INDEX

Aim:

To design and implement the comprehensive database management system for an online food delivery system

Description:

- The system will be responsible for storing and managing data related to user orders, menus, restaurant information, delivery person, customer details and payment details.

➢ **Two Functions are there:** a)*place_order*

b)*cancel_order*

➢ **Two procedures are there:** a)*cart_update*

b)*assign_delivery person*

➢ **Two triggers are there:**

1)*update_delivery_person_avaiabilty*

2)*prevent_restaurant_deletion*

3)*update_orderstatus_and_deliverystatus*

The system have following features:

User account management: User can create an account and manage their personal information such as name ,address, phone number and payment details.

Restaurant account management: Restaurants can create an account and manage their menu items, prices, and availability.

Cart management: Users can add items to their cart and place an order.

Order management: The system should be able to manage orders placed by users, including order status, delivery details, and payment processing.

Delivery boy management: The system should be able to assign delivery boys to orders and track their delivery status.

Assumptions:

Only online payments are allowed: The system assumes that customers will pay for their orders online using a secure payment gateway. Cash-on-delivery or other payment methods are not supported.

Tables:

NOTE:
1)Entities in green color are primary key.

2)Entities in light blue color are foreign key.

3)Entities in *italic* should be always unique.

USERS(user_id, *username*, password, *email*, phone_number, address)

RESTAURANTS(restaurant_id, restaurant_name , description, phone_number, address)

FOOD_CATEGORIES(category_id , *category_name*)

FOOD_ITEMS(food_item_id, food_item_name, description, price, category_id, restaurant_id)

ORDERS(order_id, user_id, restaurant_id, order_date, total_amount, order_status)

ORDER_ITEMS(order_item_id, order_id, food_item_id, quantity, total_cost,price)

CART (cart_id, user_id, food_item_id, quantity, added_date)

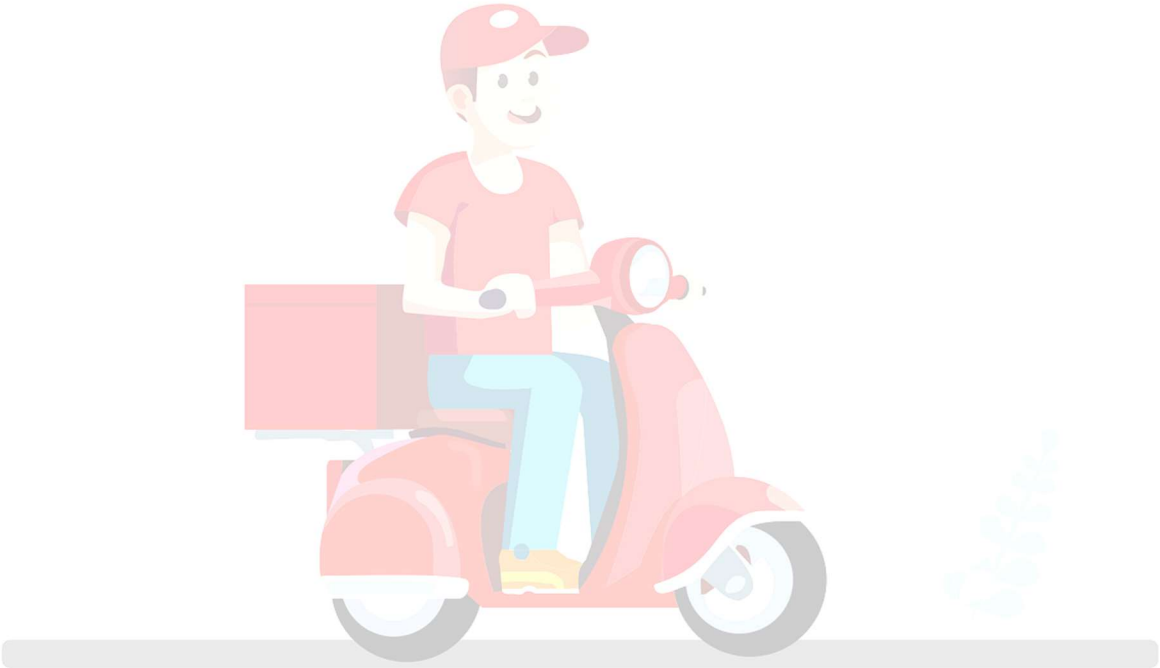
PAYMENTS(payment_id, order_id, payment_method, amount, payment_date, payment_time, payment_status)

DELIVERY_PERSON (dp_id, dp_name, *dp_email* , *dp_phone* , dp_address ,is_available)

DELIVERY (delivery_id, order_id, dp_id, delivery_date, delivery_status)

ORDER_HISTORY (order_id, user_id, restaurant_id, order_date, total_amount,delivery_status)

CANCELLATION (cancellation_id, order_id, user_id, reason, created_at)



Tables with Constraints

```
CREATE TABLE USERS(
   user_id              NUMBER(10) NOT NULL PRIMARY KEY,
   username            VARCHAR2(50) NOT NULL UNIQUE,
   password             VARCHAR2(50) NOT NULL,
   email                VARCHAR2(100) NOT NULL UNIQUE,
   phone_number        VARCHAR2(20),
   address VARCHAR2(200)
   );

CREATE TABLE RESTAURANTS(
   restaurant_id            NUMBER(10) NOT NULL PRIMARY KEY,
   restaurant_name          VARCHAR2(100) NOT NULL,
   description              VARCHAR2(500),
   phone_number             VARCHAR2(20),
   address                  VARCHAR2(200)
   );

CREATE TABLE FOOD_CATEGORIES(
    category_id             NUMBER(10) NOT NULL PRIMARY KEY,
    category_name           VARCHAR2(100) NOT NULL UNIQUE
    );

CREATE TABLE FOOD_ITEMS(
   food_item_id             NUMBER(10) NOT NULL PRIMARY KEY,
   food_item_name           VARCHAR2(100) NOT NULL,
   description              VARCHAR2(500),
   price                    NUMBER(10,2) NOT NULL,
   category_id              NUMBER(10) NOT NULL,
   restaurant_id            NUMBER(10) NOT NULL,
   CONSTRAINT fk_category_id FOREIGN KEY (category_id)
REFERENCES FOOD_CATEGORIES(category_id),
```

```
   CONSTRAINT fk_restaurant_id FOREIGN KEY (restaurant_id)
REFERENCES RESTAURANTS(restaurant_id)
   );

CREATE TABLE ORDERS(
order_id            NUMBER(10) NOT NULL PRIMARY KEY,
user_id             NUMBER(10) NOT NULL,
restaurant_id       NUMBER(10) NOT NULL,
order_date          DATE NOT NULL,
total_amount        NUMBER(10,2) NOT NULL,
order_status        VARCHAR2(50) NOT NULL,
CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES
USERS(user_id),
CONSTRAINT fk_restaurant_id_2 FOREIGN KEY (restaurant_id)
REFERENCES RESTAURANTS(restaurant_id)
   );

CREATE TABLE ORDER_ITEMS(
order_item_id       NUMBER(10) NOT NULL PRIMARY KEY,
order_id            NUMBER(10) NOT NULL,
food_item_id        NUMBER(10) NOT NULL,
quantity            NUMBER(5) NOT NULL,
total_cost          number(10) not null,
price               NUMBER(10,2) NOT NULL,
CONSTRAINT fk_order_id FOREIGN KEY (order_id) REFERENCES
ORDERS(order_id),
CONSTRAINT fk_food_item_id FOREIGN KEY (food_item_id)
REFERENCES FOOD_ITEMS(food_item_id)
   );

CREATE TABLE CART (
cart_id             NUMBER(10) NOT NULL PRIMARY KEY,
user_id             NUMBER(10) NOT NULL,
food_item_id        NUMBER(10) NOT NULL,
quantity            NUMBER(5) NOT NULL,
```

```
added_date        DATE NOT NULL,
CONSTRAINT fk_cart_user FOREIGN KEY (user_id) REFERENCES
USERS(user_id),
CONSTRAINT fk_cart_item FOREIGN KEY (food_item_id) REFERENCES
FOOD_ITEMS(food_item_id)
  );

CREATE TABLE PAYMENTS(
payment_id          NUMBER(10) NOT NULL PRIMARY KEY,
order_id            NUMBER(10) NOT NULL,
payment_method      VARCHAR2(50) NOT NULL,
amount              NUMBER(10,2) NOT NULL,
payment_date        DATE NOT NULL,
payment_time        timestamp not null,
payment_status      varchar2(50),
CONSTRAINT fk_order_id_2 FOREIGN KEY (order_id) REFERENCES
ORDERS(order_id)
  );

CREATE TABLE DELIVERY_PERSON (
dp_id               NUMBER(10) PRIMARY KEY,
dp_name             VARCHAR2(50) NOT NULL,
dp_email            VARCHAR2(50) UNIQUE,
dp_phone            VARCHAR2(20) UNIQUE,
dp_address          VARCHAR2(500),
is_available        varchar2(50)
  );

CREATE TABLE DELIVERY (
delivery_id         NUMBER(10) PRIMARY KEY,
order_id            NUMBER(10) NOT NULL,
dp_id               NUMBER(10) NOT NULL,
delivery_date       DATE NOT NULL,
delivery_status     VARCHAR2(50) NOT NULL,
```

```
CONSTRAINT fk_order_delivery FOREIGN KEY (order_id) REFERENCES
ORDERS (order_id),
CONSTRAINT fk_delivery_person_delivery FOREIGN KEY (dp_id)
REFERENCES DELIVERY_PERSON (dp_id)
   );

CREATE TABLE ORDER_HISTORY (
order_id                    NUMBER(10) NOT NULL,
user_id                     NUMBER(10) NOT NULL,
restaurant_id               NUMBER(10) NOT NULL,
order_date                  DATE NOT NULL,
total_amount                NUMBER(10,2) NOT NULL,
delivery_status             varchar2(50) not null,
CONSTRAINT fk_p PRIMARY KEY (order_id,user_id),
CONSTRAINT fk_user_order FOREIGN KEY (user_id) REFERENCES
USERS(user_id),
CONSTRAINT fk_restaurant_order FOREIGN KEY (restaurant_id)
REFERENCES RESTAURANTS(restaurant_id)
   );

CREATE TABLE cancellation (
   cancellation_id      NUMBER NOT NULL PRIMARY KEY,
   order_id             NUMBER NOT NULL,
   user_id              NUMBER NOT NULL,
   reason               VARCHAR(255) NOT NULL,
   created_at           TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (order_id) REFERENCES orders(order_id),
   FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

➢ Cardinalities:

- Users-Orders 1:M
- Users-Cart 1:M
- Orders-Order_items 1:M
- Delivery_person-delivery M:1
- ORDERS-DELIVERY 1:1
- ORDERS-CANCELLATION 1:M
- ORDERS-ORDER_HISTORY 1:1
- RESTAURANTS-ORDERS 1:M
- PAYMENTS-ORDERS 1:1
- RESTAURANTS-FOOD_ITEMS 1:M
- FOOD_CATEGORIES-FOOD_ITEMS 1:M
- USERS-ORDER_HISTORY 1:M

FUNCTIONS

1) The place_order creates a new order with specified details. It also updates orders, order_items tables and process payment for the order and make the payment_status as received. Finally it updates the order history of a user in a order_history table.

```
CREATE OR REPLACE FUNCTION place_order(
    p_user_id           IN orders.user_id%TYPE,
    p_restaurant_id     IN orders.restaurant_id%TYPE,
    p_order_date        IN orders.order_date%TYPE,
    p_order_items       IN order_items%ROWTYPE,
    p_payment_method IN payments.payment_method%TYPE,
    p_payment_date      IN payments.payment_date%TYPE,
    p_amount            IN payments.amount%TYPE
) RETURN NUMBER IS
    v_order_id          orders.order_id%TYPE;
    v_total_amount      orders.total_amount%TYPE;
    v_order_status      orders.order_status%TYPE := 'Processing';
    v_delivery_status   order_history.delivery_status%TYPE :=
'Processing';
BEGIN
    SELECT SUM(oi.quantity * fi.price)
    INTO v_total_amount
```

```
    FROM order_items oi
    JOIN food_items fi ON oi.food_item_id = fi.food_item_id
    WHERE oi.order_item_id = p_order_items.order_item_id;

    INSERT INTO orders (order_id, user_id, restaurant_id, order_date,
total_amount, order_status)
    VALUES (orders_seq.NEXTVAL, p_user_id, p_restaurant_id,
p_order_date, v_total_amount, v_order_status)
    RETURNING order_id INTO v_order_id;

    UPDATE order_items
    SET order_id = v_order_id, total_cost = p_order_items.quantity *
p_order_items.price
    WHERE order_item_id = p_order_items.order_item_id;
 INSERT INTO payments (payment_id, order_id, payment_method,
payment_status, payment_date, amount)
    VALUES (payment_seq.NEXTVAL, v_order_id, p_payment_method,
'Pending', p_payment_date, p_amount);
 INSERT INTO order_history (order_id, user_id, restaurant_id,
order_date, total_amount, delivery_status)
    VALUES (v_order_id, p_user_id, p_restaurant_id, p_order_date,
v_total_amount, v_delivery_status);
   COMMIT;
  RETURN v_order_id;
END place_order;
```

2) The cancel_order checks if time difference between payment and current time is less than 10 minutes, if it is then it updates the order status to to “cancelled” and payment_status to “refunded soon”. It also update the cancellation table with all details. But if the time limit exceeds 10 minutes than function does not allow cancellation and returns message.

```
CREATE OR REPLACE FUNCTION cancel_order(
    p_order_id       IN orders.order_id%TYPE,
    p_user_id        IN orders.user_id%TYPE,
    p_reason         IN cancellation.reason%TYPE,
    p_created_at     IN cancellation.created_at%TYPE
) RETURN BOOLEAN IS
    v_payment_time  payments.payment_time%TYPE;
    v_time_diff     INTERVAL DAY TO SECOND;
BEGIN
   SELECT payment_time INTO v_payment_time FROM payments
    WHERE order_id = p_order_id;
    v_time_diff := SYSTIMESTAMP - v_payment_time;
    IF v_time_diff <= INTERVAL '10' MINUTE THEN

        UPDATE order_history
        SET delivery_status = 'Cancelled'
        WHERE order_id = p_order_id AND user_id = p_user_id;
```

```
        UPDATE payments
        SET payment_status = 'Refunded soon'
        WHERE order_id = p_order_id;

        INSERT INTO cancellation (cancellation_id, order_id, user_id,
reason, created_at)
        VALUES (cancellation_seq.NEXTVAL, p_order_id, p_user_id,
p_reason, p_created_at);

        COMMIT;

        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END cancel_order;
```

Procedures

1) The cart_update procedure inserts a new item into the cart table and caculates the total cost of all items in the cart for the specified user. If the total cost is greater than 1000, it applies a 10% discount to total cost and updates the cart table with the discounted amount.

```
CREATE OR REPLACE PROCEDURE cart_update(
    p_user_id          IN cart.user_id%TYPE,
    p_food_item_id     IN cart.food_item_id%TYPE,
    p_quantity         IN cart.quantity%TYPE,
    p_added_date       IN cart.added_date%TYPE,
    p_total_price      IN OUT NUMBER
) AS
    v_item_price      food_items.price%TYPE;
BEGIN
    SELECT price
    INTO v_item_price
    FROM food_items
    WHERE food_item_id = p_food_item_id;

    INSERT INTO cart (cart_id, user_id, food_item_id, quantity,
added_date)
```

```
   VALUES (cart_seq.NEXTVAL, p_user_id, p_food_item_id,
p_quantity, p_added_date);
  p_total_price := p_total_price + (v_item_price * p_quantity);
  IF p_total_price > 1000 THEN
      p_total_price := p_total_price * 0.9;
      DBMS_OUTPUT.PUT_LINE('Congratulations! You have received a
10% discount on your cart total.');
   END IF;
 COMMIT;
END cart_update;
```

2) The assign_delivery_person procedure assign the delivery person to all the orders whose is_available status is yes. And assign the person to those orders only whose order_status is processing. If no one is available then it prints the message that Delivery person is not available.

```
CREATE OR REPLACE PROCEDURE assign_delivery_person (
   p_order_id                IN orders.order_id%TYPE,
   p_delivery_person_id IN delivery_person.dp_id%TYPE
   ) AS
  v_restaurant_id orders.restaurant_id%TYPE;
  v_order_status orders.order_status%TYPE;
v_delivery_person_id delivery_person.dp_id%TYPE;
```

```
BEGIN
  BEGIN
    SELECT restaurant_id, order_status
    INTO v_restaurant_id, v_order_status
    FROM orders
    WHERE order_id = p_order_id;

    IF v_order_status != 'processing' THEN
      RAISE_APPLICATION_ERROR(-20001, 'Delivery person can only be
assigned to orders in processing status.');
    END IF;
  SELECT dp_id
  INTO v_delivery_person_id
  FROM delivery_person
  WHERE is_available='yes'
  AND ROWNUM=1
  ORDER BY dp_id;
    IF v_delivery_person_id IS NULL THEN
    RAISE_APPLICATION_ERROR(-20002, 'Delivery person is not
available.');
  END IF;
    UPDATE delivery
    SET dp_id = p_delivery_person_id,
        delivery_date = SYSDATE
```

```
    WHERE order_id = p_order_id;
    UPDATE delivery_person
    SET is_available = 'no'
    WHERE dp_id = p_delivery_person_id;

    UPDATE orders
    SET order_status = 'assigned'
    WHERE order_id = p_order_id;
    DBMS_OUTPUT.PUT_LINE('Delivery person has been assigned to
order ' || p_order_id || '.');
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR(-20003, 'Order ID does not exist.');
  END;
EXCEPTION
  WHEN OTHERS THEN
    RAISE;
END;
```

Triggers

1) If the order gets delivered than the trigger update_delivery_person_avaiabilty will work to update the availability status of a delivery person.

```
CREATE OR REPLACE TRIGGER update_delivery_person_availability
AFTER UPDATE ON delivery
FOR EACH ROW
DECLARE
  v_delivery_status order_history.delivery_status%TYPE;
BEGIN
  SELECT delivery_status INTO v_delivery_status
  FROM order_history
  WHERE order_id = :new.order_id;

  IF :new.delivery_status = 'delivered' AND v_delivery_status != 'delivered' THEN
    UPDATE delivery_person
    SET is_available = 'yes'
    WHERE dp_id = :new.dp_id;
  END IF;
END;
```

2) If any order of a restaurant is pending then we cannot delete it, and that will be take care by this trigger prevent_restaurant_deletion

```
CREATE TRIGGER prevent_restaurant_deletion
BEFORE DELETE ON RESTAURANTS
FOR EACH ROW
BEGIN
  DECLARE restaurant_id INT;
  DECLARE pending_orders_count INT;

  SET restaurant_id = OLD.restaurant_id;

  SELECT COUNT(*) INTO pending_orders_count
  FROM ORDERS
  WHERE restaurant_id = restaurant_id AND order_status =assigned';

  IF pending_orders_count > 0 THEN
    DBMS_OUTPUT.PUT_LINE(' cannot delete restaurant with pending order');
  END IF;
END;
```

3) The trigger update_orderstatus_and_deliverystatus will work when delivery gets over. It will change order_status and delivery_status to delivered.

```
CREATE TRIGGER update_orderstatus_and_deliverystatus
AFTER UPDATE ON DELIVERY
FOR EACH ROW
BEGIN

  IF NEW.delivery_status = 'Delivered' THEN
    UPDATE ORDERS
    SET order_status = 'Delivered'
    WHERE order_id = NEW.order_id;

    UPDATE DELIVERY
    SET delivery_status = 'Delivered'
    WHERE delivery_id = NEW.delivery_id;
  END IF;
END;
```