**HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

\*\*\*\*\*\*\*\*\*\*\*



REPORT FOR DATABASE PROJECT

**Cafe Chain Management System**

**Lecturer:** PhD. Nguyen Thi Oanh

|  |  |
| --- | --- |
| **Student:** | Nguyen Hoang Viet - 20215255  Nguyen Van Duong –20215189  Nguyen Viet Thang - 20215245 |

***Hanoi, January 2024***

Contents

1. [DESCRIPTION OF THE PROJECT 4](#_bookmark0)
   1. OVERALL 4
   2. CAFE MANAGEMENT SYSTEM 4
   3. [FEATURES 5](#_bookmark3)
      1. [FOR EMPLOYEES AT A BRANCH (CASHIER, BRACH MANAGER) 5](#_bookmark4)
      2. [FOR MANAGERS AND ABOVE (CEOS, CHAIN MANAGER) 6](#_bookmark5)
2. [ENTITY RELATION DIAGRAM 7](#_bookmark6)
3. [RELATIONSHIP DIAGRAM 10](#_bookmark7)
4. [QUERY 12](#_bookmark8)
   1. [CREATE TABLE QUERY 12](#_bookmark9)
   2. [STORED PROCEDURE 12](#_bookmark10)
   3. ROLE
   4. [INDEX 12](#_bookmark11)
   5. [FUNCTIONS 12](#_bookmark12)
      1. [CALCULATE THE SHIPPING FEE 12](#_bookmark13)
      2. [GET THE PAYMENT FOR A BILL 13](#_bookmark14)
      3. [FIND MOST EFFICIENT BRANCH IN A CHAIN 14](#_bookmark15)
      4. [GET THE DAILY INCOME 15](#_bookmark16)
      5. [GET THE INFORMATION OF A BILL 16](#_bookmark17)
      6. [FIND LOYAL CUSTOMERS 17](#_bookmark18)
      7. [FIND MOST/LEAST FAVORITED PRODUCT 18](#_bookmark19)
      8. [CALCULATING THE SALARY 21](#_bookmark20)
      9. [SUGGEST ITEM FOR A CUSTOMER 21](#_bookmark21)

4.5.10 DELETE/INSERT FUNCTION…………………………………………………….

1. TRIGGER……………………………………………………………………………
2. VIEW…………………………………………………………………………………
3. [CONTRIBUTION 23](#_bookmark22)

# Description of the project

# Overall

* + - Nowadays, coffee shops come with all shapes and sizes (vintage, modern...). And with the keep-improving service, coffee is being enjoyed by people from walks of life, gender in even the farthest regions of the world. They have mushroomed over the area all over the world.
    - Vietnam is not an exception. You almost can’t find any street that doesn’t have a coffee shop on it. However, to operate smoothly and effectively, what they need is a database management system of **a chain coffee shop system**.
    - In this project, we will build a database for a chain coffee shop system based on our demand of small part of this system: The flow of the management in products given out, the interaction between customers, businesses, and orders.
    - We apply the Top-down database design method. From the requirements of reality, we determine the entities, then build the entity relation diagram (ERD). Once, we have the ERD, we figure the relation schema. Finally, we construct all the database in PostgreSQL database management system and the features around demanding.

# Cafe management system.

* + - **NOTE:** Our system is about a cafe chain. There are many branches in a chain, each branch is a coffee shop.
    - **Decentralization:** In the system, there are different roles: CEOs, chain managers, employees (Bartender, cashier, shipper, waiter, manager...). Foreach role, they have their own attributes and method: name, date of birth, address...

# Branch management:

* + - * As a chain stores, we will have a lot of branches in the system. Our system will control the data, workflow of the branches.
      * Each branch will have the information: branch\_id, address, contact\_number, manager\_id.

# Menu management:

* + - * To manage all the products are being sold, we will have a menu to store all data of them.
      * In the menu, the products are saved as: item\_id, item\_name, price.

# Job management:

* + - * There are many positions in a chain, we classify these into jobs to hierarchize people working in the system.
      * To be easier for understanding, we will give an example:

 A CEO can access all the data in the system. CEO has the right to modify system: menu, job, staff...

 A chain manager can access all the data of all branches in the chain.

 A branch manager can access all the data of the branch he/she manages

 A cashier can only add the order, export out the bill, the data of the order and bill will be sent to the server.

 Every people in the system can take, interact with customer.

* + - * However, in our project, we are focus on the data flow in system. The interaction between user and system classification is not included. When the project is developed into application having GUI, we will add more features to our project.

# Staff management:

* + - * The number of staffs in a branch is big and it is bigger in a whole chain system. To control and arrange them effectively we create database of staffs
      * The database will include staff\_id, basic information (name, hometown, date of birth...), work hour, status (parttime or fulltime).
      * The salary is based status and work hour and his or her job.

# Order management:

* + - * The customer will place orders including drinks, foods, products... This database will store all the information of all order.
      * Each order will have the order\_id, status (Online or Offline), discount, distance, date...
      * The total payment will be calculated once the ordered is taken.

# Customer management:

* + - * The more developed the system is, the larger number of customers is. The branches are existed everywhere, the customers can be come from anywhere, therefore, we need a database to manage all the customer.
      * This database can be access from any branch in the system for the convenience.
      * We manage the information of the customer as follow: name, customer\_id, phone number, there order history...

# Features

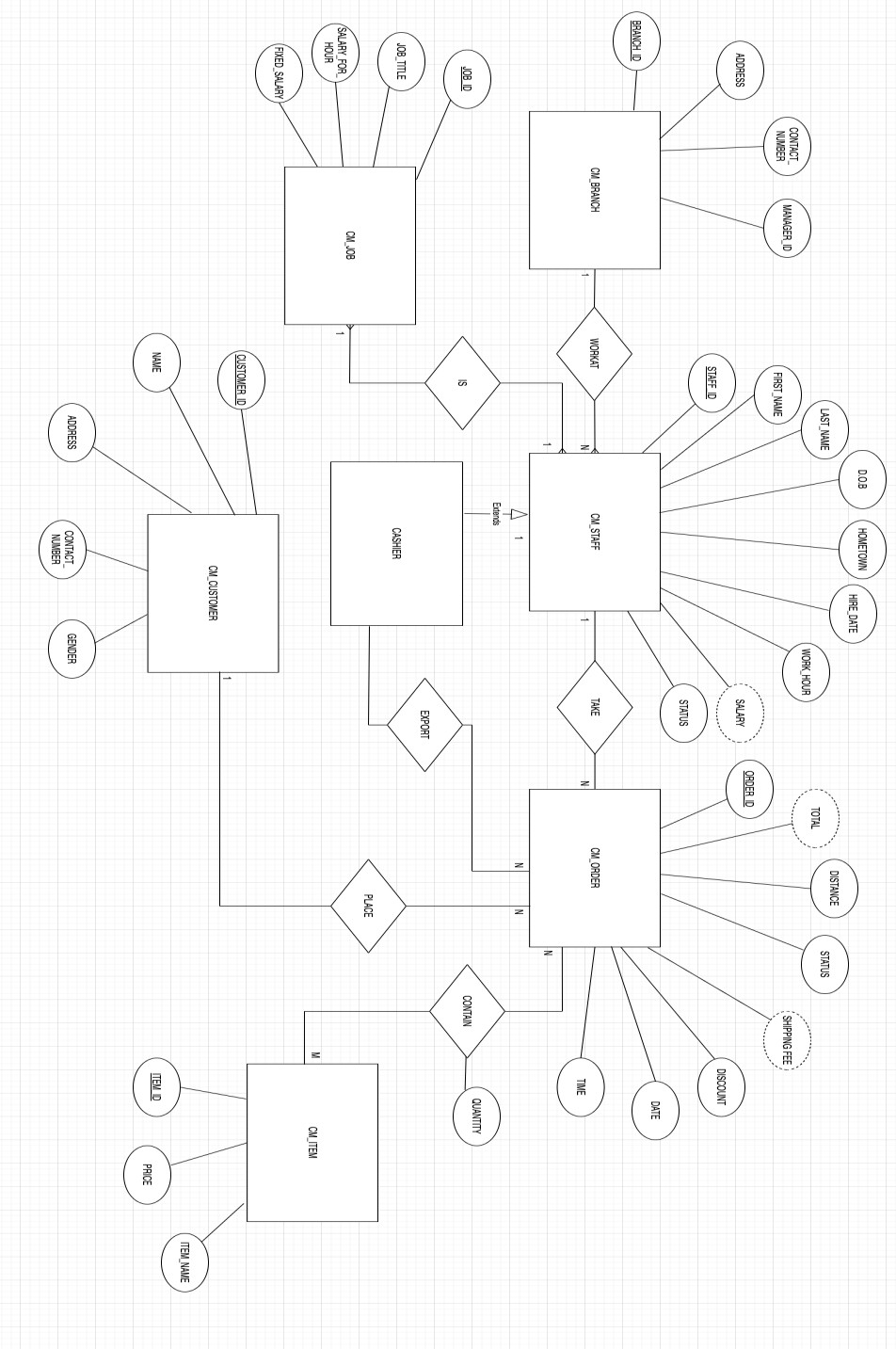
# For employees at a branch (Cashier, Brach manager)

* + - * Calculate shipping fee.
      * Calculate the total payment for a bill and send back data to the system.
      * Export the bill by order.
      * Suggest item for each customer.

# For managers and above (CEOs, Chain manager)

* + - * Get data of customers:
        + Classify the customers into levels of the membership.
        + Know which products are interested or disregarded.
      * Get data of branch, system:
        + Get the revenue of the branch and whole system.
        + Know which branch the employee works.
        + The efficiency of branches, staffs...

# Entity relation diagram



*Entity relation diagram of Cafe chain management system*

* There are 6 entities in this diagram:
  + CM\_BRANCH
  + CM\_STAFF
  + CM\_JOB
  + CM\_ITEM
  + CM\_CUSTOMER
  + CM\_ORDER
* CM\_BRANCH:
  + For branch, we create an entity called CM\_BRANCH to present branch in the system.
  + The BRANCH\_ID is primary key for this entity.
  + CM\_BRANCH entity has attributes: ADDRESS, CONTACT\_NUMBER, MANAGER\_ID
* CM\_STAFF:
  + This entity presents the employees who work for this cafe chain system.
  + Attributes: STAFF\_ID, FIRST\_NAME, LAST\_NAME, D.O.B, HOMETOWN, HIRED\_DATE, WORK\_HOUR, STATUS, SALARY.
  + SALARY is a dependent attribute: This attribute will be calculated by the attributes: WORK\_HOUR, STATUS, and entity CM\_JOB.
  + STAFF\_ID is PRIMARY KEY for staff.
  + Many staffs can work at a branch.
  + Entity CM\_CASHIER is extended from CM\_STAFF because in application, we classify the right of user.
* CM\_JOB:
  + As mentioned, this entity is to classify the position of the staffs in the system.
  + An entity CM\_JOB has 4 attributes: JOB\_ID, JOB\_TITLE, SALARY\_FOR\_HOUR, FIXED\_SALARY.
  + JOB\_ID is primary key for this entity.
  + SALARY\_FOR\_HOUR is used for employees who work as parttime job.
  + FIXED\_SALARY is used for employees who work as fulltime job.
  + This entity connects with CM\_STAFF by use case: <IS>. A staff can only do one position in the system.
* CM\_CUSTOMER:
  + This entity presents for customer.
  + As usual, a customer will have name, contact number... Therefore, we create this entity with attributes: CUSTOMER\_ID, NAME, ADDRESS, CONTACT\_NUMBER, GENDER.
  + CUSTOMER\_ID is primary key for this entity.
* CM\_ORDER:
  + When a customer wants to place a list of items, we use entity CM\_ORDER for this.
  + This entity will have these attributes: ORDER\_ID, TOTAL, DISTANCE, STATUS, SHIPPING FEE, DISCOUNT, DATE, TIME.
  + ORDER\_ID is primary key of entity CM\_ORDER.
  + TOTAL and SHIPPING FEE are dependent attributes.

TOTAL: the payment of the order, including shipping fee SHIPPING FEE: the fee of the shipping based on the distance.



* + One customer can have many orders.
  + In application:

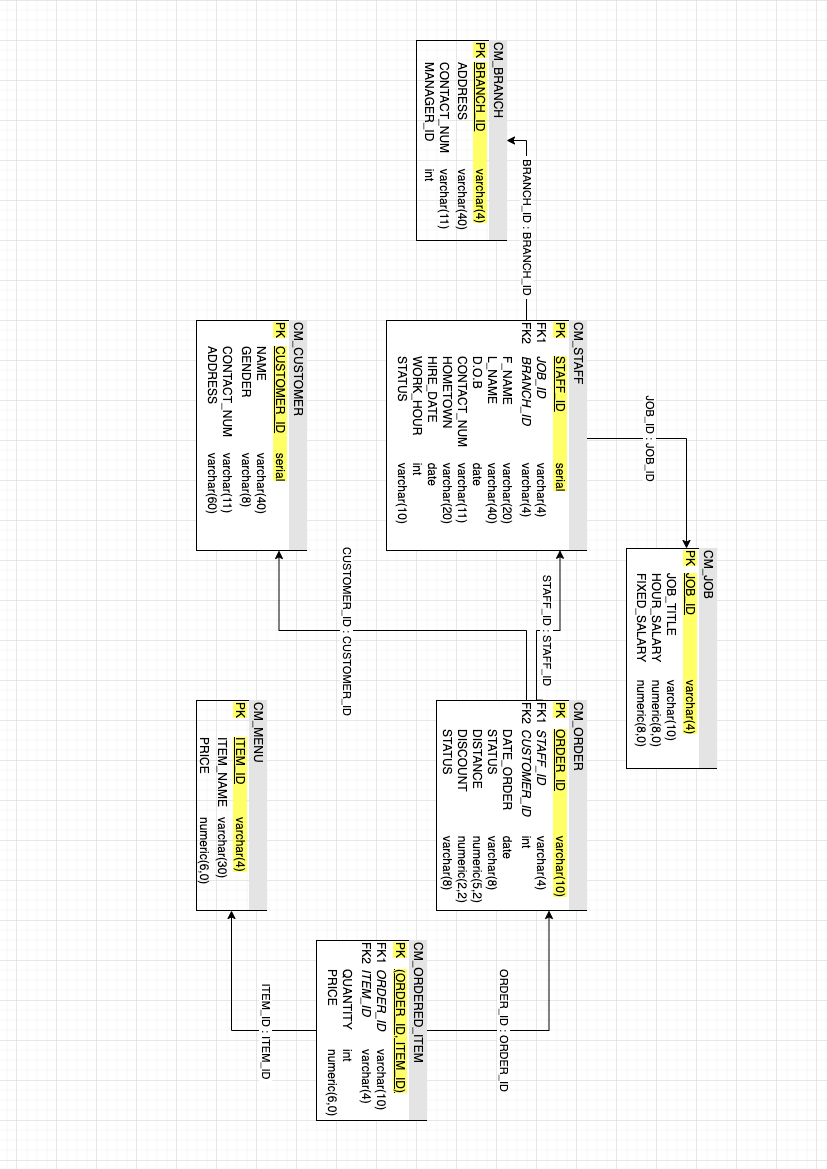


The order can be taken by any employees.

The order can be exported into bill only by cashier.

* CM\_ITEM:
  + Every product sold in the chain will be presented by entity CM\_ITEM.
  + An item will have following attribute: ITEM\_ID, PRICE, ITEM\_NAME
  + Many items can be in many orders, many orders can contain many items. The relation has attribute: QUANTITY. It is used for defining the number of items in an order.

# Relationship diagram



*Relation diagram on Cafe chain management system*

* From the Entity Relationship Diagram that we built in previous, we map all these entities and their relations between them to Relational Diagram.
* First, a staff have only one position in the system, so we decide the relation between Staff and Job as CM\_STAFF 1-1 CM\_JOB.
* A branch in the chain can have many staffs, we define the relation between branch and staff is CM\_BRANCH 1-N CM\_STAFF
* Next, a staff can take as many orders as possible, meanwhile an order can be only taken by a staff and an order is placed by only 1 customer. The relations defined are CM\_STAFF 1-N CM\_ORDER and CM\_CUSTOMER 1-N CM\_ORDER.
* Finally, the order and item are N-M relation. An order can have many items and an item can be contained by many orders. We solve this problem by:
  + Create a class contain all the items sold in the system.
  + Create an additional table as CM\_ORDER\_ITEM.

# Query

# Create table query

* + - We create tables following to the relation schema.
    - In the creating queries, we defined the type of the data input, the constraint between classes.

# Stored procedure

* + - We create stored function for quick insertion into the database.
    - One of the design principles is make common case. The stored procedures below are usually used; therefore, they will be convenient in using when we need it.

# Role

# The role.sql file is a critical component in managing database security, providing a structured and maintainable approach to defining roles, permissions, and access controls within the database. Its existence helps ensure that our database security is both robust and adaptable, meeting the changing needs of the application and organization it supports.

# Index

* + - Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.
    - In our database, the search engine will use most the order id, customer id, and order id. All the attributes above are in the tables with huge data and if we want to get to it, we might have to traverse all the data from that table.
    - Base on principle design, we make common case fast by creating index for those.
      * ORDER\_ID

CREATE INDEX idx\_order ON cm\_order(order\_id);

* + - * CUSTOMER\_ID

CREATE INDEX idx\_customer ON cm\_customer(customer\_id);

* + - * ITEM\_ID

CREATE INDEX idx\_item ON cm\_menu(item\_id);

# Functions

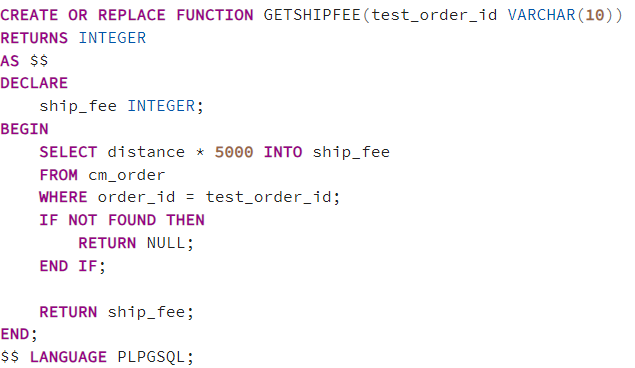
* + - We create functions for most common case for cashiers, managers, CEOs, president... like exporting out the bill, getting the information for evaluating some branches...

# Calculate the shipping fee

* + - * The shipping fee will be calculated by the formula:

Ship fee = Distance \* 5000 (vnd)

* + - * The input of the function will be ORDER\_ID then it will return the shipping fee.
      * The query will traverse all the data from table CM\_ORDER and get the information of distance corresponding to the input id.
      * We can improve performance by using INDEX for ORDER\_ID in searching engine.



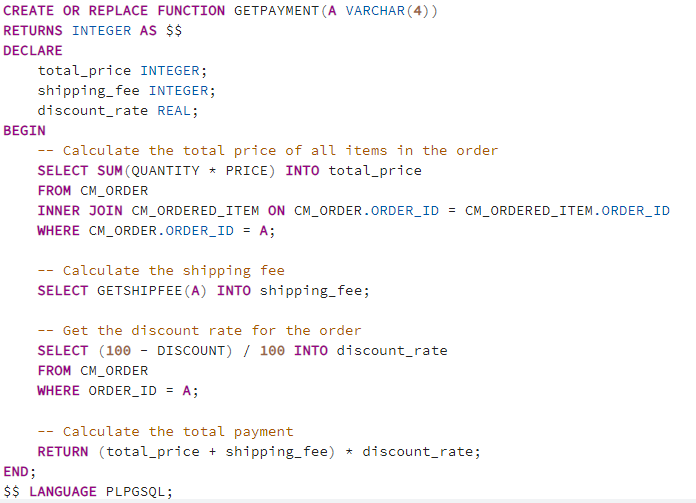
# Get the payment for a bill

* + - * The payment will be the sum of all items in the order list and the shipping fee after subtracting the discount part.

Sum =  (quantity \* price of the item)

Payment = Sum \* (100 - discount) /100

* + - * The query will traverse all the data from table CM\_ORDER joining with table CM\_ORDER\_ITEM for the item list and table CM\_ITEM for the price of the item. Then we get the payment by the formula above.
      * We can improve performance by using INDEX for ORDER\_ID in table CM\_ORDER\_ITEM because the many ORDER\_ID are repeated.



# Find most efficient branch in a chain

* + - * Most efficient branches are defined as the coffee shops which have the highest income in total
      * The income will be calculated by sum of all orders in a branch. To do that, we must join 4 tables: CM\_BRANCH for getting the branch, CM\_ORDER to get the order, CM\_ORDER\_ITEM and CM\_MENU for the value of the item to sum up the income.
      * When we call the function, the function will return the BRANCH with most revenue. If there are multiple equal results, the function return all.

A screenshot of a computer code

Description automatically generated

# Get the daily income

* + - * This function is created to calculate income of all branches in a identified date. It takes data sorted by order\_id from joining 4 tables (cm\_branch, cm\_staff, cm\_order and cm\_order\_item) then choose all record having the date\_ordered equals to parameter d.

A computer screen shot of a code

Description automatically generated

# Get the information of a bill

* + - * This function is created to print a bill of a customer with an identified order ID. It takes data sorted by order\_id from joining 5 tables (cm\_branch, cm\_staff, cm\_order, cm\_order\_items, and cm\_customer) then choose all records having the order\_id equals to parameter id.

**A screenshot of a computer code

Description automatically generated**

# Find loyal customers

* + - * This function is created to find all the loyal customers who has pay more than an identified amount of money. It takes data sorted by total payment from joining 5 tables (cm\_branch, cm\_staff, cm\_order, cm\_order\_items, and cm\_customer) then choose all records having the sum of total payment(= quantity x price of each item) more than parameter p.

A computer code with text

Description automatically generated with medium confidence

# Find most/least favorited product

* + - * The query takes data from CM\_MENU left joining CM\_ORDERED\_ITEM. If any item has not been ordered, QUANTITY column will be null. COALESCE(cm\_ordered\_item.quantity, 0) turns null to 0
      * ORDER BY quantity ASC for least favourite item and DESC for most favourite item

A computer code with text

Description automatically generated

A computer code with text

Description automatically generated with medium confidence

A computer code with text

Description automatically generated

# Calculating the salary

* + - * The query takes data from CM\_JOB joining CM\_STAFF
      * If the staff’s status is “full time”, use FIXED\_SALARY as salary
      * Else if it is “part time”, use SALARY = WORK\_HOUR \* SALARY\_PER\_HOUR
      * Else if the staff is not working anymore, salary is 0

A computer screen shot of a code

Description automatically generated

# Suggest item for a customer

* + - * The query returns the most-ordered item by the customer
      * Select from records of CM\_ORDER by the customer, then join CM\_ORDERED\_ITEM and CM\_ITEM to know quantity, item name

A computer screen shot of a code

Description automatically generated

**4.4.10 Delete and Insert function**

We implemented some delete and insert function to add/remove data from our database.

# Trigger

# Trigger to enforce branch manager existence

# The trigger helps ensure that every branch has a valid manager. It prevents the insertion or updating of a branch with a manager who doesn't exist in the cm\_staff table.

# For operational purposes, every branch typically needs a manager. This trigger enforces that rule at the database level, ensuring that the data reflects real-world business requirements.

# By ensuring that each branch has a valid manager, the trigger improves the overall quality and reliability of the database.

# 

# 5.2. Trigger prevent deleting a job

# If a job is deleted while staff members are still associated with it, those staff records would refer to a non-existent job, creating orphan records. This trigger prevents such a situation by ensuring that jobs are only deleted when no staff members are associated with them.

# It ensures that the database remains consistent with the real-world state of the organization. Job titles and positions usually have organizational meaning and importance. Removing a job title that's in active use would lead to inconsistencies between the database and the actual organizational structure.

# When it's time to clean up or reorganize job records, this trigger ensures that users first reassign or remove associated staff members, leading to a more orderly and intentional data management process.

# A computer screen shot of a program code Description automatically generated

# 

# 5.3. Trigger to Validate New Staff Age

# This trigger is designed to check if the age of a new staff member is at least 18 years old based on the provided date\_of\_birth.

# The trigger function is named validate\_staff\_age.

# The trigger is set to execute this function before inserting a new record into the cm\_staff table for each affected row.

# A computer screen shot of a program code Description automatically generated

# 

# 5.4. Trigger to Automatically Set Discount for Returning Customer

# Creating the apply\_returning\_discount trigger demonstrates how businesses can use database automation to enhance marketing strategies, improve customer relationships, and streamline operations. By leveraging triggers, companies can enforce business rules directly within the database, ensuring consistent application and reducing the need for manual intervention or complex application logic. This approach leads to more efficient operations, better customer service, and potentially increased revenues through heightened customer loyalty.

# 

# 5.5. Trigger to validate taken staff for the order

# By implementing the validate\_taken\_staff trigger, we embed a critical check directly into your database, enhancing data integrity, operational efficiency, and overall system reliability. This proactive approach ensures that our database reflects the actual capabilities and availability of staff, leading to smoother operations and better service.

# View

# View of All Branches and Their Managers

# Displays details about branches and their respective managers.

# 

# View of Menu Prices

# Shows the item\_id, item\_name, and price from the menu.

# A close up of text Description automatically generated

# View of Job Roles and Salaries

# Provides information about job roles, including job\_id, job\_title, salary\_per\_hour, and fixed\_salary.

# A black text on a white background Description automatically generated

# View of Staff Details

# Presents detailed information about staff members, including staff\_id, name, date\_of\_birth, contact\_number, hometown, job\_id, branch\_id, hire\_date, work\_hour, and status.

# 

# View of Customer Details

# Displays customer details, including customer\_id, customer\_name, address, contact\_number, and gender.

# A close up of a text Description automatically generated

# View of Order Summaries

# Aggregates order details, including order\_id, customer\_id, customer\_name, date\_order, status, and total\_order\_value (calculated as the sum of quantity \* price for each order item).

# A computer code on a white background Description automatically generated

# View of Total Sales Per Item

# Shows the total quantity sold for each item based on order items.

# A close-up of a white background Description automatically generated

# View of Daily Sales

# Presents daily sales information, including the order\_date and total\_sales calculated as the sum of quantity \* price for each order item.

# A white background with text Description automatically generated

# View of Staff Per Branch

# Provides a count of staff members per branch.

# A close up of text Description automatically generated

# View of Items Never Ordered

# Lists items from the menu that have never been ordered.

# A white background with black text Description automatically generated

# View of Active Customers

# Displays active customer information, including customer\_id, customer\_name, and the number\_of\_orders (count of orders).

# A close-up of a computer code Description automatically generated

# View of Staff Full-time and Part-time

# Shows staff members who are either full-time or part-time.

# A close up of a computer screen Description automatically generated

# View of Most Recent Orders

# Lists the most recent orders within the last 30 days, including order\_id, date\_order, and status.

# A close-up of a white background Description automatically generated

# View of Average Distance for Online Orders

# Calculates and displays the average distance for online orderA close-up of a computer code Description automatically generated

# View of Customer Order History

# Presents the order history for customers, including customer\_id, customer\_name, order\_id, date\_order, and status, ordered by customer\_id and date\_order in descending order.

# A computer code with text Description automatically generated with medium confidence

# Contribution

* **ERD, Relational Schema, Stored Procedure, Create Table – Nguyễn Văn Dương 20215189.**
* **Insert/Delete Function, Role, Trigger 1,2,3 – Nguyễn Việt Thắng 20215245**
* **Functions, View, Trigger 4 5, Index, Report – Nguyễn Hoàng Việt 20215255**