**PHARMACY SUPPLY CHAIN MANAGEMENT SYSTEM**

UCS2304 – Database Management Systems

Documentation

Submitted By

Prithivirajan D (3122 22 5001 099)

Rishab S (3122 22 5001 104)

Rithekha K (3122 22 5001 106)

A blue and white logo

Description automatically generated

Department of Computer Science and Engineering

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

Kalavakkam – 603110

2023-24

**PHARMACY SUPPLY CHAIN MANAGEMENT SYSTEM**

**Assumptions:**

1)

* Medical companies have factories to produce medicines.
* The medicines produced by the factory are stored in a Central WareHouse(CWH).
* Now, hospitals order certain quantities of medicine to Distributors.
* Distributors are allocated based on location.
* Agents acquire a certain amount of medicine stock from the CWH.
* Distributors act like middlemen between pharmacies and agents, and acquire the medicine from agents.
* Each factory has 1 warehouse
* Agents will the people who maintains the orders.

2)

* Medical companies also have a marketing team(Employees) that will reach out to the doctors to promote the products they have produced. If the doctor is satisfied with the product, they will sign a contract with the medical company to buy these products for a period of time.

3)

* E-Pharmacy retail chains like Apollo and MedPlus give bulk orders to agents, who in turn will distribute the required supplies to each Apollo/MedPlus store.
* No external middleman(distributors).

4)

* Customers buys medicine from hospital or E-pharmacy.

**RELATIONS:**

1. Medical company
2. Employee
3. Distributor
4. Agent
5. Hospital
6. E\_pharmacy
7. Customer
8. Medicine
9. Orders
10. Doctor

**ATTRIBUTES:**

**1. Medical Company**

Mc\_id (PK)

Location

F\_ID

F\_loc

Capacity

Number of Employees

CWH ID

Location

CWH\_Capacity

**2. Agent**

A\_ID (PK)

A\_name

A\_Loc

CWH\_ID (FK)

A\_contact

**3. Distributor**

D\_ID (PK)

A\_ID(FK)

D\_Loc

D\_Contact

O\_ID

**4. Hospital**

H\_ID (PK)

H\_Name

D\_ID(FK)

H\_Contact

**5. E\_Pharmacy**

EPH\_ID (PK)

EPH\_Name

A\_ID(FK)

EPH\_Contact

**6. Customer**

Cus\_ID (PK)

Cus\_name

Dob

Age

Cus\_add

**7.Medicine**

M\_ID (PK)

M\_Name

M\_Date

Price

F\_ID (FK)

Expiry\_Date

**8. Orders**

O\_ID(PK)

M\_ID

A\_ID(FK)

Ordered\_by

Quantity

Order date

Delivery date

Total\_price

**9. Doctor**

Doc\_Lisense

Doc\_name

Specialization

Qualification

h\_id

**10. Employee**

E\_id

E\_name

Dob

Age

Salary

Position

E\_contact

Calls

Sales

**Relations and Attributes:**

**1 . A diagram of a company

Description automatically generated**

**2 .** **A diagram of a company

Description automatically generated**

3 . A diagram of a doctor

Description automatically generated

4. A diagram of hospitals

Description automatically generated

5. A diagram of medicine with many words

Description automatically generated

6. A diagram of a customer

Description automatically generated

7. A diagram of a pharmacy

Description automatically generated

8 . A diagram of a order

Description automatically generated

9 . A diagram of a company

Description automatically generated

10 . A diagram of a company

Description automatically generated

**Relationships and Structural Constraints:**

1. **A white square with black text

   Description automatically generated**
2. **A white diamond with black text

   Description automatically generated**
3. **A white diamond with black text

   Description automatically generated**
4. **A white diamond with black text

   Description automatically generated**
5. **A diagram of a diamond

   Description automatically generated**
6. **A white diamond with black text

   Description automatically generated**
7. **A white diamond with black text

   Description automatically generated**
8. **A diagram of a sales diagram

   Description automatically generated**
9. **A diamond with a word on it

   Description automatically generated with medium confidence**
10. **A graph with a white rhombus

    Description automatically generated**
11. **A white diamond with black text

    Description automatically generated**
12. **A white diamond with black text

    Description automatically generated**
13. **A diagram of a order to

    Description automatically generated**

**Entity Relationship Diagram:**

A diagram of a company

Description automatically generated

**Functional Dependencies:**

1. **Employee:**

**Attributes**

{e\_id, e\_name, dob, e\_contact, position, salary, sales, age, calls}

**Possible FDs:**

* Employee ID (e\_id) uniquely determines all other attributes: Assuming that e\_id is a unique identifier for each employee, it should determine all other attributes.

**e\_id→ {e\_name, dob, e\_contact, position, salary, sales, age, calls}**

* Date of birth (dob) determines age: Age can be derived from the date of birth.

**dob→age**

* Position might determine salary range: If there are standard salaries associated with each position.

**position→salary**

* Employee contact (e\_contact) uniquely identifies an employee: If each employee has a unique contact number.

**e\_contact→ {e\_id, e\_name, dob, position, salary, sales, age, calls}**

Here, e\_contact may change in the future, so we are removing e\_contact→ {e\_id, e\_name, dob, position, salary, sales, age, calls}

.

**Irreducible Set of Fds:**

To find the irreducible set of FDs, we need to ensure that each FD is in its simplest form, with minimal left-hand side attributes, and no extraneous dependencies. This involves a few steps: removing redundant attributes from the left-hand side and ensuring that the FDs are non-redundant.

**The FDs are:**

​(a) e\_id→ {e\_name, dob, e\_contact, position, salary, sales, age, calls}

(c) dob→age

(d) position→salary

**Decomposition:**

(a)

e\_id→e\_name

e\_id→dob

e\_id→e\_contact

e\_id→position

e\_id→salary

e\_id→sales

e\_id→age

e\_id→calls

(b) dob→age

(c) position→salary

**Remove extraneous attributes**

In this case, there are no left-hand sides with extraneous attributes since all are minimal.

**Check for redundant FDs:**

* If e\_id→dob, and dob→age then e\_id→age

e\_id→age is redundant.

**After removing redundant FDs:**

​(1) e\_id→e\_name

(2) e\_id→dob

(3) e\_id→e\_contact

(4) e\_id→position

(5) e\_id→salary

(6) e\_id→sales

(7) e\_id→calls

(8) dob→age

(9) position→salary

**Primary key:**

We can use e\_id as the primary key (as it uniquely identifies all the other attributes of the relation).

1. **MEDICAL\_COMPANY**

**Attributes:**

mc\_id (Medical Company ID)

mc\_name (Medical Company Name)

mc\_loc (Medical Company Location)

f\_id (Factory ID)

f\_loc (Factory Location)

f\_capacity (Factory Capacity)

f\_employees (Number of Employees in Factory)

cwh\_id (Central Warehouse ID)

cwh\_loc (Central Warehouse Location)

cwh\_capacity (Central Warehouse Capacity)

**Possible Functional Dependencies (FDs):**

Medical Company ID (mc\_id) uniquely determines all other attributes of the medical company:

**mc\_id → {mc\_name, mc\_loc}**

Factory ID (f\_id) uniquely determines all attributes related to the factory:

**f\_id → {f\_loc, f\_capacity, f\_employees}**

Central Warehouse ID (cwh\_id) uniquely determines all attributes related to the central warehouse:

**cwh\_id → {cwh\_loc, cwh\_capacity}**

Medical Company ID (mc\_id) and Factory ID (f\_id) together uniquely determine the factory-related attributes:

**{mc\_id, f\_id} → {f\_loc, f\_capacity, f\_employees}**

Medical Company ID (mc\_id) and Central Warehouse ID (cwh\_id) together uniquely determine the central warehouse-related attributes:

**{mc\_id, cwh\_id} → {cwh\_loc, cwh\_capacity}**

**Irreducible Set of FDs:**

**The initial FDs are:**

(a) mc\_id → {mc\_name, mc\_loc}

(b) f\_id → {f\_loc, f\_capacity, f\_employees}

(c) cwh\_id → {cwh\_loc, cwh\_capacity}

(d) {mc\_id, f\_id} → {f\_loc, f\_capacity, f\_employees}

(e) {mc\_id, cwh\_id} → {cwh\_loc, cwh\_capacity}

**Decomposition:**

(a)

mc\_id → mc\_name

mc\_id → mc\_loc

(b)

f\_id → f\_loc

f\_id → f\_capacity

f\_id → f\_employees

(c)

cwh\_id → cwh\_loc

cwh\_id → cwh\_capacity

(d)

{mc\_id, f\_id} → f\_loc

{mc\_id, f\_id} → f\_capacity

{mc\_id, f\_id} → f\_employees

(e)

{mc\_id, cwh\_id} → cwh\_loc

{mc\_id, cwh\_id} → cwh\_capacity

**Removing Extraneous Attributes:**

**Check for redundant FDs:**

If f\_id → f\_loc, f\_id → f\_capacity, and f\_id → f\_employees hold, then {mc\_id, f\_id} → {f\_loc, f\_capacity, f\_employees} are redundant because the factory attributes are already uniquely determined by f\_id alone.

Similarly, if cwh\_id → cwh\_loc and cwh\_id → cwh\_capacity hold, then {mc\_id, cwh\_id} → {cwh\_loc, cwh\_capacity} are redundant because the warehouse attributes are already uniquely determined by cwh\_id alone.

**After removing redundant FDs:**

mc\_id → mc\_name

mc\_id → mc\_loc

f\_id → f\_loc

f\_id → f\_capacity

f\_id → f\_employees

cwh\_id → cwh\_loc

cwh\_id → cwh\_capacity

**Primary Key:**

The primary key for the relation can be mc\_id for the medical company attributes. However, since f\_id and cwh\_id uniquely identify their respective attributes, they serve as natural primary keys for the factory and central warehouse attributes. So Candidate key is {mc\_id, f\_id, cwh\_id}

1. **Doctor**

**Attributes:**

doc\_license (Doctor License)

doc\_name (Doctor Name)

qualification (Educational Qualification)

specialization (Area of Expertise)

h\_id (hospital ID)

I**nitial Functional Dependencies (FDs):**

Doctor license (doc\_license) uniquely determines all other attributes:

**doc\_license → {doc\_name, qualification, specialization,h\_id}**

Qualification determines specialization:

**qualification → specialization**

**Decomposition of Initial Functional Dependencies (FDs):**

1. Decomposing doc\_license → {doc\_name, qualification, specialization,h\_id}:

doc\_license → doc\_name

doc\_license → qualification

doc\_license → specialization

doc\_license → h\_id

2. Decomposing qualification → specialization:

qualification → specialization (no decomposition needed)

**Removing Redundant Functional Dependencies:**

Let's check for any redundant dependencies among the decomposed FDs.

doc\_license → doc\_name

doc\_license → qualification

doc\_license → specialization

doc\_license → h\_id

qualification → specialization

**Checking for Redundancies:**

Redundancy between doc\_license → qualification and qualification → specialization combined with doc\_license → specialization: No redundancy directly visible.

After examining, none of the functional dependencies appear to be redundant.

**Irreducible Set of FDs:**

The irreducible set of FDs for the relation with attributes doc\_license, doc\_name, qualification, and specialization is:

doc\_license → doc\_name

doc\_license → qualification

doc\_license → h\_id

qualification → specialization

**Primary Key:**

Based on the irreducible set of FDs, doc\_license can be used as the primary key as it uniquely identifies all other attributes of the relation.

**4. MEDICINE:**

**Attributes:**

m\_id (Medicine ID)

m\_name (Medicine Name)

m\_date (Manufacture Date)

price (Price)

f\_id (Factory ID, foreign key)

expiry\_date (Expiry Date)

**Possible FDs:**

Medicine ID (m\_id) uniquely determines all other attributes:

Assuming that m\_id is a unique identifier for each medicine, it should determine all other attributes.

**m\_id → {m\_name, m\_date, price, f\_id, expiry\_date}**

With Medicine ID(m\_id) and manufacturing date(m\_date), we can determine the expiry date of the medicine

**{m\_id, m\_date} → expiry\_date**

**Irreducible Set of FDs:**

To find the irreducible set of FDs, we need to ensure that each FD is in its simplest form, with minimal left-hand side attributes, and no extraneous dependencies. This involves a few steps: removing redundant attributes from the left-hand side, and ensuring that the FDs are non-redundant.

**The FDs are:**

(a) m\_id → {m\_name, m\_date, price, f\_id, expiry\_date}

(b) **{**m\_id, m\_date} → expiry\_date

**Decomposition:**

m\_id → m\_name

m\_id → m\_date

m\_id → price

m\_id → f\_id

m\_id → expiry\_date

{m\_id, m\_date} → expiry\_date

**Remove Extraneous Attributes:**

In this case, there are no left-hand sides with extraneous attributes since all are minimal.

**Check for Redundant FDs:**

No redundant FDs are present.

**After Removing Redundant FDs:**

m\_id → m\_name

m\_id → m\_date

m\_id → price

m\_id → f\_id

m\_id → expiry\_date

{m\_id, m\_date} → expiry\_date

**Primary Key:**

We can use m\_id as the primary key since it uniquely identifies all the other attributes of the relation.

**5.ORDERS:**

**Attributes:**

o\_id (Order ID)

a\_id (Agent ID, foreign key)

ordered\_by (Customer who placed the order)

ordered\_date (Date when the order was placed)

delv\_date (Delivery date)

total\_price (Total price of the order)

m\_id (Multivalued attribute, Medicine IDs)

**Possible FDs:**

Order ID (o\_id) uniquely determines all other attributes: Assuming that o\_id is a unique identifier for each order, it should determine all other attributes.

**o\_id → {a\_id, ordered\_by, ordered\_date, delv\_date, total\_price, m\_id}**

Delivery date can be derived from the ordered date.

**ordered\_date → delv\_date**

**Agent ID (a\_id) determines total price:** If there are standard prices associated with each agent.

**a\_id → total\_price**

**Irreducible Set of FDs:**

To find the irreducible set of FDs, we need to ensure that each FD is in its simplest form, with minimal left-hand side attributes, and no extraneous dependencies. This involves a few steps: removing redundant attributes from the left-hand side, and ensuring that the FDs are non-redundant.

**The FDs are:**

(a) o\_id → {a\_id, ordered\_by, ordered\_date, delv\_date, total\_price, m\_id}

(b) ordered\_date → delv\_date

(c) a\_id → total\_price

**Decomposition:**

o\_id → a\_id

o\_id → ordered\_by

o\_id → ordered\_date

o\_id → delv\_date

o\_id → total\_price

o\_id → m\_id

(c) ordered\_date → delv\_date

(d) a\_id → total\_price

**Remove Extraneous Attributes:**

In this case, there are no left-hand sides with extraneous attributes since all are minimal.

**Check for Redundant FDs:**

If o\_id → ordered\_date, and ordered\_date → delv\_date, then o\_id → delv\_date

o\_id → delv\_date is redundant.

**After Removing Redundant FDs:**

o\_id → a\_id

o\_id → ordered\_by

o\_id → ordered\_date

o\_id → total\_price

o\_id → m\_id

ordered\_date → delv\_date

a\_id → total\_price

**Primary Key:**

We can use o\_id as the primary key since it uniquely identifies all the other attributes of the relation.

**6. AGENT:**

a\_id (Agent ID)

cwh\_id (Central Warehouse ID, foreign key)

a\_name (Agent Name)

a\_loc (Agent Location)

a\_contact (Agent Contact)

o\_id(Order ID)

**Possible Functional Dependencies (FDs):**

Agent ID (a\_id) uniquely determines all other attributes:

**a\_id → {cwh\_id, a\_name, a\_loc, a\_contact}**

Agent location determines CWH\_id

**a\_loc → cwh\_id**

A particular order goes only to 1 agent.

**o\_id →a\_id**

**Irreducible Set of FDs:**

To find the irreducible set of FDs, we need to ensure that each FD is in its simplest form, with minimal left-hand side attributes, and no extraneous dependencies.

**The initial FDs are:**

(a) a\_id → {cwh\_id, a\_name, a\_loc, a\_contact}

(b) a\_loc → cwh\_id

(c)o\_id →a\_id

**Decomposition:**

(a)a\_id → cwh\_id

a\_id → a\_name

a\_id → a\_loc

a\_id → a\_contact

(b)a\_loc → cwh\_id

(c)o\_id →a\_id

**Removing Extraneous Attributes:**

**Check for redundant FDs:**

If a\_id → a\_loc and a\_loc → cwh\_id, then a\_id → cwh\_id

So ,a\_id → cwh\_id

**After removing redundant FDs:**

a\_id → a\_name

a\_id → a\_loc

a\_id → a\_contact

a\_loc → cwh\_id

o\_id →a\_id

**Primary Key:**

We can use a\_id as the primary key since it uniquely identifies all the other attributes of the relation.

**7. Customer**

**Attributes:**

cus\_id

cus\_name

cus\_contatc

dob

age

cus\_add(multi valued)

**Initial Functional Dependencies (FDs):**

Customer ID (cus\_id) uniquely determines all other attributes:

**cus\_id → {cus\_name, cus\_contact, dob, age, cus\_add}**

Date of birth (dob) determines age:

**dob → age**

Customer address (cus\_add) is multivalued and may involve dependencies within it:

**cus\_add → {street, area, pin}** (this is a multivalued dependency and needs to be handled differently)

**Decomposition of Initial Functional Dependencies (FDs):**

1. Decomposing cus\_id → {cus\_name, cus\_contact, dob, age, cus\_add}:

cus\_id → cus\_name

cus\_id → cus\_contact

cus\_id → dob

cus\_id → age

cus\_id → cus\_add

3. dob → age is already atomic.

4. Decomposing cus\_add → {street, area, pin}

cus\_add → street

cus\_add → area

cus\_add → pin

**Removing Redundant Functional Dependencies:**

Let's check for any redundant dependencies among the decomposed FDs.

cus\_id → cus\_name

cus\_id → cus\_contact

cus\_id → dob

cus\_id → age

cus\_id → cus\_add

dob → age

cus\_add → street

cus\_add → area

cus\_add → pin

**Checking for Redundancies:**

Redundancy between cus\_id → dob and dob → age: If cus\_id → dob and dob → age both hold, then cus\_id → age is redundant.

**After removing redundancies:**

cus\_id → cus\_name

cus\_id → cus\_contact

cus\_id → dob

cus\_id → cus\_add

dob → age

**Irreducible Set of FDs:**

The irreducible set of FDs for the relation with attributes cus\_id, cus\_name, cus\_contact, dob, age, cus\_add is:

cus\_id → cus\_name

cus\_id → cus\_contact

cus\_id → dob

cus\_add → street

cus\_add → area

cus\_add → pin

dob → age

**Handling Multivalued Attributes:**

For the multivalued attribute cus\_add (comprising street, area, pin), we can treat it as a separate relation if necessary to maintain normalization. Here, we consider it as a composite attribute rather than decomposing it further, as the focus is on identifying primary functional dependencies.

**Primary Key:**

Based on the irreducible set of FDs, cus\_id can be used as the primary key as it uniquely determines all other attributes of the relation.

**8. Hospital**

**Attributes:**

h\_id (Hospital ID)

h\_loc (Hospital Location)

d\_id (Distributor ID, foreign key)

h\_contact (Hospital Contact)

h\_name (Hospital Name)

**Possible Functional Dependencies (FDs):**

Hospital ID (h\_id) uniquely determines all other attributes: Assuming that h\_id is a unique identifier for each hospital, it should determine all other attributes.

**h\_id → {h\_loc, d\_id, h\_contact, h\_name}**

Hospital Location (h\_loc) determines the Distributor ID (d\_id):

**h\_loc → d\_id**

**Irreducible Set of FDs:**

To find the irreducible set of FDs, we need to ensure that each FD is in its simplest form, with minimal left-hand side attributes, and no extraneous dependencies. This involves a few steps: removing redundant attributes from the left-hand side and ensuring that the FDs are non-redundant.

**The initial FDs are:**

(a) h\_id → {h\_loc, d\_id, h\_contact, h\_name}

(b) h\_loc → d\_id

**Decomposition:**

(a)h\_id → h\_loc

h\_id → d\_id

h\_id → h\_contact

h\_id → h\_name

(b) h\_loc → d\_id

**Removing Extraneous Attributes:**

**Check for redundant FDs:**

**After removing redundant FDs:**

h\_id → h\_loc

h\_id → h\_contact

h\_id → h\_name

h\_loc → d\_id

**Primary Key:**

We can use h\_id as the primary key since it uniquely identifies all the other attributes of the relation.

**9. E-pharmacy**

**Attributes:**

eph\_id

eph\_name

a\_id

eph\_contact

**Initial Functional Dependencies (FDs):**

E-Pharmacy ID (eph\_id) uniquely determines all other attributes:

**eph\_id → {eph\_name, a\_id, eph\_contact}**

**Decomposition of Initial Functional Dependencies (FDs):**

1. Decomposing eph\_id → {eph\_name, a\_id, eph\_contact}:

eph\_id → eph\_name

eph\_id → a\_id

eph\_id → eph\_contact

**Removing Redundant Functional Dependencies:**

Let's check for any redundant dependencies among the decomposed FDs.

eph\_id → eph\_name

eph\_id → a\_id

eph\_id → eph\_contact

**After removing redundancies:**

eph\_id → eph\_name

eph\_id → a\_id

eph\_id → eph\_contact

**Irreducible Set of FDs:**

The irreducible set of FDs for the relation with attributes eph\_id, eph\_name, a\_id, eph\_contact is:

eph\_id → eph\_name

eph\_id → a\_id

eph\_id → eph\_contact

**Primary Key:**

Based on the irreducible set of FDs, eph\_id can be used as the primary key as it uniquely identifies all other attributes of the relation.

**10. Distributors:**

**Attributes:**

d\_id

d\_loc

o\_id

a\_id

d\_contact

**Initial Functional Dependencies (FDs):**

Distributor ID (d\_id) uniquely determines all other attributes:

**d\_id → {a\_id, d\_loc, d\_contact}**

Order ID uniquely determines the Agent who is handling the order.

**o\_id→a\_id**

**Decomposition of Initial Functional Dependencies (FDs):**

1. Decomposing d\_id → {a\_id, d\_loc, d\_contact}:

d\_id → a\_id

d\_id → d\_loc

d\_id → d\_contact

o\_id→a\_id

**Removing Redundant Functional Dependencies:**

Let's check for any redundant dependencies among the decomposed FDs.

d\_id → a\_id

d\_id → d\_loc

d\_id → d\_contact

o\_id→a\_id

**Checking for Redundancies:**

No redundant FDs present in this table.

**After removing redundancies:**

d\_id → a\_id

d\_id → d\_loc

d\_id → d\_contact

o\_id→a\_id

**Irreducible Set of FDs:**

The irreducible set of FDs for the relation with attributes d\_id, a\_id, d\_loc, o\_id, d\_contact is:

d\_id → a\_id

d\_id → d\_loc

d\_id → d\_contact

o\_id→a\_id

**Primary Key:**

Based on the irreducible set of FDs, d\_id, o\_id can be used as the primary key as it uniquely identifies all other attributes of the relation.

**NORMALIZATION**

**R1. Employee**

**Schema Diagram:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **e\_id** | **e\_name** | **dob** | **e\_contact** | **position** | **salary** | **sales** | **age** | **calls** |

**Attributes**

{e\_id, e\_name, dob, e\_contact, position, salary, sales, age, calls}

**FDs:**

1. e\_id -> e\_name, dob, e\_contact, position, salary, sales, age, calls
2. dob→age
3. position→salary

**Minimal set of fds:**

​(1) e\_id→e\_name

(2) e\_id→dob

(3) e\_id→e\_contact

(4) e\_id→position

(5) e\_id→salary

(6) e\_id→sales

(7) e\_id→calls

(8) dob→age

(9) position→salary

**Candidate Key:**

e\_id

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **e\_id** | **e\_name** | **dob** | **e\_contact** | **position** | **salary** | **sales** | **age** | **calls** |

**Checking for 1NF:**

**Conditions:**

• Domain of the attribute must include only atomic values.

• No multivalued ,composite attributes and nested relations allowed.

Here the conditions are not violated, so relation Employee is in **1NF.**

**Checking for 2NF:**

**Conditions:**

* The relation R should be in 1NF.
* A relation R is in second normal form (2NF) if no proper subset of the candidate key is functionally dependent on a non prime attribute. (no partial dependencies)

There are no partial dependencies in the FDs given above, so the relation Employee is **in 2NF.**

**Checking for 3NF:**

**Conditions**:

* The relation R should be in 2NF.
* no non-prime attribute A in R is transitively dependent on the primary key of R.(NPA -> NPA)

Consider these FDs:

dob→age

position→salary

These 2 FDs are transitively dependent. So the relation Employee is **not 3NF.**

**Decompose:**

**R11**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **e\_id** | **e\_name** | **dob** | **e\_contact** | **position** | **sales** | **calls** |

**R12**

|  |  |
| --- | --- |
| **dob** | **age** |

**R13**

|  |  |
| --- | --- |
| **position** | **salary** |

**R14**

|  |  |  |
| --- | --- | --- |
| **e\_id** | **dob** | **position** |

So, now it's in 3NF.

**Checking for BCNF:**

**Condition:**

* The relation R must be in 3NF.
* LHS of the FD must be a superkey

Relations above satisfy the condition of BCNF.

So, It is in BCNF.

**R2.Medical Company**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **mc\_id** | **mc\_name** | **mc\_loc** | **f\_id** | **f\_loc** | **f\_capacity** | **f\_employees** | **cwh\_id** | **cwh\_loc** | **cwh\_capacity** |

**Attributes:**

mc\_id (Medical Company ID)

mc\_name (Medical Company Name)

mc\_loc (Medical Company Location)

f\_id (Factory ID)

f\_loc (Factory Location)

f\_capacity (Factory Capacity)

f\_employees (Number of Employees in Factory)

cwh\_id (Central Warehouse ID)

cwh\_loc (Central Warehouse Location)

cwh\_capacity (Central Warehouse Capacity)

**Minimal Set of FDS:**

mc\_id → mc\_name

mc\_id → mc\_loc

f\_id → f\_loc

f\_id → f\_capacity

f\_id → f\_employees

cwh\_id → cwh\_loc

cwh\_id → cwh\_capacity

**Candidate Key:**

{mc\_id, f\_id,cwh\_id}

**Checking for 1NF:**

**Conditions:**

• Domain of the attribute must include only atomic values.

• No multivalued ,composite attributes and nested relations allowed.

Here no conditions are violated, so relation Medical Company is in 1NF.

**Checking for 2NF**

**Conditions:**

* The relation R should be in 1NF.
* A relation R is in second normal form (2NF) if no proper subset of the candidate key is functionally dependent on a non prime attribute. (no partial dependencies).

Consider these FDs:

mc\_id → mc\_name,mc\_loc

f\_id → f\_loc, f\_employees,f\_capacity

cwh\_id → cwh\_loc,cwh\_capacity

All these FDs are partially dependent. So the Medical Company is **not in 2NF.**

**Decompose:**

**R21**

|  |  |  |
| --- | --- | --- |
| **mc\_id** | **mc\_name** | **mc\_loc** |

**R22**

|  |  |  |  |
| --- | --- | --- | --- |
| **f\_id** | **f\_name** | **f\_capacity** | **f\_employees** |

**R23**

|  |  |  |
| --- | --- | --- |
| **cwh\_id** | **cwh\_capacity** | **cwh\_loc** |

**R24**

|  |  |  |
| --- | --- | --- |
| **mc\_id** | **f\_id** | **cwh\_id** |

Now, these relations are i**n 2NF**

**Checking for 3NF:**

**Conditions**:

* The relation R should be in 2NF.
* no non-prime attribute A in R is transitively dependent on the primary key of R.(NPA -> NPA)

No FDs are violating these conditions. So relation R21, R22, R23 are **in 3NF.**

**Checking for BCNF:**

**Condition:**

* The relation R must be in 3NF.
* LHS of the FD must be a superkey

All FDs have a superkey as LHS.

So, the above relations are **in BCNF**

**R3. Medicine**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **m\_id** | **m\_name** | **m\_date** | **price** | **f\_id** | **expiry\_date** |

**Attributes:**

m\_id (Medicine ID)

m\_name (Medicine Name)

m\_date (Manufacture Date)

price (Price)

f\_id (Factory ID, foreign key)

expiry\_date (Expiry Date)

**FDs:**

(a) m\_id → {m\_name, m\_date, price, f\_id, expiry\_date}

(b) **{**m\_id, m\_date} → expiry\_date

**Minimal Set of FDS:**

m\_id → m\_name

m\_id → m\_date

m\_id → price

m\_id → f\_id

m\_id → expiry\_date

{m\_id, m\_date} → expiry\_date

**Candidate Key:**

m\_id

**Checking for 1NF:**

**Conditions:**

• Domain of the attribute must include only atomic values.

• No multivalued ,composite attributes and nested relations allowed.

Here no conditions are violated, so relation Medicine is in 1NF.

**Checking for 2NF:**

**Conditions:**

* The relation R should be in 1NF.
* A relation R is in second normal form (2NF) if no proper subset of the candidate key is functionally dependent on a non prime attribute. (no partial dependencies)

There are no partial dependencies in the FDs given above, so the relation Medicine is **in 2NF.**

**Checking for 3NF:**

**Conditions**:

* The relation R should be in 2NF.
* no non-prime attribute A in R is transitively dependent on the primary key of R.(NPA -> NPA)

**Consider these FD:**

{m\_id, m\_date} → expiry\_date

These 2 FDs are transitively dependent. So the relation Medicine is **not 3NF.**

**Decompose:**

**R31**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **m\_id** | **m\_name** | **m\_date** | **price** | **f\_id** |

**R32**

|  |  |  |
| --- | --- | --- |
| **m\_id** | **m\_date** | **expiry\_date** |

**R33**

|  |  |
| --- | --- |
| **m\_id** | **m\_date** |

Now, the above relations are in 3NF.

**Checking for BCNF:**

**Condition:**

* The relation R must be in 3NF.
* LHS of the FD must be a superkey

Relations above satisfy the condition of BCNF.

So, It is **in BCNF.**

**R4. Doctor**

**Schema Diagram:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **doc\_license** | **doc\_name** | **qualification** | **specialization** | **h\_id** |

**Attributes:**

doc\_license (Doctor License)

doc\_name (Doctor Name)

qualification (Educational Qualification)

specialization (Area of Expertise)

h\_id (hospital ID)

**FDs:**

doc\_license → {doc\_name, qualification, specialization,h\_id}

qualification → specialization

**Minimal Set of FDS:**

doc\_license → doc\_name

doc\_license → qualification

doc\_license → h\_id

qualification → specialization

**Candidate Key:**

doc\_license

**Checking for 1NF:**

**Conditions:**

• Domain of the attribute must include only atomic values.

• No multivalued ,composite attributes and nested relations allowed.

Here no conditions are violated, so relation Doctor is in 1NF.

**Checking for 2NF:**

**Conditions:**

* The relation R should be in 1NF.
* A relation R is in second normal form (2NF) if no proper subset of the candidate key is functionally dependent on a non prime attribute. (no partial dependencies)

There are no partial dependencies in the FDs given above, so the relation Doctor is **in 2NF.**

**Checking for 3NF:**

**Conditions**:

* The relation R should be in 2NF.
* no non-prime attribute A in R is transitively dependent on the primary key of R.(NPA -> NPA)

Consider this fd: qualification → specialization

This FD is transitively dependent. So the relation Doctor is **not in 3NF.**

**Decompose:**

**R41**

|  |  |  |  |
| --- | --- | --- | --- |
| **doc\_license** | **doc\_name** | **qualification** | **h\_id** |

**R42**

|  |  |
| --- | --- |
| **qualifiaction** | **specialization** |

**R43**

|  |  |
| --- | --- |
| **qualifiaction** | **doc\_license** |

The above relations are now **in 3NF.**

**Checking for BCNF:**

**Condition:**

* The relation R must be in 3NF.
* LHS of the FD must be a superkey

Relations above satisfy the condition of BCNF.

So, It is **in BCNF.**

**R5. Orders**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **o\_id** | **a\_id** | **ordered\_by** | **ordered\_date** | **delv\_date** | **tot\_price** | **m\_id** |

**Attributes**

o\_id (Order ID)

a\_id (Agent ID, foreign key)

ordered\_by (Customer who placed the order)

ordered\_date (Date when the order was placed)

delv\_date (Delivery date)

total\_price (Total price of the order)

m\_id (Multivalued attribute, Medicine IDs)

**FDs:**

(a) o\_id → {a\_id, ordered\_by, ordered\_date, delv\_date, total\_price, m\_id}

(b) ordered\_date → delv\_date

(c) a\_id → total\_price

**Minimal set of fds:**

​o\_id → a\_id

o\_id → ordered\_by

o\_id → ordered\_date

o\_id → total\_price

o\_id → m\_id

ordered\_date → delv\_date

a\_id → total\_price

**Candidate Key:**

o\_id

**Checking for 1NF:**

**Conditions:**

• Domain of the attribute must include only atomic values.

• No multivalued ,composite attributes and nested relations allowed.

Here m\_id is multi-valued. So Orders is **not in 1NF**

**Decompose:**

**R51**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **o\_id** | **a\_id** | **ordered\_by** | **ordered\_date** | **delv\_date** | **tot\_price** |

**R52**

|  |  |
| --- | --- |
| **o\_id** | **m\_id** |

Now these 2 tables are i**n 1NF**

**Checking for 2NF:**

**Conditions:**

* The relation R should be in 1NF.
* A relation R is in second normal form (2NF) if no proper subset of the candidate key is functionally dependent on a non prime attribute. (no partial dependencies)

There are no partial dependencies in the FDs given above, so the relation Orders is **in 2NF.**

**Checking for 3NF:**

**Conditions**:

* The relation R should be in 2NF.
* no non-prime attribute A in R is transitively dependent on the primary key of R.(NPA -> NPA)

Consider these FDs:

ordered\_date → delv\_date

a\_id → total\_price

These 2 FDs are transitively dependent. So the relation Orders R51 is **not 3NF.**

**Decompose:**

**R511**

|  |  |  |  |
| --- | --- | --- | --- |
| **o\_id** | **a\_id** | **ordered\_by** | **ordered\_date** |

**R512**

|  |  |
| --- | --- |
| **ordered\_date** | **delv\_date** |

**R513**

|  |  |
| --- | --- |
| **a\_id** | **tot\_price** |

**R514**

|  |  |  |
| --- | --- | --- |
| **o\_id** | **a\_id** | **ordered\_date** |

So, now it's in 3NF.

**Checking for BCNF:**

**Condition:**

* The relation R must be in 3NF.
* LHS of the FD must be a superkey

Relations above satisfy the condition of BCNF.

So, It is in BCNF.

**R6. Agent**

**Schema Diagram**

**R6**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a\_id** | **a\_name** | **cwh\_id** | **a\_contact** | **a\_loc** | **o\_id** |

**Attributes:**

a\_id (Agent ID)

cwh\_id (Central Warehouse ID, foreign key)

a\_name (Agent Name)

a\_loc (Agent Location)

a\_contact (Agent Contact)

o\_id(Order ID)

**FDs:**

(a) a\_id → {cwh\_id, a\_name, a\_loc, a\_contact}

(b) a\_loc → cwh\_id

(c)o\_id →a\_id

**Minimal Set of FDs:**

a\_id → a\_name

a\_id → a\_loc

a\_id → a\_contact

a\_loc → cwh\_id

o\_id →a\_id

**Candidate Key:**

a\_id

**Checking for 1NF:**

**Conditions:**

• Domain of the attribute must include only atomic values.

• No multivalued ,composite attributes and nested relations allowed.

Here, cus\_id is a composite attribute. So, the agent relation is in 1NF.

**Checking for 2NF:**

**Conditions:**

* The relation R should be in 1NF.
* A relation R is in second normal form (2NF) if no proper subset of the candidate key is functionally dependent on a non prime attribute. (no partial dependencies)

There are no partial dependencies in the FDs given above, so the relation Hospital is **in 2NF.**

**Checking for 3NF:**

**Conditions**:

* The relation R should be in 2NF.
* no non-prime attribute A in R is transitively dependent on the primary key of R.(NPA -> NPA)

Consider this FD:

a\_loc → cwh\_id

This FD is transitively dependent. So the relation is **not 3NF.**

**Decompose:**

**R61**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **a\_id** | **a\_loc** | **a\_contact** | **o\_id** | **a\_name** |

**R62**

|  |  |
| --- | --- |
| **a\_loc** | **cwh\_id** |

**R63**

|  |  |
| --- | --- |
| **a\_id** | **a\_loc** |

So, now it's **in 3NF.**

**Checking for BCNF:**

**Condition:**

* The relation R must be in 3NF.
* LHS of the FD must be a superkey

For the FD o\_id →a\_id, the condition is violated.

So, R63 is **not in BCNF.**

**R7. Customer:**

**Schema Diagram:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **cus\_id** | **cus\_name** | **dob** | **cus\_contact** | **age** | **cus\_add** |

**Attributes**

cus\_id

cus\_name

cus\_contatc

dob

age

cus\_add(multi valued)

**FDs:**

cus\_id → {cus\_name, cus\_contact, dob, age, cus\_add}

dob → age

cus\_add → {street, area, pin}

**Minimal set of fds:**

cus\_id → cus\_name

cus\_id → cus\_contact

cus\_id → dob

cus\_id → cus\_add

dob → age

**Candidate Key:**

cus\_id

**Checking for 1NF:**

**Conditions:**

• Domain of the attribute must include only atomic values.

• No multivalued ,composite attributes and nested relations allowed.

Here, cus\_id is a composite attribute. So, the customer relation is **not in 1NF.**

Decomposing cus\_add to street, area and pin

**R7**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **cus\_id** | **cus\_name** | **dob** | **cus\_contact** | **age** | **street** | **area** | **pin** |

Now, this relation is **in 1NF.**

**Now, the FDs are:**

cus\_id → cus\_name

cus\_id → cus\_contact

cus\_id → dob

cus\_add → street

cus\_add → area

cus\_add → pin

dob → age

**Checking for 2NF:**

**Conditions:**

* The relation R should be in 1NF.
* A relation R is in second normal form (2NF) if no proper subset of the candidate key is functionally dependent on a non prime attribute. (no partial dependencies)

There are no partial dependencies in the FDs given above, so the relation Customer is **in 2NF.**

**Checking for 3NF:**

**Conditions**:

* The relation R should be in 2NF.
* no non-prime attribute A in R is transitively dependent on the primary key of R.(NPA -> NPA)

Consider these FDs:

cus\_add → street

cus\_add → area

cus\_add → pin

dob → age

These 4 FDs are transitively dependent. So the relation is **not 3NF.**

**Decompose:**

**R71**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **cus\_id** | **cus\_name** | **cus\_contact** | **dob** | **cus\_add** |

**R72**

|  |  |
| --- | --- |
| **dob** | **age** |

**R73**

|  |  |  |  |
| --- | --- | --- | --- |
| **cus\_add** | **street** | **area** | **pin** |

**R74**

|  |  |  |
| --- | --- | --- |
| **cus\_id** | **dob** | **cus\_add** |

So, now it's in 3NF.

**Checking for BCNF:**

**Condition:**

* The relation R must be in 3NF.
* LHS of the FD must be a superkey

Relations above satisfy the condition of BCNF.

So, It is in BCNF.

**R8. Hospital:**

**Schema Diagram:**

**R8**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **cus\_id** | **cus\_name** | **dob** | **cus\_contact** | **age** | **cus\_add** |

**Attributes**

h\_id (Hospital ID)

h\_loc (Hospital Location)

d\_id (Distributor ID, foreign key)

h\_contact (Hospital Contact)

h\_name (Hospital Name)

**FDs:**

h\_id → {h\_loc, d\_id, h\_contact, h\_name}

h\_loc → d\_id

**Minimal set of FDs:**

h\_id → h\_loc

h\_id → h\_contact

h\_id → h\_name

h\_loc → d\_id

**Candidate Key:**

h\_id

**Checking for 1NF:**

**Conditions:**

• Domain of the attribute must include only atomic values.

• No multivalued ,composite attributes and nested relations allowed.

Here, cus\_id is a composite attribute. So, the hospital relation is in 1NF.

**Checking for 2NF:**

**Conditions:**

* The relation R should be in 1NF.
* A relation R is in second normal form (2NF) if no proper subset of the candidate key is functionally dependent on a non prime attribute. (no partial dependencies)

There are no partial dependencies in the FDs given above, so the relation Hospital is **in 2NF.**

**Checking for 3NF:**

**Conditions**:

* The relation R should be in 2NF.
* no non-prime attribute A in R is transitively dependent on the primary key of R.(NPA -> NPA)

Consider these FDs:

h\_loc → d\_id

This FD is transitively dependent. So the relation is **not 3NF.**

**Decompose:**

**R81**

|  |  |  |  |
| --- | --- | --- | --- |
| **h\_id** | **h\_loc** | **h\_contact** | **h\_name** |

**R82**

|  |  |
| --- | --- |
| **h\_loc** | **d\_id** |

**R83**

|  |  |
| --- | --- |
| **h\_id** | **h\_loc** |

So, now it's in 3NF.

**Checking for BCNF:**

**Condition:**

* The relation R must be in 3NF.
* LHS of the FD must be a superkey

Relations above satisfy the condition of BCNF.

So, It is in BCNF.

**R10. Distributor:**

**Schema Diagram:**

**R10**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **d\_id** | **d\_loc** | **o\_id** | **d\_contact** | **a\_id** |

**Attributes**

d\_id(Distributor ID)

d\_loc(Distributor Location)

o\_id (Order ID)

a\_id (Agent ID)

d\_contact( Distributor Contact)

**FDs:**

d\_id → {a\_id, d\_loc, d\_contact}

o\_id→a\_id

**Minimal set of FDs:**

d\_id → a\_id

d\_id → d\_loc

d\_id → d\_contact

o\_id→a\_id

**Candidate Key:**

d\_id, o\_id

**Checking for 1NF:**

**Conditions:**

• Domain of the attribute must include only atomic values.

• No multivalued ,composite attributes and nested relations allowed.

All the conditions are satisfied. So, the Distributor relation **is in 1NF.**

**Checking for 2NF:**

**Conditions:**

* The relation R should be in 1NF.
* A relation R is in second normal form (2NF) if no proper subset of the candidate key is functionally dependent on a non prime attribute. (no partial dependencies)

All the FDs are partially dependent on the given table, so the relation Distributor is **in not 2NF.**

**Decompose:**

**R101**

|  |  |  |  |
| --- | --- | --- | --- |
| **d\_id** | **d\_loc** | **d\_contact** | **a\_id** |

**R102**

|  |  |  |
| --- | --- | --- |
| **o\_id** | **d\_id** | **a\_id** |

**R103**

|  |  |
| --- | --- |
| **d\_id** | **o\_id** |

Now , the relations are **in 2NF**

**Checking for 3NF:**

**Conditions**:

* The relation R should be in 2NF.
* no non-prime attribute A in R is transitively dependent on the primary key of R.(NPA -> NPA)

All relations satisfy the above conditions. So the relation is **3NF.**

**Checking for BCNF:**

**Condition:**

* The relation R must be in 3NF.
* LHS of the FD must be a superkey

Relations above satisfy the condition of BCNF.

So, It is in BCNF.

**Refined ER Diagram:**

**A diagram of a company

Description automatically generated**

**IMPLEMENTATING GUI AND DATABASE CANNECTIVITY   
DATABASE CONNECTIVITY CODE**

**Imports to be done:**

**import java.sql. \*;** Includes classes/interfaces for database operations (e.g., Connection, Statement, PreparedStatement, ResultSet).

**import java.util.logging.\*;** Used for logging messages and exceptions.

**import javax.swing.JOptionPane;** Used for showing dialog boxes to the user.

package com.mycompany.factory;

import java.sql.\*;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.JOptionPane;

**Connectivity:**

**Load JDBC Driver:**

Dynamically loads the Oracle JDBC driver.

**Display Driver Loaded Message:**

Shows a dialog box confirming that the JDBC driver is loaded.

**Establish Database Connection:**

Connects to the Oracle database using the specified URL, username, and password.

**Display Connected Message:**

Shows a dialog box confirming that the connection to the database is successful.

**Exception Handling:**

Logs exceptions if the driver class is not found or if there's an SQL error.

**Source Code:**

public class login extends javax.swing.JFrame {

    /\*\*

     \* Creates new form login

     \*/

    Connection con;

    Statement st;

    PreparedStatement ps;

    ResultSet rs;

    public login() {

        initComponents();

        try

        {

            Class.forName("oracle.jdbc.OracleDriver");

            JOptionPane.showMessageDialog(null,"Driver Loaded");

            try{

                con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL","scott","tiger");

                JOptionPane.showMessageDialog(null, "Connected to Database");

            }

            catch(SQLException ex){

                Logger.getLogger(medicalcompany.class.getName()).log(Level.SEVERE,null,ex);

            }

        }

        catch(ClassNotFoundException ex){

                Logger.getLogger(medicalcompany.class.getName()).log(Level.SEVERE,null,ex);

        }

    }

**Output: (Successfully loaded and connectivity)**

A screenshot of a computer error message

Description automatically generatedA screenshot of a computer error

Description automatically generated

**1.Sign Up / Login Page:**

**Source Code:**

**Sign up:**

 private void sign\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_sign\_bActionPerformed

        String sql="insert into signin(username, pwd) values(?,?)";

        try{

            ps=con.prepareStatement(sql);

            ps.setString(1,user\_t.getText());

            ps.setString(2,pwd\_t.getText());

            ps.executeUpdate();

            JOptionPane.showMessageDialog(null,"Signed Up !");

        }

        catch(SQLException ex){

            Logger.getLogger(medicalcompany.class.getName()).log(Level.SEVERE,null,ex);

            JOptionPane.showMessageDialog(null,"Cant Sign Up!!");

        }

    } //GEN-LAST:event\_sign\_bActionPerformed

**Login:**

private void login\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_login\_bActionPerformed

        String sql="select \* from signin where username='"+user\_t.getText()+"' and pwd='"+pwd\_t.getText()+"'";

        try

        {

            st=con.createStatement();

            rs=st.executeQuery(sql);

            if(rs.next()){

                JOptionPane.showMessageDialog(null, "Login successful!");

                mc login\_b = new mc();

                login\_b.setVisible(true);

            }

            else{

                JOptionPane.showMessageDialog(null, "Login unsuccessful!");

            }

        }

        catch(SQLException ex){

            Logger.getLogger(medicalcompany.class.getName()).log(Level.SEVERE,null,ex);

            JOptionPane.showMessageDialog(null,"Cant Sign Up!!");

        }

    }//GEN-LAST:event\_login\_bActionPerformed

**Output:**

A screenshot of a computer

Description automatically generated

**Sign Up (succesful):**

A screenshot of a login page

Description automatically generatedA screenshot of a computer

Description automatically generated

**Login: (on success):**

A screenshot of a login page

Description automatically generatedA screenshot of a computer

Description automatically generated

On clicking Login Button, if the credentials are the correct, you will be directed to Medical Company details Page.

**(On failure wrong credentials):**

A screenshot of a login page

Description automatically generatedA screenshot of a computer screen

Description automatically generated

**2. Medical Company:**

A screenshot of a computer

Description automatically generated

**Source Code:**

private void insert\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_insert\_bActionPerformed

        String sql="insert into mc(mc\_id,mc\_name,location) values(?,?,?)";

        try{

            ps=con.prepareStatement(sql);

            ps.setString(1,id\_t.getText());

            ps.setString(2,name\_t.getText());

            ps.setString(3,location\_t.getText());

            ps.executeUpdate();

            JOptionPane.showMessageDialog(null,"Inserted !");

        }

        catch(SQLException ex){

            Logger.getLogger(medicalcompany.class.getName()).log(Level.SEVERE,null,ex);

            JOptionPane.showMessageDialog(null,"Cant Insert!!");

        }

    }//GEN-LAST:event\_insert\_bActionPerformed

**Output:**

**Insert: (valid values)**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Insert (invalid values):**

Mc\_id is defined as varchar (5). Entering value more than length of 5 is causing error!

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**Update:**

private void update\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_update\_bActionPerformed

        try{

            String sql="update mc set mc\_name=?,location=? where mc\_id=?";

            ps=con.prepareStatement(sql);

            ps.setString(1,name\_t.getText());

            ps.setString(2,location\_t.getText());

            ps.setString(3,id\_t.getText());

            int rowsUpdated=ps.executeUpdate();

            if(rowsUpdated>0){

                JOptionPane.showMessageDialog(null,"Updated successfully !");

            }

            else{

                JOptionPane.showMessageDialog(null,"No record found with the given ID");

            }

        }

        catch(SQLException ex){

            Logger.getLogger(medicalcompany.class.getName()).log(Level.SEVERE,null,ex);

            JOptionPane.showMessageDialog(null,"Cant update!!");

        }

    }//GEN-LAST:event\_update\_bActionPerformed

**Output: (valid)**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Invalid update (mc\_id doesn’t exists)**

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**Delete:**

 private void delete\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_delete\_bActionPerformed

        String sql = "DELETE FROM mc WHERE mc\_id = ?";

        try {

            ps = con.prepareStatement(sql);

            ps.setString(1, id\_t.getText());

            int rowsDeleted = ps.executeUpdate();

            if (rowsDeleted > 0) {

                JOptionPane.showMessageDialog(null, "Deleted successfully!");

            } else {

                JOptionPane.showMessageDialog(null, "No record found with the given ID.");

            }

        } catch (SQLException ex) {

            Logger.getLogger(medicalcompany.class.getName()).log(Level.SEVERE, null, ex);

            JOptionPane.showMessageDialog(null, "Error occurred while deleting!");

        }

    }//GEN-LAST:event\_delete\_bActionPerformed

**Output(valid):**

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**Invalid (mc\_id doesn’t exixts)**

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**Search:**

private void search\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_search\_bActionPerformed

        String sql = "SELECT \* FROM mc WHERE mc\_id = ?";

        try {

            ps = con.prepareStatement(sql);

            ps.setString(1, id\_t.getText());

            rs = ps.executeQuery();

            if (rs.next()) {

                name\_t.setText(rs.getString("mc\_name"));

                location\_t.setText(rs.getString("location"));

                JOptionPane.showMessageDialog(null, "Record Found");

            } else {

                JOptionPane.showMessageDialog(null, "No record found with the given ID");

            }

        } catch (SQLException ex) {

            Logger.getLogger(medicalcompany.class.getName()).log(Level.SEVERE, null, ex);

            JOptionPane.showMessageDialog(this, "Error occurred while searching!");

        }

    }//GEN-LAST:event\_search\_bActionPerformed

**OUTPUT:(valid search)**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Result:**

A screenshot of a computer

Description automatically generated

On clicking “Add Factory Details here” button, you will be directed to Factory details page.

**3 . Factory**

A computer screen shot of a factory

Description automatically generated

**Insert:**

    private void insert\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_insert\_bActionPerformed

        String sql="insert into fac(f\_id,f\_name,f\_capacity,f\_emp,mc\_id) values(?,?,?,?,?)";

        try{

            ps=con.prepareStatement(sql);

            ps.setString(1, id\_t.getText());

            ps.setString(2, name\_t.getText());

            ps.setInt(3, Integer.parseInt(capacity\_t.getText()));

            ps.setInt(4, Integer.parseInt(emp\_t.getText()));

            ps.setString(5, mc\_id\_t.getText());

            ps.executeUpdate();

            JOptionPane.showMessageDialog(null,"Inserted!");

        }

        catch(SQLException ex){

                JOptionPane.showMessageDialog(null,"Can't Insert!");

                Logger.getLogger(fac.class.getName()).log(Level.SEVERE,null,ex);

            }

    }//GEN-LAST:event\_insert\_bActionPerformed

**Output: (valid)**

A screenshot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

**Invalid: (factory ID aldready exists)**

A screenshot of a computer

Description automatically generatedA computer screen shot of a message

Description automatically generated

**Update:**

    private void update\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_update\_bActionPerformed

        String sql="update fac set f\_name=?, f\_capacity=?, f\_emp=?, mc\_id=? WHERE f\_id=?";

        try{

            ps=con.prepareStatement(sql);

            ps.setString(1, name\_t.getText());

            ps.setInt(2, Integer.parseInt(capacity\_t.getText()));

            ps.setInt(3, Integer.parseInt(emp\_t.getText()));

            ps.setString(4, mc\_id\_t.getText());

            ps.setString(5, id\_t.getText());

            int rowsUpdated=ps.executeUpdate();

            if(rowsUpdated>0){

                JOptionPane.showMessageDialog(null, "Updated successfully!");

            }

            else{

                JOptionPane.showMessageDialog(null, "No record found with the given ID.");

            }

        }

        catch(SQLException ex){

            Logger.getLogger(fac.class.getName()).log(Level.SEVERE, null, ex);

            JOptionPane.showMessageDialog(this, "Can't Update!");

        }

    }//GEN-LAST:event\_update\_bActionPerformed

**Output: (valid)**

A computer screen shot of a factory

Description automatically generatedA screenshot of a computer error

Description automatically generated

**Invalid: (factory id doesn’t exist):**

A computer screen shot of a factory details

Description automatically generatedA screen shot of a computer

Description automatically generated

**Delete:**

 private void delete\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-

        String sql = "delete fac where f\_id=?";

        try{

            ps=con.prepareStatement(sql);

            ps.setString(1, id\_t.getText());

            int rowsDeleted=ps.executeUpdate();

            if(rowsDeleted>0){

                JOptionPane.showMessageDialog(null, "Deleted successfully!");

            }

            else{

                JOptionPane.showMessageDialog(null, "No record found with the given ID.");

            }

        }

        catch(SQLException ex){

            Logger.getLogger(fac.class.getName()).log(Level.SEVERE, null, ex);

            JOptionPane.showMessageDialog(this, "Can't Delete!");

        }

    }//GEN-LAST:event\_delete\_bActionPerformed

**Output: (valid)**

A computer screen shot of a factory details

Description automatically generatedA screenshot of a message

Description automatically generated

**Invalid: (f\_id doesn’t exixts)**

A computer screen shot of a factory

Description automatically generatedA screenshot of a computer

Description automatically generated

**Search:**

private void search\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_search\_bActionPerformed

        String sql="select \* from fac where f\_id = ?";

        try{

            ps=con.prepareStatement(sql);

            ps.setString(1, id\_t.getText());

            rs=ps.executeQuery();

            if(rs.next()){

                name\_t.setText(rs.getString("f\_name"));

                capacity\_t.setText(rs.getString("f\_capacity"));

                emp\_t.setText(rs.getString("f\_emp"));

                mc\_id\_t.setText(rs.getString("mc\_id"));

                JOptionPane.showMessageDialog(null, "Record Found");

            }

            else{

                JOptionPane.showMessageDialog(null, "No record found with the given ID");

            }

        }

        catch (SQLException ex) {

            Logger.getLogger(fac.class.getName()).log(Level.SEVERE, null, ex);

            JOptionPane.showMessageDialog(this, ex.getMessage());

        }

    }//GEN-LAST:event\_search\_bActionPerformed

**Output:**

A screenshot of a computer

Description automatically generated

**Result:**

A computer screen shot of a factory

Description automatically generated

On clicking “Add CWH Details here” button, you will be directed to Central WareHouse details page.

**4 . Central WareHouse**

A screenshot of a computer

Description automatically generated

**Insert:**

    private void insert\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_insert\_bActionPerformed

        String sql="insert into cwh(cwh\_id,cwh\_loaction,cwh\_capacity,f\_id) values(?,?,?,?)";

        try{

            ps=con.prepareStatement(sql);

            ps.setString(1, id\_t.getText());

            ps.setString(2, loc\_t.getText());

            ps.setInt(3, Integer.parseInt(cap\_t.getText()));

            ps.setString(4, f\_id\_t.getText());

            ps.executeUpdate();

            JOptionPane.showMessageDialog(null,"Inserted!");

        }

        catch(SQLException ex){

                JOptionPane.showMessageDialog(null,"Can't Insert!");

                Logger.getLogger(fac.class.getName()).log(Level.SEVERE,null,ex);

            }

    }//GEN-LAST:event\_insert\_bActionPerformed

**Output: (valid)**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Invalid(invalid factory id):**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Update:**

 private void update\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_update\_bActionPerformed

       String sql="update cwh set cwh\_loaction=?, cwh\_capacity=?, f\_id=? WHERE cwh\_id=?";

        try{

            ps=con.prepareStatement(sql);

            ps.setString(1, loc\_t.getText());

            ps.setInt(2, Integer.parseInt(cap\_t.getText()));

            ps.setString(3, f\_id\_t.getText());

            ps.setString(4, id\_t.getText());

            int rowsUpdated=ps.executeUpdate();

            if(rowsUpdated>0){

                JOptionPane.showMessageDialog(null, "Updated successfully!");

            }

            else{

                JOptionPane.showMessageDialog(null, "No record found with the given ID.");

            }

        }

        catch(SQLException ex){

            Logger.getLogger(fac.class.getName()).log(Level.SEVERE, null, ex);

            JOptionPane.showMessageDialog(this, "Can't Update!");

        }

    }//GEN-LAST:event\_update\_bActionPerformed

**Output: (valid)**

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**Invalid: (cwh\_id doesn’t exists)**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Delete:**

private void delete\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_delete\_bActionPerformed

        String sql = "delete cwh where cwh\_id=?";

        try{

            ps=con.prepareStatement(sql);

            ps.setString(1, id\_t.getText());

            int rowsDeleted=ps.executeUpdate();

            if(rowsDeleted>0){

                JOptionPane.showMessageDialog(null, "Deleted successfully!");

            }

            else{

                JOptionPane.showMessageDialog(null, "No record found with the given ID.");

            }

        }

        catch(SQLException ex){

            Logger.getLogger(fac.class.getName()).log(Level.SEVERE, null, ex);

            JOptionPane.showMessageDialog(this, "Can't Delete!");

        }

    }//GEN-LAST:event\_delete\_bActionPerformed

**Output: (valid)**

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**Invalid: (cwh\_id doesn’t exists)**

A screenshot of a computer

Description automatically generatedA screenshot of a chat

Description automatically generated

**Search:**

    private void search\_bActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_search\_bActionPerformed

        String sql="select \* from cwh where cwh\_id = ?";

        try{

            ps=con.prepareStatement(sql);

            ps.setString(1, id\_t.getText());

            rs=ps.executeQuery();

            if(rs.next()){

                loc\_t.setText(rs.getString("cwh\_loaction"));

                cap\_t.setText(rs.getString("cwh\_capacity"));

                f\_id\_t.setText(rs.getString("f\_id"));

                JOptionPane.showMessageDialog(null, "Record Found");

            }

            else{

                JOptionPane.showMessageDialog(null, "No record found with the given ID");

            }

        }

        catch (SQLException ex) {

            Logger.getLogger(fac.class.getName()).log(Level.SEVERE, null, ex);

            JOptionPane.showMessageDialog(this, ex.getMessage());

        }

    }//GEN-LAST:event\_search\_bActionPerformed

**Output: (valid)**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Result:**

A screenshot of a computer

Description automatically generated