PROJECT REPORT

ETSY DATABASE MANAGEMENT SYSTEM

CS 6360.001 Database Design

Instructor: Nurcan Yuruk

Etsy 2

Team - 22

Team members:

Aditya Khandare
(ark200000)
Department of
Computer Science
The University of Texas
at Dallas
Dallas, USA
aditya.khandare@utdallas.edu

Priyanshi Shah
(pds200000)
Department of
Computer Science
The University of Texas
at Dallas
Dallas, USA
priyanshi.shah@utdallas.edu

Varun Nair
(vxn190034)
Department of
Computer Science
The University of Texas
at Dallas
Dallas, USA
varununnikrishnan.nair@utdallas.edu

Table of Contents:

1. Requirements:

For this project, we have created a database system for the e-commerce website *ETSY.* On this platform;

a. A user can list products to be sold.

b. A user can buy products listed on the website.

c. A user can track the shipment details of his/her order.

d. A user can save his payment details.

e. A user can add products to his/her shopping cart.

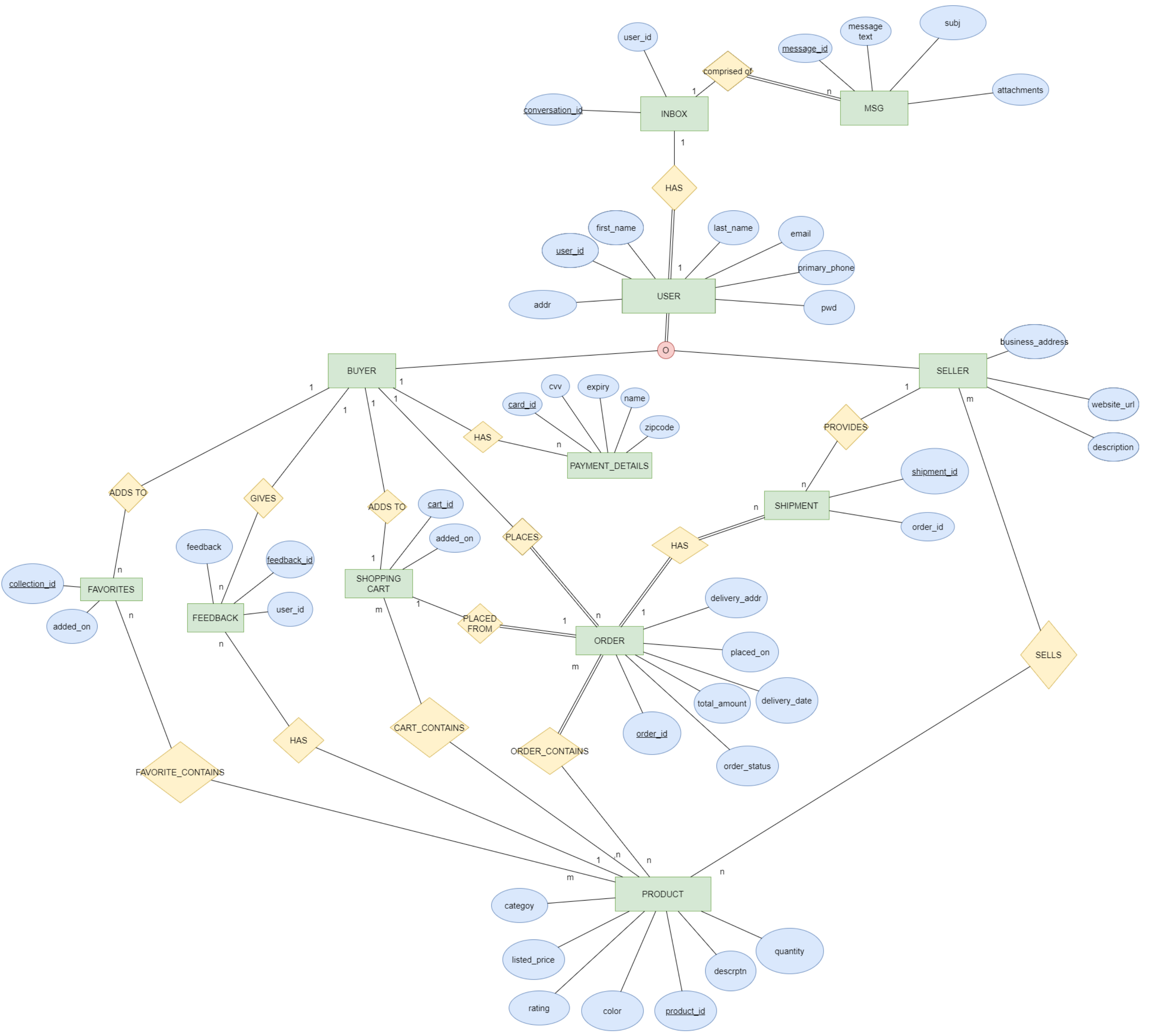
f. A user can create multiple favourites lists and add products to these lists. (eg. create a holiday shopping list, Gift shopping list, Home decor shopping list, etc)

g. A user can provide feedback on any of the listed products.

h. A user can directly message an another user (consider a buyer messaging a seller for some product/shipment details or a seller messaging a buyer to provide updates or give more information to the buyer about any product he/she is selling)

2) Entity Relationship Diagram: Erd can be found on the below link

ERD Diagram Link



Facts:

1. A user can be both buyer and seller from the same login account in *ETSY*.

Assumptions:

1. An order is always placed from a shopping cart. If a user clicks the "buy now" option on a product, that product will be automatically added to the cart.
2. A seller can operate from only 1 business address.
3. A shipment can be associated with only 1 order, but a shipment does not necessarily have all products in that order (can ship products separately)
4. Same products of different colours or sizes will have different product_id

3. Relational Schema:

To map ER diagram into a relational schema, we considered the following mapping rules.

1. For each 1: 1 binary relationship, in the total participation entity add the primary key of the other entity as the foreign key.
2. For 1: N binary relationship, add to the entity on the N side the primary key of the other entity as the foreign key.
3. For M: N binary relationship, make a new entity with foreign key as the primary key of the two participating entities. Their combination forms the new primary key.

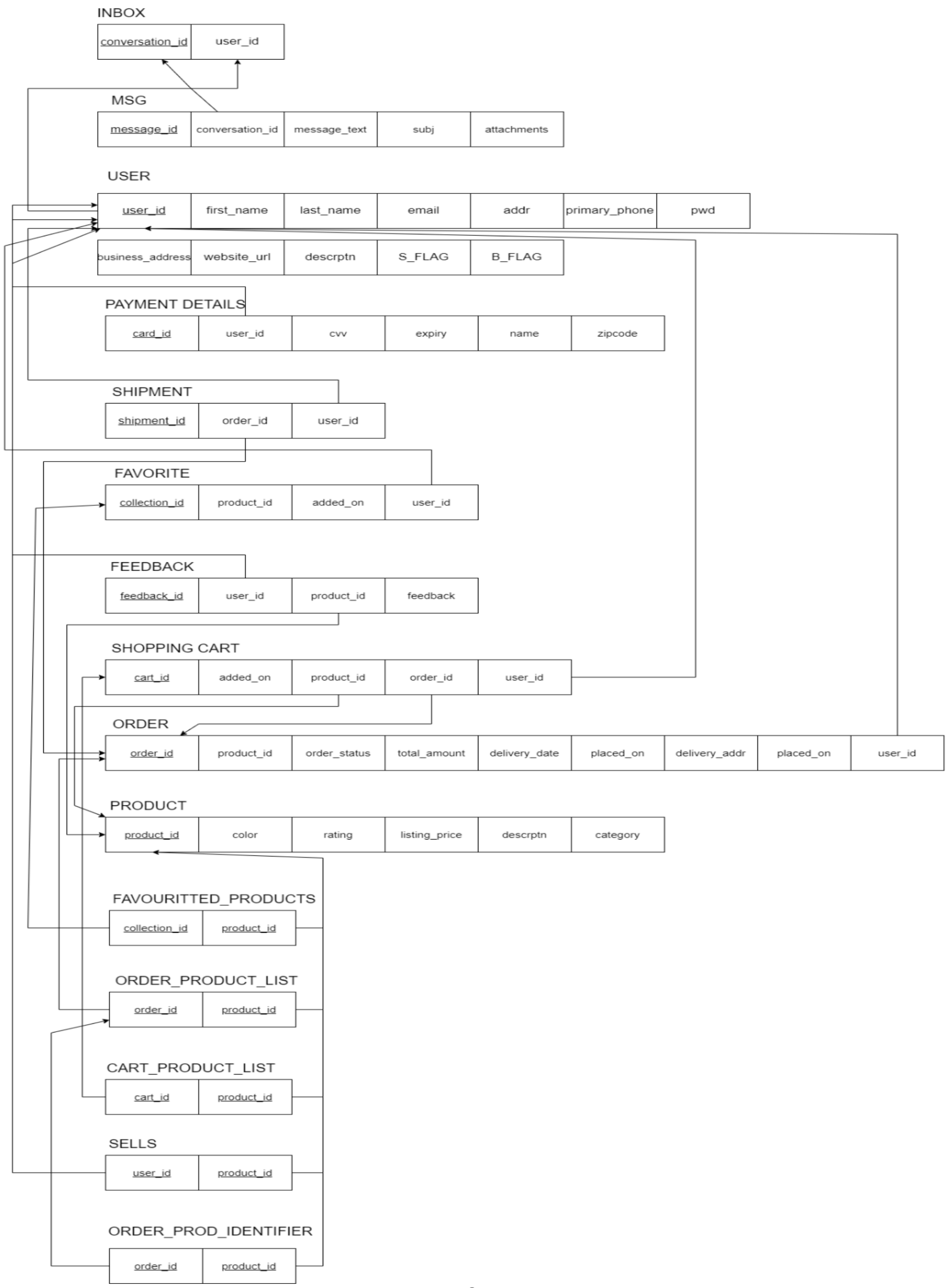
Note: The final relational schema diagram is posted after the normalization step in this report.

4. Normalization:

Once we converted the ER Model into Relational tables by following the mapping guidelines, we analysed and confirmed that our design does not violate any conditions of 3NF.

Thus, the resultant Relational Tables were already in 3NF form.

5. Final Relational Schema: Relational Schema Diagram Link



6. SQL Commands:

Following are the queries to create databases, based on the normalised relation created in the previous step:

1. To create the main user table:

```sql
CREATE TABLE user (
    userId VARCHAR(10) PRIMARY KEY,
    email VARCHAR(25),
    first_name VARCHAR(25) NOT NULL,
    last_name VARCHAR(25),
    primary_phone NUMBER(10),
    addr VARCHAR(50),
    pwd VARCHAR(30) NOT NULL,
    s_flag NUMBER(1) NOT NULL,
    b_flag NUMBER(1) NOT NULL
);
```

2. To create the inbox table:

```sql
CREATE TABLE inbox(
    conversation_id VARCHAR(10) PRIMARY KEY,
    userId VARCHAR(10) NOT NULL,
      FOREIGN KEY (userId) REFERENCES USER (userId) ON DELETE
CASCADE
);
```

3. To create the message table:

```sql
CREATE TABLE msg(
    message_id VARCHAR(10) PRIMARY KEY,
    message_text VARCHAR(255),
    subj VARCHAR(100),
    conversation_id VARCHAR(10),
                FOREIGN KEY (conversation_id) REFERENCES
inbox(conversation_id) ON DELETE CASCADE);
```

4. To create the Payment details table:

```
CREATE TABLE payment_details(
    card_id INTEGER PRIMARY KEY,
    expiry DATE NOT NULL,
    cvv NUMBER(3) NOT NULL,
    zipcode NUMBER(5) NOT NULL,
    userId VARCHAR(10) NOT NULL,
      FOREIGN KEY (userId) REFERENCES USER (userId) ON DELETE CASCADE
);
```

5. To create the product table:

```
CREATE TABLE product(
    product_id VARCHAR(10) PRIMARY KEY,
    dscrptn VARCHAR(50),
    Quantity INTEGER NOT NULL CHECK (Quantity >= 0),
    Color VARCHAR(10),
    Rating NUMBER(10),
    Category VARCHAR(20),
    Listed_price NUMBER(10) NOT NULL
);
```

6. To create the order table:

```
CREATE TABLE order(
    order_id VARCHAR(10) PRIMARY KEY,
    order_status VARCHAR(20) NOT NULL,
    total_amount NUMBER(10) NOT NULL,
    delivery_date DATE,
    Placed_on DATE NOT NULL,
    Delivery_addr VARCHAR(50),
    userId VARCHAR(10) NOT NULL,
     FOREIGN KEY (userId) REFERENCES USER (userId) ON DELETE
CASCADE,
);
```

7. To create the relational table between order and product tables:

```
CREATE TABLE order_product_list(
    order_id VARCHAR(10) NOT NULL,
    product_id VARCHAR(10) NOT NULL,
     FOREIGN KEY (product_id) REFERENCES PRODUCT(product_id) ON
DELETE CASCADE,
     FOREIGN KEY (oder_id) REFERENCES ORDER(order_id) ON DELETE
CASCADE
);
```

8. To create the shipment table:

```
CREATE TABLE shipment(
    shipment_id INTEGER PRIMARY KEY,
    order_id VARCHAR(10) NOT NULL,
    userId VARCHAR(10) NOT NULL,
     FOREIGN KEY (userId) REFERENCES USER (userId) ON DELETE
CASCADE,
    FOREIGN KEY (order_id) REFERENCES ORDER(order_id) ON DELETE
CASCADE
);
```

9. To create the Favourites table:

```
CREATE TABLE favourites(
    collection_id VARCHAR(10) PRIMARY KEY,
    userId VARCHAR(10) NOT NULL,
      FOREIGN KEY (userId) REFERENCES USER (userId) ON DELETE
CASCADE,
);
```

10. To create the relational table between favourites and product tables:

```
CREATE TABLE favourited_products(
    collection_id VARCHAR(10) NOT NULL,
    product_id VARCHAR(10) NOT NULL,
      FOREIGN KEY (product_id) REFERENCES USER (product_id) ON
DELETE CASCADE,
                FOREIGN    KEY    (collection_id)    REFERENCES
FAVOURITES(collection_id) ON DELETE CASCADE
);
```

11. To create the feedback table:

```
CREATE TABLE feedback(
    feedback_id VARCHAR(10) PRIMARY KEY,
    userId VARCHAR(10) NOT NULL,
    product_id VARCHAR(10) NOT NULL,
    feedback VARCHAR(50) NOT NULL,
      FOREIGN KEY (userId) REFERENCES USER (userId) ON DELETE
CASCADE,
     FOREIGN KEY (product_id) REFERENCES PRODUCT(product_id) ON
DELETE CASCADE
);
```

12. To create the shopping cart table:

```
CREATE TABLE shopping_cart(

    cart_id VARCHAR(10) PRIMARY KEY,

    userId VARCHAR(10) NOT NULL,

    order_id VARCHAR(10) NOT NULL,

      FOREIGN  KEY  (userId)  REFERENCES  USER  (userId)  ON  DELETE
CASCADE,

     FOREIGN KEY (order_id) REFERENCES ORDER(order_id) ON DELETE
CASCADE

);
```

13. To create the relational table between shopping cart and product tables:

```
CREATE TABLE cart_product_list(

    cart_id VARCHAR(10) NOT NULL,

    product_id VARCHAR(10) NOT NULL,

     FOREIGN  KEY  (product_id)  REFERENCES  PRODUCT(product_id)  ON
DELETE CASCADE,

     FOREIGN  KEY  (cart_id)  REFERENCES  SHOPPING_CART(cart_id)  ON
DELETE CASCADE

);
```

14. To create the relational table between user(basically a seller) and product tables:

```
CREATE TABLE sells(

    product_id VARCHAR(10) NOT NULL,

    userId VARCHAR(10) NOT NULL,

     FOREIGN  KEY  (product_id)  REFERENCES  PRODUCT(product_id)  ON
DELETE CASCADE,

       FOREIGN  KEY  (userId)  REFERENCES  USER(userId)  ON  DELETE
CASCADE

);
```

7. Procedures and Triggers:

7.1 Procedures:

1. **Register a user**: This is a generic procedure, to be invoked while registering a user. The flags that specify if a user is a user is a seller or a buyer are set to 1 by default as *Esty* allows a user to buy and sell using the same account.

```
CREATE
OR REPLACE PROCEDURE registerUser(
    userId IN VARCHAR,
    email  IN VARCHAR,
    first_name IN VARCHAR,
    last_name IN VARCHAR,
    primary_phone IN NUMBER,
    addr in VARCHAR,
    pswd IN VARCHAR
) AS BEGIN
INSERT INTO
    USER
VALUES
    (
        userId,
        email,
        first_name,
        last_name,
        primary_phone,
        addr,
        pswd,
        1,
        1
    ); END registerUser;
```

2. **Add payment details**: This procedure adds the payment information of the user.

```
CREATE
OR REPLACE PROCEDURE addPaymentDetails(
    card_id IN INTEGER,
    expiry IN DATE,
    cvv IN NUMBER,
    zipcode IN NUMBER,
    userId IN VARCHAR
) AS BEGIN
INSERT INTO
    cardDetails
VALUES
    (
        card_id,
        expiry,
        cvv,
        zipcode,
        userId
    );
END addPaymentDetails;
```

3. **Order**: When invoked, this procedure iterates over all the products in the shopping cart, calculates the total amount and places an order ie. inserts the data in the order table.

```
CREATE

OR REPLACE PROCEDURE order (

    v_orderId IN VARCHAR,

    v_userId IN VARCHAR,

    v_shippingPrice IN NUMBER,

    v_deliveryDate IN DATE

) AS v_address INTEGER;

v_listPrice NUMBER := 0;

v_cartTotal NUMBER := 0;

CURSOR prodId IS

SELECT

    cart_product_list.product_id

FROM

    cart_product_list

      INNER JOIN shopping_cart ON cart_product_list.cart_id =
shopping_cart.cart_id

WHERE

    userId = v_userId;

pId VARCHAR;

BEGIN OPEN prodId;

LOOP FETCH prodId INTO pId;

EXIT

WHEN prodId % notfound;

SELECT

    Listed_price INTO v_listPrice

FROM

    product

WHERE
```

```
    product_id = pId;
v_cartTotal := (v_cartTotal + v_listPrice);
INSERT INTO
    order_product_list
VALUES
    (v_orderId, pId);
END IF;
END LOOP;
CLOSE prodId;
SELECT
    addr INTO v_address
FROM
    user
WHERE
    userId = v_userId;

v_cartTotal := v_cartTotal + v_shippingPrice;
INSERT INTO
    order
VALUES
    (
        v_orderId,
        'Order Placed',
        v_cartTotal,
        v_deliveryDate,
        sysdate,
        v_address,
        v_userId,
    );
END order;
```

7.2 Triggers

1. **Update Quantity**: Once an order is placed, this trigger will be triggered and the quantity of the item in the product table will be decreased.

```
CREATE
OR REPLACE TRIGGER updateQuantity
AFTER
INSERT
    ON order
    FOR EACH ROW
DECLARE
v_productId INTEGER;
v_quantity INTEGER;
CURSOR prodId IS
SELECT
    product_id
FROM
    order_product_list
WHERE
    order_id = :new.order_id;
BEGIN OPEN prodId;
LOOP FETCH prodId INTO v_productId;
EXIT
WHEN prodId % notfound;
SELECT
    Quantity INTO v_quantity
FROM
    product
WHERE
    product_id = v_productId;
```

```
UPDATE
    product
SET
    Quantity = Quantity - 1
WHERE
    product_id = v_productId;
END LOOP;
CLOSE prodId;
END updateQuantity;
```

2. **Empty Cart:** Once an order is placed, this trigger will be invoked and the cart will be emptied.

```
CREATE
OR REPLACE TRIGGER emptyCart
AFTER
INSERT
    ON order
    FOR EACH ROW
    DECLARE
BEGIN
DELETE FROM
    shopping_cart
WHERE
    cart_id = :new.cart_id;
DELETE FROM
    cart_product_list
WHERE
    cart_id = :new.cart_id;
END;
```