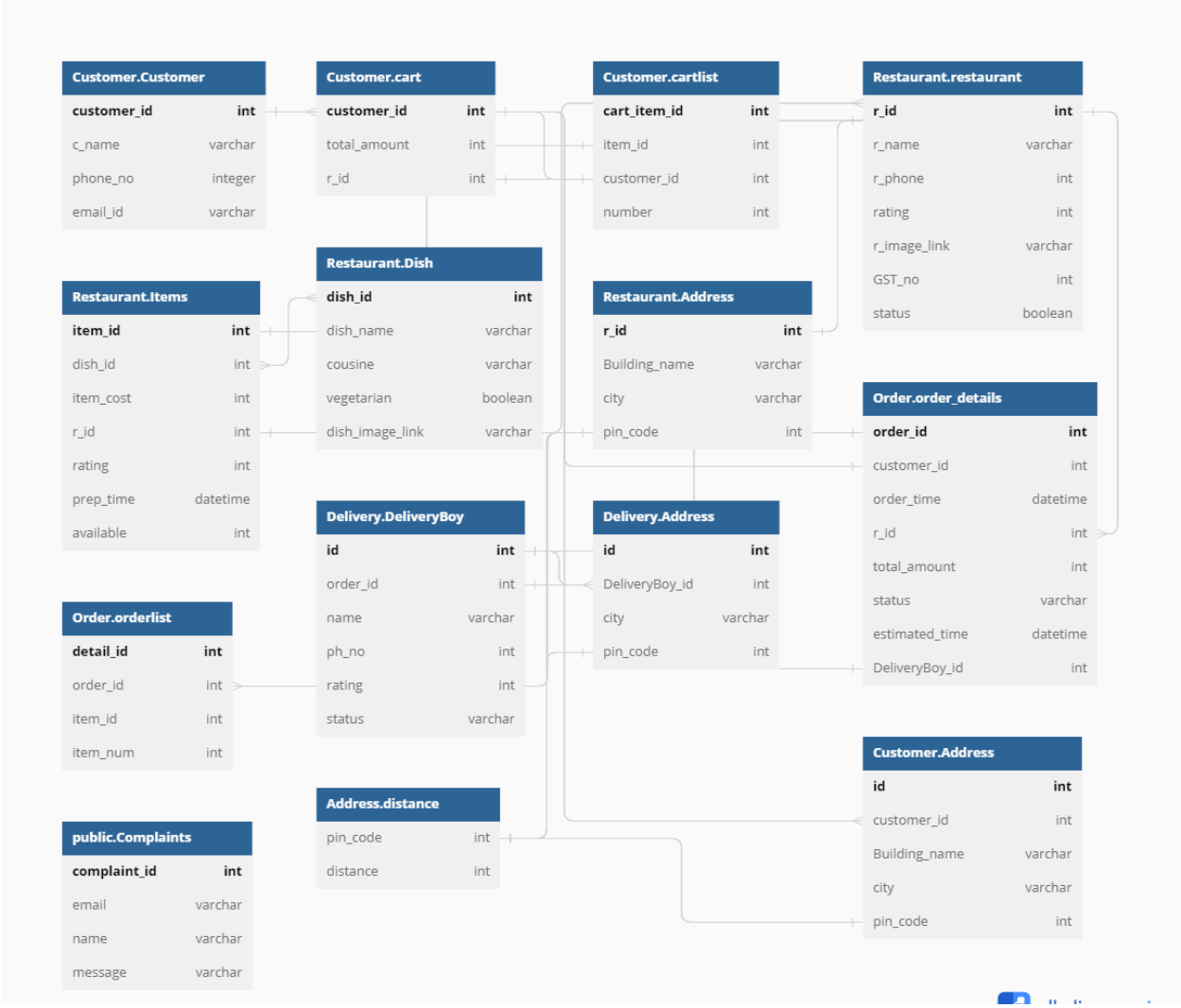
DBMS PROJECT FINAL REPORT

Food Delivery Website

112001019 Lekshmi R Nair
112001042 Sreelakshmi A
112001004 Aghnash Ranjith



Foreign key constraints on our table:

Key	Refers to
Order.order_details.DeliveryBoy_id	Delivery.DeliveryBoy.id
Restaurant.Address.pin_code	Address.distance.pin_code
Customer.Address.pin_code	Address.distance.pin_code
Delivery.Address.pin_code	Address.distance.pin_code
Customer.cart.r_id	Restaurant.restaurant.r_id
Customer.cartlist.item_id	Restaurant.Items.item_id

customer.cart.customer_id	Customer.Customer.customer_id
Delivery.Address.DeliveryBoy_id	Delivery.DeliveryBoy.id
Customer.Address.customer_id	Customer.Customer.customer_id
Order.orderlist.order_id	Order.order_details.order_id
Order.order_details.r_id	Restaurant.restaurant.r_id
Restaurant.Address.r_id	Restaurant.restaurant.r_id
Restaurant.Items.r_id	Restaurant.restaurant.r_id
Restaurant.Dish.dish_id	Restaurant.Items.dish_id
Delivery.DeliveryBoy.order_id	Order.order_details.order_id
Order.order_details.customer_id	Customer.Customer.customer_id
Customer.cartlist.customer_id	Customer.cart.customer_id

All primary keys are auto incremented

Other constraints:

- Customer.customer
  - Phone_no : NOT NULL
  - Name: NOT NULL
- Customer.cart:
  - Total_amount >= 0
  - Customer_id NOT NULL
- Customer.Address:
  - Pincode: NOT NULL
  - Buildingname: NOT NULL
- Customer.cart_list:
  - Number >= 0
  - Item_id NOT NULL

- Restaurant.restaurant:
    - Name NOT NULL
    - r_phone NOT NULL
    - GST_no UNIQUE, NOT NULL
- Restaurant.item:
    - Dish_id NOT NULL
    - item_cost>0
    - R_id NOT NULL
- Restaurant.dish:
    - Dish_name NOT NULL
- Restaurant,address:
    - Pincode: NOT NULL
    - Buildingname: NOT NULL
- Order.order_details:
    - total_amount>=0
    - Estimated time>0
    - Customer_id NOT NULL
    - Restaurant_id NOT NULL
- Order,order_list:
    - Order_id NOT NULL
    - Item_id NOT NULL
    - item_num>0
- Delivery,address:
    - Pincode: NOT NULL
    - Deliveryboy_id NOT NULL
- Delivery,DeliveryBoy:
    - Name NOT NULL
    - Phone_no NOT NULL

NOT NULL constraints are used in areas where the system will not work without those details, for example, without pin_code, all estimations of time will not be possible, since we use pincode for estimation of time.
CHECK constraints, wherever amount consistency has to be maintained.

FUNCTIONS:

```
CREATE FUNCTION find_delivery_time(pin1 integer, pin2 integer)
RETURNS INTEGER
language plpgsql
AS $$

DECLARE time_in_minutes integer;
BEGIN
SELECT  p1.time-p2.time INTO time_in_minutes
```

```
FROM distance as p1
JOIN distance as p2
      ON p2.pin_code = pin1 AND p1.pin_code=pin2;
RETURN time_in_minute;
END;
$$;
```

Explanation: Since we take in pincode, we use this to find time between two locations based on the pincode. We use it in adding estimated time to order.

LOGIN FUNCTIONS:::

```
CREATE FUNCTION loginCustomer(name varchar, id int )
RETURNS Boolean
language plpgsql
AS $$
BEGIN
if exists (select * from Customers where c_name=name and customer_id=id)
then
return 1;
else return 0;
end if;
END;
$$;
```

```
CREATE FUNCTION loginRest(name varchar, id int )
RETURNS Boolean
language plpgsql
AS $$
BEGIN
if exists (select * from restaurants where r_name=name and r_id=id)
then
return 1;
else return 0;
end if;
END;
$$;
```

```
CREATE FUNCTION loginDelivery(name varchar, id int )
RETURNS Boolean
language plpgsql
```

```
AS $$
BEGIN
if exists (select * from deliveryBoy where name=name and id=id)
then
return 1;
else return 0;
end if;
END;
$$;
```

Explanation: We have set a system that username is name and password is corresponding id. So as to facilitate checking and switching the roles in flask, which we have used as our backend, we are using these functions.

FUNCTION/PROCEDURE TRIGGER PAIRS

```
CREATE OR REPLACE FUNCTION create_customer_role() RETURNS TRIGGER
 LANGUAGE plpgsql
 AS $$
DECLARE
    role_name TEXT;
BEGIN
    -- Generate a role name using c_name and customer_id columns of the
inserted row
    role_name := NEW.c_name ||NEW.customer_id;

    -- Create a new role with the generated name
    EXECUTE format('CREATE ROLE %I LOGIN PASSWORD %L', role_name,
NEW.customer_id);

    -- Grant the 'customer' role to the newly created role
    EXECUTE format('GRANT customer TO %I', role_name);

    RETURN NEW;
END;
$$;
CREATE TRIGGER customer_insert_trigger
AFTER INSERT ON Customer
FOR EACH ROW
EXECUTE FUNCTION create_customer_role();
```

Explanation: Creation of roles whenever a user logs in, WHICH WILL BE USED IN A TRIGGER. Similarly there are for restaurant and delivery boy.

```
create or replace function insert_customer_address()
returns trigger
language plpgsql
as $$
begin
if EXISTS (SELECT 1 FROM distance WHERE new.pin_code = pin_code)
then
      return new ;
else
      raise exception 'PIN_CODE Does not exist';
end if ;
return new;
end;
$$;


create trigger trg_customer_address
before insert
on Customer_Address
for each row
execute function insert_customer_address();
```

Explanation: When inserting an address, checking if the pincode belongs to the area where the app delivers, that is, pincodes inside the distance table.

```
CREATE OR REPLACE FUNCTION create_restaurant_role() RETURNS TRIGGER
 LANGUAGE plpgsql
 AS $$
DECLARE
    role_name TEXT;
BEGIN
    -- Generate a role name using c_name and customer_id columns of the
inserted row
    role_name := NEW.r_name ||NEW.r_id;

    -- Create a new role with the generated name
    EXECUTE format('CREATE ROLE %I LOGIN PASSWORD %L', role_name,
NEW.r_id);

    -- Grant the 'customer' role to the newly created role
```

```
    EXECUTE format('GRANT restaurant TO %I', role_name);

    RETURN NEW;
END;
$$;
CREATE TRIGGER restaurant_role_trigger
AFTER INSERT ON Customer
FOR EACH ROW
EXECUTE FUNCTION create_restaurant_role();
```

Explanation: When registering, creating a role and granting required privileges.

```
CREATE OR REPLACE FUNCTION create_delivery_role() RETURNS TRIGGER
 LANGUAGE plpgsql
 AS $$
DECLARE
    role_name TEXT;
BEGIN
        role_name := NEW.name ||NEW.id;

    -- Create a new role with the generated name
    EXECUTE format('CREATE ROLE %I LOGIN PASSWORD %L', role_name, NEW.id);

    -- Grant the 'customer' role to the newly created role
    EXECUTE format('GRANT delivery TO %I', role_name);

    RETURN NEW;
END;
$$;

CREATE TRIGGER delivery_role_trigger
AFTER INSERT ON Customer
FOR EACH ROW
EXECUTE FUNCTION create_delivery_role();
```

Explanation: Similarly creating role for delivery boy

PROCEDURES:

```
create or replace procedure make_order(
   cart_id1 int,
   item_id1 int,
number1 int)
language plpgsql
as $$
begin
    -- adding a new order to Order.order
INSERT INTO Customer_cartlist(item_id,cart_id,number) VALUES
      (item_id1,cart_id1,number1);
    commit;
end;$$
```

Explanation : Whenever we add something to cart, we add it to cartlist. This helps us main

```
CREATE OR REPLACE PROCEDURE insert_customer(
  p_customer_id INT,
  p_c_name VARCHAR,
  p_phone_no INTEGER,
  p_email_id VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO Customer (customer_id, c_name, phone_no, email_id)
  VALUES (p_customer_id, p_c_name, p_phone_no, p_email_id);
END;
$$;
```

Explanation: We can directly call this procedure to add a customer, given inputs from the login form.

```
CREATE OR REPLACE PROCEDURE insert_customer_address(
  p_address_id INT,
  p_customer_id INT,
  p_building_name VARCHAR,
  p_locality VARCHAR,
  p_city VARCHAR,
  p_landmark VARCHAR,
  p_pin_code INT
)
LANGUAGE plpgsql
```

```
AS $$
BEGIN
  INSERT INTO Customer_Address (address_id, customer_id, Building_name,
locality, city, landmark, pin_code)
  VALUES (p_address_id, p_customer_id, p_building_name, p_locality, p_city,
p_landmark, p_pin_code);
END;
$$;
```

Explanation: Add address to customer, taking in input from the form.

```
CREATE OR REPLACE PROCEDURE shift_cart_to_orderlist(customer_id INT)
LANGUAGE plpgsql
AS $$
DECLARE
  total_amount INT;
  order_id INT;
  d_id INT;
  r_id INT;
BEGIN
  -- Get the total amount of the customer's cart
  SELECT total_amount INTO total_amount
  FROM Customer_cart
  WHERE customer_id = $1;

  -- Generate a new order_id
  SELECT COALESCE(MAX(order_id), 0) + 1 INTO order_id
  FROM Order_details;

  -- Assign Delivery Boy
  SELECT id INTO d_id FROM deliveryBoy where status = 'free' LIMIT 1;

  -- Get the r_id of the restaurant from customer's cart
  SELECT r_id INTO r_id
  FROM Customer_cart
  WHERE customer_id = $1;

  -- Insert into Order_details
  INSERT INTO Order_details (order_id, customer_id, order_time, r_id,
deliveryBoy_id, total_amount, status, estimated_time)
  VALUES (order_id, customer_id, NOW(), r_id, d_id, total_amount,
'processing', NULL);
```

```
  -- Insert into Orderlist
  INSERT INTO Orderlist (detail_id, order_id, item_id, number)
  SELECT cart_item_id, order_id, item_id, number
  FROM Customer_cartlist
  WHERE customer_id = $1;

  -- Delete from Customer_cartlist
  DELETE FROM Customer_cartlist
  WHERE customer_id = $1;

  -- Delete from Customer_cart
  DELETE FROM Customer_cart
  WHERE customer_id = $1;
END;
$$;
```

Explanation: When clicked order, items in cartlist are added to order, correspondingly cart is cleared. This is to maintain the integrity of our ordering model, once we click order, elements are shifted to order and this table can be correspondingly viewed by restaurants to prepare order.

ALL ROLES:

- **Proj_admin :** All privileges on all tables, CREATEROLE, and ownership of our database is given.
- **Customer:** This is the role allotted to all customers: ALL ACCESS to customer_address, customer_cart, customer_cartlist, access to the order_details_view and customer_view(Described below)
- **Deliveryboy:** This role is allotted to all delivery boy, delivery_boy_address and a view, order_address_view (Described below) is given select access.
- **Restaurants:** This role is for all restaurants, here we have all access to resturant_address, and select privilege on order_list. Also has ALL ACCESS on restaurants_items. SELECT PRIVILEGE on restaurant_dish. SELECT access to View order_details_restaurant_view.

VIEWS:

```
CREATE VIEW order_details_view AS
SELECT od.order_id, c.customer_id, od.order_time, r.r_name, db.name,
od.total_amount, od.status, od.estimated_time
FROM order_details od
JOIN Restaurants r ON od.r_id = r.r_id
JOIN Customer c ON od.customer_id = c.customer_id
JOIN deliveryBoy db ON od.deliveryBoy_id = db.id;
```

Explanation: This is available for customers to view. Sensitive details which can be used for logging in, like deiveryboy_id, restaurant_id etc are hidden from the user, along with details that don't belong to this customer.

```
CREATE VIEW Customer_View AS
SELECT
  Restaurants.r_name,
  Restaurants.r_phone,
  Restaurants.rating,
  Restaurants.r_image_link,
  Restaurant_Dish.dish_name,
  Restaurant_Dish.cousine,
  Restaurant_Dish.vegetarian,
  Restaurant_Dish.dish_image_link,
  Restaurant_Items.item_cost,
  Restaurant_Items.rating AS item_rating,
  Restaurant_Items.prep_time,
  Restaurant_Items.available,

FROM Restaurants
JOIN Restaurant_Items ON Restaurants.r_id = Restaurant_Items.r_id
JOIN Restaurant_Dish ON Restaurant_Items.dish_id = Restaurant_Dish.dish_id;
```

Explanation: When we are expanding to see the items orderable, we don't need to view unnecessary details like address, r_id etc. So we created a view for easier access.

```
CREATE VIEW order_details_restaurant_view AS
SELECT
```

```
    od.order_id,
    r.r_name AS restaurant_name,
    db.name AS deliveryboy_name,
    od.order_time,
    od.status,
    ol.item_id,
    ol.item_num
FROM
    order_details od
    JOIN Restaurants r ON od.r_id = r.r_id
    JOIN Deliveryboy db ON od.deliveryBoy_id = db.id
    JOIN Orderlist ol ON od.order_id = ol.order_id;
```

Explanation: This is for restaurants to view orders, they don't need customer_id, delivery_boy_id etc(Deliveryboy can verify identity by just showing their website page) . We are joining several tables so as to see what to prepare for the restaurant. Background details like total amount are hidden, since it's under the discretion of the app and agreement made for the payment.

```
CREATE VIEW order_address_view AS
SELECT
od.order_id,
  ra.Building_name AS from_Building_name,
  ra.city AS from_city,
  ra.pin_code AS from_pin_code,
  ca.Building_name AS to_Building_name,
  ca.city AS to_city,
  ca.pin_code AS to_pin_code,
  od.deliveryBoy_id AS deliveryboy_id
FROM
  order_details od
  JOIN Restaurant_Address ra ON od.r_id = ra.r_id
  JOIN Customer_Address ca ON od.customer_id = ca.customer_id;
```

Explanation: Deliveryboy just needs to see from where they are picking up the order, and where they are dropping it off. No other details like sensitive information like customer_id, restaurant_id, no additional details like item list etc. are shown. This also makes it easier for deliveryboy to observe their required data, without being confused about the background fluff.

INDEXING:

On restaurants:

```
CREATE INDEX restaurant_rating ON Restaurant(rating);
```

This is a non clustered index, particularly useful since we are displaying top 5 restaurants. In this instances a Btree index will be used since Of the index types currently supported by PostgreSQL, only B-tree can produce sorted output — the other index types return matching rows in an unspecified, implementation-dependent order

On deliveryboy:

```
CREATE INDEX deliveryboy_status ON Deliveryboy (status);
```

We have used a BTree index, initially we wanted to use hash index, since in theory, there is equality matching, where we are finding delivery boys with status free. But since there are many collisions possible, ie more than one deliveryboy with same status, we use BTree index

On order_details:

```
CREATE INDEX order_details_order_time ON order_details (order_time);
cluster order_details using order_Details_order_time;
```

Since queries often only need to access a subset of columns, columnstore indexes can reduce the amount of data that needs to be read from disk, improving query performance. We need this since are displaying the sorted past orders of customer

Queries

SELECT * FROM RESTAURANTS
ORDER BY rating
LIMIT 5;

SELECT * from deliveryboy
WHERE status = ‘FREE’;

SELECT * FROM order_details
ORDER BY order_time;