Database Systems Final Project

Inventory Management System

Prepared By:

Christensen Mario Frans - 2301963316
Jeconiah Richard - 231947905
Sunny Jovita - 2301939046
Muhammad Lukman Ismail Hanafi - 2301929493

Odd Semester 2020-2021
Computer Science of Binus international University

Table Of Contents

1. Background

Every single business has their inventory management system, either the old-fashioned one or the modern one. Both serve the exact same purpose, to keep track the available products in their storage. As we are moving forward in time, we believe that efficiency is important, even in inventory management systems.

Some businesses keep track of their inventory by using paperwork or microsoft excel, which is fine. However, we notice that by using these methods, human errors are prone to happen which leads to miscalculation between the number written in the system and in the actual inventory. By this, we came up with an idea of an inventory management system by using the Java programming language that is connected to MySQL.

2. Problem

Our topic that we chose for this Database Systems Final Project is Inventory Control Management System. The reason behind why we chose this topic is that upon our group member's experience and knowledge, many firms, especially new and smaller businesses, have trouble managing the accounting of their own products when running from day to day. This includes the calculation of their stocks (restocking and sales), removing/returning defective products, and even managing the available catalog. This leads to the inaccurate warehouse stocking reports, which may further lead to inaccurate financial reports, or even bad customer service (customers may be promised that their desired products are in stock but rather it's out of stock or doesn't fulfill the required quantity), as well as bad reputation to the company. Moreover, this becomes an even more sophisticated problem for businesses running on abundance of sales every single day.

3. Target Market

Our program is mainly made for wholesalers, supermarkets or other kinds of stores. Its ability to input and store multiple products, staff and multiple store branches make the program applicable for all of them. Not to mention, it can be used also for small businesses or any businesses that require inventory management systems.

4. Member's Role

For this project, our group had agreed to appoint Sunny as our project manager. She is the main coder in our group, where we always brainstormed altogether via share-screen video calls while she was coding on her computer. (The reason why we came up with this plan was because working on a database project online causes alot of unprecedented issues; according to Mario, Jeco, and Lukman, we have much trouble in accessing, managing, coding, and even testing the database program through our own computers. The situation may be otherwise if we were to work on this project offline, as we can reach out to hands-on help from others.) Since the program is also working perfectly on her computer, she also records the running program for the final project video. Altogether, we brainstormed and compiled our knowledge to create the final report together.

For the other members, Mario, Jeco, and Lukman, aside from brainstorming through video call for the program, our tasks are based on:

- Mario: coding java (share screen vid call), making final report, designing ERD and its relations, project idea & background & solution
- Jeco: coding java (share screen vid call), making final report, designing ERD and its relations, create & design presentations (proposal)
- Lukman: coding java (share screen vid call), create query for inventory system (save, delete, update, modify, add), making final report

As seen in the bulleted points above, most of the tasks are overlapping for each member, this is because we are mostly in charge of the assigned task.

5. Database Design

Section 5.1 Identify Entity Types

The purpose of this part is to help us identify the required entity types, from attributes or properties, and values associated with another table.

Table 1.1 Identify entity types

Entity Name	***Description***	***Aliases***	***Occurrence***
Customer	General term describing people who make transactions and already have a membership.	Customer	Each customer has his/her information, such as their name, phone number, address, and their payment method.
PaymentMethod	General term describing the number of ways in which merchants can collect payment from customers	Payment Method	Payment method can only be selected in the customer table, and each payment method has a unique id.
Product	General term describing all products that are offered for sale and still available or in stock.	Product	Each product has its own name, quantity, and price. The quantity will be modified automatically if there are changes in restock and transaction table.

Restock - RestockHeader - RestockDetails	General term describing all restock information.	Restock	Each restock has its arrival date, it can be viewed by all staff, but can only be edited by the manager.
Staff	General term describing all staff employed by the store/branch.	Employee	Each staff works at one particular store/branch.
Store	General term describing all stores which use this database system.	Store/Branch	Each store has its own unique id, name, and location.
Transaction - TransactionHeader - TransactionDetails	General term describing all successful transactions.	Transaction	Each transaction can has a membership customer or an anonymous customer, it is managed by one member of staff. Every transaction has a date, a transaction can be viewed by all employees, but can only be updated and deleted by the manager.
Vendor	General term describing all vendors that supply the products	Vendor	Each vendor can supply any products and it is managed by employees.

Section 5.2 Identify Relationship Types

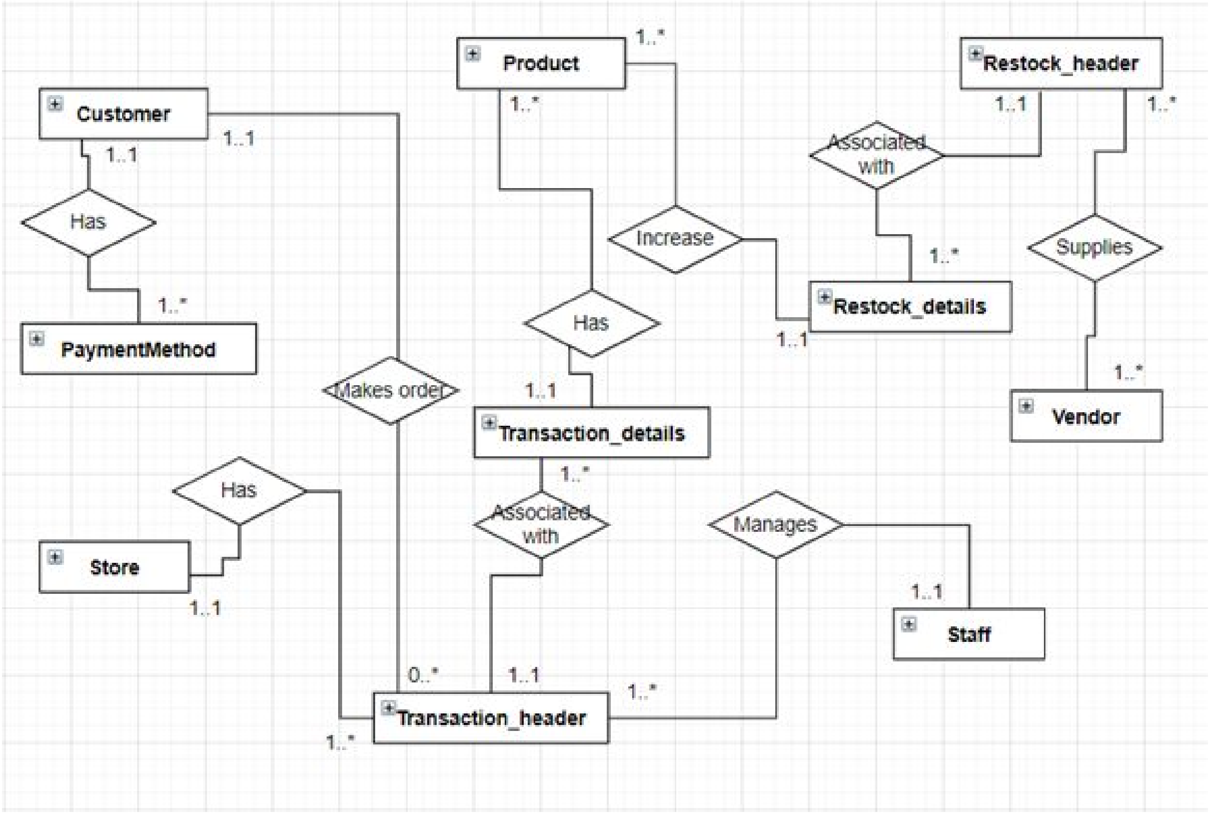
The purpose of this section is to identify the important relationships that exists between the entity types.

Table 1.2 Identify relationship types

Entity Name	***Multiplicity***	***Relationship***	***Entity Name***	***Multiplicity***
Customer	1..1	Has	PaymentMethod	1..*
Customer	1..1	Makes order	Transaction_header	0..*
Store	1..1	Has	Transaction_header	1..*

Vendor	1..*	Supplies	Restock_header	1..*
Restock_details	1..1	Increase	Product	1..*
Transaction_details	1..1	Has	Product	1..*
Staff	1..1	Manages	Transaction_header	1..*
Restock_header	1..1	Associated with	Restock_details	1..*
Transaction_header	1..1	Associated with	Transaction_details	1..*

Figure 1.2 ER diagram showing entity types and relationships without primary key and attributes



Section 5.3 Identify and associate attributes with entity or relationship types

The purpose of this section is to identify the attributes, data type, and length which are used by each of entity.

Table 1.3 Identify attributes, data types, etc

Entity Name	***Attributes***	***Description***	***Data Type & Length***	***Nulls***
Customer	**CustomerID (PK)**	Uniquely identifies customer	Int(20)	No
	CustomerName	Name of customer	Varchar(25)	No
	PhoneNo	Phone number of customer	Int(20)	No
	Address	Address of customer	Varchar(200)	Yes
	PaymentMethodID (FK)	Uniquely identifies payment method	Int(11)	No
PaymentMethod	**PaymentMethodID (PK)**	Uniquely identifies payment method	Int(11)	No
	PaymentMethod	Name of payment method	Varchar(25)	No
Product	**ProductID (PK)**	Uniquely identifies product	Int(11)	No
	ProductName	Name of product	Varchar(64)	No
	Qty	Quantity of product	Int(11)	No
	Price	Price of product	Decimal(10,0)	No
RestockHeader	**RestockID (PK)**	Uniquely identifies restock	Int(11)	No
	ArrivalDate	Date of restock	Date	No
	VendorID (FK)	Uniquely identifies vendor	Int(11)	No
RestockDetails	**RestockID (PK, FK)**	Uniquely identifies restock	Int(11)	No
	ProductID (PK, FK)	Uniquely identifies product	Int(11)	No
	Qty	Quantity of restock product	Int(11)	No
Staff	**StaffID (PK)**	Uniquely identifies staff	Int(11)	No
	FirstName	First name of staff	Varchar(64)	No

	LastName	Last name of staff	Varchar(64)	No
	Position	Position of staff	Varchar(25)	No
	Email	Email of staff	Varchar(100)	No
	Password	Password of staff	Varchar(50)	No
	Gender	Gender of staff	Varchar(25)	No
Store	**StoreID (PK)**	Uniquely identifies store	Int(11)	No
	StoreName	Name of store	Varchar(64)	No
	Address	Address of store	Varchar(225)	No
TransactionHeader	**OrderID (PK)**	Uniquely identifies transaction	Int(11)	No
	Date	Date of transaction	Date	No
	StaffID (FK)	Uniquely identifies staff	Int(11)	No
	StoreID (FK)	Uniquely identifies store	Int(11)	No
	CustomerID (FK)	Uniquely identifies customer	Int(11)	Yes
TransactionDetails	**OrderID (PK,FK)**	Uniquely identifies transaction	Int(11)	No
	ProductID (PK,FK)	Uniquely identifies product	Int(11)	No
	Qty	Quantity of transaction product	Int(11)	No
	GrandTotal	Grand total of transaction product	Decimal(10,0)	No
Vendor	**VendorID (PK)**	Uniquely identifies vendor	Int(11)	No
	VendorName	Name of vendor	Varchar(64)	No
	VendorAddress	Address of vendor	Varchar(225)	Yes
	VendorNo	Phone number of vendor	Int(11)	No

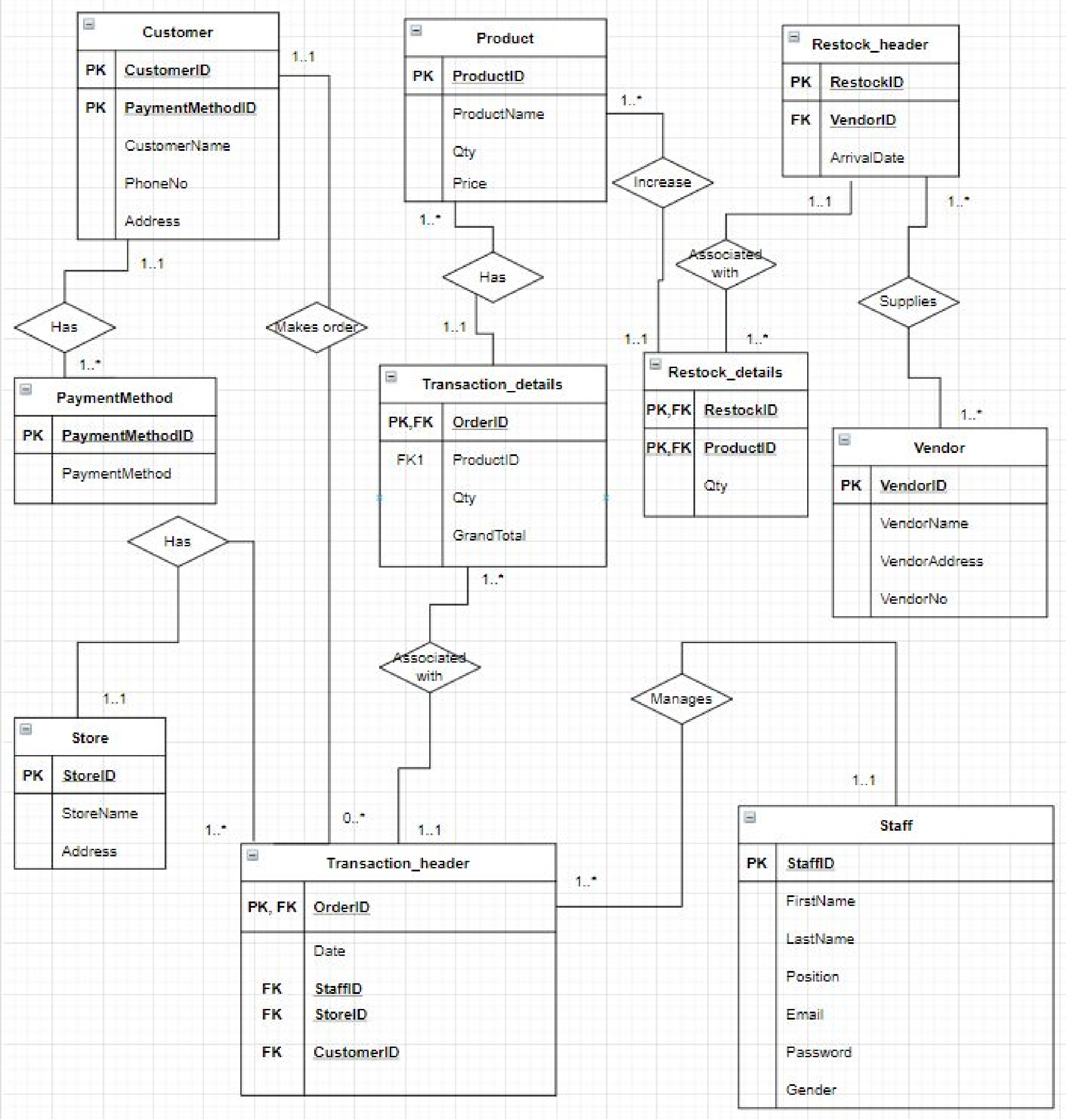
Section 5.4 Determine candidate, primary, and alternate key attributes

The purpose of this section is to identify the candidate keys, primary key, alternate keys for each entity.

Table 1.4 Candidate, Primary, and Alternate key

Entity	Candidate Key	Primary Key	Alternate Key
Customer	CustomerID CustomerName PhoneNo Address	CustomerID	CustomerName PhoneNo Address
PaymentMethod	PaymentMethodID PaymentMethod	PaymentMethodID	PaymentMethod
Product	ProductID ProductName	ProductID	ProductName
RestockHeader	RestockID	RestockID	-
RestockDetails	RestockID ProductID	**Composite key** RestockID ProductID	-
Staff	StaffID FirstName LastName Email Password	StaffID	FirstName LastName Email Password
Store	StoreID StoreName Address	StoreID	StoreName Address
TransactionHeader	OrderID Date StaffID StoreID CustomerID	OrderID	StaffID StoreID CustomerID
TransactionDetails	OrderID ProductID	**Composite key** OrderID ProductID	-
Vendor	VendorID VendorName VendorAddress VendorNo	VendorID	VendorName VendorAddress VendorNo

Figure 1.4 Full ER diagram showing entity types and relationships



These ER diagrams contain different symbols that use rectangles to represent the entity, and diamond shapes to represent relationships. There are 10 different entities with different attributes and relationships in it. These ER diagrams help to explain the logical structure of our Inventory System databases.

Section 5.5 Derive relations for logical data model

1. Strong entity types

Strong entity is an entity whose existence does not depend on the existence of other entities in a schema. The purpose of this section is to show strong entities in the data model, including all the attributes of the entity.

a. **Product** (ProductID, ProductName, Qty, Price)
   Primary key (ProductID)
b. **Store** (StoreID, StoreName, Address)
   Primary key (StoreID)
c. **Vendor** (VendorID, VendorName, VendorAddress, VendorNo)
   Primary key (VendorID)
d. **Staff** (StaffID, FirstName, LastName, Email, Password, Gender, Position)
   Primary key (StaffID)
e. **PaymentMethod** (PaymentMethodID, PaymentMethod)
   Primary key (PaymentMethodID)

2. Weak entity types

A weak entity is an entity that cannot be uniquely identified by its attributes alone, it must use a foreign key in conjunction with its attributes to create a primary key. Weak entity depends on a strong entity to ensure its existence. The purpose of this section is to show weak entities in the data model, including all the attributes.

a. **Customer** (CustomerID, CustomerName, PhoneNo, Address, PaymentMethodID)
   Primary key (CustomerID)
   Customer is a weak entity because it depends on other entity (paymentmethod entity). Therefore, the foreign key in the customer entity is PaymentMethodID which is typically a primary key of paymentmethod entity.

b. **Restock_details** (RestockID, ProductID, Qty)
   Primary key = Composite key (RestockID, ProductID)
   In restock_details entity, it has 2 primary keys. They are RestockID and ProductID. Both of them are used to uniquely identify each row in the table, and they also become foreign keys as well since they are connected to other entities (restock_header and product entity).

c. **Restock_header** (RestockID, ArrivalDate, VendorID)
Primary key = RestockID
Same as customer entity, restock_header is a weak entity with a foreign key (VendorID).

d. **Transaction_header** (OrderID, Date, StaffID, StoreID, CustomerID)
Primary key = OrderID
In order to connect with other entities, transaction_header has some foreign keys such as StaffID, StoreID, CustomerID. Hence, those foreign keys make transaction_header becoming a weak entity since its existence depends on the strong entity.

e. **Transaction_details** (OrderID, ProductID, Qty, Price)
Primary key = Composite key (OrderID, ProductID)
In transaction_details, there are 2 primary keys (OrderID, and ProductID). Both of them are used to uniquely identify an entity occurrence. They also become foreign keys because those attributes are connected to other entities (transaction_header and product).

Section 5.6 Validate relations using normalization

The purposes of this section are to validate all relations in the tables, organize the attributes and tables of a relational database to minimize data redundancy. Normalization commonly is divided into 3 forms;

First Normal Form (1NF) :
This form is defined in the definition of relations (tables) itself. Here, all derived attributes (such as total, tax, subtotal) and the repetitive data (all attributes should be single valued) are removed.

Second Normal Form (2NF) :
In here, partial dependencies must be removed. It means that every non primary key attributes should be fully functionally dependent on primary key attribute or composite key. If there is a non-primary key attribute that only depends on 1 of the composite key, it should be deleted.

Third Normal Form (3NF) :
For a relation to be in this form, the table has to be in Second Normal Form and there is no transitive functional dependency. Transitive dependency occurs when a non primary key depends on another non primary key attribute (they don’t depend on the primary key itself, they just rely on each other).

1. Customer

CustomerID (PK)	CustomerName	PhoneNo	Address	**PaymentMethod (FK)**

a. Relation has already been in 1NF since there is no repetitive data.
b. Relation has already been in 2NF since there is no partial dependency.
c. Relation has already been in 3NF because there is no transitive dependency.

2. Product

ProductID (PK)	ProductName	Qty	Price

a. Relation has already been in 1NF since there is no repetitive data.
b. Relation has already been in 2NF since there is no partial dependency.
c. Relation has already been in 3NF because there is no transitive dependency.

3. PaymentMethod

PaymentMethodID (PK)	PaymentMethod

a. Relation has already been in 1NF since there is no repetitive data.
b. Relation has already been in 2NF since there is no partial dependency.
c. Relation has already been in 3NF because there is no transitive dependency.

4. Staff

StaffID (PK)	FirstName	LastName	Email	Password	Position	Gender

a. Relation has already been in 1NF since there is no repetitive data.
b. Relation has already been in 2NF since there is no partial dependency.
c. Relation has already been in 3NF because there is no transitive dependency.

5. Store

StoreID (PK)	StoreName	StoreAddress

a. Relation has already been in 1NF since there is no repetitive data.
b. Relation has already been in 2NF since there is no partial dependency.
c. Relation has already been in 3NF because there is no transitive dependency.

6. Transaction

Table 1.6.1 The previous form of Transaction (**before normalization**)

OrderID	ProductID	ProductName	StaffID	StaffName	Date	Store	CustID	CustName	Qty	Price	GrandTotal

a. GrandTotal is removed since it is a derived attribute

b. Separate the table into 2 tables:
   - o Transaction_header
   - o Transaction_details

c. Any partial dependencies are removed (ProductID, ProductName) (making new separate table)

d. Any transitive dependencies are removed (StaffID, StaffName, CustID, CustName, StoreID, StoreName) (making new separate tables)

Transaction is divided into 2 tables (transaction_header and transaction_details)

1. **Transaction_header**

OrderID (PK)	Date	**StaffID (FK)**	**StoreID (FK)**	**CustomerID (FK)**

a. Relation has already been in 1NF since there is no repetitive data.
b. Relation has already been in 2NF since there is no partial dependency.
c. Relation has already been in 3NF because there is no transitive dependency.

2. **Transaction_details**

OrderID (PK, FK)	**ProductID (PK,FK)**	Qty	Price

a. Relation has already been in 1NF since there is no repetitive data.
b. Relation has already been in 2NF since there is no partial dependency.
c. Relation has already been in 3NF because there is no transitive dependency.

7. **Restock**

Table 1.6.2 The previous form of restock table (**before normalization**)

RestockID	ProductID	ProductName	ArrivalDate	Qty	GrandTotal	VendorID	VendorName

a. GrandTotal is removed since it is a derived attribute

b. Separate the table into 2 tables
   - o Restock_header
   - o Restock_details

c. Any partial dependencies are removed (ProductID, ProductName) (making new separate table)

d. Any transitive dependencies are removed (VendorID, VendorName) (making new separate tables)

Restock is divided into 2 tables (restock_header and restock_details)

1. **Restock_header**

RestockID (PK)	ArrivalDate	**VendorID (FK)**

a. Relation has already been in 1NF since there is no repetitive data.
b. Relation has already been in 2NF since there is no partial dependency.
c. Relation has already been in 3NF because there is no transitive dependency.

2. Restock_details

RestockID (PK, FK)	**ProductID (PK,FK)**	Qty

a. Relation has already been in 1NF since there is no repetitive data.
b. Relation has already been in 2NF since there is no partial dependency.
c. Relation has already been in 3NF because there is no transitive dependency.

8. Vendor

VendorID (PK)	VendorName	VendorAddress	VendorNo

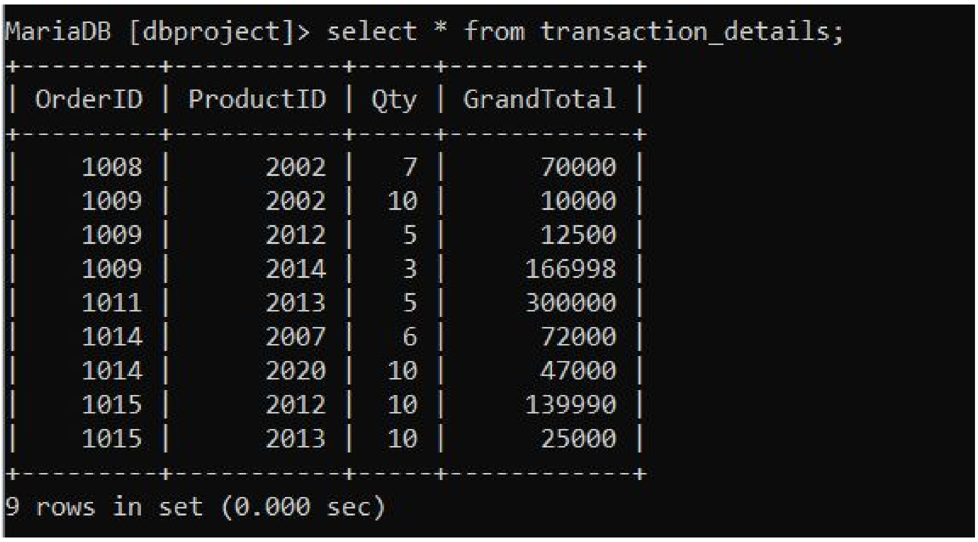
a. Relation has already been in 1NF since there is no repetitive data.
b. Relation has already been in 2NF since there is no partial dependency.
c. Relation has already been in 3NF because there is no transitive dependency.

6. Sample Queries

Query for transaction_details:

- insert into transaction_details values(1015, 2013, 10, 25000);
- insert into transaction_details values(1014, 2020, 10, 47000);
- insert into transaction_details values(1009, 2012, 5, 12500);

Final results :

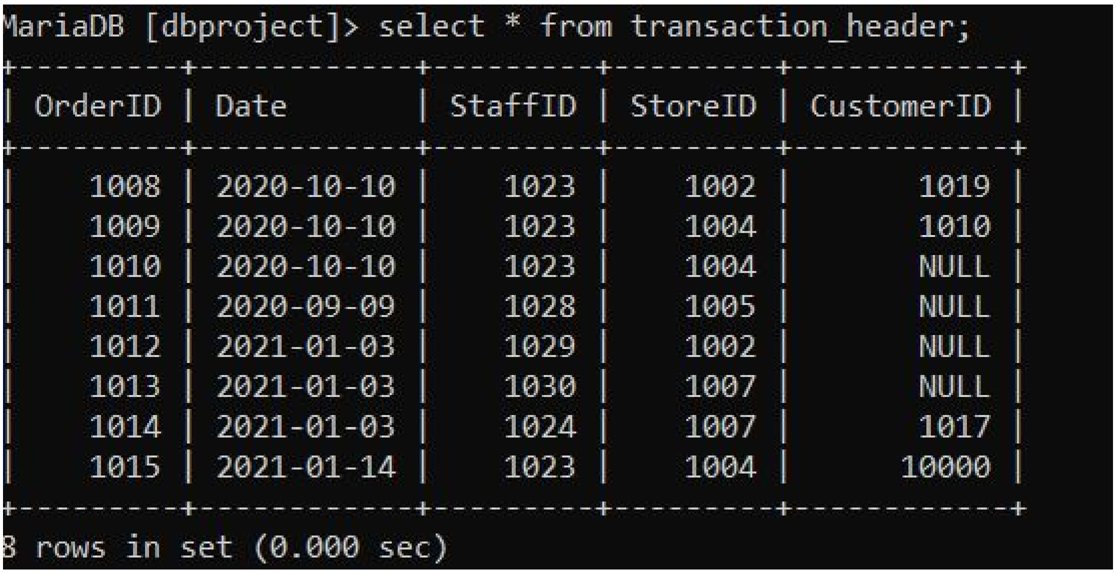


```
MariaDB [dbproject]> select * from transaction_details;
+---------+-----------+-----+------------+
| OrderID | ProductID | Qty | GrandTotal |
+---------+-----------+-----+------------+
|    1008 |      2002 |   7 |      70000 |
|    1009 |      2002 |  10 |      10000 |
|    1009 |      2012 |   5 |      12500 |
|    1009 |      2014 |   3 |     166998 |
|    1011 |      2013 |   5 |     300000 |
|    1014 |      2007 |   6 |      72000 |
|    1014 |      2020 |  10 |      47000 |
|    1015 |      2012 |  10 |     139990 |
|    1015 |      2013 |  10 |      25000 |
+---------+-----------+-----+------------+
9 rows in set (0.000 sec)
```

Query for Transaction header:

- insert into transaction_header (OrderID, Date, StaffID, StoreID) values (1012, "2021-01-03", 1029, 1002;
- insert into transaction_header (OrderID, Date, StaffID, StoreID) values (1013, "2021-01-03", 1030, 1007);

Final results :

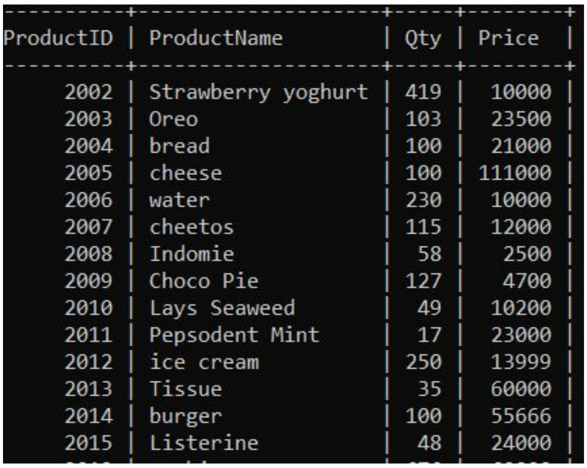


```
MariaDB [dbproject]> select * from transaction_header;
+---------+------------+---------+---------+------------+
| OrderID | Date       | StaffID | StoreID | CustomerID |
+---------+------------+---------+---------+------------+
|    1008 | 2020-10-10 |    1023 |    1002 |       1019 |
|    1009 | 2020-10-10 |    1023 |    1004 |       1010 |
|    1010 | 2020-10-10 |    1023 |    1004 |       NULL |
|    1011 | 2020-09-09 |    1028 |    1005 |       NULL |
|    1012 | 2021-01-03 |    1029 |    1002 |       NULL |
|    1013 | 2021-01-03 |    1030 |    1007 |       NULL |
|    1014 | 2021-01-03 |    1024 |    1007 |       1017 |
|    1015 | 2021-01-14 |    1023 |    1004 |      10000 |
+---------+------------+---------+---------+------------+
8 rows in set (0.000 sec)
```

Query for product:

a. insert into product values(2008, "Indomie", 58, 2500);
b. insert into product values(2009, "Choco Pie", 127, 4700);
c. insert into product values(2010, "Lays seaweed", 49, 10200);
d. insert into product values(2011, "Pepsodent Mint", 17, 23000);
e. insert into product values(2015,"Listerine", 48, 24000);

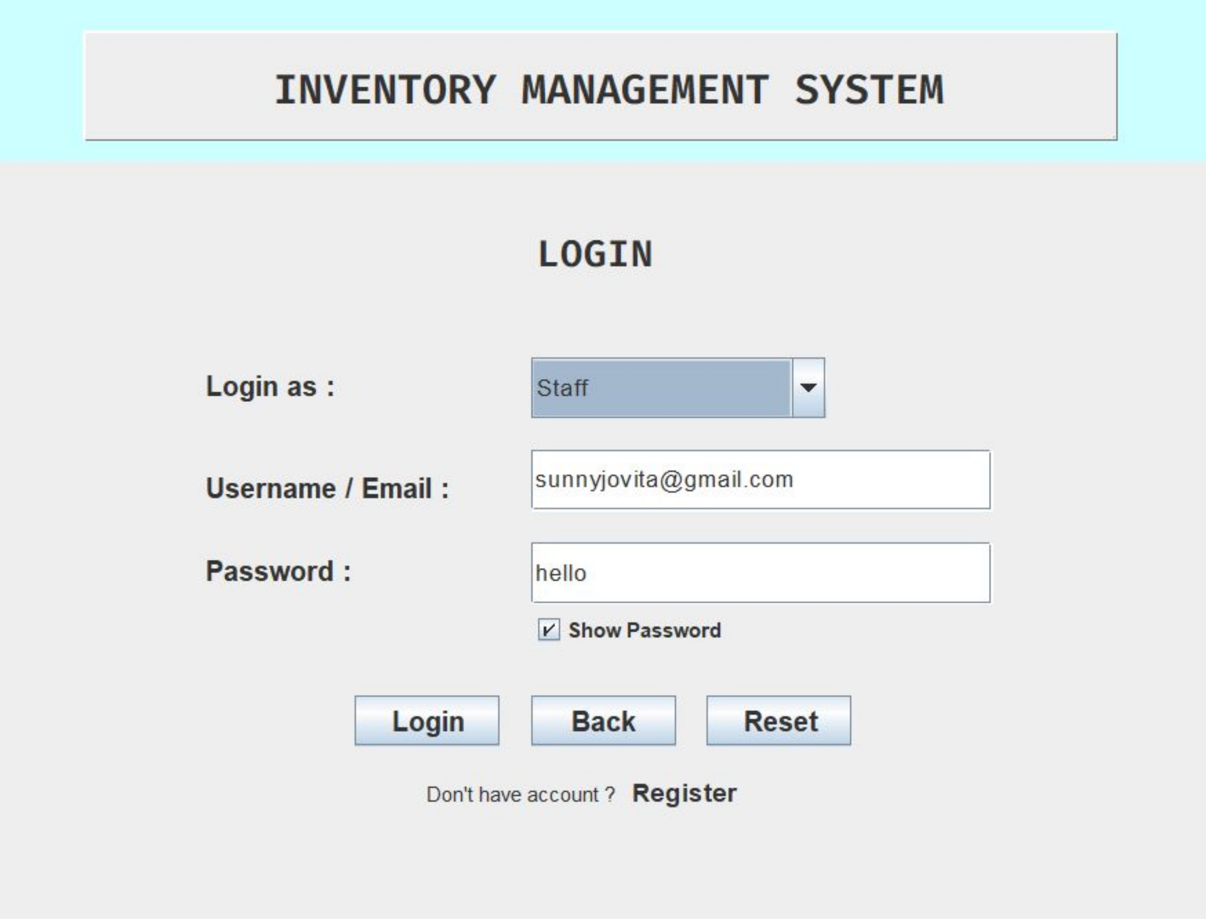
Final results :



```
----------+--------------------+-----+--------+
ProductID | ProductName        | Qty | Price  |
----------+--------------------+-----+--------+
     2002 | Strawberry yoghurt | 419 |  10000 |
     2003 | Oreo               | 103 |  23500 |
     2004 | bread              | 100 |  21000 |
     2005 | cheese             | 100 | 111000 |
     2006 | water              | 230 |  10000 |
     2007 | cheetos            | 115 |  12000 |
     2008 | Indomie            |  58 |   2500 |
     2009 | Choco Pie          | 127 |   4700 |
     2010 | Lays Seaweed       |  49 |  10200 |
     2011 | Pepsodent Mint     |  17 |  23000 |
     2012 | ice cream          | 250 |  13999 |
     2013 | Tissue             |  35 |  60000 |
     2014 | burger             | 100 |  55666 |
     2015 | Listerine          |  48 |  24000 |
```

7. User interfaces

Figure 1.8.1 Login Form



INVENTORY MANAGEMENT SYSTEM

LOGIN

Login as : Staff

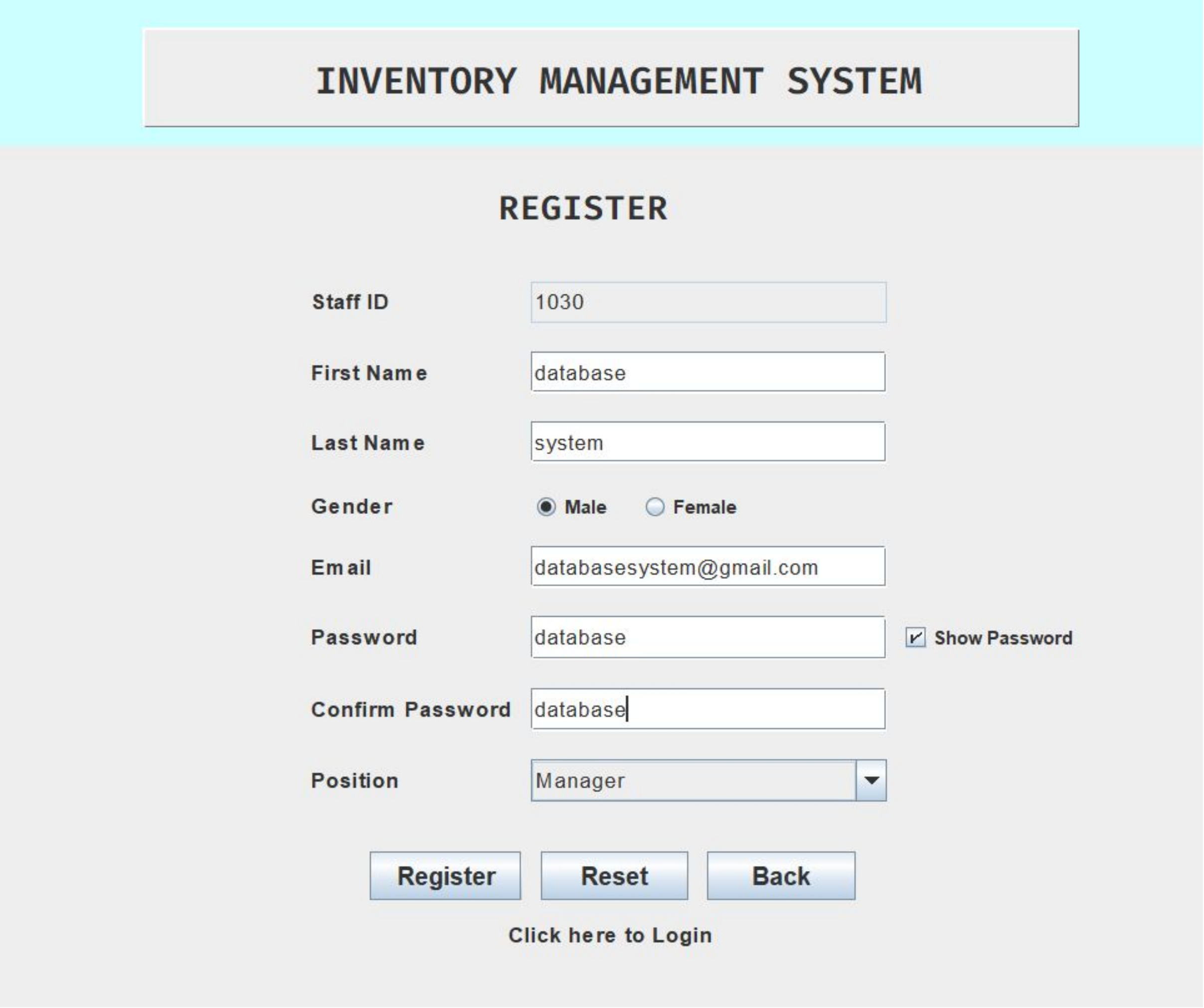
Username / Email : sunnyjovita@gmail.com

Password : hello

☑ Show Password

This is the login screen when the program is launched through Java. Note that there are 2 types of admin access when logging into the system, which are Staff & Manager accounts. The reason why we created 2 types of admin due to the different staff roles and necessities to interact and modify with the data. Therefore, if a “Staff” tries to login as a “Manager”, there will be an error.

Figure 1.8.2 Register Form



INVENTORY MANAGEMENT SYSTEM

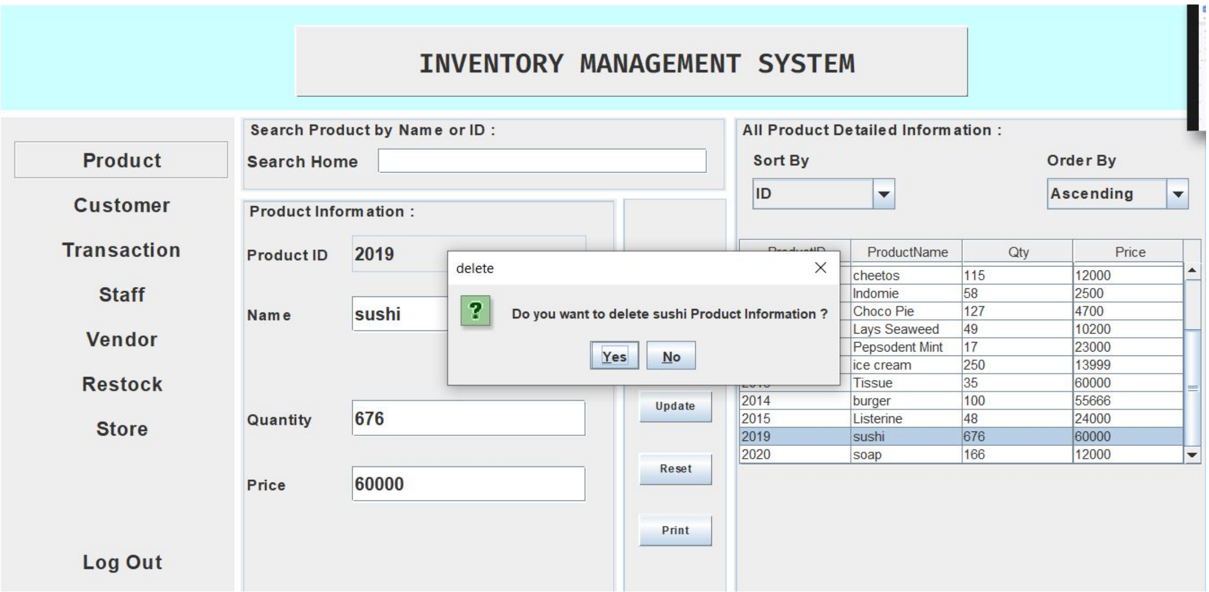
REGISTER

Staff ID	1030
First Name	database
Last Name	system
Gender	● Male ○ Female
Email	databasesystem@gmail.com
Password	database ☑ Show Password
Confirm Password	database
Position	Manager

The picture above shows the interface when a user registers for an account. The program asks the user to input a few details about themselves, including First Name, Last Name, Gender, Email, Password, Confirm Password, and select position of the Staff.

Note: If the user fails to type the same password to be confirmed, there will be an error message. Likewise, if the registration was successful, the user will be taken back to the login page.

Figure 1.8.3 Product Form



Search Product by Name or ID :

Search Home

Product Information :

Product ID	2019
Name	sushi
Quantity	676
Price	60000

delete

Do you want to delete sushi Product Information ?

All Product Detailed Information :

Sort By: ID

Order By: Ascending

ProductID	ProductName	Qty	Price
	cheetos	115	12000
	Indomie	58	2500
	Choco Pie	127	4700
	Lays Seaweed	49	10200
	Pepsodent Mint	17	23000
	ice cream	250	13999
	Tissue	35	60000
2014	burger	100	55666
2015	Listerine	48	24000
2019	sushi	676	60000
2020	soap	166	12000

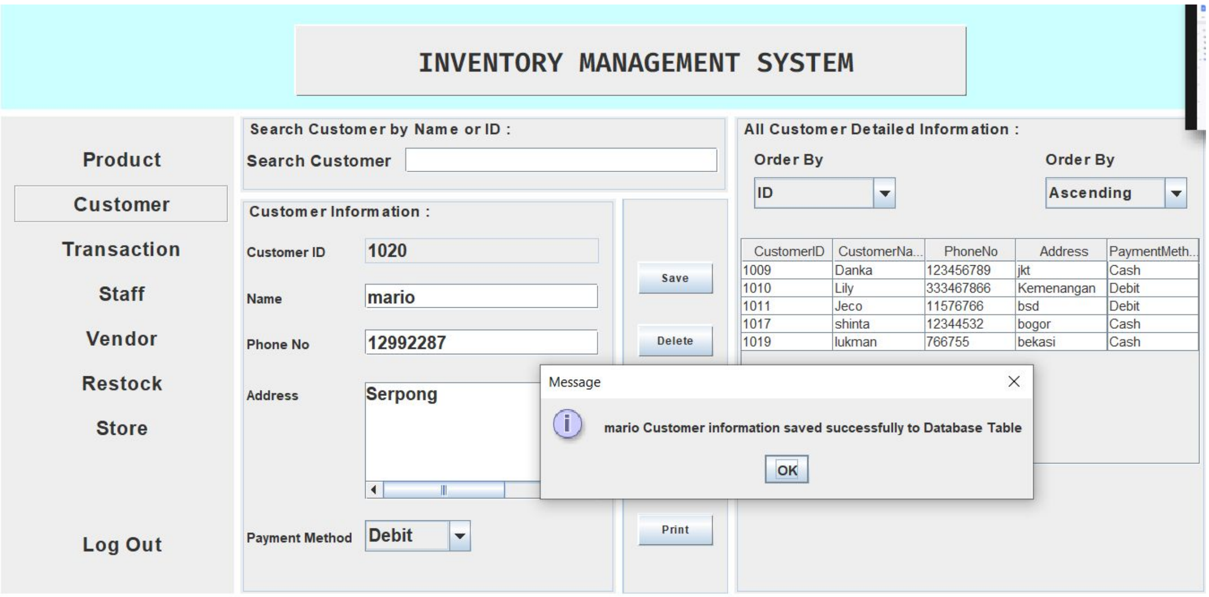
The picture above shows the interface when the user wants to add/delete a product from the database. To add a product, the user must fill in the required fields, such as the Product Name, Quantity, Price, etc. This will then be added when the user clicks the "Save" button (covered by delete interface, see next picture for the "Save" button)

Note: When the user successfully added a new product, the program will automatically generate a unique Product ID for the specific product

To delete a product, a user can use the "Search Home" bar to search for the product they want to delete, then, click on the product on the right hand side of the screen which shows the filtered data based on the search engine. When the row is clicked, it will turn blue, then the details of the selected product will appear automatically in the "Product Information" fields. Finally, the user is able to manipulate the data from here, including deleting it.

Note: Everytime the user chooses one of the buttons, there will be a pop up interface to confirm the action.

Figure 1.8.4 Customer Form



Search Customer by Name or ID :

Search Customer

Customer Information :

Customer ID: 1020

Name: mario

Phone No: 12992287

Address: Serpong

Payment Method: Debit

All Customer Detailed Information :

Order By: ID

Order By: Ascending

CustomerID	CustomerNa...	PhoneNo	Address	PaymentMeth...
1009	Danka	123456789	jkt	Cash
1010	Lily	333467866	Kemenangan	Debit
1011	Jeco	11576766	bsd	Debit
1017	shinta	12344532	bogor	Cash
1019	lukman	766755	bekasi	Cash

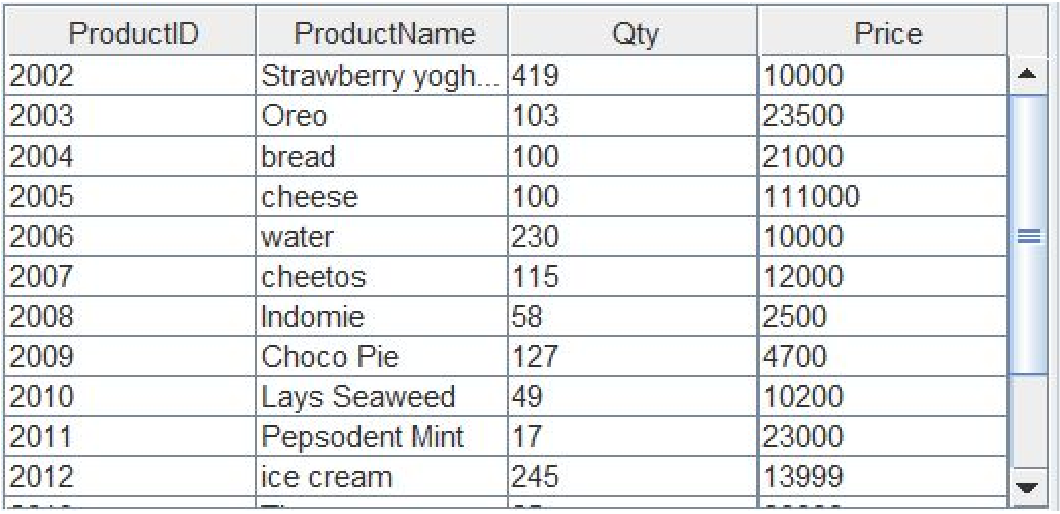
mario Customer information saved successfully to Database Table

The screenshot above shows the interface when an admin (user) of the program wants to add, manipulate, or delete a customer. Similar to the Product interface in the previous screenshot, the user is able to filter the data, manipulate, and save the customer data.

8. Transaction Management and Concurrency Control

The purpose of this section is to explain the transaction function control in mysql and Java Netbeans.

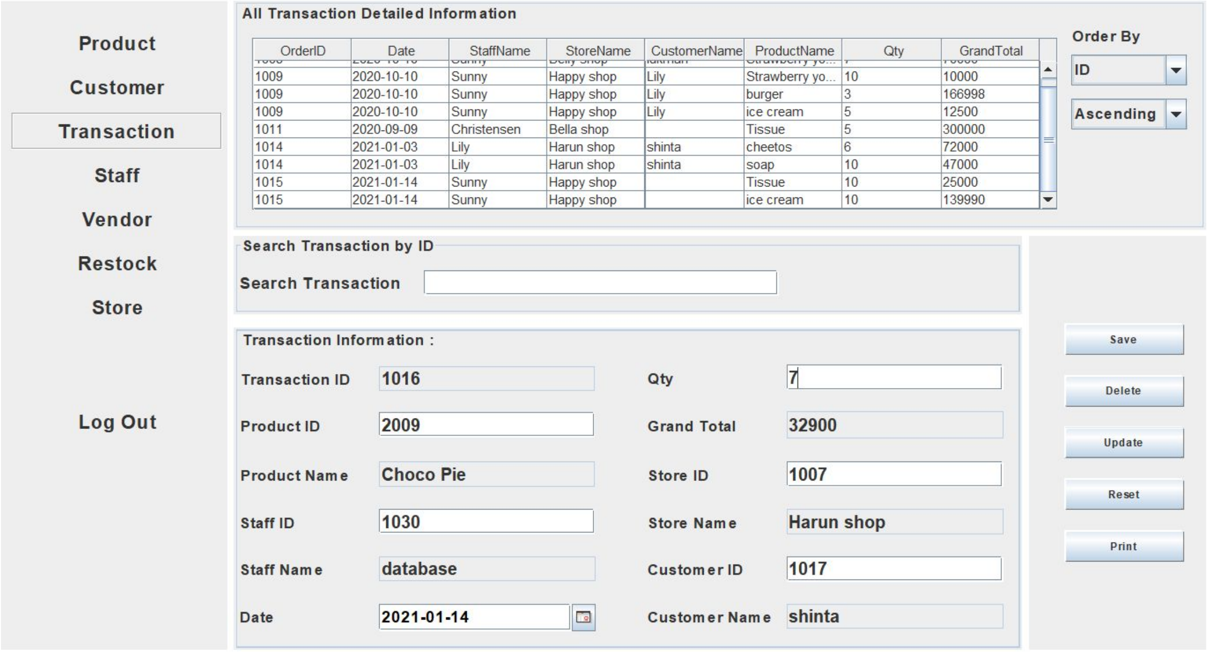
Figure 1.9.1 Product Table



ProductID	ProductName	Qty	Price
2002	Strawberry yogh...	419	10000
2003	Oreo	103	23500
2004	bread	100	21000
2005	cheese	100	111000
2006	water	230	10000
2007	cheetos	115	12000
2008	Indomie	58	2500
2009	Choco Pie	127	4700
2010	Lays Seaweed	49	10200
2011	Pepsodent Mint	17	23000
2012	ice cream	245	13999

In this picture, all the product's information here as well the product's attributes. For example, we choose the Choco Pie product with the Quantity 127 as the product sample in the transaction process.

Figure 1.9.2 Transaction Form



All Transaction Detailed Information

OrderID	Date	StaffName	StoreName	CustomerName	ProductName	Qty	GrandTotal
1009	2020-10-10	Sunny	Happy shop	Lily	Strawberry yo...	10	10000
1009	2020-10-10	Sunny	Happy shop	Lily	burger	3	166998
1009	2020-10-10	Sunny	Happy shop	Lily	ice cream	5	12500
1011	2020-09-09	Christensen	Bella shop		Tissue	5	300000
1014	2021-01-03	Lily	Harun shop	shinta	cheetos	6	72000
1014	2021-01-03	Lily	Harun shop	shinta	soap	10	47000
1015	2021-01-14	Sunny	Happy shop		Tissue	10	25000
1015	2021-01-14	Sunny	Happy shop		ice cream	10	139990

Order By: ID; Ascending

Search Transaction by ID

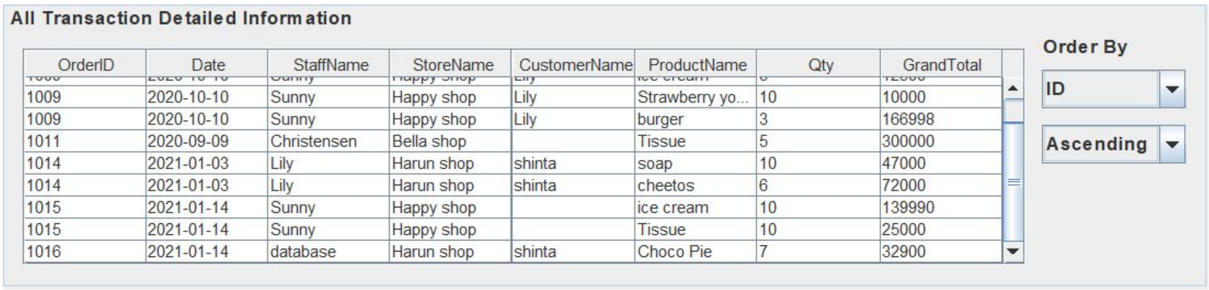
Search Transaction:

Transaction Information :

Transaction ID	1016	Qty	7
Product ID	2009	Grand Total	32900
Product Name	Choco Pie	Store ID	1007
Staff ID	1030	Store Name	Harun shop
Staff Name	database	Customer ID	1017
Date	2021-01-14	Customer Name	shinta

As we can see in the transaction information, all the fields are already filled within the text fields. Here, we are going to take the quantity of 7 choco pies, and the price will automatically follow the number of product's quantity listed. After that the user can save the transaction information by clicking the save button.

Figure 1.9.3 Transaction Table



All Transaction Detailed Information

OrderID	Date	StaffName	StoreName	CustomerName	ProductName	Qty	GrandTotal
1009	2020-10-10	Sunny	Happy shop	Lily	Strawberry yo...	10	10000
1009	2020-10-10	Sunny	Happy shop	Lily	burger	3	166998
1011	2020-09-09	Christensen	Bella shop		Tissue	5	300000
1014	2021-01-03	Lily	Harun shop	shinta	soap	10	47000
1014	2021-01-03	Lily	Harun shop	shinta	cheetos	6	72000
1015	2021-01-14	Sunny	Happy shop		ice cream	10	139990
1015	2021-01-14	Sunny	Happy shop		Tissue	10	25000
1016	2021-01-14	database	Harun shop	shinta	Choco Pie	7	32900

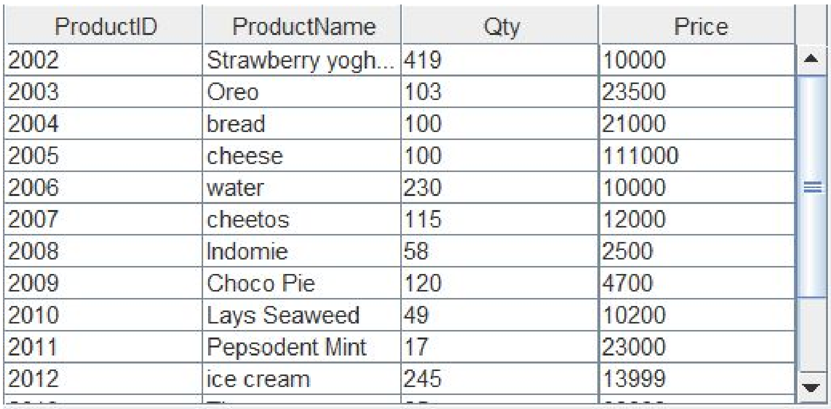
Order By

ID

Ascending

As a result of that, the transaction information that we already saved before has been successfully saved into the transaction table and also in the product table, the quantity of the choco price has decreased and it becomes 120 from 127.

Figure 1.9.4 Product Table



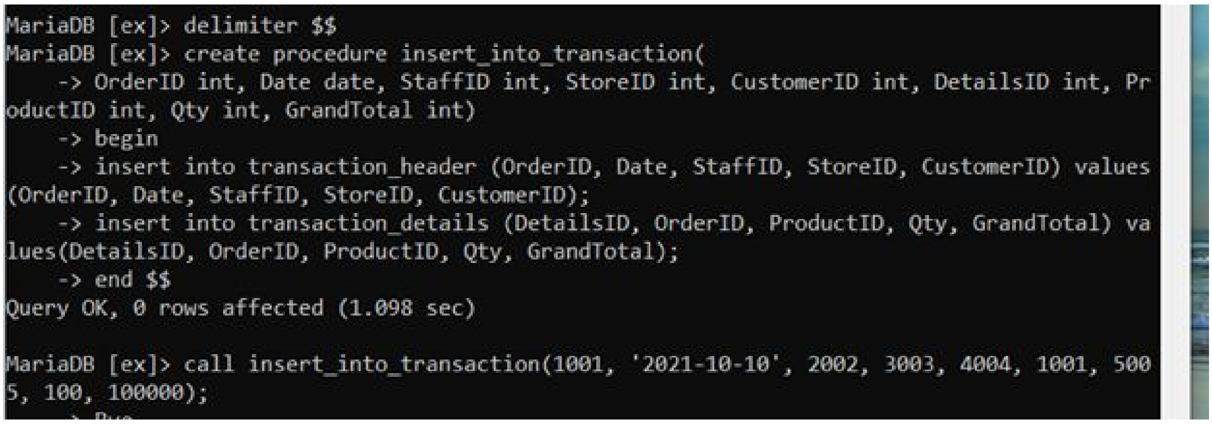
ProductID	ProductName	Qty	Price
2002	Strawberry yogh...	419	10000
2003	Oreo	103	23500
2004	bread	100	21000
2005	cheese	100	111000
2006	water	230	10000
2007	cheetos	115	12000
2008	Indomie	58	2500
2009	Choco Pie	120	4700
2010	Lays Seaweed	49	10200
2011	Pepsodent Mint	17	23000
2012	ice cream	245	13999

Triggers and Stored Procedures among Product, Restock, and Transaction program.

In order to create a relationship among product, transaction and restock, we use stored procedures or database procedure and DML Trigger (insert/update/delete) to help the operation of transaction and restock systems. First of all, since we separated the transaction table into 2 tables (transaction_header and transaction_details) and also restock table (restock_header and restock_details), we have to input values for both of them manually. Therefore, we use stored procedures to concatenate one or more SQL statements which will perform a specific task to make it efficient.

For instance, to insert some values into transaction_header and transaction_details, stored procedures can be used to minimize redundancy.

Figure 1.10.1 Creating and calling procedure for insert into transaction

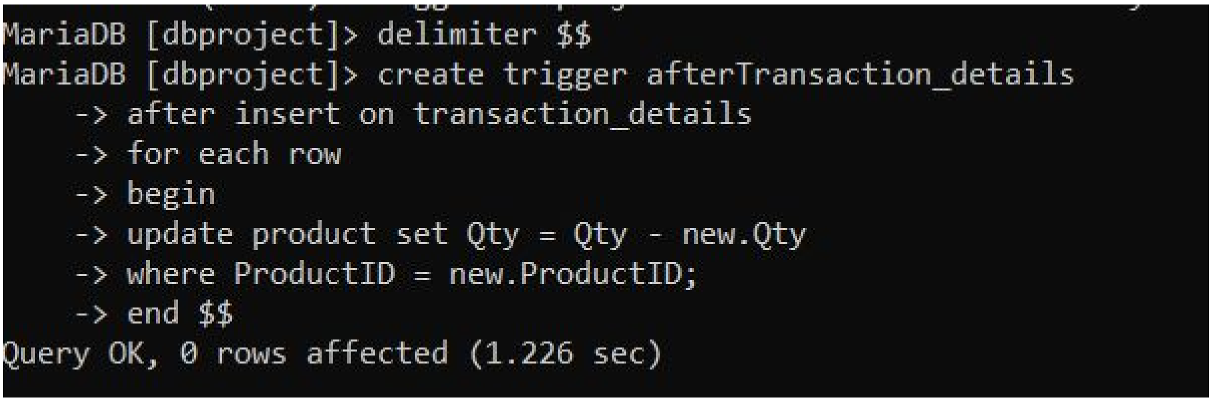


```
MariaDB [ex]> delimiter $$
MariaDB [ex]> create procedure insert_into_transaction(
    -> OrderID int, Date date, StaffID int, StoreID int, CustomerID int, DetailsID int, Pr
oductID int, Qty int, GrandTotal int)
    -> begin
    -> insert into transaction_header (OrderID, Date, StaffID, StoreID, CustomerID) values
(OrderID, Date, StaffID, StoreID, CustomerID);
    -> insert into transaction_details (DetailsID, OrderID, ProductID, Qty, GrandTotal) va
lues(DetailsID, OrderID, ProductID, Qty, GrandTotal);
    -> end $$
Query OK, 0 rows affected (1.098 sec)

MariaDB [ex]> call insert_into_transaction(1001, '2021-10-10', 2002, 3003, 4004, 1001, 500
5, 100, 100000);
```

In **Figure 1.10.1**, we can call the procedure ‘insert_into_transaction’ (without entering OrderID twice). But if we try to input manually, we have to input the OrderID twice because OrderID is the primary key of transaction_header and transaction_details. After the query “call insert_into_transaction” runs, it will show the user that the process is successful.

Figure 1.10.2 Trigger after insert on transaction_details

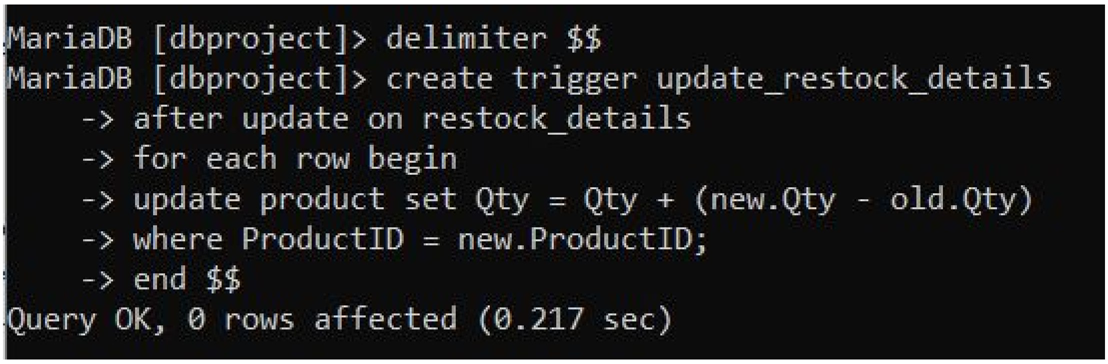


```
MariaDB [dbproject]> delimiter $$
MariaDB [dbproject]> create trigger afterTransaction_details
    -> after insert on transaction_details
    -> for each row
    -> begin
    -> update product set Qty = Qty - new.Qty
    -> where ProductID = new.ProductID;
    -> end $$
Query OK, 0 rows affected (1.226 sec)
```

In terms of product quantity, we can reduce the number of products by creating a trigger which is indicated to update the product quantity in the product table after inserting on transaction_details. The number of products will be deducted by the product quantity in the transaction program.

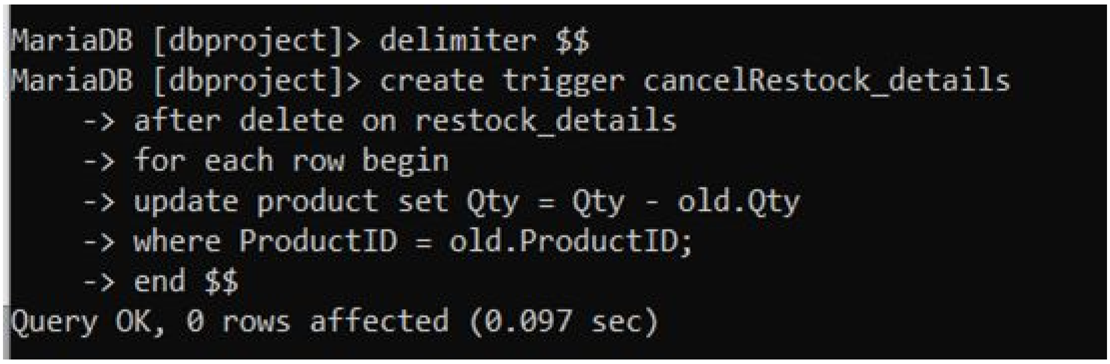
Same as the insert functions, these systems also use triggers for updating and deleting the transaction / restock process which will impact the actual product’s quantity. The delete and update triggers will be shown on the **Figure 1.10.3** and **Figure 1.10.4**

Figure 1.10.3 Trigger after update on restock_details



```
MariaDB [dbproject]> delimiter $$
MariaDB [dbproject]> create trigger update_restock_details
    -> after update on restock_details
    -> for each row begin
    -> update product set Qty = Qty + (new.Qty - old.Qty)
    -> where ProductID = new.ProductID;
    -> end $$
Query OK, 0 rows affected (0.217 sec)
```

Figure 1.10.4 Trigger after delete on restock_details



```
MariaDB [dbproject]> delimiter $$
MariaDB [dbproject]> create trigger cancelRestock_details
    -> after delete on restock_details
    -> for each row begin
    -> update product set Qty = Qty - old.Qty
    -> where ProductID = old.ProductID;
    -> end $$
Query OK, 0 rows affected (0.097 sec)
```

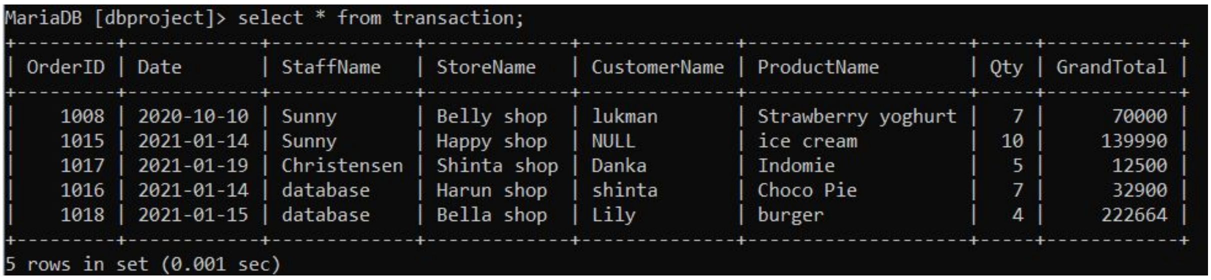
9. Database security

Creating view

1. **View for transaction**

   create view transaction as select transaction_header.OrderID, transaction_header.Date, staff.FirstName as StaffName, store.StoreName, customer.CustomerName, product.ProductName, transaction_details.Qty, transaction_details.GrandTotal from transaction_header join staff on staff.StaffID = transaction_header.StaffID join store on store.StoreID = transaction_header.StoreID left join customer on customer.CustomerID = transaction_header.CustomerID join transaction_details on transaction_details.OrderID = transaction_header.OrderID join product on transaction_details.ProductID = product.ProductID;

   Final result :

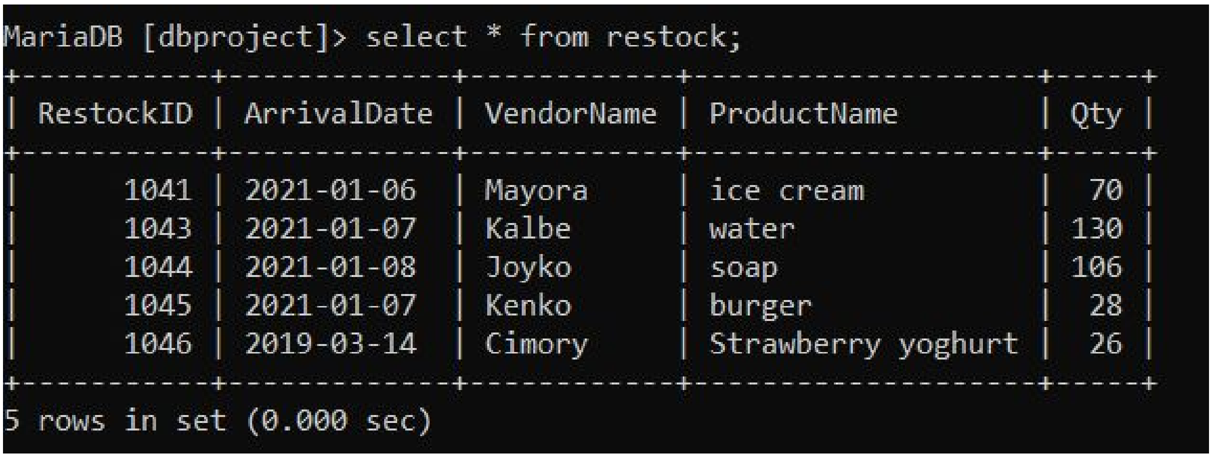


```
MariaDB [dbproject]> select * from transaction;
+---------+------------+-------------+------------+--------------+--------------------+-----+------------+
| OrderID | Date       | StaffName   | StoreName  | CustomerName | ProductName        | Qty | GrandTotal |
+---------+------------+-------------+------------+--------------+--------------------+-----+------------+
|    1008 | 2020-10-10 | Sunny       | Belly shop | lukman       | Strawberry yoghurt |   7 |      70000 |
|    1015 | 2021-01-14 | Sunny       | Happy shop | NULL         | ice cream          |  10 |     139990 |
|    1017 | 2021-01-19 | Christensen | Shinta shop | Danka       | Indomie            |   5 |      12500 |
|    1016 | 2021-01-14 | database    | Harun shop | shinta       | Choco Pie          |   7 |      32900 |
|    1018 | 2021-01-15 | database    | Bella shop | Lily         | burger             |   4 |     222664 |
+---------+------------+-------------+------------+--------------+--------------------+-----+------------+
5 rows in set (0.001 sec)
```

2. **View for restock**

Create view restock as select restock_header.RestockID, restock_header.ArrivalDate, vendor.VendorName, product.ProductName, restock_details.Qty from restock_header join restock_details on restock_header.RestockID = restock_details.RestockID join vendor on vendor.VendorID = restock_header.VendorID join product on product.ProductID = restock_details.ProductID;

Final result :



```
MariaDB [dbproject]> select * from restock;
+-----------+-------------+------------+--------------------+-----+
| RestockID | ArrivalDate | VendorName | ProductName        | Qty |
+-----------+-------------+------------+--------------------+-----+
|      1041 | 2021-01-06  | Mayora     | ice cream          |  70 |
|      1043 | 2021-01-07  | Kalbe      | water              | 130 |
|      1044 | 2021-01-08  | Joyko      | soap               | 106 |
|      1045 | 2021-01-07  | Kenko      | burger             |  28 |
|      1046 | 2019-03-14  | Cimory     | Strawberry yoghurt |  26 |
+-----------+-------------+------------+--------------------+-----+
5 rows in set (0.000 sec)
```

10. Granting access for outsiders and creators

Permissions are actions where the user is allowed to perform in the database. The purpose of this section is to set permissions for the user before logging in with the new account.

Regarding the database security for our Inventory Database system, we as the creators grant full control. It means that we as the users have full access to the database. The ones who got the all privileges authority are able to insert, delete, create, drop, select, update, and modify other user account privileges. But for other people (outsiders), we decided to grant them one privilege, and it is a select privilege.

Since we don’t want to grant the outsider full control, we give them the credentials of a non-root user, so we can keep track of what they can and cannot not with our data.

11. GitHub & Video Link

GitHub Link:

https://github.com/sunnyjovita/Final-Database-Project-LEC/tree/main

Google Drive Link:

https://drive.google.com/drive/folders/1A3gMwyLsDfbkSCxcTiZBTE_yvf49iG05?usp=sharing

Note: If in any case, the and one or both of the link above doesn’t work, please kindly contact one of our group members for assistance.