**The University Of Azad Jammu & Kashmir,**

**Muzaffarabad**

**Department of Software Engineering**

**LAB TASK 13**

**Database Systems**

**Course Code**: **CS-2204**

**Submitted To:**

Engr. Tahir Jahangir

**Submitted By:**

Ahmed Ali

**Roll No:**

2023-SE-41

**Session**: 2023-2027

Contents

[Lab Task – 13: Database Normalization 4](#_Toc207376310)

[🎯 Objective: 4](#_Toc207376311)

[📘 Task 01: Analyze Unnormalized Data 4](#_Toc207376312)

[Instructions: 4](#_Toc207376313)

[Unnormalized Table: 4](#_Toc207376314)

[**Analysis** 4](#_Toc207376315)

[📘 Task 02: Normalize the Table to 1NF 5](#_Toc207376316)

[Instructions: 5](#_Toc207376317)

[**Conversion to 1NF** 5](#_Toc207376318)

[**Table in 1NF** 5](#_Toc207376319)

[**Explanation of Changes** 6](#_Toc207376320)

[📘 Task 03: Convert to 2NF 6](#_Toc207376321)

[Instructions: 6](#_Toc207376322)

[Step 1: Identify Dependencies 6](#_Toc207376323)

[Step 2: Convert to 2NF 7](#_Toc207376324)

[**Table 1: Orders** 7](#_Toc207376325)

[**Table 2: OrderDetails** 7](#_Toc207376326)

[Step 3: Explanation 7](#_Toc207376327)

[📘 Task 04: Convert to 3NF 7](#_Toc207376328)

[Instructions: 8](#_Toc207376329)

[Step 1: Identify Transitive Dependencies 8](#_Toc207376330)

[Step 2: Convert to 3NF 8](#_Toc207376331)

[**Customers** 8](#_Toc207376332)

[**Orders** 8](#_Toc207376333)

[**Products** 9](#_Toc207376334)

[**OrderDetails** 9](#_Toc207376335)

[Step 3: Justification 9](#_Toc207376336)

[Relationships (ER-style sketch in words) 9](#_Toc207376337)

[📘 Task 05: Scenario-Based Normalization Challenge 10](#_Toc207376338)

[Scenario: 10](#_Toc207376339)

[Instructions: 11](#_Toc207376340)

[Step 1: Convert to 1NF (First Normal Form) 11](#_Toc207376341)

[Step 2: Convert to 2NF (Second Normal Form) 11](#_Toc207376342)

[Step 3: Convert to 3NF (Third Normal Form) 12](#_Toc207376343)

[Final Summary 13](#_Toc207376344)

[**Summary of Learning** 14](#_Toc207376345)

# Lab Task – 13: Database Normalization

# 🎯 Objective:

In this lab, students will apply normalization techniques (1NF, 2NF, 3NF) to analyze, restructure, and improve unnormalized data. The goal is to reduce redundancy, identify functional dependencies, and organize tables logically. Tasks involve real-world scenarios, requiring students to document each normalization step clearly and explain their design decisions.

# 📘 Task 01: Analyze Unnormalized Data

🎯 **Objective**: Understand how unnormalized data causes anomalies and identify problems.

## Instructions:

1. Given the unnormalized table below, identify and highlight:

o Repeating groups o Multivalued fields o Anomalies (insertion, update, deletion)

## Unnormalized Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerName** | **Products Ordered** | **Quantities** | **Address** |
| O001 | Adeel Khan | Mouse, Keyboard | 1, 2 | G-10 Islamabad |
| O002 | Sana Tariq | Monitor | 1 | F-8 Islamabad |
| O003 | Adeel Khan | Mouse | 3 | G-10 Islamabad |

📝 Write your analysis in a short paragraph, and tabulate the problems found.

### **Analysis:**

The given table is **unnormalized** because it contains repeating groups and multivalued fields. For example, in OrderID O001, the products Mouse and Keyboard are stored together in a single cell, and their corresponding quantities 1, 2 are also grouped, creating a repeating group. This structure makes it difficult to maintain data consistency. If we want to update Adeel Khan’s address, it must be changed in multiple rows (update anomaly). Adding a new product for a customer without an order would not be possible (insertion anomaly). Similarly, deleting an order might remove customer information entirely (deletion anomaly). These problems arise due to the lack of normalization.

| **Problem Type** | **Example from Table** | **Issue Caused** |
| --- | --- | --- |
| **Repeating Groups** | Products Ordered = *Mouse, Keyboard*; Quantities = *1, 2* | Multiple values stored in one field. |
| **Multivalued Fields** | Both Products Ordered and Quantities contain more than one value in a single cell. | Breaks atomicity rule. |
| **Insertion Anomaly** | Cannot insert a new customer who hasn’t ordered yet. | Loss of customer info until order exists. |
| **Update Anomaly** | Updating *Adeel Khan’s address* requires multiple row changes. | Inconsistency risk. |
| **Deletion Anomaly** | If Order O002 (Sana Tariq) is deleted, her customer info is also lost. | Unwanted data loss. |

# 📘 Task 02: Normalize the Table to 1NF

🎯 **Objective**: Convert the unnormalized table to First Normal Form (1NF).

## Instructions:

1. Remove multivalued fields and repeating groups.
2. Create a new structure where each field contains atomic (single) values.
3. Represent the new table and explain your changes.

### **Conversion to 1NF:**

**Rules of 1NF:**

1. Each column must have **atomic (single) values**.
2. Remove **repeating groups / multivalued fields**.
3. Each row should represent a **unique record**.

To achieve this, I will split the multivalued Products Ordered and Quantities into separate rows.

### **Table in 1NF**

| **OrderID** | **CustomerName** | **Product** | **Quantity** | **Address** |
| --- | --- | --- | --- | --- |
| O001 | Adeel Khan | Mouse | 1 | G-10 Islamabad |
| O001 | Adeel Khan | Keyboard | 2 | G-10 Islamabad |
| O002 | Sana Tariq | Monitor | 1 | F-8 Islamabad |
| O003 | Adeel Khan | Mouse | 3 | G-10 Islamabad |

### **Explanation of Changes**

* **Removed multivalued fields:** Instead of storing Mouse, Keyboard in one column, each product is now in a separate row.
* **Separated repeating groups:** Quantities were aligned with each product.
* **Ensured atomicity:** Now every cell has a single value (no commas inside fields).
* **Preserved information:** All details from the original table are still available but structured properly.

# 📘 Task 03: Convert to 2NF

🎯 **Objective**: Eliminate partial dependencies from the 1NF table.

## Instructions:

1. Identify the composite primary key (if any).
2. Remove attributes that do not fully depend on the composite key.
3. Create separate tables for entities and link them logically.

Submit:

* List of identified dependencies
* Final 2NF tables (at least two or more)
* Explanation of how partial dependencies were removed

## Step 1: Identify Dependencies:

* **Composite Primary Key in 1NF:** (OrderID, Product)  
  → Because one order can have multiple products.
* **Dependencies Found:**
  1. Quantity → depends on **(OrderID, Product)** (valid, no issue).
  2. CustomerName → depends only on OrderID, **not** on Product (partial dependency).
  3. Address → depends only on OrderID (or really the customer), **not** on Product (partial dependency).

Thus, CustomerName and Address are **partially dependent** attributes → must be moved.

## Step 2: Convert to 2NF

### **Table 1: Orders**

(Stores order-level information)

| **OrderID** | **CustomerName** | **Address** |
| --- | --- | --- |
| O001 | Adeel Khan | G-10 Islamabad |
| O002 | Sana Tariq | F-8 Islamabad |
| O003 | Adeel Khan | G-10 Islamabad |

### **Table 2: OrderDetails**

(Stores product-specific information per order)

| **OrderID** | **Product** | **Quantity** |
| --- | --- | --- |
| O001 | Mouse | 1 |
| O001 | Keyboard | 2 |
| O002 | Monitor | 1 |
| O003 | Mouse | 3 |

## Step 3: Explanation:

* **Removed Partial Dependencies:**
  + CustomerName and Address were moved into the **Orders** table since they depend only on OrderID.
  + Quantity correctly stays in **OrderDetails**, since it depends on the combination (OrderID, Product).
* **Composite Key Issue Fixed:** Now OrderDetails has a composite primary key (OrderID, Product) with no partial dependencies.
* The design is now in **2NF**.

# 📘 Task 04: Convert to 3NF

🎯 **Objective**: Eliminate transitive dependencies from the 2NF tables.

## Instructions:

1. Inspect each table for non-key attributes depending on other non-key attributes.
2. Create new tables where required and assign appropriate primary and foreign keys.
3. Clearly mark relationships between tables.

✅ Submit:

* Final 3NF table structures
* Justification for each new table
* ER-style relationship sketch (optional but recommended)

## Step 1: Identify Transitive Dependencies:

* In **Orders table**:
  + CustomerName → Address (address depends on customer, not directly on order).
  + This is a **transitive dependency**, so we must separate Customers into its own table.
* In **OrderDetails table**:
  + Product might have more attributes (like price, category, etc.) that don’t depend on order.
  + To keep the structure clean, we should also separate **Products** into their own table.

## Step 2: Convert to 3NF

### **Customers**

(Each customer is stored once with their details)

| **CustomerID** | **CustomerName** | **Address** |
| --- | --- | --- |
| C001 | Adeel Khan | G-10 Islamabad |
| C002 | Sana Tariq | F-8 Islamabad |

### **Orders**

(Each order belongs to a customer)

| **OrderID** | **CustomerID** |
| --- | --- |
| O001 | C001 |
| O002 | C002 |
| O003 | C001 |

### **Products**

(Independent list of products)

| **ProductID** | **ProductName** |
| --- | --- |
| P001 | Mouse |
| P002 | Keyboard |
| P003 | Monitor |

### **OrderDetails**

(Bridge between orders and products, with quantities)

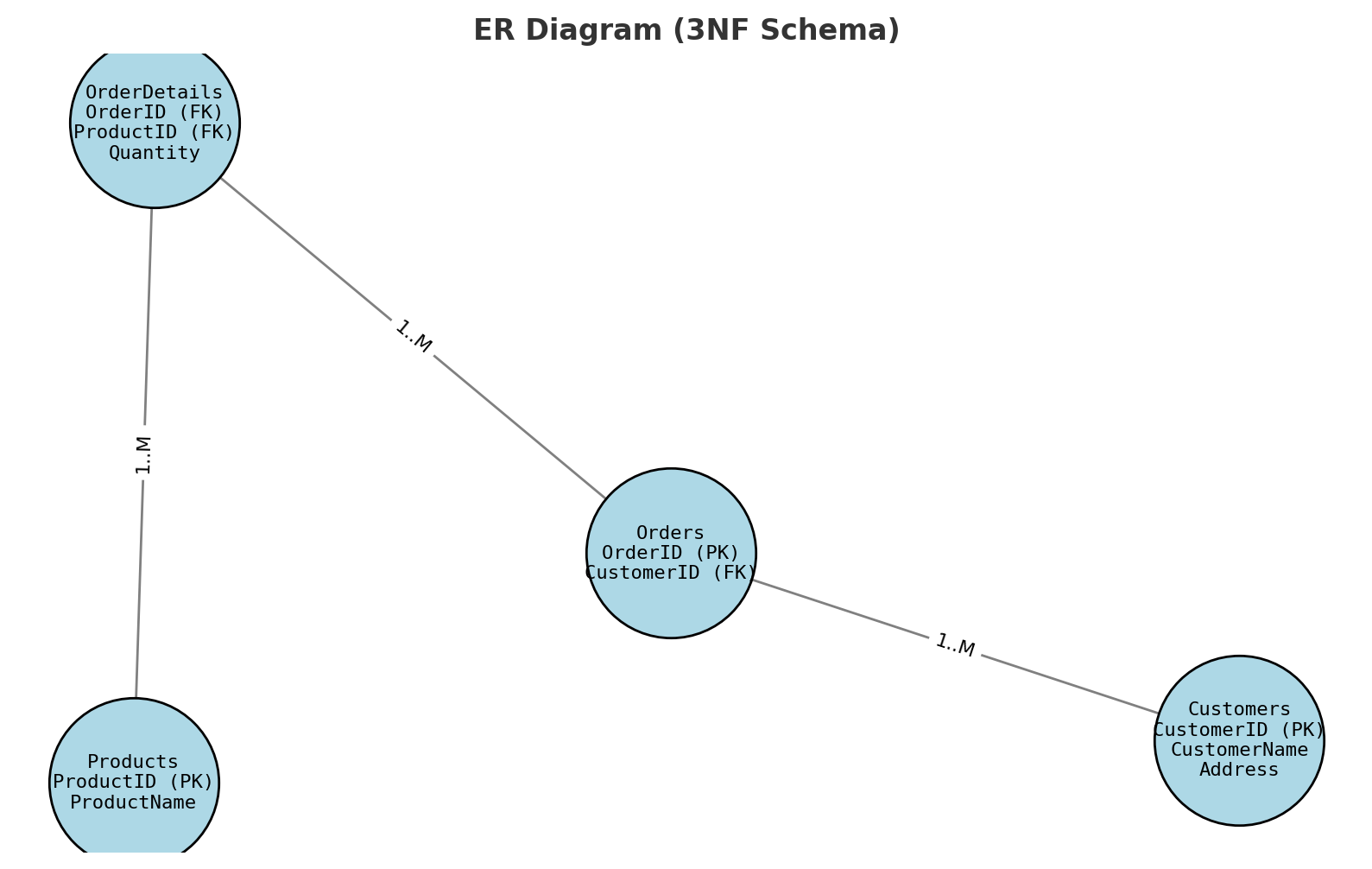
| **OrderID** | **ProductID** | **Quantity** |
| --- | --- | --- |
| O001 | P001 | 1 |
| O001 | P002 | 2 |
| O002 | P003 | 1 |
| O003 | P001 | 3 |

## Step 3: Justification

1. **Customers Table** created to eliminate transitive dependency (CustomerName → Address).
2. **Products Table** created to ensure product details don’t repeat inside OrderDetails.
3. **Orders Table** now only contains OrderID and CustomerID (foreign key).
4. **OrderDetails Table** connects orders and products with a composite key (OrderID, ProductID).

## Relationships (ER-style sketch in words)

* **Customer (1) → (M) Orders** (One customer can place many orders).
* **Order (1) → (M) OrderDetails** (One order can have many product entries).
* **Product (1) → (M) OrderDetails** (One product can appear in many orders).



Now the schema is fully in **3NF**:

* No repeating groups
* No partial dependencies
* No transitive dependencies

# 📘 Task 05: Scenario-Based Normalization Challenge

🎯 **Objective**: Apply normalization techniques in a real-world case study.

## Scenario:

You are given the following flat structure of a hospital system:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PatientID** | **PatientName** | **DoctorName** | **Department** | **VisitDate** | **Fee** |
| P001 | Uzair Ahmed | Dr. Sohail | Cardiology | 2024-01-11 | 2000 |
| P002 | Sana Zafar | Dr. Tania | Neurology | 2024-01-13 | 2500 |
| P001 | Uzair Ahmed | Dr. Sohail | Cardiology | 2024-02-05 | 2000 |

## Instructions:

1. Normalize the table step-by-step up to 3NF.
2. At each step (1NF → 2NF → 3NF), clearly:

o Identify keys o Explain dependencies o Show the new table structure

## Step 1: Convert to ****1NF (First Normal Form):****

**Rule:** Remove repeating groups and make sure all values are atomic (single values, no lists).

Our table is already atomic, so just recognize the **primary key**:

A visit is identified by **(PatientID, DoctorName, VisitDate)**.

**1NF Table**

| **PatientID** | **PatientName** | **DoctorName** | **Department** | **VisitDate** | **Fee** |
| --- | --- | --- | --- | --- | --- |
| P001 | Uzair Ahmed | Dr. Sohail | Cardiology | 2024-01-11 | 2000 |
| P002 | Sana Zafar | Dr. Tania | Neurology | 2024-01-13 | 2500 |
| P001 | Uzair Ahmed | Dr. Sohail | Cardiology | 2024-02-05 | 2000 |

Now it’s **1NF**.

## Step 2: Convert to ****2NF (Second Normal Form):****

**Rule:** Remove **partial dependencies** (when some attributes depend only on part of the composite key).

Problems:

* PatientName depends only on **PatientID**.
* Department and Fee depend only on **DoctorName**.
* They don’t depend on the whole key (PatientID, DoctorName, VisitDate).

Solution: Split into 3 tables.

**Patients Table**

| **PatientID** | **PatientName** |
| --- | --- |
| P001 | Uzair Ahmed |
| P002 | Sana Zafar |

**Doctors Table**

| **DoctorName** | **Department** | **Fee** |
| --- | --- | --- |
| Dr. Sohail | Cardiology | 2000 |
| Dr. Tania | Neurology | 2500 |

**Visits Table**

| **PatientID** | **DoctorName** | **VisitDate** |
| --- | --- | --- |
| P001 | Dr. Sohail | 2024-01-11 |
| P002 | Dr. Tania | 2024-01-13 |
| P001 | Dr. Sohail | 2024-02-05 |

Now it’s **2NF**.

## Step 3: Convert to ****3NF (Third Normal Form):****

**Rule:** Remove **transitive dependencies** (non-key → non-key).

Problem:

* In Doctors table: DoctorName → Department.
* If Department info changes, we must update in multiple rows.

Solution: Create a **Departments table** and use IDs.

**Departments Table**

| **DepartmentID** | **DepartmentName** |
| --- | --- |
| D01 | Cardiology |
| D02 | Neurology |

**Doctors Table**

| **DoctorID** | **DoctorName** | **DepartmentID** | **Fee** |
| --- | --- | --- | --- |
| D101 | Dr. Sohail | D01 | 2000 |
| D102 | Dr. Tania | D02 | 2500 |

**Patients Table**

| **PatientID** | **PatientName** |
| --- | --- |
| P001 | Uzair Ahmed |
| P002 | Sana Zafar |

**Visits Table**

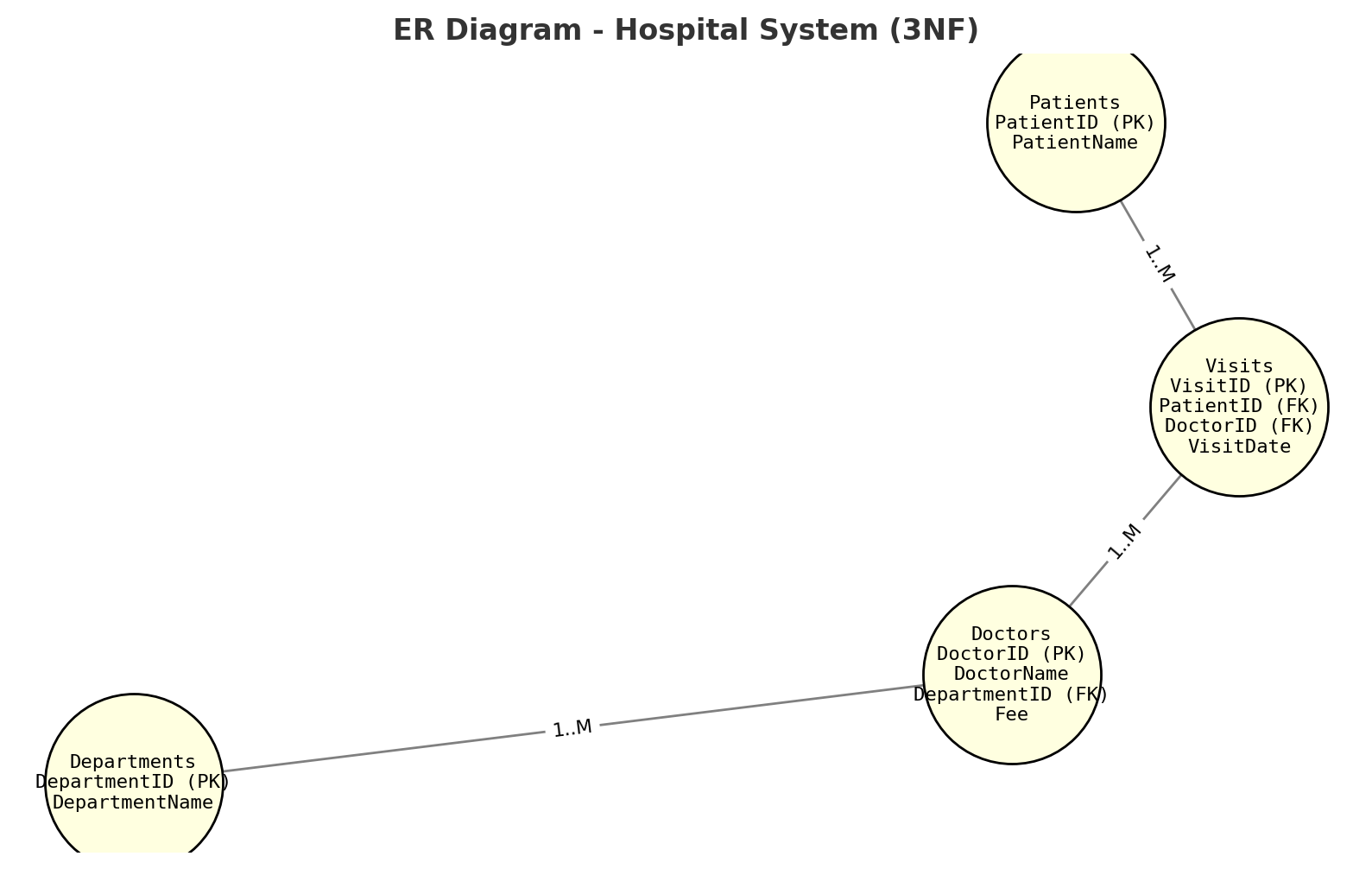
| **VisitID** | **PatientID** | **DoctorID** | **VisitDate** |
| --- | --- | --- | --- |
| V001 | P001 | D101 | 2024-01-11 |
| V002 | P002 | D102 | 2024-01-13 |
| V003 | P001 | D101 | 2024-02-05 |

Now it’s **3NF**:

* No repeating data.
* No partial dependency.
* No transitive dependency.
* All data is stored once in the right place.

# Final Summary

* **1NF:** Original table, atomic values, composite key = (PatientID, DoctorName, VisitDate).
* **2NF:** Split into Patients, Doctors, and Visits. Removed partial dependencies.
* **3NF:** Added Departments table. Removed transitive dependency (Doctor → Department).



### **Summary of Learning**

Through this exercise, I learned how **normalization** removes redundancy and prevents anomalies in databases. In **1NF**, I ensured that all values are atomic and identified the composite primary key. In **2NF**, I removed partial dependencies by separating data into Patients, Doctors, and Visits. Finally, in **3NF**, I eliminated transitive dependencies by creating a Departments table and linking it with Doctors. This process taught me how normalization improves **data integrity, reduces duplication, and ensures consistency** in real-world database design.