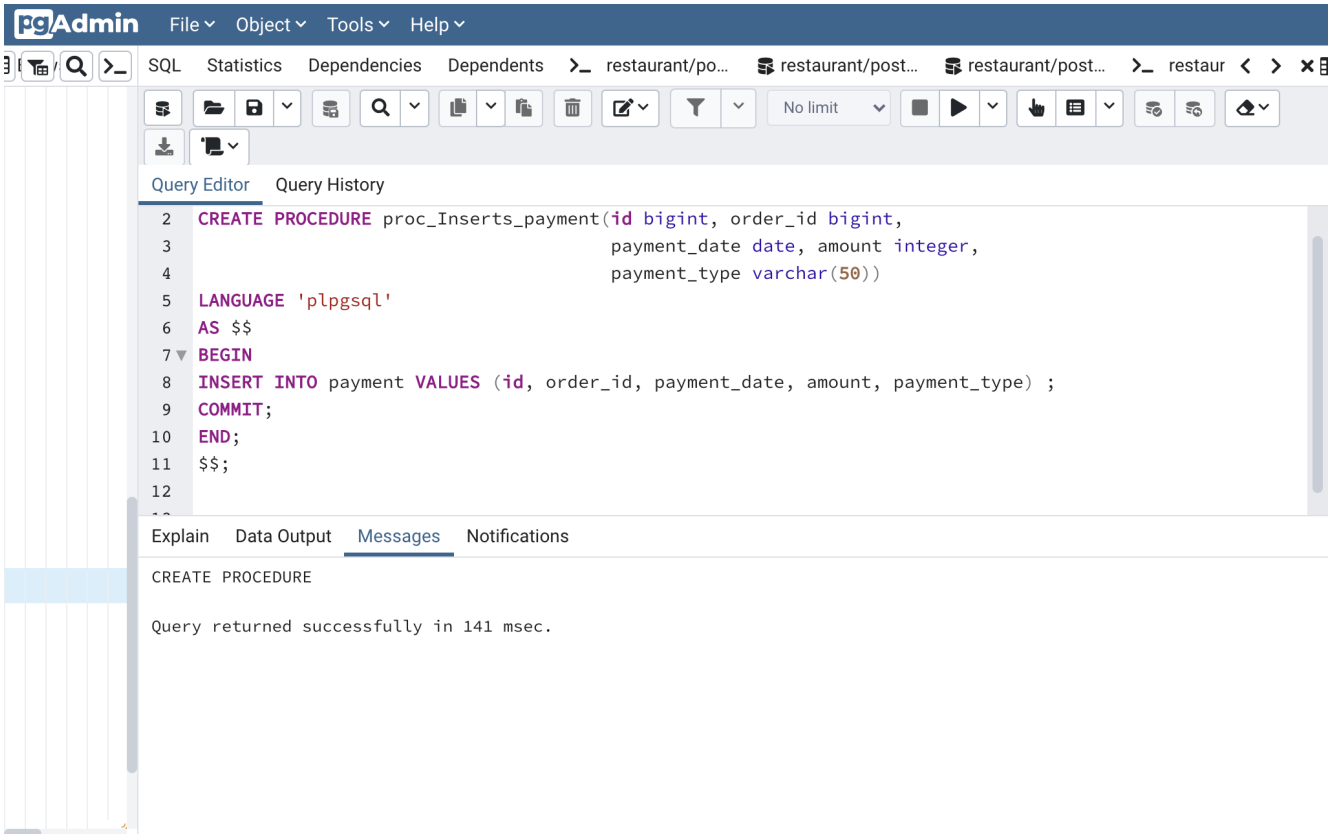
- *UPDATE*

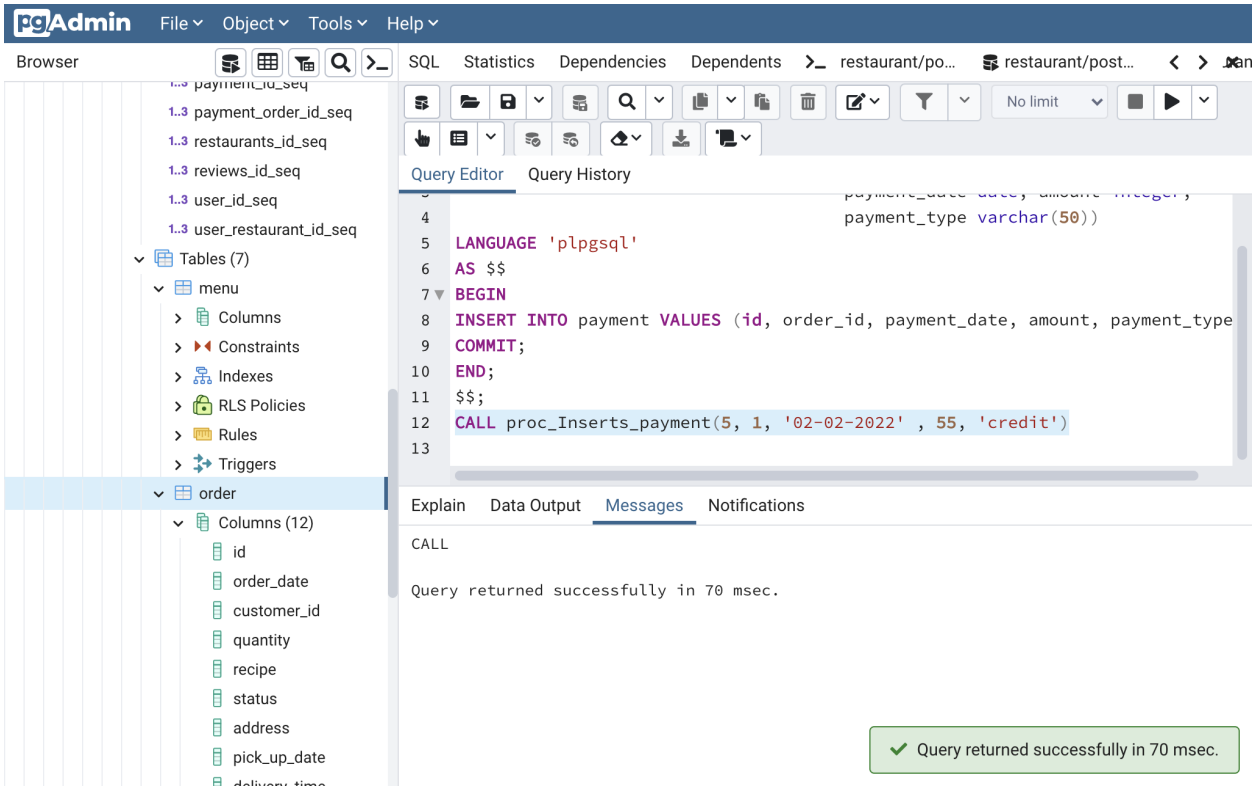
UPDATE public.menu
SET id=11, list_food='Lunch special: Appetizers: Cesar Salad, Spring Salad. Entrees: Laghman, Kurdak, Plov,Desserts: Cheesecake', unit_price=10, discount=0.0, restaurant_id=57, quantity=25
WHERE id=11;

- *DELETE*

DELETE FROM menu WHERE id = 11;

# ● Procedure DDL



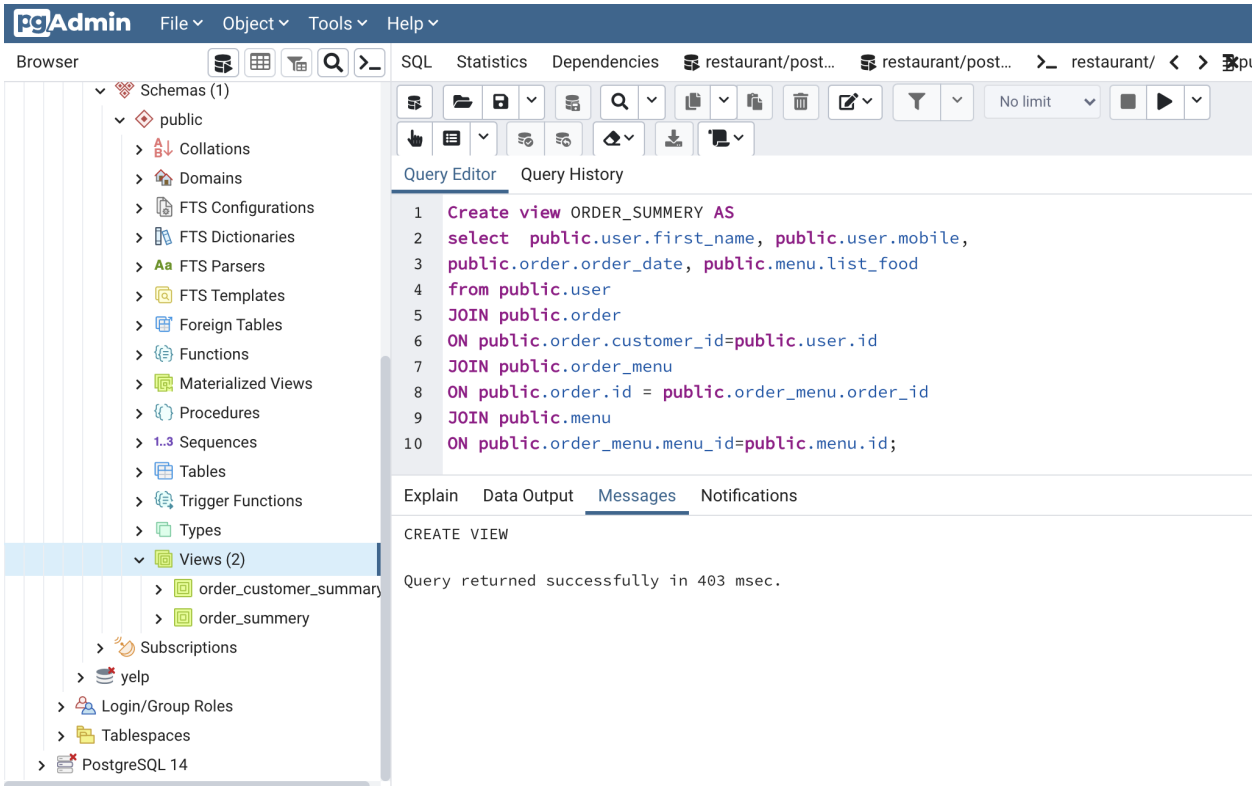



## • Function DDL

```
CREATE OR REPLACE FUNCTION increment(i integer) RETURNS integer AS $$
        BEGIN
                RETURN i + 1;
        END;
$$ LANGUAGE plpgsql;
```

## • Trigger DDL

```
CREATE TRIGGER payment_made_trigger
  AFTER INSERT
  ON payment
  FOR EACH ROW
  EXECUTE PROCEDURE payment_made();
INSERT INTO public.payment(
            order_id, payment_date, amount, payment_type)
           VALUES ( 3, '09-09-2019', 100, 'debit');
```




**Normalization**
My normalization satisfies 2F, 3F

Integrity can fall into many categories first one is I don't have any duplicate rows in any table. Second, I have all the tables appropriately reference data in other tables for foreign keys, so If I update one it would affect the other ones. For example, if I delete my customers, my order will be deleted. Orders can't exist without customers.

**Read Committed –** This isolation level guarantees that any data read is committed at the moment it is read. Thus it does not allow dirty reading. The transaction holds a read or write lock on the current row, and thus prevents other transactions from reading, updating, or deleting it.
Read Committed is the default isolation level in PostgreSQL. When a transaction runs on this isolation level, a SELECT query sees only data committed before the query began and never sees either uncommitted data or changes committed during query execution by concurrent transactions. Serializable is not required in my project.

**Source code:** https://github.com/aigOffline/DATABASE_SQL/tree/master

**Presentation video:** https://vimeo.com/707331325