Documentation

Team Members: Kshitij Tandon (2020A7PS0972P)
Devashish Siwatch (2020A7PS0113P)

Project Name: Supermarket Management System
Project Number: 20

Explanation video:

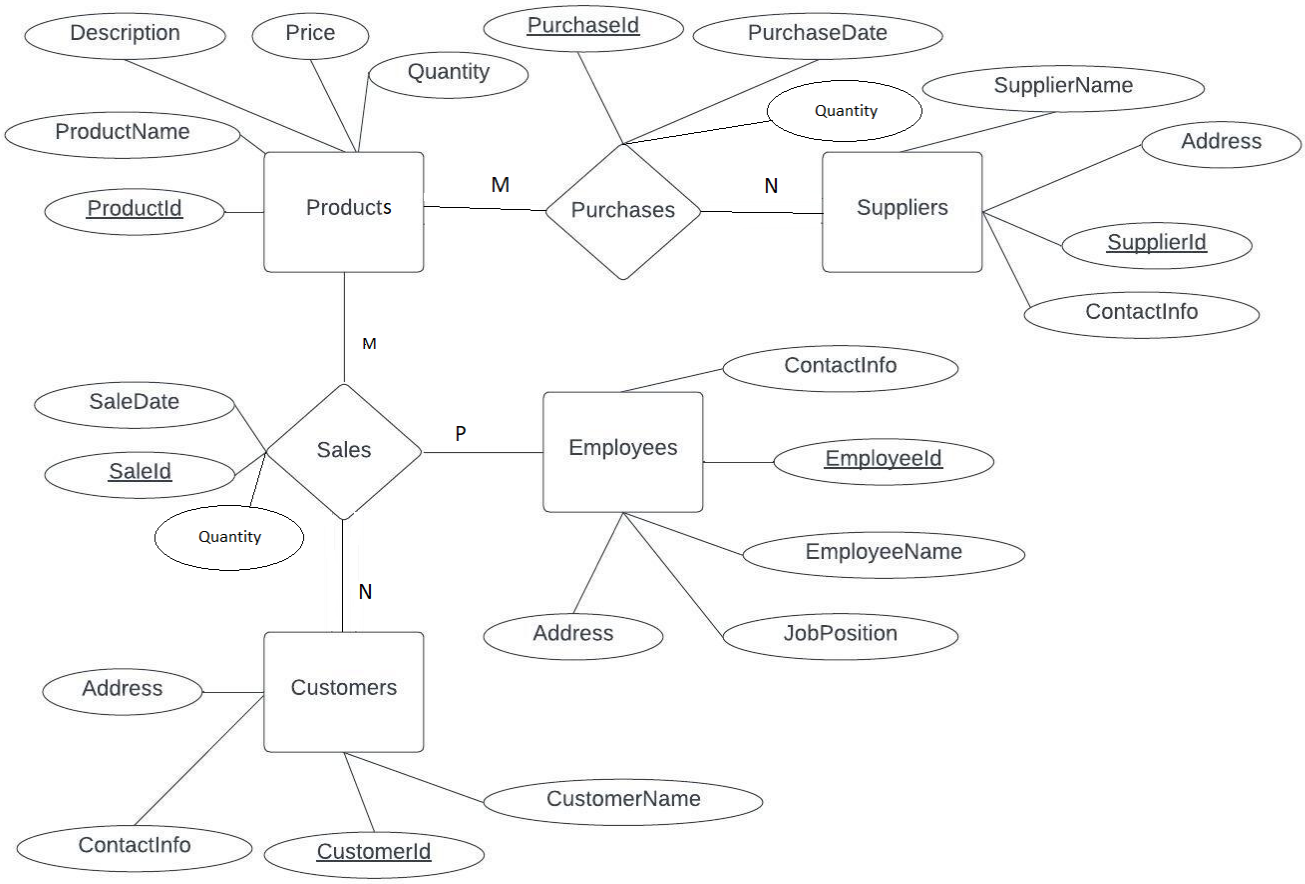
1)Kshitij Tandon:

https://drive.google.com/file/d/18QaJKNLxVjziwM8a3mKutw0ECbgsj-De/view?usp=share_link

2) Devashish Siwatch:

https://drive.google.com/file/d/1eh-NDyc3BB41AcS1ls92lrt6TMcCzfsE/view?usp=share_link

ER-MODEL



The entity-relationship model for our proposed database system has been shown above. We have taken 4 entities in our model and have included 2 relations among them. The entities used in our ER model are:

1) Customers: The entity customers has been made to store the details of the customers of the supermarket. It has four attributes associated with it namely address, for storing the address of the customer, conatctInfo for

storing the phone number of the customer, customerId which is also the primary key for it as every customer has a unique Id associated with him and finally the customerName for storing the name of the customer. All its attributes are single valued and there is no multi valued or derived attribute in it.

2) Employees: The entity employees has been made to store the details of the employees of the supermarket. It has five attributes associated with it namely address, for storing the address of the employee, conatctInfo for storing the phone number of the employee, employeeId which is also the primary key for it as every employee has a unique Id associated with him, JobPosition for storing the current position of the employee in the supermarket and finally the employeeName for storing the name of the employee. All its attributes are single valued and there is no multi valued or derived attribute in it.

3) Suppliers: The entity suppliers has been made to store the details of the suppliers of the supermarket. It has four attributes associated with it namely address, for storing the address of the supplier, conatctInfo for storing the phone number of the supplier, supplierId which is also the primary key for it as every supplier has a unique Id associated with it and finally the supplierName for storing the name of the supplier. All its attributes are single valued and there is no multi valued or derived attribute in it.

4) Products: The entity products has been made to store the details of the products in the supermarket. It has five attributes associated with it namely description, for storing a brief description about the product, price for storing the market price of the product, quantity for storing the quantity of the product that the supermarket currently has in store, productId which is also the primary key for it as every product has a unique Id associated with it and finally the productName for storing the name of the product. All its

attributes are single valued and there is no multi valued or derived attribute in it.

Now there are two relationships in the model, which are:

1) Purchases: The purchases relation is used to show the relationship between the products and the suppliers entity. It is basically used to keep a record of any of the purchases of the products the supermarket makes from any of its registered suppliers. It stores the following attributes in it: PurchaseDate which is used to record the date when the purchase was made from the supplier, Quantity to store the quantity of the product purchased, PurchaseId which is used to uniquely identify a purchase of a product which also makes it the primary key for the relation.

2) Sales: The sales relation is used to show the relationship between the products, customers and the employees entity. It is basically used to keep a record of any of the sales of the products the supermarket makes to any of its registered customers and also the corresponding employee associated with the sale. It stores the following attributes in it: SaleDate which is used to record the date when the sale was made to the customer, Quantity to store the quantity of the product sold, SaleId which is used to uniquely identify the sale of a product which also makes it the primary key for the relation.

ER TO RELATIONAL

Now for converting our ER model to relational schema we see that there are no fully functional dependencies in our Entity-Relationship model and also there are no weak entity types. Therefore we can simply make a separate table for each of the entities and also separate table for both of our relations i.e. Sales and Purchases. Also note that there are no total dependencies so we can't merge any relations and tables. Also only 2 relations are there for sales and purchases which will have foreign keys for the entities they're relating.

Conversions:

1) customers(customer_name,
customer_id,
address,
contact_info)

Primary key - customer_id
Candidate key- customer_id
F.D.: {customer_id->address, contact_info, customer_name}.

2) employees(employee_name,
employee_id,
address,
contact_info,
job_position)

Primary key - employee_id
Candidate key- employee_id
F.D.: {employee_id->address, contact_info, employee_name, job_position}.

3) suppliers(
supplier_name,
supplier_id,
address,
contact_info)

Primary key- supplier_id
Candidate key- supplier_id
F.D.: {supplier_id->address, contact_info, supplier_name}.

4) products(
product_name,
product_id,
description,
price,
quantity)

Primary key- product_id
Candidate key- product_id
F.D. : {product_id->description,price,quantity,product_name}

5) sales(
date_of_sale,
sales_id,
customer_id,
product_id,
employee_id,
quantity,
)

Primary key- sales_id
Candidate key- sales_id
Foreign keys- customer_id, product_id, employee_id
F.D. : {sales-Id-> date_of_sale, customer_id, product_id, employee_id, quantity}

6) purchases(
date_of_purchase,
purchases_id,
supplier_id,
product_id,
quantity)

Primary key- purchases_id
Candidate key- purchases_id
Foreign keys- supplier_id, product_id
F.D. : {purchases-Id-> date_of_purchase, supplier_id, product_id, quantity}

NORMALIZATION

Now our proposed schema is already in 3NF as we have ensured that no partial or transitive dependencies exist within any of our tables. Also by assuming in our database design that one sale can only record sale of one particular product and same for purchases, we have tried to cut down on inconsistencies and redundancies. It has also ensured that our schema stays in 1NF since every attribute of a particular record can have only 1 value associated with it. Other tables i.e. Customers, Suppliers, Employees and Products are also ensured to be in 1NF by putting the above constraint. For eg. everyone is allowed to have just one contact no. and address associated with them and the other attributes our surely going to be in 1NF.

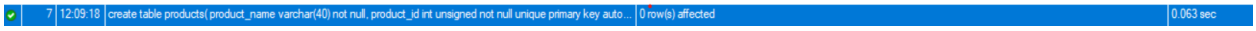
Now for ensuring schema to be in 2NF, we have ensured that no partial dependencies exist in it. Now since we have just one primary key for all the tables i.e. the respective unique ID's associated with each of the tables, there can never be a problem of partial dependency since the full functional dependency will only derive another attribute.

Now for 3NF, our schema is again already in 3NF as there are no transitive dependencies in it. This is because of the Id's which we have associated with each of our entities since any functional dependecy will necessarily have to contain the Id of that entity (as shown above in the functional dependencies) in it since it is going to be unique for every tuple and therefore to identify all the tuple we will need to compulsorily have it in the super key. Thus this satisfies the constraint required for an entity to be in 3NF.

SQL QUERIES WITH OUTPUT SCREENSHOTS

1) Create a table for products, including columns for product name, description, price, and quantity in stock.

```
create table products(
product_name varchar(40) not null,
product_id int unsigned not null unique primary key auto_increment,
description varchar(200),
price decimal(10,2) not null check(price>0),
quantity int not null default '0' check(quantity>=0)
);
```



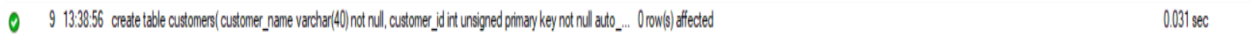
2) Create a table for suppliers, including columns for supplier name, address, and contact information.

```
create table suppliers(
supplier_name varchar(40) not null,
```

```
supplier_id int unsigned primary key not null auto_increment,
address varchar(100),
contact_info char(10)
);
```

3) Create a table for customers, including columns for customer name, address, and contact information.

```
create table customers(
customer_name varchar(40) not null,
customer_id int unsigned primary key not null auto_increment,
address varchar(100),
contact_info char(10)
);
```



4) Create a table for employees, including columns for employee name, address, contact information, and job position.

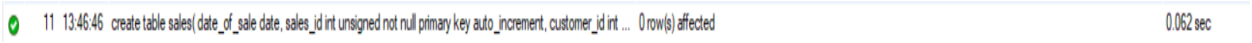
```
create table employees(
employee_name varchar(40) not null,
employee_id int unsigned primary key not null auto_increment,
address varchar(100),
contact_info char(10),
job_position varchar(40)
);
```

5) Create a table for sales, including columns for the date of the sale, the customer who made the purchase, and the products sold.

7) Create a foreign key constraint between the sales table and the customers table, linking them on the customer ID.

8) Create a foreign key constraint between the sales table and the products table, linking them on the product ID.

```
create table sales(
date_of_sale date,
sales_id int unsigned not null primary key auto_increment,
customer_id int unsigned not null,
product_id int unsigned not null,
employee_id int unsigned not null,
quantity int unsigned not null check(quantity>=0),
FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
FOREIGN KEY (product_id) REFERENCES products(product_id),
FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);
```



Here we have done the 3 queries together as while making the table only we have specified the foreign key constraints of the sales table with those from the customers, products and employees tables.

6) Create a table for purchases, including columns for the date of the purchase, the supplier who provided the product, and the products purchased.
9) Create a foreign key constraint between the purchases table and the suppliers table, linking them on the supplier ID.
10) Create a foreign key constraint between the purchases table and the products table, linking them on the product ID.

```
create table purchases(
date_of_purchase date,
purchases_id int unsigned not null primary key auto_increment,
supplier_id int unsigned not null,
product_id int unsigned not null,
quantity int unsigned not null check(quantity>=0),
FOREIGN KEY (supplier_id) REFERENCES suppliers(supplier_id),
```

```
FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

```
12  13:53:00  create table purchases( date_of_purchase date, purchases_id int unsigned not null primary key auto_increment, ...  0 row(s) affected  0.047 sec
```

Here we have done the 3 queries together as while making the table only we have specified the foreign key constraints of the purchases table with those from the suppliers and products tables.

11) Insert a new product into the products table.

```
insert into products(product_name,description,price,quantity) values ('Bourbon','Biscuit',100,30);
```

```
13  14:02:17  insert into products(product_name,description,price,quantity) values ('Bourbon','Biscuit',100,30)  1 row(s) affected  0.015 sec
```

product_name	product_id	description	price	quantity
Bourbon	1	Biscuit	100.00	30
Cadbury	2	Biscuit	100.00	40
NULL	NULL	NULL	NULL	NULL

40

Here we inserted a new value to products. Since the product_id is auto increment therefore we do not have to insert that and it is handled automatically by the code.

12) Insert a new supplier into the suppliers table.

```
insert into suppliers(supplier_name,address,contact_info) values ('Nestle','Mumbai',5789345673);
```

```
16  14:09:09  insert into suppliers(supplier_name,address,contact_info) values ('Nestle','Mumbai',5789345673)  1 row(s) affected  0.016 sec
```

supplier_name	supplier_id	address	contact_info
Nestle	1	Mumbai	5789345673
NULL	NULL	NULL	NULL

Here again, we inserted a new value this time to suppliers. Here also since supplier_id is auto_increment we do not need to manually insert it into the table.

13) Insert a new customer into the customers table.

```
insert into customers (customer_name,address,contact_info) values('Kshitij','Lucknow',9335300496);
```

```
34  18:11:16  insert into customers (customer_name,address,contact_info) values('Kshitij','Lucknow',9335300496)  1 row(s) affected  0.047 sec
```

customer_name	customer_id	address	contact_info
Kshitij	1	Lucknow	9335300496
NULL	NULL	NULL	NULL

Here again, we inserted a new value this time to customers. Here also since customer_id is auto_increment we do not need to manually insert it into the table.

14) Insert a new employee into the employees table.

```
INSERT INTO employees (employee_name, address, contact_info, job_position) VALUES ('John Smith', '123 Main St, Springfield', '555-1234', 'Manager');
```

```
45  18:13:36  INSERT INTO employees (employee_name, address, contact_info, job_position) VALUES ('John Smith', '123 ...  1 row(s) affected  0.047 sec
```

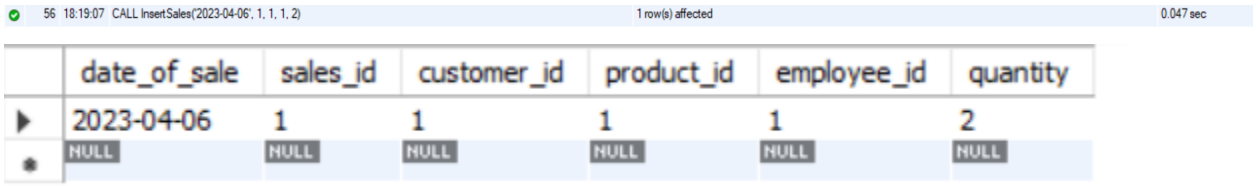
employee_name	employee_id	address	contact_info	job_position
John Smith	1	123 Main St, Springfield	555-1234	Manager
NULL	NULL	NULL	NULL	NULL

Here again, we inserted a new value this time to customers. Here also since customer_id is auto_increment we do not need to manually insert it into the table.

15) Insert a new sale into the sales table, including the customer ID, the product ID, and the date of the sale.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertSales`(IN
date_of_sale date,
IN cid int unsigned,
IN pid int unsigned,
IN eid int unsigned,
IN quan int unsigned)
 READS SQL DATA
```

```
 DETERMINISTIC
 SQL SECURITY INVOKER
 COMMENT 'To handle insertion of a sale'
BEGIN
    DECLARE quan1 int unsigned;
    select quantity from products where product_id=pid into quan1;
    if(quan1>=quan) then
    begin
       insert into sales(date_of_sale,customer_id,employee_id,product_id,quantity)
values (date_of_sale,cid,pid,eid,quan);
    end;
    end if;
    update products set quantity=quan1-quan where product_id=pid;
END$$
DELIMITER ;
CALL InsertSales('2023-04-06', 1, 1, 1, 2);
```



56 18:19:07 CALL InsertSales('2023-04-06', 1, 1, 1, 2) 1 row(s) affected 0.047 sec

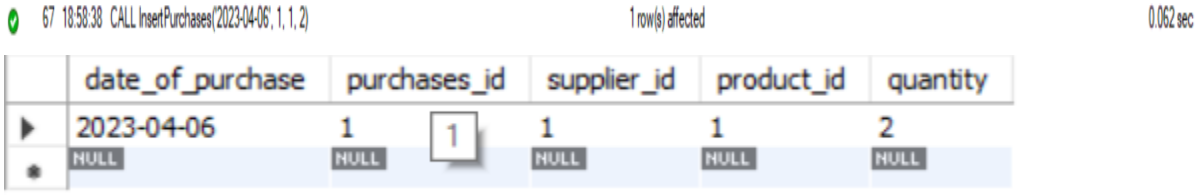
	date_of_sale	sales_id	customer_id	product_id	employee_id	quantity
▶	2023-04-06	1	1	1	1	2
	NULL	NULL	NULL	NULL	NULL	NULL

In this case we need to handle many things since whenever a sale happens we need to correspondingly decrease the quantity of the products left with us and also need to make sure that we only insert the sale if the quantity of teh product being purchased is greater the quantity of the product already available with us. Therefore we have created a procedure for the same where all these things are handled together. Therefore whenever we want to insert a sale, we need to make call to the procedure like: Call InsertSales(Date,CustomerId, ProductId, EmployeeId, Quantity);

16) Insert a new purchase into the purchases table, including the supplier ID, the product ID, and the date of the purchase.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertPurchases`(IN
date_of_purchase date,
```

```
IN sid int unsigned,
IN pid int unsigned,
IN quan int unsigned)
 READS SQL DATA
 DETERMINISTIC
 SQL SECURITY INVOKER
 COMMENT 'To handle insertion of a purchase'
BEGIN
   DECLARE quan1 int unsigned;
   select quantity from products where product_id=pid into quan1;
        insert into purchases(date_of_purchase,supplier_id,product_id,quantity)
values (date_of_purchase,sid,pid,quan);
   update products set quantity=quan1+quan where product_id=pid;
END$$
DELIMITER ;
CALL InsertPurchases('2023-04-06', 1, 1, 2);
```



In this case we need to handle many things since whenever a purchase happens we need to correspondingly increase the quantity of the products left with us. Therefore we have created a procedure for the same where both the things are handled together. Therefore whenever we want to insert a purchase, we need to make call to the procedure like: Call InsertPurchases(Date, SupplierId, ProductId, Quantity);

17) Update the quantity in stock for a specific product in the products table.

```
update products set quantity=60 where product_id=1;
```

product_name	product_id	description	price	quantity
Bourbon	1	Biscuit	100.00	30
Cadbury	2	Biscuit	100.00	38
JimJam	3	Biscuit	130.00	12
Bourbon	4	Drink	100.00	30
Parle	5	Biscuit	100.00	30
Fries	6	Eatables	155.00	30
Banana	7	Fruit	10.00	50
Frooti	8	Cold Drink	20.00	30
Pepsi	9	Cold Drink	40.00	300
Coke	10	Cold Drink	40.00	300
NULL	NULL	NULL	NULL	NULL

Before update

product_name	product_id	description	price	quantity
Bourbon	1	Biscuit	100.00	60
Cadbury	2	Biscuit	100.00	38
JimJam	3	Biscuit	130.00	12
Bourbon	4	Drink	100.00	30
Parle	5	Biscuit	100.00	30
Fries	6	Eatables	155.00	30
Banana	7	Fruit	10.00	50
Frooti	8	Cold Drink	20.00	30
Pepsi	9	Cold Drink	40.00	300
Coke	10	Cold Drink	40.00	300
NULL	NULL	NULL	NULL	NULL

After Update

18) Update the contact information for a specific supplier in the suppliers table.

```
update suppliers set contact_info=9415678901 where suppliers.supplier_id=1;
```

supplier_name	supplier_id	address	contact_info
Nestle	1	Mumbai	5789345673
Cocacola	2	Pune	7786345673
Britanica	3	Delhi	1234567890
Britanica	4	Mumbai	1234568790
Parle	5	Mumbai	1234554321
Parle	6	Ahemdabad	5789345673
ITC	7	Mumbai	9415022732
Grend	8	Lucknow	9020404673
Yurpho	9	Hyderabad	9075643654
NULL	NULL	NULL	NULL

Before update

supplier_name	supplier_id	address	contact_info
Nestle	1	Mumbai	9415678901
Cocacola	2	Pune	7786345673
Britanica	3	Delhi	1234567890
Britanica	4	Mumbai	1234568790
Parle	5	Mumbai	1234554321
Parle	6	Ahemdabad	5789345673
ITC	7	Mumbai	9415022732
Grend	8	Lucknow	9020404673
Yurpho	9	Hyderabad	9075643654
NULL	NULL	NULL	NULL

After Update

19) Update the job position for a specific employee in the employees table.

```
update employees set job_position='Director' where employee_id=1;
```

employee_name	employee_id	address	contact_info	job_position
John Smith	1	123 Main St, Springfield	555-1234	Manager
Jane Doe	2	456 Elm St, New York	555-5678	HR Specialist
Michael Johnson	3	789 Oak St, Los Angeles	555-9876	Software Engineer
Sarah Williams	4	321 Pine St, Chicago	555-5432	Data Analyst
David Lee	5	567 Cedar St, San Francisco	555-6789	Product Manager
Emily Thompson	6	890 Birch St, Miami	555-3456	Marketing Coordinator
Jason Brown	7	432 Maple St, Dallas	555-8765	Sales Associate
Jessica Davis	8	765 Willow St, Atlanta	555-2345	Finance Manager
Daniel Wilson	9	987 Pine St, Boston	555-6780	IT Specialist
Olivia Turner	10	210 Cedar St, Seattle	555-4567	HR Generalist

Before update

employee_name	employee_id	address	contact_info	job_position
John Smith	1	123 Main St, Springfield	555-1234	Director
Jane Doe	2	456 Elm St, New York	555-5678	HR Specialist
Michael Johnson	3	789 Oak St, Los Angeles	555-9876	Software Engineer
Sarah Williams	4	321 Pine St, Chicago	555-5432	Data Analyst
David Lee	5	567 Cedar St, San Francisco	555-6789	Product Manager
Emily Thompson	6	890 Birch St, Miami	555-3456	Marketing Coordinator
Jason Brown	7	432 Maple St, Dallas	555-8765	Sales Associate
Jessica Davis	8	765 Willow St, Atlanta	555-2345	Finance Manager
Daniel Wilson	9	987 Pine St, Boston	555-6780	IT Specialist
Olivia Turner	10	210 Cedar St, Seattle	555-4567	HR Generalist

After Update

20) Delete a specific sale from the sales table based on the sale ID.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteSales`(IN id INT unsigned)
 READS SQL DATA
 DETERMINISTIC
 SQL SECURITY INVOKER
 COMMENT 'To handle delete of a sale'
BEGIN
        DECLARE pid int unsigned;
   DECLARE quan int unsigned;
   DECLARE quan1 int unsigned;
   select product_id, quantity from sales where sales_id=id into pid,quan;
   delete from sales where sales_id=id;
   select quantity from products where product_id=pid into quan1;
   update products set quantity=quan+quan1 where product_id=pid;
END$$
DELIMITER ;
CALL DeleteSales(1);
```

For this query again, we need to keep multiple things in mind. Like we need to make sure that whenever a sale is deleted then the quantity of the product that was sold in that sale is reverted back i.e. added back. Therefore to handle those cases, we made this procedure which can be called by invoking Call DeleteSales(1).

date_of_sale	sales_id	customer_id	product_id	employee_id	quantity
2023-04-06	1	1	1	1	2
2023-04-05	2	2	4	3	3
2023-04-04	3	3	1	2	5
2023-04-03	4	4	3	4	1
2023-04-02	5	5	2	1	4
2023-04-01	6	1	5	3	2
2023-03-31	7	2	4	2	3
2023-03-30	8	3	3	4	1
2023-03-29	9	4	1	1	5
2023-03-28	10	5	2	3	2
NULL	NULL	NULL	NULL	NULL	NULL

Before delete

date_of_sale	sales_id	customer_id	product_id	employee_id	quantity
2023-04-05	2	2	4	3	3
2023-04-04	3	3	1	2	5
2023-04-03	4	4	3	4	1
2023-04-02	5	5	2	1	4
2023-04-01	6	1	5	3	2
2023-03-31	7	2	4	2	3
2023-03-30	8	3	3	4	1
2023-03-29	9	4	1	1	5
2023-03-28	10	5	2	3	2
NULL	NULL	NULL	NULL	NULL	NULL

After Delete

Additional queries:

1)Sales Report

```
Select sales_id, date_of_sale, customer_name, product_name,employee_name,sales.quantity, (sales.quantity*products.price) as TotalAmount from (((employees join sales on employees.employee_id=sales.employee_id) join products on products.product_id=sales.product_id) join customers on customers.customer_id=sales.customer_id);
```

sales_id	date_of_sale	customer_name	product_name	employee_name	quantity	TotalAmount
5	2023-04-02	Withf	Cadbury	John Smith	4	400.00
9	2023-03-29	Rachel	Bourbon	John Smith	5	500.00
3	2023-04-04	Gerth	Bourbon	Jane Doe	5	500.00
7	2023-03-31	Firht	Bourbon	Jane Doe	3	300.00
2	2023-04-05	Firht	Bourbon	Michael Johnson	3	300.00
6	2023-04-01	Kshitij	Parle	Michael Johnson	2	200.00
10	2023-03-28	Withf	Cadbury	Michael Johnson	2	200.00
4	2023-04-03	Rachel	JimJam	Sarah Williams	1	130.00
8	2023-03-30	Gerth	JimJam	Sarah Williams	1	130.00

2) Inventory Report

```
Select * from products;
```

product_name	product_id	description	price	quantity
Bourbon	1	Biscuit	100.00	62
Cadbury	2	Biscuit	100.00	38
JimJam	3	Biscuit	130.00	12
Bourbon	4	Drink	100.00	30
Parle	5	Biscuit	100.00	30
Fries	6	Eatables	155.00	30
Banana	7	Fruit	10.00	50
Frooti	8	Cold Drink	20.00	30
Pepsi	9	Cold Drink	40.00	300
Coke	10	Cold Drink	40.00	300
NULL	NULL	NULL	NULL	NULL

3) Employee Report

```
Select employees.employee_id ,employees.employee_name,employees.address,employees.contact_info,employees.job_position,sum(sales.quantity*products.price) as total_sales from ((products join sales on products.product_id=sales.product_id) right join employees on employees.employee_id=sales.employee_id) group by employees.employee_id;
```

employee_id	employee_name	address	contact_info	job_position	total_sales
1	John Smith	123 Main St, Springfield	555-1234	Director	900.00
2	Jane Doe	456 Elm St, New York	555-5678	HR Specialist	800.00
3	Michael Johnson	789 Oak St, Los Angeles	555-9876	Software Engineer	700.00
4	Sarah Williams	321 Pine St, Chicago	555-5432	Data Analyst	260.00
5	David Lee	567 Cedar St, San Francisco	555-6789	Product Manager	NULL
6	Emily Thompson	890 Birch St, Miami	555-3456	Marketing Coordinator	NULL
7	Jason Brown	432 Maple St, Dallas	555-8765	Sales Associate	NULL
8	Jessica Davis	765 Willow St, Atlanta	555-2345	Finance Manager	NULL
9	Daniel Wilson	987 Pine St, Boston	555-6780	IT Specialist	NULL
10	Olivia Turner	210 Cedar St, Seattle	555-4567	HR Generalist	NULL

FRONTEND

Installation Guide

To install and run the frontend app make sure you have java package installed.
JDBC mysql connector needs to be present as well. You can download it from the internet or we have even provided it in our folder from where you can download it.
Now first run the create table queries in the MySQL workbench i.e. queries 1-10 as given in our SQL file.
After that, in the beginning of our java code specifically line 35,38 and 39 change the URL, user and password according to your respective workbench database.
After that you can simply run the code.
Note: As explained in the video, if you are taking the java file then make sure to include the mysql connector,which is given in the submitted folder, in the java folder, by using the build path. You can also try to download folder and run it directly. But if it does not work then you can do the above thing.

Introduction

This Java code represents a SuperMarket Management System. The system is implemented using Swing for the graphical user interface (GUI) components, and JDBC for database connectivity. The GUI allows users to perform various tasks related to managing products, suppliers, customers, employees, sales, and purchases in a supermarket. The system connects to a MySQL database to store and retrieve data related to these tasks.

Class Structure

The code contains a single class called App, which represents the main entry point of the application. The App class has several member variables, including size, frame, panel, parent, mainPanel, screens, cardLayout, buttons, and buttonNames, which are used to create the GUI components and manage the different screens of the application. The App class also contains several methods that represent the different screens of the application, such as addNewProduct(), addNewSupplier(), addNewCustomer(), addNewEmployee(), addNewSale(), addNewPurchase(), updateProductQuantity(), updateSupplierContactInfo(), updateEmployeeJobPosition(), deleteSaleScreen(), showAllProducts(), showAllSuppliers(), showAllCustomers(), showAllEmployees(), showAllSales(), and showAllPurchases(), which are responsible for creating the different panels and handling user interactions.

Database Connectivity

The App class uses JDBC (Java Database Connectivity) to connect to a MySQL database. The JDBC_DRIVER and DB_URL member variables are used to specify the JDBC driver and the database URL, respectively. The USER and PASS member variables store the username and password for the database connection, respectively. The addNewProduct() method demonstrates how to insert data into the products table in the database using a prepared statement.

GUI Components

The GUI components are created using Swing classes such as JFrame, JPanel, JButton, JTextField, JLabel, FlowLayout, GridLayout, and CardLayout. The mainPanel is used as the main container for all the screens, and the cardLayout is used to manage the different screens and switch between them. The buttons array is used to create buttons for the different tasks in the supermarket management system. The ButtonClickListener class implements the ActionListener interface to handle button clicks and switch to the appropriate screen based on the clicked button. The different screens are created as separate JPanel objects in the addNewProduct(), addNewSupplier(), addNewCustomer(), addNewEmployee(), addNewSale(), addNewPurchase(),

updateProductQuantity(), updateSupplierContactInfo(), updateEmployeeJobPosition(), deleteSaleScreen(), showAllProducts(), showAllSuppliers(), showAllCustomers(), showAllEmployees(), showAllSales(), and showAllPurchases() methods, and they are added to the mainPanel using the CardLayout. The showAllProducts(), showAllSuppliers(), showAllCustomers(), showAllEmployees(), showAllSales(), and showAllPurchases() methods demonstrate how to retrieve data from the database and display it in a JTable component using a DefaultTableModel.