PHARMACY SUPPLY CHAIN MANAGEMENT SYSTEM

UCS2404 - DATABASE MANAGEMENT SYSTEMS

Report

Submitted By

Kavin T(3122225001058)
Krishna varun R(3122225001064)
Krishnaraj N(3122225001065)

Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering
(An Autonomous Institution, Affiliated to Anna University)
Kalavakkam – 603110

2023-24

Why we chose this topic ?

1. **Industry Demand:** There's a high demand for efficient supply chain management systems in the pharmaceutical industry, making this project highly relevant to current industry needs.

2. **Interdisciplinary Knowledge:** The project provides an opportunity to learn about various aspects of supply chain management, healthcare, and database technologies.

3. **Problem-Solving:** Tackling the challenges in this project will enhance your problem-solving skills and ability to work on complex systems.

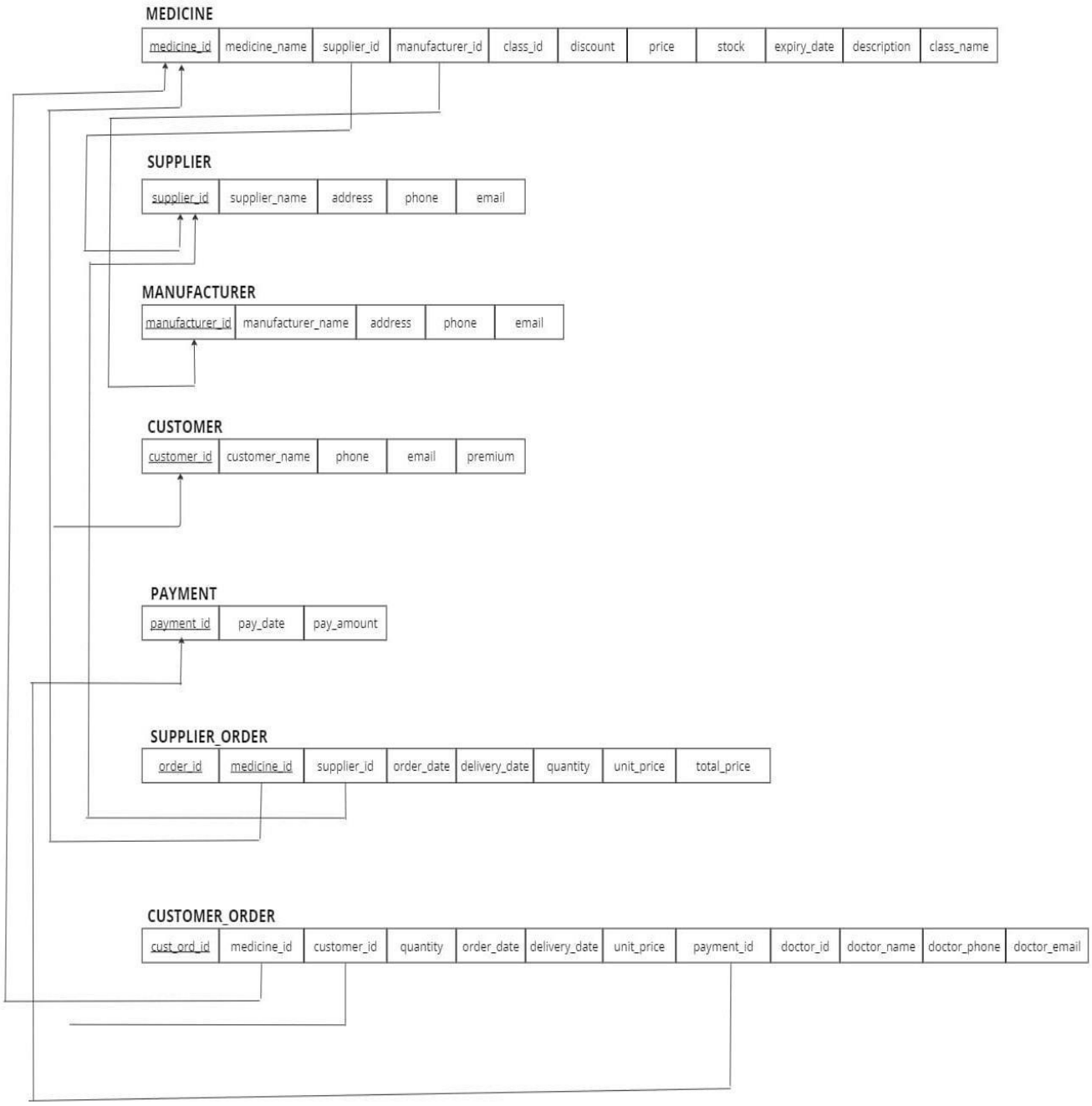
4. **Varied Data Types:** The project involves handling diverse data types, from inventory and order details to customer and supplier information, offering a comprehensive challenge in database management.

UCS2404 - DATABASE MANAGEMENT SYSTEMS
Regulations – R2021

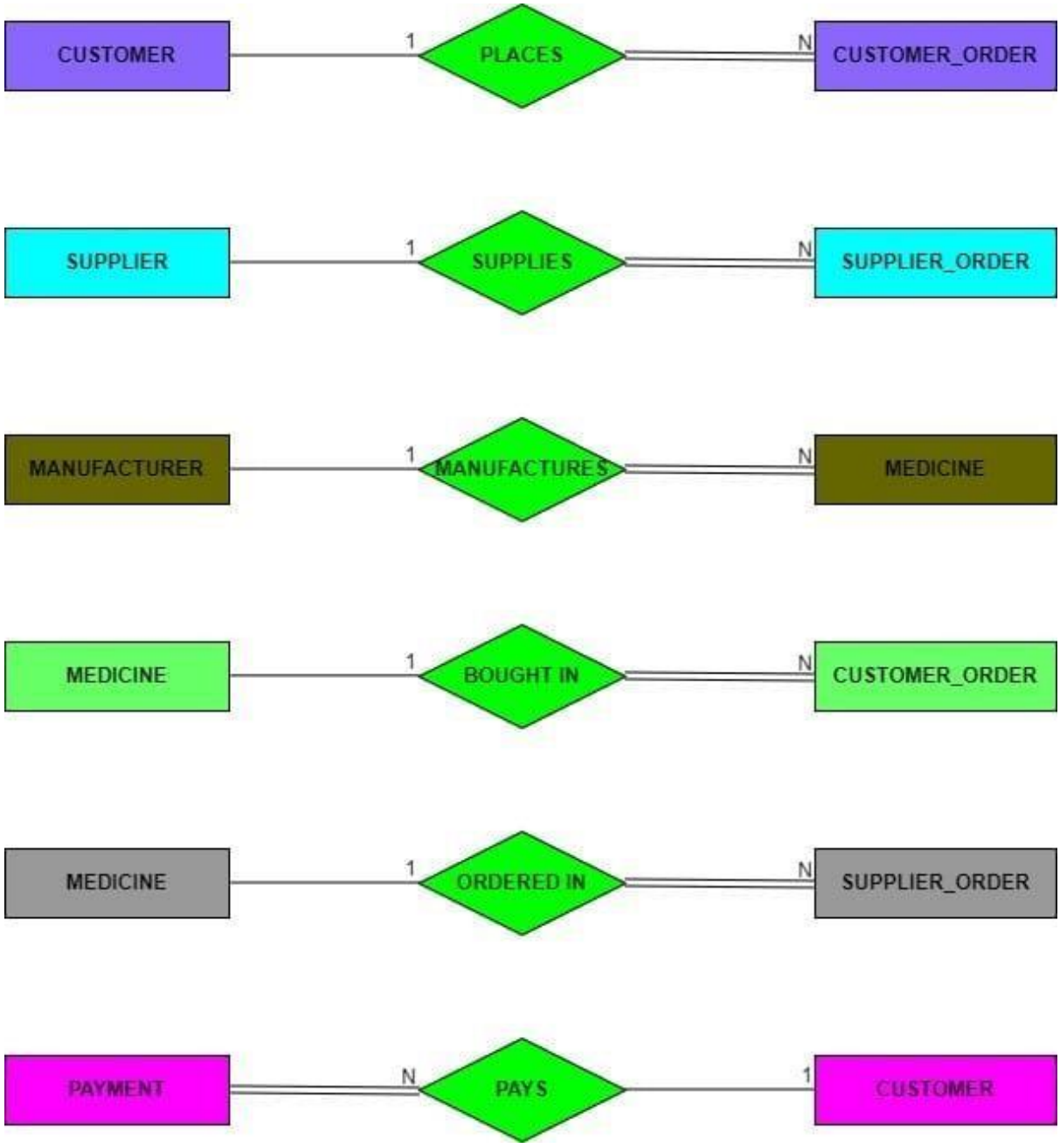
Project Report

Pharmacy supply chain Management system

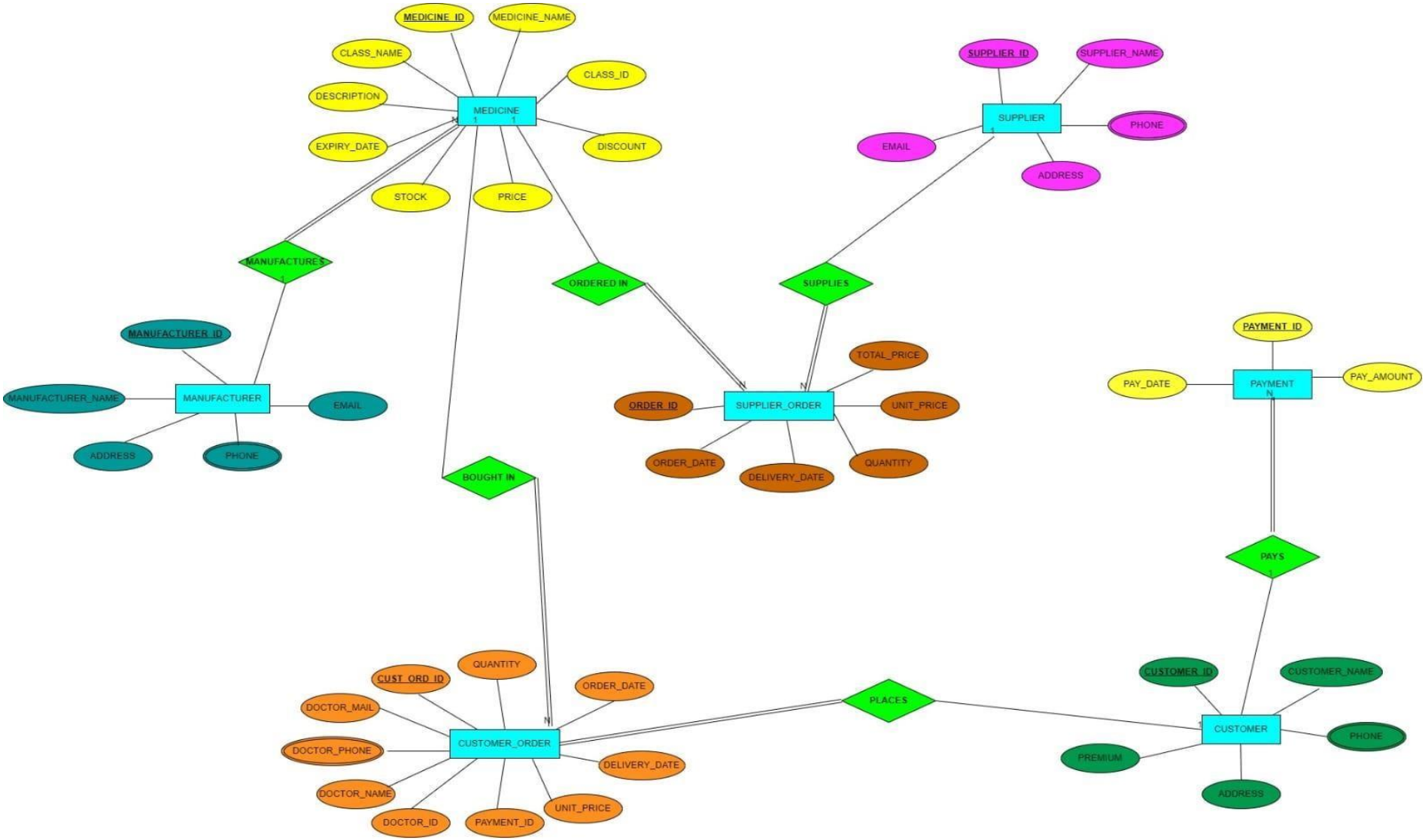
DATABASES:



RELATION:


CUSTOMER
1
PLACES
N
CUSTOMER_ORDER
SUPPLIER
1
SUPPLIES
N
SUPPLIER_ORDER
MANUFACTURER
1
MANUFACTURES
N
MEDICINE
MEDICINE
1
BOUGHT IN
N
CUSTOMER_ORDER
MEDICINE
1
ORDERED IN
N
SUPPLIER_ORDER
PAYMENT
N
PAYS
1
CUSTOMER

ER DIAGRAM:


MEDICINE_ID
MEDICINE_NAME
CLASS_NAME
CLASS_ID
DESCRIPTION
MEDICINE
EXPIRY_DATE
DISCOUNT
STOCK
PRICE
SUPPLIER_ID
SUPPLIER_NAME
SUPPLIER
PHONE
EMAIL
ADDRESS
MANUFACTURES
ORDERED IN
SUPPLIES
MANUFACTURER_ID
MANUFACTURER_NAME
MANUFACTURER
EMAIL
ADDRESS
PHONE
TOTAL_PRICE
ORDER_ID
SUPPLIER_ORDER
UNIT_PRICE
ORDER_DATE
DELIVERY_DATE
QUANTITY
PAYMENT_ID
PAY_DATE
PAYMENT
PAY_AMOUNT
BOUGHT IN
PAYS
QUANTITY
CUST_ORD_ID
ORDER_DATE
DOCTOR_MAIL
PLACES
CUSTOMER_ID
CUSTOMER_NAME
DOCTOR_PHONE
CUSTOMER_ORDER
CUSTOMER
PHONE
DELIVERY_DATE
PREMIUM
DOCTOR_NAME
ADDRESS
DOCTOR_ID
PAYMENT_ID
UNIT_PRICE

Tables :

- Medicine -

1. Medicine_id
2. Medicine_name
3. Supplier_id
4. Manufacturer_id
5. Class_id
6. Stock
7. Expiry_date
8. Discount
9. Price

- Supplier -

1. Supplier_id
2. Supplier_name
3. Email
4. Phone
5. Door_no
6. Street
7. City
8. Pincode

- Customer -

1. Customer_id
2. Customer_name
3. Email
4. Phone
5. Door_no
6. Street
7. City
8. Pincode
9. Premium

❖ Customer Order -

1. Cust_order_id
2. Customer_id
3. Order_date
4. Delivery_date
5. Doctor_id

❖ Doctor -

1. Doctor_id
2. Doctor_name
3. Doctor_phone
4. Doctor_email

❖ Customer Order List -

1. Cust_order_id
2. Medicine_id
3. Quantity
4. Unit_price

❖ Class -

1. Class_id
2. Class_name
3. Description

FUNCTIONAL DEPENDENCIES

Possible FDs :

MEDICINE

Medicine_id → Medicine_name, Supplier_id, Manufacturer_id, discount, price, stock, expiry_date

Class_id → class_name
Supplier_id - > discount,price
Manufactured_id , stock -> expiry_date Class_id,class_name - > description **DECOMPOSE:**

Medicine_id → Medicine_name
Medicine_id → Supplier_id
Medicine_id → Manufacturer_id
Medicine_id → discount
Medicine_id → price
Medicine_id → stock
Medicine_id → expiry_date
Class_id → class_name
Class_id,Class_name - > description
Supplier_id - > discount
Supplier_id - > price
Manufactured_id , stock -> expiry_date **Redundancy:**

1)Medicine_id → Medicine_name
With Medicine_id → Medicine_name
{Medicine_id}$^+$ = { Medicine_name, Supplier_id, Manufacturer_id, discount, price, stock, expiry_date }
Without
{Medicine_id} = { Supplier_id, Manufacturer_id, discount, price, stock, expiry_date }
NOT REDUNDANT

2) Medicine_id → Supplier_id With Medicine_id → Supplier_id
{Medicine_id}$^+$ = { Medicine_name, Supplier_id, Manufacturer_id, discount, price, stock, expiry_date }
Without Medicine_id → Supplier_id
{Medicine_id}$^+$ = { Medicine_name, Manufacturer_id, discount, price, stock, expiry_date }
NOT REDUNDANT

3) Medicine_id → Manufacturer_id
With Medicine_id → Manufacturer_id
{Medicine_id}$^+$ = { Medicine_name, Supplier_id, Manufacturer_id, discount, price, stock, expiry_date }
Without Medicine_id → Manufacturer_id

{Medicine_id}$^+$ = { Medicine_name, Supplier_id,discount, price, stock, expiry_date }

NOT REDUNDANT

4) Medicine_id → discount

With Medicine_id → discount

{Medicine_id}$^+$ = { Medicine_name, Supplier_id, Manufacturer_id, discount, price, stock, expiry_date }

Without Medicine_id → discount

{Medicine_id}$^+$ = { Medicine_name, Supplier_id, Manufacturer_id, discount , price, stock, expiry_date }

REDUNDANT

5) Medicine_id → price

With Medicine_id → price

{Medicine_id}$^+$ = { Medicine_name, Supplier_id, Manufacturer_id, discount, price, stock, expiry_date }

Without Medicine_id → price

{Medicine_id}$^+$ = { Medicine_name, Supplier_id, Manufacturer_id, discount, stock, price,expiry_date }

REDUNDANT

6) Medicine_id → stock

With Medicine_id → stock

{Medicine_id}$^+$ = { Medicine_name, Supplier_id, Manufacturer_id, discount, price, stock, expiry_date }

Without Medicine_id → stock

{Medicine_id}$^+$ = { Medicine_name, Supplier_id, Manufacturer_id, discount, price, expiry_date }

7) Medicine_id → expiry_date

With Medicine_id → expiry_date

{Medicine_id}$^+$ = { Medicine_name, Supplier_id, Manufacturer_id, discount, price, stock, expiry_date }

Without Medicine_id → expiry_date

{Medicine_id}$^+$ = { Medicine_name, Supplier_id, Manufacturer_id, discount, price, stock,expiry_date}

REDUNDANT

8)Class_id → class_name

BOTH ARE NOT REDUNDANT

9) Supplier_id - > discount

With, {Supplier_id} = {discount,price}

Without , {Supplier_id} = {price}

NOT REDUNDANT

10)Supplier_id - > price

With, {supplier_id } = {discount,price}

Without , { Supplier_id } = {discount }

NOT REDUNDANT

11)Manufactured_id , stock -> expiry_date

NOT REDUNDANT

12)Class_id,Class_name - > description

NOT REDUNDANT

FDs = {

Medicine_id → Medicine_name, Supplier_id, Manufacturer_id, stock

Class_id → Class_name

Class_id,Class_name - > description

Supplier_id - > discount,price

Manufacturer_id , stock -> expiry_date

}

3) Extraneous Attributes:

1. Class_id,Class_name - > description {Class_id}= {Class_name }

Class_name belongs to {class_id}

Class_name is extraneous

FD = {

Medicine_id → Medicine_name, Supplier_id, Manufacturer_id, stock

Class_id → Class_name

Class_id - > description

Supplier_id - > discount,price

Manufacturer_id , stock -> expiry_date

}

2. Manufacturer_id , stock -> expiry_date

{Manufacturer_id} = {}

Not extraneous

{stock} = {}

Not extraneous

FINAL FDs = {

Medicine_id → Medicine_name, Supplier_id, Manufacturer_id, stock

Class_id → Class_name

Class_id - > description

Supplier_id - > discount,price

Manufacturer_id , stock -> expiry_date

}

R (Medicine_id , Medicine_name , Supplier_id , Manufacturer_id , Stock ,

Class_id , Class_name , Description , Expiry Date)
{Medicine_id , Class_id } is the super Key
PA = {Medicine_id , Class_id }
It is also the Candidate key
It is in 1NF.

CHECK 2NF:

Proper subset {Medicine_id} has partial Dependency PA - > NPA
{Class_id } has partial dependency too PA -> NPA

R1 (Medicine_id , Medicine_name , Supplier_id , Manufacturer_id , Stock , discount , price , expiry_date)

FDs : Medicine_id → Medicine_name, Supplier_id, Manufacturer_id, stock
Supplier_id - > discount,price
Manufacturer_id , stock -> expiry_date

Now it is in 2NF.

R2 (Class_id , Class_name , Description)

FDs: Class_id → Class_name
Class_id - > description

Now it is in 2NF

R3 (Medicine_id , Class_id)

Now it is in 2NF.

CHECK 3NF:

R1 (Medicine_id , Medicine_name , Supplier_id , Manufacturer_id , Stock , discount , price , expiry_date)

FDs : Medicine_id → Medicine_name, Supplier_id, Manufacturer_id, stock
Supplier_id - > discount,price
Manufacturer_id , stock -> expiry_date

Supplier_id - > discount,price

NPA -> NPA
TD

R11 (Supplier_id , Discount , Price)
FD: Supplier_id - > discount,price
LHS is superkey
It is in 3NF and BCNF

Manufacturer_id , stock -> expiry_date

NPA - > NPA
TD

R12 (Manufacturer_id , Stock ,expiry_date)
FD: Manufacturer_id , stock -> expiry_date
LHS is superKey .
It is in 3NF and BCNF.

R13 (Medicine_id , Medicine_name , Supplier_id , Manufacturer_id , Stock)

FD : Medicine_id → Medicine_name, Supplier_id, Manufacturer_id, stock

LHS is SuperKey

It is in 3NF and BCNF.

R2 (Class_id , Class_name , Description)

FDs: Class_id → Class_name

`Class_id - > description`

LHS is Superkey

It is in 3NF and BCNF.

R3 (Medicine_id , Class_id)

It is in 3NF and BCNF.

SUPPLIER:

Possible FDs:

supplier_id → supplier_name,email , phone supplier_id, email →

address

supplier_id, phone → email **Decomposition** supplier_id

→ supplier_name supplier_id → email supplier_id →

phone supplier_id, email → address (redundant) sup-

plier_id, phone → email (extraneous attribute)

Redundancy Check

1.supplier_id → supplier_name

Compute the closure of {supplier_id}:

{supplier_id}+ = {supplier_id, supplier_name}

As supplier_name ∉ {supplier_id}+, it is not redundant.

2.supplier_id → email

Compute the closure of {supplier_id}:

{supplier_id}+ = {supplier_id, email}

As email ∉ {supplier_id}+, it is not redundant.

3.supplier_id → phone

Compute the closure of {supplier_id}:

{supplier_id}+ = {supplier_id, supplier_name, phone}

As phone ∉ {supplier_id}+, it is not redundant.

4.supplier_id, email → address

Compute the closure of {supplier_id}:

{supplier_id}+ = {supplier_id, supplier_name, phone, email}

Since {supplier_id}+ already includes email, let's see if we need email in the FD:

Compute the closure of {supplier_id} without email:

{supplier_id}+ = {supplier_id, supplier_name, phone, address}

(considering supplier_id alone should determine address if email is redundant)
Since address ∉ {supplier_id}+, email is needed.

However, let's assume that we have:

supplier_id → address

So, supplier_id, email → address is redundant because supplier_id alone should be able to determine address.

EXTRANEOUS:

supplier_id, phone → email

Check if phone is extraneous:

Compute the closure of {supplier_id}:

{supplier_id}+ = {supplier_id, supplier_name, phone, address}

Compute the closure of {supplier_id, phone} without phone:

{supplier_id, phone}+ = {supplier_id, supplier_name, phone, email}

Since email ∈ {supplier_id}+, phone is extraneous.

Simplified FDs:

supplier_id → supplier_name

supplier_id → email supplier_id →

phone supplier_id → address

{supplier_id } is the only Candidate key

CUSTOMER:

Given FDs: customer_id → cus-

tomer_name customer_id → email

customer_id → phone customer_id

→ premium customer_id, email →

address customer_id, phone →

email **Decomposition** customer_id

→ customer_name customer_id →

email customer_id → phone cus-

tomer_id → premium customer_id,

email → address customer_id,

phone → email

Redundancy Check

1.customer_id → customer_name

Compute the closure of {customer_id}:

{customer_id}+ = {customer_id, customer_name}

As customer_name ∉ {customer_id}+, it is not redundant.

2.customer_id → email

Compute the closure of {customer_id}:

{customer_id}+ = {customer_name, id}

As email ∉ {customer_id}+, it is not redundant.

3.customer_id → phone

Compute the closure of {customer_id}:

{customer_id}+ = {customer_id, customer_name, phone}

As phone ∉ {customer_id}+, it is not redundant.

4.customer_id → premium

Compute the closure of {customer_id}:

{customer_id}+ = {customer_id, customer_name, phone, premium}

As premium ∉ {customer_id}+, it is not redundant.

5.customer_id, email → address

Compute the closure of {customer_id}:

{customer_id}+ = {customer_id, customer_name, phone, email, address}

Since {customer_id}+ already includes email, let's see if we need email in the FD:

Compute the closure of {customer_id} without email:

{customer_id}+ = {customer_id, customer_name, phone, address} (considering customer_id alone should determine address if email is redundant)

Since address ∉ {customer_id}+, email is needed.

However, let's assume that we have:

customer_id → address

So, customer_id, email → address is redundant because customer_id alone should be able to determine address.

EXTRANEOUS:

1.customer_id, phone → email

Check if phone is extraneous:

Compute the closure of {customer_id}:

{customer_id}+ = {customer_id, customer_name, phone, email, address}

Compute the closure of {customer_id, phone} without phone:

{customer_id, phone}+ = {customer_id, customer_name, phone, email, address}

Since email ∈ {customer_id}+, phone is extraneous.

Simplified FDs:

customer_id → customer_name cus-

tomer_id → email customer_id →

phone customer_id → premium cus-

tomer_id → address

{customer_id } is the only candidate key.

NORMALIZATION: SUPPLIER:

supplier_id → supplier_name supplier_id →

email supplier_id → phone supplier_id → ad-

dress

{supplier_id } is the only Candidate key **1NF:**

As address has Door_no,Street,City and pincode as attributes 1NF is violated. Table will be

Supplier_id	Supplier_name	Email	Phone	Door_no	Street	City	Pincode

2NF:

{supplier_id } is the super key and the only candidate key. No partial dependencies So , 2NF is Satisfied.

3NF:

LHS (supplier_id) is the super key No transitive dependency and so , it satisfies both 3NF and BCNF.

CUSTOMER:

customer_id → customer_name cus-

tomer_id → email customer_id →

phone customer_id → premium cus-

tomer_id → address

{Customer_id } is the only Candidate key **1NF:**

As address has Door_no,Street,City and pincode as attributes 1NF is violated. Table will be

Customer_id	Customer_name	Email	Phone	Door_no	Street	City	Pincode	Premium

2NF:

{Customer_id } is the super key and the only candidate key. No partial dependencies So , 2NF is Satisfied.

3NF:

LHS (Customer_id) is the super key No transitive dependency and so , it satisfies both 3NF and BCNF.

CUSTOMER ORDER :

Possible FDs : {cust_order_id -> customer_id, order_date, delivery_date, doctor_id, doctor_name, doctor_phone, doctor_email; doctor_id -> doctor_name, doctor_phone, doctor_email; cust_order_id, medicine_id ->quantity, unit_price }

DECOMPOSE :

FDs : {cust_order_id -> customer_id;
cust_order_id -> order_date;
cust_order_id -> delivery_date;
cust_order_id -> doctor_id; cust_order_id -> doctor_name; cust_order_id -> doctor_phone; cust_order_id -> doctor_email; doctor_id -> doctor_name; doctor_id -> doctor_phone; doctor_id -> doctor_email; cust_order_id, medicine_id ->quantity; cust_order_id, medicine_id ->unit_price} **REDUNDANCY :**

a. cust_order_id -> customer_id

   Without it $\{cust\_order\_id\}^+$={cust_order_id, order_date, delivery_date, doctor_id, doctor_name, doctor_phone, doctor_email}

   As customer_id $\notin \{cust\_order\_id\}^+$, it is not redundant

b. cust_order_id -> order_date

   Without it $\{cust\_order\_id\}^+$={cust_order_id, customer_id, delivery_date, doctor_id, doctor_name, doctor_phone, doctor_email}

   As order_date $\notin \{cust\_order\_id\}^+$, it is not redundant

c. cust_order_id -> delivery_date

   Without it $\{cust\_order\_id\}^+$={cust_order_id,customer_id, order_date, doctor_id, doctor_name, doctor_phone, doctor_email}

   As delivery_date $\notin \{cust\_order\_id\}^+$, it is not redundant

d. cust_order_id -> doctor_id

Without it $\{cust\_order\_id\}^+$={cust_order_id,customer_id, order_date, delivery_date, doctor_name, doctor_phone, doctor_email}

As doctor_id $\notin \{cust\_order\_id\}^+$, it is not redundant

e. cust_order_id -> doctor_name

Without it $\{cust\_order\_id\}^+$={cust_order_id,customer_id, order_date, delivery_date, doctor_id, doctor_name, doctor_phone, doctor_email}

As doctor_name $\in \{cust\_order\_id\}^+$, it is redundant
So we can remove it from FD

f. cust_order_id -> doctor_phone

Without it $\{cust\_order\_id\}^+$={cust_order_id,customer_id, order_date, delivery_date, doctor_id, doctor_name, doctor_phone, doctor_email}

As doctor_phone $\in \{cust\_order\_id\}^+$, it is redundant
So we can remove it from FD

g. cust_order_id -> doctor_email

Without it $\{cust\_order\_id\}^+$={cust_order_id,customer_id, order_date, delivery_date, doctor_id, doctor_name, doctor_phone, doctor_email}

As doctor_email $\in \{cust\_order\_id\}^+$, it is redundant
So we can remove it from FD

h. doctor_id -> doctor_name

Without it $\{doctor\_id\}^+$={doctor_phone, doctor_email}

As doctor_name $\notin \{doctor\_id\}^+$, it is not redundant

i. doctor_id -> doctor_phone

Without it {doctor_id}$^+$={doctor_name, doctor_email}
As doctor_phone $\notin$ {doctor_id}$^+$, it is not redundant

j. doctor_id -> doctor_email

Without it {doctor_id}$^+$={doctor_phone, doctor_name}
As doctor_email $\notin$ {doctor_id}$^+$, it is not redundant

k. cust_order_id, medicine_id ->quantity

Without it {cust_order_id, medicine_id}$^+$={ cust_order_id, medicine_id , customer_id, order_date, delivery_date, doctor_id, doctor_name, doctor_phone, doctor_email, unit_price}
As quantity $\notin$ { cust_order_id, medicine_id}$^+$ so it is not redundant

l. cust_order_id, medicine_id -> unit_price

Without it {cust_order_id, medicine_id}$^+$={ cust_order_id, medicine_id , customer_id, order_date, delivery_date, doctor_id, doctor_name, doctor_phone, doctor_email, quantity}
As unit_price $\notin$ { cust_order_id, medicine_id}$^+$ so it is not redundant

Extravaneous Attributes :

a. cust_order_id, medicine_id ->quantity, unit_price

It is not an extravaneous attributes

Final FDs = { cust_order_id -> customer_id; cust_order_id -> order_date; cust_order_id -> delivery_date; cust_order_id -> doctor_id; doctor_id -> doctor_name; doctor_id -> doctor_phone; doctor_id -> doctor_email; cust_order_id, medicine_id ->quantity; cust_order_id, medicine_id ->unit_price}

Here, there is only one candidate key i.e {cust_order_id, medicine_id}

Prime Attributes = { cust_order_id, medicine_id } Remaining are non prime attributes

Check for 1NF :

As there is no multi valued or composite attributes it satisfices

1NF

Check for 2NF :

As one FD cust_order_id -> customer_id, order_date, delivery_date, doctor_id is a partial dependency violate 2NF i.e PA->NPA so, we need to divide it into two relations

R1 : Customer_order(cust_order_id, customer_id, order_date , delivery_date, doctor_id, doctor_name, doctor_phone, doctor_email)
With FD = { cust_order_id -> customer_id, order_date, delivery_date, doctor_id;
doctor_id -> doctor_name, doctor_phone, doctor_email}
with cust_order_id as only candidate key
PA = {Φ}

R2 : Customer_order_list(cust_order_id, medicine_id, quantity, unit_price)
With FD = { cust_order_id, medicine_id ->quantity, unit_price}
With {cust_order_id, medicine_id} as only candidate key
PA ={ cust_order_id, medicine_id} and Cust_order_id will be refered from
R1

Check for 3NF :

In R1, doctor_id -> doctor_name, doctor_phone, doctor_email is a transitive dependency violates 3NF as NPA->NPA
So, we need to separate it into two relations
R11 : Doctor(doctor_id, doctor_name, doctor_phone, doctor_email)
With FD = { doctor_id -> doctor_name, doctor_phone, doctor_email}
With doctor_id as only candidate key
R12 : Customer_order(cust_order_id, customer_id, order_date , delivery_date, doctor_id)
With FD = cust_order_id -> customer_id, order_date, delivery_date, doctor_id
With cust_order_id as its only candidate key and doctor_id will be referenced from R11
R2 : Customer_order_list(cust_order_id, medicine_id, quantity, unit_price)
With FD = { cust_order_id, medicine_id ->quantity, unit_price}
With {cust_order_id, medicine_id} as only candidate key

PA ={ cust_order_id, medicine_id} and Cust_order_id will be refered from
R12

Check for BCNF:

All relations follows BCNF as it has super key in lhs

So, finally the relations are

R11 : Doctor(doctor_id, doctor_name, doctor_phone, doctor_email)
With FD = { doctor_id -> doctor_name, doctor_phone, doctor_email}
With doctor_id as only candidate key

R12 : Customer_order(cust_order_id, customer_id, order_date , delivery_date, doctor_id)
With FD = cust_order_id -> customer_id, order_date, delivery_date, doctor_id
With cust_order_id as its only candidate key and doctor_id will be referenced from R11

R2 : Customer_order_list(cust_order_id, medicine_id, quantity, unit_price)
With FD = { cust_order_id, medicine_id ->quantity, unit_price}
With {cust_order_id, medicine_id} as only candidate key
PA ={ cust_order_id, medicine_id} and Cust_order_id will be refered from R12

PAYMENT :

Possible FD : {payment_id->pay_date, pay_amount}

Decomposition :

FD = {payment_id->pay_date; Payment_id->pay_amount}

Redundancy :

a. payment_id -> pay_date

Without it $\{payment\_id\}^{+}$={payment_id, pay_amount}
So it is not redundant

b. payment_id -> pay_amount

Without it $\{payment\_id\}^{+}$={payment_id, pay_date} So it is not redundant **Extravaneous :**

There are no extravaneous variables **Normalisation :**

Payment_id is the only candidate key

So PA = {}

So it satisfies 1NF as it has no multivalued attributes

It satisfies 2NF as there is no Prime Attributes

It satisfies 3NF as there is no NPA->NPA

It satisfies BCNF as all FD has super key in LHS

So, finally the FDs are

FD = {payment_id->pay_date;

Payment_id->pay_amount}

MANUFACTURER –

F1 = {

MANUFACTURER_ID -> MANUFACTURER_NAME , ADDRESS, PHONE , EMAIL }

DECOMPOSE –

MANUFACTURER_ID -> MANUFACTURER_NAME, ADDRESS, PHONE, EMAIL

MANUFACTURER_ID -> MANUFACTURER_NAME

MANUFACTURER_ID -> ADDRESS

MANUFACTURER_ID -> PHONE

MANUFACTURER_ID -> EMAIL

F1 = {

MANUFACTURER_ID -> MANUFACTURER_NAME

MANUFACTURER_ID -> ADDRESS

MANUFACTURER_ID -> PHONE

MANUFACTURER_ID -> EMAIL

}

EXTRANEOUS –

THERE IS NO EXTRANEOUS IN THESE FD'S

REDUNDANCY –

1. CHECKING IF MANUFACTURER_ID -> MANUFACTURER_NAME IS REDUNDANT

{ MANUFACTURER_ID } = { MANUFACTURER_ID, ADDRESS, PHONE, EMAIL}

{ MANUFACTURER_NAME} IS NOT IN { MANUFACTURER_ID, ADDRESS, PHONE,
EMAIL}

NOT REDUNDANT.

2. CHECKING IF MANUFACTURER_ID -> ADDRESS IS REDUNDANT

{ MANUFACTURER_ID } = { MANUFACTURER_ID, MANUFACTURER_NAME, PHONE, EMAIL}

{ADDRESS} IS NOT IN {MANUFACTURER_ID, MANUFACTURER_NAME, PHONE, EMAIL}

NOT REDUNDANT.

3. CHECKING IF MANUFACTURER_ID -> PHONE IS REDUNDANT

{ MANUFACTURER_ID } = { MANUFACTURER_ID, MANUFACTURER_NAME, ADDRESS, EMAIL}

{PHONE} IS NOT IN {MANUFACTURER_ID, MANUFACTURER_NAME, ADDRESS, EMAIL}

NOT REDUNDANT.

4. CHECKING IF MANUFACTURER_ID -> EMAIL IS REDUNDANT

{ MANUFACTURER_ID } = { MANUFACTURER_ID, MANUFACTURER_NAME, ADDRESS, PHONE}

{EMAIL} IS NOT IN {MANUFACTURER_ID, MANUFACTURER_NAME, ADDRESS, PHONE}

NOT REDUNDANT.

FINAL FD = {
MANUFACTURER_ID -> MANUFACTURER_NAME
MANUFACTURER_ID -> ADDRESS
MANUFACTURER_ID -> PHONE
MANUFACTURER_ID -> EMAIL

}

AS {MANUFACTURER_ID} -> { MANUFACTURER_ID, MANUFACTURER_NAME, AD-
DRESS,
PHONE,EMAIL}
IT IS THE ONLY CANDIDATE KEY.

CHECKING FOR 1NF –
THIS RELATION HAS MULTI-VALUED ATTRIBUTES (PHONE).
THEREFORE, IT IS NOT IN 1NF.

WE HAVE TO DIVIDE THE RELATION INTO TWO.
R1 : MANUFACTURER_ID, MANUFACTURER_NAME , ADDRESS, EMAIL
R2 : MANUFACTURER_ID, PHONE
BY DOING THIS, IT FOLLOWS
1NF .

CHECKING FOR 2NF –
R1 : MANUFACTURER_ID, MANUFACTURER_NAME , ADDRESS, EMAIL
FD'S : MANUFACTURER_ID -> MANUFACTURER_NAME
MANUFACTURER_ID -> ADDRESS
MANUFACTURER_ID -> EMAIL
{ MANUFACTURER_ID}$^+$ = { MANUFACTURER_ID, MANUFACTURER_NAME , AD-
DRESS,
EMAIL}
MANUFACTURER_ID IS THE PRIMARY ATTRIBUTE AND HAS NO PROPER SUBSET.
THEREFORE, IT IS IN 2NF

R2 : MANUFACTURER_ID, PHONE
FD'S : MANUFACTURER_ID -> PHONE {
MANUFACTURER_ID}$^+$ = { MANUFACTURER_ID,
PHONE} IT IS PA AND HAS NO PROPER SUBSET.
THEREFORE, IT IS IN 2NF.

CHECKING FOR 3NF –
R1 : MANUFACTURER_ID, MANUFACTURER_NAME , ADDRESS, EMAIL
FD'S : MANUFACTURER_ID -> MANUFACTURER_NAME
MANUFACTURER_ID -> ADDRESS
MANUFACTURER_ID -> EMAIL
LHS IS A SUPER KEY AND HAS NO TD(NPA -> NPA)
THEREFORE, IT IS IN 3NF AND ALSO BCNF.

R2 : MANUFACTURER_ID, PHONE
FD'S : MANUFACTURER_ID -> PHONE
LHS IS A SUPER KEY AND HAS NO TD(NPA ->
NPA) THEREFORE, IT IS IN 3NF AND ALSO
BCNF.

SUPPLIER_ORDER –

```
F1 = {
   ORDER_ID -> ORDER_DATE , DELIVERY_DATE, QUANTITY, TOTAL_PRICE
    MEDICINE_ID -> SUPPLIER_ID, UNIT_PRICE
    ORDER_ID, QUANTITY -> TOTAL_PRICE
}
```

DECOMPOSE –

1. ORDER_ID -> ORDER_DATE, DELIVERY_DATE
   ORDER_ID -> ORDER_DATE
   ORDER_ID -> DELIVERY_DATE
   ORDER_ID ->QUANTITY
   ORDER_ID -> TOTAL_PRICE

2. MEDICINE_ID -> SUPPLIER_ID, UNIT_PRICE
   MEDICINE_ID -> SUPPLIER_ID
   MEDICINE_ID -> UNIT_PRICE

```
F1 = {
      ORDER_ID -> ORDER_DATE
      ORDER_ID -> DELIVERY_DATE
      ORDER_ID ->QUANTITY
      ORDER_ID -> TOTAL_PRICE
      MEDICINE_ID -> SUPPLIER_ID
      MEDICINE_ID -> UNIT_PRICE
      ORDER_ID, QUANTITY -> TOTAL_PRICE
}
```

EXTRANEOUS –

1. ORDER_ID, QUANTITY -> TOTAL_PRICE

   CHECKING IF ORDER_ID IS EXTRANEOUS
   QUANTITY -> TOTAL_PRICE

{ QUANTITY } = { QUANTITY, TOTAL_PRICE }

{ORDER_ID} DOES NOT BELONGS TO {QUANTITY, TOTAL_PRICE }

IT IS NOT EXTRANEOUS

CHECKING IF QUANTITY IS EXTRANEOUS
ORDER_ID -> TOTAL_PRICE

{ORDER_ID } ={ ORDER _ID, ORDER_DATE , DELIVERY_DATE, QUANTITY,
TOTAL_PRICE }

QUANTITY BELONGS TO {ORDER _ID, ORDER_DATE , DELIVERY_DATE,
QUANTITY, TOTAL_PRICE}

IT IS EXTRANEOUS.

F1 = {
ORDER_ID -> ORDER_DATE
ORDER_ID -> DELIVERY_DATE
ORDER_ID ->QUANTITY
ORDER_ID -> TOTAL_PRICE
MEDICINE_ID -> SUPPLIER_ID
MEDICINE_ID -> UNIT_PRICE
}

REDUNDANCY –

1. CHECKING IF ORDER_ID -> ORDER_DATE IS REDUNDANT
   {ORDER_ID= {ORDER_ID, DELIVERY_DATE, QUANTITY, TOTAL_PRICE}
   {ORDER_DATE} DOES NOT BELONGS TO {ORDER_ID, DELIVERY_DATE,
   QUANTITY, TOTAL_PRICE}

   SO, IT IS NOT REDUNDANT.

2. CHECKING IF ORDER_ID -> DELIVERY_DATE IS REDUNDANT
   {ORDER_ID } = {ORDER_ID, ORDER_DATE, QUANTITY, TOTAL_PRICE}
   {DELIVERY_DATE} DOES NOT BELONGS TO {ORDER_ID, ORDER_DATE,
   QUANTITY, TOTAL_PRICE}

```
      SO, IT IS NOT REDUNDANT.

   3. CHECKING IF ORDER_ID -> QUANTITY IS REDUNDANT
      {ORDER_ID } = {ORDER_ID, ORDER_DATE, DELIVERY_DATE, TO-
      TAL_PRICE}
      {QUANTITY} DOES NOT BELONGS TO {ORDER_ID, ORDER_DATE, QUAN-
      TITY,
      TOTAL_PRICE}

      SO, IT IS NOT REDUNDANT.

   4. CHECKING IF ORDER_ID -> TOTAL_PRICE IS REDUNDANT
      {ORDER_ID } = {ORDER_ID, ORDER_DATE, DELIVERY_DATE, QUANTITY
      }

      {TOTAL_PRICE} DOES NOT BELONGS TO {ORDER_ID,
      ORDER_DATE,DELIVERY_DATE, QUANTITY}

      SO, IT IS NOT REDUNDANT.

   5. CHECKING IF MEDICINE_ID -> SUPPLIER_ID IS REDUNDANT
      {MEDICINE_ID} = {MEDICINE_ID, UNIT_PRICE}

      {SUPPLIER_ID} DOES NOT BELONGS TO { MEDICINE_ID, UNIT_PRICE}

      SO, IT IS NOT REDUNDANT.

   6. CHECKING IF MEDICINE_ID -> UNIT_PRICE IS REDUNDANT
      {MEDICINE_ID} = {MEDICINE_ID, SUPPLIER_ID}

      {UNIT_PRICE} DOES NOT BELONGS TO { MEDICINE_ID, SUPPLIER_ID}

      SO, IT IS NOT REDUNDANT.

FINAL FD = {
      ORDER_ID -> ORDER_DATE
      ORDER_ID -> DELIVERY_DATE
      ORDER_ID ->QUANTITY
      ORDER_ID -> TOTAL_PRICE
      MEDICINE_ID -> SUPPLIER_ID
      MEDICINE_ID -> UNIT_PRICE
}
```

AS {ORDER_ID, MEDICINE_ID} -> { ORDER_DATE , DELIVERY_DATE, QUANTITY,
UNIT_PRICE, SUPPLIER_ID, TOTAL_PRICE}
IT IS THE ONLY CANDIDATE KEY.

CHECKING FOR 1NF –
AS THERE IS NO COMPOSITE AND MULTI-VALUED ATTRIBUTES, THESE FD'S ARE IN 1NF.

CHECKING FOR 2NF –
RELATION R (ORDER_ID, ORDER_DATE, DELIVERY_DATE, QUANTITY, TOTAL_PRICE, MEDICINE_ID -> SUPPLIER_ID, UNIT_PRICE)
HAS PARTIAL DEPENDENCY WHICH VIOLATES 2NF (THAT IS : PA -> NPA). SO, WE NEED TO DIVIDE INTO TWO RELATIONS.

R1: (ORDER_ID, ORDER_DATE, DELIVERY_DATE, QUANTITY, TOTAL_PRICE)
FD'S : ORDER_ID -> ORDER_DATE
ORDER_ID -> DELIVERY_DATE
ORDER_ID ->QUANTITY
ORDER_ID -> TOTAL_PRICE

$\{ORDER\_ID\}^+$ = { ORDER_ID, ORDER_DATE, DELIVERY_DATE, QUANTITY, TOTAL_PRICE}
ORDER_ID IS THE PRIMARY ATTRIBUTE AND IT HAS NO PARTIAL SUBSET.
THEREFORE, IT IS IN 2NF

R2: (MEDICINE_ID -> SUPPLIER_ID, UNIT_PRICE)
FD'S : MEDICINE_ID -> SUPPLIER_ID
MEDICINE_ID -> SUPPLIER_ID
$\{MEDICINE\_ID\}^+$ = { MEDICINE_ID, SUPPLIER_ID, UNIT_PRICE}
MEDICINE_ID IS THE PRIMARY ATTRIBUTE AND IT HAS NO PARTIAL SUBSET.
THEREFORE, IT IS IN 2NF

CHECKING FOR 3NF –
R1: (ORDER_ID, ORDER_DATE, DELIVERY_DATE, QUANTITY, TOTAL_PRICE)
FD'S : ORDER_ID -> ORDER_DATE
ORDER_ID -> DELIVERY_DATE
ORDER_ID ->QUANTITY
ORDER_ID -> TOTAL_PRICE

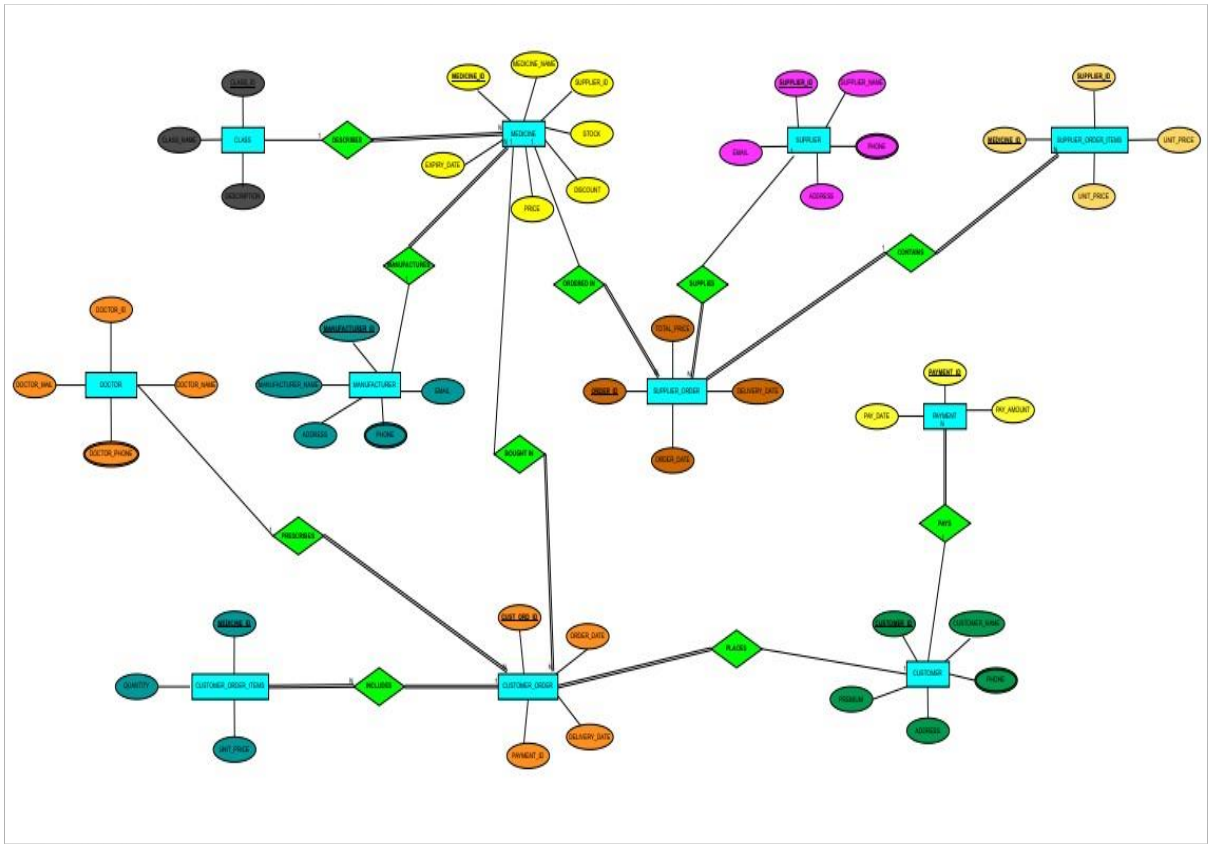
LHS IS A SUPER KEY AND IT DOES NOT HAVE TRANSITIVE DEPENDENCY (NPA -> NPA) THEREFORE, IT IS IN 3NF

R2: (MEDICINE_ID -> SUPPLIER_ID, UNIT_PRICE)
FD'S : MEDICINE_ID -> SUPPLIER_ID
MEDICINE_ID -> SUPPLIER_ID
LHS IS A SUPER KEY AND IT DOES NOT HAVE TD
THEREFORE, IT IS ALSO IN 3NF

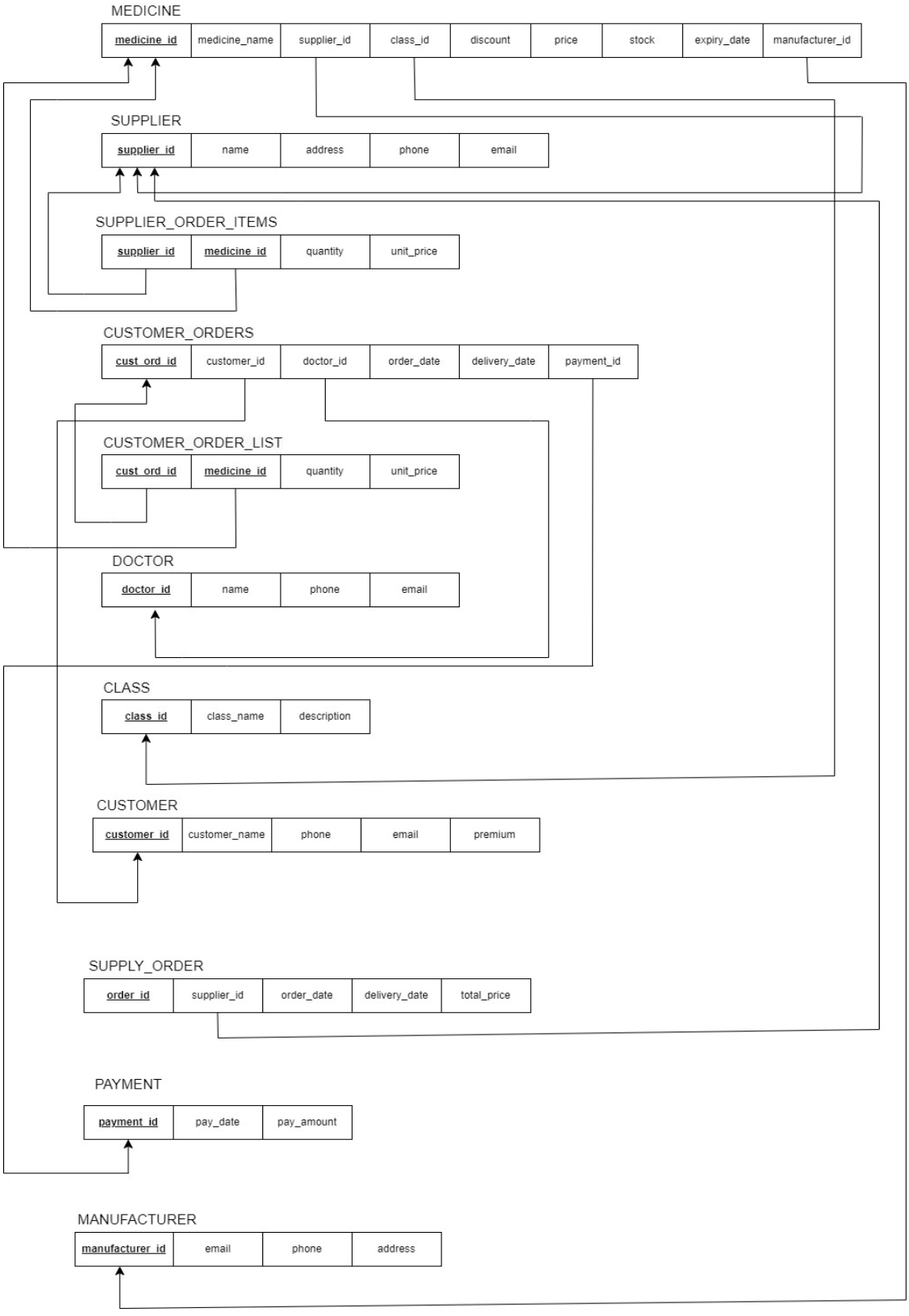
CHECKING FOR BCNF –
SINCE, BOTH RELATIONS HAS LHS AS THE SUPER KEY AND IT IS IN 3NF, IT MUST BE IN BCNF.

After Normalisation -

Modified ER Diagram :



Modified Schema Diagram -

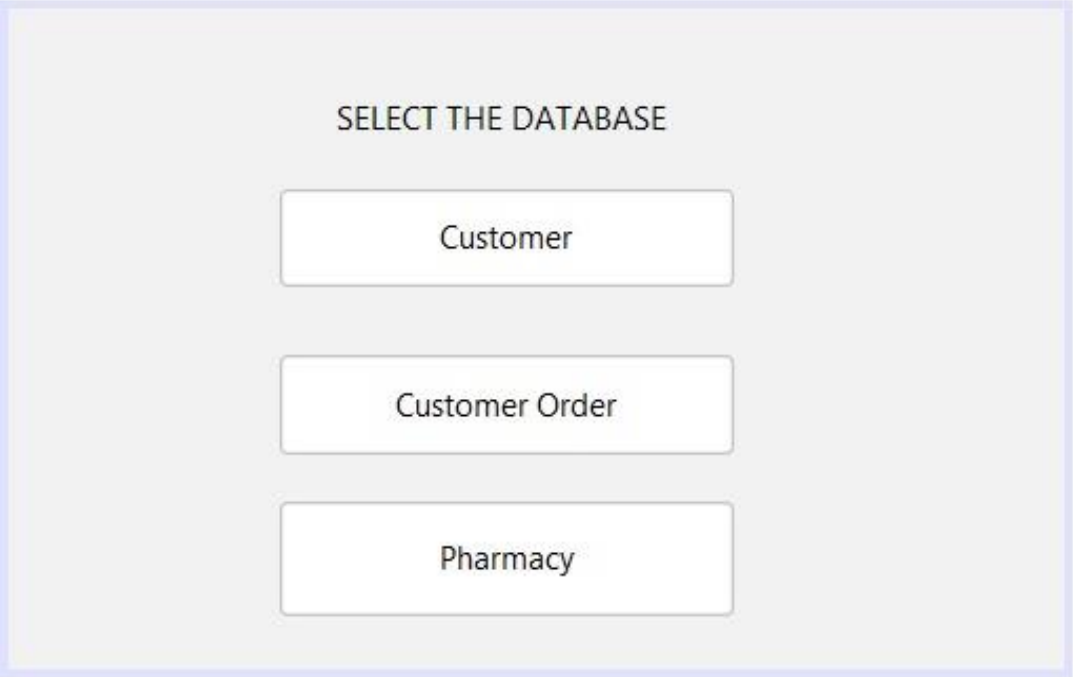


Implementing GUI and Database Connectivity Database Connectivity Code:

Homepage -

```
private void initComponents() {

        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();
        jButton3 = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jButton1.setText("Customer");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }









        });

        jButton2.setText("Pharmacy");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        jLabel1.setText("SELECT THE DATABASE");


        jButton3.setText("Customer Order");
        jButton3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton3ActionPerformed(evt);
```

```
        }
    });
```



 Initializing UI Components: Creating buttons and labels for the interface. Assigning text to the buttons and defining their click actions

This interface shows whether we are going
to access the application for

- Customer Details
- Customer Medicine Order
- Pharmacist

Customer:

INSERT –

```
        String query = "INSERT INTO Customer (customer_id, customer_name,
email, phone, premium_subscription, address) VALUES (?, ?, ?, ?, ?, ?)";

        try  {
            PreparedStatement pst = con.prepareStatement(query);
            int customerId = getNextCustomerId(con);
            pst.setInt(1, customerId);
            pst.setString(2, txtCustomerName.getText());
            pst.setString(3, txtEmail.getText());
            pst.setString(4, txtPhone.getText());
            pst.setString(5, txtPremiumSubscription.getText());
            pst.setString(6, txtAddress.getText());
            pst.executeUpdate();
            JOptionPane.showMessageDialog(this, "Customer inserted success-
fully!");
            txtCustomerId.setText(String.valueOf(customerId)); // Update the
customer ID field
        } catch (SQLException ex) {

            JOptionPane.showMessageDialog(this, "Error inserting customer: " +
ex.getMessage());
        }
```

DELETE –

```
 String query = "DELETE FROM Customer WHERE customer_id = ?";
        try {
             PreparedStatement pst = con.prepareStatement(query);
            pst.setInt(1, Integer.parseInt(txtCustomerId.getText()));
            pst.executeUpdate();
            JOptionPane.showMessageDialog(this, "Customer deleted success-
fully!");
        } catch (SQLException ex) {

            JOptionPane.showMessageDialog(this, "Error deleting customer: " +
ex.getMessage());
```

```
        }
```

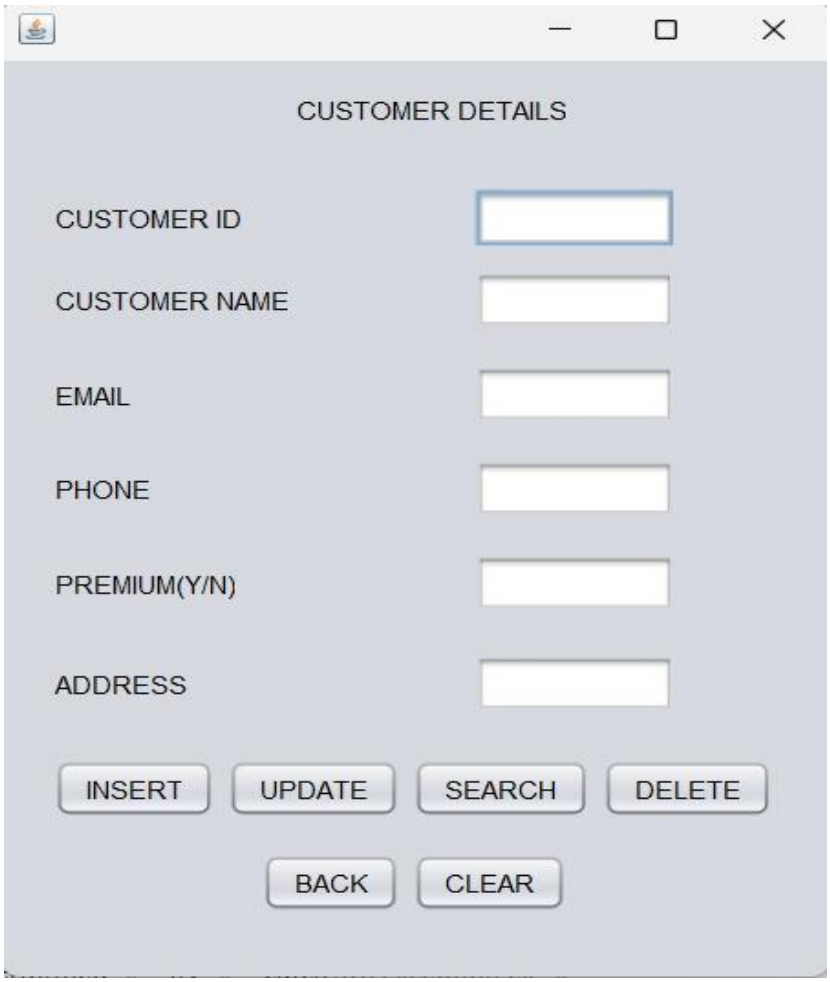
SEARCH –

```
   String query = "SELECT * FROM Customer WHERE customer_id = ? OR (cus-
tomer_name = ? AND phone = ?)";
        try {
             PreparedStatement pst = con.prepareStatement(query);
            pst.setInt(1, Integer.parseInt(txtCustomerId.getText().isEmpty() ?
"0" : txtCustomerId.getText()));
            pst.setString(2, txtCustomerName.getText());
            pst.setString(3, txtPhone.getText());
            try (ResultSet rs = pst.executeQuery()) {
                if (rs.next()) {
                    txtCustomerId.setText(String.valueOf(rs.getInt("cus-
tomer_id")));
                    txtCustomerName.setText(rs.getString("customer_name"));
                    txtEmail.setText(rs.getString("email"));
                    txtPhone.setText(rs.getString("phone"));
                    txtPremiumSubscription.setText(rs.getString("premium_sub-
scription"));
                    txtAddress.setText(rs.getString("address"));
                } else {
                    JOptionPane.showMessageDialog(this, "Customer not
found!");
                }
            }
        } catch (SQLException ex) {

            JOptionPane.showMessageDialog(this, "Error searching customer: " +
ex.getMessage());
        }
```

UPDATE –

```
    String query = "UPDATE Customer SET customer_name = ?, email = ?, phone =
?, premium_subscription = ?, address = ? WHERE customer_id = ?";
        try {
             PreparedStatement pst = con.prepareStatement(query);
            pst.setString(1, txtCustomerName.getText());
            pst.setString(2, txtEmail.getText());
            pst.setString(3, txtPhone.getText());
            pst.setString(4, txtPremiumSubscription.getText());
```

```
            pst.setString(5, txtAddress.getText());
            pst.setInt(6, Integer.parseInt(txtCustomerId.getText()));
            pst.executeUpdate();
            JOptionPane.showMessageDialog(this, "Customer updated success-
fully!");
        } catch (SQLException ex) {

            JOptionPane.showMessageDialog(this, "Error updating customer: " +
ex.getMessage());
        }
```



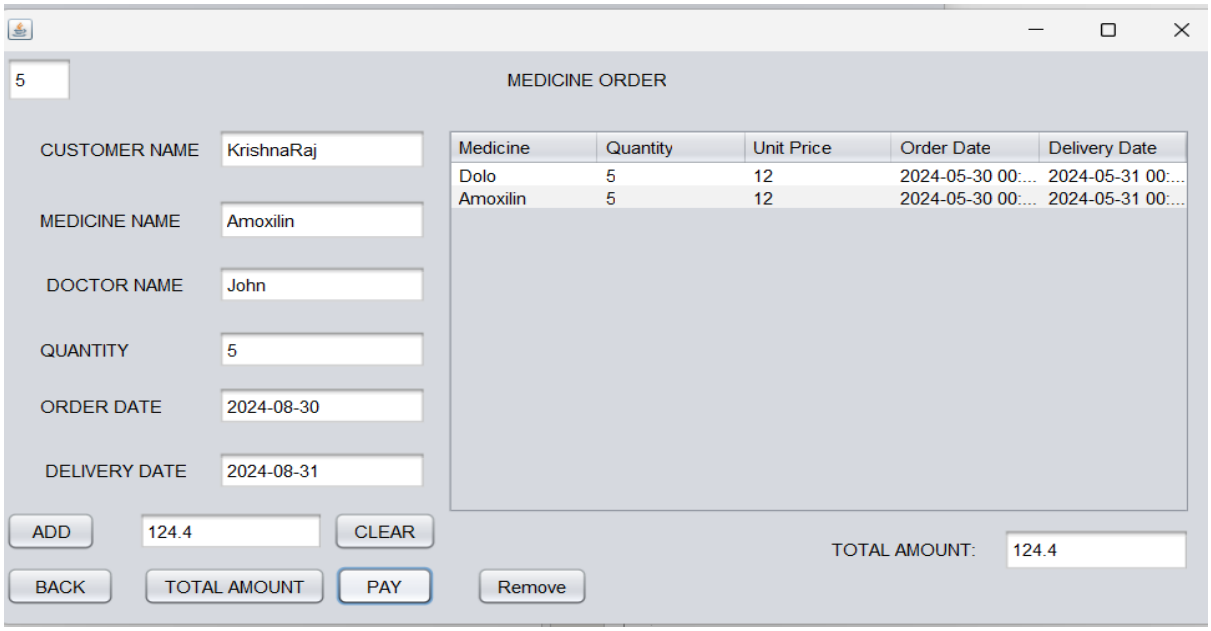
❖ This interface Customer Details is to add a new customer in the database

- We can search the customer details using customer id
- We can update the customer's details if needed
- When customer wants to delete their info , it can also be done using delete option which deletes their information

CUSTOMER MEDICINE ORDER-

```
private double fetchUnitPrice(Connection con, int medicineId) throws SQLExcep-
tion {
    String query = "SELECT price FROM Medicine WHERE medicine_id = ?";
    try (PreparedStatement pst = con.prepareStatement(query)) {
        pst.setInt(1, medicineId);
        try (ResultSet rs = pst.executeQuery()) {
            if (rs.next()) {
                return rs.getDouble(1);
            }
        }
    }
    return -1; // Not found
}

private int generateNewOrderId(Connection con) throws SQLException {
    String query = "SELECT MAX(cust_order_id) FROM Customer_order";
    try (Statement st = con.createStatement(); ResultSet rs = st.exe-
cuteQuery(query)) {
        if (rs.next()) {
            return rs.getInt(1) + 1;
        }
    }
    return 1; // If no records found, start with ID 1
}

private int generateNewPaymentId(Connection con) throws SQLException {
    String query = "SELECT MAX(payment_id) FROM Payment";
    try (Statement st = con.createStatement(); ResultSet rs = st.exe-
cuteQuery(query)) {
        if (rs.next()) {
            return rs.getInt(1) + 1;
        }
```

```
    }
    return 1; // If no records found, start with ID 1
}
private int fetchExistingPaymentId(Connection con, int customerId, Date deliv-
eryDate) throws SQLException {
    String query = "SELECT p.payment_id " +
                   "FROM Payment p " +
                   "JOIN Customer_order co ON p.payment_id = co.payment_id " +
                   "WHERE co.customer_id = ? AND p.pay_date = ?";
    try (PreparedStatement pst = con.prepareStatement(query)) {
        pst.setInt(1, customerId);
        pst.setDate(2, deliveryDate);
        try (ResultSet rs = pst.executeQuery()) {
            if (rs.next()) {
                return rs.getInt(1);
            }
        }
    }
    return -1; // No existing payment found
}

private void updatePaymentAmount(Connection con, int paymentId, double addi-
tionalAmount) throws SQLException {
    String query = "UPDATE Payment SET pay_amount = pay_amount + ? WHERE pay-
ment_id = ?";
    try (PreparedStatement pst = con.prepareStatement(query)) {
        pst.setDouble(1, additionalAmount);
        pst.setInt(2, paymentId);
        pst.executeUpdate();
    }
}
```

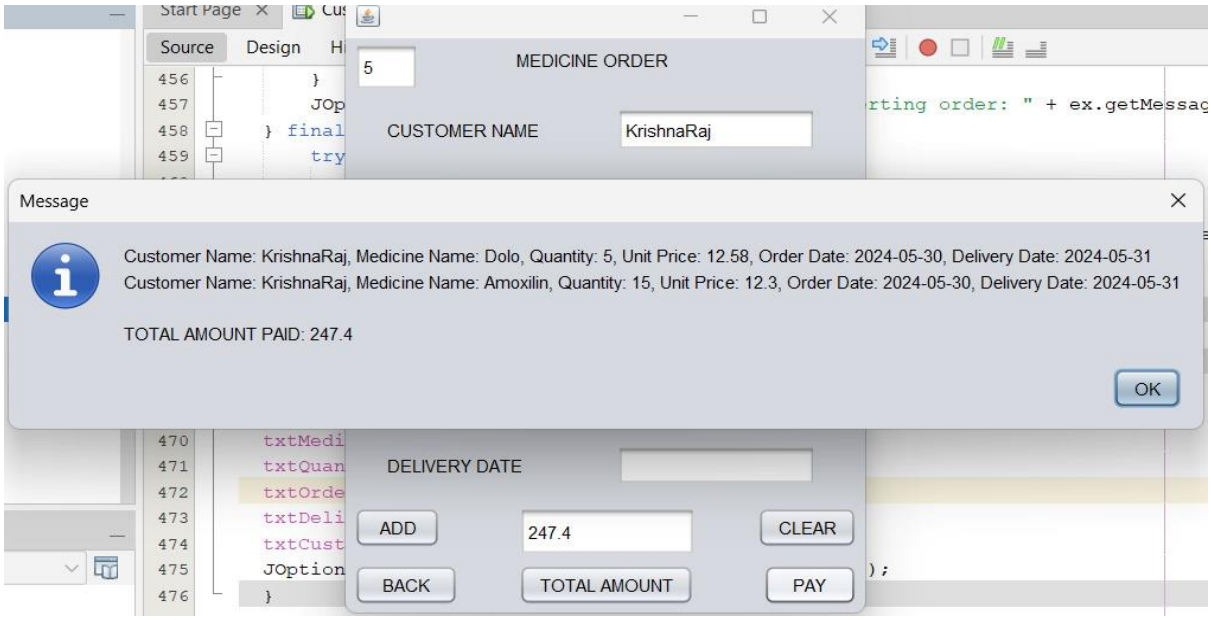


- This interface is to order medicine which is done by the customer
- We can see the total amount before Paying the bill also we can view the Medicines added to the list with the help of the table
- This needs the details about customer name, medicine name, quantity, order date and delivery date

Bill Generated –

```
  String billQuery = "SELECT cu.customer_name, m.medicine_name, cl.quantity,
cl.unit_price, m.discount, c.order_date, c.delivery_date, p.pay_amount " +
                           "FROM customer_order_list cl " +
                           "JOIN customer_order c ON cl.cust_order_id =
c.cust_order_id " +
                           "JOIN customer cu ON c.customer_id = cu.customer_id
" +
                           "JOIN payment p ON c.payment_id = p.payment_id " +
                           "JOIN medicine m ON m.medicine_id = cl.medicine_id
" +
                           "WHERE c.customer_id = ? AND c.cust_order_id = ?";

        try (PreparedStatement pst = con.prepareStatement(billQuery)) {
            pst.setInt(1, customerId);
            pst.setInt(2, custOrderId);
```

```
            try (ResultSet rs = pst.executeQuery()) {
                if (rs.next()) {
                    StringBuilder billDetails = new StringBuilder();
                    double totalAmount = 0;
                   do {
                        String customerNameFetched = rs.getString("cus-
tomer_name");
                        String medicineName = rs.getString("medicine_name");
                        int quantity = rs.getInt("quantity");
                        double unitPrice = rs.getDouble("unit_price");
                        double discountPrice = rs.getDouble("discount");
                        Date orderDateFetched = rs.getDate("order_date");
                        Date deliveryDate = rs.getDate("delivery_date");

                        if (isPremium) {
                            unitPrice -= discountPrice;
                        }

                billDetails.append("Customer Name: ").append(customerNameFetched)
                                        .append(", Medicine Name: ").append(medi-
cineName)
                                        .append(", Quantity: ").append(quantity)
                                        .append(", Unit Price: ").append(unitPrice)
                                        .append(", Order Date: ").append(orderDate-
Fetched)
                                        .append(", Delivery Date: ").append(deliv-
eryDate)
                                        .append("\n");

                        totalAmount += unitPrice * quantity;
                    } while (rs.next());

                    billDetails.append("\nTOTAL AMOUNT PAID: ").append(to-
talAmount);
                    JOptionPane.showMessageDialog(this, billDe-
tails.toString());
                } else {
                    JOptionPane.showMessageDialog(this, "No orders found for
the specified customer. Please recheck IDs.");
                }
            }
        }
    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(this, "Error generating bill: " +
ex.getMessage());
    }
```



- This is the bill generated after paying for the Medicine which contains the details of the order which includes medicine name, quantity, price, date, total price etc …

Pharmacy Medicine Order –

INSERT –

```
  String sql = "INSERT INTO Medicine (medicine_id, medicine_name, supplier_id,
manufacturer_id, stock, discount, price, expiry_date, class_id) VALUES (?, ?,
?, ?, ?, ?, ?, ?, ?)";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            ps.setString(2, jTextField2.getText());
            ps.setInt(3, Integer.parseInt(jTextField3.getText()));
            ps.setInt(4, Integer.parseInt(jTextField4.getText()));
            ps.setInt(5, Integer.parseInt(jTextField5.getText()));
            ps.setDouble(6, Double.parseDouble(jTextField6.getText()));
            ps.setDouble(7, Double.parseDouble(jTextField7.getText()));
            ps.setString(8, (jTextField8.getText()));
            ps.setInt(9, Integer.parseInt(jTextField9.getText()));
            ps.executeQuery();
            JOptionPane.showMessageDialog(null, "Inserted");
        }
```

```
        catch(SQLException ex)
        {
                Logger.getLogger(medicine.class.getName()).log(Level.SEVERE,
null, ex);
        }
```

DELETE –

```
  String sql ="DELETE FROM Medicine WHERE medicine_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            ps.executeQuery();
            JOptionPane.showMessageDialog(null, "Deleted");
        }
        catch(SQLException ex)
            {
                Logger.getLogger(medicine.class.getName()).log(Level.SEVERE,
null, ex);
            }
```

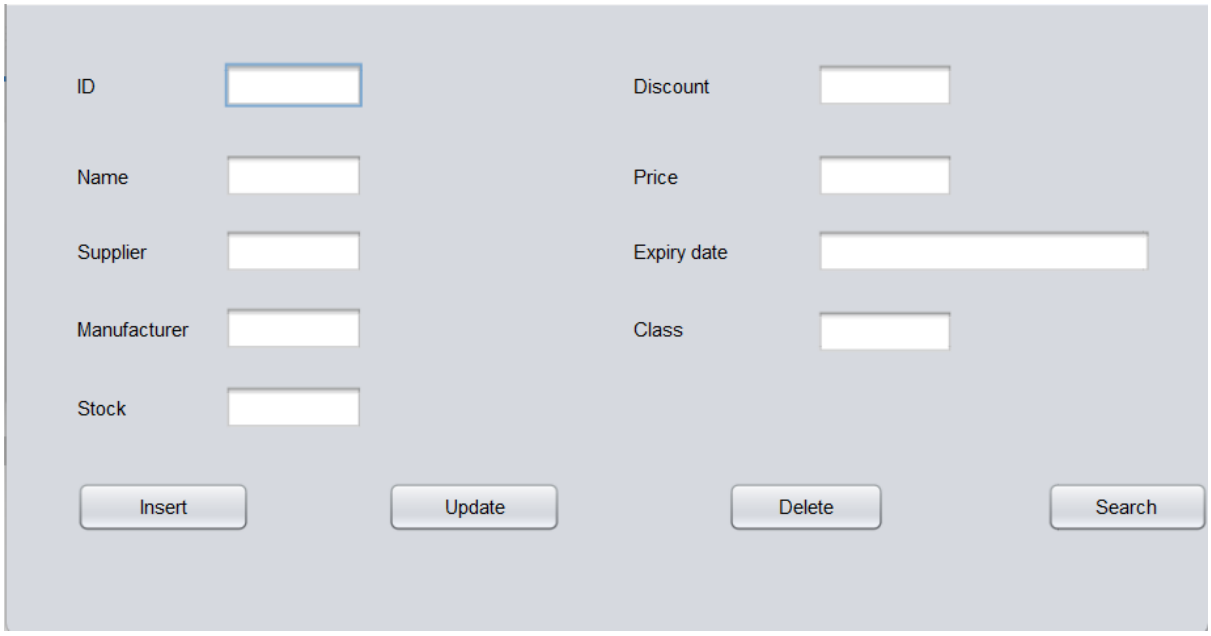
UPDATE –

```
 String sql = "UPDATE Medicine SET medicine_name = ?, supplier_id = ?, manu-
facturer_id = ?, stock = ?, discount = ?, price = ?, expiry_date = ?, class_id
= ? WHERE medicine_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setString(1, jTextField2.getText());
            ps.setInt(2, Integer.parseInt(jTextField3.getText()));
            ps.setInt(3, Integer.parseInt(jTextField4.getText()));
            ps.setInt(4, Integer.parseInt(jTextField5.getText()));
            ps.setDouble(5, Double.parseDouble(jTextField6.getText()));
            ps.setDouble(6, Double.parseDouble(jTextField7.getText()));
            ps.setString(7, jTextField8.getText()); // Set expiry_date as
string
            ps.setInt(8, Integer.parseInt(jTextField9.getText()));
            ps.setInt(9, Integer.parseInt(jTextField1.getText()));
            ps.executeQuery();
            JOptionPane.showMessageDialog(null, "Updated");
        }
        catch(SQLException ex)
        {
```

```
            Logger.getLogger(medicine.class.getName()).log(Level.SEVERE, null,
ex);
        }
```

SEARCH –

```
 String sql = "SELECT * FROM Medicine WHERE medicine_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            rs = ps.executeQuery();
            if (rs.next()) {
                    jTextField2.setText(rs.getString(2));
                    jTextField3.setText(rs.getString(3));
                    jTextField4.setText(rs.getString(4));
                    jTextField5.setText(rs.getString(5));
                    jTextField6.setText(rs.getString(6));
                    jTextField7.setText(rs.getString(7));
                    jTextField8.setText(rs.getString(8));
                    jTextField9.setText(rs.getString(9));
            }
        }
        catch(SQLException ex)
        {
                Logger.getLogger(medicine.class.getName()).log(Level.SEVERE,
null, ex);
        }
```



- This interface is used by Pharmacy to add medicine bought from the supplier into the database.
- Also be used to update and Search the stock, Supplier , Manufacturer ,Stock , Price ,Expiry date , Class of a specific medicine using the Medicine ID.
- This interface offers user-friendly CRUD operations (Create, Read, Update, Delete)

Manufacturer -

INSERT –

```
String sql = "INSERT INTO Manufacturer VALUES (?, ?, ?, ?, ?)";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            ps.setString(2, jTextField2.getText());
            ps.setString(3, jTextField3.getText());
            ps.setString(4, jTextField4.getText());
            ps.setString(5, jTextField5.getText());
            ps.executeQuery();
            JOptionPane.showMessageDialog(null,"Inserted");
        }
```

DELETE –

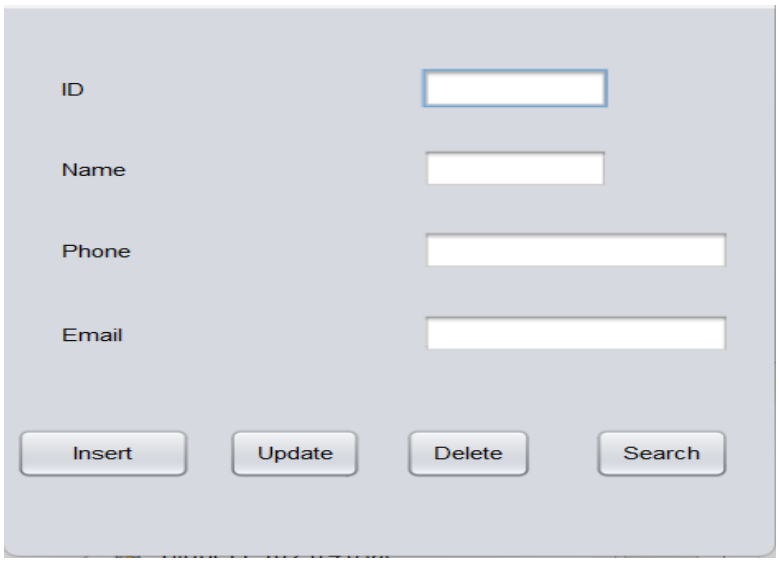
```
        // TODO add your handling code here:
        String sql = "DELETE FROM Manufacturer WHERE manufacturer_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            ps.executeQuery();
            JOptionPane.showMessageDialog(null, "Deleted");
        }
```

UPDATE –

```
  String sql = "UPDATE Manufacturer SET manufacturer_name = ?, email = ?,
phone = ?, address = ? WHERE manufacturer_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setString(1, jTextField2.getText());
            ps.setString(2, jTextField3.getText());
            ps.setString(3, jTextField4.getText());
            ps.setString(4, jTextField5.getText());
            ps.setInt(5, Integer.parseInt(jTextField1.getText()));
            ps.executeQuery();
            JOptionPane.showMessageDialog(null, "Updated");

        }
```

SEARCH –

```
String sql = "SELECT * FROM Manufacturer WHERE manufacturer_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            rs=ps.executeQuery();
            if (rs.next()) {
                    jTextField2.setText(rs.getString(2));
                    jTextField3.setText(rs.getString(3));
                    jTextField4.setText(rs.getString(4));
                    jTextField5.setText(rs.getString(5));
            }
```



- This interface is used to add Manufacturer details into the database when a new manufacturer is added
- Also used to see , update , delete their details by using their Manufacturer_id number

Supplier -

INSERT –

```
String sql = "INSERT INTO Supplier (supplier_id, supplier_name, email, phone,
address) VALUES (?, ?, ?, ?, ?)";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            ps.setString(2, jTextField2.getText());
            ps.setString(3, jTextField3.getText());
            ps.setString(4, jTextField4.getText());
            ps.setString(5, jTextField5.getText());
            ps.executeQuery();
            JOptionPane.showMessageDialog(null,"Inserted");
        }
```

UPDATE –

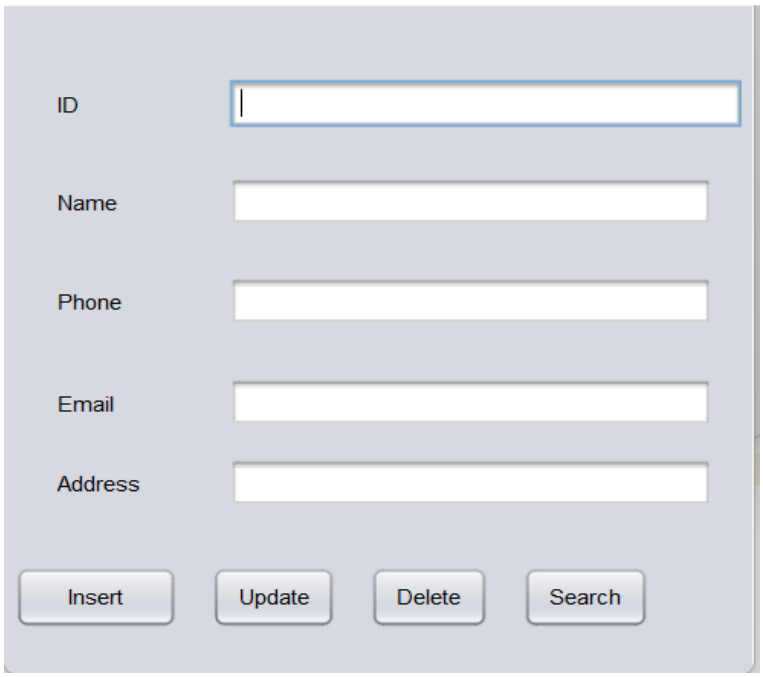
```
String sql = "UPDATE Supplier SET supplier_name = ?, email = ?, phone = ?, ad-
dress = ? WHERE supplier_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setString(1, jTextField2.getText());
            ps.setString(2, jTextField3.getText());
            ps.setString(3, jTextField4.getText());
            ps.setString(4, jTextField5.getText());
            ps.setInt(5, Integer.parseInt(jTextField1.getText()));
            ps.executeQuery();
            JOptionPane.showMessageDialog(null, "Updated");

        }
```

DELETE –

```
 String sql = "DELETE FROM Supplier WHERE supplier_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            ps.executeQuery();
            JOptionPane.showMessageDialog(null, "Deleted");
        }
```

SEARCH –

```
 String sql = "SELECT * FROM Supplier WHERE supplier_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            rs=ps.executeQuery();
            if (rs.next()) {
                    jTextField2.setText(rs.getString(2));
                    jTextField3.setText(rs.getString(3));
                    jTextField4.setText(rs.getString(4));
                    jTextField5.setText(rs.getString(5));
            }

        }
```



- This interface is used to add supplier details into the database when a new supplier is added
- Also used to see , update , delete their details by using their Supplier_id number.

Doctor -

INSERT -

```
 String sql = "INSERT INTO Doctor (doctor_id, doctor_name, doctor_phone, doc-
tor_email) VALUES (?, ?, ?, ?)";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            ps.setString(2, jTextField2.getText());
            ps.setString(3, jTextField3.getText());
            ps.setString(4, jTextField4.getText());
            ps.executeQuery();
            JOptionPane.showMessageDialog(null, "Inserted");
```

UPDATE –

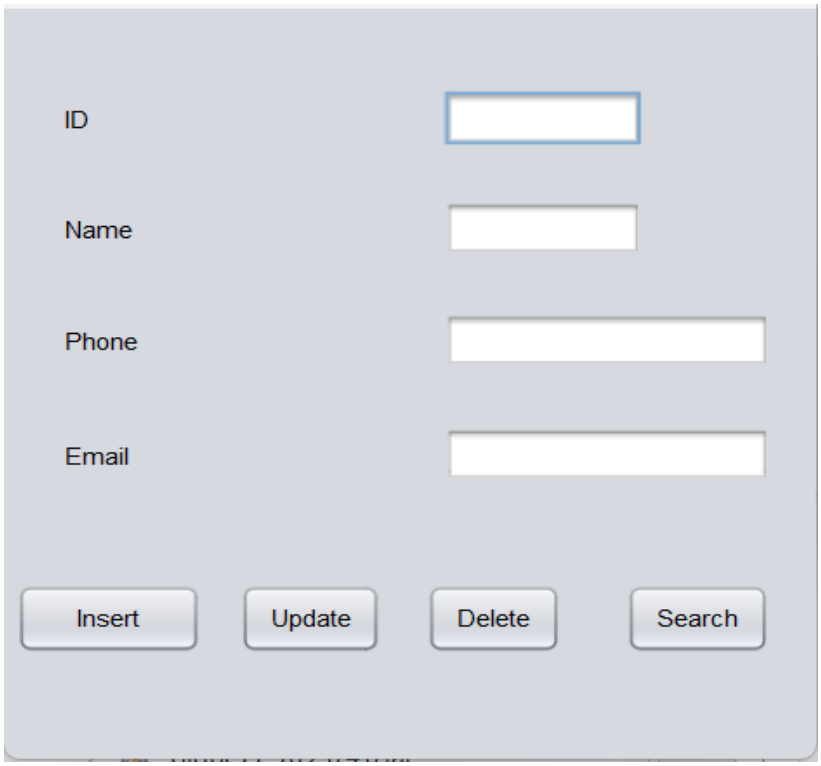
```
String sql = "UPDATE Doctor SET doctor_name = ?, doctor_phone = ?, doc-
tor_email = ? WHERE doctor_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setString(1, jTextField2.getText());
            ps.setString(2, jTextField3.getText());
            ps.setString(3, jTextField4.getText());
            ps.setInt(4, Integer.parseInt(jTextField1.getText()));
            ps.executeQuery();
            JOptionPane.showMessageDialog(null, "Updated");
        }
```

DELETE –

```
   String sql = "DELETE FROM Doctor WHERE doctor_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            ps.executeQuery();
            JOptionPane.showMessageDialog(null, "Deleted");
        }
```

SEARCH –

```
String sql = "SELECT * FROM Doctor WHERE doctor_id = ?";
        try
        {
            ps = con.prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(jTextField1.getText()));
            rs=ps.executeQuery();
            if (rs.next()) {
                    jTextField2.setText(rs.getString(2));
                    jTextField3.setText(rs.getString(3));
                    jTextField4.setText(rs.getString(4));
                }
```

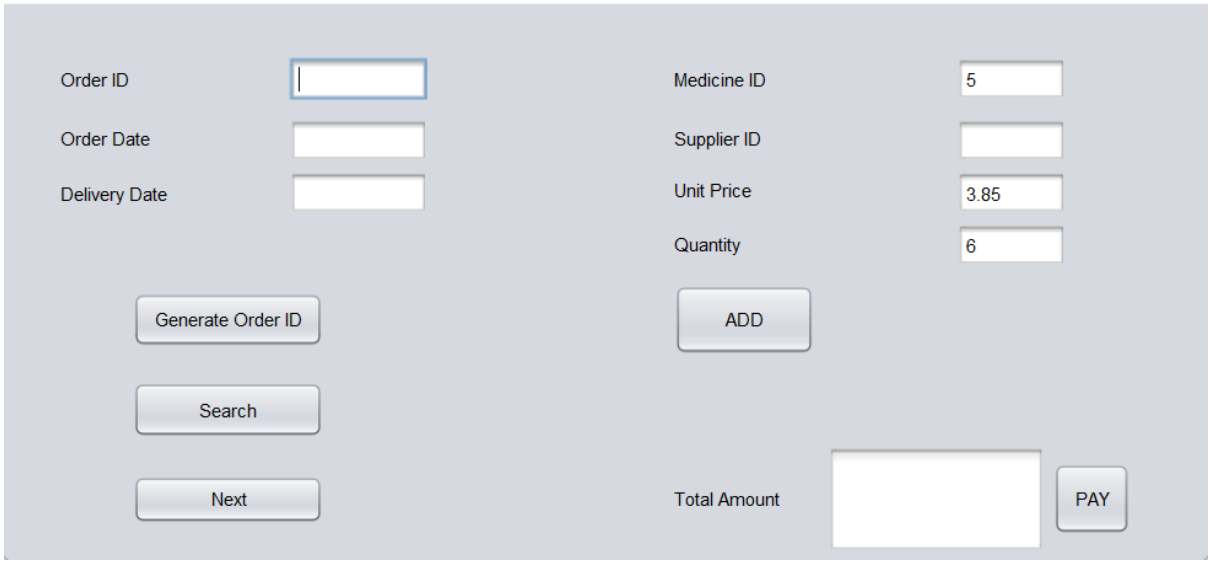


- This is to provide information about Doctor.
- This table can be used to add a new doctor in the database with a ID number
- This can be used to verify if the Medicine prescription is given by a verified doctor as the customer order checks the doctor_id is available before adding the order

Supply Order -

```
  String insertOrderItemSql = "INSERT INTO Supply_order_List (order_id, medi-
cine_id, unit_price, qty) VALUES (?, ?, ?, ?)";
        String updateTotalPriceSql = "UPDATE Supplier_Order so SET total_price
= (SELECT SUM(sul.unit_price * sul.qty) FROM Supply_order_List sul WHERE
sul.order_id = so.order_id) WHERE order_id = ?";

    try {

        ps = con.prepareStatement(insertOrderItemSql);
        int orderId = Integer.parseInt(jTextField1.getText());
        int medicineId = Integer.parseInt(jTextField5.getText());
        double unitPrice = Double.parseDouble(jTextField7.getText());
        int quantity = Integer.parseInt(jTextField8.getText());

        ps.setInt(1, orderId);
        ps.setInt(2, medicineId);
        ps.setDouble(3, unitPrice);
        ps.setInt(4, quantity);

        ps.executeUpdate();


        ps = con.prepareStatement(updateTotalPriceSql);
        ps.setInt(1, orderId); // Set the order_id for the update
        ps.executeUpdate();


        ps = con.prepareStatement("SELECT total_price FROM Supplier_Order
WHERE order_id = ?");
        ps.setInt(1, orderId);
        rs = ps.executeQuery();
        if (rs.next()) {
            double newTotalPrice = rs.getDouble("total_price");
            jTextField4.setText(String.valueOf(newTotalPrice));
        }
        String str = "UPDATE Medicine SET stock = stock + ? where medicine_id
= ?";
        ps = con.prepareStatement(str);
        ps.setInt(1, quantity);
        ps.setInt(2, medicineId);
        ps.executeQuery();
        JOptionPane.showMessageDialog(null, "Item Added");
```

```
}
```



- This interface is used by the pharmacy to order medicine as a whole from the Supplier
- Also used to set the price , Quantity to be ordered and Search if a medicine is available and Pay for the medicines