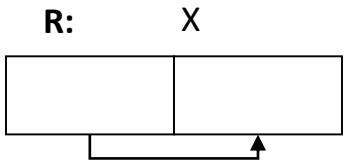
Describe functional dependency. OR Explain different types of FD with the help of example.

Functional Dependency

- Let R be a relation schema having n attributes A1, A2, A3,… An.
- Let attributes X and Y are two subsets of attributes of relation R.
- If the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component, then, there is a functional dependency from X to Y.
- This is denoted by X → Y.
- It is referred as: Y is functionally dependent on the X, or X functionally determines Y.
- The abbreviation for functional dependency is FD or fd.
- The set of attributes X is called the left hand side of the FD, and Y is called the right hand side.
- The left hand side of the FD is also referred as determinant whereas the right hand side of the FD is referred as dependent.

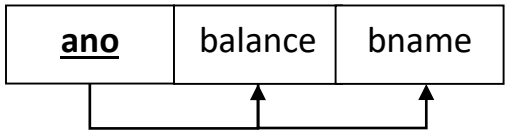
Diagrammatic representation



Example of functional dependency:

- Consider the relation Account (ano, balance, bname).
- In this relation ano can determines balance and bname. So, there is a functional dependency from ano to balance and bname.
- This can be denoted by ano → {balance, bname}.

Account:



Types of Functional Dependencies

Full Functional Dependency

- In a relation, the attribute B is fully functional dependent on A if B is functionally dependent on A, but not on any proper subset of A.
- Eg. {Roll_No, Department_Name, Semester} →SPI

Partial Functional Dependency

- In a relation, the attribute B is partial functional dependent on A if B is functionally dependent on A as well as on any proper subset of A.
- If there is some attribute that can be removed from A and the still dependency holds.

- Eg. {Enrollment_No, Department_Name} → SPI

Transitive Functional Dependency

- In a relation, if attribute(s) A→B and B→C, then C is transitively depends on A via B (provided that A is not functionally dependent on B or C).
- Eg. Staff_No → Branch_No and Branch_No → Branch_Address

Trivial Functional Dependency

- X→Y is trivial FD if Y is a subset of X
- Eg.{Roll_No, Department_Name} → Roll_No

Nontrivial Functional Dependency

- X→Y is nontrivial FD if Y is not a subset of X
- Eg. {Roll_No, Department_Name} → Student_Name

List and explain Armstrong's axioms (inference rules).

- Let A, B, and C is subsets of the attributes of the relation R.

Reflexivity

- If B is a subset of A then A → B

Augmentation

- If A → B then AC → BC

Transitivity

- If A → B and B → C then A → C

Pseudo Transitivity

If A → B and BC → D then AC → D

Self-determination

- A → A

Decomposition

- If A → BC then A → B and A → C

Union

- If A → B and A → C then A → BC

Composition

- If A → B and C → D then A,C → BD

What is closure of a set of FDs? How to find closure of a set of FDs.

Closure of set of functional dependency (FDs)

- Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F.
- E.g.: F = {A → B and B → C}, then we can infer that A → C
- The set of functional dependencies (FDs) that is logically implied by F is called the closure of F.
- It is denoted by F^+.

Example-1

Suppose a relation R is given with attributes A, B, C, G, H and I.
Also, a set of functional dependencies F is given with following FDs.
F = {A → B, A → C, CG → H, CG → I, B → H}
Find Closure of F.

We have	Using rule	Derived FD
A → B and B → H	Transitivity rule	A → H
CG → H and CG → I	Union rule	CG → HI
A → C and CG → I	Pseudo-transitivity rule	AG → I
A → C and CG → H	Pseudo-transitivity rule	AG → H

- F^+ = { A → H, CG → HI, AG → I, AG → H }

Example-2

Compute the closure of the following set F of functional dependencies for relational schema R = (A, B, C, D, E, F):
F = (A → B, A → C, CD → E, CD → F, B → E).

We have	Using rule	Derived FD
A → B and A → C	Union rule	A → BC
CD → E and CD → F	Union rule	CD → EF
A → B and B → E	Transitivity rule	A → E
A → C and CD → E	Pseudo- Transitivity rule	AD → E
A → C and CD → F	Pseudo- Transitivity rule	AD → F

- F^+ = { A → BC, CD → EF, A → E, AD → E, AD → F }

Example-3

Compute the closure of the following set F of functional dependencies for relational schema R = (A, B, C, D, E):
F = (AB → C, D → AC, D → E).

We have	Using rule	Derived FD
D → AC	Decomposition rule	D → A and D → C
D → AC and D → E	Union rule	D → ACE

F^+ = { D → A, D → C, D → ACE }

What is closure of a set of attributes? Explain how (Write an algorithm) to find closure of a set of attributes.

Closure of set of attributes

- Given a set of attributes α, the closure of α under F is the set of attributes that are functionally determined by α under F.
- It is denoted by α^+.

Algorithm

- Algorithm to compute α^+, the closure of a under F

```
result = α;
while (changes to result) do
        for each β → γ in F do
                begin
                        if β ⊆ result then  result = result ∪ γ
                end
```

Example:

Consider the following relation schema Depositer_Account (cid, ano, acess_date, balance, bname).
For this relation, a set of functional dependencies F can be given as
F = { {cid, ano} → access_date and **ano → {balance, bname} }**
Find out the closure of {cid, ano} and ano.

Solution

- Find out { cid, ano }$^+$
  - Step-1 :{ cid, ano }$^+$ = { cid, ano }
  - Step-2 :{ cid, ano }$^+$ = { cid, ano, acess_date } # {cid, ano} $\subseteq X^+$
    { cid, ano }$^+$ = { cid, ano, acess_date, balance, bname } # ano $\subseteq X^+$
  - Step-3 :{ cid, ano }$^+$ = { cid, ano, acess_date, balance, bname }

  So { cid, ano }$^+$ = { cid, ano, acess_date, balance, bname }
- Find out ano$^+$
  - Step-1 :ano$^+$ = ano
  - Step-2 :ano$^+$ = { ano, balance, bname } # ano $\subseteq X^+$
  - Step-3 :ano$^+$ = { ano, balance, bname }

  So ano$^+$ = { ano, balance, bname }

Given relation R with attributes A, B, C, D, E, F and set of FDs as F: {A → BC, E → CF, B → E and CD → EF}
Find out closure {A, B}+ of the set of attributes.

Steps to find the closure {A, B}+

Step-1: result = AB

Step-2: First loop

result = ABC	# for **A → BC**, A ⊆ result so result=result ∪ BC
result = ABC	# for **E→ CF**, E ∉ result so result=result
result = ABCE	# for **B→ E**, B ⊆ result so result=result ∪ E
result = ABCE	# for **CD→ EF**, CD ∉ result so result=result

result before step2 is AB and after step 2 is ABCE which is different so repeat same as step 2.

Step-3: Second loop

result = ABCE	# for **A → BC**, A ⊆ result so result=result ∪ BC
result = ABCEF	# for **E→ CF**, E ⊆ result so result=result ∪ CF
result = ABCEF	# for **B→ E**, B ⊆ result so result=result ∪ E
result = ABCEF	# for **CD→ EF**, CD ∉ result so result=result

result before step3 is ABCE and after step 3 is ABCEF which is different so repeat same as step 3.

Step-4: Third loop

result = ABCEF	# for **A → BC**, A ⊆ result so result=result ∪ BC
result = ABCEF	# for **E→ CF**, E ⊆ result so result=result ∪ CF
result = ABCEF	# for **B→ E**, B ⊆ result so result=result ∪ E
result = ABCEF	# for **CD→EF**, CD ∉result so result=result

Result before step4 is ABCEF and after step 3 is ABCEF which is same so stop.

So, Closure of {A, B}+ is {A, B, C, E, F}.

What is decomposition? Explain different types of decomposition.

Decomposition

- Decomposition is the process of breaking down given relation into two or more relations.
- Here, relation R is replaced by two or more relations in such a way that -
    1. Each new relation contains a subset of the attributes of R, and
    2. Together, they all include all tuples and attributes of R.
- Relational database design process starts with a universal relation schema R = {A1, A2, A3,... An), which includes all the attributes of the database. The universal relation states that every attribute name is unique.
- Using functional dependencies, this universal relation schema is decomposed into a set of relation schemas D = {R1, R2, R3,…, Rm}.
- Now, D becomes the relational database schema and D is referred as decomposition of R.
- Generally, decomposition is used to eliminate the pitfalls of the poor database design during normalization process.
- For example, consider the relation Account_Branch given in figure:

Account_Branch			
Ano	**Balance**	**Bname**	**Baddress**
A01	5000	Vvn	Mota bazaar, VVNagar
A02	6000	Ksad	Chhota bazaar, Karamsad
A03	7000	Anand	Nana bazaar, Anand
A04	8000	Ksad	Chhota bazaar, Karamsad
A05	6000	Vvn	Mota bazaar, VVNagar

- This relation can be divided with two different relations
    1. Branch (Bname, Baddress)
    2. Account (Ano, Balance, Bname)
- These two relations are shown in below figure

Branch	
Bname	**Baddress**
Vvn	Mota bazaar, VVNagar
Ksad	Chhota bazaar, Karamsad
Anand	Nana Bazar, Anand

-

Account		
Ano	**Balance**	**Bname**
A01	5000	Vvn
A02	6000	Ksad
A03	7000	Anand
A04	8000	Ksad
A05	6000	Vvn

- A decomposition of relation can be either lossy decomposition or lossless decomposition.
- There are two types of decomposition

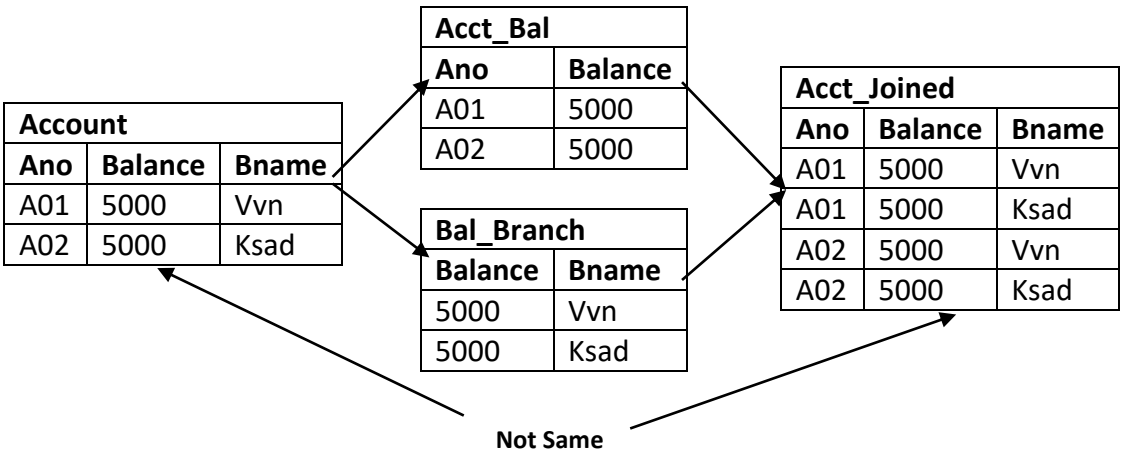
1. **Lossy decomposition**
2. **Lossless decomposition** (non-loss decomposition)

Lossy Decomposition

- The decomposition of relation R into R1 and R2 is lossy when the join of R1 and R2 does not yield the same relation as in R.
- This is also referred as lossy-join decomposition.
- The disadvantage of such kind of decomposition is that some information is lost during retrieval of original relation. And so, such kind of decomposition is referred as lossy decomposition.
- From practical point of view, decomposition should not be lossy decomposition.

Example of Lossy decomposition:

- A figure shows a relation Account. This relation is decomposed into two relations Acc_Bal and Bal_Branch.
- Now, when these two relations are joined on the common attributeBalance, the resultant relation will look like Acct_Joined. This Acct_Joined relation contains rows in addition to those in original relation Account.
- Here, it is not possible to specify that in which branch account A01 or A02 belongs.
- So, information has been lost by this decomposition and then join operation.



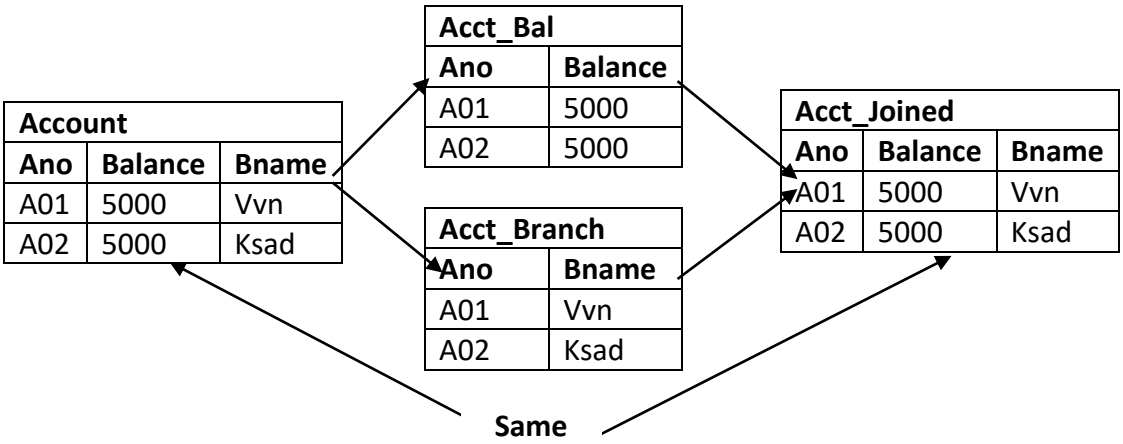
- In other words, decomposition is lossy if decompose into R1 and R2 and again combine (join) R1 and R2 we cannot get original table as R1, over X, where R is an original relation, R1 and R2 are decomposed relations, and X is a common attribute between these two relations.

Lossless (Non-loss) Decomposition

- The decomposition of relation R into R1 and R2 is lossless when the join of R1 and R2 produces the same relation as in R.
- This is also referred as a non-additive (non-loss) decomposition.
- All decompositions must be lossless.

Example of lossless decomposition:

- Again, the same relation Account is decomposed into two relations Acct_Bal and Acct_Branch.
- Now, when these two relations are joined on the common column Ano, the resultant relation will look like Acc_Joined relation. This relation is exactly same as that of original relation Account.
- In other words, all the information of original relation is preserved here.
- In lossless decomposition, no any fake tuples are generated when a natural join is applied to the relations in the decomposition.
- In other words, decomposition is lossy if R = join of R1 and R2, over X, where R is an original relation, R1 an R2 are decomposed relations, and x is a common attribute between these two relations.



What is an Anomaly in database design? How it can be solved.

Anomaly in database design:

- Anomalies are problems that can occur in poorly planned, un-normalized database where all the data are stored in one table.
- There are three types of anomalies that can arise in the database because of redundancy are
  - ✓ Insert anomalies
  - ✓ Delete anomalies
  - ✓ Update / Modification anomalies
- Consider a relation emp_dept (E#, Ename, Address, D#, Dname, Dmgr#) with E# as a primary key.

Insert anomaly:

- Let us assume that a new department has been started by the organization but initially there is no employee appointed for that department, then the tuple for this department cannot be inserted in to this table as the E# will have NULL value, which is not allowed because E# is primary key.
- This kind of problem in the relation where some tuple cannot be inserted is known as insert anomaly.

Delete anomaly:

- Now consider there is only one employee in some department and that employee leaves the organization, then the tuple of that employee has to be deleted from the table, but in addition to that information about the department also will be deleted.
- This kind of problem in the relation where deletion of some tuples can lead to loss of some other data not intended to be removed is known as delete anomaly.

Update / Modification anomaly:

- Suppose the manager of a department has changed, this requires that the Dmgr# in all the tuples corresponding to that department must be changed to reflect the new status. If we fail to update all the tuples of given department, then two different records of employee working in the same department might show different Dmgr# lead to inconsistency in the database.
- This kind of problem is known as update or modification anomaly.

How anomalies in database design can be solved:

- Such type of anomalies in database design can be solved by using normalization.

What is normalization? Why it is needed? OR What is normalization? Why normalization process is required?

Normalization

- Database normalization is the process of removing redundant data from your tables to improve storage efficiency, data integrity, and scalability.
- In the relational model, methods exist for quantifying how efficient a database is. These classifications are called normal forms (or NF), and there are algorithms for converting a given database between them.
- Normalization generally involves splitting existing tables into multiple ones, which must be re-joined or linked each time a query is issued.

Need/Requirement of Normalization

- Eliminates redundant data
- Reduces chances of data errors
- Reduces disk space
- Improve data integrity, scalability and data consistency.

Explain different types of normal forms with example. OR Explain 1NF, 2NF, 3NF, BCNF, 4NF and 5NF with example.

1NF

- A relation R is in first normal form (1NF) if and only if all underlying domains contain atomic values only. **OR**
- A relation R is in first normal form (1NF) if and only if it does not contain any composite or multi valued attributes or their combinations.

Example of 1NF:

Cid	**Name**	**Address**		**TypeofAccountHold**
		Society	**City**	
C01	Riya	SaralSoc, Aand		Saving, Current, Salary
C02	Jiya	Birla Gruh, Rajkot		Saving, Current

- Above relation has four attributes Cid, Name, Address, Contact_no. Here address is composite attribute which is further divided in to sub attributes as Society and City. Another attribute TypeofAccountHold is multi valued attribute which can store more than one values. So above relation is not in 1NF.

Problem

- Suppose we want to find all customers for some particular city then it is difficult to retrieve. Reason is city name is combined with society name and stored whole as address.

Solution

- Divide composite attributes into number of sub- attribute and insert value in proper sub attribute. AND
- Split the table into two tables in such a way that
  - first table contains all attributes except multi-valued attribute and
  - other table contains multi-valued attribute and
  - insert primary key of first table in second table as a foreign key.
- So above table can be created as follows.

Cid	**Name**	**Society**	**City**
C01	Riya	SaralSoc	Aand
C02	Jiya	Birla Gruh	Rajkot

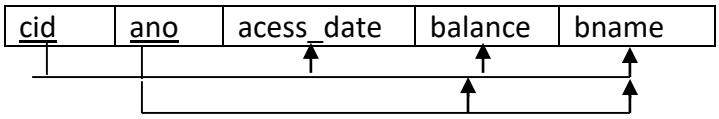
PhID	**Cid**	**Contact_no**
P01	C01	9879898798
P02	C01	9898052340
P03	C02	9825098254

2NF

- A relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key. **OR**

- A relation R is in second normal form (2NF) if and only if it is in 1NF and no any non-key attribute is partially dependent on the primary key.

Example of 2NF:



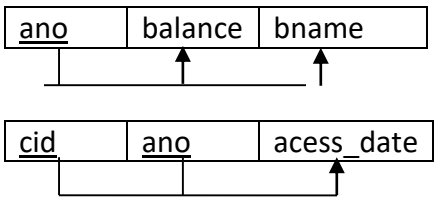
- Above relation has five attributes cid, ano, acess_date, balance, bname and two FDS
  FD1 {cid,ano}→{acess_date,balance,bname} and
  FD2 ano→{balance,bname}
- We have cid and ano as primary key. As per FD2 balace and bname are only depend on ano not cid. In above table balance and bname are not fully dependent on primary key but these attributes are partial dependent on primary key. So above relation is not in 2NF.

Problem

- For example in case of joint account multiple customers have common accounts. If some account says 'A02' is jointly by two customers says 'C02' and 'C04' then data values for attributes balance and bname will be duplicated in two different tuples of customers 'C02' and 'C04'.

Solution

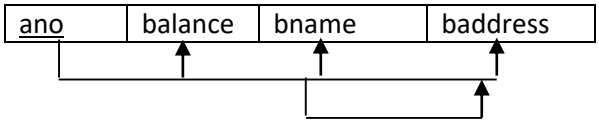
- Decompose relation in such a way that resultant relation does not have any partial FD.
- For this purpose remove partial dependent attribute that violets 2NF from relation. Place them in separate new relation along with the prime attribute on which they are full dependent.
- The primary key of new relation will be the attribute on which it if fully dependent.
- Keep other attribute same as in that table with same primary key.
- So above table can be decomposed as per following.



3NF

- A relation R is in third normal form (3NF) if and only if **it is in 2NF** and every non-key attribute is **non-transitively dependent on the primary key**.
- An attribute C is transitively dependent on attribute A if there exist an attribute B such that: A → B and B → C.

Example of 3NF:



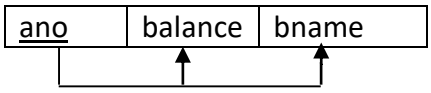
- Above relation has four attributes ano, balance, bname, baddress and two FDS
  FD1 ano→{balance, bname, baddress} and
  FD2 bname→baddress
- So from FD1 and FD2 and using transitivity rule we get ano→baddress.
- So there is transitively dependency from ano to baddress using bname in which baddress is non-prime attribute.
- So there is a non-prime attribute baddress which is transitively dependent on primary key ano.
- So above relation is not in 3NF.

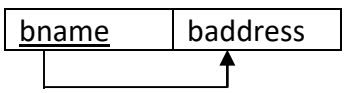
Problem

- Transitively dependency results in data redundancy.
- In this relation branch address will be stored repeatedly from each account of same branch which occupy more space.

Solution

- Decompose relation in such a way that resultant relation does not have any non-prime attribute that are transitively dependent on primary key.
- For this purpose remove transitively dependent attribute that violets 3NF from relation. Place them in separate new relation along with the non-prime attribute due to which transitive dependency occurred. The primary key of new relation will be this non-prime attribute.
- Keep other attributes same as in that table with same primary key.
- So above table can be decomposed as per following.



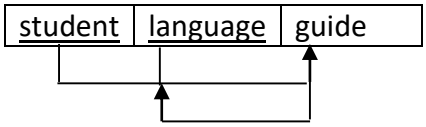


BCNF

- A relation R is in BCNF if and only if it is in 3NF and no any prime attribute is transitively dependent on the primary key. **OR**
- A relation R is in BCNF if and only if it is in 3NF and for every functional dependency X → Y, X should be the super key of the table **OR**
- A relation R is in BCNF if and only if it is in 3NF and for every functional dependency X → Y, X should be the primary key of the table.
- An attribute C is transitively dependent on attribute A if there exist an attribute B such that A→B and B→C.

Example of BCNF:

Student_Project		
Student	**Language**	**Guide**
Mita	JAVA	Patel
Nita	VB	Shah
Sita	JAVA	Jadeja
Gita	VB	Dave
Rita	VB	Shah
Nita	JAVA	Patel
Mita	VB	Dave
Rita	JAVA	Jadeja



- Above relation has five attributes cid, ano, acess_date, balance, bname and two FDS
  FD1 {student, language}→guide and
  FD2 guide→language
- So from FD1 and FD2 and using transitivity rule we get student→language
- So there is transitively dependency from student to language in which language is prime attribute.
- So there is on prime attribute language which is transitively dependent on primary key student.
- So above relation is not in BCNF.

Problem

- Transitively dependency results in data redundancy.
- In this relation one student have more than one project with different guide then records will be stored repeatedly from each student and language and guides combination which occupies more space.

Solution

- Decompose relation in such a way that resultant relation does not have any prime attribute transitively dependent on primary key.
- For this purpose remove transitively dependent prime attribute that violets BCNF from relation. Place them in separate new relation along with the non-prime attribute due to which transitive dependency occurred. The primary key of new relation will be this non-prime attribute.
- So above table can be decomposed as per following.

Student	**Guide**
Mita	Patel
Nita	Shah
Sita	Jadeja
Gita	Dave
Rita	Shah
Nita	Patel
Mita	Dave
Rita	Jadeja

Guide	**Language**
Patel	JAVA
Shah	VB
Jadeja	JAVA
Dave	VB

4NF

- A table is in the 4NF if **it is in BCNF** and has **no non multivalued dependencies**.

Example of 4NF:

- The multi-valued dependency X $\twoheadrightarrow$ Y holds in a relation R if for a dependency X → Y, if for a single value of X, multiple (more than one) values of Y exists.
- Suppose a student can have more than one subject and more than one activity.

Student_Info		
Student_Id	**Subject**	**Activity**
100	Music	Swimming
100	Accounting	Swimming
100	Music	Tennis
100	Accounting	Tennis
150	Math	Jogging

- Note that all three attributes make up the Primary Key.
- Note that Student_Id can be associated with many subject as well as many activities (multi-valued dependency).
- Suppose student 100 signs up for skiing. Then we would insert (100, Music, Skiing). This row implies that student 100 skies as Music subject but not as an accounting subject, so in order to keep the data consistent we must add one more row (100, Accounting, Skiing). This is an insertion anomaly.

- Suppose we have a relation R(A) with a multivalued dependency X $\twoheadrightarrow$ Y. The MVD can be removed by decomposing R into R1(R - Y) and R2(X U Y).
- Here are the tables Normalized

Student_Id	Subject
100	Music
100	Accounting
150	Math

StudentId	Activity
100	Swimming
100	Tennis
150	Jogging

5NF (Optional)

- A table is in the 5NF if it is in 4NF and if for all Join dependency (JD) of (R_1, R_2, R_3, ...,R_m) in R, every R_i is a superkey for R. **OR**
- A table is in the 5NF if it is in 4NF and if it cannot have a lossless decomposition in to any number of smaller tables (relations).
- It is also known as Project-join normal form (PJ/NF).

Example of 5NF:

- We have a table that contains students subectwise result as per follows:

ResultID	RollNo	StudentName	SubjectName	Result
1	101	Raj	DBMS	Pass
2	101	Raj	DS	Pass
3	101	Raj	DE	Pass
4	102	Meet	DBMS	Pass
5	102	Meet	DS	Fail
6	102	Meet	DE	Pass
7	103	Suresh	DBMS	Fail
8	103	Suresh	DS	Pass
9	103	Suresh	DE	Fail

- Above table is not in 5NF because we can decompose into sub tables.
- If we decompose above table into multiple table as per follows:

ResultID	RollNo	SubjectID	Result
1	101	1	Pass
2	101	2	Pass
3	101	3	Pass
4	102	1	Pass
5	102	2	Fail
6	102	3	Pass
7	103	1	Fail

8	103	2	Pass
9	103	3	Fail

RollNo	**StudentName**
101	Raj
102	Meet
103	Suresh

SubjectID	**SubjectName**
1	DBMS
2	DS
3	DE

- We cannot decomposition any of above three tables into the sub tables so above three tables are in 5NF.

Test Case – I

A college maintains details of its lecturers' subject area skills. These details comprise: Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name, Subject Code, Subject Name and Subject Level. Assume that each lecturer may teach many subjects but may not belong to more than one department. Subject Code, Subject Name and Subject Level are repeating fields. Normalize this data to Third Normal Form.

UNF

- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name, Subject Code, Subject Name, Subject Level

1NF

- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name
- Lecturer Number, Subject Code, Subject Name, Subject Level

2NF

- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name
- Lecturer Number, Subject Code
- Subject Code, Subject Name, Subject Level

3NF

- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code
- Department Code, Department Name
- Lecturer Number, Subject Code
- Subject Code, Subject Name, Subject Level

Test Case – II

A software contract and consultancy firm maintain details of all the various projects in which its employees are currently involved. These details comprise: Employee Number, Employee Name, Date of Birth, Department Code, Department Name, Project Code, Project Description and Project Supervisor

Assume the following:

1. Each employee number is unique.
2. Each department has a single department code.
3. Each project has a single code and supervisor.
4. Each employee may work on one or more projects.
5. Employee names need not necessarily be unique.
6. Project Code, Project Description and Project Supervisor are repeating fields.

Normalize this data to Third Normal Form.

UNF

- Employee Number, Employee Name, Date of Birth, Department Code, Department Name, Project Code, Project Description, Project Supervisor

1NF

- Employee Number, Employee Name, Date of Birth, Department Code, Department Name
- Employee Number, Project Code, Project Description, Project Supervisor

2NF

- Employee Number, Employee Name, Date of Birth, Department Code, Department Name
- Employee Number, Project Code,
- Project Code, Project Description, Project Supervisor

3NF

- Employee Number, Employee Name, Date of Birth, Department Code
- Department Code, Department Name
- Employee Number, Project Code
- Project Code, Project Description, Project Supervisor