CS 6360.004 Database Design

Project Report: Ebay - 5

Team #5

Team Members:

Mehroos Ali (mxa200089)

Anany Dwivedi (axd210027)

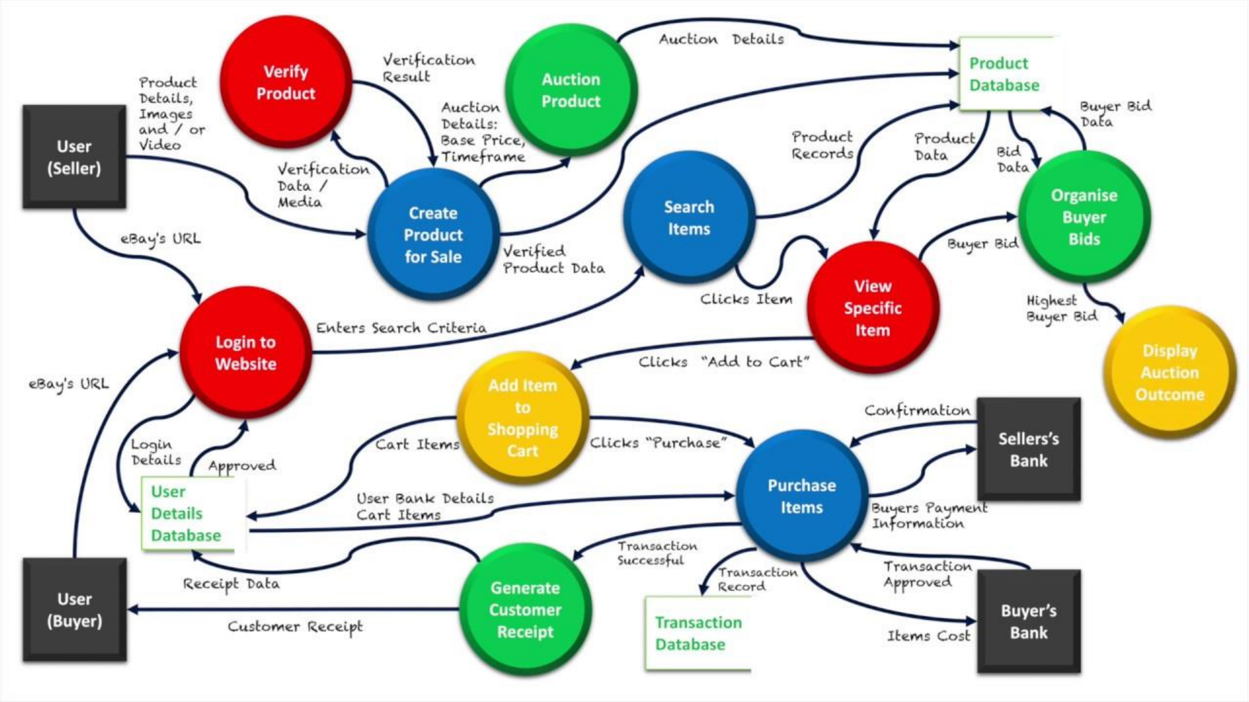
Yash Niraj Majmudar (ynm210000)

PROJECT DESCRIPTION:

eBay is an online marketplace that allows users to search and purchase products, as well as sell their own items through the online system. Trade between users can be arranged through postage for global transactions, though users can also refine their purchasing to local traders, which is established using their postcode/zip code data. eBay has been a major player in e-commerce since its rise, with its specific niche being the ability to allow for users to sell their items by having other users bid on those items. The user who bids the highest price by a specific date set by a seller would be the receiver of the specific item.

eBay performs consumer-to-consumer and business-to-consumer sales through its website which is an online auction and shopping website in which people and businesses buy and sell a wide variety of goods and services worldwide. The website is free to use for buyers, but sellers are charged fees for listing items after a limited number of free listings, and an additional or separate fee when those items are sold. eBay generates revenue by a complex system of fees for services, listing product features, and a final value fee for sales proceeds by sellers. In addition to eBay's original auction-style sales, the website has evolved and expanded to include instant "Buy It Now" shopping, shopping by Universal Product Code, ISBN and other services.

Context and Data Flow Diagram of eBay Market place can be represented as below:



DATA REQUIREMENTS:

The database system fulfils the following functional requirements –

1. A **user** identified by a **user_id** can create a **user account** with a **username** and **password**. The **user account** must be of the following type:
   - **Individual** – An account for personal use. User must enter the **first name** and **last name** at the time of registration.
   - **Business** – A corporate account used by various business types. The business registering under this type of account should enter their **business name**, **company url** and **registration number**.
2. A **user** can register as a **buyer** or a **seller** or both.
3. A **user** can have multiple **shipping addresses**. An address has a surrogate key **contact id, street name, apartment number, city, zip code, country, country code** and **phone number**. A Boolean field should indicate if the address is a **default** address or not.
4. A **seller** has a **description, selling limit, feedback score** and **number of items sold**. A **seller** also has an **average rating** attribute derived from the **ratings** it receives from the **buyers**.
5. A **seller** can **place** multiple **products** for sale.
6. A **seller** can have multiple **bank account** details with an **account number** and **routing number**.
7. A **buyer** can add **products** to the **cart**. The **cart** indicates the **quantity** of each **product** and the **total cost** of each **product**.
8. A **buyer** can **watch** multiple **product** he or she wishes to buy in future and be notified about its availability and price change.
9. A **buyer** can **bid** on multiple **auction product**. This bidding is represented by a **bid amount, bid time,** and **bid status**.
10. A **buyer** can **review** multiple **products** with a **rating, comment,** and **multiple review images**.
11. A **buyer** can **review** multiple **sellers** with a **rating** and **comment**.
12. A **buyer** can **place** an **order**. Each **order** contains an **order id, order status, shipping status, order date, shipping cost, estimated delivery date, tracking id** and **delivered date**.
13. Each **order** can have multiple **product** and this is indicated by the **quantity** and total **selling price** of each **product**.
14. Each **order** has a card **payment** details. Payment details include **card number, card type, card holder name**, and **expiry date**. A Boolean field should indicate if the card is a **default** or not.
15. Each **order** is also shipped by a particular **shipper**. The **shipper** has a **shipping id**, **company name, email address, website,** and a **phone number**.

16. A **product** is uniquely identified by a **product id**. It has a **description, name, condition, available units, users watching** and **images**. A **product** also has an **average rating** attribute derived from the **ratings** it receives from the **buyers**. A **product** must be of the following type:
    - **Fixed Price** – has a fixed **price** and an optional **discount**.
    - **Auction** – Products which are **placed** for the auction by a **seller**. It has a **starting price, current price or final price, number of bids, start date, end date** and a **winning buyer**.
17. A **product** belongs to a **subcategory** which is part of multiple **categories**.

RELATIONSHIPS

1. **Ebay_User -HAS- User_Account:** each user can have one account, and one account must be linked to only one user. Thus, cardinality is 1: 1.

2. **Ebay_User -HAS- Shipping_Address:** each Ebay_User can have multiple Shipping_Address, while each Shipping_Address is associated with only Ebay_User. Thus, cardinality is 1: N.

3. **Buyer -REVIEWS- Product:** Buyer can review multiple products and each product can be reviewed by multiple buyers. Thus, cardinality is M: N.

4. **Buyer -REVIEWS- Seller:** Buyer can give review for multiple sellers and each seller can be reviewed by multiple buyers. Thus, cardinality is M: N.

5. **Buyer -ADDS_TO- Cart:** each buyer can have 0 or 1 shopping cart while each shopping cart is associated with 1 buyer. Thus, cardinality is 1:1.

6. **Buyer -WATCHES- Product:** Buyer can watch multiple products and each product can be watched by multiple buyers. Thus, cardinality is M: N.

7. **Cart -CONTAINS- Product:** each cart contains many products, and each product can be in many carts. Thus, cardinality is N: M.

8. Buyer -BIDS- Auction_Product: Buyer can give bid on multiple auction product and each auction product can have multiple bidders. Thus, cardinality is M: N.

9. Buyer -PLACES- Order: buyer may or may not place an order. A buyer can place many orders, while each order is linked with only 1 buyer. Thus, cardinality is 1: N.

10. Seller -HAS- Bank_Details: Seller can be associated with multiple bank details. Bank_Details must be linked to only one seller. Thus, cardinality is 1: N.

11. Seller -PLACES- Product: seller sells multiple products. Each seller may not sell anything or can sell many products, while each product has only 1 seller. Thus, cardinality is 1: N**.**

12. Product -HAS- Sub_Category: Product is part of only one subcategory and each subcategory may have multiple products. Thus, cardinality is 1: N.

13. Category -HAS- Sub_Category: A category can have multiple subcategories and a subcategory can be a part of multiple categories. Thus, cardinality is M: N.

14. **Order -HAS- Product:** Order may contain multiple products and each product can be linked with multiple order. Thus, cardinality is M: N.

15. **Order -Paid_By- Payment:** Each order must be paid using single payment method and a payment method can be used to place multiple orders. Thus, cardinality is 1: N.

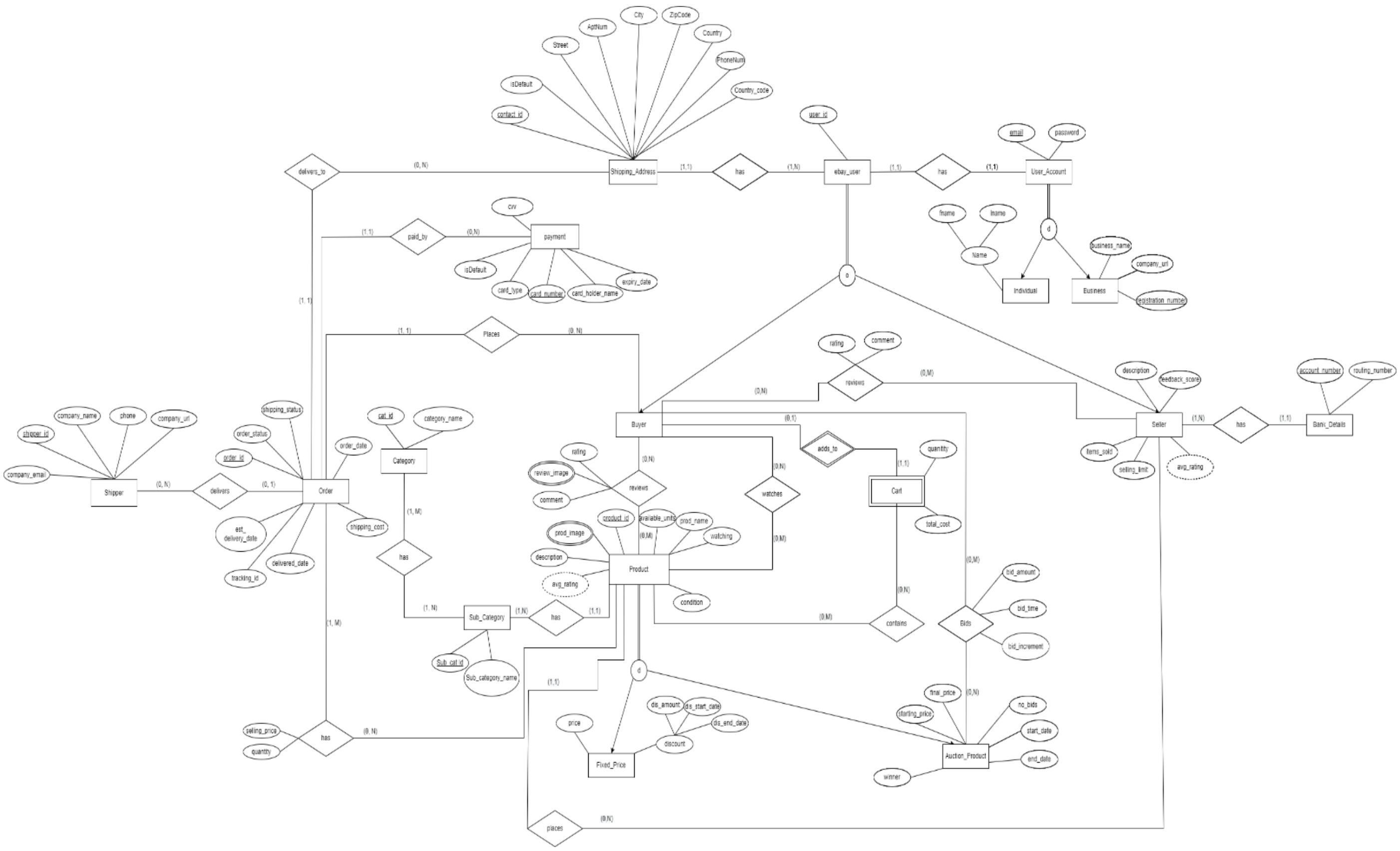
16. **Order -Delivers_To- Shipping_Address:** each order has one shipping address, while each shipping address is linked with multiple orders. Thus, cardinality is 1: N.

17. **Shipper -Delivers- Order**: Shipper can deliver multiple order and order must be delivered by one shipper. Thus, cardinality is 1: N.

- Number 1:1 relationship = 2
- Number of 1: N relationships = 8
- Number of N: M relationships = 7
- Total Relationships = 17

ENHANCED ENTITY RELATION DIAGRAM (EER MODEL):

IMAGE URL: https://drive.google.com/file/d/1--gmuDRbOdMV8tNkxBvbXZ7Sle7mo4No/view?usp=sharing



PFA the EER image with the submission.

MAPPING EER DIAGRAM TO RELATIONAL MODEL:

To map EER diagram into a relational schema, we considered the following mapping rules:

- For each 1: 1 binary relationship, in the total participation entity add the primary key of the other entity as the foreign key.
- For 1: N binary relationship, add to the entity on the N side the primary key of the other entity as the foreign key.
- 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.
- For mapping weak entities, the primary key of the owner entity is added as foreign key in the weak entity relation.
- For each multivalued attribute, create a new relation which will include the primary key of the relation the attribute is part of as a foreign key.
- For composite attributes include the last level of the attributes in the relation.
- For Overlapping and Disjoint specializations, a new relation is created with the primary key of this relation same as that of the primary key of the parent relation.

After applying these rules, the following changes were done for the mapping:

- In User_Account we have user_id as foreign key.
- In Business, we have email as foreign key and primary key.
- In Shipping_Address, we have user_id as foreign key.
- In Country, we have country_code as foreign key.
- In Buyer, we have buyer_id as foreign key and primary_key.
- In Seller, we have seller_id as foreign key and primary_key.
- In Bank_Detail, we have seller_id as foreign key.
- In Product, we have seller_id and sub_cat_id as foreign keys.
- In Product_Images, we have product_id as foreign key.
- In Buyer_Reviews_Product, we have buyer_id and product_id as foreign keys.
- In Buyer_Reviews_Seller, we have buyer_id and seller_id as foreign keys.
- In Review_Images ,we have buyer_id and product_id as foreign keys.
- In Cart, we have buyer_id as foreign key.
- In Watches, we have buyer-id and product_id as foreign keys.
- In Bids, we have product_id and buyer_id as foreign keys.
- In Cart_Contains_Product, we have buyer_id and product_id as foreign keys.
- In Sub_Category, we have product_id as foreign key.
- In Category_Has_Sub_Category, we have sub_cat_id and cat_id as foreign key.
- In Order, we have card_no, buyer_id, shipper_id and contact_id as foreign key.
- In Tracking, we have tracking_id as foreign key.
- In Order_Detail, we have order_id as foreign key.
- In Order _Has_Product, we have product_id and order_id as foreign key.

Ebay_User

user_id

User_Account

email	password	*user_id*
		FK -> Ebay_User

Individual

email	first_name	last_name

Business

registration_number	*email*	business_name	company_url
	FK -> User_Account		

Buyer

buyer_id
FK -> Ebay_User

Seller

seller_id	description	items_sold	selling_limit	feedback_score
FK -> Ebay_User				

Shipping_Address

contact_id	street	apt_no	city	zipcode	country_code	phone_no	isDefault	*user_id*
								FK -> Ebay_User

Country

country_code	country
FK -> Shipping_Address	

Payment

card_no	card_type	card_holder_name	cvv	expiry_date	isDefault

Bank_Details

account_number	routing_number	*seller_id*
		FK -> Seller

Product

product_id	prod_name	condition	description	available_units	*seller_id*	*sub_cat_id*
					FK -> Seller	FK -> Sub_Category

Fixed_Price

product_id	dis_amount	dis_start_date	dis_end_date	price

Auction_Product

product_id	start_date	end_date	starting_price	no_bids	winner	final_price

Product_Images

product_id	prod_image
FK -> Product	

Buyer_Reviews_Product

buyer_id	product_id	rating	comment
FK -> Buyer	FK -> Product		

Buyer_Reviews_Seller

buyer_id	seller_id	rating	comment
FK -> Buyer	FK -> Seller		

Review_Images

buyer_id	product_id	review_image
FK -> Buyer	FK -> Product	

Cart

buyer_id	quantity	total_price
FK -> Buyer		

Watches

buyer_id	product_id
FK -> Buyer	FK -> Product

Bids

product_id	buyer_id	bid_amount	bid_time	bid_status
FK -> Product	FK -> Buyer			

Cart_Contains_Product

buyer_id	product_id
FK -> Buyer	FK -> Product

Category

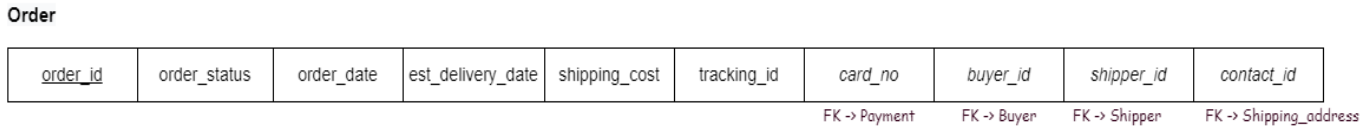
cat_id	category_name

Sub_Category

sub_cat_id	sub_category_name	*product_id*
		FK -> Product

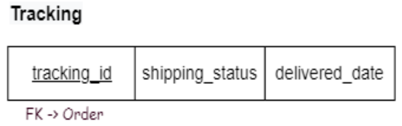
Category_Has_Sub_Category

sub_cat_id	cat_id
FK -> Sub_Category	FK -> Category



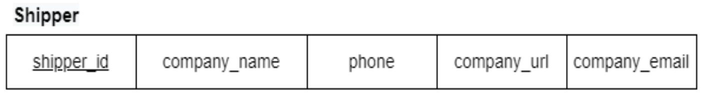
Order

order_id	order_status	order_date	est_delivery_date	shipping_cost	tracking_id	*card_no*	*buyer_id*	*shipper_id*	*contact_id*
						FK -> Payment	FK -> Buyer	FK -> Shipper	FK -> Shipping_address



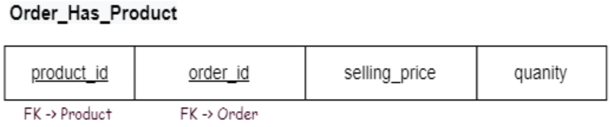
Tracking

tracking_id	shipping_status	delivered_date
FK -> Order		



Shipper

shipper_id	company_name	phone	company_url	company_email



Order_Has_Product

product_id	order_id	selling_price	quanity
FK -> Product	FK -> Order		

The above relations have been normalized after resolving 3NF violations.

FUNCTIONAL DEPENDENCIES AND NORMALIZATION:

Functional Dependencies (before Normalization):

User_Account

email -> password, user_id

Individual

email -> first_name, last_name

Business

registration_number -> email, business_name, company_url

every attribute is unique. every attribute can determine every other attribute i.e., all the attributes are super keys. Hence there is no 3NF violation.

Seller

seller_id -> description, items_sold, selling_limit, feedback_score

Shipping_Address

contact_id -> street, apt_no, city, zipcode, county, country_code, phone_no, isDefault, user_id

country_code -> country

country code can determine country. This is a 3NF violation. We create a separate table Country in the relational schema.

Payment

card_no -> card_type, card_holder_name, cvv, expiry_date, isDefault

Bank_Details

account_no -> rounting_number, seller_id

Product

product_id -> prod_name, condition, description, available_units, seller_id, sub_cat_id

Fixed_Price

product_id -> dis_amount, dis_start_date, dis_end_date, price

Auction_Product

product_id -> start_date, end_date, starting_price, no_bids, winner, final_price

Product_Images

product_id -> prod_image

Buyer_Reviews_Product

buyer_id, product_id -> rating, comment

Buyer_Reviews_Seller

buyer_id, seller_id -> rating, comment

Review_Images

buyer_id, product_id -> review_image

Cart

buyer_id -> quantity, total_price

Bids

product_id, buyer_id -> bid_amount, bid_time, bid_increment

Category

cat_id -> category_name

Sub_Category

sub_cat_id -> sub_category_name, product_id

Order

order_id -> order_status, order_date, est_delivery_date, delivered_date, shipping_cost, tracking_id, shipping_status, card_no, buyer_id, shipper_id, contact_id

tracking_id -> shipping_status, delivered_date

There is a transitive dependency order_id -> tracking_id -> shipping_status, delivered_date.

This is a 3NF violation. We create a separate table Tracking in the relational schema.

Order_Detail

order_id -> selling_price, quantity

Shipper

shipper_id -> company_name, phone, company_url, company_email

Normalization:

1NF

The relations are already in 1NF.

2NF

The relations are already in 2NF.

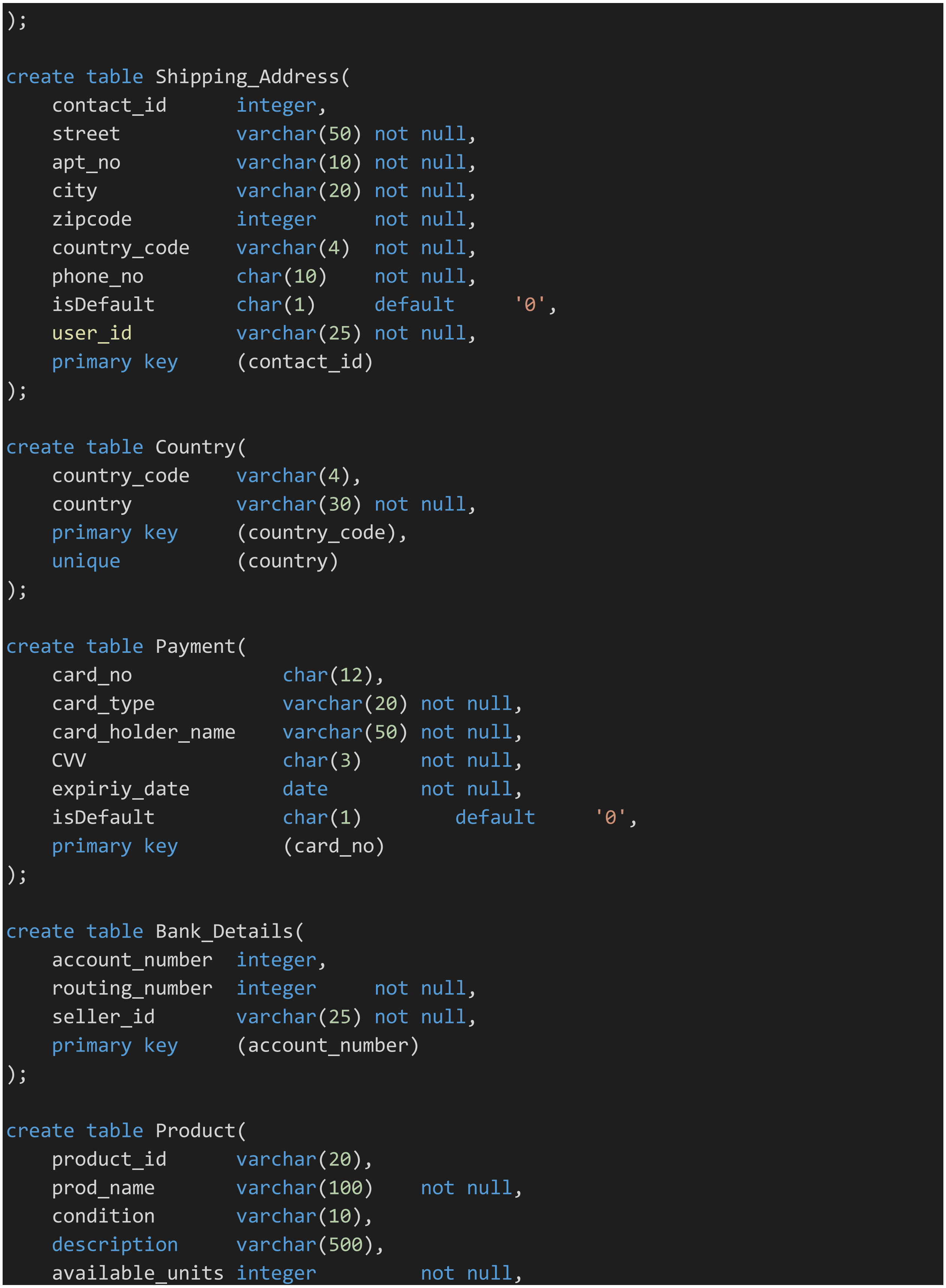
3NF

There are 3NF violations in tables Shipping_Address and Order which is resolved in the relational schema.

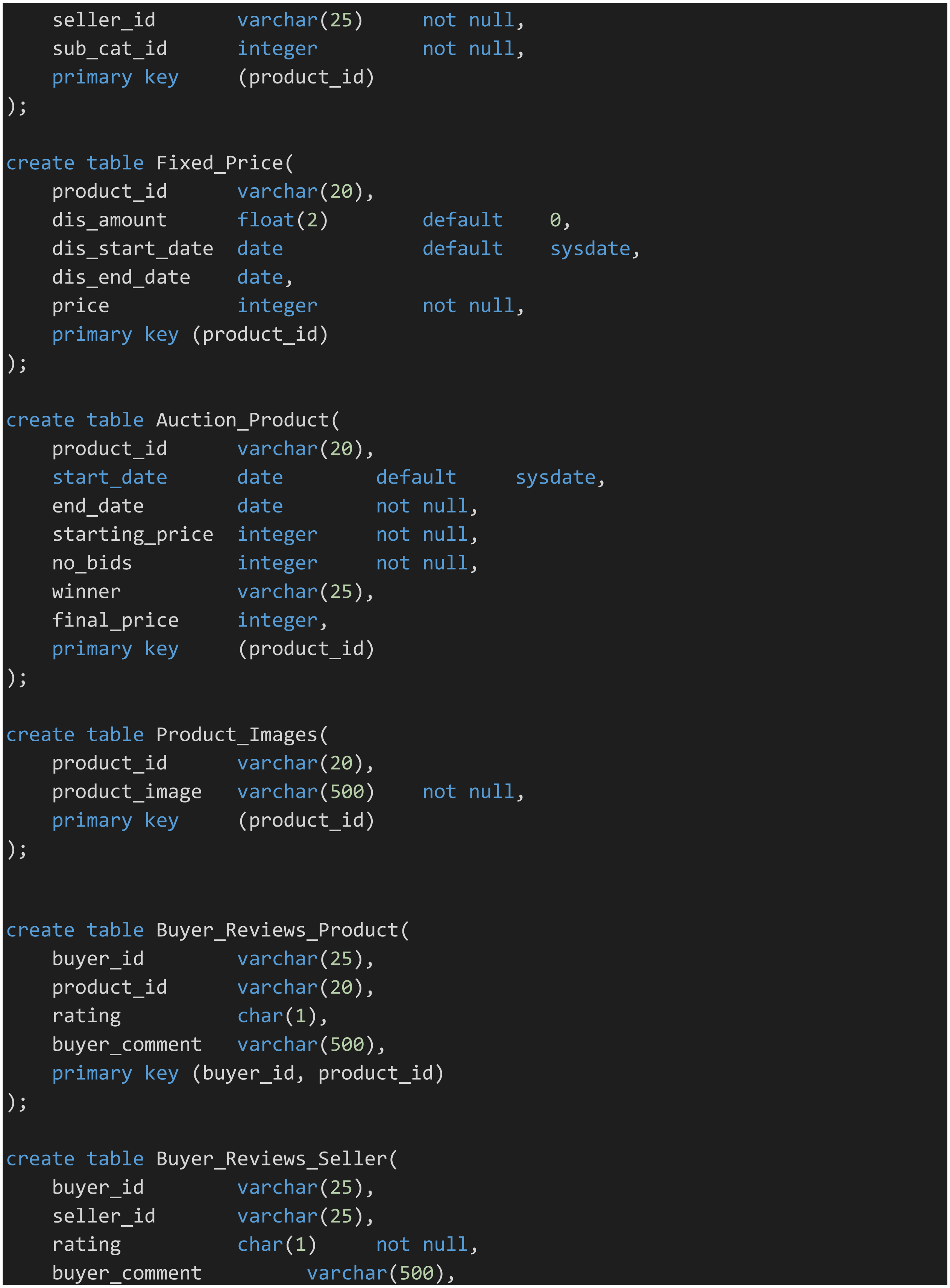
SQL CODE:

Creating Tables

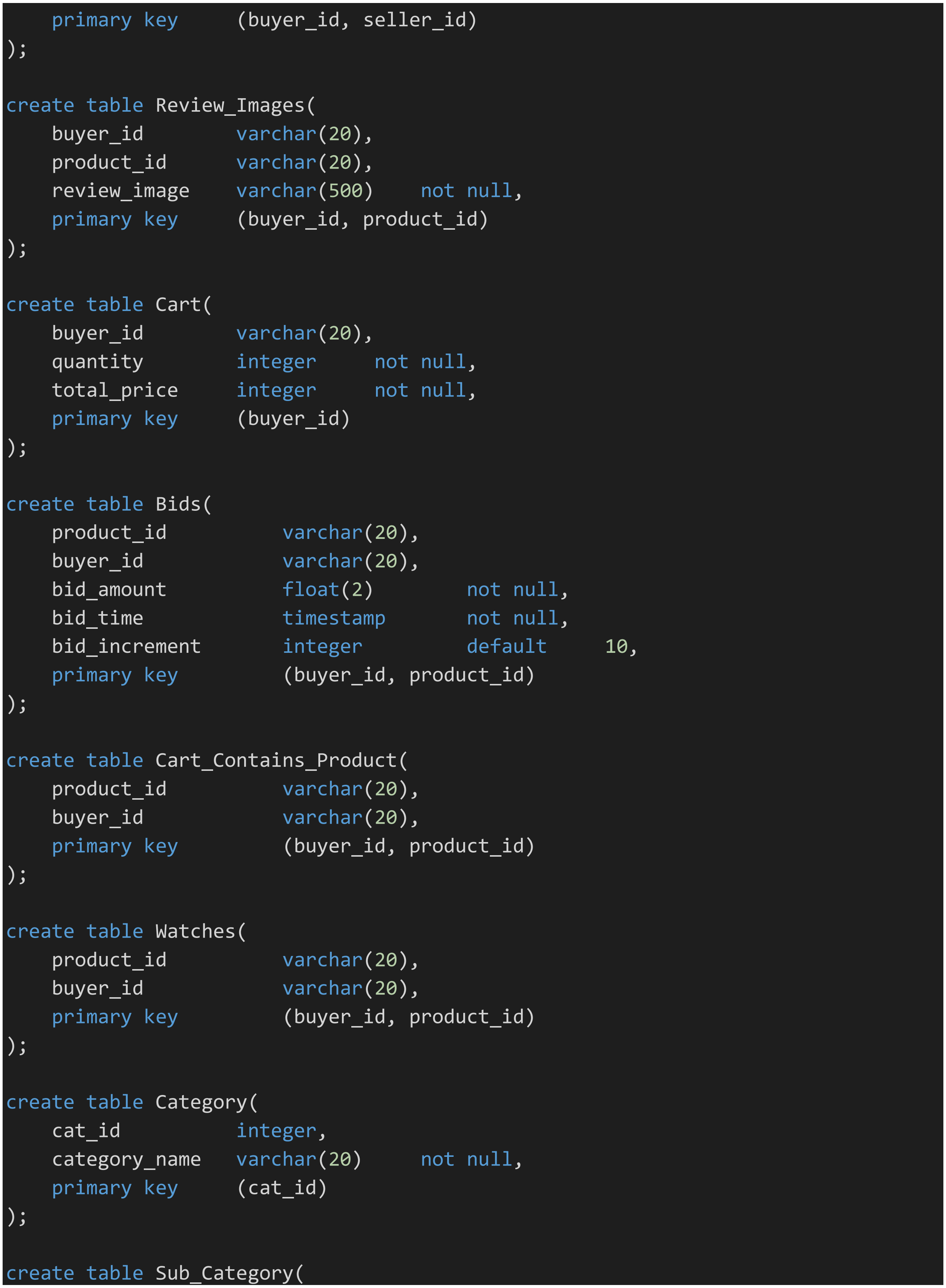
```
create table Ebay_User(
    user_id     varchar(25),
    primary key (user_id)
);

create table User_Account(
    email       varchar(50),
    password    varchar(50) not null,
    user_id     varchar(25) not null,
    primary key (email)
);

create table Individual(
    email       varchar(50),
    first_name  varchar(50) not null,
    last_name   varchar(50) not null,
    primary key (email)
);

create table Business(
    registration_number char(9),
    email               varchar(50),
    business_name       varchar(50) not null,
    company_url         varchar(50) not null,
    primary key         (registration_number),
    unique              (company_url),
    unique              (business_name)
);

create table Buyer(
    buyer_id    varchar(25),
    primary key (buyer_id)
);

create table Seller(
    seller_id       varchar(25),
    description     varchar(1000)   not null,
    items_sold      integer         not null,
    selling_limit   integer         default     100,
    feedback_score  smallint,
    primary key     (seller_id)
```



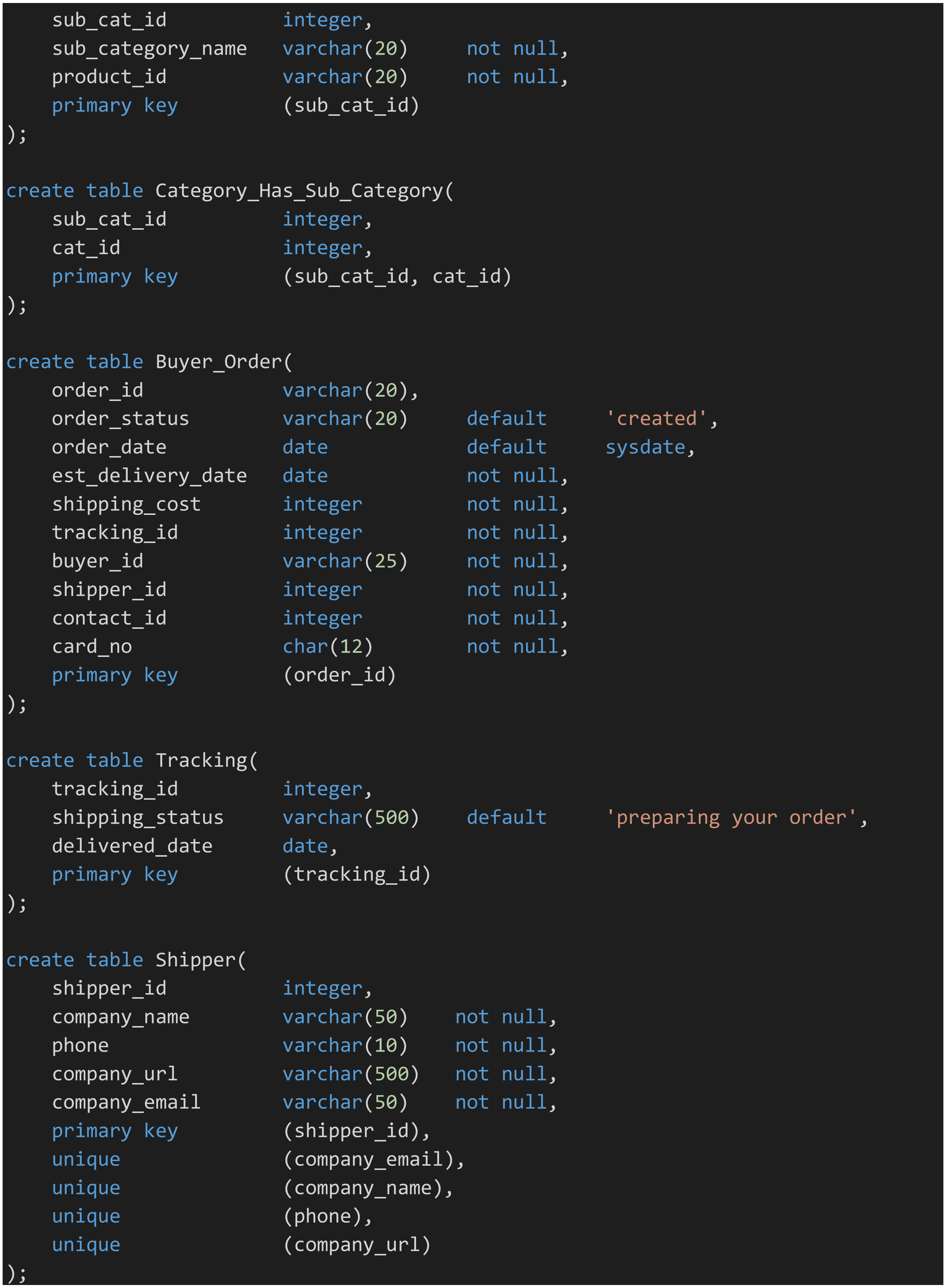
```
);

create table Shipping_Address(
    contact_id      integer,
    street          varchar(50) not null,
    apt_no          varchar(10) not null,
    city            varchar(20) not null,
    zipcode         integer     not null,
    country_code    varchar(4)  not null,
    phone_no        char(10)    not null,
    isDefault       char(1)     default     '0',
    user_id         varchar(25) not null,
    primary key     (contact_id)
);

create table Country(
    country_code    varchar(4),
    country         varchar(30) not null,
    primary key     (country_code),
    unique          (country)
);

create table Payment(
    card_no             char(12),
    card_type           varchar(20) not null,
    card_holder_name    varchar(50) not null,
    CVV                 char(3)     not null,
    expiriy_date        date        not null,
    isDefault           char(1)        default     '0',
    primary key         (card_no)
);

create table Bank_Details(
    account_number  integer,
    routing_number  integer     not null,
    seller_id       varchar(25) not null,
    primary key     (account_number)
);

create table Product(
    product_id      varchar(20),
    prod_name       varchar(100)    not null,
    condition       varchar(10),
    description     varchar(500),
    available_units integer         not null,
```



```
    seller_id       varchar(25)     not null,
    sub_cat_id      integer         not null,
    primary key     (product_id)
);

create table Fixed_Price(
    product_id      varchar(20),
    dis_amount      float(2)        default    0,
    dis_start_date  date            default    sysdate,
    dis_end_date    date,
    price           integer         not null,
    primary key (product_id)
);

create table Auction_Product(
    product_id      varchar(20),
    start_date      date        default     sysdate,
    end_date        date        not null,
    starting_price  integer     not null,
    no_bids         integer     not null,
    winner          varchar(25),
    final_price     integer,
    primary key     (product_id)
);

create table Product_Images(
    product_id      varchar(20),
    product_image   varchar(500)    not null,
    primary key     (product_id)
);


create table Buyer_Reviews_Product(
    buyer_id        varchar(25),
    product_id      varchar(20),
    rating          char(1),
    buyer_comment   varchar(500),
    primary key (buyer_id, product_id)
);

create table Buyer_Reviews_Seller(
    buyer_id        varchar(25),
    seller_id       varchar(25),
    rating          char(1)     not null,
    buyer_comment         varchar(500),
```



```
    primary key     (buyer_id, seller_id)
);

create table Review_Images(
    buyer_id        varchar(20),
    product_id      varchar(20),
    review_image    varchar(500)    not null,
    primary key     (buyer_id, product_id)
);

create table Cart(
    buyer_id        varchar(20),
    quantity        integer     not null,
    total_price     integer     not null,
    primary key     (buyer_id)
);

create table Bids(
    product_id          varchar(20),
    buyer_id            varchar(20),
    bid_amount          float(2)        not null,
    bid_time            timestamp       not null,
    bid_increment       integer         default     10,
    primary key         (buyer_id, product_id)
);

create table Cart_Contains_Product(
    product_id          varchar(20),
    buyer_id            varchar(20),
    primary key         (buyer_id, product_id)
);

create table Watches(
    product_id          varchar(20),
    buyer_id            varchar(20),
    primary key         (buyer_id, product_id)
);

create table Category(
    cat_id          integer,
    category_name   varchar(20)     not null,
    primary key     (cat_id)
);

create table Sub_Category(
```



```
    sub_cat_id          integer,
    sub_category_name   varchar(20)     not null,
    product_id          varchar(20)     not null,
    primary key         (sub_cat_id)
);

create table Category_Has_Sub_Category(
    sub_cat_id          integer,
    cat_id              integer,
    primary key         (sub_cat_id, cat_id)
);

create table Buyer_Order(
    order_id            varchar(20),
    order_status        varchar(20)     default     'created',
    order_date          date            default     sysdate,
    est_delivery_date   date            not null,
    shipping_cost       integer         not null,
    tracking_id         integer         not null,
    buyer_id            varchar(25)     not null,
    shipper_id          integer         not null,
    contact_id          integer         not null,
    card_no             char(12)        not null,
    primary key         (order_id)
);

create table Tracking(
    tracking_id         integer,
    shipping_status     varchar(500)    default     'preparing your order',
    delivered_date      date,
    primary key         (tracking_id)
);

create table Shipper(
    shipper_id          integer,
    company_name        varchar(50)    not null,
    phone               varchar(10)    not null,
    company_url         varchar(500)   not null,
    company_email       varchar(50)    not null,
    primary key         (shipper_id),
    unique              (company_email),
    unique              (company_name),
    unique              (phone),
    unique              (company_url)
);
```

```
create table Order_Has_Product(
    product_id               varchar(20),
    order_id                 varchar(20),
    selling_price            integer         not null,
    quantity                 integer         not null,
    primary key              (product_id, order_id)
);
```

Adding foreign keys

```
alter table User_Account add constraint fkuapkeu foreign key (user_id) REFERENCES
Ebay_User(user_id) on delete cascade;
alter table Business add constraint fkbpkua foreign key (email) REFERENCES
User_Account(email) on delete cascade;
alter table Shipping_Address add constraint fksapkeu foreign key (user_id)
REFERENCES Ebay_User(user_id);
alter table Shipping_Address add constraint fksapkc foreign key (country_code)
REFERENCES Country(country_code);
alter table Bank_Details add constraint fkbdpks foreign key (seller_id)
REFERENCES Seller(seller_id);
alter table Product add constraint fkppks foreign key (seller_id) REFERENCES
Seller(seller_id);
alter table Product add constraint fkppksc foreign key (sub_cat_id) REFERENCES
Sub_Category(sub_cat_id);
alter table Product_Images add constraint fkpipkp foreign key (product_id)
REFERENCES Product(product_id);
alter table Buyer_Reviews_Product add constraint fkbrppkb foreign key (buyer_id)
REFERENCES Buyer(buyer_id);
alter table Buyer_Reviews_Product add constraint fkbrppkp foreign key
(product_id) REFERENCES Product(product_id);
alter table Buyer_Reviews_Seller add constraint fkbrspks foreign key (seller_id)
REFERENCES Seller(seller_id);
alter table Buyer_Reviews_Seller add constraint fkbrspkb foreign key (buyer_id)
REFERENCES Buyer(buyer_id);
alter table Review_Images add constraint fkripkb foreign key (buyer_id)
REFERENCES Buyer(buyer_id);
alter table Review_Images add constraint fkripkp foreign key (product_id)
REFERENCES Product(product_id);
alter table Cart add constraint fkbipkb foreign key (buyer_id) REFERENCES
Buyer(buyer_id);
alter table Bids add constraint fkbspkp foreign key (product_id) REFERENCES
Product(product_id);
```

```
alter table Bids add constraint fkbsspkp foreign key (buyer_id) REFERENCES
Buyer(buyer_id);
alter table Cart_Contains_Product add constraint fkccppkb foreign key (buyer_id)
REFERENCES Buyer(buyer_id);
alter table Cart_Contains_Product add constraint fkccppkp foreign key
(product_id) REFERENCES Product(product_id);
alter table Watches add constraint fkwlcppkp foreign key (product_id) REFERENCES
Product(product_id);
alter table Watches add constraint fkwlcppkb foreign key (buyer_id) REFERENCES
Buyer(buyer_id);
alter table Sub_Category add constraint fkscpkp foreign key (product_id)
REFERENCES Product(product_id);
alter table Category_Has_Sub_Category add constraint fkchscpksc foreign key
(sub_cat_id) REFERENCES Sub_Category(sub_cat_id);
alter table Category_Has_Sub_Category add constraint fkchscpkc foreign key
(cat_id) REFERENCES Category(cat_id);
alter table Buyer_Order add constraint fkopkpy foreign key (card_no) REFERENCES
Payment(card_no);
alter table Buyer_Order add constraint fkopkb foreign key (buyer_id) REFERENCES
Buyer(buyer_id);
alter table Buyer_Order add constraint fkopkpsh foreign key (shipper_id)
REFERENCES Shipper(shipper_id);
alter table Buyer_Order add constraint fkopkpsa foreign key (contact_id)
REFERENCES Shipping_Address(contact_id);
alter table Buyer_Order add constraint fkto foreign key (tracking_id) REFERENCES
Tracking(tracking_id);
alter table Order_Detail add constraint fkodpko foreign key (order_id) REFERENCES
Buyer_Order(order_id);
alter table Order_Has_Product add constraint fkohppkp foreign key (product_id)
REFERENCES Product(product_id);
alter table Order_Has_Product add constraint fkohppko foreign key (order_id)
REFERENCES Buyer_Order(order_id);
```

PL/SQL:

Stored Procedures

1. Procedure to print the available units of a given product.

The procedure below takes in a product id and prints its available units in the inventory, available units are also stored in a OUT variable so that it can be used outside the function.

```
create or replace PROCEDURE Get_Available_Units(id IN
Fixed_Price.product_id%type, num_units OUT integer) AS
thisProduct Product.prod_name%TYPE;
BEGIN
    select P.prod_name INTO thisProduct from Product P where P.product_id = id;
    select P.available_units INTO num_units from Product P where p.product_id =
id;
    dbms_output.put_line(thisProduct || ' has ' || num_units || ' available
units.');
END;
```

2. Procedure to update discount for a given product.

This procedure updated the discount value for a given product and also sets its validity period.

```
create or replace PROCEDURE UpdateDiscount(id IN Fixed_Price.product_id%type,
discount IN Fixed_Price.dis_amount%type,
            st_date IN Fixed_Price.dis_start_date%type, end_date IN
Fixed_Price.dis_end_date%type) AS

    thisProduct Product.prod_name%TYPE;

BEGIN
    SELECT P.prod_name INTO thisProduct FROM Product P WHERE P.product_id = id;
    UPDATE Fixed_Price F SET F.dis_amount = discount, F.dis_start_date = st_date,
F.dis_end_date = end_date WHERE F.product_id = id;

    dbms_output.put_line('Discount for ' || thisProduct || ' has been set to ' ||
discount);
END;
```

Triggers

1. Trigger to reduce the quantity of the product units after order is placed.

The trigger reduces the product available units by reducing it by the quantity of that product in the order.

```
 create or replace TRIGGER Update_Available_Units

AFTER INSERT ON Order_Has_Product
FOR EACH ROW

DECLARE
    num_units integer;
    id  Buyer.buyer_id%type;
    prod_quant  integer;

CURSOR get_products IS
    select product_id, quantity from Order_Has_Product where order_id =
:new.order_id;

BEGIN
    OPEN get_products;
    LOOP
        FETCH get_products INTO id, prod_quant;
        EXIT WHEN get_products%NOTFOUND;

        select p.available_units into num_units from Product p where p.product_id
= id;

        if num_units > prod_quant then
            update Product p set available_units = available_units-prod_quant
where p.product_id = id;
        elsif num_units = prod_quant then
            update Product p set available_units = available_units-prod_quant
where p.product_id = id;
            dbms_output.put_line('Item is out of stock\n');
        end if;

    END LOOP;
    CLOSE get_products;
END;
```

2. Trigger to update winning status of bidders for a bid.

The following trigger checks whether the placed bid is valid or not by comparing its value with the bid amount plus bid increment, if the bid is valid then bid amount is also updated in the Bids table.

```
create or replace TRIGGER Update_Winning_Status
AFTER INSERT ON Bids
FOR EACH ROW

DECLARE
    b_amount Bids.bid_amount%TYPE;
    bid_thr Bids.bid_amount%TYPE;
    bid_inc Bids.bid_increment%TYPE;

BEGIN

    select b.bid_amount, b.bid_increment into b_amount, bid_inc from Bids b where
b.product_id = :new.product_id and b.buyer_id = :new.buyer_id;


    bid_thr = bid_amount * (1 + bid_inc/100);

    if b_amount >= bid_thr then
        update Bids set bid_amount = b_amount where product_id = :new.product_id
and bbuyer_id = :new.buyer_id;
        dbms_output.put_line('The bid is placed');
    else
        Raise_Application_Error(-20000, 'The amount is less than threshold');

END;
```