Zaiyang Zhong
Database CS669
Nov. 30

Team Project Iteration5 - Amazon Project

Iteration Requirements:

Reusable, Transaction-Oriented Store Procedures – You create and execute reusable stored procedures that complete the steps of transactions necessary to add data to your database.

Complete the SQL statements including the statements, procedures or functions needed to maintain your database.

Questions and Queries – You define questions useful to the organization or application that will use your database, then write queries to address the questions.

Data Visualizations – You tell effective data stories with data visualizations.
NOTE: SQL programs should be run to show results.

Table of Contents:

PART I: Create tables based on ER diagram

In this part, I will create 9 tables based on the ER diagram.

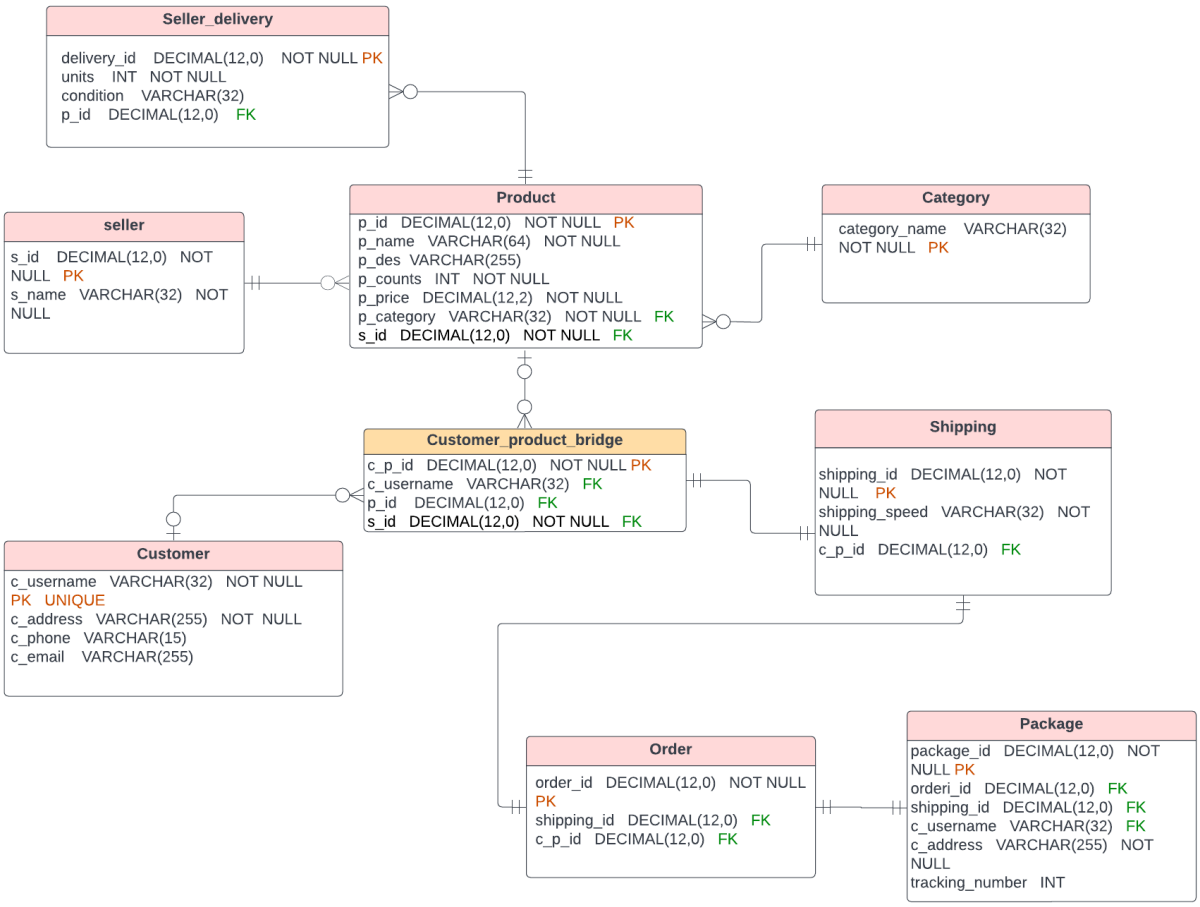

Figure 1. FINAL ER diagram

Before creating tables, I determined the order to implement. Reviewing the relationships of entities in *Figure1*, I decided the order to be.

Table1	Table2	Table3	Table4	Table5	Table6	Table7	Table8	Table9
Seller	Category	Product	Seller_delivery	Customer	Customer_product_bridge	Shipping	Order	Package

Figure 2. The order of Implementation

And they are implemented with CREATE TABLE commands.

Query Query History

```
-- populate tables 1 ~ 4
CREATE SEQUENCE seller_seq START WITH 1;
CREATE SEQUENCE product_seq START WITH 1;
CREATE SEQUENCE seller_delivery_seq START WITH 1;

```

Loading...

Query Query History

```
-- create table 1, 2
CREATE TABLE seller(
s_id DECIMAL(12,0) NOT NULL,
s_name VARCHAR(32) NOT NULL,
PRIMARY KEY (s_id)
);

CREATE TABLE category(
category_name VARCHAR(32) NOT NULL,
PRIMARY KEY (category_name)
);

```

Data output Messages Notifications

```
CREATE TABLE

Query returned successfully in 871 msec.
```

Figure 1.1. Implementation of table 1 and 2 - Seller and Category

Query Query History

```sql
-- create table 3
CREATE TABLE product(
p_id DECIMAL(12,0) NOT NULL,
p_name VARCHAR(64) NOT NULL,
p_des VARCHAR(255),
p_counts INT NOT NULL,
p_price DECIMAL(12,2) NOT NULL,
p_category VARCHAR(12) NOT NULL,
s_id DECIMAL(12,0) NOT NULL,
PRIMARY KEY (p_id),
FOREIGN KEY (p_category) REFERENCES category,
FOREIGN KEY (s_id) REFERENCES seller
);
```

Data output Messages Notifications

```
CREATE TABLE

Query returned successfully in 49 msec.
```

Figure 1.2. Implementation of table 3- product

Query Query History

```sql
-- create table 4
CREATE TABLE seller_delivery(
delivery_id DECIMAL(12,0) NOT NULL,
units INT NOT NULL,
condition VARCHAR(32),
p_id DECIMAL(12,0),
PRIMARY KEY (delivery_id),
FOREIGN KEY (p_id) REFERENCES product
);
```

Data output Messages Notifications

```
CREATE TABLE

Query returned successfully in 4 secs 44 msec.
```

Figure 1.3. Implementation of table 4 - Seller_delivery

Query Query History

```
-- create table 5
CREATE TABLE customer(
c_username VARCHAR(32) NOT NULL UNIQUE,
c_address VARCHAR(32) NOT NULL,
c_phone VARCHAR(15),
c_email VARCHAR(255),
PRIMARY KEY (c_username)
);
```

Data output Messages Notifications

```
CREATE TABLE

Query returned successfully in 51 msec.
```

Figure 1.4. Implementation of table5 - Customer

Query Query History

```
-- create table 6
CREATE TABLE customer_product_bridge(
c_p_id DECIMAL(12,0) NOT NULL,
c_username VARCHAR(32),
p_id DECIMAL(12,0),
s_id DECIMAL(12,0) NOT NULL,
PRIMARY KEY (c_p_id),
FOREIGN KEY (c_username) REFERENCES customer,
FOREIGN KEY (p_id) REFERENCES product,
FOREIGN KEY (s_id) REFERENCES seller
);
```

Loading...

Data output Messages Notifications

```
CREATE TABLE

Query returned successfully in 3 min 35 secs.
```

Figure 1.5. Implementation of table 6 - Customer_product_bridge

Query Query History

```sql

-- create table 7, 8
CREATE TABLE shipping(
shipping_id DECIMAL(12,0) NOT NULL,
shipping_speed VARCHAR(32) NOT NULL,
c_p_id DECIMAL(12,0),
PRIMARY KEY (shipping_id),
FOREIGN KEY (c_p_id) REFERENCES customer_product_bridge
);

CREATE TABLE orders(
order_id DECIMAL(12,0) NOT NULL,
shipping_id DECIMAL(12,0) NOT NULL,
c_p_id DECIMAL(12,0),
PRIMARY KEY (order_id),
FOREIGN KEY (c_p_id) REFERENCES customer_product_bridge
);
```

Figure 1.6. Implementation of table 7 and 8 - Shipping and Orders

Query Query History

```sql
-- create table 9
CREATE TABLE package(
package_id DECIMAL(12,0) NOT NULL,
order_id DECIMAL(12,0),
shipping_id DECIMAL(12,0),
c_username VARCHAR(32),
c_address VARCHAR(255) NOT NULL,
tracking_number INT,
PRIMARY KEY (package_id),
FOREIGN KEY (order_id) REFERENCES orders,
FOREIGN KEY (shipping_id) REFERENCES shipping,
FOREIGN KEY (c_username) REFERENCES customer
);
```

Data output Messages Notifications

```
CREATE TABLE

Query returned successfully in 816 msec.
```

Figure 1.7. Implementation of table 9 - Package

PART II: Populate tables

In this part, I will populate the 9 tables I have. Before doing it, I decided what I want to insert into the tables. And these sample values are:

- **Seller** (2)
  - "Furniture Home"
  - "World-changing Technologies"
- **Category** (2)
  - "Electronics"
  - "Furniture"
- **Product** (3)
  - "Refurbished iPhone 13"
  - "Wood L-shape Table"
  - "Comfortable Sofa"
- **Customers** (2)
  - "iHateJobs"
  - "iLoveApple"

And

- Customer "iHateJobs" buys Table and Sofa, in Furniture Category, from "Furniture Home"
- Customer "iLoveApple" buys iPhone13, in Electronics Category, from "World-changing Technologies"

Following are the results for the population.

Query Query History

```
INSERT INTO seller
VALUES
(nextv Loading...  q'), 'Furniture Home'),
(nextval('seller_seq'), 'World-changing Technologies');

SELECT * from seller;
```

Data output Messages Notifications

	s_id [PK] numeric	s_name character varying (32)
1	1	Furniture Home
2	2	World-changing Technologies

Figure 2.1. Populate table 1 - seller

Query Query History

```sql
-- populate table 1 - 4
INSERT INTO category
VALUES
('Furniture'), ('Electronics');

SELECT * from category;
```

Data output Messages Notifications

	category_name [PK] character varying (32)
1	Furniture
2	Electronics

Figure 2.2. Populate table 1 - category

Query Query History

```sql
INSERT INTO product(p_id, p_name, p_des, p_counts, p_price, p_category, s_id)
VALUES
(nextval('product_seq'), 'Wood L-shape Table', 'Wood table with a L-shaped that can fit in a corner in the room.',
 0, 299, 'Furniture', (SELECT s_id FROM seller WHERE s_name='Furniture Home')),
(nextval('product_seq'), 'Comfortable Sofa', 'The sofa is made of linen fabric with hardwood frame.',
 0, 599, 'Furniture', (SELECT s_id FROM seller WHERE s_name='Furniture Home')),
 (nextval('product_seq'), 'Refurbished iPhone 13', 'Apple iPhone 13',
 0, 899, 'Electronics', (SELECT s_id FROM seller WHERE s_name='World-changing Technologies'));

 SELECT * from product;
```

Data output Messages Notifications

	p_id [PK] numeric	p_name character varying (64)	p_des character varying (255)	p_counts integer	p_price numeric (12,2)	p_category character varying (12)	s_id numeric (12)
1	1	Wood L-shape Table	Wood table with a L-shaped that can fit in a corner in the room.	0	299.00	Furniture	1
2	2	Comfortable Sofa	The sofa is made of linen fabric with hardwood frame.	0	599.00	Furniture	1
3	3	Refurbished iPhone 13	Apple iPhone 13	0	899.00	Electronics	2

Figure 2.3. Populate table 3 - Product

Query Query History

```sql
INSERT INTO seller_delivery(delivery_id, units, condition, p_id)
VALUES
(nextval('seller_delivery_seq'), 3, 'New', (SELECT p_id FROM product WHERE p_name='Wood L-shape Table')),
(nextval('seller_delivery_seq'), 1, 'New', (SELECT p_id FROM product WHERE p_name='Comfortable Sofa')),
(nextval('seller_delivery_seq'), 1, 'Like New', (SELECT p_id FROM product WHERE p_name='Refurbished iPhone 13'));

SELECT * from seller_delivery;
```

Data output Messages Notifications

	delivery_id [PK] numeric (12)	units integer	condition character varying (32)	p_id numeric (12)
1	1	3	New	1
2	2	1	New	2
3	3	1	Like New	3

Figure 2.4. Populate table 4 - Seller_delivery

Query Query History

```sql
-- populate table 5~9

INSERT INTO customer
VALUES
('iLoveApple', '1000 Commenwealth Ave, Boston, MA', '5082947592', 'applesheaven@google.com'),
('iHateJobs', '27 West 4th Street, New York, NY', '2539683659', 'privateaccount@google.com');

SELECT * from customer;
```

Loading...

Data output Messages Notifications

	c_username [PK] character varying (32)	c_address character varying (255)	c_phone character varying (15)	c_email character varying (255)
1	iLoveApple	1000 Commenwealth ...	5082947592	applesheaven@google....
2	iHateJobs	27 West 4th Street, Ne...	2539683659	privateaccount@googl...

Figure 2.5. Populate table 5 - Customer

Query Query History

```sql
-- sequence for table 5 - 9
CREATE SEQUENCE customer_product_seq START WITH 1;
CREATE SEQUENCE shipping_id START WITH 1;
CREATE SEQUENCE order_id START WITH 1;
CREATE SEQUENCE package_id START WITH 1;

-- iLoveApple buys iPhone13,
-- iHateJobs buys sofa and table
-- manutally (without procedure) populate the bridge
INSERT INTO customer_product_bridge
VALUES
(nextval('customer_product_seq'), 'iLoveApple',
 (SELECT p_id FROM product WHERE p_name='Refurbished iPhone 13'),
 (SELECT s_id FROM product WHERE p_name='Refurbished iPhone 13'));

INSERT INTO customer_product_bridge
VALUES
(nextval('customer_product_seq'), 'iHateJobs',
 (SELECT p_id FROM product WHERE p_name='Wood L-shape Table'),
 (SELECT s_id FROM product WHERE p_name='Wood L-shape Table'));

INSERT INTO customer_product_bridge
VALUES
(nextval('customer_product_seq'), 'iHateJobs',
 (SELECT p_id FROM product WHERE p_name='Comfortable Sofa'),
 (SELECT s_id FROM product WHERE p_name='Comfortable Sofa'));

```

Data output Messages Notifications

```
INSERT 0 1

Query returned successfully in 171 msec.
```

Figure 2.6. Populate table 5 - customer_product_bridge

Query Query History

```
-- manually populate shipping, order, and package
INSERT INTO shipping
VALUES
(nextval('shipping_seq'), '2 Day premium', 1),   -- iPhone
(nextval('shipping_seq'), '2 day premium', 2),   -- table
(nextval('shipping_seq'), 'Same Day', 3);  -- Sofa

Loading...  shipping;
```

Data output Messages Notifications

	shipping_id [PK] numeric (12)	shipping_speed character varying (32)	c_p_id numeric (12)
1	1	2 Day premium	1
2	2	2 day premium	2
3	3	Same Day	3

Figure 2.7. Populate table 7 - Shipping

Query Query History

```
-- manually populate shipping, order, and package

INSERT INTO orders
VALUES
(nextval('order_seq'), (SELECT shipping_id FROM shipping WHERE c_p_id=1), 1),  -- iphone
(nextval('order_seq'), (SELECT shipping_id FROM shipping WHERE c_p_id=2), 2),  -- table
(nextval('order_seq'), (SELECT shipping_id FROM shipping WHERE c_p_id=3), 3);  -- sofa

SELECT * from orders;
```

Loading...

Data output Messages Notifications

	order_id [PK] numeric (12)	shipping_id numeric (12)	c_p_id numeric (12)
1	1	1	1
2	2	2	2
3	3	3	3

Figure 2.8. Populate table 8 - Order

Query Query History

```
INSERT INTO package VALUES
(nextval('package_seq'),   -- PK
 1,                        -- order 1
 (SELECT shipping_id FROM shipping WHERE c_p_id=1),   -- FK
 (SELECT c_username FROM customer_product_bridge WHERE c_p_id=1), -- FK
 (SELECT c_address FROM customer WHERE customer.c_username =
  (SELECT c_username FROM customer_product_bridge WHERE c_p_id=1)),  --FK
 0);  -- default tracking number

INSERT INTO package VALUES
(nextval('package_seq'),   -- PK
 2,                        -- order 1
 (SELECT shipping_id FROM shipping WHERE c_p_id=2),   -- FK
 (SELECT c_username FROM customer_product_bridge WHERE c_p_id=2), -- FK
 (SELECT c_address FROM customer WHERE customer.c_username =
  (SELECT c_username FROM customer_product_bridge WHERE c_p_id=2)),  --FK
 0);  -- default tracking number

INSERT INTO package
VALUES
(nextval('package_seq'),   -- PK
 3,                        -- order 1
 (SELECT shipping_id FROM shipping WHERE c_p_id=2),   -- FK
 (SELECT c_username FROM customer_product_bridge WHERE c_p_id=3), -- FK
 (SELECT c_address FROM customer WHERE customer.c_username =
  (SELECT c_username FROM customer_product_bridge WHERE c_p_id=3)),  --FK
 0);  -- default tracking number

 SELECT * from package;
```

Data output Messages Notifications

	package_id [PK] numeric (12)	order_id numeric (12)	shipping_id numeric (12)	c_username character varying (32)	c_address character varying (255)	tracking_number integer
1	1	1	1	iLoveApple	1000 Commenwealth ...	0
2	3	2	2	iHateJobs	27 West 4th Street, Ne...	0
3	4	3	2	iHateJobs	27 West 4th Street, Ne...	0

Figure 2.9. Populate table 9 - Package

PART III: Determine the History Tables

In this iteration, I need to add the history tables for different use cases. I will determine what kind of history tables the project needs, reviewing the use cases.

Aspect 1. New Product Use Case

This occurs when a seller plans to sell a product it has not sold before.
1. The seller searches Amazon's product list to determine if another seller is already selling the product.
2. If a different seller is already selling the product, a new listing is not required; the seller re-uses the same listing.
3. If the product is not yet sold on Amazon, a new listing is created with the product's name, description, price, and other relevant items. Every product added is linked to a product category (all categories are predefined by Amazon), for example, "Computers", "Electronics", "Appliances", and similar.

History Table Name	Function	Attributes
updateSeller_history	Keep track of updates of seller, selling the same product	p_id FK, – product id old_s_id, –old seller id new_s_id, –new seller id updateSeller_time DATE

Condition: if s_id changes in product

Aspect 2. Product Delivery Use Case

This occurs when a seller sends one or more units of a product to Amazon so that they can be sold.
1. The seller ships one or more units of a product to Amazon's warehouse, along with information that indicates to Amazon what the product is, how many units there are, and the condition (new, used, etc …).
2. After Amazon receives the product(s), it updates the seller's inventory so that customers can purchase the product.

History Table Name	Function	Attributes
updateProduct_history	Keep track of updates of new incoming products, especially on its counts	p_id FK, old_counts INT, New_counts INT, updateProduct_time DATE

Condition: if p_counts change in product

Aspect 3. New Customer Account Use Case

This occurs when a customer signs up for an account on Amazon, so they can begin purchasing products.
1. The customer provides Amazon with basic information including a username, an address, phone number, and an email address.
2. Amazon creates an account for the customer, enabling the customer to purchase products. Page 3 of 9

N/A - because we don't need any information on the customer list's updates.

Aspect 4. Product Purchase Use Case

This occurs when a customer purchases a product from Amazon that was provided by a seller.
1. The user logs in to Amazon under their account.
2. A customer selects one or more products on Amazon's website. When selecting a product, the customer is actually selecting a particular seller's inventory while doing so, though they might not realize this because the process is seamless on Amazon's website.
3. The customer selects a shipping speed (super saver shipping, standard shipping, two-day, one-day) and finalizes their choices.
4. Amazon decrements the seller's inventory for the products purchased.
5. Amazon creates an order which tracks which customer purchased which products from which sellers.

History Table Name	Function	Attributes
order_history	Keep track of customers' order when customer purchases an item. Triggered when insert into Order.	Order_id FK, c_username, Order_date DATE

Condition: inserts into order, or any update

Aspect 5. Product Shipment Use Case

This occurs when Amazon ships the products a customer purchased.
1. Amazon packages up the purchased products, and assigns an identifier to package so that it can be tracked.
2. Amazon links the package to the customer's order.
3. Amazon ships the package to the default address linked to the customer's account.
4. Amazon notifies the customer that it has been shipped and provides the customer with the tracking ID.

History Table Name	Function	Attributes
updatePackage_history	Keep track of a package's shipping, and triggered when the package is updated, especially when its tracking ID updates	package_id FK, Old_tracking_id, New_tracking_id, updatePackage_time DATE

Condition: if tracking_id changes in package

PART IV: Implementing And Testing the History Tables

4.1 updateSeller_history

Condition: if s_id changes in product

Query Query History

```
CREATE TABLE updateSeller_history(
p_id DECIMAL(12,0) NOT NULL,
old_s_id DECIMAL(12,0),
new_s_id DECIMAL(12,0),
updateSeller_date DATE,
FOREIGN KEY (p_id) REFERENCES product(p_id)
);

CREATE FUNCTION updateSeller_func()
RETURNS TRIGGER LANGUAGE plpgsql
AS $$
BEGIN
    IF OLD.s_id != NEW.s_id THEN
        INSERT INTO updateSeller_history(p_id, old_s_id, new_s_id, updateSeller_date)
        VALUES(NEW.p_id, OLD.s_id, NEW.s_id, CURRENT_DATE);
    END IF;
    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER updateSeller_trg
BEFORE UPDATE on product
FOR EACH ROW
EXECUTE PROCEDURE updateSeller_func();
```

Data output Messages Notifications

```
CREATE TRIGGER

Query returned successfully in 2 min 48 secs.
```

Figure 4.1.1 History Table - updateSeller_history Implementation

Query Query History

```sql
-- Testing for updateSeller_history
--     We have a new seller "Technology Era" selling iPhone13,
--     instead of "World changing Technologies", so product
--     "Refurbished iPhone 13" will have a new s_id

INSERT INTO seller
VALUES(nextval('seller_seq'), 'Technology Era');

UPDATE product
SET s_id = (SELECT s_id FROM seller WHERE s_name = 'Technology Era')
WHERE p_name = 'Refurbished iPhone 13';

SELECT product.p_name, seller.s_name, product.p_id, old_s_id, new_s_id from updateSeller_history
INNER JOIN seller ON updateSeller_history.new_s_id = seller.s_id
INNER JOIN product ON updateSeller_history.p_id = product.p_id;
```

Data output Messages Notifications

	p_name character varying (64)	s_name character varying (32)	p_id numeric (12)	old_s_id numeric (12)	new_s_id numeric (12)
1	Refurbished iPhone 13	Technology Era	3	2	5

Figure 4.1.2 History Table - updateSeller_history Testing

4.2 updateProduct_history

Condition: if p_counts change in product

Query Query History

```sql
DROP TABLE updateProduct_history;

CREATE TABLE updateProduct_history(
p_id DECIMAL(12,0) NOT NULL,
old_p_counts INT,
new_p_counts INT,
updateProduct_date DATE,
FOREIGN KEY (p_id) REFERENCES product(p_id)
);

CREATE OR REPLACE FUNCTION updateProduct_func()
RETURNS TRIGGER LANGUAGE plpgsql
AS $$
BEGIN
    IF OLD.p_counts != NEW.p_counts THEN
        INSERT INTO updateSeller_history(p_id, old_p_counts, new_p_counts, updateProduct_date)
        VALUES(NEW.p_id, OLD.p_counts, NEW.p_counts, CURRENT_DATE);
    END IF;
    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER updateProduct_trg
BEFORE UPDATE on product
FOR EACH ROW
EXECUTE PROCEDURE updateProduct_func();
```

Data output Messages Notifications

```
CREATE TRIGGER

Query returned successfully in 53 msec.
```

Figure 4.2.1 History Table - updateProduct_history Implementation

Query Query History

```sql
-- Testing for updateProduct_history
--      we have one more iphone 13 into the storange.

UPDATE product
SET p_counts = 2
WHERE p_name = 'Refurbished iPhone 13';

SELECT * from updateProduct_history;
```

Data output Messages Notifications

	p_id numeric (12)	old_p_counts integer	new_p_counts integer	updateproduct_date date
1	3	0	2	2022-11-09

Figure 4.2.1 History Table - updateProduct_history Testing

4.3 order_history

Condition: inserts into order, or any update

z

Query Query History

```sql
CREATE TABLE order_history(
order_id DECIMAL(12,0),
c_username VARCHAR(32),
order_date DATE,
FOREIGN KEY (order_id) REFERENCES orders(order_id)
);

CREATE OR REPLACE FUNCTION order_history_func()
RETURNS TRIGGER LANGUAGE plpgsql
AS $$
DECLARE
_username VARCHAR(32);
_c_p_id DECIMAL(12,0);
BEGIN

SELECT c_p_id
INTO _c_p_id
FROM orders;

SELECT customer_product_bridge.c_username
INTO _username
FROM customer_product_bridge
WHERE customer_product_bridge.c_p_id = _c_p_id;

INSERT INTO order_history(order_id, c_username, order_date)
VALUES(NEW.order_id, _username, CURRENT_DATE);
RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER order_history_trg
BEFORE UPDATE on orders
FOR EACH ROW
EXECUTE PROCEDURE order_history_func();
```

Figure 4.3.1 History Table - order_history Implementation

Query Query History

```
UPDATE orders
SET c_p_id = 1
WHERE order_id = 1;

SELECT * from order_history;
```

Data output Messages Notifications

	order_id numeric (12)	c_username character varying (32)	order_date date
1	1	iLoveApple	2022-11-09

Figure 4.2.2 History Table - order_history Implementation

4.4 updatePackage_history

Condition: if tracking_id changes in package

Query Query History

```sql
DROP TABLE updatepackage_history;

CREATE TABLE updatePackage_history(
package_id DECIMAL(12,0),
old_tracking_num INT,
new_tracking_num INT,
updatePackage_date DATE,
FOREIGN KEY (package_id) REFERENCES package(package_id)
);

CREATE OR REPLACE FUNCTION updatePackage_func()
RETURNS TRIGGER LANGUAGE plpgsql
AS $$
BEGIN
    IF OLD.tracking_number != NEW.tracking_number THEN
        INSERT INTO updatePackage_history(package_id, old_tracking_num, new_tracking_num, updatePackage_date)
        VALUES(NEW.package_id, OLD.tracking_number, NEW.tracking_number, CURRENT_DATE);
    END IF;
    RETURN NEW;
END;
$$;

CREATE OR REPLACE TRIGGER updatePackage_trg
BEFORE UPDATE on package
FOR EACH ROW
EXECUTE PROCEDURE updatePackage_func();
```

Data output Messages Notifications

```
CREATE TRIGGER

Query returned successfully in 1 secs 760 msec.
```

Figure 4.4.1 History Table - updatePackage_history Implementation

Query Query History

```
-- Testing for updatePackage_history

UPDATE package
SET tracking_number = 99823950
WHERE package_id = 1;

SELECT * FROM updatePackage_history;
```

Loading...

Data output Messages Notifications

	package_id numeric (12)	old_tracking_num integer	new_tracking_num integer	updatepackage_date date
1	1	0	99823950	2022-11-09

Figure 4.4.2 History Table - updatePackage_history Testing

PART V: Determining the Procedures

In this iteration, I need to add the procedures for different use cases. I will determine what kind of procedure the project needs, reviewing the use cases.

Aspect 1. New Product Use Case

This occurs when a seller plans to sell a product it has not sold before.
1. The seller searches Amazon's product list to determine if another seller is already selling the product.
2. If a different seller is already selling the product, a new listing is not required; the seller re-uses the same listing.
3. If the product is not yet sold on Amazon, a new listing is created with the product's name, description, price, and other relevant items. Every product added is linked to a product category (all categories are predefined by Amazon), for example, "Computers", "Electronics", "Appliances", and similar.

Procedure	**Function**	**Condition**
newSeller (name VARCHAR(32)	Add new seller	If same s_name doesn't exist in Seller

Procedure	**Function**	**Condition**
newProduct (seller_name, Product_name, Product_counts, Product_price, product_category)	Add new Product, If product_name exists, simply update the seller_name; If product_name does not exist, Update the product listings.	N/A

Aspect 2. Product Delivery Use Case

This occurs when a seller sends one or more units of a product to Amazon so that they can be sold.
1. The seller ships one or more units of a product to Amazon's warehouse, along with information that indicates to Amazon what the product is, how many units there are, and the condition (new, used, etc …).
2. After Amazon receives the product(s), it updates the seller's inventory so that customers can purchase the product.

Procedure	Function	Condition
deliverProduct(Product_name, product_condition New_arriving_units INT)	Update product's counts	If product_name does exist in Product

Aspect 3. New Customer Account Use Case

This occurs when a customer signs up for an account on Amazon, so they can begin purchasing products.
1. The customer provides Amazon with basic information including a username, an address, phone number, and an email address.
2. Amazon creates an account for the customer, enabling the customer to purchase products.

Procedure	Function	Condition
newCustomer(Customer_name, Customer_address, Customer_phone, customer_email)	Add new customer	If customer_name does not currently exist

Aspect 4. Product Purchase Use Case

This occurs when a customer purchases a product from Amazon that was provided by a seller.
1. The user logs in to Amazon under their account.
2. A customer selects one or more products on Amazon's website. When selecting a product, the customer is actually selecting a particular seller's inventory while doing so, though they might not realize this because the process is seamless on Amazon's website.
3. The customer selects a shipping speed (super saver shipping, standard shipping, two-day, one-day) and finalizes their choices.
4. Amazon decrements the seller's inventory for the products purchased.
5. Amazon creates an order which tracks which customer purchased which products from which sellers.

Procedure	Function	Condition
buyProduct(	Descrement counts in	If Product_name and username are

Product_name, Counts, Customer_Username, shipping_speed)	Product; Add new Shipping	both valid

Aspect 5. Product Shipment Use Case

This occurs when Amazon ships the products a customer purchased.

1. Amazon packages up the purchased products, and assigns an identifier to package so that it can be tracked.
2. Amazon links the package to the customer's order.
3. Amazon ships the package to the default address linked to the customer's account.
4. Amazon notifies the customer that it has been shipped and provides the customer with the tracking ID.

Procedure	**Function**	**Condition**
newPackages(Customer_username, Shipping_id, tracking_number	When shipped, a new Package created	Trigger: When a new Shipping is created

PART VI: Implementing And Testing The Stored Procedures

5.1 addSeller() & newProduct()

Query Query History

```
CREATE OR REPLACE PROCEDURE newSeller(
    _name IN VARCHAR)
    LANGUAGE plpgsql
AS
$reusableproc$
    BEGIN
        INSERT INTO seller (s_id, s_name)
        VALUES (nextval('seller_seq'), _name);
    END;
$reusableproc$
```

Data output Messages Notifications

```
CREATE PROCEDURE

Query returned successfully in 81 msec.
```

Figure 5.1.1 Procedure - newSeller() implementation

Query Query History

```
CALL newSeller('Home of Furniture');
SELECT * from seller;
```

Data output Messages Notifications

	s_id [PK] numeric (12)	s_name character varying (32)
1	1	Furniture Home
2	2	World-changing Tech...
3	5	Technology Era
4	8	Home of Furniture

Figure 5.1.2 Procedure - newSeller() Testing

Query Query History

```
CREATE OR REPLACE PROCEDURE newProduct(
    seller_name IN VARCHAR,
    product_name IN VARCHAR,
    product_counts IN INT,
    product_price IN DECIMAL,
    product_category IN VARCHAR)
    LANGUAGE plpgsql
AS
$reusableproc$
BEGIN
    IF EXISTS(
    SELECT p_name FROM product WHERE p_name = product_name) THEN

    UPDATE product
    SET p_name = product_name, s_id = (SELECT s_id FROM seller WHERE s_name = seller_name)
    WHERE p_name = product_name;

        ELSE
            INSERT INTO product (p_id, p_name, p_des, p_counts, p_price, p_category, s_id)
            VALUES (nextval('product_seq'), product_name, 'N/A', product_counts, product_price, product_category, (SELECT s_id FROM seller where s_name = seller_name));
        END IF;
    END;
    $reusableproc$
```

Data output Messages Notifications

```
CREATE PROCEDURE

Query returned successfully in 49 msec.
```

Figure 5.1.3 Procedure - newProduct() implementation

Query Query History

```
-- newProduct
--     update the seller. old Sofa has 'Furniture Home' as seller, now updates to 'Home of Furniture'

CALL newProduct('Home of Furniture', 'Comfortable Sofa', 10, 599.00, 'Furniture');

SELECT p_id, p_name, p_counts, p_price, p_category, seller.s_name from Product
JOIN seller ON seller.s_id = product.s_id;
```

Data output Messages Notifications

	p_id numeric (12)	p_name character varying (64)	p_counts integer	p_price numeric (12,2)	p_category character varying (12)	s_name character varying (32)
1	1	Wood L-shape Table	3	299.00	Furniture	Furniture Home
2	3	Refurbished iPhone 13	2	899.00	Electronics	Technology Era
3	2	Comfortable Sofa	11	599.00	Furniture	Home of Furniture
4	4	Big Iron Chair	5	100.00	Furniture	Home of Furniture

Query Query History

```
CALL newProduct('Home of Furniture', 'Big Iron Chair', 5, 100.00, 'Furniture');
SELECT * from product;
```

Data output Messages Notifications

	p_id [PK] numeric (12)	p_name character varying (64)	p_des character varying (255)	p_counts integer	p_price numeric (12,2)	p_category character varying (12)	s_id numeric (12)
1	1	Wood L-shape Table	Wood table with a L-sh...	3	299.00	Furniture	1
2	3	Refurbished iPhone 13	Apple iPhone 13	2	899.00	Electronics	5
3	2	Comfortable Sofa	The sofa is made of lin...	1	599.00	Furniture	1
4	4	Big Iron Chair	N/A	5	100.00	Furniture	8

Figure 5.1.4 Procedure - newProduct() Testing 1 & 2

5.2 deliverProduct()

Query Query History

```
-- seller 'Home of Furniture' delivers 10 'Comfortable Sofa'
-- to Amazon. A new row in Seller_delivery table is created.

CREATE OR REPLACE PROCEDURE deliverProduct(
    product_name IN VARCHAR,
    product_condition IN VAR Loading...
    arriving_units IN INT)
    LANGUAGE plpgsql
AS
$reusableproc$
BEGIN
    -- first, create a row in table Seller_delivery
    INSERT INTO seller_delivery
    VALUES(nextval('seller_delivery_seq'), arriving_units, product_condition, (SELECT p_id FROM product WHERE p_name = product_name));

    -- second, update product counts
    UPDATE product
    SET p_counts = p_counts + arriving_units
    WHERE p_name = product_name;

    END;
    $reusableproc$
```

Data output Messages Notifications

```
CREATE PROCEDURE

Query returned successfully in 1 secs 309 msec.
```

Figure 5.2.1 Procedure - deliverProduct() implementation

Query Query History

```
-- seller 'Home of Furniture' delivers 10 'Comfortable Sofa'
-- to Amazon. A new row in Seller_delivery table is created.

-- test deliverProduct
CALL deliverProduct('Comfortable Sofa', 'New', 10);

-- test is new row is added
SELECT delivery_id, units, condition, product.p_name from Seller_delivery
JOIN product on product.p_id = seller_delivery.p_id;
```

Data output Messages Notifications

	delivery_id numeric (12)	units integer	condition character varying (32)	p_name character varying (64)
1	1	3	New	Wood L-shape Table
2	2	1	New	Comfortable Sofa
3	3	1	Like New	Refurbished iPhone 13
4	6	10	New	Comfortable Sofa

Query Query History

```
-- seller 'Home of Furniture' delivers 10 'Comfortable Sofa'
-- to Amazon. A new row in Seller_delivery table is created.

-- test if product_counts are updates:
--             1 -> 1 + 10
SELECT * from product;
```

Data output Messages Notifications

	p_id [PK] numeric (12)	p_name character varying (64)	p_des character varying (255)	p_counts integer	p_price numeric (12,2)	p_category character varying (12)	s_id numeric (12)
1	1	Wood L-shape Table	Wood table with a L-shaped that can fit in a corner in the ro...	3	299.00	Furniture	1
2	3	Refurbished iPhone 13	Apple iPhone 13	2	899.00	Electronics	5
3	4	Big Iron Chair	N/A	5	100.00	Furniture	8
4	2	Comfortable Sofa	The sofa is made of linen fabric with hardwood frame.	11	599.00	Furniture	8

Figure 5.2.2 Procedure - deliverProduct() Testing 1 & 2

5.3 newCustomer()

Query Query History

```
CREATE OR REPLACE PROCEDURE newCustomer(
    customer_username IN VARCHAR,
    customer_address IN VARCHAR,
    customer_phone IN VARCHAR,
    customer_email IN VARCHAR)
    LANGUAGE plpgsql
AS
$reusableproc$
BEGIN
    -- if username is taken, give error message
    IF EXISTS(
    SELECT c_username FROM customer WHERE c_username = customer_username) THEN
    RAISE EXCEPTION USING MESSAGE = 'username already exists';

    ELSE
        INSERT INTO customer(c_username, c_address, c_phone, c_email)
        VALUES(customer_username, customer_address, customer_phone, customer_email);
    END IF;
END;
    $reusableproc$
```

Data output Messages Notifications

```
CREATE PROCEDURE

Query returned successfully in 7 secs 458 msec.
```

Figure 5.3.1 Procedure - newCustomer() implementation

Query Query History

```
-- test 2
CALL newCustomer('Aliceee', '1099 Commonwealth Ave, Boston', '5087236524', 'alice99@gmail.com');

SELECT * from customer;

```

Data output Messages Notifications

	c_username [PK] character varying (32)	c_address character varying (255)	c_phone character varying (15)	c_email character varying (255)
1	iLoveApple	1000 Commenwealth ...	5082947592	applesheaven@google....
2	iHateJobs	27 West 4th Street, Ne...	2539683659	privateaccount@googl...
3	Aliceee	1099 Commonwealth ...	5087236524	alice99@gmail.com

Query Query History

```


-- test 1: username already exists
CALL newCustomer('iLoveApple', '1099 Commonwealth Ave, Boston', '5087236524', 'apple999@gmail.com');

```

Data output Messages Notifications

```
ERROR:  username already exists
CONTEXT:  PL/pgSQL function newcustomer(character varying,character varying,character varying,character varying) line 6 at RAISE
SQL state: P0001
```

Figure 5.3.2 Procedure - newCustomer() Testing

5.4 buyProduct()

Query Query History

```
CREATE OR REPLACE PROCEDURE buyProduct(
    product_name IN VARCHAR,
    product_counts IN INT,
    customer_username IN VARCHAR,
    shipping_speed IN VARCHAR)
    LANGUAGE plpgsql
AS
$reusableproc$
DECLARE
    _c_p_id DECIMAL;
BEGIN
    -- first, check if product_name is valid

    _c_p_id = nextval('customer_product_bridge_seq');

    IF EXISTS(
    SELECT p_name FROM product WHERE p_name = product_name) THEN

        -- decrement the count
    UPDATE product
    SET p_counts = p_counts - product_counts;

        -- add new customer_product_bridge
    INSERT INTO customer_product_bridge(c_p_id, c_username, p_id, s_id)
    VALUES(_c_p_id, customer_username, (SELECT p_id FROM product WHERE p_name = product_name),
           (SELECT s_id FROM product WHERE p_name = product_name));

        -- add new shipping
    INSERT INTO shipping(shipping_id, shipping_speed, c_p_id)
    VALUES (nextval('shipping_seq'), shipping_speed, _c_p_id);

    END IF;
END;
$reusableproc$
```

Data output Messages Notifications

```
CREATE PROCEDURE

Query returned successfully in 44 msec.
```

Total rows: 3 of 3 Query complete 00:00:00.044

Figure 5.4.1 Procedure - buyProduct() implementation

Query Query History

```sql
-- test buyProduct
--    Aliceee buys 2 Comfortable Sofa, with shipping_spped = Same Day Delivery

CALL buyProduct('Comfortable Sofa', 2, 'Aliceee', 'Same Day Delivery');

-- test Customer_product_bridge and Shipping
SELECT c_p_id, c_username, product.p_name, customer_product_bridge.s_id from customer_product_bridge
JOIN product on product.p_id = customer_product_bridge.p_id;
```

Data output Messages Notifications

	c_p_id numeric (12)	c_username character varying (32)	p_name character varying (64)	s_id numeric (12)
1	1	iLoveApple	Refurbished iPhone 13	2
2	2	iHateJobs	Wood L-shape Table	1
3	3	iHateJobs	Comfortable Sofa	1
4	5	Aliceee	Comfortable Sofa	8

Query Query History

```sql
-- test buyProduct
--    Aliceee buys 2 Comfortable Sofa, with shipping_spped = Same Day Delivery

-- test Customer_product_bridge and Shipping
SELECT * from shipping;
```

Data output Messages Notifications

	shipping_id [PK] numeric (12)	shipping_speed character varying (32)	c_p_id numeric (12)
1	1	2 Day premium	1
2	2	2 day premium	2
3	3	Same Day	3
4	5	Same Day Delivery	5

Figure 5.4.2 Procedure - buyProduct() Testing 1 & 2

5.5 newPackage()

Query Query History

```
-- new package is triggered each time a new shipping is created
CREATE OR REPLACE PROCEDURE newPackages(
    customer_username in VARCHAR,
    _c_p_id in DECIMAL,
    tracking_number in INT)
    LANGUAGE plpgsql
AS
$$
DECLARE
    _order_id DECIMAL;
BEGIN

    _order_id = nextval('order_seq');

        -- add new order
    INSERT INTO orders(order_id, shipping_id, c_p_id)
    VALUES (_order_id, (SELECT shipping_id FROM shipping WHERE c_p_id = _c_p_id),
            _c_p_id);

        -- add new shipping
    INSERT INTO package(package_id, order_id, shipping_id, c_username, c_address, tracking_number)
    VALUES(nextval('package_seq'), _order_id, (SELECT shipping_id FROM shipping WHERE c_p_id = _c_p_id)
           , customer_username,
           (SELECT c_address FROM customer WHERE c_username = customer_username),
           tracking_number);
END;
$$;
```

Loading...

Data output Messages Notifications

```
CREATE PROCEDURE

Query returned successfully in 834 msec.
```

Figure 5.5.1 Procedure - newPackage() implementation

Query Query History

```
-- Aliceee buys 2 sofa, now we add the package

CALL newPackage('Aliceee', 5 ,0);  -- default tracking number is 0

SELECT package.c_username,package.c_address,package.tracking_number, orders.c_p_id, shipping.shipping_speed, product.p_name from package
JOIN orders on orders.order_id = package.order_id
JOIN Shipping on orders.shipping_id = shipping.shipping_id
JOIN customer_product_bridge on customer_product_bridge.c_p_id = shipping.c_p_id
JOIN product on product.p_id = customer_product_bridge.p_id;
```

Loading...

Data output Messages Notifications

	c_username character varying (32)	c_address character varying (255)	tracking_number integer	c_p_id numeric (12)	shipping_speed character varying (32)	p_name character varying (64)
1	iHateJobs	27 West 4th Street, Ne...	0	2	2 day premium	Wood L-shape Table
2	iHateJobs	27 West 4th Street, Ne...	0	3	Same Day	Comfortable Sofa
3	iLoveApple	1000 Commenwealth ...	99823950	1	2 Day premium	Refurbished iPhone 13
4	Aliceee	1099 Commonwealth ...	0	5	Same Day Delivery	Comfortable Sofa

Figure 5.5.2 Procedure - newPackage() Testing