MODULE 4

Question 1: Illustrate different anomalies in designing a database

Answer: Database design anomalies are issues that can occur in poorly normalized databases, leading to data redundancy and inconsistency. They include:

- **Insertion Anomaly**: Adding new data can be problematic if mandatory fields need to be filled which may not have data available.
    - **Example**: Unable to add a product in a sales database without knowing its supplier because the supplier's information is in the same table.
- **Deletion Anomaly**: Deleting data can accidentally result in losing additional valuable data.
    - **Example**: Deleting the last product of a certain supplier could remove the supplier information from the database if no other products from that supplier exist.
- **Update Anomaly**: Changes to data can result in inconsistencies if not updated thoroughly across all rows.
    - **Example**: Changing a customer's address in one transaction record but failing to update it in others can lead to inconsistent data about the customer’s address.

Question 2: What is normalization ? why do we need normalization?

Answer: Normalization is a process of organizing data in a database to reduce redundancy and improve data integrity. Here’s why it's necessary:

1. **Reduce Redundancy**: Normalization minimizes duplicate data, saving storage and simplifying data management.
2. **Eliminate Anomalies**: It helps avoid issues during data insertion, update, and deletion, which can lead to inconsistencies.
    - **Insertion Anomalies**: Difficulty adding data without other necessary data.
    - **Update Anomalies**: Risks of partial updates leading to data mismatches.
    - **Deletion Anomalies**: Potential loss of important data when deleting other data.
3. **Improve Data Integrity**: Ensures data accuracy and consistency across the database.
4. **Optimize Queries**: Enhances query performance by streamlining data structure.
5. **Facilitate Concurrency Control**: Reduces deadlock risks and maintains data integrity during concurrent accesses.

Question 3: What are the key principles or informal guidelines to follow when designing a database schema? Additionally, could you explain the potential consequences or impact of violating these guidelines?

Key Guidelines for Database Schema Design

1. **Clear Semantics of Attributes**: Attributes should have clear, unambiguous meanings.
2. **Reduce Redundancy**: Avoid redundant information to save storage and simplify data management.
3. **Minimize NULL Values**: Limit NULL values to streamline query processing and enhance data integrity.
4. **Avoid Spurious Tuples**: Ensure schema design prevents invalid tuples when joining tables.

Consequences of Violating These Guidelines

- **Poor Data Quality**: Ambiguous attributes can degrade data reliability.
- **Increased Complexity**: Managing redundant data and NULLs can complicate updates, deletions, and insertions.
- **Anomalies**: Redundancy can lead to update, insertion, and deletion anomalies, impacting data integrity.
- **Inefficiencies**: Excessive NULLs and spurious tuples can slow down query processing and affect performance.

Question 4: Define functional dependency, illustrate it with a relevant example, and explain the purpose or significance of functional dependency in database design?

Definition of Functional Dependency: A functional dependency, noted as $X \rightarrow Y$, describes a relationship between two sets of attributes in a database schema where if two tuples have the same values for attributes in set X, they must have the same values for attributes in set Y. This indicates that the value of Y is determined by the value of X.

Example: In a company database, the functional dependency Employee Number $\rightarrow$ Employee Name suggests that each employee number uniquely determines an employee name.

Significance in Database Design: Functional dependencies are crucial for:

1. **Normalization**: They help organize the database to reduce redundancy and improve data integrity by identifying proper relationships for structuring tables.
2. **Integrity Constraints**: They are used to enforce data accuracy and consistency across the database.
3. **Query Optimization**: They assist in optimizing queries by facilitating efficient data retrieval plans based on attribute relationships.
4. **Robust Database Design**: They guide the structuring of tables to support application logic efficiently and ensure scalability.

Overall, functional dependencies are key to effective database design, ensuring data integrity, and optimizing database performance.

Question 5: What is MINIMAL COVER, How to find minimal cover

If a Functional Dependency F is given, then F' is Minimal cover of this FD set if F'
does not have
◦ Redundant Attributes
◦ Redundant Functional Dependency

How to find Minimal Cover
Steps
1. RHS of Functional Dependency must contain single attribute
eg if $A \rightarrow BC$ can be applied decomposition rule as $A \rightarrow B$, $A \rightarrow C$

2. LHS of Functional Dependency should contain single attribute, if possile only

Let the given set of FDs be E : {B → A, A → D, **AB → D**}
Here LHS has two attribute.
Check in E, if D of AB->D is already implied by any dependency
So, write AB->D as A->D and B->D
If any one is available in the FD set, we can remove AB->D
Here we have A->D, that is D is already implied, Hence we can remove AB->D
Resulting E = E : {B → A, A → D}

3. IF there is any trivial Functional Dependency , that can be removed
Eg : {AB ->B} is B is trivial since RHS & LHS have attributes B in common
4. Remove redundant Functional Dependency By using the transitive rule
◦ eg: E' : {B → A, D → A, B → D}
◦ By using the transitive rule on B → D and D → A, we derive B
→ A. Hence B → A is redundant and can be removed

Question 6: Let the given set of FDs be E : {B → A, A → D, AB → D}. Find the minimal cover of E.

Step 1: RHS must be single attributes
- it satisfies

Step 2: LHS must be single attributes
- it not satisfies as we have AB → D,
rewrite A → D and B → D
Check any one of above already there in FD set
E : {B → A, A → D, AB → D}
A → D exists, hence we can remove AB → D
reslting **E : {B → A, A → D}**

Step 3: Remove any FD with trivial FD
- Satisfies

Step 4: Remove any FD with transitive dependency
- Satisfies

Ans : Minimal Set = **E : {B → A, A → D}**

Question 7

F={A →B, C →B, D →ABC, AC →D}. Find the minimal cover

Step 1: RHS must be single attributes
- **D →ABC Not** satisfies
Using distribution rule we can write **D →ABC = (D →A, D →B, D →C)**

Set F = **F={A →B, C →B, D →A, D →B, D →C, AC →D}**

Step 2: LHS must be single attributes
- it not satisfies as we have AC → D,

rewrite A → D and C → D

Check any one of above already there in FD set

F={A →B, C →B, D →A, D →B, D →C, AC →D}

Both does NOT exists, hence we can NOT remove AC → D

reslting **F={A →B, C →B, D →A, D →B, D →C, AC →D}**

Step 3: Remove any FD with trivial FD

- Satisfies, no trivial

Step 4: Remove any FD with transitive dependency

- Does not Satisfy

we have **D →A, A →B, imply D →B**

D →B – already exists, so can be removed.

Ans : Minimal Set **F={A →B, C →B, D →A, D →C, AC →D}**

Question 8

find the minimal cover of set of Fds be E : {B → A, D → A, AB → D}

Step 1: RHS must be single attributes

- it satisfies

Step 2: LHS must be single attributes

- it not satisfies as we have AB → D,

rewrite A → D and B → D

Check any one of above already there in FD set

{B → A, D → A, AB → D}

Does not Exists.

Again check, if anyting can be transitively dependen on AB->D

Augment B with **B → A giving BB → AB that is B → AB**

So **B → AB , AB → D, therefore B->D can be in FD**

new set of FD = {B → A, D → A, AB → D, B->D}

Now AB->D rewrite A->D and B->D

Now we got matching B->D in FG set.

So **AB → D can be removed**

reslting F **: {B → A, D → A, B->D}**

Step 3: Remove any FD with trivial FD

- Satisfies

Step 4: Remove any FD with transitive dependency

- Not Satisfies

Set = F **: {B → A, D → A, B->D}**

B->D, and D → A, therefore B → A already implied

So B → A can bew removed

Ans : Minimal Set F **: {D → A, B->D}**

Question 9: what are Super Key, Candidate Key, Primary Key, Alternate Key ?. explan with example?

1. **Super Key:** Any set of attributes that can uniquely identify a record in a table. Example: {Student ID, Name}.
2. **Candidate Key**: A minimal super key; no subset of attributes can uniquely identify a record. All candidate keys are potential primary keys. Example: {Student ID} or {Email}.
3. **Primary Key**: A chosen candidate key used uniquely to identify table records. It must not contain NULL values. Example: {Student ID}.
4. **Alternate Key**: Candidate keys not chosen as the primary key. Example: {Email}, if {Student ID} is the primary key.

These keys ensure each record is uniquely identifiable and maintain database integrity.

Question 10: Inference Axioms and Armstrong Axioms

The six inference axioms used in relational databases for deriving functional dependencies: The first 3 are Armstrongs Axioms

1. **Reflexivity**: If $Y \subseteq X$, then $X \rightarrow Y$.
2. **Augmentation**: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any attribute set Z.
3. **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Extended rules derived from the primary axioms include:

4. **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.
5. **Decomposition**: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.
6. **Pseudotransitivity**: If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$.

These axioms help maintain database integrity by ensuring a consistent derivation of dependencies during database design, particularly in normalization processes.

Question 11: Explain 1NF, 2NF, 3NF, BCNF

The normal forms explained in your book are:

1. **First Normal Form (1NF)**: A relation is in 1NF if it only contains atomic values, meaning each attribute value must be indivisible. No attribute can have a set of values or a tuple as a value .
2. **Second Normal Form (2NF)**: A relation is in 2NF if it is already in 1NF and every non-prime attribute is fully functionally dependent on the primary key. This means there should be **no partial dependency** of any attribute on a part of the primary key .

Prime Attribute of CK --> NPA : Should NOT exist in 2NF

3. **Third Normal Form (3NF)**: A relation is in 3NF if it is in 2NF and there are **no transitive dependencies**; this means that non-prime attributes are not dependent on other non-prime attributes. Essentially, every non-prime attribute must be directly dependent only on the primary key .
    1. If X-->Y that is of the form NPA --> NPA

Y is transitively depend on a CK : Should NOT exist in 3NF

4. **Boyce-Codd Normal Form (BCNF)**: A relation schema R is in BCNF if every non-trivial functional dependency X → A involves a superkey X. This form is a stricter version of 3NF where every determinant must be a candidate key .

These definitions help ensure that databases are free from update anomalies, redundancy, and inconsistencies, thereby making them more efficient for operations such as query processing and maintaining integrity.

Question 12: Differentiate 3NF, BCNF with an example

Third Normal Form (3NF)

Definition: A table is in 3NF if it's in 2NF and all non-prime attributes are solely dependent on the primary key, not on any other non-prime attributes.

Example: Consider a table with `EmployeeID`, `EmployeeName`, `OfficeID`, and `OfficeAddress`. If `OfficeAddress` is dependent on `OfficeID`, which in turn is dependent on `EmployeeID`, the table is not in 3NF due to the transitive dependency.

To achieve 3NF:

- **Employees Table**: EmployeeID, EmployeeName, OfficeID
- **Offices Table**: OfficeID, OfficeAddress

Boyce-Codd Normal Form (BCNF)

Definition: A table is in BCNF if it's in 3NF and every determinant is a superkey, i.e., each functional dependency's left-hand side must be a candidate key.

Example: Consider a table with `CourseID`, `RoomNo`, and `Building`. If `RoomNo` and `Building` together can determine `CourseID`, then to be in BCNF, this combination should be a candidate key.

To achieve BCNF:

- **Courses Table**: CourseID, RoomID
- **Rooms Table**: RoomID, RoomNo, Building

Question 13: what is lossy join and lossless join . Explain with example

Decomposition in DBMS removes redundancy, anomalies and inconsistencies from a database by dividing the table into multiple tables.

• There are mainly two types of decompositions in DBMS-

1. Lossless Decomposition

2. Lossy Decomposition

Lossless Join Decomposition

A decomposition is **lossless** if, after splitting a relation R into subrelations and then performing a natural join on these subrelations, the original relation R is perfectly reconstructed without loss of information.

Example:
For relation R(A,B,C) with dependencies $A \rightarrow B$ and $B \rightarrow C$, decomposing into R1(A,B) and R2(B,C) is lossless.

1. R1 Union R2 = R

2. R1 Intersection R2 = Not NULL, that is there should be a common attribute

3. The common atribute should be a super key for R1, or R2 or Both R1 and R2.

    Thst is Let Common Attribute here in example is B.

    B+ = Super Key.

Lossy Join Decomposition

A decomposition is **lossy** if it introduces spurious tuples or loses data when the subrelations are joined back together, failing to recreate the original relation R.

Example:
If R(A,B,C) is decomposed into R1(A,C) and R2(B,C), the join of R1 and R2 might produce tuples not present in the original R, making this a lossy decomposition.

These concepts ensure data integrity in database normalization, preventing information loss and erroneous data entries during decomposition.

Question 14: check wheather below is lossless join or not

Given R(A,B,C)
• FD={$A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$}
• is decomposed to R1(A B) and R2(BC) check wheather it is lossless join or not

Answer. There is a common attribute in R1 and R2 ie, B.
Now check B is a candidate key for R1 or R2
B+=BCA
ie B can be a candidate key in both R1 and R2 so is Lossless decomposition

Question 15: check wheather below is lossless join or not

Given R(A,B,C,D)
• FD={$AB \rightarrow CD$, $D \rightarrow A$}
• is decomposed to R1(A C) and R2(B C D) check wheather it is lossless join or not

Answer. There is a common attribute in R1 and R2 ie, C.
Now check C is a candidate key for R1 or R2
C+=C

ie C is not a candidate key in both R1 and R2 so is Lossy decomposition

Question 16: Consider a relation schema R (A,B,C,D) with the following functional dependencies A → B, B → C, C → D, D → B. Determine whether the decomposition of R into R1 (A , B) , R2 (B , C) and R3 (B , D) is lossless or lossy. Write the complete steps.

Method 1 is not adequate for this. So follow algorithm method as below.

Step 1 – Fill ‘a’ label in rows corresponding to atributes present, that is R1(A,B) , mark a1 and a2 values in colums corresponding to A and B, Similarly for R2 (B , C), mark a2 and a3 in colums corresponds to B and C and so on for R3(B,D)

	A (a1)	B (a2)	C (a3)	D (a4)
R1 (A , B)	a1	a2		
R2 (B , C)		a2	a3	
R3 (B , D)		a2		a4

Step 2 – Fill all other columns with place holder ‘b’ values

	A (a1)	B (a2)	C (a3)	D (a4)
R1 (A , B)	a1	a2	b13	b14
R2 (B , C)	b21	a2	a3	b24
R3 (B , D)	b31	a2	b33	a4

Step 3 – Chek Fds { A → B, B → C, C → D, D → B }

For each row

A → B : If A column got lable ‘a’ values, then replace ‘b’ with ‘a’ values in B columns

B → C: If B column got lable ‘a’ values, then replace ‘b’ with ‘a’ values in C columns

C → D : If C column got lable ‘a’ values, then replace ‘b’ with ‘a’ values in D columns

D → B : If D column got lable ‘a’ values, then replace ‘b’ with ‘a’ values in B columns

	A (a1)	B (a2)	C (a3)	D (a4)
R1 (A , B)	a1	a2	~~b13~~ a3	~~b14~~ a4
R2 (B , C)	b21	a2	a3	~~b24~~ a4
R3 (B , D)	b31	a2	~~b33~~ a3	a4

Step 4 : Check

if any row has full ‘a’ values , then Lossless join

Else

repeat from Step 1, till no more chamge is possible.

	A (a1)	B (a2)	C (a3)	D (a4)
R1 (A , B)	a1	a2	~~b13~~ a3	~~b14~~ a4
R2 (B , C)	b21	a2	a3	~~b24~~ a4
R3 (B , D)	b31	a2	~~b33~~ a3	a4

First row has all ‘a’ vaues --> Hence lossless join.

Question 17: What is Dependency Preservation, explain with examples

Dependency Preservation ensures that after decomposing a database schema, each functional dependency (FD) can be enforced within individual decomposed relations without needing to join them. This is crucial for maintaining data integrity efficiently.

Example of Dependency Preservation

Suppose a relation R(A,B,C) has FDs $A \rightarrow B$ and $B \rightarrow C$. If we decompose R into:

- R1(A,B)
- R2(B,C)

This decomposition is dependency-preserving because:

- R1 enforces $A \rightarrow B$ on its own.
- R2 enforces $B \rightarrow C$ on its own.

Counterexample (Non-Preserving Decomposition)

Consider R(A,B,C,D) with FDs $A \rightarrow B$ and $C \rightarrow D$. Decomposing R into:

- R1(A,B,C)
- R2(C,D)

This may not preserve dependencies if $A \rightarrow B$ cannot be verified without considering C, which could be the case if indirect dependencies involving C affect B

That means For example, suppose there exists an unstated dependency like $BC \rightarrow A$ or any other dependency involving C that impacts A or B. In such cases, just looking at R1 might lead us to incorrect conclusions about the dependency $A \rightarrow B$ because changes in C (handled in R2) could indirectly affect B through A. Therefore, to ensure the dependency $A \rightarrow B$ holds as expected, we might need information from R2 as well, thereby requiring a join of R1 and R2 to fully enforce and verify this dependency.

Question 18. Check if dependency preservation satisfied or not ?

Example

R(A,B,C,D)
FD={$A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B$}
Decomposed into R1(AB), R2(B,C) and R3(B,D)

To check if the decomposition of a relation R into smaller relations R1,R2, and R3 preserves dependencies, we need to ensure that every functional dependency in R can be deduced from the dependencies in the decomposed relations **without the need for a join operation across tables for validation.** Let's apply this to the given decomposition:

Original Relation and Functional Dependencies

- **Relation R**: R(A,B,C,D)
- **Functional Dependencies F**: {$A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B$}

Decomposed Relations

- **R1(A,B)**: Possible FDs: $A \rightarrow B$ and $B \rightarrow A$

- **R2(B,C)**: Possible FDs: $B \rightarrow C$ and $C \rightarrow B$
- **R3(B,D)**: Possible FDs: $B \rightarrow D$ and $D \rightarrow B$

Checking Dependency Preservation

1. **$A \rightarrow B$**:
    - This is directly preserved in R1.
2. **$B \rightarrow C$**:
    - This is directly preserved in R2.
3. **$C \rightarrow D$**:
    - This dependency is **not directly preserved** in any single decomposed relation. R2 contains C, but not D, and R3 contains D, but not C.
    - To check $C \rightarrow D$ using the decomposed relations, a join operation would be necessary between R2 and R3, but these two relations do not share a common attribute that includes both C and D to validate the dependency directly.
4. **$D \rightarrow B$**:
    - This is directly preserved in R3.

Conclusion on Dependency Preservation

The dependency $C \rightarrow D$ is NOT preserved.

Question 19: Check if dependency preservation satisfied or not ?

Example R(A,B,C,D) FD={$A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B$}
Decomposed into R1(A,B), R2(B,C,D) and R3(B,D).

Original Relation and Functional Dependencies

Functional Dependencies Set

- $F=\{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B\}$

Decomposed Relations

- R1(A,B) Possible FDs: $A \rightarrow B$
- R2(B,C,D) Possible FDs: $B \rightarrow C$, $B \rightarrow D$, $C \rightarrow D$ $D \rightarrow B$
- R3(B,D) Possible FDs: $B \rightarrow D$ and $D \rightarrow B$

Verification of Dependency Preservation

1. **$A \rightarrow B$**
    - Clearly preserved in R1.
2. **$B \rightarrow C$**
    - Preserved in R2 since both B and C are in R2.
3. **$C \rightarrow D$**
    - Preserved in R2 since both C and D are in R2.
4. **$D \rightarrow B$**

- Preserved in R2 since both D and B are in R2.
- Additionally, it is also preserved in R3

Conclusion on Dependency Preservation

The dependency is preserved.

Question 20: Find Candidate Keys
Given R(ABCD),
F= {A->B, B->C, C->A}

1. Find the closure of all the attributes which will be definitely the superkey
(ABCD)+ = {A B C D} is a super key

2.Try to minimize the SK by looking at the FD
Given F= {A->B, B->C, C->A}
SK + {A B C D}
Since A->B is given B can be removed from SK
therefore SK = {A,C,D}

Since A->B, B->C therfore A->C, C can be removed from SK {A C D}
SK = {A,D}
N more reduction possible
Check if AD is SK, that is take closure AD+ = {A D B C}
Therefore {A D } is a Super Key
3. Check if (AD) is a CK
Check if candidate key (CK) by taking the closure of subset of SK and check if it is **NOT** a SK
SK ={A D}
Subset of AD is {A, D}. Find A+ and B+
A+ = {A, B, C} ---- Not SK
D+ = {D} ----- Not SK
Therefore A D is a Candidate Key
4. Check if any more Candidate Keys are there – If any of the Prime attribute is present in RHS of Functional dependency
Prime attributes = {A,D}
F= {A->B, B->C, C->A}, so we have C->**A**, prime attribute A is on RHS.
So there can be more CK

We have C->A, where A is on RHS From SK = {A D} replace A with C
So SK = {C D}

Closure of subset of CD
C+ = {C A B} ----- Not SK
D+ = { D} ----- Not SK
Therefore CD is a Candidate Key

Prime Attribute {C, D}Prime Attribute = {C, D}
F= {A->B, B->C, C->A}, so we have B->**C**, prime attribute C is on RHS
There can be more CK

We have B->C, where C is on RHS From SK = {C D} replace C with B
So SK = {B D}

Closure of subset of BD
B+ = {B C A} ----- Not SK
D+ = { D} ----- Not SK
Therefore BD is a Candidate Key

Prime Attribute = {B, D}
F= {A->B, B->C, C->A}, so we have A->**B**, prime attribute B is on RHS.
In SK ={B D} replace B with A
So SK = {A D}
We have already found {A D } as CK. So We can stop

There fore candidate keys are = CK ={AD, CD, BD}
Prime Attributes = {A,B,C,D}

Question 21: Consider the relation R = {A, B, C, D, E, F, G, H} and the set of functional dependencies F = {A→DE, B→F, AB→C, C→GH, G→H}. What is the key for R? Decompose R into 2NF and then 3NF relations.

To solve this problem, we first need to determine the candidate key for the relation R using the provided functional dependencies F. Then, we can proceed to decompose the relation into 2NF and then 3NF.

Step 1: Determining the Key for R

Minimising Default Super key (ABCDEFGH) ----> ABCFGH (Removed DE, as A->DE)

--> ABCGH (Removed F as B-->F) --> ABC (Removed GH, as C->GH) ---> AB (Removed C as AB->C)

So minimal SK = AB

1. Check if AB is CK
   Subset = (A, B)
   A+ = ADE Not s SK, Satisfies condition for CK
   B+ = BF Not s SK, Satisfies condition for CK

Step 2: Decomposing R into 2NF

2NF requires that no non-prime attribute is partially dependent on any candidate key. Here, the candidate key is AB, and all dependencies should be analyzed for partial dependencies on subsets of AB.

Given relations and FDs:

- R={A,B,C,D,E,F,G,H}
- FDs = {A→DE,B→F,AB→C,C→GH,G→H}
- PA = (A,B), NPA = (C,D,E,F,G,H)

From these, we note:

- A→DE: Subset of CK -> NPA, partial dependency : Not Satisfying 2NF
- B→F : Subset of CK -> NPA . partial dependency Not Satisfying 2NF
- AB→ C : CK -> NPA, Satisfy

- $C \rightarrow GH$ is a dependency that NPA --> NPA.; Satisfies 2NF
- $G \rightarrow H$ is a dependency that NPA --> NPA. Satisfies 2NF

To achieve 2NF, we eliminate partial dependencies:

- R3(A,D,E) with $A \rightarrow DE$
- R2(B,F) with $B \rightarrow F$
- R1(A,B,C,G,H) with $\{AB \rightarrow C, C \rightarrow GH, G \rightarrow H\}$

Step 3: Decomposing into 3NF

3NF requires that there are no transitive dependencies, i.e., a non-prime attribute must not depend on another non-prime attribute.

From our 3NF relations:

- R3(A,D,E) with $A \rightarrow DE$
- R2(B,F) with $B \rightarrow F$
- R1(A,B,C,G,H) with $\{AB \rightarrow C, C \rightarrow GH, G \rightarrow H\}$
- $C \rightarrow GH$ is a dependency that NPA --> NPA.; AB-> C, C->GH therefore AB->GH. Transitie dependency on CK AB, Hence not satsfying 3NF
- $G \rightarrow H$ is a dependency that NPA --> NPA. : AB-> C C->GH can be C->G and C->H, and G->H, therefore AB->H is transitive dependency on CK AB, Hence not satsfying 3NF

So Again Decompose

R1(ABCGH) into

R11(CGH) with $C \rightarrow GH$

R12(GH) $G \rightarrow H$

R1(A,B,C) with $AB \rightarrow C$

So final decomposition of R(ABCDEFGH)

1. R1(A,B,C) with $AB \rightarrow C$
2. R11(CGH) with $C \rightarrow GH$
3. R12(GH) with $G \rightarrow H$
4. R3(A,D,E) with $A \rightarrow DE$
5. R2(B,F) with $B \rightarrow F$

Question 22: Consider the following relation:

CAR_SALE(Car#, Date_sold, Salesperson#, Commission%, Discount_amt)

Assume that a car may be sold by multiple salespeople, and hence {Car#,Salesperson#} is the primary key.

Additional dependencies are :

Date_sold → Discount_amt and Salesperson# → Commission%

(i) Based on the given primary key and functional dependencies, is this relation in 1NF, 2NF, or 3NF? Why or why not?
(ii) How would you successively normalize it completely?

1. By default RDBMS tables are 1NF
2. Checking 2NF

CK is given

CK = {Car#,Salesperson#}
NPA = { Date_sold, , Commission%, Discount_amt}
Subset of CK = {Car#, Salesperson#}

FD = {Date_sold → Discount_amt,
Salesperson# → Commission%}

Checking 2NF:
Date_sold → Discount_amt : NPA-> NPA , Satisfies 2NF
Salesperson# → Commission%: Subset of CK -> NPA , Violates 2NF
Not in 2NF
Checking 3NF
Since not in 2NF, its also not in 3NF
3. Normalization
Decompsing 2NF
New Table R1 = CarSales1(Salesperson# , Commission%)
with Salesperson# → Commission%:
CAR_SALE(Car#, Date_sold, Salesperson#, Commission%, Discount_amt)
Resulting Table
CarSales1(Salesperson# , Commission%)
CAR_SALE(Car#, Date_sold, Salesperson#, Discount_amt)
Decomposing 3NF
Date_sold → Discount_amt : NPA-> NPA , But no transitivity
Hence in 3NF. No decomposition needed.

Resulting Table
CarSales1(Salesperson# , Commission%)
CAR_SALE(Car#, Date_sold, Salesperson#, Discount_amt)

Question 23: What is equivalet set of Functional dipendencis ?.

E = {A → B, AB → C, D → AC, D → E }
F = {A → BC, D → AE }
A set of functional dependencies E and F is Equivalent
if
◦ E covers F and F covers E.
E covers F means that all the Functional dependency
in F can be inferred from E , (i.e whether E is covering
functional dependencies of F)
F covers E means that all the Functional dependency
in E can be inferred from F (i.e whether F is covering
functional dependencies of E)

Example : Given two sets F and E of FDs for a relation.
E = {A → B, AB → C, D → AC, D → E }
F = {A → BC, D → AE }

Are the two sets equivalent?

E Covers F

Given E = {A → B, AB → C, D → AC, D → E }

F = {A → BC, D → AE }

A → BC

A+ = {A B C}

A+ includes B and C

D → AE

D+ = { D A C E B}

D+ includes A and E

Therefore E covers F

F Covers E

Given E = {A → B, AB → C, D → AC, D → E }

F = {A → BC, D → AE }

A → B

A+ = {A B C}

A+ includes B

AB → C

AB+ = {A B C}

AB+ includes C

D → AC

D+ = {D A E B C }

D+ includes A C

D → E

D+= { D A E B C }

D+ includes E

Therefore E covers F

Since E Covers F and F covers E , E is equivalent to F