Database Final Report

Supermarket Management System

Group 2 - ICT

University of Science and Technology of Hanoi

December, 2022

List of members in group 2:

- Mai Hải Đăng - BI12-076
- Nguyễn Duy An - BI12-006
- Đoàn Đình Đăng - BI12-074
- Trần Ngọc Ánh - BI12-040
- Trần Hải Đăng - BI12-073

Table of Contents

1. Introduction

Trading commodities is fundamental and essential in our society. It occurs everyday, at places like the supermarket. Most types of food and other goods needed in our homes are sold at the local supermarket, so the amount of data is vast and complex. As a result, the problem in administering it is extremely important.

In this project, our group has developed a simple database design for a supermarket. This database system is designed to manage products and trading activities at the supermarket. The administrator can manage all the related data, such as price, brand, or type of product, customers' information, their feedback on every single product, the information of employees working at the supermarket, invoice details, return slips, etc. On the other hand, specific product management contains every related content of the product customers are looking for; therefore, they can search for and buy goods effectively.

2. User requirements

To create an efficient database, the first and also the most important step is to collect user requirements. For a supermarket management system, there are two main types: the administrator and the customers.

2.1 Administrator

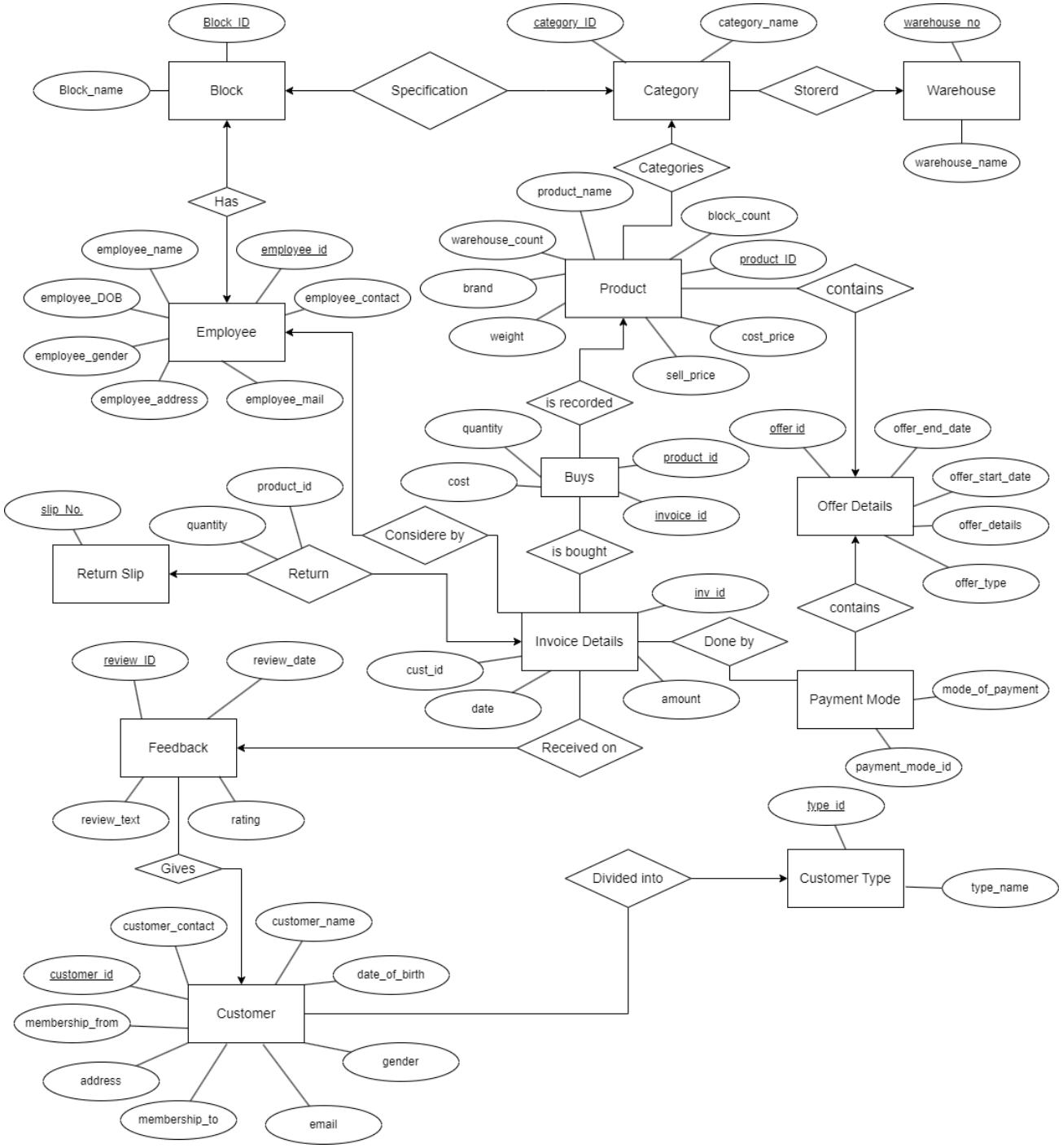
- Administrators can upload new products and their related information to the website, including selling price, cost price, brand, type, and weight.
- Administrators can administer the number of products in stock.
- Administrators can see the data of employee information, invoice details, return slips, and customer feedback.
- Administrators can classify customer types (Silver, Gold, Platinum, Premium).

2.2 Customers

- Customers can change their personal information, like changing their emails, addresses, or phone numbers.
- Customers can see their orders and choose the payment mode.
- Customers can purchase orders online or go shopping in person.
- Customers can give feedback about products they bought and the services at the supermarket.
- Customers can ask for a return slip.

3. Entity Relationship Diagram

3.1 Diagram



For ease of viewing please use this link: [Supermarket Management System - ERD]

3.2 Entity Relationship

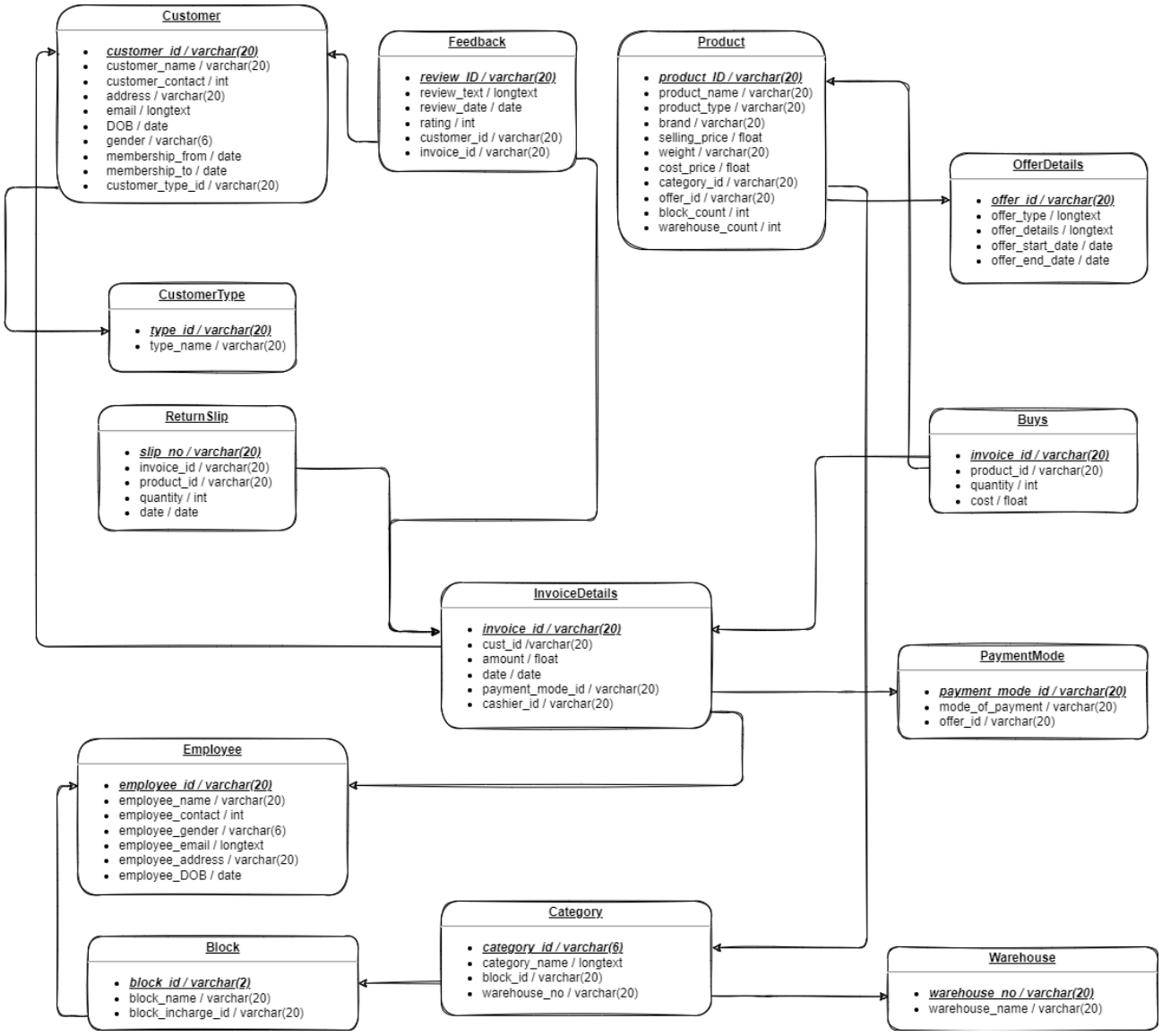
- **Block - Category** (one to one): In the supermarket system, each block is equivalent to a category and vice versa.
- **Category - Warehouse** (many to one): A warehouse used to store many categories of products.
- **Product - Category** (many to one): A product only has one specific category. But a category displays all products.
- **Block - Employee** (one to one): Each block is managed by an employee.
- **Invoice Details - Employee** (many to one): The invoice details will be produced by the employee for every payment.
- **Invoice Details - Payment Mode** (many to one): Many different payment modes are used to pay for the invoice. But the invoice can only be paid by a specific payment mode.
- **Payment mode - Offer Details** (many to one): Customers can apply the offers from the supermarket when making the payment. But an offer can only be used once.
- **Product - Category** (many to one): Many products with the same usages or properties will be considered as one category. But the category represents all the products.
- **Product - Buys** (many to one): The supermarket system will record every product purchased.
- **Invoice Details - Buys** (one to one): Every bought product will have its own invoice details and be recorded by the supermarket's system.
- **Product - Offer Details** (many to one): An offer can be applied for several products. But once an offer is used, it is no longer available.
- **Invoice Details - Return Slip** (one to one): For every created invoice, the customers may request to return the products. Each return will be written in the invoice.
- **Invoice Details - Feedback** (one to one): Per invoice, customers can give feedback, and each invoice can receive feedback for each time making the payment.
- **Feedback - Customer** (many to one): Customers can give several feedbacks.
- **Customer - Customer Type** (many to one): Many types of customers frequent the supermarket.

4. Functional Dependencies

1. **Product**
   - product_ID -> {brand, product_name, sell_price, weight, cost_price, category_id, offer_id, block_count, warehouse_count}
   - Primary Key: product_ID
   - Foreign Key: Product(category_id) references Category(category_ID)
     Product(offer_id) references OfferDetails(offer_id)
2. **Customer**
   - customer_id -> {customer_name, email, address, gender, customer_contact, membership_to, membership_from, customer_type_id, DOB}
   - Primary Key: customer_id
   - Foreign Key: Customer(customer_type) references CustomerType(type_id)
3. **Employee**
   - employee_id -> {employee_name, employee_email, employee_address, employee_gender, employee_contact, employee_DOB}
   - Primary Key: employee_id
4. **InvoiceDetails**
   - invoice_id -> {cust_id, amount, date, payment_mode, cashier_id}
   - Primary Key: invoice_id
   - Foreign Keys: InvoiceDetails(cust_id) references Customer(customer_id), InvoiceDetails(cashier_id) references Employee(employee_id), InvoiceDetails(payment_mode) references PaymentMode(payment_mode_id)
5. **PaymentMode**
   - payment_mode_id -> {mode_of_payment, offered_id}
   - Primary Key: payment_mode_id
   - Foreign Key: PaymentMode(offered_id) references OfferDetails(offer_id)
6. **OfferDetails**
   - offer_id - > {offer_end_date, offer_start_date, offer_details, offer_type}
   - Primary Key : offer_id
7. **Block**
   - block_id -> {block_name, block_incharge_id}
   - Primary Key: block_id
   - Foreign Key: Block(block_incharge_id) references Employee(employee_id)

8. **Feedback**
   - review_ID -> {review_date, review_text, rating, customer_id, invoice_id}
   - Primary Key: review_ID
   - Foreign Key: Feedback(customer_id) references Customer(customer_id),
     Feedback(invoice_id) references InvoiceDetails(inv_id),
9. **Buys**
   - {product_id, invoice_id} -> {quantity, cost}
   - Primary Key: product_id, invoice_id
   - Foreign Key: Buys(product_id) references Product(product_ID),
     Buys(invoice_id) references InvoiceDetails(inv_id)
10. **Warehouse**
   - warehouse_no -> {warehouse_name}
   - Primary Key: warehouse_no
11. **CustomerType**
   - type_id -> {type_name}
   - Primary key: type_id
12. **ReturnSlip**
   - slip_No -> {invoice_id, product_id, quantity, date}
   - Primary Key: slip_No
   - Foreign Key: ReturnSlip(product_id) references Product(product_ID),
     ReturnSlip(invoice_id) references InvoiceDetails(inv_id)
13. **Category**
   - category_ID -> {category_name, block_id, warehouse_no}
   - Primary Key: category_id
   - Foreign Key: Category(block_id) references Block(block_id)
     Category(warehouse_no) references Warehouse(warehouse_no)

5. Schema



For ease of viewing please use this link: [Supermarket Management System - Schema]

Descriptions:

Employee	Details of employees (cashier and store incharge)

Product	Details of products available
Category	Details of categories of different products
OfferDetails	Contains offers details (discount …)
Warehouse	Details of warehouse
Block	Store shop details (grocery, watches, …)
Customer	Records of customers
CustomerType	Types of customers
PaymentMode	Details of various modes of payment
Feedback	Records of feedbacks
InvoiceDetails	Records of invoice details
ReturnSlip	Records of products returned
Buys	Records of which invoices correlate to which products

6. Implementations

6.1 Structure

6.1.0 Create Database

```
DROP SCHEMA IF EXISTS supermarket;
CREATE SCHEMA IF NOT EXISTS `supermarket`;
USE `supermarket`;
```

6.1.1 Create Warehouse Table

```
CREATE TABLE IF NOT EXISTS Warehouse(
    warehouse_no varchar(20) not null,
    warehouse_name varchar(20) null,
    PRIMARY KEY (warehouse_no)
);
```

6.1.2 Create Employee Table

```
CREATE TABLE IF NOT EXISTS Employee(
    employee_id varchar(20) not null,
    employee_name varchar(20) null,
    employee_contact int null,
    employee_DOB date null,
    employee_email longtext null,
    employee_address varchar(20) null,
    employee_gender varchar(6) null,
    PRIMARY KEY (employee_id)
);
```

6.1.3 Create Block Table

```
CREATE TABLE IF NOT EXISTS Block(
    block_id varchar(20) not null,
    block_name varchar(20) null,
    block_incharge_id varchar(20) null,
    PRIMARY KEY (block_id),
    FOREIGN KEY (block_incharge_id) REFERENCES Employee(employee_id)
);
```

6.1.4 Create Category Table

```
CREATE TABLE IF NOT EXISTS Category(
    category_id varchar(20) not null,
    category_name longtext null,
    block_id varchar(20) null,
    warehouse_no varchar(20) null,
    PRIMARY KEY (category_id),
    FOREIGN KEY (block_id) REFERENCES Block(block_id),
    FOREIGN KEY (warehouse_no) REFERENCES Warehouse(warehouse_no)
);
```

6.1.5 Create OfferDetails Table

```
CREATE TABLE IF NOT EXISTS OfferDetails(
    offer_id varchar(20) not null,
    offer_details LONGTEXT null,
    offer_type longtext null,
    offer_start_date date null,
    offer_end_date date null,
    PRIMARY KEY (offer_id)
);
```

6.1.6 Create CustomerType Table

```
CREATE TABLE IF NOT EXISTS CustomerType(
    type_id varchar(20) not null,
    type_name varchar(20) null,
    PRIMARY KEY (type_id)
);
```

6.1.7 Create Product Table

```
CREATE TABLE IF NOT EXISTS Product(
    product_id varchar(20) not null,
    product_name varchar(20) null,
    brand varchar(20) null,
    cost_price float null,
    weight varchar(20) null,
    selling_price float null,
    category_id varchar(20) null,
    offer_id varchar(20) null,
    block_count int null,
    warehouse_count int null,
    PRIMARY KEY (product_id),
    FOREIGN KEY (category_id) REFERENCES Category(category_id),
    FOREIGN KEY (offer_id) REFERENCES OfferDetails(offer_id)
);
```

6.1.8 Create Customer Table

```
CREATE TABLE IF NOT EXISTS Customer(
    customer_id varchar(20) not null,
    customer_name varchar(20) null,
    customer_contact int null,
    DOB date null,
    email longtext null,
    address varchar(20) null,
    gender varchar(6) null check (gender = 'female' or gender = 'male'),
    customer_type_id varchar(20) null,
    membership_to date null,
    membership_from date null,
    PRIMARY KEY (customer_id),
    FOREIGN KEY (customer_type_id) REFERENCES CustomerType(type_id)
);
```

6.1.9 Create PaymentMode Table

```
CREATE TABLE IF NOT EXISTS PaymentMode(
    payment_mode_id varchar(20) not null,
    mode_of_payment varchar(20) null,
    offer_id varchar(20) null,
    PRIMARY KEY (payment_mode_id),
    FOREIGN KEY (offer_id) REFERENCES OfferDetails(offer_id)
);
```

6.1.10 Create InvoiceDetails Table

```
CREATE TABLE IF NOT EXISTS InvoiceDetails(
    invoice_id varchar(20) not null,
    customer_id varchar(20) null,
    amount float null,
    date date null,
    payment_mode varchar(20) null,
    cashier_id varchar(20) null,
    PRIMARY KEY (invoice_id),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY (cashier_id) REFERENCES Employee(employee_id),
    FOREIGN KEY (payment_mode) REFERENCES PaymentMode(payment_mode_id)
);
```

6.1.11 Create Feedback Table

```
CREATE TABLE IF NOT EXISTS Feedback(
    review_ID varchar(20) not null ,
    review_text LONGTEXT null,
    rating int null,
    review_date date null,
    customer_id varchar(20) null,
    invoice_id varchar(20) null,
    PRIMARY KEY (review_ID),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY (invoice_id) REFERENCES InvoiceDetails(invoice_id)
);
```

6.1.12 Create Buys Table

```
CREATE TABLE IF NOT EXISTS Buys(
    product_id varchar(20) not null,
    invoice_id varchar(20) not null,
    quantity int null,
    cost float null,
    PRIMARY KEY (invoice_id, product_id),
    FOREIGN KEY (product_id) REFERENCES Product(product_id),
    FOREIGN KEY (invoice_id) REFERENCES InvoiceDetails(invoice_id)
);
```

6.1.13 Create ReturnSlip Table

```
CREATE TABLE IF NOT EXISTS ReturnSlip(
    slip_No varchar(20) not null,
    invoice_id varchar(20) null,
    product_id varchar(20) null,
    quantity int null,
    return_date date null,
    PRIMARY KEY (slip_No),
    FOREIGN KEY (product_id) REFERENCES Product(product_id),
    FOREIGN KEY (invoice_id) REFERENCES InvoiceDetails(invoice_id)
);
```

6.2 Data

(In attached file "Supermarket-data.sql")
Or Drive: Supermarket-data.sql

6.3 Statements (Applications)

6.3.1 Basic Statements

1. Create a procedure to Create, Delete product

```
-- Procedure to add a new product
DELIMITER $$
CREATE PROCEDURE add_product
(IN id varchar(20),
IN name varchar(20),
IN brand varchar(20),
IN cost_price float,
IN weight varchar(20),
IN selling_price float,
IN category_id varchar(20),
IN offer_id varchar(20),
IN block_count int,
IN warehouse_count int)
BEGIN
Insert into Product values(id, name, brand, cost_price, weight, selling_price,
category_id, offer_id, block_count, warehouse_count);
END$$

DELIMITER ;
```

```
-- Procedure to delete a product
DELIMITER $$
CREATE PROCEDURE delete_product
(IN product_id varchar(20))
BEGIN
Delete from Product where product_id = product_id;
END$$

DELIMITER ;
```

- Example:

```
CALL add_product('PI14584', 'Dried Jackfruit', 'Amul', '20', '0.2kg', '30',
'GRO003','OFF10', '10', '100');
```

```
CALL delete_product('PI14584');
```

2. Create a procedure to Create, Delete category

```
-- Procedure to add a new category
DELIMITER $$
CREATE PROCEDURE add_category
(IN id varchar(20),
IN name longtext,
IN block_id varchar(20),
IN warehouse_no varchar(20))
BEGIN
Insert into Category values(id, name, block_id, warehouse_no);
END$$

DELIMITER ;
```

```
-- Procedure to delete a category
DELIMITER $$
CREATE PROCEDURE delete_category
(IN id varchar(20))
BEGIN
Delete from Category where category_id = id;
END$$
```

```
DELIMITER ;
```

- Example:

```
CALL add_category('VHC001', 'Bikes', 'C1', 'WARC001');
```

```
CALL delete_category('C1');
```

3. Create a procedure to Create, Delete employee

```
-- Procedure to add new employee
DELIMITER $$
CREATE PROCEDURE add_employee
(IN id varchar(20),
IN name varchar(20),
IN contact int,
IN DOB date,
IN email longtext,
IN address varchar(20),
IN gender varchar(6))
BEGIN
Insert into Employee values(id, name, contact, DOB, email, address, gender);
END$$

DELIMITER ;
```

```
-- Procedure to delete an employee
DELIMITER $$
CREATE PROCEDURE delete_employee
(IN id varchar(20))
BEGIN
Delete from Employee where employee_id = id;
END$$

DELIMITER ;
```

- Example:

```
CALL add_employee("EMP2007", "John Cena", "065445", "1977-4-23",
"BingChilling@gmail.com", "123 Beijing", "male");
```

```
CALL delete_employee("EMP2007");
```

4. Create a procedure create new invoice

```
-- Procedure to add invoice
DELIMITER $$
CREATE PROCEDURE add_invoice
(IN id varchar(20),
IN customer_id varchar(20),
IN amount varchar(20),
IN invoice_date date,
IN payment_mode varchar(20),
IN cashier_id varchar(20))
BEGIN
INSERT INTO InvoiceDetails VALUES(id, customer_id, amount, invoice_date,
payment_mode, cashier_id);
END$$

DELIMITER ;
```

- Example

```
CALL add_invoice('AAP003', 'CTM00021', '10', '2022-12-1', 'PAY003', 'EMP2002');
```

5. Create a procedure for customer to give feedback

```
-- Procedure to add new feedback
DELIMITER $$
CREATE PROCEDURE add_feedback(IN id VARCHAR(20),
IN rtext LONGTEXT,
IN rating INT,
IN rdate date,
IN customer_id VARCHAR(20),
IN invoice_id VARCHAR(20))
BEGIN
INSERT INTO Feedback VALUES (id, rtext, rating, rdate, customer_id, invoice_id);
END$$

DELIMITER ;
```

- Example

```
CALL add_feedback('RV005', 'Good', 5, '2022-10-12', 'CTM00005', 'AAK003');
```

6.3.2 Advance Statements

1. List the employee name and id in charge of the grocery block.

```
SELECT
      employee_name,
      employee_id
FROM Employee, Block WHERE Block.block_incharge_id = Employee.employee_id AND
Block.block_name = 'Grocery';
```

employee_name	employee_id
Công Huy	EMP1001

2. List product id and product name with the highest bought amount.

```
SELECT new.product_id, new.product_name, total AS bought
FROM(
      SELECT P.product_id, P.product_name, sum(B.quantity) AS total
      FROM Buys B
      INNER JOIN Product P
      ON B.product_id = P.product_id
    GROUP BY P.product_id) AS new
ORDER BY total DESC limit 1;
```

product_id	product_name	bought
PI10304	Analogue	9

3. List the product id, product name, warehouse count, and block count of products that were returned and whose block count is 30 % more than their warehouse count.

```
SELECT
    p.product_id,
    p.product_name,
    p.warehouse_count,
    p.block_count
FROM returnslip AS r JOIN Product AS p ON (r.product_id=p.product_id) WHERE
block_count > 0.3 * warehouse_count;
```

product_id	product_name	warehouse_count	block_count
PI10532	presssure cooker	10	20
PI10203	Casual Shoes	100	50
PI10041	soyabean oil	100	100
PI10112	half sleeve	100	350
PI10112	half sleeve	100	350

4. List the number of Banana left inside the block and warehouse

```
SELECT block_count, warehouse_count
FROM Product WHERE product_name = 'Banana';
```

block_count	warehouse_count
200	100

5. Sort the number of products left in stock (including those inside a block and the warehouse) for all categories from highest to lowest quantity.

```
SELECT new.category_id, new.category_name, new.total_stock
FROM(
```

```
    SELECT P.category_id, C.category_name, sum(P.block_count) +
sum(P.warehouse_count) as total_stock
    FROM Product P
    INNER JOIN Category C
    ON P.category_id = C.category_id
    GROUP BY P.category_id) new
ORDER BY new.total_stock DESC;
```

category_id	category_name	total_stock
GRO004	Rice & Pulses	2150
GRO001	Fruits & Vegetables	1900
DNG001	TableWare	1425
DN002	Coffee, Tea & Beverages	1310
GRO002	Spices	1150
GRO003	Dry Fruits	1150
DN001	Soaps & Detergents	1020
FW002	Women Footwear	960
DN003	Fragrances	950
WF002	Women Jeans	950
MF002	Men Trousers	950
MF001	Men Shirts	950
FW001	Men Footwear	940
GRO005	Oils	860
DNG002	Storage	847
CLK001	Men Watches	840
WF004	Lingerie	760
KW003	Kids Ethnic Wear	660
CLK002	Women Watches	650
STN001	Office Supplies	650
STN002	Notebooks, Writing Pads	620
STN003	Art & Craft Supplies	620
STN004	Pens, Pencils & Writing S	620
MF003	Men Innerwear	610
WF001	Women Tops	610
KW001	Tops & Tees	540
CLK003	Kids Watches	520
KW002	Baby Jeans	520
FW003	Kids Footwear	510
DNG003	Kitchen Tools	177
DNG004	Cooking Essentials	177
ELC005	Speaker , Woofer & MP3	125
ELC004	LED Television	125
ELC003	Laptop	125
ELC002	Tablets	125
ELC001	Smartphones	125

6. Calculate the total profit

```
SELECT sum(B.quantity*(P.selling_price - P.cost_price)) as profit FROM Buys B
INNER JOIN Product P
```

```
ON B.product_id = P.product_id;
```

profit
23635

7. List the customer id and name of all customer with Silver membership

```
SELECT C.customer_id, C.customer_name
FROM CustomerType CT INNER JOIN Customer C ON C.customer_type_id = CT.type_id
WHERE CT.type_name = "Silver";
```

customer_id	customer_name
CTM00010	Công Anh
CTM00011	Ái Vân
CTM00015	Vị Lan
CTM00017	Ái Vỹ
CTM00018	Hoài Vân
CTM00020	Yên Hải
CTM00027	Thảo Vân
CTM00030	Yên Yến
CTM00032	Khánh Minh
CTM00033	Mạnh An
CTM00037	Như Khánh
CTM00038	Hoài Vị
CTM00039	Thuỷ Vân
CTM00040	Yến Công
CTM00043	Hoàng Xuân
CTM00044	Vân Tuấn
CTM00048	Yến Tuấn

8. List the offer_id, offer_details and offer_type of all the offers that has the end date of '2022-03-31'

```
SELECT offer_id, offer_details, offer_type
```

```
FROM offerdetails WHERE offer_end_date = '2022-03-31';
```

offer_id	offer_details	offer_type
CASH10	Cash payment offer	10% off

9. List the invoice id, the amount of money paid, and the date of all the Invoice Details after the date '2022-11-30'

```
SELECT invoice_id, amount, date
FROM invoicedetails WHERE date > '2022-11-30';
```

invoice_id	amount	date
AAE002	300.88	2022-12-30

10. Find the most profitable product sold.

```
SELECT profit, product_id, brand
FROM (SELECT DISTINCT product_id, brand, (selling_price-cost_price) as profit
FROM buys NATURAL join product) AS A1
NATURAL JOIN (SELECT MAX(most_profit) AS profit
FROM (SELECT DISTINCT product_id,brand,(selling_price-cost_price) AS most_profit
FROM buys NATURAL JOIN product) AS r2) AS A2
WHERE A2.profit=A1.profit;
```

profit	product_id	brand
3490	PI10433	samsung

11. List the offer id and quantity of products under that offer that was returned.

```
SELECT number_of_quantity, offer_id
```

```
FROM (
SELECT offer_id ,sum(quantity) AS number_of_quantity
    FROM returnslip
    NATURAL JOIN product
    GROUP BY offer_id) AS A2
NATURAL JOIN (
      SELECT max(sum) AS number_of_quantity
     FROM (
         SELECT sum(quantity) AS sum, offer_id
        FROM returnslip
        NATURAL JOIN product
        GROUP BY offer_id)
      AS A1)
AS A3 WHERE A3.number_of_quantity=A2.number_of_quantity;
```

number_of_quantity	offer_id
5	OFF10