

Name: Xiaonan Liu

Table of Contents

Project Direction Overview	2
Use Cases and Fields	3
Structural Database Rules	7
Conceptual Entity-Relationship Diagram	8
Full DBMS Physical ERD	9
Stored Procedure Execution and Explanations	13
Question Identification and Explanations	27
Query Executions and Explanations	29
Index Identification and Creations	34
History Table Demonstration	36
Data Visualizations	38
Summary and Reflection	40

Project Direction Overview

Update, if necessary, the overview that describes who the database will be for, what kind of data it will contain, how you envision it will be used, and most importantly, why you are interested in it.

Functionality:

The database will be used to store and organize information about the wholesale store's products, and customers. Here is an overview of how the database will be utilized:

1. Employee management: the database will store the employee's information, and give them different rights to operate in this database, in this case, the data will be changed timely and better serve the customer, and be convenient for the management for the owner.
2. Product Information: The database will store detailed information about various drinks, snacks, instant noodles, etc. This includes product names, descriptions, pricing, quantity, category, and any associated discounts or promotions.
3. Customer management: The database will enable to store of the customer's information including their name, phone, customer type, and the orders which they have purchased.
4. Order records: when an order has been placed, it will be stored in the database, to track the sales of all products, this will help the management and make future business data-driven decisions.

Motivation:

The reason why I want to design this database is my mom runs a wholesale store, in their market, there are almost hundreds of people who sell different kinds of stuff and different brands. Not like Costco, or the first-level wholesale owner (In China, there are different levels of wholesale store owners, the first-level is the biggest one, they need to sell millions of products annually to achieve this level), the second-level wholesale store owner like my mom, they don't have enough money to hire professional people to manage the business for them or use business-level databases to track their business. Even though their sale volume is massive, the profit they made are really limited. Most of them still use very traditional ways to track their data, or a basic package for software that you need to pay for that. That takes a long time daily for her to take stock of what's going on in the store, and errors happen often. So, I want to develop a database that will save their time and reduce manual errors. Furthermore, the database will simplify the rebate calculation process, ensuring accuracy and transparency.

Use Cases and Fields

Update, if necessary, the five or more use cases that enumerate steps of how the database will be typically used, and also identify significant database fields needed to support each use case.

- **Signup/installation use case**

1. The person should visit the website and download this application, then install it.
2. The application asks them to create an account to set up as admin.
3. Users need to create more accounts as team members to use this application to manage this store
4. An employee gets a promotion, the wage will be raised, and the user needs to update the salary for her.
5. An employee did not come to the store without asking for leave, other people need to call him/her to check his/her situation, and they could search for his/her phone number.

Employee table

Field	What it stores	Why it's needed
employee_id	The identity number of the employee	The primary key for the employee table should be unique to identify the employee
First_name	The first name of the employee	Display the first name, to save the employee's info
last_name	Last name of the employee	Display the last name, to save the employee's info
title	The title of the employee	The different titles have different permissions
phone	The phone number of the employee	Convenient communication
email	The email of the employee	Send emails and help communication
join_date	The date when the employee join	The wage and bonus should adjust annually
leave_date	The date when this employee leave	When the employee leaves this position, they will not allow to log in this system
salary	The salary for this employee	Record the salary that this employee received

- **Product management**

1. The store buys a new product, and the employee needs to record the information in this database and set the category for this product.
2. A customer wants to buy 5 different drinks, the employee checks the drink category to see how many choices they could offer.
3. The owner wants to check the total sales volume for a brand.
4. The original supplier raises the price, and the owner found a new supplier which offers a better price, the employee needs to update the new supplier info.
5. A customer needs to buy 500 boxes of chips, the employee needs to check the quantity of the chips.

6. The new year is coming, there is a discount for specified products, and the employee needs to add reduced prices for specified brands.

We need to add a goods table

Goods Table

Field	What it stores	Why it's needed
goods_id	The identify number for the goods	Use as the primary key for the goods table
goods_name	The name of this goods	Convenient management and display the name
brand_id	Id for brand	Use as foreign key for search brand
Supplier_id	Id for supplier	Use as foreign key for search supplier
Product_id	Id for product	Use as foreign key for search product
Order_id	Id for order	Use as foreign key for search order
cost	The cost of the product	Record the cost and calculate the profit
Retail_price	The retail price for the product	Record the retail price of this product to help calculate the profit
Buying_date	The date for buying this good	Track the date
quantity	The quantity of this good	Track the quantity

Product table

Field	What it stores	Why it's needed
Product_id	The identify number for the product	Use as the primary key for the product table
Product_name	The name of this product	Convenient management and display the name
category	The category of this product	For well organized, and convenient management

- **Customer management**

1. A new customer makes a purchase in this store and wants to join as a member, the employee needs to record her/his information.
2. The customer's total shopping amount for this year is over \$10000, his/her type will change to gold.
3. A customer wants to make a return, but he/she did not have the receipt, the employee needs to search his/her order history to make sure he/she purchases this product in this store.
4. The end of the year is coming, the store needs to send different gift cards based on the total purchase amount of this year for this customer, so we need to check their total amount for this year.

5. A customer changed his/her address, the employee needs to update his/her address info.
6. The new rebate rate will be decided for this year, the employee needs to set a new rate for customers.

Customer table

Field	What it stores	Why it's needed
Customer_id	The identity number for customer	Use as primary key for customer table
First_name	The first name of customer	Display the first name, to help the communication and address the unexcepted problem
Last_name	The last name of customer	Display the last name, to help the communication and address the unexcepted problem
phone	The phone number of the customer	Record the info and help the communications
email	The email of the customer	Send receipt, emails, and help communication
address	The address of the customer	Record the info and will send gift base on the purchase in the store
amount	Record the total purchase amount for this year	For analysis and calculate the rebate
Is_silver	is customer a silver customer	For calculate rebate
Is_gold	is customer a gold customer	For calculate rebate
Is_diamond	is customer a diamond customer	For calculate rebate

- **Order record**

1. A customer makes a purchase, the employee needs to record all the products he/she bought and calculate the total amount.
2. The customer wants to check the last purchase he/she made in this store, and he/she asked the employee to resent the receipt by email.
3. The customer found there was an item been calculated twice, but he/she only bought one, he/she ask the staff to correct this mistake, and the staff needs to update the order info.
4. Allow employees to update the order information, for example, some products are calculated twice but the customer only buys one.
5. The store needs to take stock of sales and merchandise every day, so they need to tally up the data based on today's order information
6. The coco cola Supplier has a special discount for the summer sale of over 100000, the store needs to check how many coco-cola it sold for the summer.
7. After the customer placed the order, print the receipt, and give it to the customer, in case they need to check the information.

Order table

Field	What it stores	Why it's needed
Order_id	The identity number for this order	Use as primary key for order table, should be unique to help search
Customer_id	Id of the customer who make this purchase	Record the customer's purchase
Order_date	The date when this order is placed	To record the time of this order
amount	The total amount for this order	Help customer to check the information and convenient the store's management
discount	The discount available for this order	To help clearly see the discount
reduced_price	The actual price for this order	To check the final price

Structural Database Rules

Update, if necessary, your list of your structural database rules, along with supporting explanations. Note that at least 10 entities should be identified in your structural database rules, and your specialization-generalization rule should have a supertype and at least two subtypes.

Structural database rules

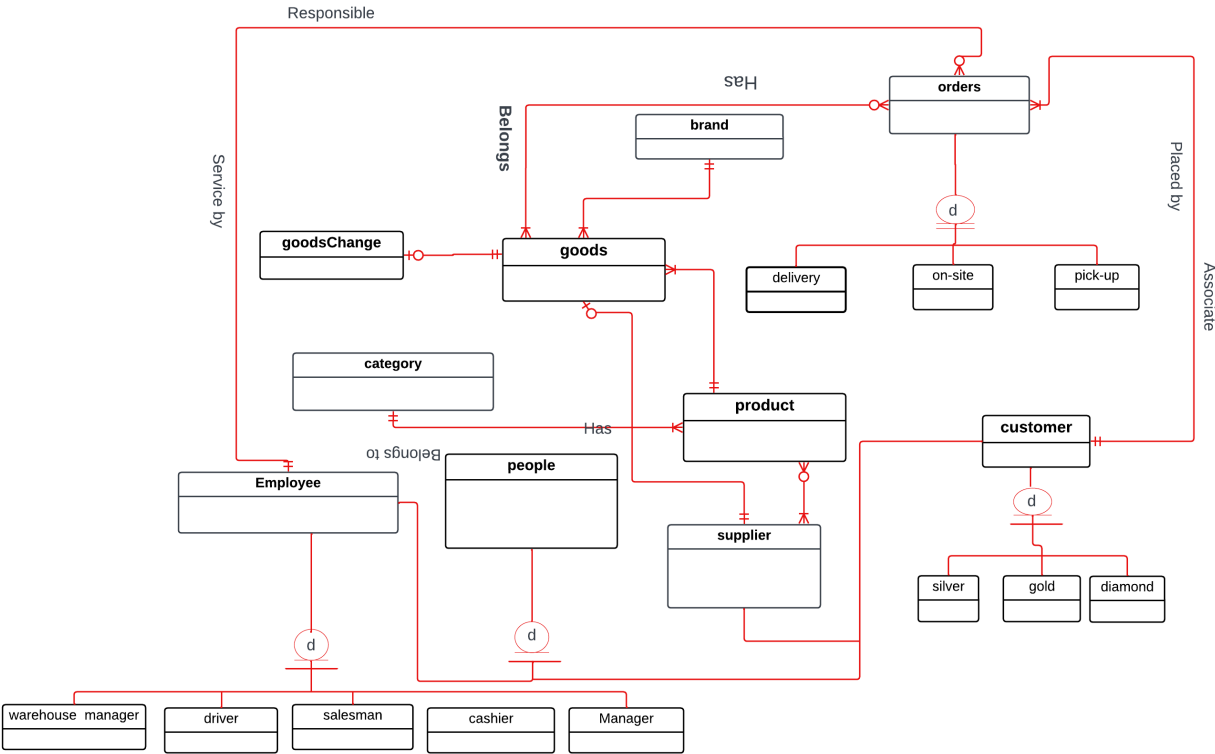
1. A brand could have more than one product, and a product could also be from different brands.
2. A brand could have different suppliers, also suppliers could not only supply one brand.
3. A product should belong to a certain category, but this category could have different products.
4. A product could be purchased or not purchased by a customer, and the customer could also choose to buy or not buy this product.
5. A product could be purchased or not in order, and an order must contain one or more products.
6. An order could have one or more products and a product could be associated with zero or different orders.
7. Customers could place one or more orders, but orders should associate with a certain customer.
8. A store could have one or more employees, and those employees should belong to this store.
9. A store could have one or more suppliers, and this supplier only supplies or does not supply this store.
10. A store could have one or more brands, and this brand could sell or not sell in this store
11. One good could have zero or more changes, but a change should only associate with one specific good.

specialization-generalization rules:

1. A customer could only be a silver customer, a gold customer, or a diamond customer.
2. An employee can only be a salesman, a cashier, a warehouse manager, a driver, or a manager.
3. An order could be a delivery order or pick-up order or an on-site shopping order.
4. A person could only be a customer, an employee, or a supplier.

Conceptual Entity-Relationship Diagram

Update, if necessary, your conceptual ERD that visualizes the structural database rules, along with supporting explanations. Note that at least 10 entities are required in your conceptual ERD, and that your specialization-generalization relationship should have a supertype and at least two subtypes.



Full DBMS Physical ERD

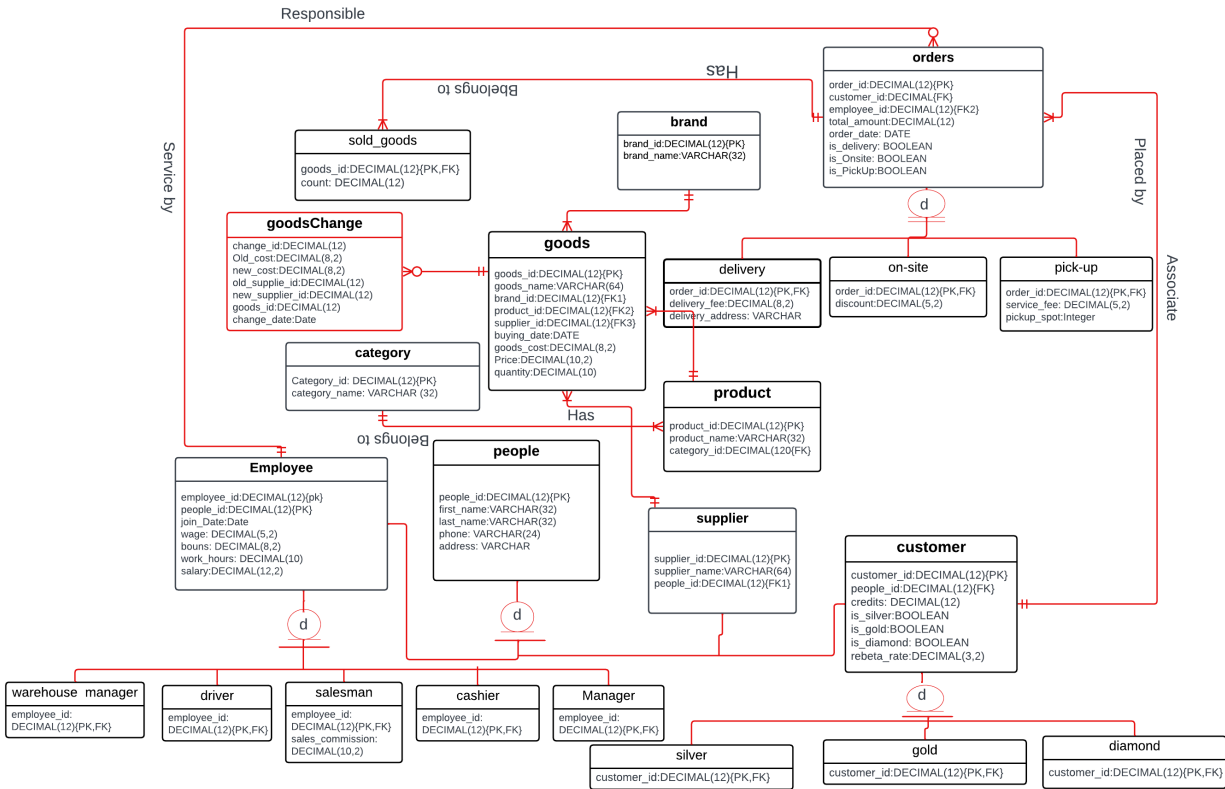
Update your normalized DBMS physical ERD to include the new history table.

Table	Attribute	Datatype	Reasoning	Example data
people	First_name	VARCHAR(32)	Every person has first name which could use for checking their info, I allow for 32 characters, it should suit for most names	Selina
	Last_name	VARCHAR(32)	Every person has first name which could use for checking their info, I allow for 32 characters, it should suit for most names	Liu
	phone	VARCHAR(24)	Every person should have a phone name which uses for contact, the phone number is usually 10 numbers which occupied 20 chars, and 2 '-' to make phone number to read easier	617-123-1234
	address	VARCHAR(128)	Record the address for this person, 128 chars should suit for address	1040 Commonwealth Ave, Brookline, 02144
employee	Join_Date	DATE	When these employees join this company, to calculate their salary, they will get a rise annually	2022-05-01
	title	VARCHAR(64)	Record their title for easily knowing their title for work, the title is up to 64 characters	manager
	wage	DECIMAL(5,2)	The wage for this employee, because the wage is increased based on the experiences, so it may become float, and wage should always below 999	56.22
	Work_hours	DECIMAL(10)	How many hours this employee worked for this company, record to calculate the salary	200

	salary	DECIMAL(12)	The salary for this employee, it always equals to wage * work_hours, up to 12 digits, and rounded to the nearest integer	5234
Customer	credits	DECIMAL(12)	How many credits customers get in this store, is most related to the how much this customer spend annually in this store, it is up to 12 digits	200,000
	Is_silver	BOOLEAN	Is this customer a silver customer, it should only has value for true or false	true
	Is_gold	BOOLEAN	Is this customer a gold customer, it should only has value for true or false	false
	Is_diamond	BOOLEAN	Is this customer a diamond customer, it should only have value for true or false	false
	Debate_rate	Decimal(3,2)	The debate rate for this customer, it should below 1, usually should be 0.02-0.06	0.02
category	Category_name	VARCHAR(32)	The name of this category, up to 32 chars, it is enough for the category name	drinks
product	Product_name	VARCHAR(32)	The name for this product, up to 32 chars, it should be enough for identify a product	Purified water
brand	Brand_name	VARCHAR(32)	The name for this product, up to 32 chars, it should be enough for identify a brand	Dasani
supplier	Supplier_name	VARCHAR(64)	The name for this supplier, to identify this supplier, up to 64 chars	CG Roxane LLG
goods	Goods_name	VARCHAR(64)	The name for this goods, it is always combined with the brand name and product name, so it should up to 64 chars, it is the sum of the brand_name	Dasani Purified water

			constrains and product_name constrains	
	Goods_cost	DECIMAL(8,2)	The cost for these goods, we need to record the cost to set the price, and calculate our profit, the cost is very likely to be a float, and unlike to over 100,000	1.05
	prices	DECIMAL(10,2)	The prices for these goods, it is very likely to be a float, and unlike to over 1,000,000	24.99
	quantity	INTEGER	The quantity for this kind of goods, the quantity could only be interger	500
Sold_goods	count	INTEGER	How many this kind of goods we have sold for each order, this should always be integer	5
order	Total_amount	DECIMAL(12,2)	The total cost for this order, it could be a float, and unlikely to over 1,000,000,000	299.56
	Order_date	DATE	The date when place this order	2023-08-12
	Is_delivery	BOOLEAN	Does this order needs to be delivered, the value could only be true or false	false
	Is_onsite	BOOLEAN	Record is this order purchase onsite, the value only be true or false	true
	Is_pickUp	BOOLEAN	Is this order a pickup order, the value only be true or false	false
Delivery_order	Delivery_fee	Decimal(5,2)	The delivery fee is the fee for delivery, it could be a float, and should not over 1,000	22.34
	Delivery_address	VARCHAR(255)	The address for delivery, we could search from people table, or add manually	12340 168 th Ave, SE,seattle,98125
Onsite_order	discount	DECIMAL(5,2)	Discount for onsite purchase, should be a float	0.95

Pickup_order	Service_fee	DECIMAL(5,2)	The service fee for pickup order, could be a float	3.05
	Pickup_spot	INTEGER	The available parking spot for customer to pickup their goods	2
Goods_change	Change_id	DECIMAL(12)	This is the primary key for this table, the DECIMAL(12) could hold many values	1
	Goods_id	DECIMAL(12)	The foreign key for this table, references to goods table	2
	Change_date	date	The date when we make change, just use the date, and default is current date	2023-08-13
	Old_goods_cost	DECIMAL(8,3)	The old cost for this goods	6.5
	New_goods_cost	DECIMAL(8,3)	The new cost for this goods	6.3
	Old_supplier_id	DECIMAL(12)	The old supplier id for this goods	3
	New_supplier_id	DECIMAL(12)	The new supplier id for this goods	4



When the relationship is one to many, we should store the primary key of the entity which is the one to the field of the entity which is many, such as the store has supplier's id info, also in the product's field we have the foreign key for the category, in the order's field we have the customer's id as the foreign key.

But for the relationship which is many to many which is a little bit complicated, I have to create a new table named goods to store both brand and product information, the goods are related to a specified brand and a specified product, but a brand could have different goods, and a products could only have different goods, so in goods' field, we should have brand_id and product_id as a foreign key, also this could also store the id for order as a foreign key. Because the relationship between brands is also many to many, so we also use goods to solve this problem, so we store the supplier_id in the goods field and put cost and price and buying date in the goods field.

To minimize data redundancy, I create a people table to store the basic info for a person, a person could be a customer, a supplier, or an employee, so we don't need a name, phone, or address for those tables, we just store that information in people table, and all sub table use people_id as the foreign key to reference people table. Also, all subclass has the same field, we add that into the supertype.

Stored Procedure Execution and Explanations

Update, if necessary, your screenshots and explanations of defining and executing your stored procedures to transactionally add data to your database.

1. A new employee join the company, we record her/his info into our database

Command Line:

```
CREATE OR REPLACE PROCEDURE insert_employee(
    p_first_name VARCHAR(32),
    p_last_name VARCHAR(32),
    p_phone VARCHAR(24),
    p_address VARCHAR(128),
    p_title VARCHAR(64),
    p_wage DECIMAL(5,2),
    p_work_hours DECIMAL(10)
)
LANGUAGE plpgsql
AS $$

DECLARE
    new_people_id DECIMAL(12);
    new_employee_id DECIMAL(12);

BEGIN

    -- Insert into people table
    INSERT INTO people (people_id,first_name, last_name, phone, address)
    VALUES (nextval('people_seq'),p_first_name, p_last_name, p_phone, p_address)
    RETURNING people_id INTO new_people_id;

    -- Insert into employee table
    INSERT INTO employee (people_id, employee_id,join_date, title, wage, work_hours)
    VALUES (new_people_id, nextval('employee_seq'),CURRENT_DATE, p_title, p_wage,
    p_work_hours)

    RETURNING employee_id INTO new_employee_id;

    COMMIT;
END;
$$;
```

```

CREATE OR REPLACE PROCEDURE insert_employee(
    p_first_name VARCHAR(32),
    p_last_name VARCHAR(32),
    p_phone VARCHAR(24),
    p_address VARCHAR(128),
    p_title VARCHAR(64),
    p_wage DECIMAL(5,2),
    p_work_hours DECIMAL(10)
)
LANGUAGE plpgsql
AS $$
DECLARE
    new_people_id DECIMAL(12);
    new_employee_id DECIMAL(12);
BEGIN
    -- Insert into people table
    INSERT INTO people (people_id, first_name, last_name, phone, address)
    VALUES (nextval('people_seq'), p_first_name, p_last_name, p_phone, p_address)
    RETURNING people_id INTO new_people_id;

    -- Insert into employee table
    INSERT INTO employee (people_id, employee_id, join_date, title, wage, work_hours)
    VALUES (new_people_id, nextval('employee_seq'), CURRENT_DATE, p_title, p_wage, p_work_hours)
    RETURNING employee_id INTO new_employee_id;

    COMMIT;
END;
$$;

```

Messages
Query returned successfully in 85 msec.

When we call this procedure:

CALL insert_employee('John', 'Doe', '123-456-7890', '123 Main St', 'Manager', 25.50, 40);

Select to see the results:

	first_name character varying (32)	last_name character varying (32)	phone character varying (24)	address character varying (128)	title character varying (64)	wage numeric (5,2)	work_hours numeric (10)
1	John	Doe	123-456-7890	123 Main St	Manager	25.50	40

Query	Query History
1	select first_name, last_name, phone, address, title, wage, work_hours
2	from people
3	right join employee on employee.people_id = people.people_id
4	
5	
6	

2. Add a new customer into our database, and auto add into silver_customer table

Create a procedure for adding a new customer:

```
CREATE OR REPLACE PROCEDURE insert_customer(  
    p_first_name VARCHAR(32),  
    p_last_name VARCHAR(32),  
    p_phone VARCHAR(24),  
    p_address VARCHAR(128)  
  
)  
LANGUAGE plpgsql  
AS $$  
  
DECLARE  
    new_people_id DECIMAL(12);  
    new_customer_id DECIMAL(12);  
  
BEGIN  
  
    -- Insert into people table  
    INSERT INTO people (people_id, first_name, last_name, phone, address)  
    VALUES (nextval('people_seq'), p_first_name, p_last_name, p_phone, p_address)  
    RETURNING people_id INTO new_people_id;  
  
    -- Insert into customer table  
    INSERT INTO customer (people_id, customer_id)  
    VALUES (new_people_id, nextval('customer_seq'))  
    RETURNING customer_id INTO new_customer_id;  
  
    COMMIT;  
END;  
$$;
```

■ Trigger for auto add to silver_customer table

```
■ CREATE OR REPLACE FUNCTION add_to_silver_customer()  
    RETURNS TRIGGER AS $$  
    BEGIN  
        INSERT INTO silver_customer (customer_id)  
        VALUES (NEW.customer_id);  
  
        RETURN NEW;  
    END;  
    $$ LANGUAGE plpgsql;  
  
CREATE TRIGGER add_to_silver_customer_trigger  
AFTER INSERT ON customer  
FOR EACH ROW  
EXECUTE FUNCTION add_to_silver_customer();
```



```

CREATE OR REPLACE PROCEDURE insert_customer(
    p_first_name VARCHAR(32),
    p_last_name VARCHAR(32),
    p_phone VARCHAR(24),
    p_address VARCHAR(128)
)
LANGUAGE plpgsql
AS $$
DECLARE
    new_people_id DECIMAL(12);
    new_customer_id DECIMAL(12);
BEGIN
    -- Insert into people table
    INSERT INTO people (people_id, first_name, last_name, phone, address)
    VALUES (nextval('people_seq'), p_first_name, p_last_name, p_phone, p_address)
    RETURNING people_id INTO new_people_id;

    -- Insert into customer table
    INSERT INTO customer (people_id, customer_id)
    VALUES (new_people_id, nextval('customer_seq'))
    RETURNING customer_id INTO new_customer_id;

    COMMIT;
END;
$$;

```

Messages

Query returned successfully in 39 msec.

Call this procedure to create a new customer:

CALL insert_customer('Alice', 'Smith', '987-654-3210', '456 Elm St');

Select to see the result:

	customer_id numeric (12)	first_name character varying (32)	last_name character varying (32)
1	1	selina	liu
2	2	ray	liu
3	3	Alice	Smith

Query		Query History
1	select customer_id, first_name, last_name from customer	
2	left join people on people.people_id = customer.people_id	
3		
4		

We can see Alice Smith has been added.

3. Add a new category

```
CREATE OR REPLACE PROCEDURE insert_category(  
  p_category_name VARCHAR(32)  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
  INSERT INTO category (category_id,category_name)  
  VALUES (nextval('category_seq'),p_category_name);  
  
  COMMIT;  
END;  
$$;
```

```
CREATE OR REPLACE PROCEDURE insert_category(  
  p_category_name VARCHAR(32)  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
  INSERT INTO category (category_id,category_name)  
  VALUES (nextval('category_seq'),p_category_name);  
  
  COMMIT;  
END;  
$$;
```

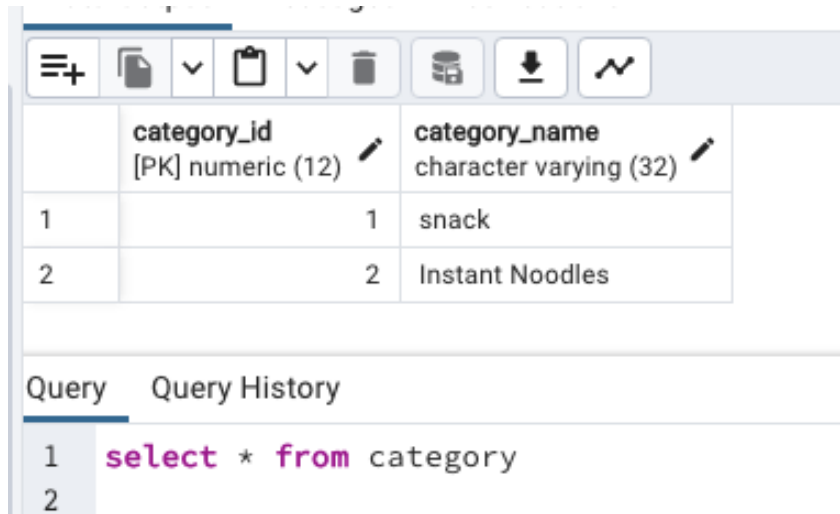
Messages

Query returned successfully in 98 msec.

Call this procedure to add a new category:

```
CALL insert_category('Instant Noodles');
```

Select to see the results:



The screenshot shows a database management interface. At the top, there is a toolbar with icons for various actions. Below the toolbar is a table with two columns: 'category_id' and 'category_name'. The 'category_id' column is marked as a primary key (PK) and is of type numeric (12). The 'category_name' column is of type character varying (32). The table contains two rows of data: one with '1' as the category_id and 'snack' as the category_name, and another with '2' as the category_id and 'Instant Noodles' as the category_name. Below the table, there is a section for 'Query' and 'Query History'. The 'Query' section shows a single query: 'select * from category'.

	category_id [PK] numeric (12)	category_name character varying (32)
1	1	snack
2	2	Instant Noodles

Query Query History

```
1 select * from category  
2
```

We can see instant noodles has been added.

4. Add a new product

Add a new product by the product name and category name

-- Procedure for inserting a new product with category name

```
CREATE OR REPLACE PROCEDURE insert_product(  
    p_product_name VARCHAR(64),  
    p_category_name VARCHAR(32)  
)  
LANGUAGE plpgsql  
AS $$  
  
DECLARE  
    new_product_id DECIMAL(12);  
    category_id_value DECIMAL(12);  
  
BEGIN  
  
    -- Get category_id based on category_name  
    SELECT category_id INTO category_id_value  
    FROM category  
    WHERE category_name = p_category_name;  
  
    -- Insert into product table  
    INSERT INTO product (product_id, product_name, category_id)  
    VALUES (nextval('product_seq'), p_product_name, category_id_value)  
    RETURNING product_id INTO new_product_id;  
  
    COMMIT;  
END;  
$$;
```

CREATE PROCEDURE

Query returned successfully in 91 msec.

Query Query History

```
1  -- Procedure for inserting a new product with category name
2  CREATE OR REPLACE PROCEDURE insert_product(
3      p_product_name VARCHAR(64),
4      p_category_name VARCHAR(32)
5  )
6  LANGUAGE plpgsql
7  AS $$
8  DECLARE
9      new_product_id DECIMAL(12);
10     category_id_value DECIMAL(12);
11 BEGIN
12     -- Get category_id based on category_name
13     SELECT category_id INTO category_id_value
14     FROM category
15     WHERE category_name = p_category_name;
16
17     -- Insert into product table
18     INSERT INTO product (product_id, product_name, category_id)
19     VALUES (nextval('product_seq'), p_product_name, category_id_value)
20     RETURNING product_id INTO new_product_id;
21
22     COMMIT;
23 END;
24 $$;
25
```

Insert a new product:

call insert_product('purified water','drinks')

Select to see results:

			product_name	category_name	
			character varying (32)	character varying (32)	
1	chips	snack			
2	purified water	drinks			

Query Query History

```
1  select product_name, category_name from product
2  join category on product.category_id = category.category_id
3
4
5
```

now we can see purified water has been added to the table

5. Add a new brand

```
CREATE OR REPLACE PROCEDURE insert_brand(
    p_brand_name VARCHAR(32)
)
LANGUAGE plpgsql
AS $$

BEGIN

    INSERT INTO brand (brand_id,brand_name)
    VALUES (nextval('brand_seq'),p_brand_name);

COMMIT;

END;
$$;
```

```
CREATE PROCEDURE
```

Query returned successfully in 108 msec.

Query	Query History
1	CREATE OR REPLACE PROCEDURE insert_brand(2 p_brand_name VARCHAR (32) 3) 4 LANGUAGE plpgsql 5 AS \$\$ 6 ▼ BEGIN 7 INSERT INTO brand (brand_id,brand_name) 8 VALUES (nextval('brand_seq'),p_brand_name); 9 10 COMMIT ; 11 END ; 12 \$\$; 13

Insert a new brand:

```
call insert_brand('pure life')
```

select to see the result:

	brand_id [PK] numeric (12)	brand_name character varying (32)
1	1	Lays
2	2	Dasani
3	3	kirkland
4	4	pure life

Query

Query History

```

1 select * from brand
2

```

We can see pure life has been add to this table.

6. Add a new supplier, given the contact people's information, add this info into the people table, and supplier_name into the supplier table

```
CREATE OR REPLACE PROCEDURE insert_supplier(  
    p_supplier_name VARCHAR(64), p_first_name VARCHAR(32), P_last_name VARCHAR(32),  
    p_phone VARCHAR(24), p_address VARCHAR(128) )  
LANGUAGE plpgsql  
AS $$  
  
DECLARE  
    new_people_id DECIMAL(12);  
  
BEGIN  
  
    -- Insert into people table  
    INSERT INTO people (people_id,first_name, last_name, phone, address)  
    VALUES (nextval('people_seq'),p_first_name, p_last_name, p_phone, p_address)  
    RETURNING people_id INTO new_people_id;  
  
    --insert into supplier table  
    INSERT INTO supplier(supplier_id,supplier_name,people_id)  
    VALUES (nextval('supplier_seq'),p_supplier_name, new_people_id);  
  
    COMMIT;  
END;  
$$;
```

Insert a new supplier:

call insert_supplier('xinzhe','Vivian','Lee','425-123-1234','12345 160th ave, seattle,98290');

select to see the result:

Data Output	Messages	Notifications
<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>		
supplier_name	first_name	last_name
character varying (64)	character varying (32)	character varying (32)
1 xinzhe	Vivian	Lee

Query	Query History
1	
2 select supplier_name, first_name, last_name	
3 from supplier	
4 join people on supplier.people_id = people.people_id	
5	

Now we can see this supplier has been add to our supplier table

7. When the store buys some products from suppliers and needs to add these products to our database, the price will be set the 120% of the cost of these goods.

-- Procedure for inserting a new goods

```
CREATE OR REPLACE PROCEDURE insert_goods(  
    p_brand_name VARCHAR(32), p_product_name VARCHAR(32), p_supplier_name VARCHAR(32),  
    p_quantity INTEGER, p_goods_cost DECIMAL(8,2)  
)  
LANGUAGE plpgsql  
AS $$
```

DECLARE

```
new_goods_id DECIMAL(12);  
new_brand_id DECIMAL(12);  
new_product_id DECIMAL(12);  
new_supplier_id DECIMAL(12);  
new_prices DECIMAL(10,2);  
goods_name_value VARCHAR(64);
```

BEGIN

```
-- Generate goods_name by concatenating brand_name and category_name  
goods_name_value := p_brand_name || ' ' || p_product_name;
```

```
-- Get brand_id based on brand_name
```

```
SELECT brand_id INTO new_brand_id  
FROM brand  
WHERE brand_name = p_brand_name;
```

```
-- Get product_id based on category_name
```

```
SELECT product_id INTO new_product_id  
FROM product  
WHERE product_name = p_product_name;
```

```
-- Get supplier_id based on supplier_name
```

```
SELECT supplier_id INTO new_supplier_id  
FROM supplier  
WHERE supplier_name = p_supplier_name;
```

```
-- Calculate prices (cost * 1.2)
```

```
new_prices := p_goods_cost * 1.2;
```

```
-- Insert into goods table
```

```
INSERT INTO goods (goods_id, brand_id, product_id, goods_name, supplier_id, buying_date, goods_cost,  
prices, quantity)  
VALUES (nextval('goods_seq'), new_brand_id, new_product_id, goods_name_value, new_supplier_id,  
CURRENT_DATE, p_goods_cost, new_prices, p_quantity)  
RETURNING goods_id INTO new_goods_id;
```

```

COMMIT;
END;
$$;

```

Insert a new goods:

call insert_goods('kirkland','purified water','costco','500','4.8');

select to see the results:

	goods_name character varying (64)	goods_cost numeric (8,2)	prices numeric (10,2)	quantity integer
1	kirkland purified water	4.80	5.76	500

Query	Query History
1	select goods_name, goods_cost,prices, quantity
2	from goods

8. Create a new order

Create a procedure for an new empty order, which only record the customer's info and employee info

-- Procedure for inserting a new order

```

CREATE OR REPLACE PROCEDURE insert_order(
    p_customer_first_name VARCHAR(32),
    p_customer_last_name VARCHAR(32),
    p_employee_id DECIMAL(12)
)
LANGUAGE plpgsql
AS $$

```

DECLARE

```

    new_order_id DECIMAL(12);
    customer_id_value DECIMAL(12);

```

BEGIN

-- Get customer_id based on first name and last name

```

SELECT c.customer_id INTO customer_id_value
FROM customer c

```

Join people on people.people_id = c.people_id

```

WHERE people.first_name = p_customer_first_name AND people.last_name =
p_customer_last_name;

```



```

-- Insert into orders table
INSERT INTO orders (order_id,customer_id, employee_id, total_amount)
VALUES (nextval('orders_seq'),customer_id_value, p_employee_id, 0)
RETURNING order_id INTO new_order_id;

COMMIT;
END;
$$;

```

Then create a procedure to record sold goods, and add them into previous empty order, to update the order:

```

-- Procedure for inserting multiple sold_goods entries with the same order_id
CREATE OR REPLACE PROCEDURE insert_multiple_sold_goods(
    p_goods_and_counts JSONB -- JSON array of objects: [{"goods_id": ..., "count": ...}, ...]
)
LANGUAGE plpgsql
AS $$

DECLARE
    goods_record JSONB;
    new_order_id DECIMAL(12);
    goods_id_value DECIMAL(12);
    count_value INTEGER;

BEGIN

    -- Set the order_id
    SELECT currval('orders_seq') INTO new_order_id;
    -- create a new order table

    -- Loop through the JSON array and insert sold_goods entries
    FOR goods_record IN SELECT * FROM jsonb_array_elements(p_goods_and_counts)
    LOOP
        -- Extract values from JSONB
        goods_id_value := (goods_record->>'goods_id')::DECIMAL(12);
        count_value := (goods_record->>'count')::INTEGER;

        INSERT INTO sold_goods (sold_goods_id,goods_id, order_id, count)
        VALUES (nextval('sold_goods_seq'),goods_id_value, new_order_id, count_value);

        -- Update total_amount for the order based on sold goods prices
        UPDATE orders o
        SET total_amount = total_amount + (sg.count * g.prices)
        FROM goods g
        JOIN sold_goods sg ON sg.goods_id = g.goods_id
        WHERE o.order_id = new_order_id AND sg.sold_goods_id = currval('sold_goods_seq');
    END LOOP;

```

```

COMMIT;
END;
$$;

```

Insert the data:

```

CALL insert_order('selina','liu','1');
CALL insert_multiple_sold_goods(
  '{"goods_id": 3, "count": 3}, {"goods_id": 2, "count": 2}':JSONB -- Replace with actual JSON
  array of goods and counts
);

```

Select to see the results:

Data Explorer					
Data Explorer					
	first_name character varying (32)	last_name character varying (32)	goods_name character varying (64)	count integer	total_amount numeric (12,2)
1	selina	liu	kirkland chips	3	15.84
2	selina	liu	kirkland purified water	2	15.84

Query		Query History	
1			
2	select first_name, last_name, goods_name, count, total_amount		
3	from orders		
4	join customer on customer.customer_id = orders.customer_id		
5	join people on people.people_id = customer.people_id		
6	join sold_goods on sold_goods.order_id = orders.order_id		
7	join goods on goods.goods_id = sold_goods.goods_id		

We can see this order is added to the orders table

Question Identification and Explanations

Update, if necessary, your questions useful to the organization and explanations as to why they are useful.

First Query

This query should retrieve information from at least four tables joined by associative relationships. Ensure that the question driving this query is applicable and useful for the intended use of the database.

Question:

What is the most popular brand and products in this store for last month, how many are left and how much do we earn from these goods?

This could retrieve the data for the most popular brands and their products in this store, this will track the sales data to make sure which we should purchase more to make wise business decisions. If this kind of goods has a very good sales volume, but a few left, we should buy from our own suppliers now, in case we lost our customers.

Second Query

This query should retrieve information from the subtypes and supertype you have in your project, the entities that participate in the specialization-generalization relationship. The query should require joins to one or more of the subtypes, as well as the supertype, to fully address the question. Ensure that the question driving this query is applicable and useful for the intended use of the database.

Question:

How important is the silver customer to our sales?

No store can be attractive to all kinds of people, and we must identify our core customer segments so that we can better plan our sales and target our purchases.

Third Query

Create a view that captures information that needs be accessed regularly based upon the use of your database. Utilize the view in the query. To ensure sufficient complexity, the query (or underlying view) should contain at least two of the constructs below. There should be at least one from each group.

Group 1 (choose one or more)

- joins of at least two tables.
- one or more restrictions in the WHERE clause.
- an order by statement.

Question:

How much did we sell in the category of “drinks” last month?

We need to track those data for all categories to make business decisions, for example, usually, summer is the peak season for drinks, if we only sold a little for the month like “July”, that means we need to finger out the reason and do something to improve the sell volume.

Group 2 (choose one or more)

- at least one aggregate function. • at least one subquery.
- a having clause.
- a left or right join.
- joins of four or more tables. • a union of two queries.

Question:

What is the most popular product and minimum sell product?

We need to track our most popular product and the least sold product for business needs, the product which has a good sales amount, we need to keep purchasing in case of the shortage, for the least sold product we need to finger out the reason and make a plan for a set discount on this product.

Query Executions and Explanations

Update, if necessary, your queries answering the questions, along with screenshots and explanations.

What is the most popular brand and products in this store for last month, how many are left and how much do we earn from these goods?

To answer this question, I use this query:

```
SELECT
    b.brand_name,
    p.product_name,
    g.quantity,
    SUM(sg.count) AS total_items_sold,
    SUM(sg.amount) AS total_amount,
    g.goods_cost * SUM(sg.count) AS total_cost,
    SUM(sg.amount) - g.goods_cost * SUM(sg.count) AS total_profit
FROM
    sold_goods sg

JOIN
    goods g ON sg.goods_id = g.goods_id AND g.buying_date > '2023-07-07'
JOIN
    brand b ON g.brand_id = b.brand_id
JOIN
    product p ON p.product_id = g.product_id
GROUP BY
    b.brand_name, p.product_name, g.goods_cost, g.quantity
ORDER BY
    total_items_sold DESC;
```

screenshot:

	brand_name character varying (32)	product_name character varying (32)	quantity integer	total_items_sold bigint	total_amount numeric	total_cost numeric	total_profit numeric
1	sunkist	soda	404	96	748.80	624.00	124.80
2	kirkland	purified water	498	52	299.52	249.60	49.92
3	kirkland	chips	197	43	61.92	51.60	10.32
4	sunkist	orange juice	460	40	480.00	400.00	80.00
5	sunkist	apple juice	480	20	288.00	240.00	48.00

Query
Query History

```

1  SELECT
2      b.brand_name,
3      p.product_name,
4      g.quantity,
5      SUM(sg.count) AS total_items_sold,
6      SUM(sg.amount) AS total_amount,
7      g.goods_cost * SUM(sg.count) AS total_cost,
8      SUM(sg.amount) - g.goods_cost * SUM(sg.count) AS total_profit
9  FROM
10     sold_goods sg
11
12
13  JOIN
14     goods g ON sg.goods_id = g.goods_id AND g.buying_date > '2023-07-07'
15  JOIN
16     brand b ON g.brand_id = b.brand_id
17  JOIN product p ON p.product_id = g.product_id
18  GROUP BY
19     b.brand_name, p.product_name, g.goods_cost, g.quantity
20  ORDER BY
21     total_items_sold DESC;

```

From this table, we know that:

For the last month, the most popular item is Sunkist soda, and it sold 96 in total, but it still has 404 left, which means we do not need to buy new right now. And Sunkist apple juice only sold 20, which means we need to figure out a way to make this product sell more, otherwise it may be expired when we are still not sold out.

How important is the silver customer to our sales?

To answer this question, we need to compare how much the silver customers has been spent on this store, and how much we sold in total.

SELECT

(SELECT SUM(credits) FROM customer join silver_customer sc on sc.customer_id = customer.customer_id) AS total_credits_silver,

(SELECT SUM(credits) FROM customer) AS total_credits_all

Screenshot:

Data Output			Messages	Notifications
	total_credits_silver numeric	total_credits_all numeric		
1	820	201405		

Query	Query History
1	SELECT
2	(SELECT SUM(credits) FROM customer join silver_cu
3	(SELECT SUM(credits) FROM customer) AS total_cred
4	

From this result we can easily see that the silver customer only contributes a litter to our sales, our target customers should be gold or diamond, to keep them in business, we must know what they like to purchase in our store.

How much did we sell in the category of “drinks” last month?

First, we create view :

```
CREATE VIEW sold_goods_view AS
```

```
SELECT sg.amount AS sold_amount, p.product_name, c.category_name, o.order_date
```

```
FROM sold_goods sg
```

```
JOIN goods g ON sg.goods_id = g.goods_id
```

```
JOIN product p ON g.product_id = p.product_id
```

```
JOIN category c ON p.category_id = c.category_id
```

```
join orders o ON o.order_id = sg.order_id;
```

Then we check how much we earned from this view:

```
SELECT SUM(sold_amount) AS total_earnings_in_drinks
```

```
FROM sold_goods_view
```

```
WHERE category_name = 'drinks'
```

```
AND order_date > '2023-07-12';
```

	total_earnings_in_drinks	numeric	
1		3624.00	

Query	Query History
1	CREATE VIEW sold_goods_view AS
2	SELECT sg.amount AS sold_amount, p.product_name, c.category_name, o.order_date
3	FROM sold_goods sg
4	JOIN goods g ON sg.goods_id = g.goods_id
5	JOIN product p ON g.product_id = p.product_id
6	JOIN category c ON p.category_id = c.category_id
7	join orders o ON o.order_id = sg.order_id;
8	
9	SELECT SUM(sold_amount) AS total_earnings_in_drinks
10	FROM sold_goods_view
11	WHERE category_name = 'drinks'
12	AND order_date > '2023-07-12';
13	

From this result, we could see how many drinks we have sold for last month, we could compare will history to see how is our business going.

What is the most popular product and minimum sell product?

First, create a view for max and minimum sell product:

```
CREATE VIEW product_sales_summary AS
SELECT product_amount_sum.product_name AS name, product_amount_sum.total_sold_amount AS
total_sale, 'max' AS summary_type
FROM (
    SELECT product_name, SUM(amount) AS total_sold_amount
    FROM product
    JOIN goods ON product.product_id = goods.product_id
    JOIN sold_goods ON goods.goods_id = sold_goods.goods_id
    GROUP BY product_name
    ORDER BY total_sold_amount DESC
    LIMIT 1
) AS product_amount_sum

UNION

SELECT product_amount_sum.product_name AS name, product_amount_sum.total_sold_amount AS
total_sale, 'min' AS summary_type
FROM (
    SELECT product_name, SUM(amount) AS total_sold_amount
```



```

FROM product
JOIN goods ON product.product_id = goods.product_id
JOIN sold_goods ON goods.goods_id = sold_goods.goods_id
GROUP BY product_name
ORDER BY total_sold_amount ASC
LIMIT 1
) AS product_amount_sum;

```

Then, select to see the results:

```

SELECT name, total_sale, summary_type
FROM product_sales_summary;

```

	name character varying (32)	total_sale numeric	summary_type text
1	purified water	299.52	min
2	remen	3168.00	max

Query

Query History

```

1 CREATE VIEW product_sales_summary AS
2 SELECT product_amount_sum.product_name AS name, product_amount_sum.total_sold_amount AS total_sale,
3 FROM (
4     SELECT product_name, SUM(amount) AS total_sold_amount
5     FROM product
6     JOIN goods ON product.product_id = goods.product_id
7     JOIN sold_goods ON goods.goods_id = sold_goods.goods_id
8     GROUP BY product_name
9     ORDER BY total_sold_amount DESC
10    LIMIT 1
11 ) AS product_amount_sum
12
13 UNION
14
15 SELECT product_amount_sum.product_name AS name, product_amount_sum.total_sold_amount AS total_sale,
16 FROM (
17     SELECT product_name, SUM(amount) AS total_sold_amount
18     FROM product
19     JOIN goods ON product.product_id = goods.product_id
20     JOIN sold_goods ON goods.goods_id = sold_goods.goods_id
21     GROUP BY product_name
22     ORDER BY total_sold_amount ASC
23    LIMIT 1
24 ) AS product_amount_sum;
25
26 SELECT name, total_sale, summary_type
27 FROM product_sales_summary;
28

```

From this result, we can tell the least sold is purified water, which only has 299.52 as total sale amount, and the most popular product is remen, which has over 3000 amounts, so we need to keep an eye for our quantity of ramen, and why purified water has so bad sell volume.

Index Identification and Creations

Update, if necessary, the indexes identifications useful to your database, explanations as to why they help, and screenshots of their creations.

1. Identify all primary keys so you know which columns are already indexed.

Primary key column	Description
People_id	For people_table
Employee_id	For employee table
Customer_id	For Customer table
brand_id	For brand table
Category_id	For category table
Goods_id	For goods table
Product_id	For product table
Supplier_id	For supplier table
Order_id	For order table

2. Identify all foreign keys so you know which columns must have an index without further analysis.

Foreign key column	Unique	Description
people_id	true	This is the foreign key used in the customer, supplier, and employee table, this index should be unique, because customer, supplier, employee are different roles.
Customer_id	false	This foreign key is in order table, this is not unique, since a customer could order multiful times
Brand_id	false	This foreign key is used in goods, this is not unique, because a brand could have many different goods
Category_id	false	This foreign key is used in product table, this is not unique, since a category could have a lot of products
Supplier_id	false	This foreign key is used in goods table, this is also not unique, because a supplier could supply different goods
Employee_id	false	This foreign key is used in order table, this is not unique, because an employee could have operated many different orders

Create Index in PostgreSQL:

-- Creating a unique index for people_id

```
CREATE UNIQUE INDEX idx_unique_people_id ON customer USING btree (people_id);
```

```
CREATE UNIQUE INDEX idx_people_id_supplier ON supplier USING btree (people_id);
```

```
CREATE UNIQUE INDEX idx_people_id_employee ON employee USING btree (people_id);
```

-- Creating indexes for the other foreign key columns

```
CREATE INDEX idx_order_customer_id ON orders USING btree (customer_id);
```

```
CREATE INDEX idx_goods_brand_id ON goods USING btree (brand_id);
```

```
CREATE INDEX idx_product_category_id ON product USING btree (category_id);
```

```
CREATE INDEX idx_goods_supplier_id ON goods USING btree (supplier_id);  
CREATE INDEX idx_order_employee_id ON orders USING btree (employee_id);
```

3. Identify three query-driven indexes that are based on the queries you defined in this iteration. These indexes will not be placed on foreign or primary keys, but on other columns because the queries use them.

For the queries, we used in this iteration, and the using scenario in the wholesale store,

(1) order_date in order table is very common use when we are tracking the sales volume.

```
CREATE INDEX idx_order_date ON orders(order_Date);
```

(2) Product_name is often used for tracking the business because the needs for a specific product vary from seasons and holidays, we need to track their data

```
CREATE INDEX idx_product_name ON product(product_name);
```

(3) Since phone numbers should be unique, when we search for specific people, and we have an index for a phone number, it will definitely improve the efficiency of searching. Because when we add customer info to orders, the quickest way is to search for the phone number if we don't have the id number for them.

```
CREATE UNIQUE INDEX idx_unique_phone ON people (phone);
```

History Table Demonstration

Explain the specifics of your history table, including how the trigger works, and demonstrate that the history table captures changes.

In my goods_change table, it could capture the cost change and the supplier change, so my code is shown below :

```
CREATE OR REPLACE FUNCTION goods_update_trigger()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.goods_cost <> OLD.goods_cost AND NEW.supplier_id <> OLD.supplier_id THEN
        INSERT INTO goods_change (change_id,goods_id, change_date, old_cost, new_cost,
old_supplier_id, new_supplier_id)
        VALUES (nextval('goods_change_seq'),NEW.goods_id, CURRENT_DATE, OLD.goods_cost,
NEW.goods_cost, OLD.supplier_id, NEW.supplier_id);
    ELSIF NEW.goods_cost <> OLD.goods_cost THEN
        INSERT INTO goods_change (change_id,goods_id, change_date, old_cost, new_cost)
        VALUES (nextval('goods_change_seq'),NEW.goods_id, CURRENT_DATE, OLD.goods_cost,
NEW.goods_cost);
    ELSIF NEW.supplier_id <> OLD.supplier_id THEN
        INSERT INTO goods_change (change_id,goods_id, change_date, old_supplier_id, new_supplier_id)
        VALUES (nextval('goods_change_seq'),NEW.goods_id, CURRENT_DATE, OLD.supplier_id,
NEW.supplier_id);
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_goods_change_trigger
AFTER UPDATE ON Goods
FOR EACH ROW
EXECUTE FUNCTION goods_update_trigger();
```

Code	Description
CREATE OR REPLACE FUNCTION goods_update_trigger()	Create a function named goods_update_trigger
RETURNS TRIGGER AS \$\$	Return a special type called Trigger, and AS \$\$ indicate the start of this actual trigger body
BEGIN	The syntax for starting the trigger block
IF NEW.goods_cost <> OLD.goods_cost AND NEW.supplier_id <> OLD.supplier_id THEN	If cost and supplier_id both got changed, then insert a new row to goods_change table, set the new cost to new_goods_cost, and use old cost to

<pre> INSERT INTO goods_change (change_id,goods_id, change_date, old_cost, new_cost, old_supplier_id, new_supplier_id) VALUES (nextval('goods_change_seq'),NEW.goods_id, CURRENT_DATE, OLD.goods_cost, NEW.goods_cost, OLD.supplier_id, NEW.supplier_id); </pre>	<p>fill old_goods_cost field, and also set old supplier_id and new supplier_id</p>
<pre> ELSIF NEW.goods_cost <> OLD.goods_cost THEN INSERT INTO goods_change (change_id,goods_id, change_date, old_cost, new_cost) VALUES (nextval('goods_change_seq'),NEW.goods_id, CURRENT_DATE, OLD.goods_cost, NEW.goods_cost); ELSIF NEW.supplier_id <> OLD.supplier_id .THEN INSERT INTO goods_change (change_id,goods_id, change_date, old_supplier_id, new_supplier_id) VALUES (nextval('goods_change_seq'),NEW.goods_id, CURRENT_DATE, OLD.supplier_id, NEW.supplier_id); </pre>	<p>This is for when only the cost got changed, but the supplier is not change, for this situation, we only insert value for old_goods_cost and new goods_cost</p>
<pre> END IF; </pre>	<p>The end of if /else statement</p>
<pre> RETURN NEW </pre>	<p>Keep the changes made to the row in the goods table, also capturing these change in goods_change table</p>
<pre> END; </pre>	<p>The ends of the trigger definition</p>
<pre> \$\$ LANGUAGE plpgsql; </pre>	<p>The end of the trigger body, and the language is postgresql</p>
<pre> CREATE TRIGGER update_goods_change_trigger </pre>	<p>Create a trigger named update_goods_change_trigger</p>
<pre> AFTER UPDATE ON Goods </pre>	<p>When we need to call this trigger, in this case, is when we update the goods table</p>
<pre> FOR EACH ROW </pre>	<p>Execute range</p>
<pre> EXECUTE FUNCTION goods_update_trigger(); </pre>	<p>Execute the function named goods_update_trigger</p>

since we define three conditions, so let us update 3 different sets of date to check our result :

(1) both cost and supplier are changed

UPDATE Goods

SET supplier_id = 2,

goods_cost = 6.3

WHERE goods_name = 'sunkist soda';

(2) Only goods_cost changed

UPDATE Goods

SET goods_cost = 9.8

WHERE goods_name = 'sunkist orange juice';

(3) Only supplier changed

UPDATE Goods

SET supplier_id = 2

WHERE goods_name = 'sunkist apple juice';

Screenshot to demonstrate the change:

	change_id [PK] numeric (12)	goods_id numeric (12)	change_date date	old_cost numeric (8,2)	new_cost numeric (8,2)	old_supplier_id numeric (12)	new_supplier_id numeric (12)
1	1	4	2023-08-13	6.50	6.30	4	2
2	2	6	2023-08-13	[null]	[null]	4	2
3	3	5	2023-08-13	10.00	9.80	[null]	[null]

Query

Query History

1 select * from goods_change

2

3

4

We could easily see that 3 records are created, and the first one has both cost change and supplier change, and the second one only has supplier change, third one only has cost change, so that means, our code is correct.

Data Visualizations

Include and explain data visualizations and stories that tell effective stories about the information in a way that people quickly and accurately understand it.

(1) The flashback sorting of the product sales

Command Line:

```
SELECT product_amount_sum.product_name AS product, product_amount_sum.total_sold_count
FROM (
  SELECT product_name, SUM(count) AS total_sold_count
  FROM product
  JOIN goods ON product.product_id = goods.product_id
  JOIN sold_goods ON goods.goods_id = sold_goods.goods_id
  GROUP BY product_name
  ORDER BY total_sold_count DESC
) AS product_amount_sum
```

Code	description
SELECT product_amount_sum.product_name AS product, product_amount_sum.total_sold_count	Select product_name from product_amount_sum table, and name the column to product, and select the total_sold_count from product_amount_sum table
FROM (Indicate the date will select from the following statement
SELECT product_name, SUM(count) AS total_sold_count	Select product_name, and the sum of the count, rename it to total_sold_count
FROM product	From product table
JOIN goods ON product.product_id = goods.product_id	Inner join the goods table
JOIN sold_goods ON goods.goods_id = sold_goods.goods_id	Inner join the sold_goods table
GROUP BY product_name	Group the column of product_name , then help we get the result for sum of the count
ORDER BY total_sold_count DESC	Sort the total_sort_count in descreasing order
AS product_amount_sum	Name previous results as product_amount_sum

	product character varying (32)	total_sold_count bigint
1	chips	339
2	ramen	300
3	orange juice	110
4	apple juice	108
5	soda	96
6	purified water	52

Query
Query History

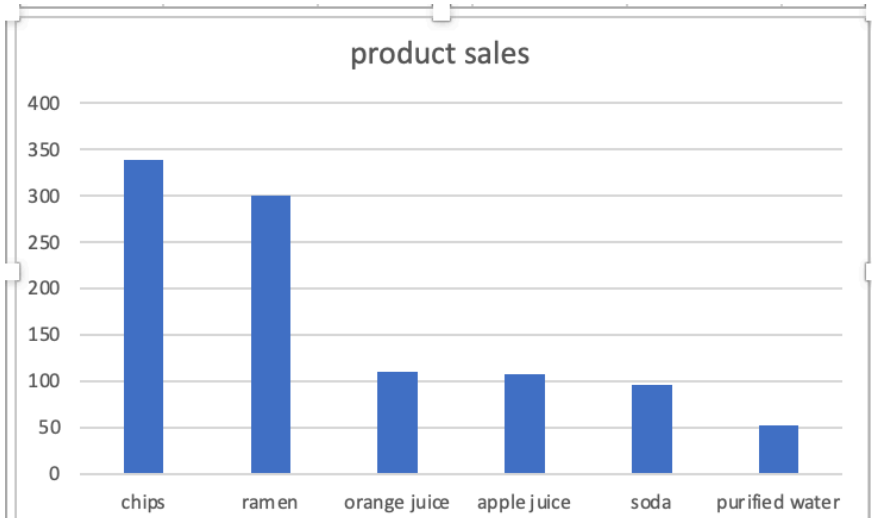
```

1
2 SELECT product_amount_sum.product_name AS product, product_amount_sum.total_sold_count
3 FROM (
4     SELECT product_name, SUM(count) AS total_sold_count
5     FROM product
6     JOIN goods ON product.product_id = goods.product_id
7     JOIN sold_goods ON goods.goods_id = sold_goods.goods_id
8     GROUP BY product_name
9     ORDER BY total_sold_count DESC
10 ) AS product_amount_sum
11
12

```

Convert to csv files:

	A	B
	product	total_sold_count
	chips	339
	ramen	300
	orange juice	110
	apple juice	108
	soda	96
	purified water	52



From this chart, we know there are only 6 products that have been sold in our store, and the most popular product is chips, and ramen is the second one, they have similar sales volume. Other products which have less consumption are beverages, and obviously, juices are more attractive to the customers, even soda is not sold very well, but it is still almost 2 times more than the water’s sales, which indicates that the customers may prefer flavored drinks.

(2) What is the average cost of soda for each season?

Command Line:

```
SELECT
CASE
    WHEN EXTRACT(MONTH FROM change_date) IN (12, 1, 2) THEN 'Winter'
    WHEN EXTRACT(MONTH FROM change_date) IN (3, 4, 5) THEN 'Spring'
    WHEN EXTRACT(MONTH FROM change_date) IN (6, 7, 8) THEN 'Summer'
    WHEN EXTRACT(MONTH FROM change_date) IN (9, 10, 11) THEN 'Fall'
END AS season,
TO_CHAR(AVG(new_cost), '$99.99') AS goods_cost
FROM goods_change
join goods on goods.goods_id = goods_change.goods_id
join product on product.product_id = goods.product_id
where product_name = 'soda'
GROUP BY season;
```

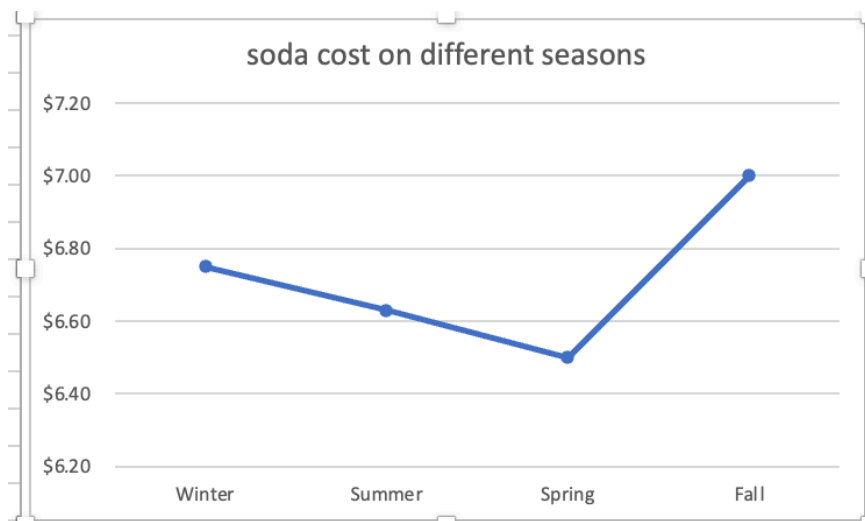
Code	Description
SELECT	Indicate the columns we need to display
CASE	Indicate the different case statement
WHEN EXTRACT(MONTH FROM change_date) IN (12, 1, 2) THEN 'Winter'	If the month from change_date is 12, 1 or 2, that means these data belongs to winter
WHEN EXTRACT(MONTH FROM change_date) IN (3, 4, 5) THEN 'Spring'	If the month from change_date is 3, 4 or 5, that means these data belongs to spring
WHEN EXTRACT(MONTH FROM change_date) IN (6, 7, 8) THEN 'Summer'	If the month from change_date is 6,7,8, that means these data belongs to summer
WHEN EXTRACT(MONTH FROM change_date) IN (9, 10, 11) THEN 'Fall'	If the month from change_date is 9,10,12, that means these data belongs to fall
END AS season,	End the case statement, and this column name season
TO_CHAR(AVG(new_cost), '\$99.99') AS goods_cost	Format the average of the new cost to "\$99.99" , and name the column goods_cost
FROM goods_change	From goods_change table
join goods on goods.goods_id = goods_change.goods_id	Join the table goods
join product on product.product_id = goods.product_id	Join the product table
where product_name = 'soda'	Filter the data only related to soda
GROUP BY season;	Group the season column to get the average cost for each season

	season text	goods_cost text
1	Winter	\$ 6.75
2	Summer	\$ 6.63
3	Spring	\$ 6.50
4	Fall	\$ 7.00

Query	Query History
1	
2	SELECT
3	CASE
4	WHEN EXTRACT(MONTH FROM change_date) IN (12, 1, 2) THEN 'Winter'
5	WHEN EXTRACT(MONTH FROM change_date) IN (3, 4, 5) THEN 'Spring'
6	WHEN EXTRACT(MONTH FROM change_date) IN (6, 7, 8) THEN 'Summer'
7	WHEN EXTRACT(MONTH FROM change_date) IN (9, 10, 11) THEN 'Fall'
8	END AS season,
9	TO_CHAR(AVG(new_cost), '\$99.99') AS goods_cost
10	FROM goods_change
11	join goods on goods.goods_id = goods_change.goods_id
12	join product on product.product_id = goods.product_id
13	where product_name = 'soda'
14	GROUP BY season
15	

Convert to csv date:

season	goods_cost
Winter	\$6.75
Summer	\$6.63
Spring	\$6.50
Fall	\$7.00



We could easily see the different costs of soda for each season, obviously, spring has the lowest cost of the whole year, which means, if we ordered soda products in spring, we got the most reasonable cost for this product, considering of high demand for summer and the expire date for soda, it should be a good choice of hoarding this product. In the fall season, the cost gets increased a lot, so we should avoid hoarding soda in the fall season, we could wait until winter or summer.

Summary and Reflection

Update the concise summary of your project and the work you have completed thus far, and additionally update your questions, concerns, and observations, so that you and your facilitator or instructor are aware of them and can communicate about them.

In this project, I am trying to design a comprehensive database for my mom's wholesale store, this database aims to streamline employee management, sales management, customer tracking, and inventory management and provide valuable insights through reporting and analysis, to help the user to make data-driven decisions and improve overall business performance.

But I still have some questions and concerns:

1. How to integrate the data, these tables are related to each other, how they are connected and influence automatically.
2. User interface and accessibility: How to design a user-friendly interface to be easier the use for non-skilled people like my mom .
3. Security and privacy: how to protect sensitive information, such as customers' basic info and product cost, etc.
4. Integration with supplier system, usually their supplier is the company who produces that product or the first-level wholesale store owners, they should have a mature database to manage their company, can we integrate their system and get real-time updates?