CS6360: DB Design Project

# DOOR-DASH SYSTEM

**Section: CS6360.004.S22**
**Team: doordash-1**
**Team 16**

Vignesh (VXT210003)
Karneeshwar (KXS200001)
Sreeja (SXG200037)

## Table of Contents

# Door Dash Introduction

It’s an online platform, which enables customers to order food from their preferred restaurants enlisted on the application. Once the order is accepted and prepared by the restaurant a door-dash driver delivers the food to the customer as soon as possible. For this service, door dash charges a small service fee and a part of it is shared to the driver as well for his service. As in normal restaurants, customers can also tip which goes directly to the driver’s account.

This model lets the customers to order food from any part they are in (wherever door dash provides service). Also, from time to time, they will be provided with coupon codes which they can use to get discount during subsequent orders.

Since there are lots of entities involved which are related to one another, a Relational DB design will be best suited. The current project is done with Oracle 19c (Oracle Cloud) and currently no performance tuning (like indexing, partitioning etc. are done)

## Functional Requirements:

As part of the model, we have identified the below entities

- Customer
- Driver
- Restaurant
- Orders
- Ticket
- Customer Care
- Promo Code
- Driver Bank Details
- Payment Details
- Restaurant Items

### Person:

They are the main entity, which separate into Customers & Drivers. A person logs on to the portal using either email or phone.

- Unique person id
- Email id
- phone number
- Name (first name, second name)
- Store the hash of login_password

***NOTE: A Customer cannot be a driver and vice-versa***

## Customer:

Customers are the people, who place orders (initiates the food ordering, and process). Customers are identified by unique_id, and register using email_id and phone number.

A Customers should be able to:

- store different payment methods - like PayPal, credit/debit card, apple pay / venom.
- receive time and money bound promo codes, which can be during orders to avail discounts
- can choose a variety of items (from the restaurant menu) from a single restaurant, and order to a single address. Also, should be able to provide special instructions regarding the order
- provide review & rating to any restaurant, as well mark multiple restaurants as their favorites
- Provide rating to a driver for his/her service on an order
- A customer who is a doordash_pass holder will have 0$ delivery fee.
- Raise ticket with customer care on any issues on the portal.

## Drivers

Drivers are sub-class of Person, and inherit the attributes of person, and additionally they have custom attributes for records purposes

Attributes of driver entity are:

- driver_id
- SSN (for background verification)
- Date_of_birth
- Joining date
- Is _active
- Description of him (customized by driver to be displayed on his / her profile)
- Gender
- Unique home address (for records purposes)
- Bank details - to transfer the amount, tips etc.
- Vehicle Details in which he will be delivering (car details like color and type/model, license number, plate number, insurance number)

Drivers should be able to:

- Accept an order and should deliver the food to the customer.
- Driver receives a share of 12.5% on every order as their fees.
- Customer tips on a specific order will totally be transferred to his / her account.

## Restaurants:

Entities which prepare the food ordered by the customer, on the requested quantity.
Restaurant’s attributes not limited to:

- Name
- Restaurant_id (auto generated by system)
- Restaurant type (cuisine)

- Contact details: Email id & Phone number
- Location of the restaurant (address)
- Opening & closing time of the restaurant
- Restaurant Owner / Manager details - with attributes as SSN, name, and bank details

A restaurant should be able to serve a variety of food, which the customers can choose from during delivery.
Menu items will have the following attributes: (menu_item is a weak entity, identified by Restaurant)

- Item_id and restaurant_id will be used as primary_key
- Item_name
- Calories
- Description & category (veg/non-veg/vegan)
- Price

## Order:

Order is created, once the customer places the food items and pays the total cost (including tax, driver_tips, delivery charge etc.)

The order relation will have certain critical attributes:

- Order status
- Address of delivery
- Total cost and cost of individual item
- And special instructions regarding the order

## Ticket:

When a customer has issue on the portal, he / she raises it to the customer care who will attend the ticket and try to resolve it. This is being tracked by a ticket for reference purposes.
It's identified by:

- Ticket no (autogenerated pkey)
- Status
- Description of the issue

## Customer Care:

They are employees of Door Dash, who will attend the ticket and try to resolve it, identified by:

- Emp_id (autogenerated pkey)
- SSN
- Salary
- Joining date

## Payments:

A customer can have N payment options, each can be of any of the below as Credit Card or Apple pay or paypal

Payment:

- Added_on

Credit Card:

- Cc no
- Cc name
- Cvv
- Expiry date

Paypal:

- Paypal id

Apple pay:

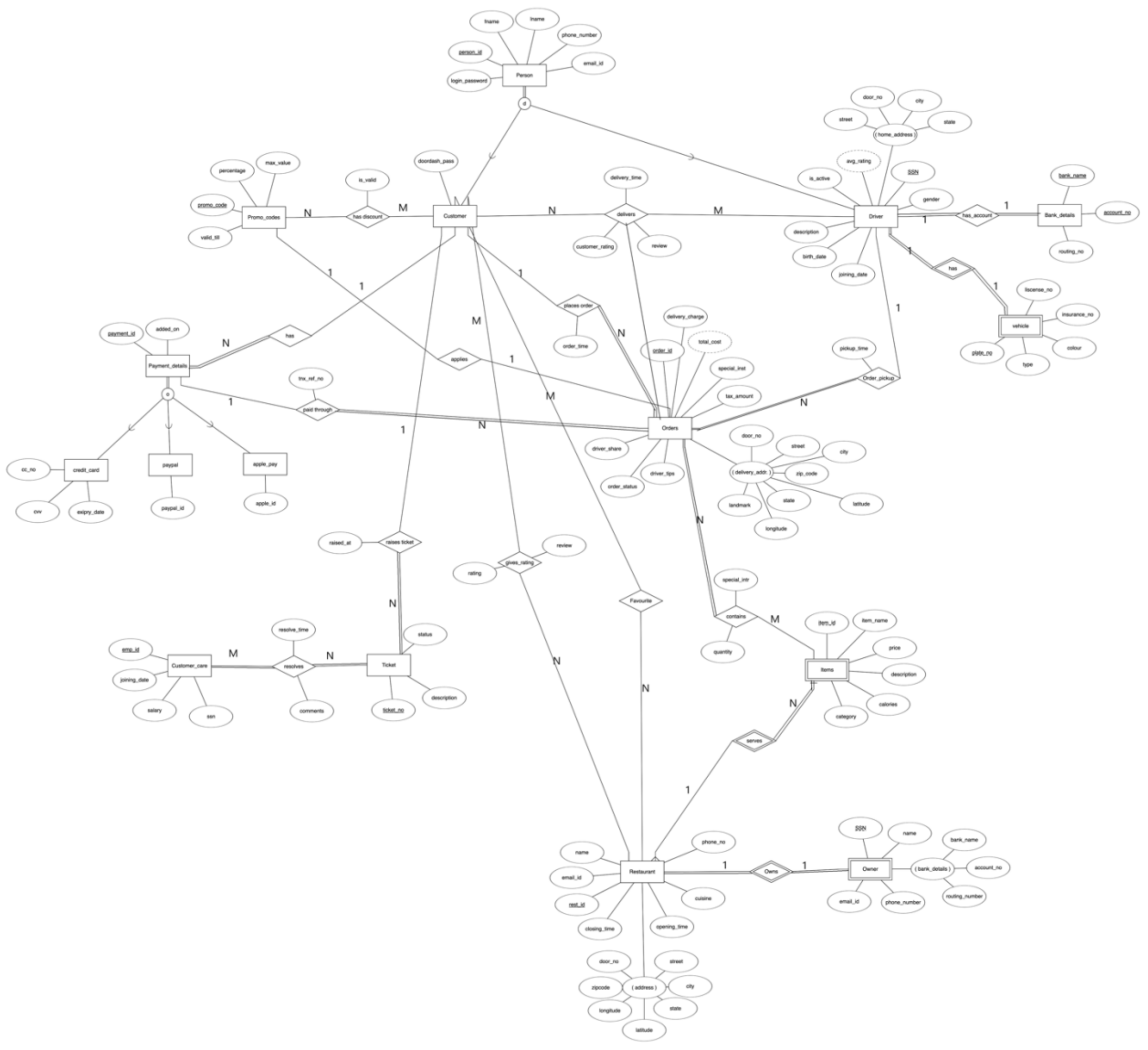
- Applepay id

## Assumptions:

In the mini world we have created,

- The delivery is within US States and Zip Codes
- All payments are accepted in $ (driver tips, customer payment) and certainly not in BTC / XBT.

# ER Diagram


fname
lname
phone_number
person_id
email_id
login_password
Person
d
door_no
city
street
state
( home_address )
avg_rating
SSN
is_active
gender
Driver
has_account
Bank_details
bank_name
account_no
routing_no
description
birth_date
joining_date
has
vehicle
license_no
insurance_no
colour
type
plate_no
percentage
max_value
promo_code
valid_till
Promo_codes
is_valid
has discount
doonbash_pass
Customer
delivery_time
delivers
customer_rating
review
places order
order_time
applies
delivery_charge
order_id
total_cost
special_inst
tax_amount
pickup_time
Order_pickup
payment_id
added_on
Payment_details
has
o
trx_ref_no
paid through
Orders
driver_share
driver_tips
order_status
( delivery_addr. )
zip_code
landmark
longitude
latitude
cc_no
credit_card
cvv
expiry_date
paypal
paypal_id
apple_pay
apple_id
raised_at
raises ticket
rating
gives_rating
Favourite
special_intr
contains
quantity
item_id
item_name
price
Items
calories
category
emp_id
Customer_care
joining_date
salary
ssn
resolve_time
resolves
comments
Ticket
status
ticket_no
serves
name
phone_no
email_id
rest_id
Restaurant
cuisine
closing_time
opening_time
Owns
Owner
bank_name
( bank_details )
routing_number
phone_number
( address )
zipcode
1
M
N

## Relations & Cardinality

Here are the list of relations and their cardinality

Customer_has_discount: Many-to-Many
The same promo code can be provided to multiple customers and a single customer can have multiple different offers

Customer_has_PaymentMethods: One-to-Many
One customer can have multiple payment options – like n Credit cards, n paypal accounts etc. But, the same credit card can't be used by others – for security reasons

Customer_Places_Order: One-to-Many
A customer can place N orders, but an order can't be related to multiple customers

Customer_raises_ticket: One-to-Many
A customer can raise N tickets, but a single Ticket can't be related to multiple customers

Customer_reviews_restaurant: Many-to-Many
A Customer can review N restaurants and a restaurant can be reviewed by M customers

Customer_marks_restaurant_favourite: Many-to-Many
A Customer can mark N restaurants as favorite and a restaurant can be marked by M other customers.

Driver_Picksup_Order: One-to-Many
A Driver can pick up N orders, but an order can't be picked up by multiple drivers

Driver_Delivers_Order: Many-to-Many
A Driver can deliver to N Customers, and a customer can receive order from M Drivers. To identify the uniqueness in each delivery, this relation is a ternary with order_id, ensuring a same driver can deliver to the same customer but for different orders

Driver_has_vehicle: One-to-One
Driver can have a vehicle, which is mapped by the Number Plate to him.

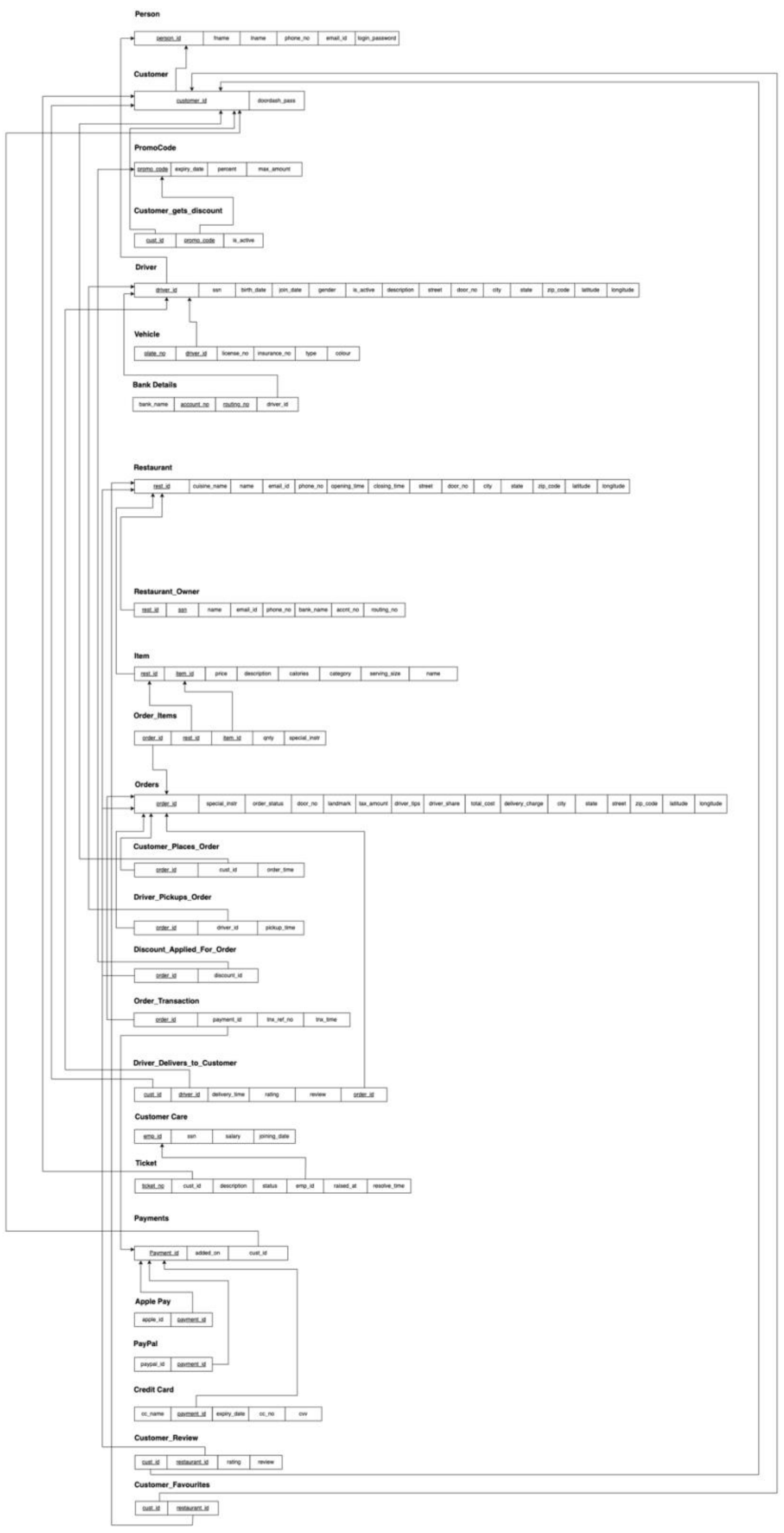
Driver_has_bankdetails: One-to-One
A driver can have a single bank account.

Restaurant_has_Owner: One-to-One
A Owner can own only one restaurant.

Restaurant_serves_items: One-to-Many
A Restaurant can serve N items, but a single item can’t be served by N restaurants

Order_has_items: Many-to-Many
An item can be ordered by N orders and a single order can contain N different items of different quantity

## ER to Relational Schema Mapping


Person
person_id | fname | lname | phone_no | email_id | login_password
Customer
customer_id | doordash_pass
PromoCode
promo_code | expiry_date | percent | max_amount
Customer_gets_discount
cust_id | promo_code | is_active
Driver
driver_id | ssn | birth_date | join_date | gender | is_active | description | street | door_no | city | state | zip_code | latitude | longitude
Vehicle
plate_no | driver_id | license_no | insurance_no | type | colour
Bank Details
bank_name | account_no | routing_no | driver_id
Restaurant
rest_id | cuisine_name | name | email_id | phone_no | opening_time | closing_time | street | door_no | city | state | zip_code | latitude | longitude
Restaurant_Owner
rest_id | ssn | name | email_id | phone_no | bank_name | accnt_no | routing_no
Item
rest_id | item_id | price | description | calories | category | serving_size | name
Order_Items
order_id | rest_id | item_id | qnty | special_instr
Orders
order_id | special_instr | order_status | door_no | landmark | tax_amount | driver_tips | driver_share | total_cost | delivery_charge | city | state | street | zip_code | latitude | longitude
Customer_Places_Order
order_id | cust_id | order_time
Driver_Pickups_Order
order_id | driver_id | pickup_time
Discount_Applied_For_Order
order_id | discount_id
Order_Transaction
order_id | payment_id | trx_ref_no | trx_time
Driver_Delivers_to_Customer
cust_id | driver_id | delivery_time | rating | review | order_id
Customer Care
emp_id | ssn | salary | joining_date
Ticket
ticket_no | cust_id | description | status | emp_id | raised_at | resolve_time
Payments
Payment_id | added_on | cust_id
Apple Pay
apple_id | payment_id
PayPal
paypal_id | payment_id
Credit Card
cc_name | payment_id | expiry_date | cc_no | cvv
Customer_Review
cust_id | restaurant_id | rating | review
Customer_Favourites
cust_id | restaurant_id

## Dependencies

Person_id -> fname, lname, phone_no, email_id, login_pwd

Customer_id -> doordash_pass

Driver_id -> ssn, birth_date, join_date, gender, is_active, description, street, door_no, city, state, zip_code, latitude, longitude

Promo_code -> expiry, percent, max_amount

cust_id, promo_code -> is_active

payment_id -> added_on, cust_id

payment_id -> apple_id

payment_id -> paypal_id

payment_id -> cc_name, expiry_date, cc_no, cvv

cust_id, restaurant_id -> rating, review

cust_id, restaurant_id

Plate_no, driver_id -> license_no, insurance_no, type, colour

Rest_id -> cuisine_name, rest_name, email_id, phone_no, opening_time, closing_time, street, door_no, city, state, zip_code, latitude, longitude

Rest_id, ssn - > owner_name, email_id, phone_no, bank_name, accnt_no, routing_no

Rest_id, item_id -> item_name, price, description, calories, category, serving_size

Order_id -> specorder_status, street, door_no, city, state, zip_code, latitude, longitude, landmark, special_instructions, tax_amount, driver_tips, driver_share, total_cost, delivery_charge

Order_id -> customer_id, order_time

Order_id -> driver_id, pickup_time

Order_id -> discount_id

Order_id -> oayment_id, tnx_ref_no, tnx_time

Cust_id, driver_id, Order_id -> delivery_time, rating, review

Emp_id -> ssn, salary, joining_date

Ticket_no -> cust_id, description, status, cust_id, raised_at

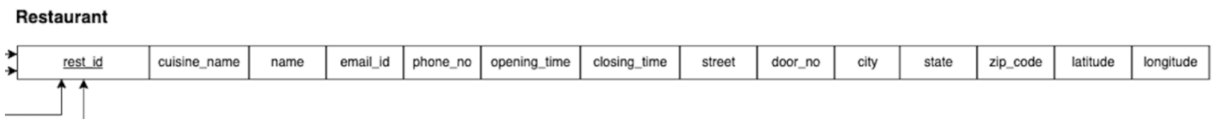
emp_id, ticket_no -> resolve_time, comments

## Normalization

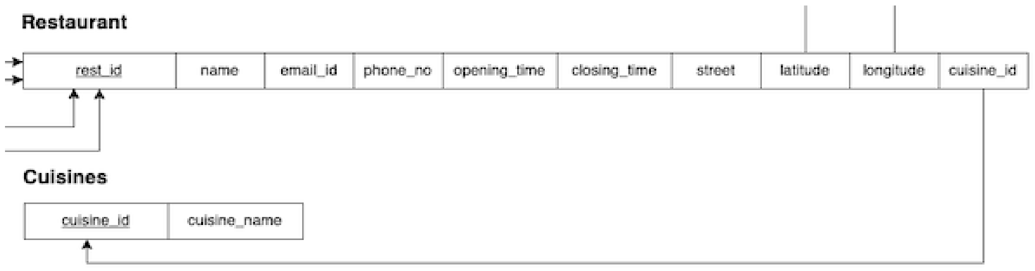
All tables, are already in 1NF form, whereby there exists no table column values which are multivalued, or nested values.

In Restaurant Cuisine, we can further relate the cuisine with a cuisine table where the values are stored, and the id alone is mapped to the Restaurant table. This can avoid repeated values as well save space.

Before Normalization



After Normalization



Additionally, we clearly see that the address zones can be normalized to avoid redundancy.

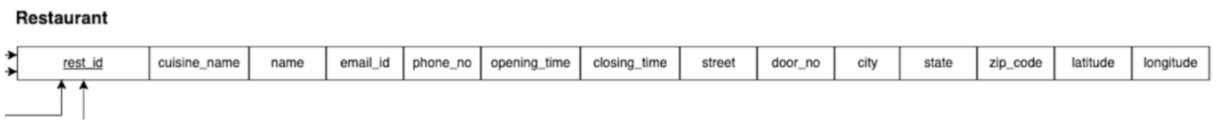
**Latitude and longitude will give the precise address, except the door no / apartment no.**

So, Latitude and longitude can be moved to another table, which will have street name, city, state & zip code.

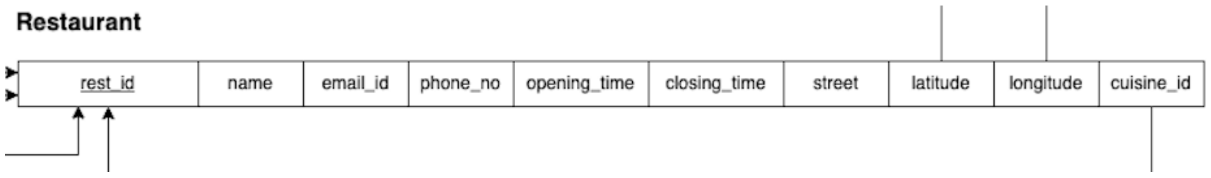
**Also, Zip codes give mapping to City and State, which can be moved to ZIPCODE_LOOKUP table.**

Thereby we have reduced the transitive dependency on locations. And this location tables can be used commonly for all addresses like restaurant, customer delivery address and driver permanent address – there by reducing space for storage.

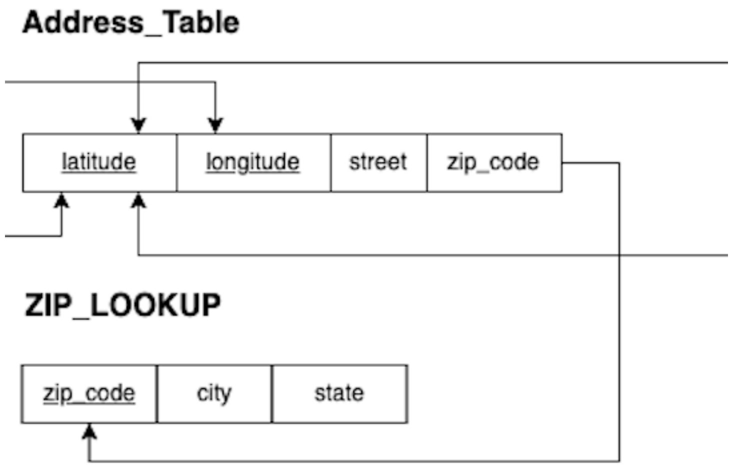
Before Normalization:



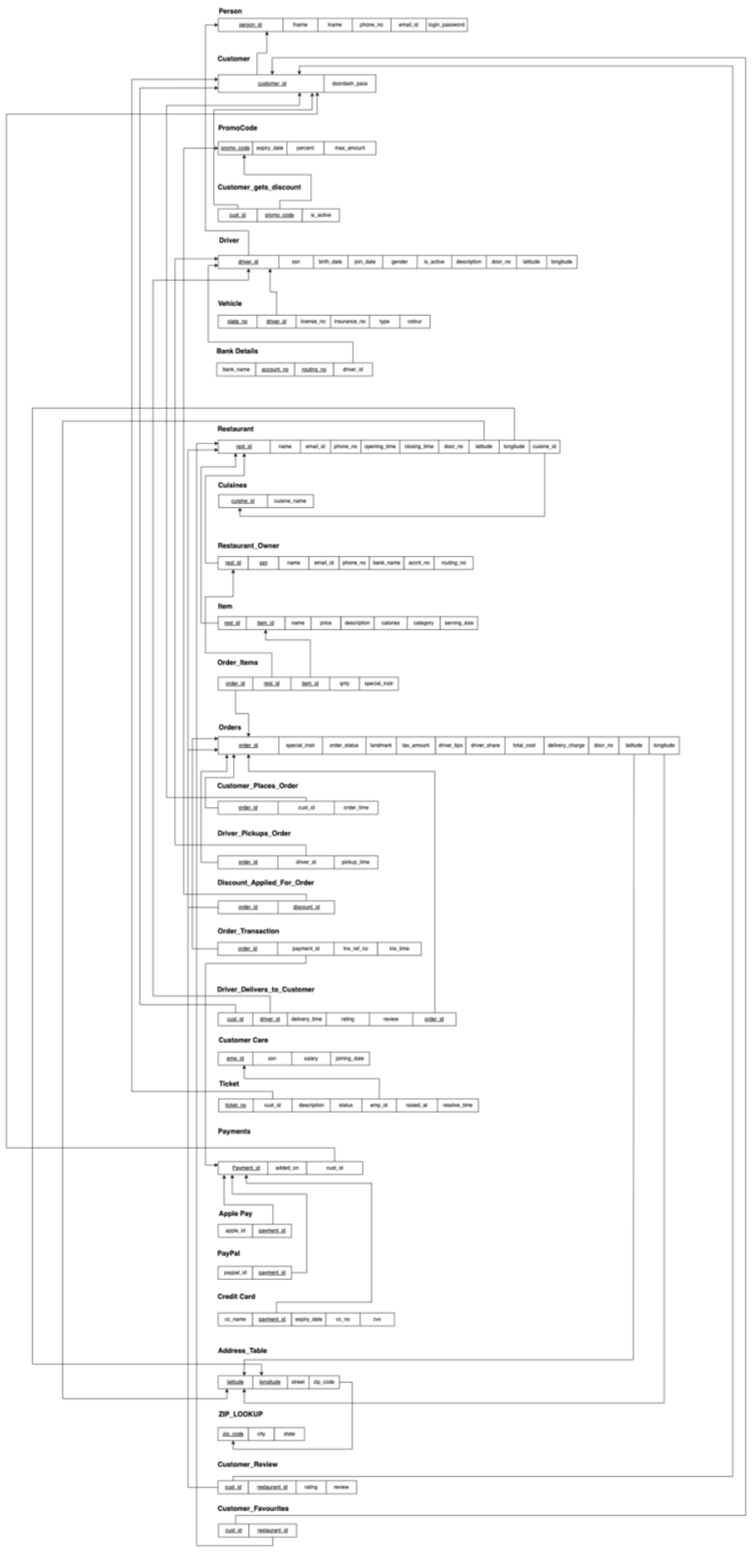
After Normalization:



Address table and Zip_lookup table



## Final Relational Schema Mapping after Normalization



# SQL

We start by creating DDL Statements, ie. Create tables. All the below tables have primary key, foreign key and default values assigned wherever necessary. As well, the columns are marked non null if mandatory.

## DDL / Table Creation Statements

```
CREATE TABLE DD_PERSON(
   PERSON_ID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
   ,FNAME VARCHAR(100) NOT NULL
   ,LNAME VARCHAR(100) NOT NULL
   ,PHONE_NUMBER NUMBER(13) NOT NULL
   ,EMAIL_ID VARCHAR(100) NOT NULL
   ,LOGIN_PWD VARCHAR(200) NOT NULL
   ,PRIMARY KEY (PERSON_ID)
);

CREATE TABLE DD_CUSTOMER (
   CUSTOMER_ID NUMBER
   , DOORDASH_PASS CHAR(1) DEFAULT 'N'
  , PRIMARY KEY (CUSTOMER_ID)
  , FOREIGN KEY (CUSTOMER_ID) REFERENCES DD_PERSON (PERSON_ID) ON DELETE CASCADE
);

CREATE TABLE DD_ZIPCODE_LOOKUP (
   ZIP_CODE NUMBER(10)
   ,CITY VARCHAR(50) NOT NULL
   ,STATE VARCHAR(30) NOT NULL
   , PRIMARY KEY (ZIP_CODE)
);

CREATE TABLE DD_ADDRESS_LOOKUP (
   LATITUDE DECIMAL(8,6)
   ,LONGITUDE DECIMAL(9,6)
   ,STREET VARCHAR(100) NOT NULL
   ,ZIP_CODE NUMBER(10) NOT NULL
   , PRIMARY KEY (LATITUDE, LONGITUDE)
   , FOREIGN KEY (ZIP_CODE) REFERENCES DD_ZIPCODE_LOOKUP (ZIP_CODE)
);

CREATE TABLE DD_DRIVER (
   DRIVER_ID NUMBER
   , SSN CHAR(11) NOT NULL UNIQUE
   , GENDER CHAR(2) NOT NULL
   , DATE_OF_BIRTH DATE NOT NULL
   , DATE_OF_JOIN DATE DEFAULT SYSDATE
   , IS_ACTIVE CHAR(1) DEFAULT 'Y' NOT NULL
   , DESCRIPTION VARCHAR(200) -- QUICK FUN DESCRIPTION ABT HIM / HER
   , DOOR_NO VARCHAR(100)
   , LATITUDE DECIMAL(8,6) NOT NULL
   , LONGITUDE DECIMAL(9,6) NOT NULL
   , PRIMARY KEY (DRIVER_ID)
   , FOREIGN KEY (DRIVER_ID) REFERENCES DD_PERSON (PERSON_ID) ON DELETE CASCADE
   , FOREIGN KEY (LATITUDE, LONGITUDE) REFERENCES DD_ADDRESS_LOOKUP (LATITUDE, LONGITUDE) ON DELETE
CASCADE
);


CREATE TABLE DD_PROMO_CODE (
   PROMO_CODE VARCHAR(20)
   , EXPIRY_DATE DATE  DEFAULT (SYSDATE + 1)
   , PERCENT DECIMAL(10, 2) NOT NULL
   , MAX_AMOUNT NUMBER
   , PRIMARY KEY (PROMO_CODE)
);

-- dependent on DD_promo_code and DD_CUSTOMER
CREATE TABLE DD_CUSTOMER_GETS_OFFER (
   PROMO_CODE VARCHAR(20)
   , CUSTOMER_ID NUMBER NOT NULL
   , IS_ACTIVE CHAR(1) DEFAULT 'Y'
   , PRIMARY KEY (PROMO_CODE, CUSTOMER_ID)
   , FOREIGN KEY (PROMO_CODE) REFERENCES DD_PROMO_CODE (PROMO_CODE) ON DELETE CASCADE
   , FOREIGN KEY (CUSTOMER_ID) REFERENCES DD_CUSTOMER (CUSTOMER_ID) ON DELETE CASCADE
);

CREATE TABLE DD_DRIVER_VEHICLE (
   PLATE_NUMBER VARCHAR(20)
   , DRIVER_ID NUMBER
   , INSURANCE_NUMBER VARCHAR(50) NOT NULL
   , LICENSE_NUMBER VARCHAR(50) NOT NULL
   , TYPE VARCHAR(50)
   , COLOR VARCHAR(30)
   , PRIMARY KEY (PLATE_NUMBER, DRIVER_ID)
   , FOREIGN KEY (DRIVER_ID) REFERENCES DD_DRIVER (DRIVER_ID) ON DELETE CASCADE
```

```
);


-- DRIVER BANK DETAILS
CREATE TABLE DD_BANKDETAILS(
   BANK_NAME VARCHAR(100) NOT NULL
   , ROUTING_NUMBER NUMBER(15)
   , ACCOUNT_NUMBER NUMBER(20)
   , DRIVER_ID NUMBER
   , PRIMARY KEY (ACCOUNT_NUMBER, ROUTING_NUMBER)
   , FOREIGN KEY (DRIVER_ID) REFERENCES DD_DRIVER (DRIVER_ID) ON DELETE CASCADE
);



----- payments related table

CREATE TABLE DD_PAYMENTS(
   PAYMENT_ID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
   , ADDED_ON DATE DEFAULT SYSDATE
   , CUSTOMER_ID NUMBER NOT NULL
   , PRIMARY KEY (PAYMENT_ID)
   , FOREIGN KEY (CUSTOMER_ID) REFERENCES DD_CUSTOMER (CUSTOMER_ID) ON DELETE CASCADE
);



CREATE TABLE DD_APPLEPAY (
   PAYMENT_ID NUMBER
   , APPLE_ID VARCHAR(100) NOT NULL UNIQUE
   , PRIMARY KEY (PAYMENT_ID)
   , FOREIGN KEY (PAYMENT_ID) REFERENCES DD_PAYMENTS (PAYMENT_ID) ON DELETE CASCADE
);

CREATE TABLE DD_PAYPAL (
   PAYMENT_ID NUMBER
   , PAYPAL_ID VARCHAR(100) NOT NULL UNIQUE
   , PRIMARY KEY (PAYMENT_ID)
   , FOREIGN KEY (PAYMENT_ID) REFERENCES DD_PAYMENTS (PAYMENT_ID) ON DELETE CASCADE
);

CREATE TABLE DD_CREDITCARD (
   PAYMENT_ID NUMBER
   , CREDIT_CARD_NUMBER VARCHAR(100) NOT NULL
```

```
    , CVV NUMBER(4) NOT NULL
    , CC_NAME VARCHAR(100) NOT NULL
    , EXPIRY_DATE DATE NOT NULL
    , PRIMARY KEY (PAYMENT_ID)
    , FOREIGN KEY (PAYMENT_ID) REFERENCES DD_PAYMENTS (PAYMENT_ID) ON DELETE CASCADE
);

CREATE TABLE DD_CUSTOMER_CARE (
    EMP_ID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
    , SSN VARCHAR(9) NOT NULL UNIQUE
    , SALARY NUMBER(6) NOT NULL
    , JOINING_DATE DATE DEFAULT SYSDATE
    , PRIMARY KEY (EMP_ID)
);

CREATE TABLE DD_TICKETS (
    TICKET_NO NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
    , CUSTOMER_ID NUMBER NOT NULL
    , EMP_ID NUMBER NOT NULL
    , DESCRIPTION VARCHAR(100)
    , STATUS VARCHAR(20) DEFAULT 'PENDING'
    , RAISED_AT DATE DEFAULT SYSDATE
    , RESOLVED_AT DATE
    , PRIMARY KEY (TICKET_NO)
    , FOREIGN KEY (CUSTOMER_ID) REFERENCES DD_CUSTOMER (CUSTOMER_ID)
    , FOREIGN KEY (EMP_ID) REFERENCES DD_CUSTOMER_CARE (EMP_ID)
);

CREATE TABLE DD_CUISINE (
    CUISINE_ID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
    , TYPE VARCHAR(20) UNIQUE
    , PRIMARY KEY (CUISINE_ID)
);

CREATE TABLE DD_RESTAURANT(
    RESTAURANT_ID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
    , NAME VARCHAR(100)
    , PHONE_NUMBER NUMBER(13) NOT NULL
    , EMAIL_ID VARCHAR(100) NOT NULL
    , CUISINE_ID NUMBER
    , OPENING_TIME NUMBER(5)
    , CLOSING_TIME NUMBER(5)
    , PRIMARY KEY (RESTAURANT_ID)
    , FOREIGN KEY (CUISINE_ID) REFERENCES DD_CUISINE (CUISINE_ID) ON DELETE SET NULL -- DON'T CASCADE
```

```
);

CREATE TABLE DD_ITEMS(
    ITEM_ID NUMBER
    , ITEM_NAME VARCHAR(100) NOT NULL
    , RESTAURANT_ID NUMBER NOT NULL
    , CALORIES NUMBER (4)
    , SERVING_SIZE NUMBER(4)
    , PRICE DECIMAL (10, 5) NOT NULL
    , DESCRIPTION VARCHAR(1000)
    , CATEGORY VARCHAR(10)
    , PRIMARY KEY (ITEM_ID, RESTAURANT_ID)
    , FOREIGN KEY (RESTAURANT_ID) REFERENCES DD_RESTAURANT (RESTAURANT_ID) ON DELETE CASCADE
);

CREATE TABLE DD_RESTAURANT_OWNER (
    SSN VARCHAR(9)
    , RESTAURANT_ID NUMBER
    , NAME VARCHAR(100) NULL
    , EMAIL_ID VARCHAR(100) NOT NULL
    , PHONE_NUMBER NUMBER(13) NOT NULL
    , BANK_NAME VARCHAR(100) NOT NULL
    , ROUTING_NUMBER NUMBER(15)
    , ACCOUNT_NUMBER NUMBER(20)
    , PRIMARY KEY (RESTAURANT_ID, SSN)
    , FOREIGN KEY (RESTAURANT_ID) REFERENCES DD_RESTAURANT (RESTAURANT_ID) ON DELETE CASCADE
);

CREATE TABLE DD_ORDERS (
    ORDER_ID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY
    , ORDER_STATUS VARCHAR(20) DEFAULT 'PENDING'
    , SPECIAL_INSTRUCTIONS VARCHAR(300)
    , DOOR_NO VARCHAR(20)
    , LATITUDE DECIMAL(8,6)  NOT NULL
    , LONGITUDE DECIMAL(9,6) NOT NULL
    , LANDMARK VARCHAR(200)
    , TAX_AMOUNT DECIMAL(8,6) DEFAULT 0.0
    , DRIVER_SHARE DECIMAL(8,6) DEFAULT 0.0
    , TOTAL_COST DECIMAL (10, 6) DEFAULT 0.0
    , DRIVER_TIP DECIMAL (10,6) DEFAULT 0.0
    , DELIVERY_CHARGE DECIMAL(8,6) DEFAULT 0.0
    , PRIMARY KEY (ORDER_ID)
    , FOREIGN KEY (LATITUDE, LONGITUDE) REFERENCES DD_ADDRESS_LOOKUP (LATITUDE, LONGITUDE) ON DELETE SET
NULL
```

```
);

CREATE TABLE DD_ORDER_ITEMS (
    ORDER_ID NUMBER
    , ITEM_ID NUMBER NOT NULL
    , RESTAURANT_ID NUMBER NOT NULL
    , QNTY NUMBER (2) NOT NULL
    , SPECIAL_INSTRUCTIONS VARCHAR(100)
    , PRIMARY KEY (ORDER_ID, ITEM_ID, RESTAURANT_ID)
    , FOREIGN KEY (ITEM_ID, RESTAURANT_ID) REFERENCES DD_ITEMS(ITEM_ID, RESTAURANT_ID) ON DELETE CASCADE
    , FOREIGN KEY (ORDER_ID) REFERENCES DD_ORDERS (ORDER_ID) ON DELETE CASCADE
);

CREATE TABLE DD_CUSTOMER_PLACES_ORDER (
    ORDER_ID NUMBER
    , CUSTOMER_ID NUMBER
    , ORDER_TIME DATE
    , PRIMARY KEY (ORDER_ID)
    , FOREIGN KEY (CUSTOMER_ID) REFERENCES DD_CUSTOMER (CUSTOMER_ID) ON DELETE CASCADE
    , FOREIGN KEY (ORDER_ID) REFERENCES DD_ORDERS (ORDER_ID) ON DELETE CASCADE
);

CREATE TABLE DD_DISCOUNT_APPLIED_FOR_ORDER (
    ORDER_ID NUMBER
    , PROMO_CODE VARCHAR(20) NOT NULL
    , PRIMARY KEY (ORDER_ID)
    , FOREIGN KEY (ORDER_ID) REFERENCES DD_ORDERS (ORDER_ID) ON DELETE CASCADE
    , FOREIGN KEY (PROMO_CODE) REFERENCES DD_PROMO_CODE (PROMO_CODE) ON DELETE CASCADE
);

CREATE TABLE DD_ORDER_TRANSACTION (
    ORDER_ID NUMBER
    , PAYMENT_ID NUMBER NOT NULL
    , TNX_REF_NO VARCHAR(100) NOT NULL
    , TNX_TIME DATE DEFAULT SYSDATE
    , PRIMARY KEY (ORDER_ID)
    , FOREIGN KEY (ORDER_ID) REFERENCES DD_ORDERS (ORDER_ID) ON DELETE CASCADE
    , FOREIGN KEY (PAYMENT_ID) REFERENCES DD_PAYMENTS (PAYMENT_ID) ON DELETE CASCADE
);

CREATE TABLE DD_DRIVER_PICKSUP_ORDER (
    ORDER_ID NUMBER
    , DRIVER_ID NUMBER NOT NULL
    , PICKUP_TIME DATE
    , PRIMARY KEY (ORDER_ID)
    , FOREIGN KEY (DRIVER_ID) REFERENCES DD_DRIVER (DRIVER_ID) ON DELETE CASCADE
    , FOREIGN KEY (ORDER_ID) REFERENCES DD_ORDERS (ORDER_ID) ON DELETE CASCADE
);


CREATE TABLE DD_DRIVER_DELIVERS_TO_CUSTOMER(
    CUSTOMER_ID NUMBER
    , DRIVER_ID NUMBER NOT NULL
    , ORDER_ID NUMBER NOT NULL
    , DELIVERY_TIME DATE DEFAULT SYSDATE
    , RATING NUMBER(1) DEFAULT -1
    , REVIEW VARCHAR(100)
    , PRIMARY KEY (CUSTOMER_ID, DRIVER_ID, ORDER_ID)
    , FOREIGN KEY (DRIVER_ID) REFERENCES DD_DRIVER (DRIVER_ID) ON DELETE CASCADE
    , FOREIGN KEY (ORDER_ID) REFERENCES DD_ORDERS (ORDER_ID) ON DELETE CASCADE
  , FOREIGN KEY (CUSTOMER_ID) REFERENCES DD_CUSTOMER (CUSTOMER_ID) ON DELETE CASCADE
);

CREATE TABLE DD_CUSTOMER_GIVES_RATING (
    CUSTOMER_ID NUMBER
    , RESTAURANT_ID NUMBER
    , PRIMARY KEY (CUSTOMER_ID, RESTAURANT_ID)
    , RATING NUMBER NOT NULL
    , REVIEW VARCHAR(200) NOT NULL
    , FOREIGN KEY (CUSTOMER_ID) REFERENCES DD_CUSTOMER (CUSTOMER_ID) ON DELETE CASCADE
    , FOREIGN KEY (RESTAURANT_ID) REFERENCES DD_RESTAURANT (RESTAURANT_ID) ON DELETE CASCADE
);

CREATE TABLE DD_CUSTOMER_FAVOURITES (
    CUSTOMER_ID NUMBER
    , RESTAURANT_ID NUMBER
    , PRIMARY KEY (CUSTOMER_ID, RESTAURANT_ID)
    , FOREIGN KEY (CUSTOMER_ID) REFERENCES DD_CUSTOMER (CUSTOMER_ID) ON DELETE CASCADE
    , FOREIGN KEY (RESTAURANT_ID) REFERENCES DD_RESTAURANT (RESTAURANT_ID) ON DELETE CASCADE
);
```

# PL/SQL

## Triggers:

### Driver_Age_Check:

It’s a BEFORE TRIGGER which checks if the driver age is above 18 based on the date of birth, if yes, then Application Error is thrown.

```
CREATE OR REPLACE TRIGGER DRIVER_AGE_CHECK
BEFORE INSERT
  ON DD_DRIVER
  FOR EACH ROW
DECLARE
  AGE NUMBER;
BEGIN
  AGE := floor(months_between( sysdate, :NEW.DATE_OF_BIRTH) /12);
  IF AGE < 18 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Driver not attained 18 years.');
  END IF;
END;
/
```

### Credit_Card_Expiry_Check:

Trigger that checks if credit card during adding, if its expiry date is greater than current system date

```
CREATE OR REPLACE TRIGGER CREDIT_CARD_EXPIRY_CHECK
BEFORE INSERT
  ON DD_CREDITCARD
  FOR EACH ROW
BEGIN
  IF :NEW.EXPIRY_DATE < SYSDATE THEN
    RAISE_APPLICATION_ERROR(-20001, 'Credit Card is expired.');
  END IF;
END;
/
```

## Update Order Status

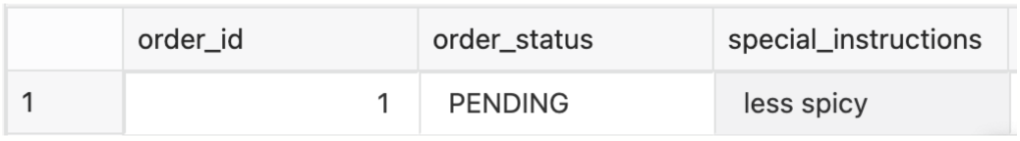
This PLSQL block is triggered, when a driver delivers to customer. The order status is updated to completed on Orders Table.

```
CREATE OR REPLACE TRIGGER UPDATE_ORDER_STATUS
AFTER INSERT
   ON DD_DRIVER_DELIVERS_TO_CUSTOMER
   FOR EACH ROW
DECLARE
   AGE NUMBER;
BEGIN
   UPDATE DD_ORDERS
   SET ORDER_STATUS = 'COMPLETED'
   WHERE ORDER_ID = :NEW.ORDER_ID;
END;
/
```

## Sample Run:

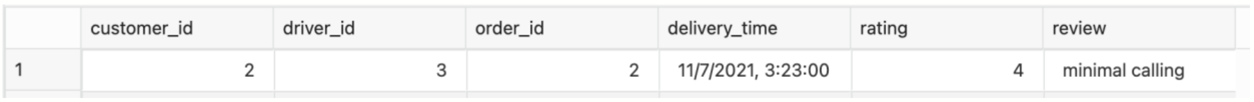
### Before insert

### Orders Table



| | order_id | order_status | special_instructions |
|---|---|---|---|
| 1 | 1 | PENDING | less spicy |

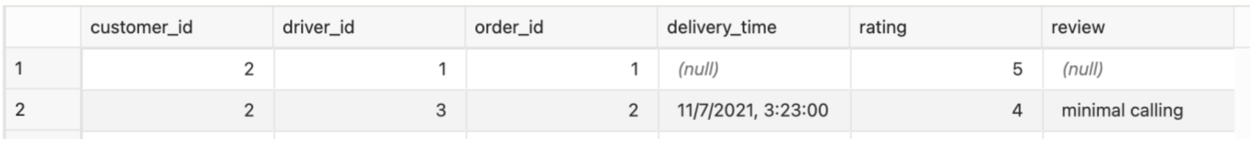
### Driver_delivers_to_customer



| | customer_id | driver_id | order_id | delivery_time | rating | review |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 11/7/2021, 3:23:00 | 4 | minimal calling |

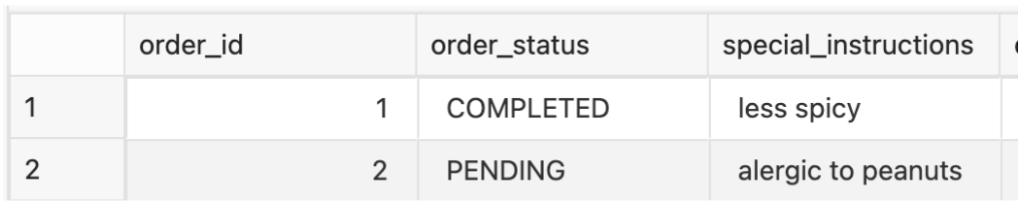
### After Insert

### Driver_delivers_to_customer



| | customer_id | driver_id | order_id | delivery_time | rating | review |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 1 | *(null)* | 5 | *(null)* |
| 2 | 2 | 3 | 2 | 11/7/2021, 3:23:00 | 4 | minimal calling |

### Orders Table



| | order_id | order_status | special_instructions |
|---|---|---|---|
| 1 | 1 | COMPLETED | less spicy |
| 2 | 2 | PENDING | alergic to peanuts |

## Procedures:

### Procedure 1:

**Provide Discount to Customers:**

Based on previous 2 weeks order count, PL/SQL procedure will provide discounts to customers with an expiry of 2 weeks from current date.
A random PROMO CODE, with max discount percentage of 60 and max of max_amount for the coupon that can be availed as 40$.

```
CREATE OR REPLACE PROCEDURE PROVIDE_DISCOUNT (CUST_ID IN NUMBER)
AS
   PROMO_SUFFIX VARCHAR(5);
   EXPIRY_DATE DATE;
   FINAL_PROMO_CODE VARCHAR(20);

   PERCENT NUMBER;
   MAX_AMOUNT NUMBER;

   V_ORDERS NUMBER;

BEGIN
   DBMS_OUTPUT.PUT_LINE('PROCEDURE FOR : ' || CUST_ID);
   -- setting the promo_suffix for the promo code
   SELECT DBMS_RANDOM.STRING('U', 5) INTO PROMO_SUFFIX FROM DUAL;
   SELECT SYSDATE + 21 INTO EXPIRY_DATE FROM DUAL; -- 21 days from current date

   -- count of orders done in past 2 weeks
   SELECT COUNT(*) INTO V_ORDERS FROM DD_CUSTOMER_PLACES_ORDER WHERE CUSTOMER_ID = 2 AND ORDER_TIME >=
SYSDATE - 14;


   PERCENT := 10*V_ORDERS;
   SELECT GREATEST(PERCENT, 60) INTO PERCENT FROM DUAL;
   MAX_AMOUNT := 20*V_ORDERS;
   SELECT GREATEST(MAX_AMOUNT, 40) INTO MAX_AMOUNT FROM DUAL;

   FINAL_PROMO_CODE := PROMO_SUFFIX || PERCENT;
   DBMS_OUTPUT.PUT_LINE(FINAL_PROMO_CODE);

   -- DBMS_OUTPUT.PUT_LINE('PROMO CODE : ' || PROMO_SUFFIX || ' EXPIRY DATE : '  || EXPIRY_DATE);
   INSERT INTO DD_PROMO_CODE (PROMO_CODE, EXPIRY_DATE, PERCENT, MAX_AMOUNT)
   VALUES ( FINAL_PROMO_CODE, EXPIRY_DATE, PERCENT, MAX_AMOUNT);

   INSERT INTO DD_CUSTOMER_GETS_OFFER (PROMO_CODE, CUSTOMER_ID, IS_ACTIVE)
   VALUES ( FINAL_PROMO_CODE, CUST_ID, 'Y');

END PROVIDE_DISCOUNT;
/
```

## Procedure 2:

Calculate total cost of food items, tax calculation, and eventually the total cost of the entire order including driver tips.

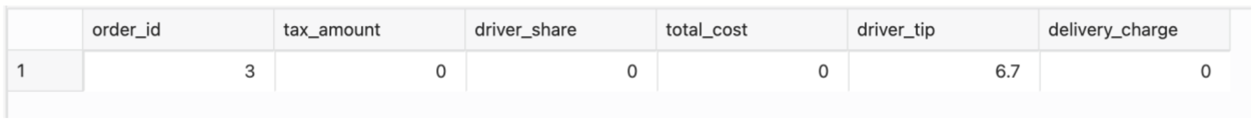
Logic:

- Calculate the total sum of all the products ordered
- 10% of the total cost is tax
- delivery charge is 15% (capped with 30$ as maximum amount)
    - if driver is a doordash_pass holder, then delivery charge is 0$
- driver share is 5% of the total item cost
- Final Cost to be paid = total_item_prices + driver_tips + delivery_charge + tax

```
CREATE OR REPLACE PROCEDURE UPDATE_PRICES(IN_ID IN NUMBER )
AS
  VTAX_AMOUNT DD_ORDERS.TAX_AMOUNT%TYPE;
  VDRIVER_SHARE DD_ORDERS.DRIVER_SHARE%TYPE;
  VDELIVERY_CHARGE DD_ORDERS.DELIVERY_CHARGE%TYPE;
  VDRIVER_TIP DD_ORDERS.TAX_AMOUNT%TYPE;

  VTOTAL_COST NUMBER;
  TOTAL_ITEM_COST DD_ITEMS.PRICE%TYPE;
  VDOORDASH_PASS NUMBER;

  ITEM_PRICE DD_ITEMS.PRICE%TYPE;
  CITEM_ID DD_ORDER_ITEMS.ITEM_ID%TYPE ;
  CREST_ID DD_ORDER_ITEMS.RESTAURANT_ID%TYPE ;
  CQNTY DD_ORDER_ITEMS.QNTY%TYPE ;


  CURSOR ITEMS_CURSOR(OID NUMBER) IS
      SELECT ITEM_ID, RESTAURANT_ID, QNTY FROM DD_ORDERS
      INNER JOIN DD_ORDER_ITEMS
      ON DD_ORDERS.ORDER_ID = DD_ORDER_ITEMS.ORDER_ID
      AND DD_ORDERS.ORDER_ID= OID;

  CUST_ID DD_CUSTOMER.CUSTOMER_ID%TYPE;

BEGIN
  VTOTAL_COST := 0;
  SELECT CUSTOMER_ID INTO CUST_ID FROM DD_CUSTOMER_PLACES_ORDER WHERE ORDER_ID = IN_ID;
  SELECT DRIVER_TIP INTO VDRIVER_TIP FROM DD_ORDERS WHERE ORDER_ID = IN_ID;
  DBMS_OUTPUT.PUT_LINE('Customer ID : ' || CUST_ID );
  TOTAL_ITEM_COST := 0;
  OPEN ITEMS_CURSOR(IN_ID);
```

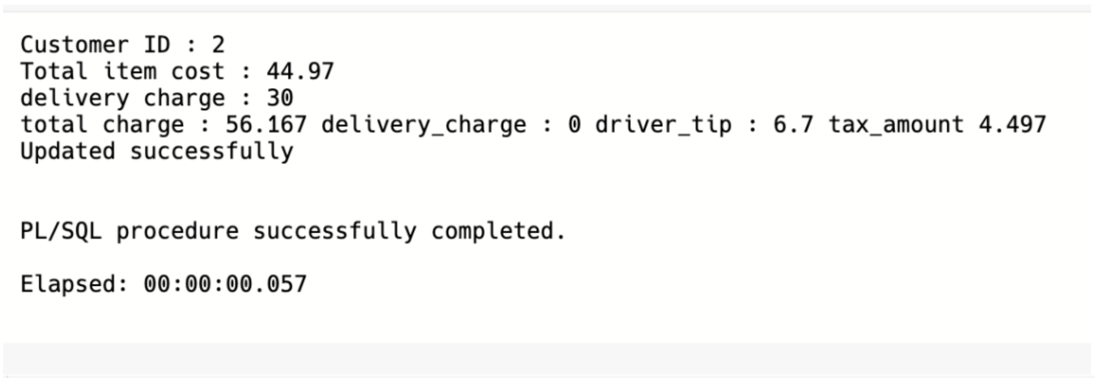
```
   LOOP
   FETCH ITEMS_CURSOR INTO CITEM_ID, CREST_ID, CQNTY;
      EXIT WHEN ITEMS_CURSOR%NOTFOUND;
      SELECT PRICE*CQNTY INTO ITEM_PRICE
      FROM DD_ITEMS WHERE RESTAURANT_ID = CREST_ID AND ITEM_ID = CITEM_ID;
      TOTAL_ITEM_COST := TOTAL_ITEM_COST + ITEM_PRICE;
   END LOOP;
   DBMS_OUTPUT.PUT_LINE('Total item cost : ' || TOTAL_ITEM_COST );



   VDELIVERY_CHARGE := TOTAL_ITEM_COST * 0.15;
   SELECT GREATEST(VDELIVERY_CHARGE, 30) INTO VDELIVERY_CHARGE FROM DUAL;
   DBMS_OUTPUT.PUT_LINE('delivery charge : ' || VDELIVERY_CHARGE );



   VDRIVER_SHARE := 0.05*TOTAL_ITEM_COST;
   VTAX_AMOUNT := TOTAL_ITEM_COST * 0.1;

   -- set delivery charge =0, if user has
   SELECT CASE WHEN DOORDASH_PASS = 'Y' THEN 1 ELSE 0 END INTO VDOORDASH_PASS FROM DD_CUSTOMER WHERE
CUSTOMER_ID = CUST_ID;
   IF  VDOORDASH_PASS = 1 THEN
      VDELIVERY_CHARGE :=0;
   END IF;
   VTOTAL_COST := TOTAL_ITEM_COST + VDELIVERY_CHARGE + VDRIVER_TIP + VTAX_AMOUNT;

   DBMS_OUTPUT.PUT_LINE('total charge : ' || vtotal_cost || ' delivery_charge : ' || vdelivery_charge || ' driver_tip : ' || vdriver_tip  || '
tax_amount ' || vtax_amount );

   UPDATE DD_ORDERS
   SET DELIVERY_CHARGE = VDELIVERY_CHARGE
      , TAX_AMOUNT = VTAX_AMOUNT
      , DRIVER_SHARE = VDRIVER_SHARE
      , TOTAL_COST = VTOTAL_COST
   WHERE ORDER_ID = IN_ID;

   COMMIT;
   DBMS_OUTPUT.PUT_LINE('Updated successfully');



END UPDATE_PRICES;
/
```

Before Procedure Calling

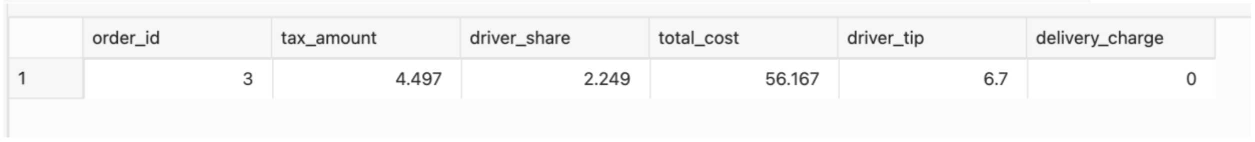


| | order_id | tax_amount | driver_share | total_cost | driver_tip | delivery_charge |
|---|---|---|---|---|---|---|
| 1 | 3 | 0 | 0 | 0 | 6.7 | 0 |

After Calling procedure



```
Customer ID : 2
Total item cost : 44.97
delivery charge : 30
total charge : 56.167 delivery_charge : 0 driver_tip : 6.7 tax_amount 4.497
Updated successfully


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.057
```



| | order_id | tax_amount | driver_share | total_cost | driver_tip | delivery_charge |
|---|---|---|---|---|---|---|
| 1 | 3 | 4.497 | 2.249 | 56.167 | 6.7 | 0 |

Delivery charge = 0, as the customer holds doordash pass

## Conclusion

There are lots of learning from this project, in designing an efficient system. Right from ER diagram to writing the Relational Mapping a review of complete course could be summarized in this project.

Also, there are various improvements that could be done on this very existent system like improving performance via index, partitioning and sub-partitioning, and efficient queries.