EGE UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

INTRODUCTION TO DATABASES

2024-2025 FALL SEMESTER

PROJECT REPORT

E-commerce Database Integration

DELIVERY DATE

14/01/2025

PREPARED BY

05220000291, Emre Bahçeci

05220000322, Emin Özgür Elmalı

05220000333, Ajenk Göktuğ Kamalı

05220000372, Arda Gülnaz

1.ANALYSIS

Here are the data requirements for each website:

AMAZON

Customers form the foundation of user management, where each customer has a unique identifier and comprehensive personal information including email address, password (stored in encrypted format), full name and phone number.

The system maintains multiple addresses for each customer with unique identifier. Each address record containing detailed location information (Street, BuildingNo, ApartmentNo, City, Country, PostalCode, and AdressLine).

Each product has a unique identifier, name and belongs to a specific category within a hierarchical category structure. Categories have a unique identifier, name and may have a parent category to establish the hierarchy. Products maintain detailed information including description about product which has product name, release date, manufacturer name, image url, and description.

Product variants (shop products) are managed by each variant has a unique SKU relates to a base product and includes specific attributes such as quantity. Each variant maintains its own pricing information and stock levels.

The shop represents merchants operating on the platform. Each shop has a unique identifier, business type, business name, contact information, tax id, selling plan, and registration date.

Orders are processed where each order has a unique identifier and is associated with a customer. Orders contain essential information including order date, discount, gift information, and total price.

Order items maintain the relationship between orders and products. The warehouse system is structured with multiple warehouses across different locations.

Each warehouse has a unique identifier, location details, capacity information, and current inventory levels.

Warehouses maintain their own stock of Amazon-owned products and handle specific geographic regions for efficient delivery. Each warehouse tracks inbound and outbound

shipments, maintains minimum stock levels, and manages storagesections by product category.

Customer engagement is facilitated through the reviews, where customers can provide feedback on products. Each review has a unique identifier, rating, review text and submission date. The platform also maintains customer wishlists allowing customers to save products for future reference, holded with a name and collaborators.

Lists system allows customers to create and manage multiple types of lists including wish lists, shopping lists, and custom lists. Each list has a unique identifier, name, privacy setting (public/private), creation date, and can contain multiple products. Lists can be shared with other users and support collaboration features.

The platform's logistics are managed through the shipment, where each shipment has a unique identifier, delivery type, carrier information, cargo company, estimated delivery and current status. Each shipment is associated with specific order items and includes delivery estimates and actual delivery confirmation. The system includes delivery points for convenient package pickup, each with unique identifier, location details, and operating hours.

Payment processing is handled by recording transaction details and payment method. Customer pays their order before the shipment has done.

TRENDYOL

Customer serves as the foundation of the user experience, where each customer has a unique identifier and maintains comprehensive profile information including email address, encrypted password, full name and mobile phone number.

The system supports multiple delivery addresses, recording detailed location data with unique identifier (Street, Town, District, City, Country) and delivery instructions.

Merchants are central to the marketplace operations, where each merchant has a unique identifier, company name, and comprehensive business information including tax identification number, registration date details and company type information.

Product organization is handled through the category, implementing a hierarchical structure where each category has a unique identifier, name, and optional parent category.

Product management maintain comprehensive product information, where each product has a unique identifier, name, barcode, image url, and description. Products in some categories include additional attributes.

Product listings manages specific variations of products where each variant has a unique SKU and includes stock level.

Export product system integrates with Alibaba's global marketplace, allowing merchants to list their products internationally. Each export listing contains product details, international pricing, shipping options, and compliance information for target markets. The system maintains export documentation, customs requirements, and international shipping arrangements.

Orders have transaction processing, where each order has a unique identifier and may contain items from multiple merchants. Orders maintain comprehensive information including order date, status, discount and total price. The system supports split deliveries through the order shipment, managing separate shipments for items from different merchants.

The payment handles financial transactions, like payment method or a card

information. There might be multiple cards contained by the system.

Customer engagement is facilitated through the shop product (product variant) reviews. Each review includes a rating, submission date and review text.

The shipping manages delivery operations, maintaining relationships with multiple carrier services and cargo companies. The system supports different shipping methods with a unique identifier.

Cart holds products and quantities of those products in a single cart session.

HEPSIBURADA

User system forms the core of customer management, where each user maintains a unique identifier and comprehensive profile information including email address, securely encrypted password, full name, mobile phone number.

Address management system maintains multiple adresses for each user, storing detailed location information with unique identifier (Street, BuildingNo, ApartmentNo, Neighborhood, Description, Town, District, City, PostalCode, AddressTitle, Country) and specific delivery instructions. The system supports address verification and optimization through integration with postal services.

Seller framework manages marketplace operations, where each seller possesses a unique identifier and maintains comprehensive business information including company name, tax identification, business registration details, and verified contact information.

Brand management system maintains comprehensive brand information where each brand

has a unique identifier, name, logo, description, and contact information. Brands can be associated with multiple products and maintain their own brand pages with custom content and promotional materials.

Product organization is handled through the categories, implementing a detailed hierarchy where each category has a unique identifier, name and parent category.

The system provides specialized attribute sets for different product types.

Product management system maintains comprehensive product information, where each product has a unique identifier, name and detailed technical specifications.

Order processing is managed through the order system, where each order has a unique identifier and may contain items from multiple sellers. The system maintains comprehensive order information including order date. And also maintains transactional information like total price and discounts. The platform supports split shipments managing separate deliveries for items from different sellers.

Payment framework handles financial transactions, recording payment method with payment options like credit cards.

Customer engagement is facilitated through the reviews, where customers can provide product ratings. Each review includes a rating detailed feedback and submission date. The system maintains product ratings and review to provide comprehensive quality metrics.

Logistic system manages delivery operations through integration with multiple carrier services. The platform supports various shipping methods including same-day delivery and scheduled delivery maintaining separate records for each delivery type. Also maintains unique identifier with informations like status and cargo company for each shipment.

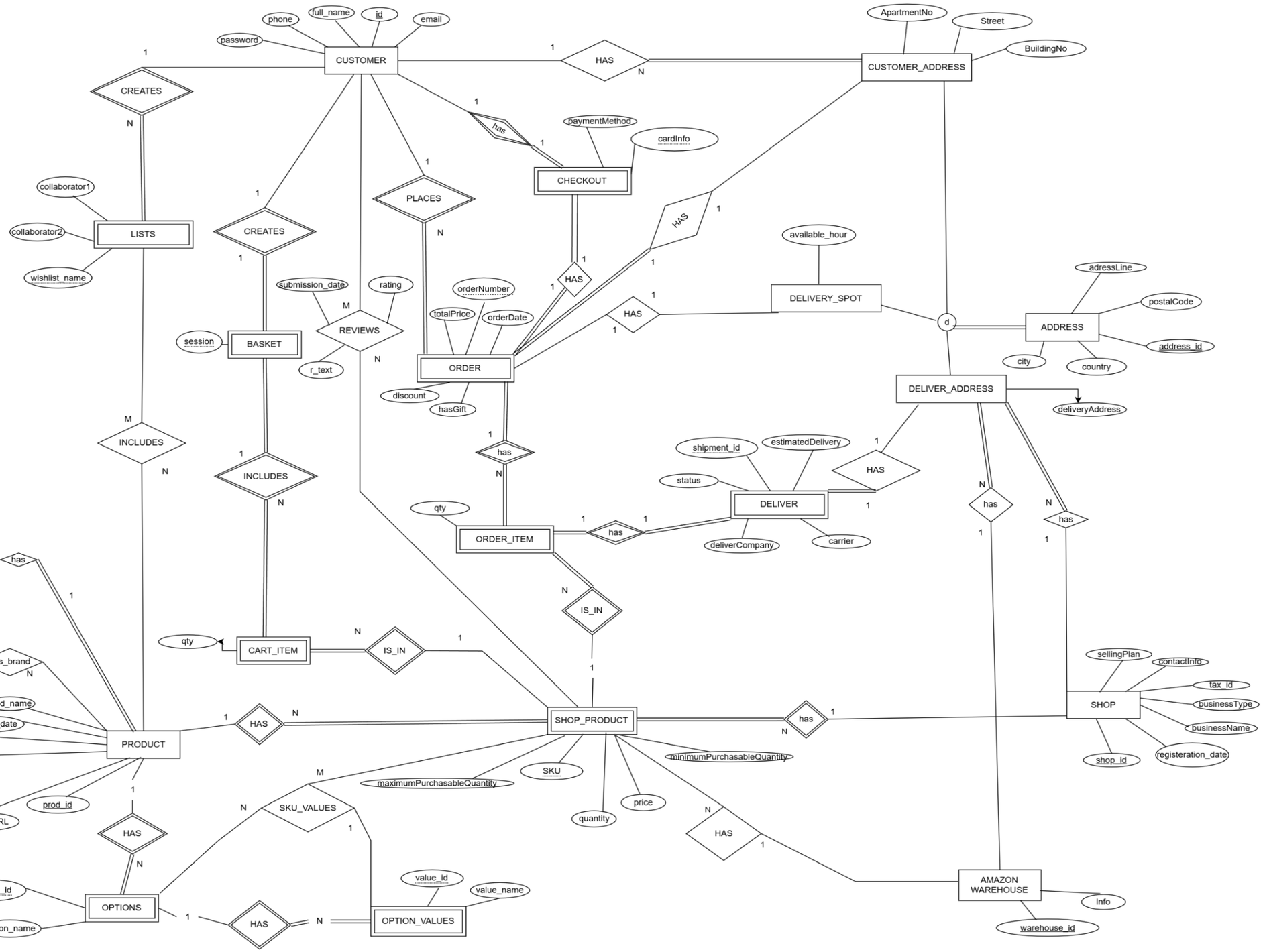
This page intentionally left blank.

2.DESIGN

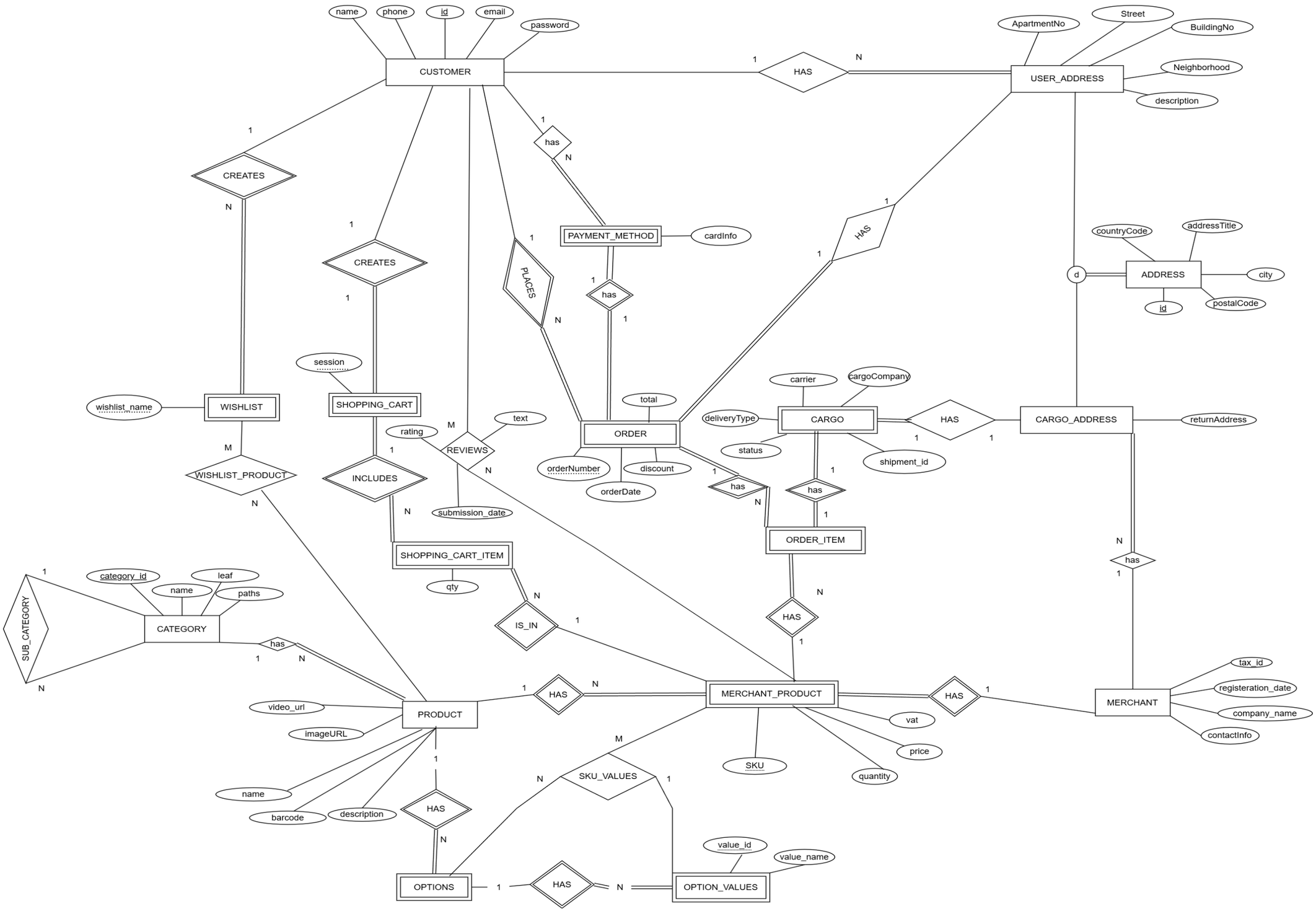
2.a Conceptual Design

Here are the EER diagrams for each website according to our data requirements:

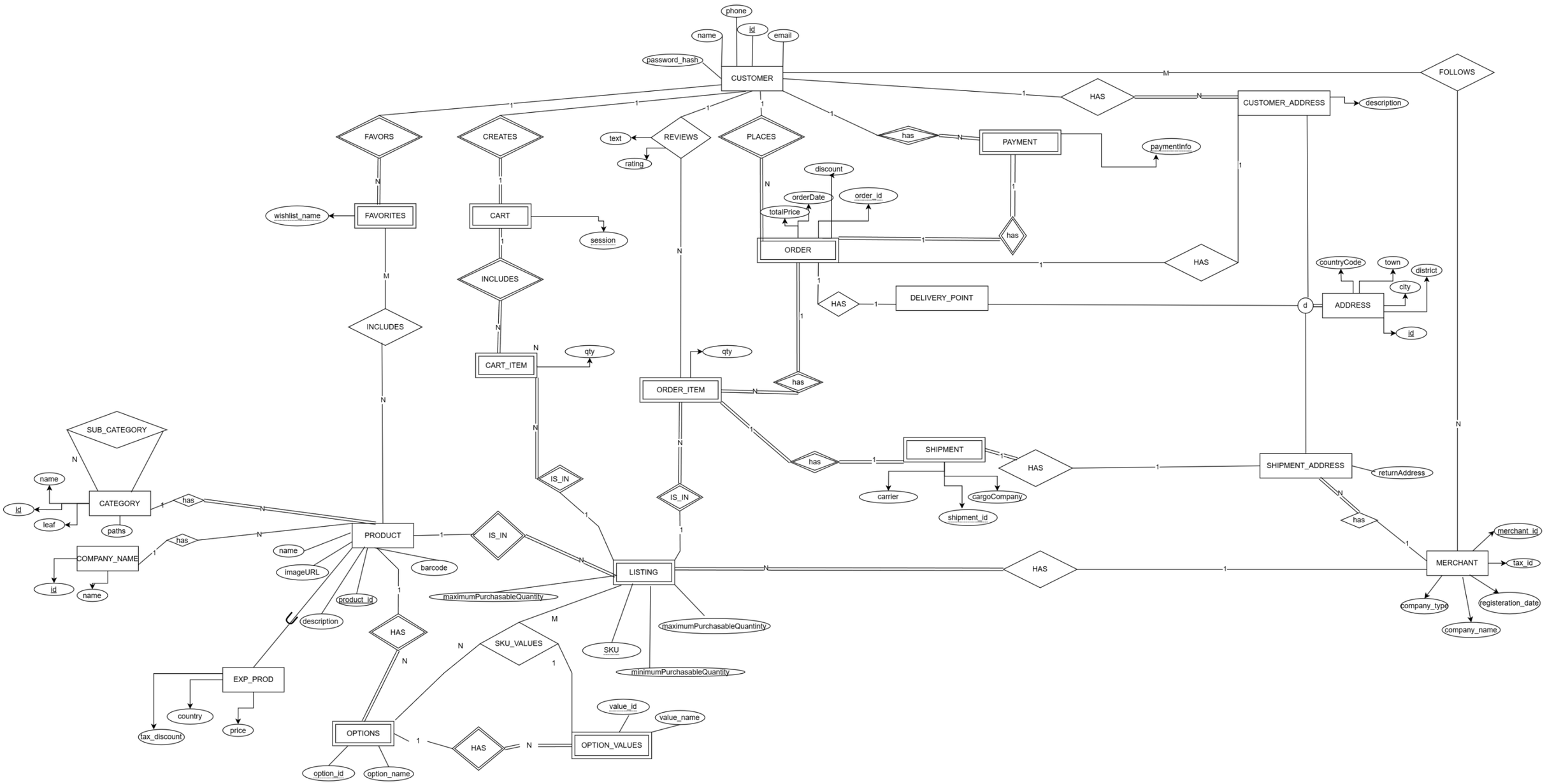
AMAZON



HEPSIBURADA



TRENDYOL


phone
id
name
email
password_hash
CUSTOMER
FOLLOWS
HAS
CUSTOMER_ADDRESS
description
FAVORS
CREATES
text
REVIEWS
rating
PLACES
has
PAYMENT
paymentInfo
discount
orderDate
order_id
totalPrice
ORDER
has
HAS
wishlist_name
FAVORITES
CART
session
INCLUDES
INCLUDES
CART_ITEM
qty
qty
ORDER_ITEM
has
HAS
DELIVERY_POINT
countryCode
town
district
city
ADDRESS
id
SUB_CATEGORY
name
id
CATEGORY
leaf
paths
has
IS_IN
IS_IN
has
SHIPMENT
carrier
cargoCompany
shipment_id
HAS
SHIPMENT_ADDRESS
returnAddress
has
COMPANY_NAME
id
name
has
PRODUCT
name
imageURL
barcode
product_id
description
IS_IN
HAS
LISTING
maximumPurchasableQuantity
maximumPurchasableQuantity
SKU_VALUES
SKU
minimumPurchasableQuantity
HAS
MERCHANT
merchant_id
tax_id
company_type
registeration_date
company_name
EXP_PROD
country
price
tax_discount
OPTIONS
option_id
option_name
HAS
value_id
value_name
OPTION_VALUES

For our merge algorithm we apply the same methodology from the paper[1]. So here is a merge example step-by-step explaining the approach between Hepsiburada and Amazon EER diagrams:

Create a connector based on mappings between the two input ER diagrams and populate mappings between connector to each input ER diagram

This is done because our use of connector models was motivated by theoretical concerns. Briefly, this approach allows us to build the mappings between models using graph homomorphisms – each mapping shows how one model is embedded in another[2].

Connector models are needed when merging ER diagrams for two key reasons:

1. They provide a mathematical foundation through graph homomorphisms, ensuring that merges are provably unique and valid
2. They explicitly capture how different models overlap, making it easier to handle complex scenarios where multiple models share common elements or when new stakeholders need to build upon existing shared components

(You can find mapping and connector model in the next pages.)

2) Now start merging set. We categorize the elements in ER diagram into four different types (or levels), they are:

- Entity
- Relationship
- Entity Attribute
- Relationship Attribute

Create a disjoint union for each element type based on the left ER diagram, the right ER diagram and the connector. Here is the example for entities:

$U_{Entities}$ = {A_CUSTOMER, C_CUSTOMER, H_CUSTOMER, A_BASKET, C_CART, H_SHOPPING_CART, A_LIST, H_WISHLIST, C_WISHLIST, A_CART_ITEM, C_CART_ITEM, H_SHOPPING_CART_ITEM, A_PRODUCT, C_PRODUCT, H_PRODUCT, A_SHOP_PRODUCT, C_MERCHANT_PRODUCT, H_MERCHANT_PRODUCT, A_OPTIONS, C_OPTIONS, H_OPTIONS, A_OPTION_VALUES, C_OPTION_VALUES, H_OPTION_VALUES, A_ORDER_ITEM, C_ORDER_ITEM, H_ORDER_ITEM, A_ORDER, C_ORDER, H_ORDER, A_DELIVER, C_CARGO, H_CARGO, A_DELIVER_ADDRESS, C_CARGO_ADDRESS, H_CARGO_ADDRESS, A_DELIVERY_SPOT, A_ADDRESS, C_ADDRESS, H_ADDRESS, A_AMAZON_WAREHOUSE, A_SHOP, C_MERCHANT, H_MERCHANT, A_CUSTOMER_ADDRESS, C_CUSTOMER_ADDRESS, H_USER_ADDRESS, A_CHECKOUT, H_PAYMENT_METHOD,C_ PAYMENT_METHOD , A_CATEGORY,C_CATEGORY,H_CATEGORY,A_BRAND}

Then we use an merging set algorithm to group entities together using mapping we provided via connector:

P_U = {{A_CUSTOMER, C_CUSTOMER, H_CUSTOMER}, {A_BASKET, C_CART, H_SHOPPING_CART}, {A_LIST, H_WISHLIST, C_WISHLIST}, {A_CART_ITEM, C_CART_ITEM, H_SHOPPING_CART_ITEM},{ A_PRODUCT, C_PRODUCT, H_PRODUCT}, {A_SHOP_PRODUCT, C_MERCHANT_PRODUCT, H_MERCHANT_PRODUCT}, {A_OPTIONS, C_OPTIONS, H_OPTIONS}, {A_OPTION_VALUES

C_OPTION_VALUES, H_OPTION_VALUES},{A_ORDER_ITEM, C_ORDER_ITEM, H_ORDER_ITEM}, {A_ORDER, C_ORDER, H_ORDER}, {A_DELIVER, C_CARGO, H_CARGO}, {A_DELIVER_ADDRESS, C_CARGO_ADDRESS, H_CARGO_ADDRESS}, {A_DELIVERY_SPOT}, {A_ADDRESS, C_ADDRESS, H_ADDRESS}, {A_AMAZON_WAREHOUSE},{A_SHOP, C_MERCHANT, H_MERCHANT}, {A_CUSTOMER_ADDRESS, C_CUSTOMER_ADDRESS, H_USER_ADDRESS}, {A_CHECKOUT, H_PAYMENT_METHOD, C_ PAYMENT_METHOD }, {A_CATEGORY,C_CATEGORY,H_CATEGORY},{A_BRAND}}

And for this part we should choose an element from every list for distinct set,

$P_{Entities}$ = {CUSTOMER, CUSTOMER_ADDRESS, ADDRESS, CART, WISHLIST, ORDER, CARGO_ADDRESS, CART_ITEM, ORDER_ITEM, CARGO, CATEGORY, PRODUCT, MERCHANT_PRODUCT, MERCHANT, OPTIONS, OPTIONS_VALUES, CHECKOUT, BRAND,AMAZON_WAREHOUSE,DELIVERY_SPOT}

By the same logic we create set for distinct relationships, entity attributes and relationship attributes:

$P_{Relations}$ = {CREATES_WL, CREATES_CART, PLACES, ADDRESS_OF_C, ADDRESS_OF_M, SHIPPED_TO, SHIPS, GOES_TO, CART_INCLUDES, REVIEWS, INCLUDES_ITEMS, WISHLIST_PRODUCT, CATEGORIZED, IN_CART,IN_ORDER, LISTED_AS, LISTS, HAS_OPT, HAS_VAL, SKU_VALUES,WAREHOUSE_LOC, IN_WAREHOUSE, DELIVERED_AT, CHECKSOUT,CHECKING_OUT,HAS_BRAND,SUBCATEGORY,IN_WAREHOUSE,WAREHOUSE_LOG}

$P_{EntityAtttributes}$ = {

CUSTOMER{name,phone,id,email,password},

CUSTOMER_ADDRESS{ApartmentNo,Street,BuildingNo,Neighborhood,description,defaultAddress}

ADDRESS{addressTitle,addressLine1,addressLine2,postalCode,address_id,

countryCode,district,town,city}

CART{session}

WISHLIST{wishlist_name,collabrator1,collabrator2}

ORDER{discount,orderDate,totalPrice,hasGift,orderNumber}

CARGO_ADDRESS{returnAddress(deliveryAddress)}

CART_ITEM{qty}

ORDER_ITEM{isGift,qty}

CARGO{status,deliveryType,carrier,cargoCompany,trackingNumber,shipment_id,

estimatedDelivery}

CATEGORY{category_id,category_name,leaf,paths}

PRODUCT{videoURL,imageURL,product_id,product_name,barcode,description,release_date,manufacturer_name,ASIN}

MERCHANT_PRODUCT{SKU,commisionRate,quantity,price,maximumPurchasableQuantity,minimumPurchasableQuantity,vat}

MERCHANT{tax_id,registeration_date,company_name,contactInfo,merchantID,sellingPlan,businessType}

OPTIONS{option_id,option_value}

OPTIONS_VALUES{value_id,value_name}

CHECKOUT{cardInfo,paymentMethod}

BRAND{brand_name,id}

AMAZON_WAREHOUSE{wh_Loc,warehouse_id,wh_info}

DELIVERY_SPOT{available_hour}

}

$P_{RelationshipAttributes}$ = {REVIEWS{rating,submission_date,r_text}}

Now we start merging graph. We create an empty ER diagram D_{merged}, and then add merged elements to this ER diagram type by type, in the following order:

- Entity type: get all the entities from $P_{distinct}$, add them to D_{merged}.

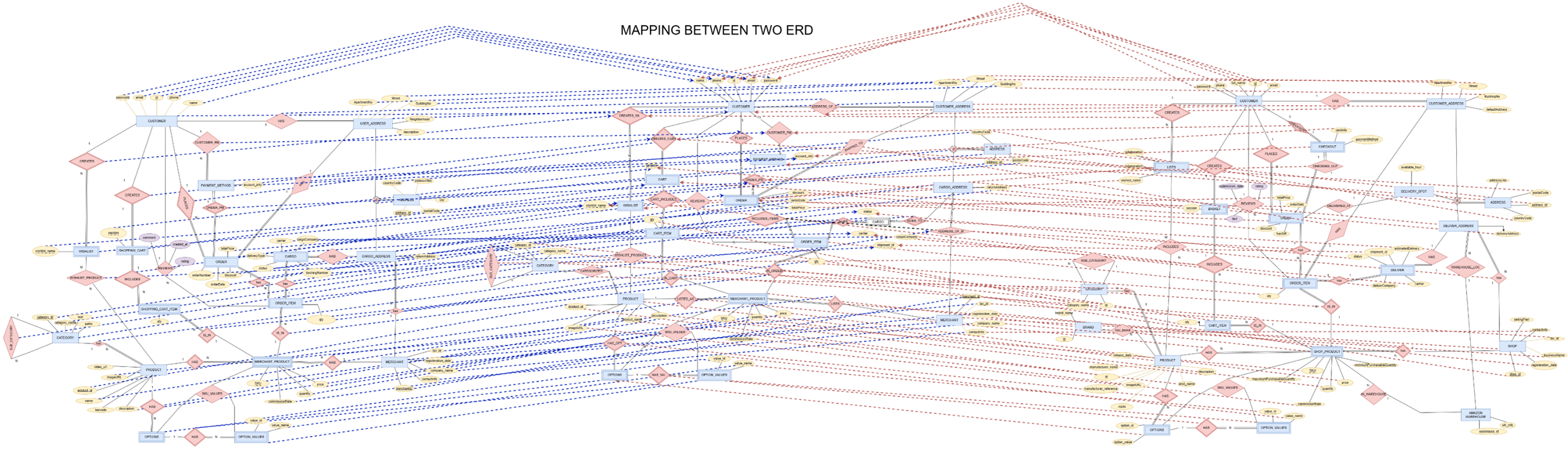
- Relationship type: get all the relationships from $P_{distinct}$, add them to D_{merged}. For each relationship, check the existence of from entity and to entity in D_{merged}, if not exists, join P to get it and set it to the relationship.

- Entity Attribute type: for each entity attribute in $P_{distinct}$, check the existence of its associated entity in D_{merged}, if exists, associate it to that entity; else, join P to get the attribute and associate it to the corresponding entity in D_{merged}.

- Relationship Attribute type: for each relationship attribute in $P_{distinct}$, check the existence of its associated relationship in D_{merged}, if exists, associate it to that relationship; else, join P to get the attribute and associate it to the corresponding relationship in D_{merged}

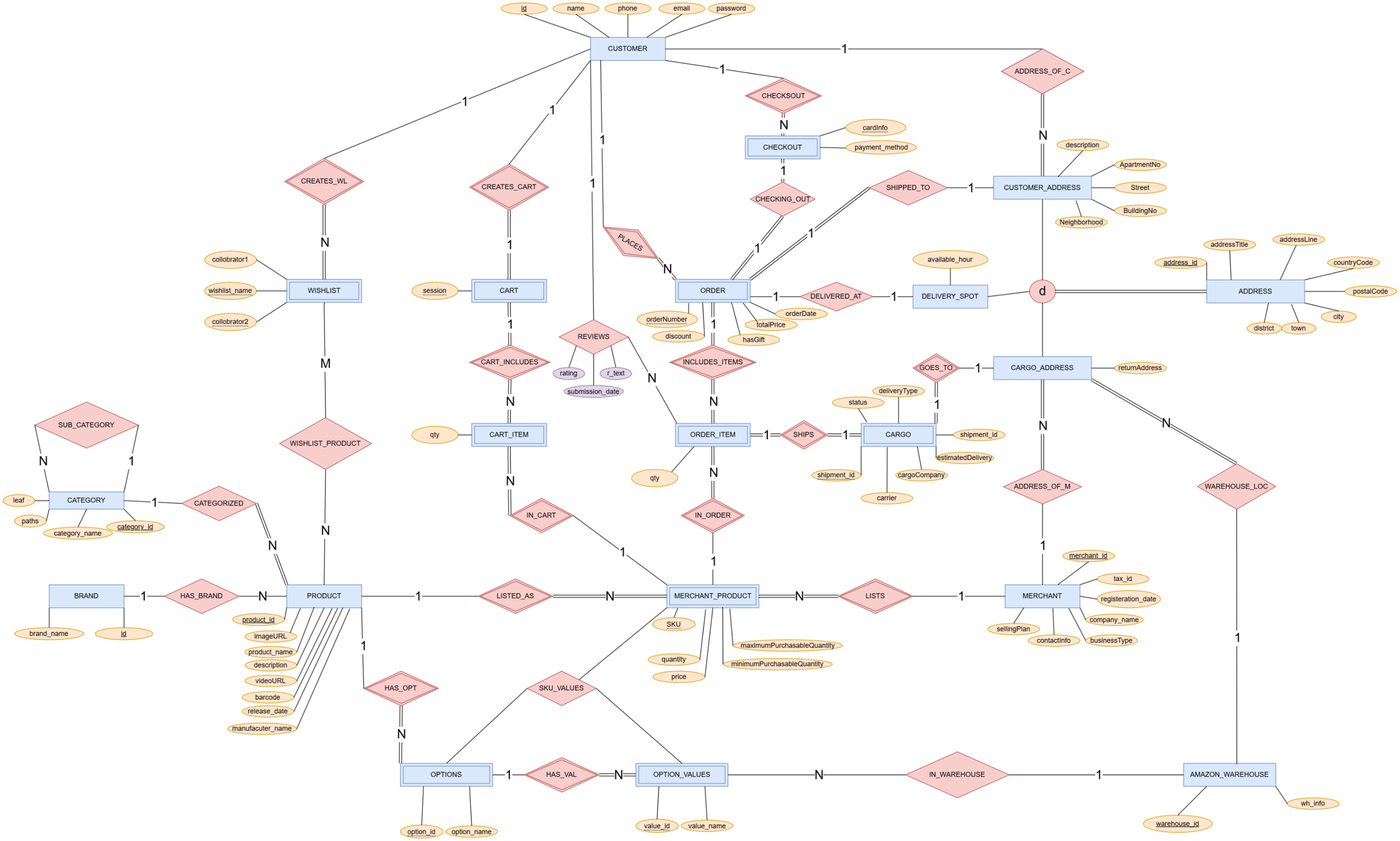
[1]https://www.semanticscholar.org/paper/An-Implementation-of-Entity-Relationship-Diagram-He/87c52a9d46ead735a1610373becd37dc089ccd23
[2] Mehrdad Sabetzadeh. Merging and Consistency Checking of Distributed Models. A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy. 2008

MAPPING BETWEEN TWO ERD

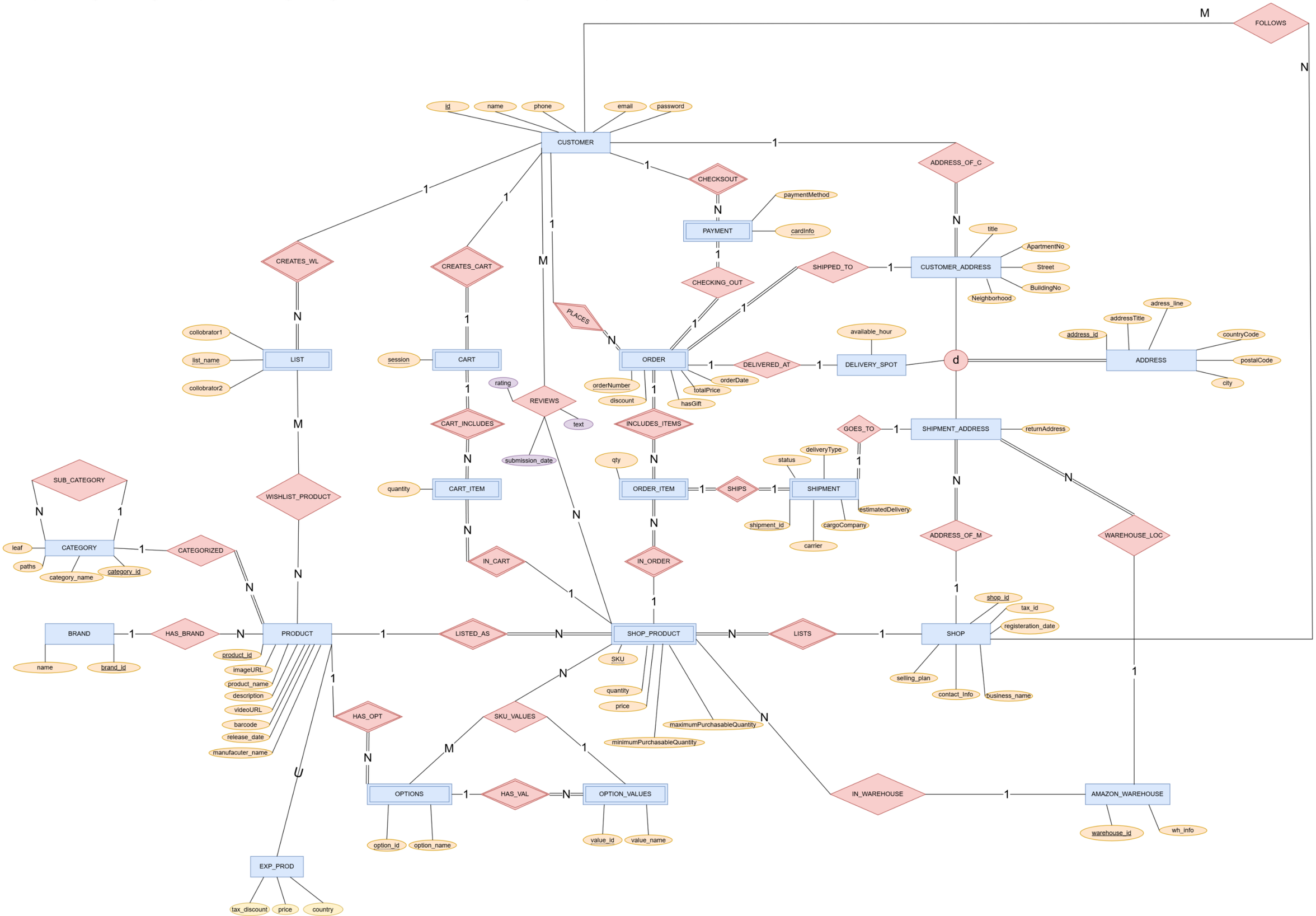


This is the connector model and mappings between Amazon and Hepsiburada EER diagram

So by the end of this part for our example we get HA_Merged EER:



By following same algorithm, the final merged diagram for all three of the EER diagrams corresponds to this:



2.b Logical Model

Here is the step-by-step conversion of merged EER diagram using methodology:

1st Iteration

1. **CUSTOMER**(customer_id, email, password, phone, full_name)

   BRAND(brand_id, brand_name)

   SHOP(shop_id, tax_id, contact_info, business_name, registration_date, sellingPlan)

   CATEGORY(category_id, category_name, leaf, paths)

   PRODUCT(product_id, release_date, manufacturer_name, prod_name, imageURL, videoURL, barcode, description)

   WAREHOUSE(warehouse_id, info)

   ADDRESS(address_id, address_title, address_line, city, country_code, postal_code)

2. **OPTIONS**(product_id, option_id, option_name)

   PAYMENT(customer_id, card_info, paymentMethod)

   SHOP_PRODUCT(product_id, shop_id, sku, price, quantity, max_purchasable_qty, min_purchasable_qty)

   CART(customer_id, session)

   LIST(customer_id, list_name, collaborator1, collaborator2)

   ORDER(CUSTOMER.customer_id, order_number, total_price, order_date, discount, has_gift)

3. **ORDER**(CUSTOMER.customer_id, order_number, PAYMENT.card_info, total_price, order_date, discount, has_gift)

4. **PRODUCT**(product_id, release_date, manufacturer_name, prod_name, imageURL, videoURL, barcode, description, BRAND.brand_id, CATEGORY.category_id)

   SHOP_PRODUCT(PRODUCT.product_id,SHOP. shop_id, SHOP.sku, price, quantity, max_purchasable_qty, min_purchasable_qty, WAREHOUSE.warehouse_id)

   CATEGORY(category_id, category_name, leaf, paths, CATEGORY.parent_category_id)

5. **REVIEWS**(CUSTOMER.customer_id, PRODUCT.product_id, PRODUCT.shop_id, PRODUCT.sku, submission_date, rating, text)

   WISHLIST_PRODUCT(LIST.customer_id, PRODUCT.product_id, LIST.wishlist_name)

   FOLLOWS(CUSTOMER.customer_id, SHOP.shop_id)

6. -
7. -

8. **ADDRESS**(address_id, address_title, address_line, city, country_code, postal_code, addressType, street, building_no, neighborhood, title, apartment, return_address, available_hours)

EXPORT_PRODUCT(PRODUCT.product_id, price, tax_discount, country)

9. -

2nd Iteration

1. -
2. **OPTION_VALUES**(PRODUCT.product_id, OPTIONS.option_id, value_id, value_name)

   CART_ITEM(CART.customer_id, CART.session, SHOP.shop_id, SHOP_PRODUCT.sku, quantity

   ORDER_ITEM(CUSTOMER.customer_id, ORDER.order_number, PRODUCT.product_id, SHOP_PRODUCT.sku, SHOP.shop_id, qty)

3. **ORDER**(CUSTOMER.customer_id, order_number, PAYMENT.card_info, total_price, order_date, discount, has_gift, ADDRESS.customer_address, ADDRESS.delivery_spot)

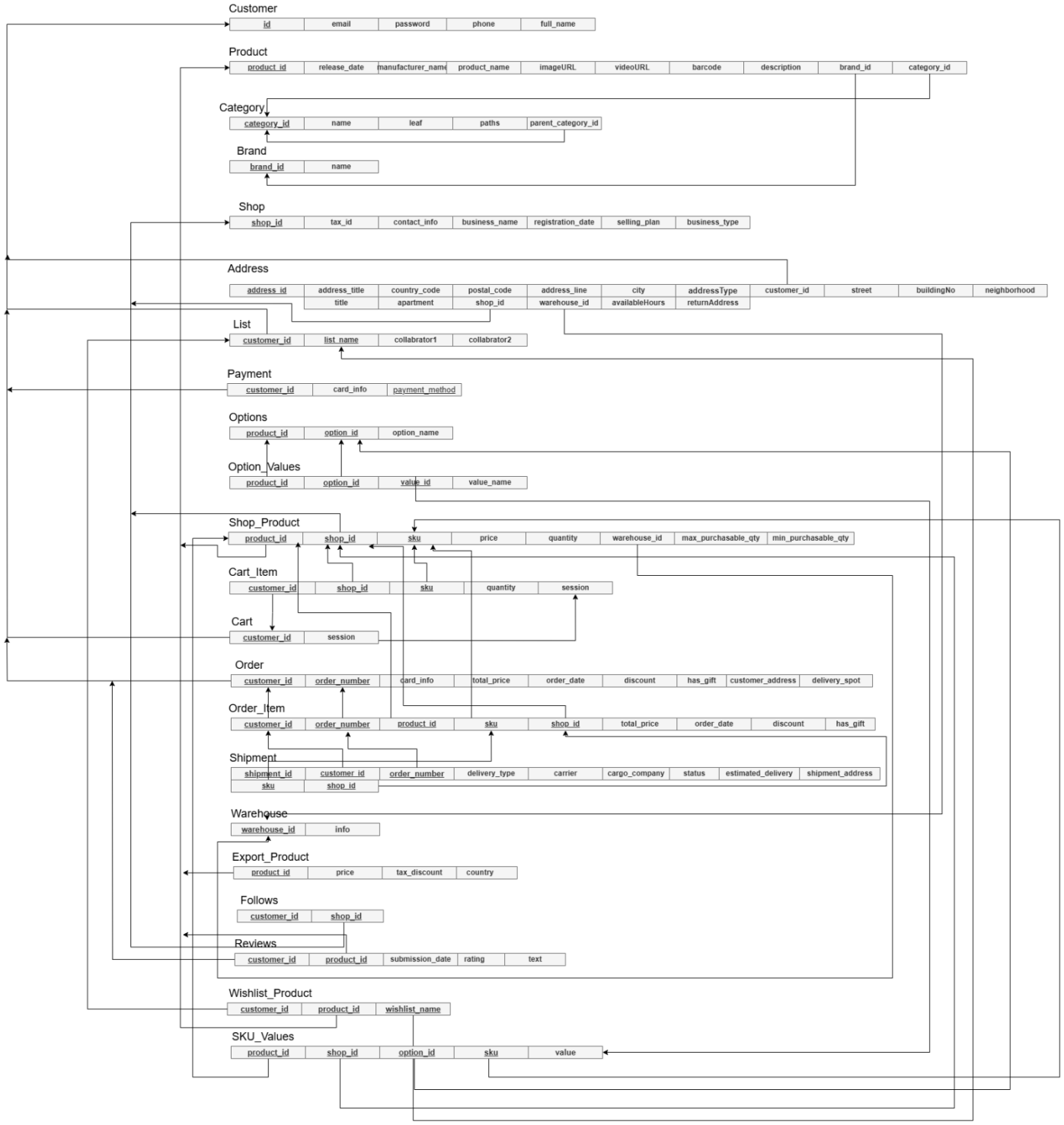
4. **ADDRESS**(address_id, address_title, address_line, city, country_code, postal_code, addressType, CUSTOMER.customer_id, street, building_no, neighborhood, title, apartment, SHOP.shop_id, WAREHOUSE.warehouse_id, return_address, available_hours)
5. -
6. **-**
7. **SKU_VALUES**(PRODUCT.product_id, SHOP.shop_id, OPTIONS.option_id, SHOP_PRODUCT.sku, OPTION_VALUES.value)
8. -
9. -

3rd Iteration

1. -
2. **SHIPMENT**(shipment_id, CUSTOMER.customer_id, , ORDER.order_number, PRODUCT.product_id, deliveryType, carrier, cargo_company, status, estimated_delivery, SHOP_PRODUCT.sku, SHOP.shop_id)
3. **SHIPMENT**(shipment_id, CUSTOMER.customer_id, ORDER.order_number, PRODUCT.product_id, deliveryType, carrier, cargo_company, status, ADDRESS.shipment_address, estimated_delivery, SHOP_PRODUCT.sku, SHOP.shop_id)

4-9. -

Final look of the relational database and referential integrity constraints:



3.IMPLEMENTATION

3.a DDL Statements

```
-- 1) Drop tables with foreign keys first to avoid dependency conflicts

DROP TABLE IF EXISTS SKU_Values;

DROP TABLE IF EXISTS Wishlist_Product;

DROP TABLE IF EXISTS Reviews;

DROP TABLE IF EXISTS Follows;

DROP TABLE IF EXISTS Export_Product;

DROP TABLE IF EXISTS Shipment;

DROP TABLE IF EXISTS Order_Item;

DROP TABLE IF EXISTS "ORDER";

DROP TABLE IF EXISTS Cart_Item;

DROP TABLE IF EXISTS Cart;

DROP TABLE IF EXISTS Shop_Product;

DROP TABLE IF EXISTS Option_Values;

DROP TABLE IF EXISTS "OPTIONS";

DROP TABLE IF EXISTS Payment;

DROP TABLE IF EXISTS List;

DROP TABLE IF EXISTS Address;


-- Drop base tables last

DROP TABLE IF EXISTS Shop;

DROP TABLE IF EXISTS Warehouse;

DROP TABLE IF EXISTS Product;

DROP TABLE IF EXISTS Category;

DROP TABLE IF EXISTS Brand;

DROP TABLE IF EXISTS Customer;


-- Drop types

DROP TYPE IF EXISTS order_status CASCADE;
```

```
DROP TYPE IF EXISTS selling_plan CASCADE;
DROP TYPE IF EXISTS payment_method CASCADE;
DROP TYPE IF EXISTS shipment_status CASCADE;
DROP TYPE IF EXISTS delivery_type CASCADE;
DROP TYPE IF EXISTS address_type CASCADE;
CREATE TYPE address_type AS ENUM ('Customer', 'Delivery', 'Shipment');
CREATE TYPE delivery_type AS ENUM ('Standard', 'Express', 'Same-day', 'Overnight');
CREATE TYPE shipment_status AS ENUM ('Pending', 'Shipped', 'In Transit', 'Delivered', 'Canceled');
CREATE TYPE payment_method AS ENUM ('Credit Card', 'Debit Card', 'PayPal', 'Bank Transfer', 'Cash on Delivery');
CREATE TYPE selling_plan AS ENUM ('Basic', 'Standard', 'Premium', 'Enterprise');



-- 3) Base tables
CREATE TABLE CUSTOMER (
customer_id INTEGER PRIMARY KEY,
email VARCHAR(255) NOT NULL UNIQUE,
"password" VARCHAR(255) NOT NULL,
phone VARCHAR(50) NOT NULL,
full_name VARCHAR(255) NOT NULL
);

CREATE TABLE BRAND (
brand_id INTEGER PRIMARY KEY,
brand_name VARCHAR(255) NOT NULL
);

CREATE TABLE CATEGORY (
  category_id INTEGER PRIMARY KEY DEFAULT 0,
  category_name VARCHAR(255) NOT NULL,
  leaf BOOLEAN DEFAULT false,
```

```
    paths TEXT DEFAULT '/',
    parent_category_id INTEGER,
        CONSTRAINT category_parent_category_id_fkey
            FOREIGN KEY (parent_category_id) REFERENCES CATEGORY(category_id)
                ON DELETE CASCADE  ON UPDATE CASCADE
);

CREATE TABLE PRODUCT (
    product_id INTEGER PRIMARY KEY,
    release_date DATE,
    manufacturer_name VARCHAR(255),
    prod_name VARCHAR(255) NOT NULL,
    imageURL TEXT,
    videoURL TEXT,
    barcode INTEGER NOT NULL,
    description TEXT,
    brand_id INTEGER,
    category_id INTEGER,
        CONSTRAINT product_brand_id_fkey
            FOREIGN KEY (brand_id) REFERENCES BRAND(brand_id)
                ON DELETE CASCADE ON UPDATE CASCADE,
        CONSTRAINT product_category_id_fkey
        FOREIGN KEY (category_id) REFERENCES CATEGORY(category_id)
                ON DELETE SET DEFAULT ON UPDATE CASCADE
);
CREATE TABLE WAREHOUSE (
    warehouse_id INTEGER PRIMARY KEY,
    "info" TEXT
);

CREATE TABLE SHOP (
```

```
    shop_id INTEGER PRIMARY KEY,
    tax_id VARCHAR(50) NOT NULL UNIQUE,
    contact_info TEXT,
    business_name VARCHAR(255) NOT NULL,
    registration_date DATE NOT NULL,
    sellingPlan selling_plan NOT NULL
);




-- 4) Create junction and dependent tables
CREATE TABLE ADDRESS (
    address_id INTEGER PRIMARY KEY,
        address_title VARCHAR(32) NOT NULL,
    address_line VARCHAR(255) NOT NULL,
    city VARCHAR(100) NOT NULL,
    country_code CHAR(2) NOT NULL,
    postal_code VARCHAR(20) NOT NULL,
        addressType address_type  NOT NULL,
        customer_id INTEGER,
    street VARCHAR(255),
    building_no VARCHAR(50),
    neighborhood VARCHAR(255),
    title VARCHAR(100),
    apartment VARCHAR(50),
        shop_id INTEGER,
    warehouse_id INTEGER,
    return_address BOOLEAN DEFAULT false,
        available_hours TEXT,
        CONSTRAINT customer_address_customer_id_fkey
```

```
        FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id)
                ON DELETE CASCADE ON UPDATE CASCADE,
        CONSTRAINT shipment_address_shop_id_fkey
        FOREIGN KEY (shop_id) REFERENCES SHOP(shop_id)
        ON DELETE SET NULL ON UPDATE CASCADE,
        CONSTRAINT shipment_address_warehouse_id_fkey
        FOREIGN KEY (warehouse_id) REFERENCES WAREHOUSE(warehouse_id)
        ON DELETE SET NULL ON UPDATE CASCADE
);



CREATE TABLE LIST (
  customer_id INTEGER,
  list_name VARCHAR(255) NOT NULL,
  collaborator1 INTEGER,
  collaborator2 INTEGER,
  PRIMARY KEY (customer_id, list_name),
        UNIQUE(list_name),
        CONSTRAINT list_customer_id_fkey
        FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id)
                ON DELETE CASCADE ON UPDATE CASCADE,
        CONSTRAINT list_collaborator1_fkey
        FOREIGN KEY (collaborator1) REFERENCES CUSTOMER(customer_id)
                ON DELETE SET NULL ON UPDATE CASCADE,
        CONSTRAINT list_collaborator2_fkey
        FOREIGN KEY (collaborator2) REFERENCES CUSTOMER(customer_id)
                ON DELETE SET NULL ON UPDATE CASCADE

);

CREATE TABLE PAYMENT (
```

```
  customer_id INTEGER,
  card_info TEXT UNIQUE NOT NULL,
  paymentMethod payment_method NOT NULL,
  PRIMARY KEY (customer_id, card_info),
      CONSTRAINT payment_customer_id_fkey
      FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id)
          ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE "OPTIONS" (
  product_id INTEGER,
  option_id INTEGER NOT NULL,
  option_name VARCHAR(255) NOT NULL,
  PRIMARY KEY (product_id, option_id),
      CONSTRAINT options_product_id_fkey
      FOREIGN KEY (product_id) REFERENCES PRODUCT(product_id)
          ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE OPTION_VALUES (
  product_id INTEGER,
  option_id INTEGER,
  value_id INTEGER NOT NULL,
  value_name VARCHAR(255) NOT NULL,
  PRIMARY KEY (product_id, option_id, value_id),
      CONSTRAINT option_values_product_id_option_id_fkey
      FOREIGN KEY (product_id, option_id) REFERENCES "OPTIONS"(product_id, option_id)
          ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE SHOP_PRODUCT (
```

```
  product_id INTEGER,
  shop_id INTEGER,
  sku VARCHAR(50) NOT NULL,
  price DECIMAL(10,2) NOT NULL,
  quantity INTEGER NOT NULL DEFAULT 0,
  warehouse_id INTEGER,
  max_purchasable_qty INTEGER,
  min_purchasable_qty INTEGER,
  PRIMARY KEY (product_id, shop_id, sku),
      UNIQUE(product_id, shop_id, sku),
      UNIQUE(shop_id, sku),
      CONSTRAINT shop_product_product_id_fkey
      FOREIGN KEY (product_id) REFERENCES PRODUCT(product_id)
          ON DELETE CASCADE ON UPDATE CASCADE,
      CONSTRAINT shop_product_shop_id_fkey
      FOREIGN KEY (shop_id) REFERENCES SHOP(shop_id)
          ON DELETE CASCADE ON UPDATE CASCADE,
      CONSTRAINT shop_product_warehouse_id_fkey
      FOREIGN KEY (warehouse_id) REFERENCES WAREHOUSE(warehouse_id)
          ON DELETE SET NULL ON UPDATE CASCADE

);

CREATE TABLE CART (
  customer_id INTEGER,
  "session" VARCHAR(255),
  PRIMARY KEY (customer_id, "session"),
      CONSTRAINT cart_customer_id_fkey
      FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id)
          ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
CREATE TABLE CART_ITEM (
  customer_id INTEGER,
      "session" VARCHAR(255),
  shop_id INTEGER,
  sku VARCHAR(50) NOT NULL,
  quantity INTEGER NOT NULL,
  PRIMARY KEY (customer_id, shop_id, sku),
      CONSTRAINT cart_item_shop_id_sku_fkey
      FOREIGN KEY (shop_id, sku) REFERENCES SHOP_PRODUCT(shop_id, sku)
          ON DELETE CASCADE ON UPDATE CASCADE,
      CONSTRAINT cart_item_cart_fkey
          FOREIGN KEY ("session", customer_id) REFERENCES CART("session", customer_id)
              ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE "ORDER" (
  customer_id INTEGER,
  order_number INTEGER NOT NULL,
  card_info TEXT,
  total_price DECIMAL(10,2) NOT NULL,
  order_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  discount DECIMAL(10,2),
  has_gift BOOLEAN DEFAULT false,
  customer_address INTEGER,
  delivery_spot INTEGER,
  PRIMARY KEY (customer_id, order_number),
      CONSTRAINT order_customer_id_fkey
      FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id)
          ON DELETE RESTRICT ON UPDATE CASCADE,
      CONSTRAINT order_card_info_fkey
```

```
        FOREIGN KEY (card_info) REFERENCES PAYMENT(card_info)
        ON DELETE RESTRICT ON UPDATE CASCADE,
        CONSTRAINT order_delivery_spot_fkey
        FOREIGN KEY (delivery_spot) REFERENCES ADDRESS(address_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
        CONSTRAINT order_customer_address_fkey
        FOREIGN KEY (customer_address) REFERENCES ADDRESS(address_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE ORDER_ITEM (
  customer_id INTEGER,
  order_number INTEGER,
  product_id INTEGER,
  sku VARCHAR(50),
  shop_id INTEGER,
        qty INTEGER NOT NULL,
  PRIMARY KEY (customer_id, order_number, product_id, sku),
        UNIQUE(customer_id, order_number, sku, shop_id),
        CONSTRAINT order_item_customer_id_order_number_fkey
        FOREIGN KEY (customer_id, order_number) REFERENCES "ORDER"(customer_id,
order_number)
        ON DELETE CASCADE ON UPDATE CASCADE,
        CONSTRAINT order_item_product_id_shop_id_sku_fkey
        FOREIGN KEY (product_id, shop_id, sku) REFERENCES SHOP_PRODUCT(product_id, shop_id,
sku)
        ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE SHIPMENT (
  shipment_id INTEGER,
```

```
  customer_id INTEGER,
  order_number INTEGER,
      product_id INTEGER,
  deliveryType delivery_type,
  carrier VARCHAR(100),
  cargo_company VARCHAR(100),
  status shipment_status,
  estimated_delivery TIMESTAMP,
  shipment_address INTEGER,
  sku VARCHAR(50),
  shop_id INTEGER,
      PRIMARY KEY(shipment_id, customer_id, order_number, product_id, sku),
  CONSTRAINT shipment_customer_id_order_number_sku_shop_id_fkey
      FOREIGN KEY (customer_id, order_number, sku, shop_id) REFERENCES
ORDER_ITEM(customer_id, order_number, sku, shop_id)
      ON DELETE RESTRICT ON UPDATE CASCADE,
      CONSTRAINT shipment_shipment_address_fkey
      FOREIGN KEY (shipment_address) REFERENCES ADDRESS(address_id)
      ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE EXPORT_PRODUCT (
  product_id INTEGER,
  price DECIMAL(10,2),
  tax_discount DECIMAL(10,2),
  country CHAR(2),
  PRIMARY KEY (product_id),
      CONSTRAINT export_product_product_id_fkey
      FOREIGN KEY (product_id) REFERENCES PRODUCT(product_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
CREATE TABLE FOLLOWS (
  customer_id INTEGER,
  shop_id INTEGER,
  PRIMARY KEY (customer_id, shop_id),
      CONSTRAINT follows_customer_id_fkey
      FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
      CONSTRAINT follows_shop_id_fkey
      FOREIGN KEY (shop_id) REFERENCES SHOP(shop_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE REVIEWS (
  customer_id INTEGER,
  product_id INTEGER,
      shop_id INTEGER,
      sku VARCHAR(50),
  submission_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  rating INTEGER,
  text TEXT,
  PRIMARY KEY (customer_id, product_id, shop_id, sku),
      CONSTRAINT reviews_customer_id_fkey
      FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
      CONSTRAINT reviews_product_id_fkey
      FOREIGN KEY (product_id, shop_id, sku) REFERENCES SHOP_PRODUCT(product_id, shop_id,
sku)
      ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
CREATE TABLE WISHLIST_PRODUCT (
  customer_id INTEGER,
  product_id INTEGER,
  wishlist_name VARCHAR(255),
  PRIMARY KEY (customer_id, product_id, wishlist_name),
      CONSTRAINT wishlist_product_customer_id_fkey
      FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
      CONSTRAINT wishlist_product_product_id_fkey
      FOREIGN KEY (product_id) REFERENCES PRODUCT(product_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
      CONSTRAINT wishlist_name_list_name_fkey
  FOREIGN KEY (wishlist_name) REFERENCES LIST(list_name)
    ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE SKU_VALUES (
  product_id INTEGER,
  shop_id INTEGER,
  option_id INTEGER,
  sku VARCHAR(50),
  "value" INTEGER,
  PRIMARY KEY (product_id, shop_id, option_id, sku),
      CONSTRAINT sku_values_product_id_shop_id_sku_fkey
      FOREIGN KEY (product_id, shop_id, sku) REFERENCES SHOP_PRODUCT(product_id, shop_id,
sku)
      ON DELETE CASCADE ON UPDATE CASCADE,
      CONSTRAINT sku_values_product_id_option_id_fkey
      FOREIGN KEY (product_id, option_id) REFERENCES "OPTIONS"(product_id, option_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
      CONSTRAINT sku_values_product_id_option_id_value_fkey
```

```
    FOREIGN KEY (product_id, option_id, "value") REFERENCES OPTION_VALUES(product_id, option_id, value_id)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

3.b Population of Database

```
-- 7) Populate db

INSERT INTO CUSTOMER (customer_id, email, "password", phone, full_name) VALUES
(1, 'john.doe@example.com', crypt('MySecurePassword1!', gen_salt('bf')), '+1234567890', 'John Doe'),
(2, 'jane.smith@example.com', crypt('AnotherSecurePassword2@', gen_salt('bf')), '+0987654321', 'Jane Smith'),
(3, 'alice.wonder@example.com', crypt('PasswordForAlice#3', gen_salt('bf')), '+1122334455', 'Alice Wonder'),
(4, 'bob.builder@example.com', crypt('CanWeFixIt4$', gen_salt('bf')), '+2233445566', 'Bob Builder'),
(5, 'emma.watson@example.com', crypt('Wizardry5&', gen_salt('bf')), '+3344556677', 'Emma Watson'),
(6, 'bob.wilson@email.com', 'hash3', '+1-555-0125', 'Bob Wilson'),
(7, 'alice.brown@email.com', 'hash4', '+1-555-0126', 'Alice Brown'),
(8, 'charlie.davis@email.com', 'hash5', '+1-555-0127', 'Charlie Davis');


INSERT INTO BRAND (brand_id, brand_name) VALUES
(1, 'Nike'),
(2, 'Adidas'),
(3, 'Sony'),
(4, 'Apple'),
(5, 'Samsung');


INSERT INTO CATEGORY (category_id, category_name, leaf, paths, parent_category_id) VALUES
(1, 'Electronics', false, '/', NULL),
(2, 'Clothing', false, '/', NULL),
(3, 'Tablets&Computers', false, 'Electronics/Tablets&Computers', 1),
```

(4, 'Laptops', true, 'Electronics/Tablets&Computers/Laptops', 3),

(5, 'Smartphones', true, 'Electronics/Smartphones', 1),

(6, 'Shoes', true, 'Clothing/Shoes', 2),

(7, 'Tablets', true, 'Electronics/Tablets&Computers/Tablets', 3);

-- Insert electronic products (Smartphone, Laptop, Tablet)

INSERT INTO PRODUCT (product_id, release_date, manufacturer_name, prod_name, imageURL, videoURL, barcode, description, brand_id, category_id) VALUES

(1, '2024-01-15', 'Samsung', 'Galaxy S21', 'https://example.com/s21.jpg', 'https://example.com/s21.mp4', 123456789, 'A high-end smartphone with a 6.2-inch screen and 5G support.', 1, 5),

(2, '2024-02-10', 'Apple', 'MacBook Pro 16"', 'https://example.com/macbookpro16.jpg', 'https://example.com/macbookpro16.mp4', 234567890, 'Powerful laptop with 16-inch Retina display, M1 Pro chip.', 2, 4),

(3, '2024-03-05', 'Sony', 'Xperia Tablet Z', 'https://example.com/xperia_tablet.jpg', 'https://example.com/xperia_tablet.mp4', 345678901, 'A tablet with a 10-inch display and waterproof features.', 3, 7),

(4, '2024-04-01', 'Google', 'Pixel 6', 'https://example.com/pixel6.jpg', 'https://example.com/pixel6.mp4', 456789012, 'Flagship smartphone with Google’s Tensor chip and 5G support.', 4, 5),

(5, '2024-05-15', 'Microsoft', 'Surface Laptop 4', 'https://example.com/surface_laptop4.jpg', 'https://example.com/surface_laptop4.mp4', 567890123, 'Laptop with 13.5-inch PixelSense display and Intel Core i7.', 5, 4);

INSERT INTO PRODUCT (product_id, release_date, manufacturer_name, prod_name, imageURL, videoURL, barcode, description, brand_id, category_id) VALUES

(6, '2024-06-10', 'Nike', 'Running Shoes', 'https://example.com/running_shoes.jpg', NULL, 678901234, 'Comfortable running shoes.', 1, 6),

(7, '2024-07-20', 'Adidas', 'Sneakers', 'https://example.com/sneakers.jpg', NULL, 789012345, 'Stylish Adidas sneakers.', 2, 6);

INSERT INTO WAREHOUSE (warehouse_id, "info") VALUES

(1, 'Main Warehouse - New York'),

(2, 'LA Warehouse - Los Angeles'),

(3, 'Chicago Warehouse - Chicago'),

(4, 'Miami Warehouse - Miami'),

(5, 'Dallas Warehouse - Dallas');

INSERT INTO SHOP (shop_id, tax_id, contact_info, business_name, registration_date, sellingPlan) VALUES

(1, 'TAX12345', 'support@shop1.com', 'Tech Store', '2023-01-01', 'Premium'),

(2, 'TAX67890', 'info@shop2.com', 'Fashion Outlet', '2023-02-15', 'Standard'),

(3, 'TAX11223', 'help@shop3.com', 'Gadget World', '2023-03-10', 'Enterprise'),

(4, 'TAX44556', 'sales@shop4.com', 'Game Center', '2023-04-20', 'Basic'),

(5, 'TAX77889', 'support@shop5.com', 'Shoe Heaven', '2023-05-30', 'Premium');

-- Customer Addresses

INSERT INTO ADDRESS (address_id, address_title, address_line, city, country_code, postal_code, addressType, customer_id, street, building_no, neighborhood, title, apartment) VALUES

(1, 'Home', '123 Elm St', 'New York', 'US', '10001', 'Customer', 1, 'Elm Street', '101', 'Downtown', 'Home', 'Apt 1'),

(2, 'MyHome', '456 Oak St', 'Los Angeles', 'US', '90001', 'Customer', 2, 'Oak Street', '202', 'Central Park', 'Home', 'Apt 2'),

(3, 'Work', '789 Pine St', 'Chicago', 'US', '60001', 'Customer', 3, 'Pine Street', '303', 'Midtown', 'Work', 'Apt 3'),

(4, 'Office', '101 Maple St', 'Miami', 'US', '33101', 'Customer', 4, 'Maple Street', '404', 'South Beach', 'Home', 'Apt 4'),

(5, 'FriendsHome', '202 Birch St', 'Dallas', 'US', '75201', 'Customer', 5, 'Birch Street', '505', 'Uptown', 'Home', 'Apt 5');

-- Shop Addresses

INSERT INTO ADDRESS (address_id, address_title, address_line, city, country_code, postal_code, addressType, shop_id) VALUES

(6, 'TechStore', '123 Market St', 'New York', 'US', '10001', 'Shipment', 1),

(7, 'FashionOutlet', '456 Commerce Blvd', 'Los Angeles', 'US', '90001', 'Shipment', 2),

(8, 'GadgetWorld', '789 Retail Rd', 'Chicago', 'US', '60001', 'Shipment', 3),

(9, 'GameCenter', '101 Trade Ave', 'Miami', 'US', '33101', 'Shipment', 4),

(10, 'ShoeHeaven', '202 Exchange St', 'Dallas', 'US', '75201', 'Shipment', 5);

-- Warehouse Addresses

INSERT INTO ADDRESS (address_id, address_title, address_line, city, country_code, postal_code, addressType, warehouse_id) VALUES

(11, 'WH-NY', '123 Supply Rd', 'New York', 'US', '10001', 'Shipment', 1),

(12, 'WH-LA', '456 Storage Ln', 'Los Angeles', 'US', '90001', 'Shipment', 2),

(13, 'WH-CH', '789 Distribution Blvd', 'Chicago', 'US', '60001', 'Shipment', 3),

(14, 'WH-MI', '101 Logistics St', 'Miami', 'US', '33101', 'Shipment', 4),

(15, 'WH-DA', '202 Fulfillment Ave', 'Dallas', 'US', '75201', 'Shipment', 5);

-- List table (for customers and collaborators)

INSERT INTO LIST (customer_id, list_name, collaborator1, collaborator2) VALUES

(1, 'Summer Wishlist', NULL, NULL),

(2, 'Holiday Shopping', 1, 4),

(3, 'Birthday Gifts', 2, 5),

(4, 'New Year Sale', 1, 5),

(5, 'Housewarming Gifts', 3, 4);

-- Payment table (customer_id, card_info, and payment method)

INSERT INTO PAYMENT (customer_id, card_info, paymentMethod) VALUES

(1, '1234567812345678', 'Credit Card'),

(2, '2345678923456789', 'Debit Card'),

(3, '3456789034567890', 'PayPal'),

(4, '4567890145678901', 'Bank Transfer'),

(5, '5678901256789012', 'Cash on Delivery');

```
-- Options for electronics (screen size, storage, color, etc.)
INSERT INTO "OPTIONS" (product_id, option_id, option_name) VALUES
(1, 1, 'Color'),
(1, 2, 'Storage Capacity'),
(1, 3, 'Screen Size'),
(2, 1, 'Color'),
(2, 2, 'Storage Capacity'),
(2, 3, 'Processor'),
(3, 1, 'Color'),
(3, 2, 'Storage Capacity'),
(3, 3, 'Screen Size'),
(4, 1, 'Color'),
(4, 2, 'Storage Capacity'),
(4, 3, 'Processor'),
(5, 1, 'Color'),
(5, 2, 'Storage Capacity'),
(5, 3, 'Processor');
INSERT INTO "OPTIONS" (product_id, option_id, option_name) VALUES
(6, 1, 'Color'),
(6, 2, 'Storage Capacity'),
(6, 3, 'Processor');



-- Option values for electronics (e.g., colors, storage, screen size)
INSERT INTO OPTION_VALUES (product_id, option_id, value_id, value_name) VALUES
(1, 1, 1, 'Black'),
(1, 1, 2, 'White'),
(1, 1, 3, 'Blue'),
(1, 2, 1, '128GB'),
(1, 2, 2, '256GB'),
```

```
(1, 2, 3, '512GB'),
(1, 3, 1, '6.2 inches'),
(2, 1, 1, 'Silver'),
(2, 1, 2, 'Space Gray'),
(2, 2, 1, '512GB'),
(2, 2, 2, '1TB'),
(2, 3, 1, 'M1 Pro'),
(3, 1, 1, 'Black'),
(3, 1, 2, 'White'),
(3, 2, 1, '64GB'),
(3, 2, 2, '128GB'),
(3, 3, 1, '10 inches'),
(4, 1, 1, 'Black'),
(4, 1, 2, 'White'),
(4, 2, 1, '128GB'),
(4, 2, 2, '256GB'),
(4, 3, 1, 'Google Tensor'),
(5, 1, 1, 'Platinum'),
(5, 1, 2, 'Ice Blue'),
(5, 2, 1, '256GB'),
(5, 2, 2, '512GB'),
(5, 3, 1, 'Intel Core i7');
INSERT INTO OPTION_VALUES (product_id, option_id, value_id, value_name) VALUES
(6, 1, 1, 'Platinum'),
(6, 2, 1, '256 GB'),
(6, 3, 1, 'Intel Core i7');



INSERT INTO CART (customer_id, "session") VALUES
(1, 'session_1'),
(2, 'session_2'),
```

(3, 'session_3'),

(4, 'session_4'),

(5, 'session_5');

-- Insert data into SHOP_PRODUCT table

INSERT INTO SHOP_PRODUCT (product_id, shop_id, sku, price, quantity, warehouse_id, max_purchasable_qty, min_purchasable_qty) VALUES

(1, 1, 'SKU-1-1', 799.99, 50, 1, 5, 1),

(2, 1, 'SKU-2-1', 2399.99, 20, 1, 3, 1),

(3, 1, 'SKU-3-1', 499.99, 15, 1, 2, 1),

(6, 2, 'SKU-6-2', 89.99, 100, 2, 10, 1),

(5, 3, 'SKU-5-3', 1299.99, 40, 3, 5, 1),

(2, 3, 'SKU-2-3', 2299.99, 18, 3, 3, 1),

(4, 4, 'SKU-4-4', 549.99, 50, 4, 5, 1),

(1, 4, 'SKU-1-4', 799.99, 35, 4, 5, 1),

(3, 4, 'SKU-3-4', 479.99, 20, 4, 2, 1),

(6, 5, 'SKU-6-5', 99.99, 200, 5, 10, 1);

INSERT INTO CART_ITEM (customer_id, "session", shop_id, sku, quantity) VALUES

(1, 'session_1', 1, 'SKU-1-1', 2),

(1, 'session_1', 4, 'SKU-1-4', 1),

(2, 'session_2', 1, 'SKU-2-1', 1),

(2, 'session_2', 3, 'SKU-2-3', 1),

(3, 'session_3', 1, 'SKU-3-1', 1),

(3, 'session_3', 4, 'SKU-3-4', 1),

(4, 'session_4', 4, 'SKU-4-4', 2),

(5, 'session_5', 5, 'SKU-6-5', 3);

-- Insert data into SKU_VALUES table based on valid combinations of options

```
-- Insert values for SKU-1-1
INSERT INTO SKU_VALUES (product_id, shop_id, sku, option_id, "value") VALUES
(1, 1, 'SKU-1-1', 1, 1), -- color: Black
(1, 1, 'SKU-1-1', 2, 1), -- storage size: 128GB
(1, 1, 'SKU-1-1', 3, 1), -- screen size: 6.2 inches

(2, 1, 'SKU-2-1', 1, 1), -- color: Silver
(2, 1, 'SKU-2-1', 2, 1), -- storage size: 512GB
(2, 1, 'SKU-2-1', 3, 1), -- processor: M1 Pro

(3, 1, 'SKU-3-1', 1, 1), -- color: Black
(3, 1, 'SKU-3-1', 2, 1), -- storage size: 64GB
(3, 1, 'SKU-3-1', 3, 1), -- screen size: 10 inches

(6, 2, 'SKU-6-2', 1, 1), -- color: Platinum
(6, 2, 'SKU-6-2', 2, 1), -- storage size: 256 GB
(6, 2, 'SKU-6-2', 3, 1), -- processor: Intel Core i7

(5, 3, 'SKU-5-3', 1, 1), -- color: Platinum
(5, 3, 'SKU-5-3', 2, 1), -- storage size: 256GB
(5, 3, 'SKU-5-3', 3, 1), -- processor: Intel Core i7

(2, 3, 'SKU-2-3', 1, 1), -- color: Silver
(2, 3, 'SKU-2-3', 2, 1), -- storage size: 512GB
(2, 3, 'SKU-2-3', 3, 1), -- processor: M1 Pro

(4, 4, 'SKU-4-4', 1, 1), -- color: Black
(4, 4, 'SKU-4-4', 2, 1), -- storage size: 128GB
(4, 4, 'SKU-4-4', 3, 1), -- processor: Google Tensor

(3, 4, 'SKU-3-4', 1, 1), -- color: Black
```

(3, 4, 'SKU-3-4', 2, 1), -- storage size: 128GB

(3, 4, 'SKU-3-4', 3, 1), -- processor: Google Tensor

(6, 5, 'SKU-6-5', 1, 1), -- color: Platinum

(6, 5, 'SKU-6-5', 2, 1), -- storage size: 256 GB

(6, 5, 'SKU-6-5', 3, 1), -- processor: Intel Core i7

(1, 4, 'SKU-1-4', 1, 1), -- color: Black

(1, 4, 'SKU-1-4', 2, 1), -- storage size: 128GB

(1, 4, 'SKU-1-4', 3, 1); -- screen size: 6.2 inches

INSERT INTO FOLLOWS (customer_id, shop_id) VALUES

(1, 1),

(1, 2),

(2, 1),

(2, 3),

(3, 4),

(3, 5),

(4, 2),

(4, 3),

(5, 1),

(5, 4),

(6,1);

INSERT INTO WISHLIST_PRODUCT (customer_id, product_id, wishlist_name) VALUES

(1, 1, 'Summer Wishlist'),

(2, 2, 'Holiday Shopping'),

(3, 3, 'Birthday Gifts'),

(4, 4, 'New Year Sale'),

(5, 5, 'Housewarming Gifts'),

(1, 2, 'Summer Wishlist'),

```
(3, 4, 'Birthday Gifts'),
(2, 5, 'Holiday Shopping');


-- Insert data into EXPORT_PRODUCT table
INSERT INTO EXPORT_PRODUCT (product_id, price, tax_discount, country) VALUES
(1, 799.99, 50.00, 'US'),  -- Product 1 exported to the US with a tax discount of 50.00
(2, 2399.99, 100.00, 'IN'), -- Product 2 exported to India with a tax discount of 100.00
(3, 499.99, 30.00, 'UK'),  -- Product 3 exported to the UK with a tax discount of 30.00
(4, 549.99, 25.00, 'CA'),  -- Product 4 exported to Canada with a tax discount of 25.00
(5, 1299.99, 150.00, 'AU'), -- Product 5 exported to Australia with a tax discount of 150.00
(6, 99.99, 10.00, 'DE'),   -- Product 6 exported to Germany with a tax discount of 10.00
(7, 1499.99, 75.00, 'FR');  -- Product 7 exported to France with a tax discount of 75.00


-- Insert sample orders while adhering to referential integrity constraints
INSERT INTO "ORDER" (customer_id, order_number, card_info, total_price, order_date, discount,
has_gift, customer_address, delivery_spot) VALUES
(1, 1001, '1234567812345678', 1299.98, '2025-01-01 10:00:00', 50.00, false, 1, NULL),  -- Order 1 for
customer sku11 sku31
(2, 1002, '2345678923456789', 2399.99, '2025-01-02 14:30:00', 100.00, false, 2, NULL),  -- Order 2 for
customer 2 sku21
(3, 1003, '3456789034567890', 1299.99, '2025-01-03 09:15:00', 30.00, false, 3, NULL),  -- Order 3 for
customer 3 sku53
(4, 1004, '4567890145678901', 639.98, '2025-01-04 16:45:00', 20.00, false, 4, NULL),   -- Order 4 for
customer 4 sku62 sku44
(5, 1005, '5678901256789012', 899.98, '2025-01-05 11:20:00', 75.00, false, 5, NULL);  -- Order 5 for
customer 5 sku14 sku65


-- Order 1 for customer 1, SKU-11 and SKU-31
```

INSERT INTO ORDER_ITEM (customer_id, order_number, product_id, sku, shop_id, qty) VALUES

(1, 1001, 1, 'SKU-1-1', 1, 1), -- SKU-1-1 for Order 1 (Customer 1)

(1, 1001, 3, 'SKU-3-1', 1, 1), -- SKU-3-1 for Order 1 (Customer 1)

(2, 1002, 2, 'SKU-2-1', 1, 1), -- SKU-2-1 for Order 2 (Customer 2)

(3, 1003, 5, 'SKU-5-3', 3, 1), -- SKU-5-3 for Order 3 (Customer 3)

(4, 1004, 6, 'SKU-6-2', 2, 1), -- SKU-6-2 for Order 4 (Customer 4)

(4, 1004, 4, 'SKU-4-4', 4, 1), -- SKU-4-4 for Order 4 (Customer 4)

(5, 1005, 1, 'SKU-1-4', 4, 1), -- SKU-1-4 for Order 5 (Customer 5)

(5, 1005, 6, 'SKU-6-5', 5, 1); -- SKU-6-5 for Order 5 (Customer 5)

INSERT INTO SHIPMENT (shipment_id, customer_id, order_number, product_id, deliveryType, carrier, cargo_company, status, estimated_delivery, shipment_address, sku, shop_id) VALUES

(1, 1, 1001, 11, 'Standard', 'UPS', 'CarrierCompanyA', 'Shipped', '2025-02-03 10:30:00', 1, 'SKU-1-1', 1), -- Shipment for Order 1 (Customer 1)

(2, 1, 1001, 31, 'Express', 'FedEx', 'CarrierCompanyB', 'Shipped', '2025-02-04 12:00:00', 1, 'SKU-3-1', 1), -- Shipment for Order 1 (Customer 1)

(3, 2, 1002, 21, 'Standard', 'DHL', 'CarrierCompanyC', 'Shipped', '2025-02-06 11:00:00', 2, 'SKU-2-1', 1), -- Shipment for Order 2 (Customer 2)

(4, 3, 1003, 53, 'Standard', 'UPS', 'CarrierCompanyA', 'Shipped', '2025-02-07 09:30:00', 3, 'SKU-5-3', 3), -- Shipment for Order 3 (Customer 3)

(5, 4, 1004, 62, 'Express', 'FedEx', 'CarrierCompanyB', 'Shipped', '2025-02-08 15:30:00', 4, 'SKU-6-2', 2), -- Shipment for Order 4 (Customer 4)

(6, 4, 1004, 44, 'Standard', 'DHL', 'CarrierCompanyC', 'Shipped', '2025-02-09 16:00:00', 4, 'SKU-4-4', 4), -- Shipment for Order 4 (Customer 4)

(7, 5, 1005, 14, 'Express', 'UPS', 'CarrierCompanyA', 'Shipped', '2025-02-10 13:30:00', 5, 'SKU-1-4', 4), -- Shipment for Order 5 (Customer 5)

(8, 5, 1005, 65, 'Standard', 'FedEx', 'CarrierCompanyB', 'Shipped', '2025-02-11 10:00:00', 5, 'SKU-6-5', 5); -- Shipment for Order 5 (Customer 5)

INSERT INTO REVIEWS (customer_id, product_id, shop_id, sku, rating, text)

VALUES

(1, 1, 1, 'SKU-1-1', 5, 'Amazing product! Quality exceeded my expectations. Fast delivery too.'),

(1, 3, 1, 'SKU-3-1', 4, 'Good quality product, but slightly different from the pictures shown.'),

(2, 2, 1, 'SKU-2-1', 5, 'Excellent purchase! Worth every penny.'),

(3, 5, 3, 'SKU-5-3', 4, 'Product works great. Minor issue with packaging but item was perfect.'),

(4, 6, 2, 'SKU-6-2', 5, 'Exactly what I needed. Would definitely buy again!'),

(4, 4, 4, 'SKU-4-4', 3, 'Product is okay but could use some improvements in build quality.'),

(5, 1, 4, 'SKU-1-4', 5, 'Perfect fit and great quality. Highly recommend this seller.'),

(5, 6, 5, 'SKU-6-5', 4, 'Good product overall. Shipping was quick and item was well-packaged.');

3.c Triggers

```
-- 6) Triggers
CREATE OR REPLACE FUNCTION validate_purchasable_qty()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.min_purchasable_qty > NEW.max_purchasable_qty THEN
    RAISE EXCEPTION 'Minimum purchasable quantity (%) cannot be greater than maximum purchasable quantity (%)',
      NEW.min_purchasable_qty, NEW.max_purchasable_qty;
  END IF;

  IF NEW.quantity > 0 AND NEW.max_purchasable_qty > NEW.quantity THEN
    NEW.max_purchasable_qty := NEW.quantity;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_purchasable_qty
BEFORE INSERT OR UPDATE ON SHOP_PRODUCT
FOR EACH ROW
EXECUTE FUNCTION validate_purchasable_qty();
```

```
-- 2. Trigger to update product quantity after order placement
CREATE OR REPLACE FUNCTION update_product_quantity()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE SHOP_PRODUCT
  SET quantity = quantity - NEW.qty
  WHERE product_id = NEW.product_id
  AND shop_id = NEW.shop_id
  AND sku = NEW.sku;

  -- Check if quantity went below 0
  IF (SELECT quantity FROM SHOP_PRODUCT
    WHERE product_id = NEW.product_id
    AND shop_id = NEW.shop_id
    AND sku = NEW.sku) < 0 THEN
    RAISE EXCEPTION 'Insufficient stock for product with SKU %', NEW.sku;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER order_quantity_update
AFTER INSERT ON ORDER_ITEM
FOR EACH ROW
EXECUTE FUNCTION update_product_quantity();

-- 3. Trigger to validate rating range in REVIEWS
CREATE OR REPLACE FUNCTION validate_rating()
RETURNS TRIGGER AS $$
BEGIN
```

```
  -- Check if rating is between 1 and 5
  IF NEW.rating < 1 OR NEW.rating > 5 THEN
    RAISE EXCEPTION 'Rating must be between 1 and 5';
  END IF;

  -- Check if customer has actually ordered this product
  IF NOT EXISTS (
    SELECT 1 FROM ORDER_ITEM oi
    WHERE oi.customer_id = NEW.customer_id
    AND oi.product_id = NEW.product_id
    AND oi.shop_id = NEW.shop_id
    AND oi.sku = NEW.sku
  ) THEN
    RAISE EXCEPTION 'Customer must purchase the product before leaving a review';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_review_rating
BEFORE INSERT OR UPDATE ON REVIEWS
FOR EACH ROW
EXECUTE FUNCTION validate_rating();
```

3.d Constraints

```
-- 5) Constraintler

ALTER TABLE SHOP_PRODUCT

ADD CONSTRAINT check_valid_price

CHECK (price > 0);

-- 2. Check constraint to ensure order quantities are within valid range

ALTER TABLE ORDER_ITEM

ADD CONSTRAINT check_order_quantity

CHECK (qty > 0 );

-- 3. Check constraint to ensure discount cannot exceed total price

ALTER TABLE "ORDER"

ADD CONSTRAINT check_valid_discount

CHECK (

(discount IS NULL) OR

(discount >= 0 AND discount <= total_price)

);
```

3.e SQL Queries

3.e.1 INSERT, DELETE, UPDATE

```
-- payment

INSERT INTO PAYMENT (customer_id, card_info, paymentmethod)

VALUES

(2, '4532123456781234', 'Credit Card'),

(3, '5412345678123456', 'Credit Card'),

(4, '3712345678912345', 'Bank Transfer');


UPDATE PAYMENT

SET paymentmethod = 'Debit Card',

WHERE card_info = '1234567812345678';
```

```
delete from payment

where paymentmethod = 'PayPal';


-- Address

INSERT INTO ADDRESS (address_id, address_title, address_line, city, country_code, postal_code, addressType, customer_id, street, building_no, neighborhood, title, apartment) VALUES

(20, 'Home2', 'Catalkaya', 'New York', 'US', '10001', 'Customer', 3, 'Elm Street', '101', 'Downtown', 'Home', 'Apt 1'),

(19, 'MyHome3', 'Huzur', 'Los Angeles', 'US', '90001', 'Customer', 4, 'Oak Street', '202', 'Central Park', 'Home', 'Apt 2');


UPDATE ADDRESS

SET address_title ='GFs HOUSE'

WHERE address_id = 1;


DELETE FROM ADDRESS

WHERE address_id = 12;



-- 1. REVIEWS table operations

-- INSERT into REVIEWS (requires existing order/purchase)

INSERT INTO REVIEWS (customer_id, product_id, shop_id, sku, rating, text)

VALUES

(2, 2, 1, 'SKU-2-1', 4, 'Good quality but shipping was slow');


-- UPDATE REVIEWS

UPDATE REVIEWS

SET rating = 3,

text = 'Updated review: Product quality declined after few uses'

WHERE customer_id = 1

AND product_id = 1

AND shop_id = 1
```

```
AND sku = 'SKU-1-1';

-- DELETE from REVIEWS

DELETE FROM REVIEWS

WHERE customer_id = 2

AND product_id = 2

AND shop_id = 1

AND sku = 'SKU-2-1';
```

3.e.1 SELECT

```
-- Queries using minimum 2 tables

-- 1. Get all products and their current stock levels by shop

SELECT

  p.prod_name,

  sp.shop_id,

  s.business_name,

  sp.sku,

  sp.price,

  sp.quantity as current_stock

FROM PRODUCT p

JOIN SHOP_PRODUCT sp ON p.product_id = sp.product_id

JOIN SHOP s ON sp.shop_id = s.shop_id

WHERE sp.quantity > 0

ORDER BY sp.quantity ASC;


-- 2. Find customers who follow shops but haven't made any purchases

SELECT

  c.customer_id,

  c.full_name,

  s.business_name as shop_name

FROM CUSTOMER c

JOIN FOLLOWS f ON c.customer_id = f.customer_id
```

```
JOIN SHOP s ON f.shop_id = s.shop_id
WHERE NOT EXISTS (
  SELECT * FROM "ORDER" o
  WHERE o.customer_id = c.customer_id
);

-- Queries using minimum 3 tables
-- 3. Get detailed order history with product and delivery information
SELECT
  oi.order_number,
  c.full_name as customer_name,
  p.prod_name,
  oi.product_id as order_item_pid,
  s.product_id as shipment_pid,
  oi.sku as order_item_sku,
  s.sku as shipment_sku,
  s.status as shipment_status
FROM ORDER_ITEM oi
JOIN SHIPMENT s ON oi.customer_id = s.customer_id
  AND oi.order_number = s.order_number
  AND oi.sku = s.sku
  AND oi.shop_id = s.shop_id
JOIN CUSTOMER c ON oi.customer_id = c.customer_id
JOIN PRODUCT p ON oi.product_id = p.product_id;
-- 4. Get category-wise sales analysis
SELECT
  c.category_name,
  COUNT(DISTINCT o.order_number) as total_orders,
  SUM(oi.qty) as total_items_sold,
  SUM(oi.qty * sp.price) as total_revenue
FROM CATEGORY c
```

```
JOIN PRODUCT p ON c.category_id = p.category_id
JOIN SHOP_PRODUCT sp ON p.product_id = sp.product_id
JOIN ORDER_ITEM oi ON sp.product_id = oi.product_id
  AND sp.shop_id = oi.shop_id
  AND sp.sku = oi.sku
JOIN "ORDER" o ON oi.customer_id = o.customer_id
  AND oi.order_number = o.order_number
WHERE o.order_date >= CURRENT_DATE - INTERVAL '90 days'
GROUP BY c.category_name
ORDER BY total_revenue DESC;

-- 5. Get product review analysis with customer and shop details
SELECT
  p.prod_name,
  s.business_name as shop_name,
  sp.sku,
  COUNT(r.rating) as total_reviews,
  ROUND(AVG(r.rating), 2) as avg_rating,
  COUNT(DISTINCT o.order_number) as total_orders,
  SUM(oi.qty) as total_units_sold
FROM PRODUCT p
JOIN SHOP_PRODUCT sp ON p.product_id = sp.product_id
JOIN SHOP s ON sp.shop_id = s.shop_id
LEFT JOIN REVIEWS r ON p.product_id = r.product_id
  AND sp.shop_id = r.shop_id
  AND sp.sku = r.sku
LEFT JOIN ORDER_ITEM oi ON sp.product_id = oi.product_id
  AND sp.shop_id = oi.shop_id
  AND sp.sku = oi.sku
LEFT JOIN "ORDER" o ON oi.customer_id = o.customer_id
  AND oi.order_number = o.order_number
```

```
GROUP BY p.prod_name, s.business_name, sp.sku
HAVING AVG(r.rating) <= 5
ORDER BY avg_rating ASC;
```

3.e.3 CRITICAL SELECTS

```
-- 1. customer lifetime value analysis
WITH customer_metrics AS (
  SELECT
    c.customer_id,
    c.full_name,
    MIN(o.order_date) as first_order_date,
    MAX(o.order_date) as last_order_date,
    COUNT(DISTINCT o.order_number) as order_count,
    SUM(o.total_price) as total_spent,
    COUNT(DISTINCT r.product_id) as reviews_written,
    COUNT(DISTINCT wp.product_id) as wishlist_items
  FROM CUSTOMER c
  LEFT JOIN "ORDER" o ON c.customer_id = o.customer_id
  LEFT JOIN REVIEWS r ON c.customer_id = r.customer_id
  LEFT JOIN WISHLIST_PRODUCT wp ON c.customer_id = wp.customer_id
  GROUP BY c.customer_id, c.full_name
)
SELECT
  *,
  ROUND(total_spent / NULLIF(order_count, 0), 2) as avg_order_value,
  CASE
    WHEN EXTRACT(DAYS FROM (last_order_date - first_order_date)) = 0
    THEN ROUND(total_spent, 2)  -- If same day, use total spent as daily value
    ELSE ROUND(total_spent / EXTRACT(DAYS FROM (last_order_date - first_order_date)), 2)
  END as daily_customer_value,
  CASE
    WHEN total_spent > 1000 AND order_count > 10 THEN 'Power User'
```

```
      WHEN total_spent > 500 OR order_count > 5 THEN 'Regular'
      ELSE 'New'
    END as customer_segment
FROM customer_metrics
ORDER BY total_spent DESC;

-- 2. Select first 5 possible product variants
SELECT
    p.prod_name || ' ' ||
    COALESCE(o1.value_name, '') || ' ' ||
    COALESCE(o2.value_name, '') || ' ' ||
    COALESCE(o3.value_name, '') || ' ' ||
    COALESCE(o4.value_name, '') || ' ' ||
    COALESCE(o5.value_name, '') AS product_description
FROM
    PRODUCT p
LEFT JOIN
    OPTION_VALUES o1 ON p.product_id = o1.product_id AND o1.option_id = 1
LEFT JOIN
    OPTION_VALUES o2 ON p.product_id = o2.product_id AND o2.option_id = 2
LEFT JOIN
    OPTION_VALUES o3 ON p.product_id = o3.product_id AND o3.option_id = 3
LEFT JOIN
    OPTION_VALUES o4 ON p.product_id = o4.product_id AND o4.option_id = 4
LEFT JOIN
    OPTION_VALUES o5 ON p.product_id = o5.product_id AND o5.option_id = 5
ORDER BY p.prod_name, product_description;

-- 3. Identify high-value customers with their preferences
SELECT
    c.customer_id,
```

```
  c.full_name,
  COUNT(DISTINCT o.order_number) as total_orders,
  SUM(o.total_price) as total_spent,
  ARRAY_AGG(DISTINCT cat.category_name) as favorite_categories,
  ROUND(AVG(r.rating), 2) as avg_rating_given
FROM CUSTOMER c
JOIN "ORDER" o ON c.customer_id = o.customer_id
JOIN ORDER_ITEM oi ON o.customer_id = oi.customer_id AND o.order_number = oi.order_number
JOIN PRODUCT p ON oi.product_id = p.product_id
JOIN CATEGORY cat ON p.category_id = cat.category_id
LEFT JOIN REVIEWS r ON c.customer_id = r.customer_id
WHERE o.order_date >= CURRENT_DATE - INTERVAL '365 days'
GROUP BY c.customer_id, c.full_name
HAVING SUM(o.total_price) > 1000
ORDER BY total_spent DESC;

-- 4. Monitor inventory levels and restock needs
SELECT
  p.prod_name,
  sp.sku,
  s.business_name as shop_name,
  sp.quantity as current_stock,
  COALESCE(
    SUM(oi.qty) FILTER (WHERE o.order_date >= CURRENT_DATE - INTERVAL '30 days'),
    0
  ) as monthly_sales,
  CASE
    WHEN sp.quantity = 0 THEN 'Out of Stock'
    WHEN sp.quantity < COALESCE(
      SUM(oi.qty) FILTER (WHERE o.order_date >= CURRENT_DATE - INTERVAL '30 days'),
      0
```

```
    ) * 0.5 THEN 'Critical - Restock Needed'
    WHEN sp.quantity < sp.max_purchasable_qty THEN 'Low Stock'
    ELSE 'Adequate Stock'
  END as stock_status
FROM PRODUCT p
JOIN SHOP_PRODUCT sp ON p.product_id = sp.product_id
JOIN SHOP s ON sp.shop_id = s.shop_id
LEFT JOIN ORDER_ITEM oi ON sp.product_id = oi.product_id
  AND sp.shop_id = oi.shop_id
  AND sp.sku = oi.sku
LEFT JOIN "ORDER" o ON oi.customer_id = o.customer_id
  AND oi.order_number = o.order_number
GROUP BY p.prod_name, sp.sku, s.business_name, sp.quantity, sp.max_purchasable_qty
HAVING COALESCE(
  SUM(oi.qty) FILTER (WHERE o.order_date >= CURRENT_DATE - INTERVAL '30 days'),
  0
) > 0
ORDER BY monthly_sales DESC;

-- 5. Calculate shop performance metrics
SELECT
  s.shop_id,
  s.business_name,
  COUNT(DISTINCT o.order_number) as total_orders,
  COUNT(DISTINCT c.customer_id) as unique_customers,
  SUM(o.total_price) as gross_revenue,
  SUM(o.discount) as total_discounts,
  SUM(o.total_price - COALESCE(o.discount, 0)) as net_revenue,
  ROUND(AVG(r.rating), 2) as avg_rating,
  COUNT(DISTINCT f.customer_id) as followers_count
FROM SHOP s
```

```
LEFT JOIN SHOP_PRODUCT sp ON s.shop_id = sp.shop_id
LEFT JOIN ORDER_ITEM oi ON sp.product_id = oi.product_id
   AND sp.shop_id = oi.shop_id
   AND sp.sku = oi.sku
LEFT JOIN "ORDER" o ON oi.customer_id = o.customer_id
   AND oi.order_number = o.order_number
LEFT JOIN CUSTOMER c ON o.customer_id = c.customer_id
LEFT JOIN REVIEWS r ON sp.product_id = r.product_id
   AND sp.shop_id = r.shop_id
   AND sp.sku = r.sku
LEFT JOIN FOLLOWS f ON s.shop_id = f.shop_id
WHERE o.order_date >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY s.shop_id, s.business_name
ORDER BY net_revenue DESC;
```