1. Consider the relation Treatment with the schema Treatment (doctorID, doctorName, patientID, diagnosis) and functional dependencies; doctorID → doctorName and (doctorID, patientID) → diagnosis. Describe different types of anomaly that can arise for this table with example records.

In a relational schema like the one described for the **Treatment** table:

Treatment(doctorID, doctorName, patientID, diagnosis)\text{Treatment(doctorID, doctorName, patientID, diagnosis)}Treatment(doctorID, doctorName, patientID, diagnosis)

with the following **functional dependencies**:

- doctorID→doctorName\text{doctorID} \to \text{doctorName}doctorID→doctorName
- (doctorID,patientID)→diagnosis(\text{doctorID}, \text{patientID}) \to \text{diagnosis}(doctorID,patientID)→diagnosis

several types of **anomalies** can arise if the table is not properly normalized. These anomalies include **update anomalies**, **insert anomalies**, and **delete anomalies**. Let's explore each with examples based on this schema:

1. Update Anomaly

An update anomaly occurs when the same data has to be updated in multiple rows, leading to data inconsistency if not all instances are updated correctly.

Example:

Consider the following records in the **Treatment** table:

doctorID	doctorName	patientID	diagnosis
101	Dr. Smith	1001	Flu
101	Dr. Smith	1002	Cold
101	Dr. Smith	1003	Headache

Here, the **doctorName** ("Dr. Smith") is repeated for every patient. If Dr. Smith changes their name (e.g., to "Dr. John Smith"), this change needs to be made in multiple rows. If the update is not applied consistently across all records, there could be inconsistency, where some rows have "Dr. Smith" and others have "Dr. John Smith".

2. Insert Anomaly

An insert anomaly occurs when certain data cannot be added to the table without having other data that may not yet be available or applicable.

Example:

In the **Treatment** table, suppose a new doctor joins the hospital, and you want to add their details to the table. However, since there is no patient yet assigned to the doctor, you don't have a valid **patientID** or **diagnosis** to enter. Due to the table's design, you can't insert a row for the new doctor without also including a patient, which creates an insert anomaly.

For example, you cannot insert:

doctorID	doctorName	patientID	diagnosis
102	Dr. Johnson	NULL	NULL

This row may not be allowed because **patientID** and **diagnosis** fields might be required.

3. Delete Anomaly

A delete anomaly occurs when deleting a record unintentionally removes other useful information.

Example:

Consider the following records:

doctorID	doctorName	patientID	diagnosis
101	Dr. Smith	1001	Flu
101	Dr. Smith	1002	Cold
101	Dr. Smith	1003	Headache

Suppose the last patient of Dr. Smith (patientID = 1003) is discharged, and you want to delete their treatment record. Deleting the row with **patientID = 1003** would remove the association between **doctorID = 101** and **doctorName = Dr. Smith** entirely if there are no other patients linked to Dr. Smith. This would cause a loss of information about Dr. Smith even though they are still employed and available for new patients.

4. Redundancy (Duplication)

Redundancy is when the same piece of information is stored multiple times in different rows, leading to increased storage requirements and potential inconsistencies.

Example:

In the following records, the **doctorName** is repeated for every patient treated by the same doctor:

doctorID	doctorName	patientID	diagnosis
101	Dr. Smith	1001	Flu
101	Dr. Smith	1002	Cold
101	Dr. Smith	1003	Headache

The **doctorName** ("Dr. Smith") is duplicated multiple times, leading to redundancy. If **doctorName** needs to be updated, it would have to be done in every row where **doctorID = 101**. This increases the risk of inconsistencies (update anomaly) and unnecessary duplication of data.

2. Normalize the below table User_Personal upto 3NF

UserID	email	Fname	Lname	City	State	Zip
A12	Mani@ymail.com	MANISH	JAIN	BILASPUR	CHATISGARH	458991
PO45	Pooja.g@gmail.co	POOJA	MAGG	KACCH	GUJRAT	832212
LA33	Lavle98@jj.com	LAVLEEN	DHALLA	RAIPUR	CHATISGARH	853578
CH99	Cheki9j@ih.com	CHIMAL	BEDI	TRICHY	TAMIL NADU	632011
DA74	Danu58@g.com	DANY	JAMES	TRICHY	TAMIL NADU	645018

To normalize the given **User_Personal** table up to **3NF** (Third Normal Form), we'll go through the following normalization steps: **1NF (First Normal Form)**, **2NF (Second Normal Form)**, and **3NF (Third Normal Form)**.

Step 1: Original Table (Unnormalized)

The original table structure looks like this:

UserID	U_email	Fname	Lname	City	State	Zip
A12	Mani@ymail.com	MANISH	JAIN	BILASPUR	CHATISGARH	458991
PO45	Pooja.g@gmail.co	POOJA	MAGG	KACCH	GUJRAT	832212
LA33	Lavle98@jj.com	LAVLEEN	DHALLA	RAIPUR	CHATISGARH	853578
CH99	Cheki9j@ih.com	CHIMAL	BEDI	TRICHY	TAMIL NADU	632011
DA74	Danu58@g.com	DANY	JAMES	TRICHY	TAMIL NADU	645018

Step 2: First Normal Form (1NF)

1NF requires that all values in the table be atomic (no repeating groups or multivalued attributes). In the given table, the data is already in atomic form, meaning each field contains a single value. Therefore, it is already in **1NF**.

Step 3: Second Normal Form (2NF)

2NF requires that the table be in **1NF** and that all non-key attributes are fully functionally dependent on the primary key. In this case, the candidate key is **UserID**. However, we can see that **City, State, and Zip** are dependent on each other rather than directly on the **UserID**. This violates **2NF**, as these columns (City, State, Zip) depend on each other.

Decompose the table into two tables to satisfy 2NF:

1. **User Table**:
   - Attributes that are dependent on **UserID** (direct user-related information).

2. **Location Table**:
    - Attributes that are location-specific (City, State, Zip), which are related to geographical location.

Decomposed Tables for 2NF:

Table 1: User

UserID	U_email	Fname	Lname	City
A12	Mani@ymail.com	MANISH	JAIN	BILASPUR
PO45	Pooja.g@gmail.co	POOJA	MAGG	KACCH
LA33	Lavle98@jj.com	LAVLEEN	DHALLA	RAIPUR
CH99	Cheki9j@ih.com	CHIMAL	BEDI	TRICHY
DA74	Danu58@g.com	DANY	JAMES	TRICHY

Table 2: Location

City	State	Zip
BILASPUR	CHATISGARH	458991
KACCH	GUJRAT	832212
RAIPUR	CHATISGARH	853578
TRICHY	TAMIL NADU	632011
TRICHY	TAMIL NADU	645018

Now, the **User** table has no partial dependency, and all non-key attributes depend fully on the primary key **UserID**. The **Location** table stores location-specific details and avoids redundancy.

Step 4: Third Normal Form (3NF)

3NF requires that the table be in **2NF**, and all attributes should be functionally dependent only on the primary key. In other words, there should be no transitive dependencies.

In the **Location** table, there is a dependency between **City** and the combination of **State** and **Zip**. Therefore, to achieve **3NF**, we need to further decompose the **Location** table.

Decomposition for 3NF:

We will break the **Location** table into two separate tables: one for **City-State** and another for **State-Zip**.

Final 3NF Tables:

Table 1: User

UserID	U_email	Fname	Lname	City
A12	Mani@ymail.com	MANISH	JAIN	BILASPUR
PO45	Pooja.g@gmail.co	POOJA	MAGG	KACCH
LA33	Lavle98@jj.com	LAVLEEN	DHALLA	RAIPUR
CH99	Cheki9j@ih.com	CHIMAL	BEDI	TRICHY
DA74	Danu58@g.com	DANY	JAMES	TRICHY

Table 2: City_State

City	State
BILASPUR	CHATISGARH
KACCH	GUJRAT
RAIPUR	CHATISGARH
TRICHY	TAMIL NADU

Table 3: State_Zip

State	Zip
CHATISGARH	458991
GUJRAT	832212
CHATISGARH	853578
TAMIL NADU	632011
TAMIL NADU	645018

Final Structure After 3NF

1. **User Table** (personal details linked to the city):
    - **UserID → U_email, Fname, Lname, City**
2. **City_State Table** (city-to-state mapping):
    - **City → State**
3. **State_Zip Table** (state-to-zip mapping):
    - **State → Zip**

Now, the database is in **3NF**, ensuring that:

- **1NF**: Atomic values are enforced.
- **2NF**: No partial dependencies (all non-key attributes depend on the full primary key).
- **3NF**: No transitive dependencies (non-key attributes depend only on the primary key).

3. **Question3:**

Q 1. Suppose, a relational schema R (A, B, C, D, E) and set of functional dependencies: F { A-> BC, CD-> E, B-> D, E-> A } Compute CD+ , E + (closure of attribute set CD, attribute E respectively).

Q 2. Suppose, a relational schema R (A, B, C, D, E, F) and set of functional dependencies: F { A-> B C, BC-> AD, D-> E, CF-> B } Compute BCF+ , CD+ , D + .

Q 3. Suppose, a relational schema R (A, B, C, D, E, F, G, H) and set of functional dependencies: F { A-> BC, E-> C, AH ->D, CD ->E, D-> AEH, DH-> BC } Compute AE + . Is BCD H valid or not?

Let's break down each of the questions and compute the closure of the attribute sets accordingly.

Question 1

Q1.1: Compute CD+\text{CD}^+CD+ (closure of the attribute set CD)

We are given the following functional dependencies on relation R(A,B,C,D,E)R(A, B, C, D, E)R(A,B,C,D,E):

1. A→BCA \to BCA→BC
2. CD→ECD \to ECD→E
3. B→DB \to DB→D
4. E→AE \to AE→A

To compute the closure CD+\text{CD}^+CD+, we need to iteratively apply the functional dependencies that are applicable based on the attributes we already know.

- Start with CDCDCD, i.e., CD+={C,D}\text{CD}^+ = \{C, D\}CD+={C,D}.
- From CD→ECD \to ECD→E, we can add EEE to the closure. Now, CD+={C,D,E}\text{CD}^+ = \{C, D, E\}CD+={C,D,E}.
- From E→AE \to AE→A, we can add AAA to the closure. Now, CD+={C,D,E,A}\text{CD}^+ = \{C, D, E, A\}CD+={C,D,E,A}.

- From A→BCA \to BCA→BC, we can add BBB and CCC to the closure. Now, CD+={A,B,C,D,E}\text{CD}^+ = \{A, B, C, D, E\}CD+={A,B,C,D,E}.

  So, CD+={A,B,C,D,E}\text{CD}^+ = \{A, B, C, D, E\}CD+={A,B,C,D,E}.

  Q1.2: Compute E+E^+E+ (closure of the attribute set E)

  Start with EEE, i.e., E+={E}E^+ = \{E\}E+={E}.

- From E→AE \to AE→A, we can add AAA to the closure. Now, E+={E,A}E^+ = \{E, A\}E+={E,A}.
- From A→BCA \to BCA→BC, we can add BBB and CCC to the closure. Now, E+={A,B,C,E}E^+ = \{A, B, C, E\}E+={A,B,C,E}.
- From B→DB \to DB→D, we can add DDD to the closure. Now, E+={A,B,C,D,E}E^+ = \{A, B, C, D, E\}E+={A,B,C,D,E}.

  So, E+={A,B,C,D,E}E^+ = \{A, B, C, D, E\}E+={A,B,C,D,E}.

Question 2

Q2.1: Compute BCF+\text{BCF}^+BCF+ (closure of the attribute set BCF)

We are given the following functional dependencies on relation R(A,B,C,D,E,F)R(A, B, C, D, E, F)R(A,B,C,D,E,F):

1. A→BCA \to BCA→BC
2. BC→ADBC \to ADBC→AD
3. D→ED \to ED→E
4. CF→BCF \to BCF→B

Start with BCF\text{BCF}BCF, i.e., BCF+={B,C,F}\text{BCF}^+ = \{B, C, F\}BCF+={B,C,F}.

- From CF→BCF \to BCF→B, we add BBB (already in the set), so no change.
- From BC→ADBC \to ADBC→AD, we can add AAA and DDD. Now, BCF+={A,B,C,D,F}\text{BCF}^+ = \{A, B, C, D, F\}BCF+={A,B,C,D,F}.
- From A→BCA \to BCA→BC, we add BBB and CCC (already in the set), so no change.
- From D→ED \to ED→E, we add EEE. Now, BCF+={A,B,C,D,E,F}\text{BCF}^+ = \{A, B, C, D, E, F\}BCF+={A,B,C,D,E,F}.

So, BCF+={A,B,C,D,E,F}\text{BCF}^+ = \{A, B, C, D, E, F\}BCF+={A,B,C,D,E,F}.

Q2.2: Compute CD+\text{CD}^+CD+ (closure of the attribute set CD)

Start with CD\text{CD}CD, i.e., CD+={C,D}\text{CD}^+ = \{C, D\}CD+={C,D}.

- From BC→ADBC \to ADBC→AD, we need both BBB and CCC to add AAA and DDD, but we only have CCC, so this dependency is not applicable yet.
- From D→ED \to ED→E, we can add EEE. Now, CD+={C,D,E}\text{CD}^+ = \{C, D, E\}CD+={C,D,E}.

So, CD+={C,D,E}\text{CD}^+ = \{C, D, E\}CD+={C,D,E}.

Q2.3: Compute D+D^+D+ (closure of the attribute set D)

Start with DDD, i.e., D+={D}D^+ = \{D\}D+={D}.

- From D→ED \to ED→E, we can add EEE. Now, D+={D,E}D^+ = \{D, E\}D+={D,E}.

So, D+={D,E}D^+ = \{D, E\}D+={D,E}.

Question 3

We are given the following functional dependencies on relation R(A,B,C,D,E,F,G,H)R(A, B, C, D, E, F, G, H)R(A,B,C,D,E,F,G,H):

1. A→BCA \to BCA→BC
2. E→CE \to CE→C
3. AH→DAH \to DAH→D
4. CD→ECD \to ECD→E
5. D→AEHD \to AEHD→AEH
6. DH→BCDH \to BCDH→BC

Q3.1: Compute AE+\text{AE}^+AE+ (closure of the attribute set AE)

Start with AEAEAE, i.e., AE+={A,E}\text{AE}^+ = \{A, E\}AE+={A,E}.

- From A→BCA \to BCA→BC, we can add BBB and CCC. Now, AE+={A,B,C,E}\text{AE}^+ = \{A, B, C, E\}AE+={A,B,C,E}.
- From E→CE \to CE→C, we can add CCC (already in the set), so no change.
- From D→AEHD \to AEHD→AEH, we need DDD, but we don't have it yet.
- From CD→ECD \to ECD→E, we need DDD, but we don't have it yet.

No further functional dependencies are applicable. Therefore:

AE+={A,B,C,E}\text{AE}^+ = \{A, B, C, E\}AE+={A,B,C,E}

Q3.2: Is BCDH\text{BCDH}BCDH valid?

To check if BCDH\text{BCDH}BCDH is valid (whether BCDH+\text{BCDH}^+BCDH+ gives us the complete set of attributes or at least specific attributes):

Start with BCDH\text{BCDH}BCDH, i.e., BCDH+={B,C,D,H}\text{BCDH}^+ = \{B, C, D, H\}BCDH+={B,C,D,H}.

- From DH→BCDH \to BCDH→BC, we can add BBB and CCC (already in the set), so no change.
- From D→AEHD \to AEHD→AEH, we can add A,E,HA, E, HA,E,H. Now, BCDH+={A,B,C,D,E,H}\text{BCDH}^+ = \{A, B, C, D, E, H\}BCDH+={A,B,C,D,E,H}.
- From A→BCA \to BCA→BC, we can add BBB and CCC (already in the set), so no change.

Therefore, BCDH+={A,B,C,D,E,H}\text{BCDH}^+ = \{A, B, C, D, E, H\}BCDH+={A,B,C,D,E,H}.

Since BCDH+\text{BCDH}^+BCDH+ does **not** cover the entire set of attributes (it does not include all attributes like FFF), **BCDH is not a superkey** for this relation and is **not valid**.