Pharmacy supply chain Management system

UCS2404 - DATABASE MANAGEMENT SYSTEMS

Report

Submitted By
G Harikumar(3122225001033)
Aditya Jyosyula(3122225001006)

Department of Computer Science and Engineering

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

2023#24 July

UCS2404 - DATABASE MANAGEMENT SYSTEMS
Regulations – R2021

Project Report

Pharmacy supply chain Management system

Assumptions:
For the given project we have assumed some base assumptions so as to come up with appropriate attributes for the database. Our assumptions include:

1)There are multiple pharmacies which a singular customer can buy medicines from.

2)Each pharmacy has a name , email, phone number and multiple suppliers who supply medicines to the pharmacy.

3)Then we have customers who can order from the pharmacies. The customer details include name ,email ,phone number , date of birth and age.

4)The medicines that is the products in a pharmacy are stored inside the inventory which has pharmacy's id , the supplier's id and the product which is available .

5)The product details include the name , the description and the price of the medicine.

6)We have assumed that a customers order can include multiple products which can be bought from multiple pharmacies.

7)The details of an order include the order id ,customer id , the pharmacy id from which the product was taken , the product id of the product being bought and other details like the quantity of each product being ordered ,the total amount of the bill and the date on which the order was placed.

8)The details of the supplier table include the supplier id of each of the supplier , their name , email ,phone number and the pharmacy id of the pharmacy to which the supplier is providing products to.

9)The payment and shipment tables contain the details about the payment method ,date and the shipment status and such.

Database:

Pharmacy:

a) pharmacy_id
b) pharmacy_name
c) pharmacy_email
d) pharmacy_ph
e) supplier_id

Customers:

a) customer_id
b) name
c) email
d) phone
e) dob
f) age

Products:
a) product_id
b) namess
c) description
d) price

Orders:
a) order_id
b) customer_id
c) pharmacy_id
d) order_date
e) total_amount
f) product_id
g) quantity
a) unit_price

Suppliers:
a) supplier_id
b) name
c) email
d) phone
e) pharmacy_id

Inventory:
a) pharmacy_id
b) product_id
c) supplier_id
d) quantity_delv

Payments:
a) payment_id
b) cus_id
c) order_id
d) payment_date
e) payment_amount
f) payment_method

Shipment:
a) shipment_id
b) order_id
c) shipment_start_date
d) shipment_end_date

e) status

Address:

a) city
b) pincode
c) house_no
d) street
e) pharmacy_id
f) customer_id

<u>Functional Dependency:</u>

<u>1) Customer:</u>

a) customer_id
b) name
c) email
d) phone
e) dob
f) age

Possible FD's:

{
customer_id ->name,email,phone,dob,age
phone -> name
email -> phone
dob ->age
}

Removing email -> phone as some customers may not have email.

<u>Forming minimal set of FD's:</u>

1) Decomposition:

=> customer_id ->name,email,phone,dob,age

- ⇨ customer_id ->name
- ⇨ customer_id ->email
- ⇨ customer_id ->phone
- ⇨ customer_id ->dob
- ⇨ customer_id ->age

F1 = {
customer_id ->name
customer_id ->email
customer_id ->phone
customer_id ->dob

customer_id ->age
phone -> name
dob ->age }

2) checking for Redundancy:

i) checking if customer_id ->name is redundant

{customer_id}+ = {customer_id,email,phone,name,dob,age}

So customer_id ->name is redundant as it contain all the attributes.

F2 = {
customer_id ->email
customer_id ->phone
customer_id ->dob
customer_id ->age
phone -> name
dob ->age }

ii) checking if customer_id ->email is redundant

{customer_id}+ = {customer_id,name,phone,dob,age}

So customer_id ->email is not redundant as it does not contain all the attributes.

iii) checking if customer_id ->phone is redundant

{customer_id}+ = {customer_id,email,dob,age}

So customer_id ->phone is not redundant as it does not contain all the attributes.

iv) checking if customer_id ->dob is redundant

{customer_id}+ = {customer_id,name,email,phone,age}

So customer_id ->dob is not redundant as it does not contain all the attributes

v) checking if customer_id ->age is redundant

{customer_id}+ = {customer_id,email,phone,name,dob,age}

So customer_id ->age is redundant as it contain all the attributes.

F3 = {
customer_id ->email
customer_id ->phone
customer_id ->dob
phone -> name
dob ->age }

vi) checking if phone -> name is redundant

{phone)+ = {phone,name}

So phone -> name is not redundant as it does not contain all the attributes

The minimal set
{
customer_id ->email,phone,dob
phone -> name
dob ->age }

{customer_id}+ = {customer_id,email,phone,name,dob,age}

Customer_id is the key.

<u>2) Products :</u>

a) product_id
b) name
c) description
d) price

Possible FD's:

{ product_id -> name,description,price
name -> description,price
}

Forming minimal set:

1) Decomposition:

=> product_id -> name,description,price

- ⇨ product_id -> name
- ⇨ product_id -> description
- ⇨ product_id -> price

=> name -> description,price

- ⇨ name -> description
- ⇨ name -> price

F1 = {
product_id -> name
product_id -> description
product_id -> price
name -> description
name -> price }

2) checking for Redundant

i) checking if product_id -> name is redundant

{product_id}+ = {product_id,description,price}

So product_id -> name is not redundant as it does not contain all the attributes

ii) checking if product_id -> description is redundant

{product_id}+ = {product_id,name,description,price}

So product_id -> description is redundant as the closure contains all the attributes

F2 = {
    product_id -> name
    product_id -> price
    name -> description
  name -> price }

iii) checking if product_id -> price id redundant

{product_id}+ = {product_id,name,description,price}

So product_id -> price is redundant as the closure contains all the attributes

F3 = {
    product_id -> name
    name -> description
  name -> price }

iv) checking if name -> description is redundant

{name}+ = {name,price}

So name -> description is not redundant as it does not contain all the attributes
v) checking if name -> price is redundant

{name}+ = {name,description}

So name -> price is not redundant as it does not contain all the attributes

The minimal set is
 {
    product_id -> name
    name -> description
  name -> price }

{product_id}+ = {product_id,name,description,price}

Product_id is the key.

3) Shipment:

a) shipment_id
b) order_id
c) start_date
d) end_date
e) status

Possible FD's:

{
shipment_id -> order_id,start_date,end_date,status
shipment_id,start_date -> order_id
order_id,start_date -> end_date
}

Forming the minimal set:

1) Decomposition:

=> shipment_id -> order_id,start_date,end_date,status

- ⇨ shipment_id -> order_id
- ⇨ shipment_id ->start_date
- ⇨ shipment_id -> end_date
- ⇨ shipment_id -> status

F1 = {
shipment_id -> order_id
shipment_id ->start_date
shipment_id -> end_date
shipment_id -> status
shipment_id,start_date -> order_id
order_id,start -> end_date
}

2) checking for extraneous attributes:

=> shipment_id,start_date -> order_id

checking if shipment_id is extraneous:

start_date -> order_id

{start_date}+ = {start_date,order_id,end_date}

Therefore shipment_id is not extraneous

Checking if start_date is extraneous attribute:

shipment_id -> order_id

{shipment_id}+ = {shioment_id,order_id,start_date,end_date,status}

This contains all attributes .therefore start_date is extraneous.

```
F2 = {
      shipment_id -> order_id
      shipment_id ->start_date
      shipment_id -> end_date
      shipment_id -> status
   order_id,start_date -> end_date
}
```

=> order_id,start_date -> end_date

Checking if order_id is extraneous_id:

start_date -> end_date

{start_date}+ = {start_date,end_date}

This is not extraneous as it does not contains all the attributes.

Checking if start_date is extraneous:

Order_id -> end_date

{order_id}+ = {order_id,end_date}

This is not extraneous as it does not contains all the attributes.

Therefore neither of the attributes are extraneous.

```
F2 = {
      shipment_id -> order_id
      shipment_id ->start_date
      shipment_id -> end_date
      shipment_id -> status
   order_id,start_date -> end_date
}
```

3) checking for redundancy:

i) checking if shipment_id -> order_id is redundant:

{shipment_id}+ = {shipment_id,start_date,end_date,status}

This is not redundant as the closure does not all the attributes

ii) checking if shipment_id -> start_date is redundant:

{shipment_id}+ = {shipment_id,order_id,end_date,status}

This is not redundant as the closure does not all the attributes

iii) checking if shipment_id -> end_date is redundant:

{shipment_id}+ = {shipment_id,order_id,start_date,end_date,status}

Therefore shipment_id -> end_date is redundant as it contains all the attributes

```
F3 = {
      shipment_id -> order_id
      shipment_id ->start_date
      shipment_id -> status
   order_id,start_date -> end_date
}
```

iv) checking if shipment_id -> status is redundant:

{shipment_id}+ = {shipment_id,order_id,start_date,end_date}

This is not redundant as the closure does not all the attributes

v) checking if order_id,start_date -> end_date is redundant :

{ order_id,start_date }+ = { order_id,start_date ,end_date}

This is not redundant as the closure does not all the attributes

```
The minimal set: {
      shipment_id -> order_id
      shipment_id ->start_date
      shipment_id -> status
   order_id,start_date -> end_date
}
```

{shipment_id}+ = {shipment_id,order_id,start_date,end_date,status}

Shipment_id is the key.

4) Address:

a) city
b) pincode
c) house_no
d) street
e) pharmacy_id
f) customer_id

possible fd's

{
Pharmacy_id -> street,city,pincode,houseno
Customer_id -> street,city,pincode,houseno
Pincode -> city
house_no -> street
street -> city
}

Forming the minimal set:

1) decomposition :

=> pharmacy_id -> street,city,pincode,houseno

- ⇨ pharmacy_id -> street
- ⇨ pharmacy_id -> city
- ⇨ pharmacy_id -> pincode
- ⇨ pharmacy_id -> houseno

=> customer_id -> street,city,pincode,houseno

- ⇨ customer_id -> street
- ⇨ customer_id -> city
- ⇨ customer_id -> pincode
- ⇨ customer_id -> houseno

F1 = {
pharmacy_id -> street
pharmacy_id -> city
pharmacy_id -> pincode
pharmacy_id -> houseno
customer_id -> street
customer_id -> city
customer_id -> pincode
customer_id -> houseno

Pincode -> city
house_no -> street
street -> city
}

2) checking for redundant:

i. checking pharmacy_id -> street

{pharmacy_id}+ = {pharmacy_id,city,pincode,houseno}

This is not redundant as it does not contain all the attribute .

ii. checking pharmacy_id -> city

{pharmacy_id}+ = {pharmacy_id,city,pincode,houseno}

This is not redundant as it does not contain all the attribute .

iii. checking pharmacy_id -> pincode

{pharmacy_id}+ = {pharmacy_id,city,pincode,houseno}

This is not redundant as it does not contain all the attribute .

iv. checking pharmacy_id -> houseno

{pharmacy_id}+ = {pharmacy_id,city,pincode,houseno}
This is not redundant as it does not contain all the attribute

v. checking customer_id -> street

{customer_id}+ = {customer_id,city,pincode,houseno}

This is not redundant as it does not contain all the attribute

vi. checking customer_id -> city

{customer_id}+ = {customer_id,city,pincode,houseno}

This is not redundant as it does not contain all the attribute

vii. checking customer_id -> pincode

{customer_id}+ = {customer_id,city,pincode,houseno}

This is not redundant as it does not contain all the attribute

viii. checking customer_id -> houseno

{customer_id}+ = {customer_id,city,pincode,houseno}

This is not redundant as it does not contain all the attribute

xi. checking pincode -> city

{pincode}+ = {pincode}

This is not redundant as it does not contain all the attribute

x. checking house_no -> street

{house_no}+ = {house_no}

This is not redundant as it does not contain all the attribute

The minimal set of fd's:

{
 pharmacy_id -> street,city,pincode,houseno
 customer_id -> street,city,pincode,houseno
 pincode -> city
 house_no -> street
 street -> city
}

5) Inventory

a) pharmacy_id
b) product_id
c) supplier_id
d) quantity

Possible FDs:

Pharmacy_id,supplier_id ->qty

Pharmacy_id ->product_id

Forming the minimal set:

1. Decomposition:

There is attributes to be decomposed

2. checking for extraneous:

Pharmacy_id,supplier_id ->qty

Checking if pharmacy_id is extraneous:

supplier_id -> qty
{supplier_id}+ = {supplier_id,qty}

Therefore pharmacy_id is not extraneous

Checking if supplier_id is extraneous:

pharmacy_id -> qty

{pharmacy_id)+ = {pharmacy_id ,qty,product_id}

supplier_id is not extraneous

3. checking for redundant :

There is no redundant FDs.

{pharmacy_id,supplier_id}+ = {pharmacy_id,supplier_id,qty,product_id}

The minimaL set of FDs

Pharmacy_id,supplier_id ->qty
Pharmacy_id ->product_id

6) Supplier

a) supplier_id
b) name
c) email
d) phone
e) pharmacy_id

possible FDs{

```
  supplier_id -> name,email,phone,pharmacy_id
  Phone ->name
}
```

Forming the minimal set of fds

1. Decomposition

```
F1 = {
  supplier_id -> name
  supplier_id -> email
  supplier_id -> phone
  supplier_id -> pharmacy_id
  Phone ->name
}
```

2. checking for extraneous attributes.

There is no extraneous attributes.

3. checking for redundancy.

i. supplier_id -> name

{supplier_id}+ = {supplier_id,email,phone,pharmacy_id,name}

Therefore supplier_id -> name is redundant.

```
F2 = {
  supplier_id -> email
  supplier_id -> phone
  supplier_id -> pharmacy_id
  Phone ->name
}
```

ii. supplier_id -> email

{supplier_id}+ = {supplier_id ,phone,pharmacy_id,name}

Therefore supplier_id -> email is not redundant.

iii. supplier_id -> phone

{supplier_id}+ = {supplier_id ,email ,pharmacy_id,name}

Therefore supplier_id -> phone is not redundant.

iv.Phone ->name

{phone)+ = {phone}

Therefore Phone ->name is not redundant.

The minimal set of FDs.

{
supplier_id -> email
supplier_id -> phone
supplier_id -> pharmacy_id
Phone ->name
}

7) Orders

a) order_id
b) customer_id
c) pharmacy_id
d) order_date
e) total_amount
f) product_id
g) quantity
h) unit_price

possible FDs

{
order_id -> customer_id,pharmacy_id,order_date,total_amount,product_id,quantity
product_id ->unitprice
product_id,qty ->total amt
}

Forming the minimal set of FDs:

1.Decomposition:

F1 = {
order_id -> customer_id
order_id -> pharmacy_id
order_id -> order_date
order_id -> total_amount
order_id -> product_id
order_id -> quantity

product_id -> unit_price
product_id,quantity -> total_amount

}

2.finding the extraneous attributes.

product_id,quantity -> total_amount

1. product_id -> total_amount

{product_id}+ = {product_id,total_amount,unit_price}

Therefore quantity is not extraneous

{quantity}+ = {quantity,totalamount}

Therefore product_id is not extraneous.

3. checking for redundant.

F1 = {
order_id -> customer_id
order_id -> pharmacy_id
order_id -> order_date
order_id -> product_id
order_id -> quantity
product_id -> unit_price
product_id,quantity -> total_amount

}

The minimal set of FDs

F1 = {
order_id -> customer_id,pharmacy_id,order_date,product_id,quantity
product_id -> unit_price
product_id,quantity -> total_amount

}

8) Pharmacy:

a) pharmacy_id
b) pharmacy_name
c) pharmacy_email
d) pharmacy_ph
e) supplier_id

Possible fds:
{pharmacy_id->pharmacy_name,pharmacy_email,pharmacy_phone,supplier_id
Pharmacy_phone->pharmacy_name
Pharmacy_email->pharmacy_name}

Forming the minimal set of FDs:

1.Decomposition:

F1 = {
Pharmacy_id->pharmacy_name
Pharmacy_id->pharmacy_email
Pharmacy_id->pharmacy_phone
Pharmacy_id->supplier_id
Pharmacy_phone->pharmacy_name
Pharmacy_email->pharmacy_name
}

2. checking for extraneous attributes.

There is no extraneous attributes.

3. checking for redundant.

i. Pharmacy_id->pharmacy_name

{pharmacy_id}+={pharmacy_id,pharmacy_name}

pharmacy_id->pharmacy_name is not redundant

ii. Pharmacy_id->pharmacy_email

{pharmac_id}+={pharmacy_id,pharmacy_name,pharmacy_email}

pharmacy_id->pharmacy_email is not redundant

iii. Pharmacy_id->pharmacy_phone
{pharmacy_id}+={pharmacy_id,pharmacy_name,pharmacy_email,pharmacy_phone}

pharmacy_id->pharmacy_phone is not redundant

iv.pharmacy_id->supplier_id

{pharmacy_id}+={pharmacy_id,pharmacy_name,pharmacy_email,pharmacy_phone,supplier_id}

Pharmacy_id->supplier_id is not redundant v.pharmacy_phone->pharmacy_name

{pharmacy_phone}+={pharmacy_phone,pharmacy_name}
pharmacy_phone->pharmacy_name is redundant as this fd already exists in {payment_id}+

vi.pharmacy_email->pharmacy_name

{pharmacy_email}+={pharmacy_email,pharmacy_name}
pharmacy_email->pharmacy_name is redundant as this fd already exists in {payment_id}+

Minimal set of fds:
F1=
{
Pharmacy_id->pharmacy_name
Pharmacy_id->pharmacy_email
Pharmacy_id->pharmacy_phone
Pharmacy_id->supplier_id
}

F1={pharmacy_id-> pharmacy_name, pharmacy_email , pharmacy_phone
, supplier_id }

Pharmacy_id is the primary key

9)Payments:

a) payment_id
b) cus_id
c) order_id
d) payment_date
e) payment_amount
f) payment_method

Possible Fds:
{
payment_id->customer_id,order_id,payment_date,payment_amount,payment_method;
order_id,payment_id->customer_id;
order_id->payment_amount;
order_id,date->payment_amount;
payment_id,payment_data->payment_amount
}

1.Decomposition:

F1={
Payment_id->customer_id;
Payment_id->order_id;
Payment_id->payment_date;
Payment_id->payment_amount;
Payment_id->payment_method;
Order_id,payment_id->customer_id;

Order_id->payment_amount;
Order_id,date->payment_amount;
Payment_id,payment_data->payment_amount;
}

2.Checking for extraneous:
A)order_id,payment_id->customer_id;

payment_id->customer_id;

order_id is extraneous so payment_id,order_id->customer_id is removed.

B)Order_id,payment_date->payment_amount;

Order_id->payment_amount;

Payment_date is extraneous so order_id,payment_date->payment_amount is removed;

C)payment_id,payment_date->payment_amount;

Payment_date is extraneous so payment_id,payment_date->payment_amount is removed;

3)Checking for redundant:
{
Payment_id->customer_id;
Payment_id->order_id;
Payment_id->payment_date;
Payment_id->payment_amount;
Payment_id->payment_method;
Order_id->payment_amount;
}

i.payment_id->customer_id;

{payment_id}+={payment,customer_id}

Payment_id->customer_id is not redundant

ii.payment_id->order_id;

{payment_id}+={payment,customer_id,order_id}

Payment_id->order_id is not redundant

iii.payment_id->payment_date;

{payment_id}+={payment_id,customer_id,order_id,payment_date}

Payment_id->payment_id is not redundant

iv.payment_id->payment_method;

{payment_id}+={payment_id,customer_id,order_id,payment_date,payment_method}

Payment_id->payment_method is not redundant

v.payment_id->payment_amount;

{payment_id}+={payment_id,customer_id,
order_id,payment_date,payment_method,payment_amount}

Payment_id->customer_id is not redundant

vi.order_id->payment_amount;

{order_id}+={order_id,payment_id}

Order_id->payment_amount is redundant as it already exists in the closure of payment_id

Minimal set of fds:

F1={
Payment_id->customer_id;
Payment_id->order_id;
Payment_id->payment_date;
Payment_id->payment_amount;
Payment_id->payment_method;
}

F1={payment_id->customer_id,order_id,payment_date,payment_amount,payment_method}

Payment_id is the primary key

UCS2404 - DATABASE MANAGEMENT SYSTEMS
Regulations – R2021

Project Report

Pharmacy supply chain Management system

Normalization

G HARIKUMAR (3122 22 5001 033)
ADITYA JYOSYULA (3122 22 5001 006)

1)Pharmacy:

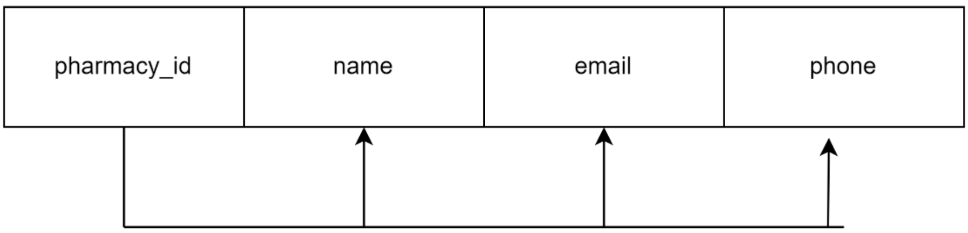
Minimal Set of FDS:

FD1 : pharmacy_id-> pharmacy_name, pharmacy_email , pharmacy_phone, supplier_id

pharmacy

pharmacy_id	name	email	phone

pharmacy



Checking 1 NF:

Conditions:

- Domain of the attribute must include only atomic values.
- No multivalued ,composite attributes and nested relations allowed.

There is no multivalued attributes in the table .

So the relational schema(pharmacy) is in 1 NF.

Checking for 2 NF :

Conditions:

- A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key OF R.

In this relation there is no partial functional dependency. As a partial functional dependency occurs when a non-prime attribute (an attribute not part of any candidate key) is functionally determined by only part of a candidate key. Here there is no such case.

Therefore the relational schema(pharmacy) is in 2 NF.

Checking for 3 NF:

Conditions:

- A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key of R.

Here as there are no transitive dependencies pharmacy table is in 3NF also.

Now the table is in 3NF.

PHARMACY 1NF

↓

PHARMACY 2NF

↓

PHARMACY 3NF

2) CUSTOMER:

CUSTOMER

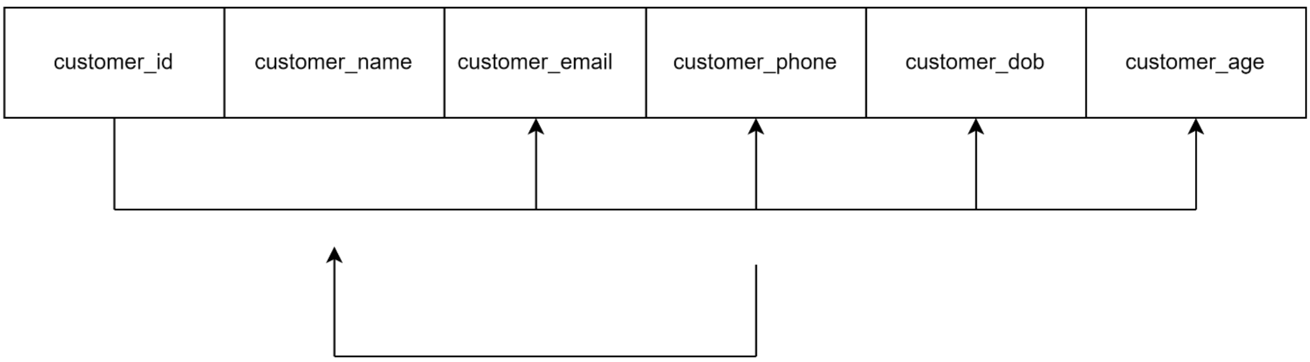
customer_id	customer_name	customer_email	customer_phone	customer_dob	customer_age

Minimal set of FDS:

FD1:customer_id ->email,phone,dob

FD2:phone -> name

customer1



Conditions:

- Domain of the attribute must include only atomic values.
- No multivalued ,composite attributes and nested relations allowed.

So it satisifes 1NF.

Checking for 2 NF :

Conditions:

- A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key OF R.

In this relation there is no partial functional dependency. As a partial functional dependency occurs when a non-prime attribute (an attribute not part of any candidate key) is functionally determined by only part of a candidate key. Here there is no such case.

Therefore the relational schema(Customer) is in 2 NF.

Checking for 3 NF:

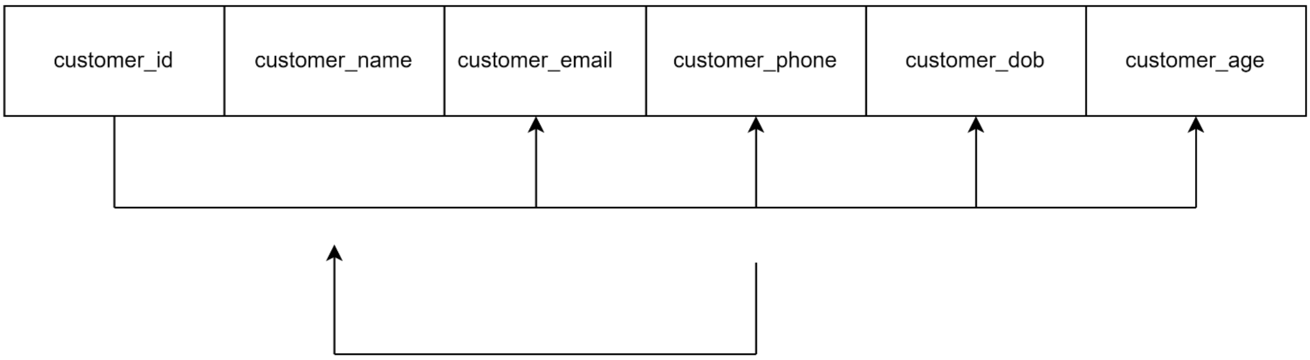
Conditions:

- A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key of R.

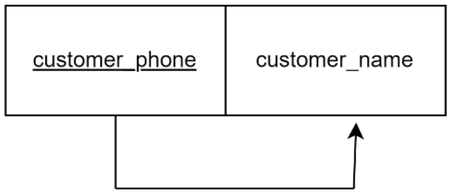
Here *customer_id ->dob* and *dob -> age* therefore *customer_id -> age* is a transitive also in customer 2 *customer_id->customer_phone* and *customer_phone->customer_name* is also transitive.

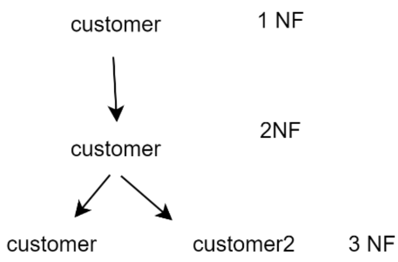
So the table is not int 3 NF.

customer1



customer2



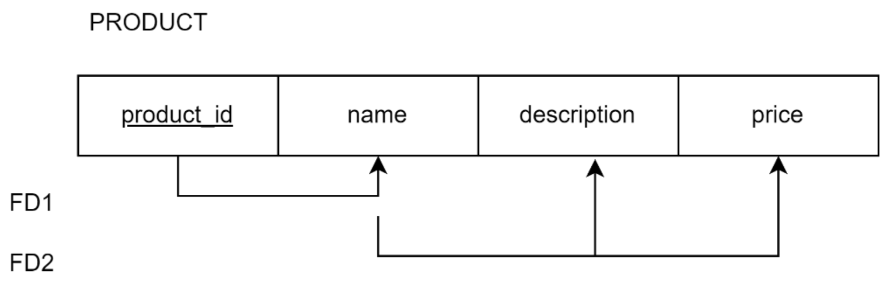


3) PRODUCT:

a) product_id
b) name
c) description
d) price

Minimal set of FDS:

FD1:product_id -> name
FD2:name -> description,price



Checking 1 NF:

Conditions:

- Domain of the attribute must include only atomic values.
- No multivalued ,composite attributes and nested relations allowed.

Here all the attributes are indivisible(atomic) and there is no multivalued, composite attributes and Nested relations.

So the relational schema(Product) is in 1 NF.

Checking for 2 NF :

Conditions:

- A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key OF R.

In this relation there is no partial functional dependency. As a partial functional dependency occurs when a non-prime attribute (an attribute not part of any candidate key) is functionally determined by only part of a candidate key. Here there is no such case.

Therefore the relational schema(Product) is in 2 NF.

Checking for 3 NF:

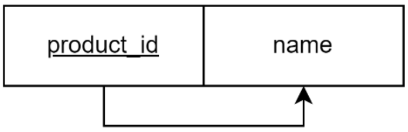
Conditions:

- A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key of R.

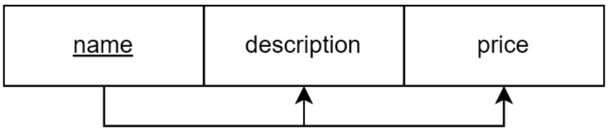
Here *product_id -> name* and *name-> description,price* therefore *product_id-> name* is a transitive So the table is not int 3 NF.

Decomposition:

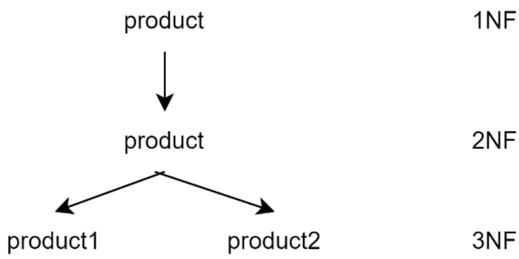
PRODUCT1



PRODUCT2



Now the relational schema(Product) is in 3NF



4)SHIPMENT:

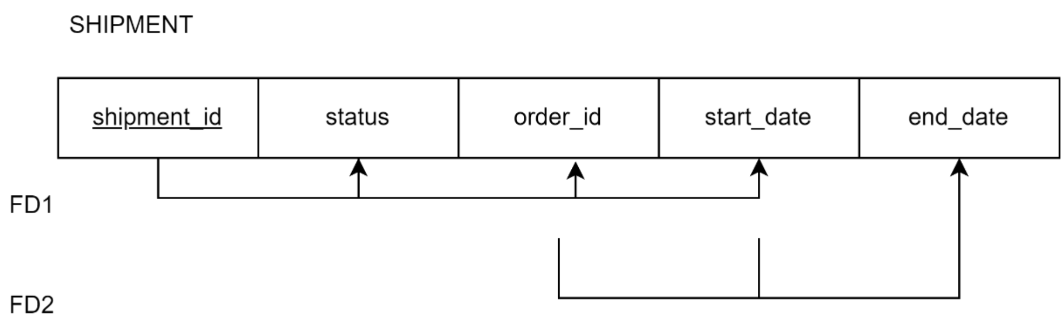
f) shipment_id
g) order_id
h) start_date
i) end_date
j) status

SHIPMENT

shipment_id	status	order_id	start_date	end_date

Minimal Set of FD:

FD1: shipment_id -> order_id,start_date,status
FD2: order_id,start_date->end_date



Checking 1 NF:

Conditions:

- Domain of the attribute must include only atomic values.
- No multivalued ,composite attributes and nested relations allowed.

Here all the attributes are indivisible(atomic) and there is no multivalued, composite attributes and Nested relations.

So the relational schema(Shipment) is in 1 NF.

Checking for 2 NF :

Conditions:

- A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key OF R.

In this relation there is no partial functional dependency. As a partial functional dependency occurs when a non-prime attribute (an attribute not part of any candidate key) is functionally determined by only part of a candidate key. Here there is no such case.

Therefore the relational schema(Shipment) is in 2 NF.

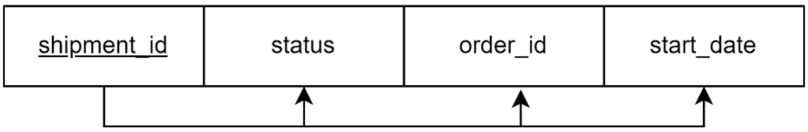
Checking for 3 NF:

Conditions:

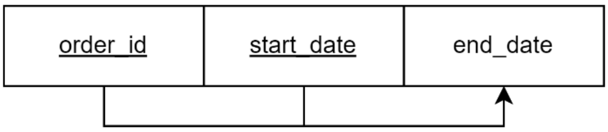
- A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key of R.

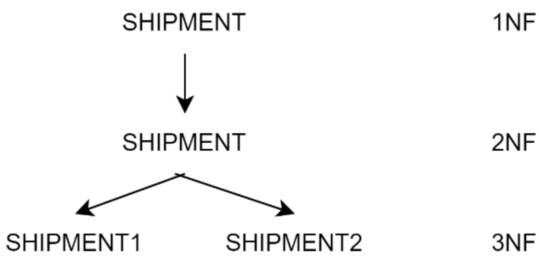
Here *shipment_id -> order_id,start_date* and *order_id,start_date-> end_date* therefore *shipment_id-> order_id,start_date* is a transitive
So the table is not int 3 NF.

SHIPMENT 1



SHIPMENT2



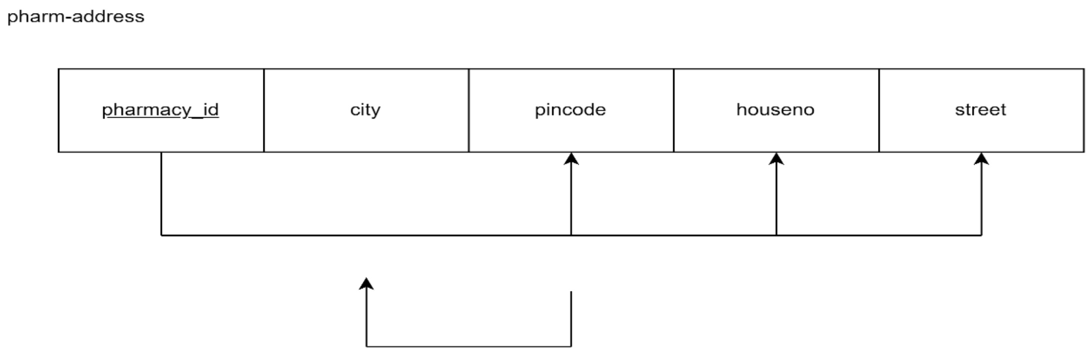


5)Pharmacy-ADDRESS:

g) city
h) pincode
i) house_no
j) street
k) pharmacy_id

Minimal Set of FDS:

FD1: pharmacy_id -> street,city,pincode,houseno
FD2: street -> city
FD3: customer_id -> street,city,pincode,houseno
FD4:_house_no -> street
FD5: pincode -> city



Checking 1 NF:

Conditions:

- Domain of the attribute must include only atomic values.
- No multivalued ,composite attributes and nested relations allowed.

Here all the attributes are indivisible(atomic) and there is no multivalued, composite attributes and Nested relations.

So the relational schema(ADDRESS) is in 1 NF.

Checking for 2 NF :

Conditions:

- A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key OF R.

So the relational schema(ADDRESS) is in 2 NF.

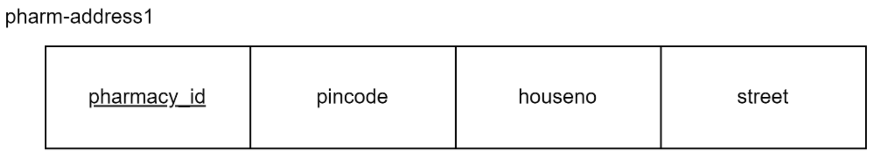
Now the relational schema (Order_items) is in 2NF.

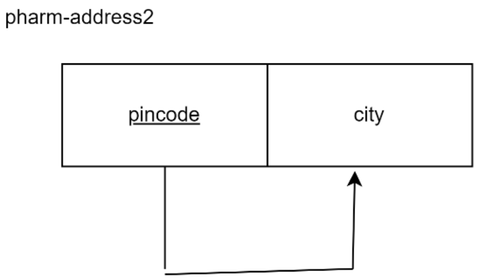
Checking for 3 NF:

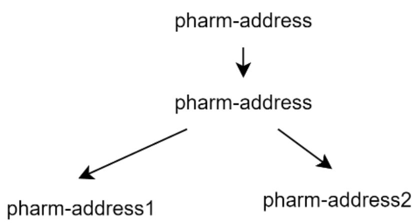
Conditions:

- A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key of R.

In this relation there is no fd that can be transitive as there is no non-prime attribute involved in the fd. So the relational schema(ADDRESS) is in 3 NF.







6) INVENTORY:

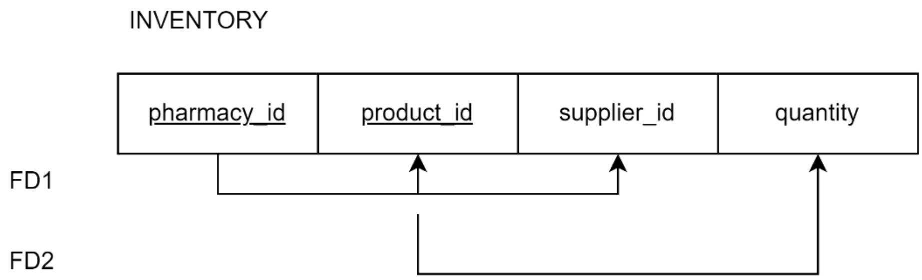
e) pharmacy_id
f) product_id
g) supplier_id
h) quantity

INVENTORY

pharmacy_id	product_id	supplier_id	quantity

Minimal Set of FDS:

FD1:pharmacy_id,product_id->supplier_id
FD2:product_id->quantity



Checking 1 NF:

Conditions:

- Domain of the attribute must include only atomic values.
- No multivalued ,composite attributes and nested relations allowed.

Here all the attributes are indivisible(atomic) and there is no multivalued, composite attributes and Nested relations.

So the relational schema(INVENTORY) is in 1 NF.

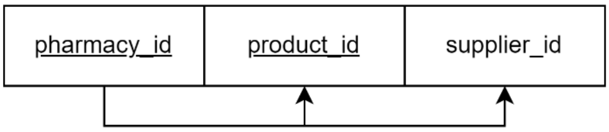
Checking for 2 NF :

Conditions:

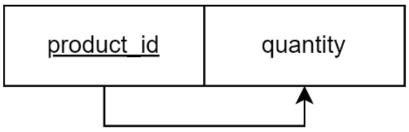
- A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key OF R.

So the relational schema(ADDRESS) is not in 2 NF.

INVENTORY1



INVENTORY2



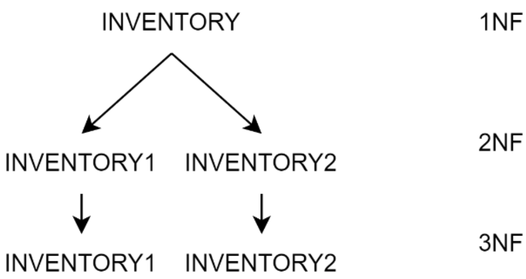
Now the table (INVENTORY) is in 2NF.

<u>Checking for 3 NF:</u>

Conditions:

- A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key of R.

In this relation there is no fd that can be transitive as there is no non-prime attribute involved in the fd. So the relational schema(INVENTORY) is in 3 NF.



7)PAYMENTS:

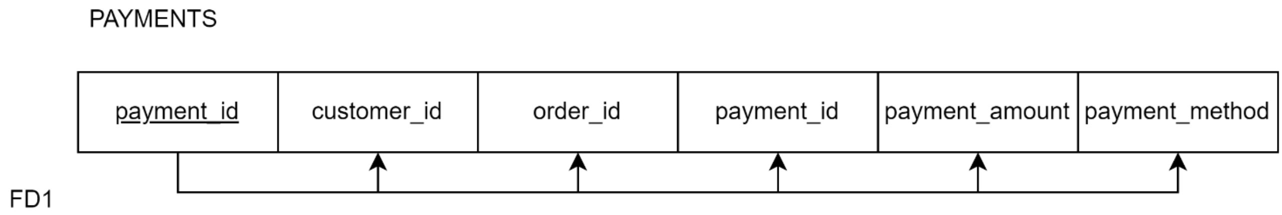
g) payment_id
h) cus_id
i) order_id
j) payment_date
k) payment_amount
l) payment_method

PAYMENTS

payment_id	customer_id	order_id	payment_id	payment_amount	payment_method

Minimal Set of FDS:

FD1: payment_id->customer_id,order_id,payment_date,payment_amount,payment_method



Checking 1 NF:

Conditions:

- Domain of the attribute must include only atomic values.
- No multivalued ,composite attributes and nested relations allowed.

Here all the attributes are indivisible(atomic) and there is no multivalued, composite attributes and Nested relations.

So the relational schema(PAYMENT) is in 1 NF.

Checking for 2 NF :

Conditions:

- A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key OF R.

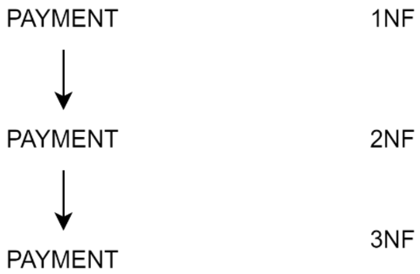
So the relational schema(PAYMENT) is in 2 NF.

Checking for 3 NF:

Conditions:

- A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key of R.

In this relation there is no fd that can be transitive as there is no non-prime attribute involved in the fd. So the relational schema(PAYMENT) is in 3 NF.

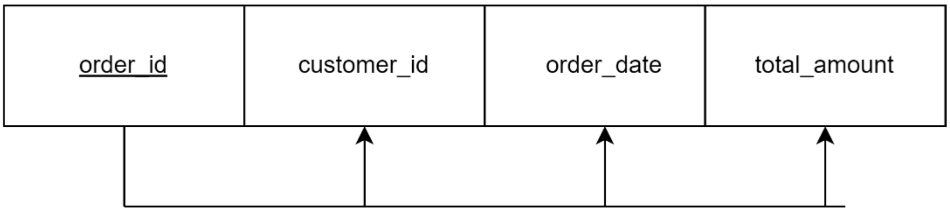


8)ORDERS:

h) order_id
i) customer_id
j) order_date
k) total_amount

<u>Minimal Set of FDS:</u>

FD1: order_id -> customer_id,order_date, product_id, total amount



<u>Checking 1 NF:</u>

Conditions:

- Domain of the attribute must include only atomic values.
- No multivalued ,composite attributes and nested relations allowed.

Here there is no multivalued attruibutes

So the relational schema(ORDERS) is in 1 NF.

<u>Checking for 2 NF :</u>

Conditions:

- A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key OF R.

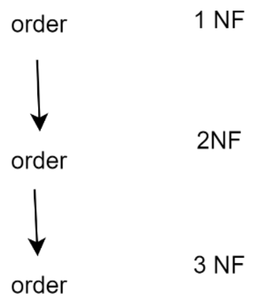
Here the is no partial FDs .so the table is in 2NF.

Checking for 3 NF:

Conditions:

- A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key of R.

In this relation there is no fd that can be transitive as there is no non-prime attribute involved in the fd. So the relational schema(ORDERS) is in 3 NF.



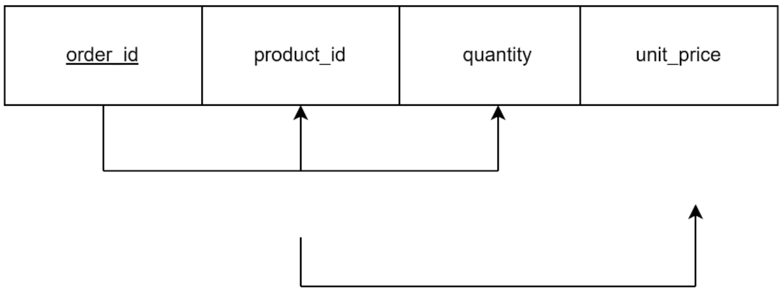
9)ORDER_ITEMS:

a) order_id
b) product_id
c) quantity
d) unit_price

minimal set of Fds:

order_id ,product_id-> quantity
product_id -> unit_price

Order_items



<u>Checking 1 NF:</u>

Conditions:

- Domain of the attribute must include only atomic values.
- No multivalued ,composite attributes and nested relations allowed.

Here there is no multivalued attruibutes

So the relational schema(ORDER_ITEMS) is in 1 NF.

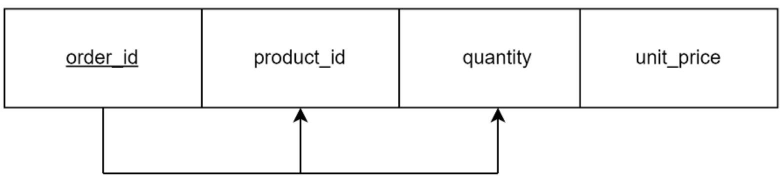
<u>Checking for 2 NF :</u>

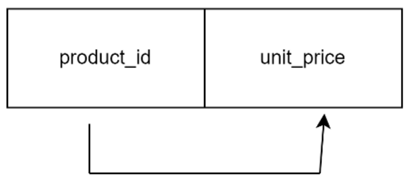
Conditions:

- A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key OF R.

Here the is partial FDs .so the table is not in 2NF.
So decomposition.

Order_items1



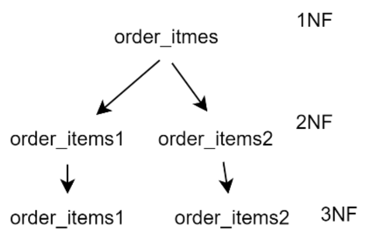


<u>Checking for 3 NF:</u>

Conditions:

- A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key of R.

In this relation there is no fd that can be transitive as there is no non-prime attribute involved in the fd. So the relational schema(ORDERS_ITEMS) is in 3 NF.



10)SUPPLIER:

f) supplier_id
g) name
h) email
i) phone
j) pharmacy_id

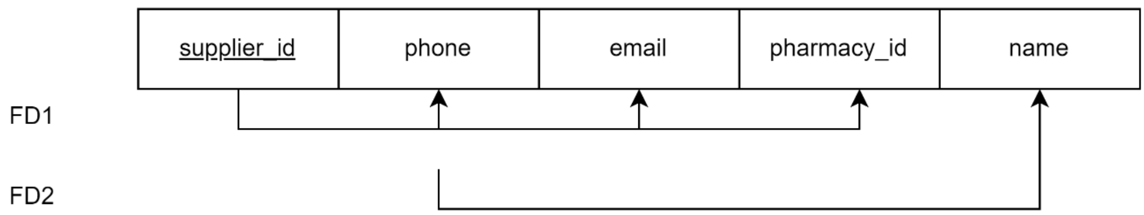
SUPPLIER

supplier_id	name	email	phone	pharmacy_id

The minimal set of FDs.

{
supplier_id -> email,phone,pharmacy_id
phone->name
}

SUPPLIER



Checking 1 NF:

Conditions:

- Domain of the attribute must include only atomic values.
- No multivalued ,composite attributes and nested relations allowed.

Here there is no multivalued attruibutes

So the relational schema(SUPPLIER) is in 1 NF.

Checking for 2 NF :

Conditions:

- A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the primary key OF R.

Here the is no partial FDs .so the table is in 2NF.

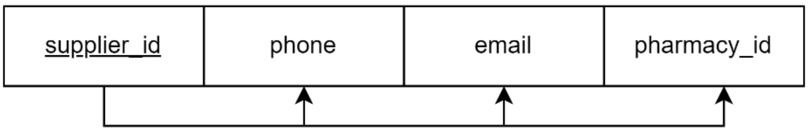
Checking for 3 NF:

Conditions:

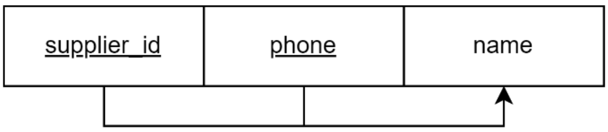
- A relation schema R is in third normal form (3NF) if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key of R.

Here *supplier_id -> phone* and *phone-> name* therefore *supplier_id-> name* is a transitive
So the table is not int 3 NF.

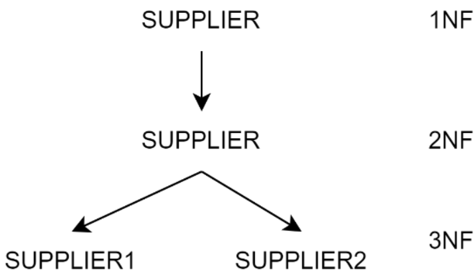
SUPPLIER1



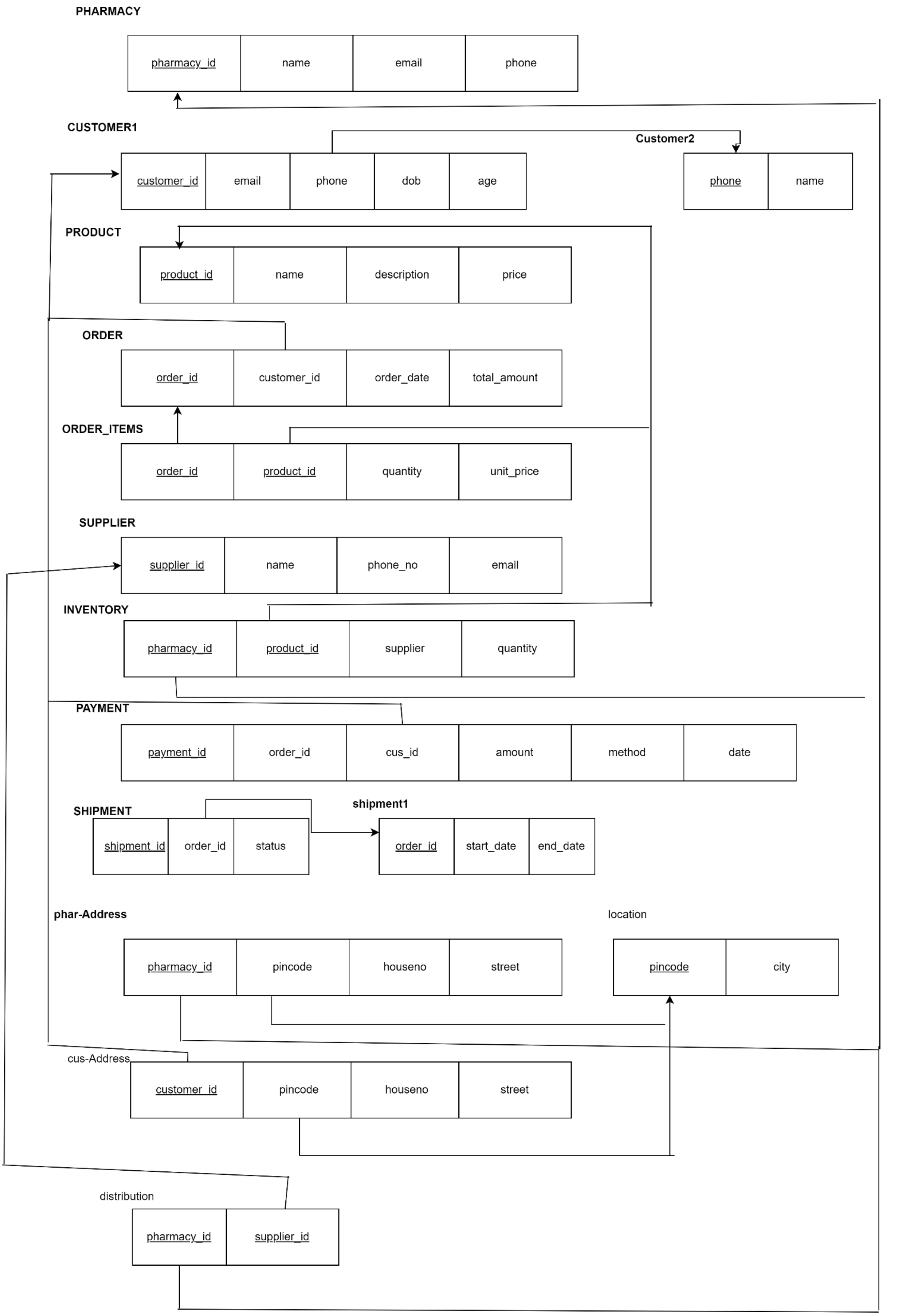
SUPPLIER2



Now the table is in 3NF.

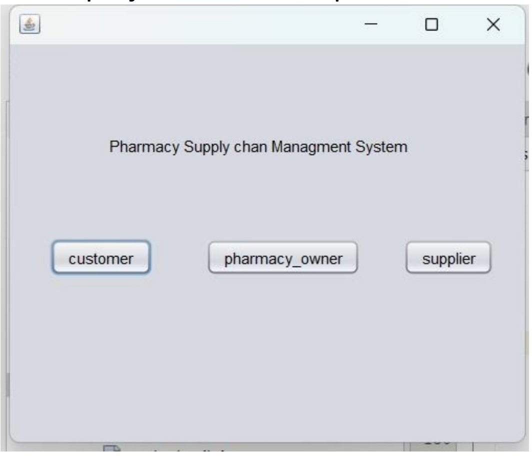


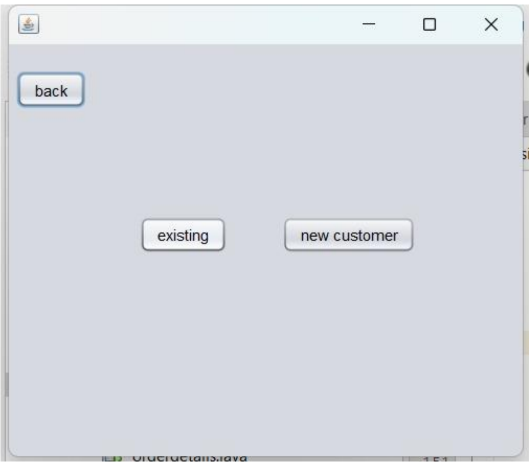
After normalization Schema:

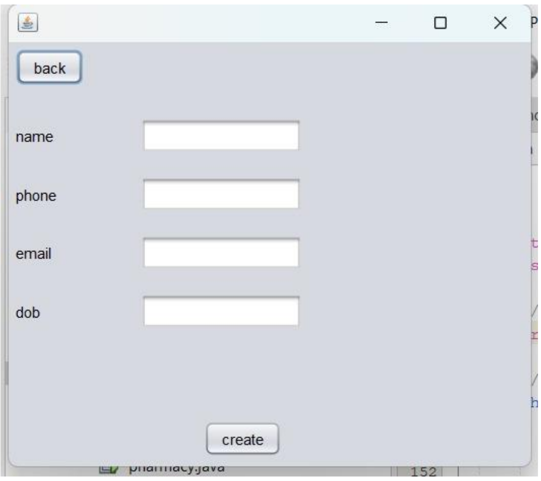

PHARMACY
pharmacy_id
name
email
phone
CUSTOMER1
customer_id
email
phone
dob
age
Customer2
phone
name
PRODUCT
product_id
name
description
price
ORDER
order_id
customer_id
order_date
total_amount
ORDER_ITEMS
order_id
product_id
quantity
unit_price
SUPPLIER
supplier_id
name
phone_no
email
INVENTORY
pharmacy_id
product_id
supplier
quantity
PAYMENT
payment_id
order_id
cus_id
amount
method
date
SHIPMENT
shipment_id
order_id
status
shipment1
order_id
start_date
end_date
phar-Address
pharmacy_id
pincode
houseno
street
location
pincode
city
cus-Address
customer_id
pincode
houseno
street
distribution
pharmacy_id
supplier_id

IMPLEMENTATION:

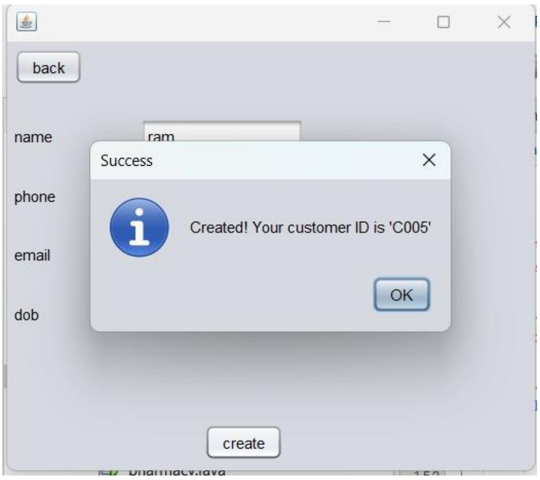
In our project we have implemented the following cases:





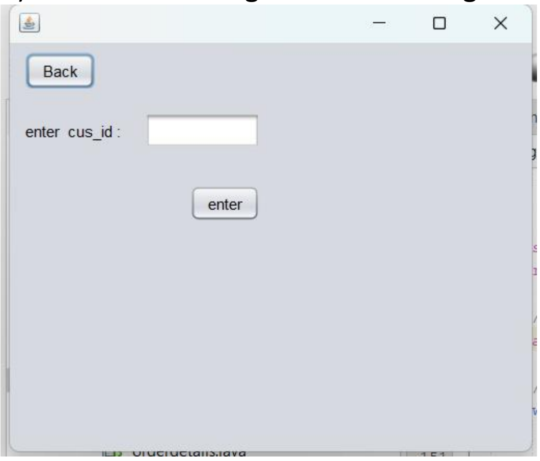


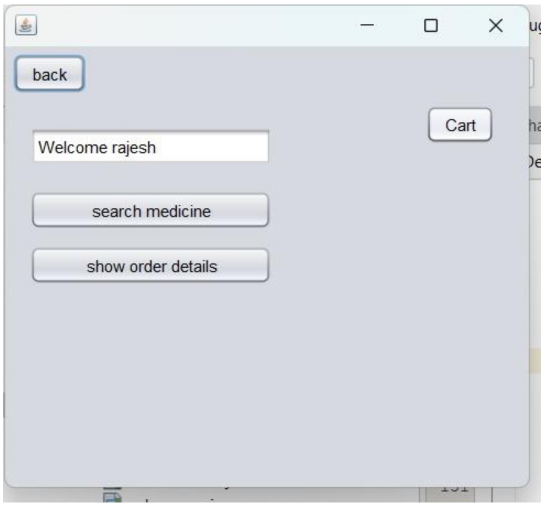
NEW CUSTOMER REGISTRATION



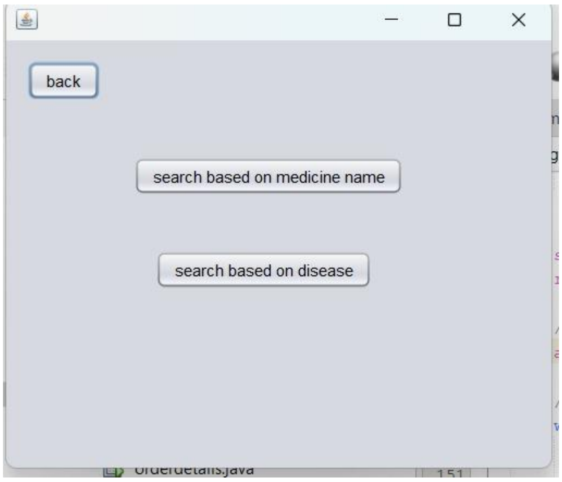
Customer:

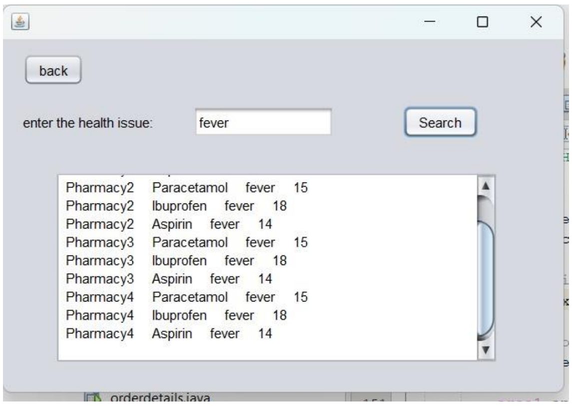
1) The customer logs into the management system :

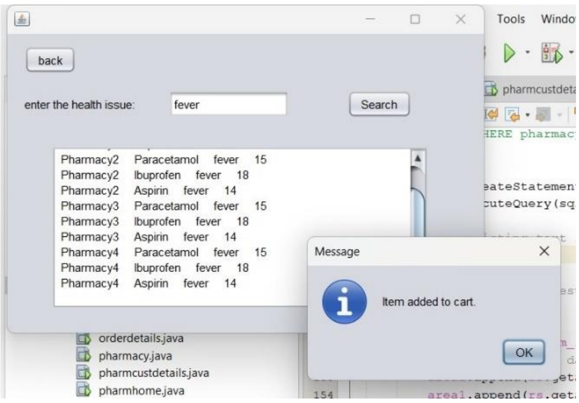


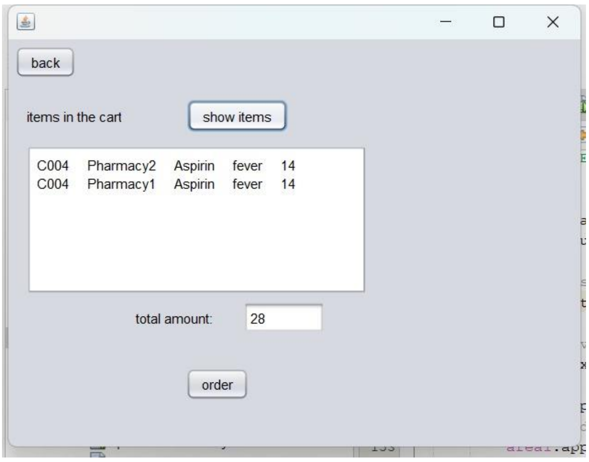


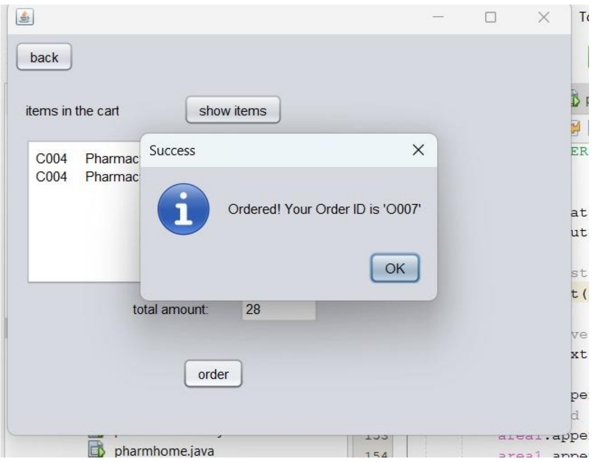
2) The customer can place an order of medicines from n number of pharmacies:


back
search based on medicine name
search based on disease

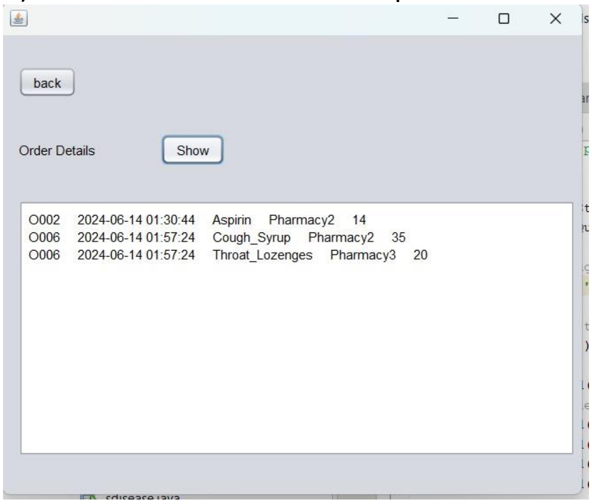

back
enter the health issue:
fever
Search
Pharmacy2 Paracetamol fever 15
Pharmacy2 Ibuprofen fever 18
Pharmacy2 Aspirin fever 14
Pharmacy3 Paracetamol fever 15
Pharmacy3 Ibuprofen fever 18
Pharmacy3 Aspirin fever 14
Pharmacy4 Paracetamol fever 15
Pharmacy4 Ibuprofen fever 18
Pharmacy4 Aspirin fever 14


back
enter the health issue:
fever
Search
Pharmacy2 Paracetamol fever 15
Pharmacy2 Ibuprofen fever 18
Pharmacy2 Aspirin fever 14
Pharmacy3 Paracetamol fever 15
Pharmacy3 Ibuprofen fever 18
Pharmacy3 Aspirin fever 14
Pharmacy4 Paracetamol fever 15
Pharmacy4 Ibuprofen fever 18
Pharmacy4 Aspirin fever 14
Message
Item added to cart.
OK
orderdetails.java
pharmacy.java
pharmcustdetails.java
pharmhome.java


back
items in the cart
show items
C004 Pharmacy2 Aspirin fever 14
C004 Pharmacy1 Aspirin fever 14
total amount:
28
order

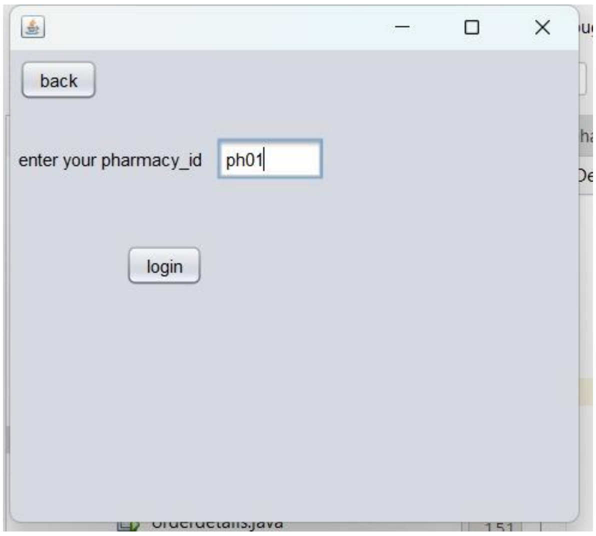


3) The customer can view the previous orders that he/she has placed:

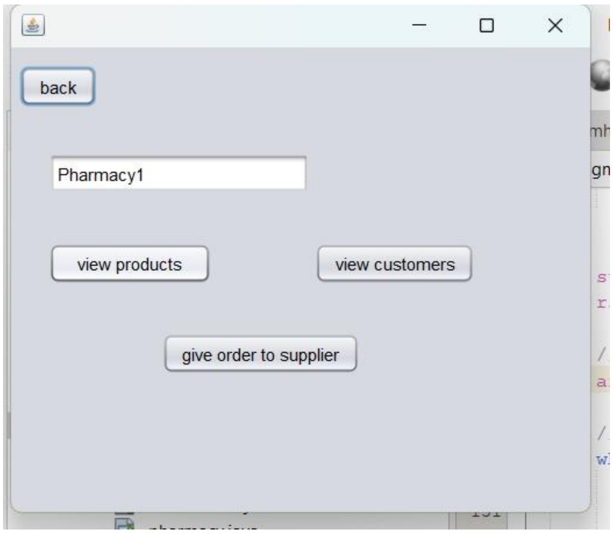


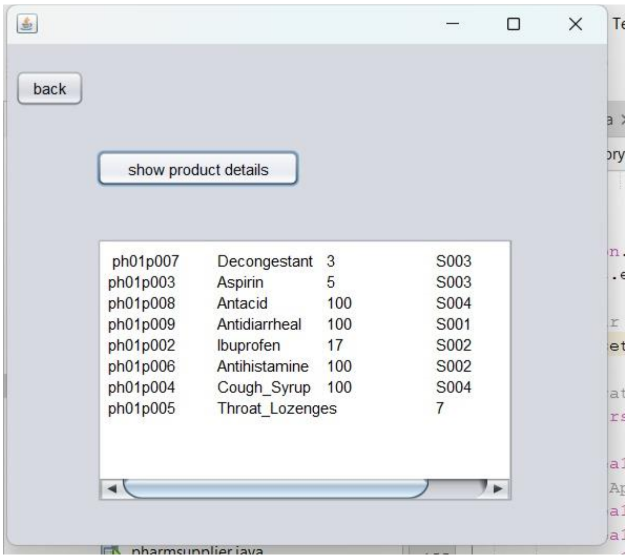
Pharmacy:

1) The pharmacy can log into the management system:

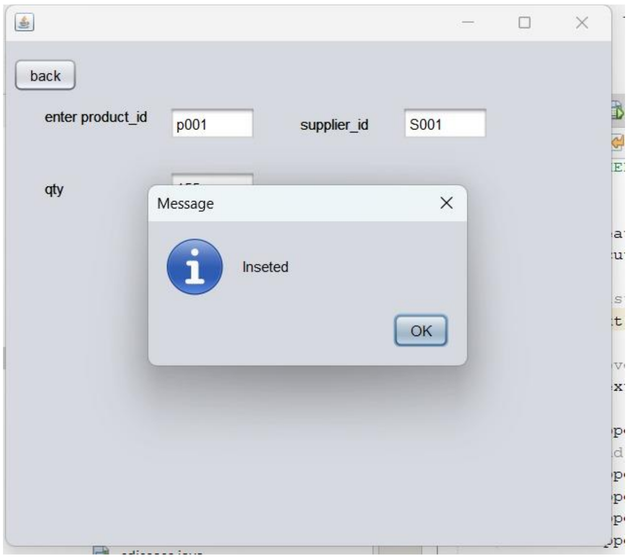


2) The pharmacy can view the product details already available in the pharmacy:

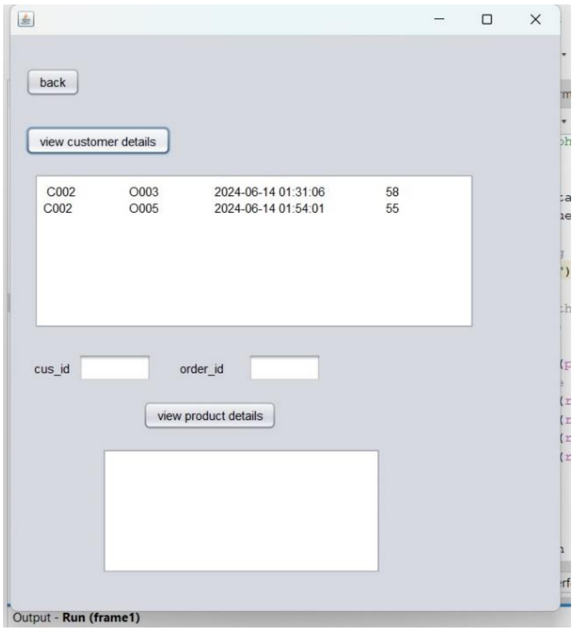




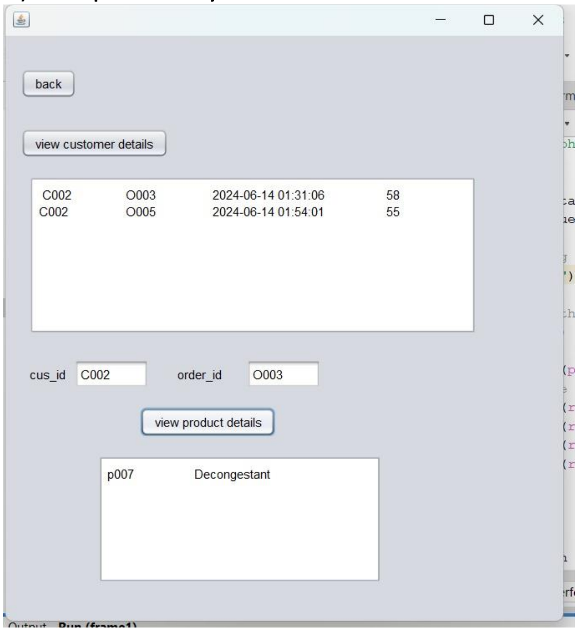
3) The pharmacy can request more products from the supplier:



4) The pharmacy can view the customers who have previously ordered from the pharmacy:

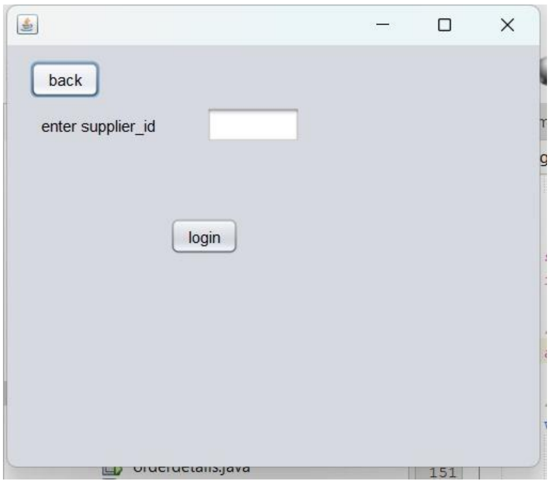


5) The pharmacy can also view the order details of previous customer:

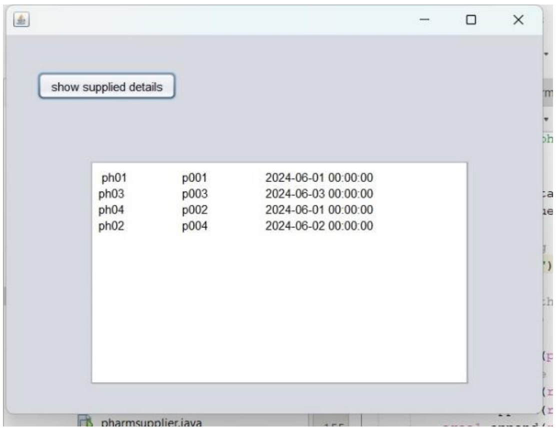


Supplier:

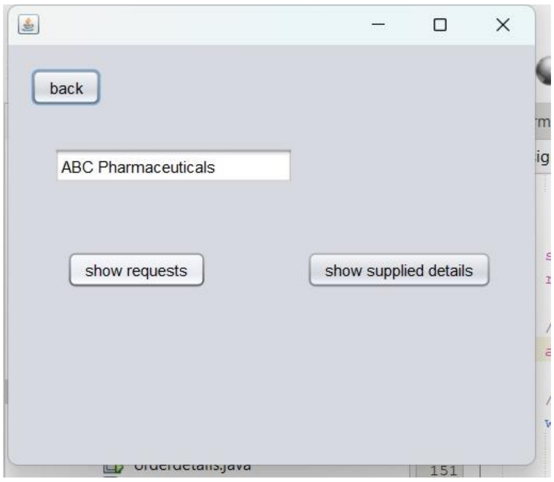
1) The supplier can log into the management system:



2) The supplier can view the pharmacies to which they have previously supplier medicines to:



3) The supplier can view the requests from various pharmacies and accept the request to deliver medicines:



IMPORTANT VIEWS:

1)
CREATE VIEW MedicineInventoryView AS
SELECT ph.pharmacy_id, ph.name AS pharmacy_name, p.product_id, p.name AS product_name, i.quantity,p.description as description,p.price as price

```
FROM product p
JOIN inventory i ON p.product_id = i.product_id
JOIN pharmacy ph ON i.pharmacy_id = ph.pharmacy_id;
```

This view is used to retrieve information about the availability of medicines across different pharmacy locations.

2)

```
CREATE VIEW pharmacy_inventory_summary AS
SELECT i.pharmacy_id, i.product_id, p.name, SUM(i.quantity) AS total_quantity, i.supplier_id
FROM inventory i
JOIN product p ON p.product_id = i.product_id
GROUP BY i.pharmacy_id, i.product_id, p.name, i.supplier_id;
```

This view is used to retrieve the products that are available in the inventory of the pharmacy so that the pharmacy can view product details.

IMPORTANT PROCEDURES AND FUNCTIONS:

1)

```
CREATE OR REPLACE PROCEDURE InsertCustomer (
   p_name IN VARCHAR2,
   p_phone IN VARCHAR2,
   p_email IN VARCHAR2,
   p_dob IN DATE,
   p_new_id OUT VARCHAR2
) IS
   last_id NUMBER;
BEGIN
   -- Get the last inserted ID
   SELECT COUNT(*) INTO last_id FROM customer;

   -- Calculate the new ID
   p_new_id := 'C' || LPAD(last_id + 1, 3, '0');

   -- Insert the new customer record
   INSERT INTO customer (cus_id, name, email, phone, dob)
   VALUES (p_new_id, p_name, p_email, p_phone, p_dob);

   -- Commit the transaction
   COMMIT;

   -- Display success message or perform any additional tasks if needed
   DBMS_OUTPUT.PUT_LINE('Customer inserted successfully with ID: ' || p_new_id);
END;
```

/

EXPLANATION:

The above procedure is used to assign a customer id to the new customer who has registered into the database.

2)

```
CREATE OR REPLACE PROCEDURE CreateOrderAndInsertItems (
   o_cust_id IN VARCHAR2,
   o_totalamount IN NUMBER,
   p_new_id OUT VARCHAR2
) IS
   last_id NUMBER;
BEGIN
   -- Get the last inserted ID
   SELECT COUNT(*) INTO last_id FROM orders;

   -- Calculate the new ID
   p_new_id := 'O' || LPAD(last_id + 1, 3, '0');

   -- Insert the new order record
   INSERT INTO orders (order_id, cus_id, order_date, totat_amount)
   VALUES (p_new_id, o_cust_id, SYSDATE, o_totalamount);

   -- Debug message
   DBMS_OUTPUT.PUT_LINE('Inserted order with ID: ' || p_new_id);

   -- Retrieve values from cart and insert into order_items
   FOR cart_row IN (SELECT product_name, pharmacy_name, price
                FROM cart
                WHERE cus_id = o_cust_id)
   LOOP
      DECLARE
         v_product_id VARCHAR2(5);
         v_pharmacy_id VARCHAR2(5);
      BEGIN
         -- Get product_id
         BEGIN
            SELECT product_id INTO v_product_id
            FROM product
            WHERE name = cart_row.product_name;
            DBMS_OUTPUT.PUT_LINE('Retrieved product_id: ' || v_product_id || ' for product_name: ' ||
cart_row.product_name);
         EXCEPTION
            WHEN NO_DATA_FOUND THEN
               DBMS_OUTPUT.PUT_LINE('Product with name ' || cart_row.product_name || ' not found.');
```

```
            RAISE;
        END;

        -- Get pharmacy_id
        BEGIN
          SELECT pharmacy_id INTO v_pharmacy_id
          FROM pharmacy
          WHERE name = cart_row.pharmacy_name;
          DBMS_OUTPUT.PUT_LINE('Retrieved pharmacy_id: ' || v_pharmacy_id || ' for
pharmacy_name: ' || cart_row.pharmacy_name);
        EXCEPTION
          WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Pharmacy with name ' || cart_row.pharmacy_name || ' not
found.');
            RAISE;
        END;

        -- Insert into order_items
        BEGIN
          INSERT INTO order_items (order_id, product_id, pharmacy_id, quantity, unit_price)
          VALUES (p_new_id, v_product_id, v_pharmacy_id, 1, cart_row.price);
          DBMS_OUTPUT.PUT_LINE('Inserted order item: ' || v_product_id || ', ' || v_pharmacy_id);
        EXCEPTION
          WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error inserting order item: ' || SQLERRM);
            RAISE;
        END;
      END;
  END LOOP;

  -- Commit the transaction
  COMMIT;

  -- Display success message
  DBMS_OUTPUT.PUT_LINE('Order inserted successfully with ID: ' || p_new_id);
EXCEPTION
  WHEN OTHERS THEN
    -- Rollback the transaction in case of error
    ROLLBACK;
    -- Display error message
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

EXPLANATION:

The above procedure assigns a new and unique product id and order id to every new product that has been added and every new order that has been placed respectively. It also inserts the orders into the orders table from which a customer can view their cart i.e, they can check what they wish to order and so can select specific items to be ordered from the cart.

3)

```
CREATE OR REPLACE PROCEDURE DeleteItemsFromCart (
   o_cust_id IN VARCHAR2
) IS
BEGIN
   -- Delete values from cart
   DELETE FROM cart WHERE cus_id = o_cust_id;

   -- Commit the transaction
   COMMIT;

   -- Display success message
   DBMS_OUTPUT.PUT_LINE('Deleted items from cart for customer: ' || o_cust_id);
EXCEPTION
   WHEN OTHERS THEN
      -- Rollback the transaction in case of error
      ROLLBACK;
      -- Display error message
      DBMS_OUTPUT.PUT_LINE('Error deleting from cart: ' || SQLERRM);
END;
/
```

EXPLANATION:

This procedure makes sure to delete the items from the cart of the customer and adds the ordered items to the order_items table which can showcase the orders of the customers to the pharmacy or to the customer itself.

NOVELTY:

The novelty of our project includes the fact that we have generated unique non overlapping values for the ids of each customer ,pharmacy and supplier.

We have also included that pharmacy can request extra medicines from the supplier once they run out and the supplier can accept the pharmacy’s requests.

Another important feature is that we have made sure to display previous orders/ customers / products / requests and accepts for customers, pharmacy and supplier respectively.
We have also generated total amounts for the orders and products where needed.