1. Explain Functional dependency diagram with example. Explain various types of functional dependencies. Explain importance of functional dependency in database design.

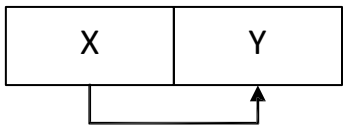
Functional Dependency

- 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 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 f.d.
- The set of attributes **X** is called the left hand side of the FD, and **Y** is called the right hand side of the FD.
- The left hand side of the FD is also referred as **determinant** whereas the right hand side of the FD is referred as **dependent**.

Functional Dependency Diagram and Examples

- Here functional dependency X → Y for relation schema R has been described
- Each functional dependency is displayed as horizontal line.
- The left hand side of the FD is also referred as determinant are connected by vertical lines to the line representing the FD.
- the right hand side of the FD is referred as dependent are connected by arrow pointing

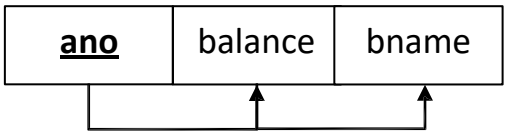
R:



Example:

- Consider the relation **Account(<u>ano</u>, 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}**.

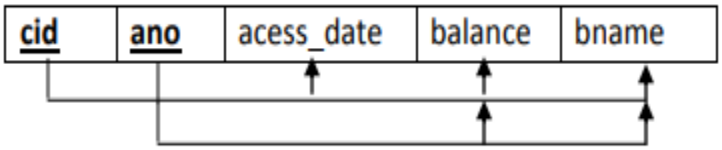
Account:



Full function Dependency

- A functional dependency **X → Y** in relation schema **R** is fully functional dependency if **Y** is not functionally dependent on any subset of **X**.

Example



- If **cid** or **ano** is removed from the primary key **access_date** cannot be determined uniquely. So **access_date** is fully functional dependent on **cid** and **ano**.
- But **balance** and **bname** depend only on the **ano**. If **cid** is removed from the primary key even then dependency of **balance** and **bname** on **ano** holds. So **balance** and **bname** do not fully functionally dependent on primary key i.e. cid and ano. In other word balance and bname partially depend on primary key.

Trivial v/s Non-Trivial Functional Dependencies

Trivial:

- If a FD: X → Y is trivial if and only if the right hand side is **subset** of the left hand side.
- If a FD: X → Y is trivial if and only if **Y** ⊆ **X** i.e Y is contained in x.
- **Example :** {cid,ano}→cid

Non-trivial:

- If a FD: X → Y is non-trivial if and only if the right hand side is **not a subset** of the left hand side.
- If a FD: X → Y is non-trivial if Y is not contained in x.
- **Example :** {cid,ano}→access_date

Armstrong's Axioms for Functional Dependencies

- Armstrong's Axioms is used to derive new FDs from other FDs.
- Let assume that a relation **R** contains attribute-sets **A,B,C and D**.

1. **Reflexivity**

- If B is a subset of A then A → B. (**B**⊆ **A**)

2. **Augmentation**

- If A → B then AC → BC.

3. **Transitivity**

- If A → B and B → C then A → C.

4. **Pseudo-transitivity**

- If A → B and BD → C then AD → C.

5. **Self-determination**

- A → A.

6. **Decomposition**

- If A → BC then A → B and A → C.

7. **Union**

- If A → B and A → C then A → BC.

8. **Composition**

- If A → B and C → D then A,C → BD.

9. **Self-accumulation**

- If A → BC and C→D then A → BCD.

2. Explain with algorithm and example how to determine redundant functional dependency from the given set of FDs.

- A FD in the set is redundant, if it can be derived from the other FDs in the set.
- Redundant FD can be detected by using **Membership** algorithm.

Algorithm

Input: Let **F** be a set of FDs for relation **R**.

Let **f: A → B** is a FD to be examined for redundancy.

Steps:

1. **F' = F – f** # find out new set of FDs by removing f from F
2. **T = A** # set T = determinant of A → B
3. For each **FD: X → Y** in F' Do
   If **X ⊆ T** Then # if X is contained in T
   T = T U Y # add Y to T
   End if
   End For
4. If **B ⊆ T** Then # if B is contained in T
   f : A → B is redundant. # given FD f : A → B is redundant.

End if

Output: Decision whether a given FD f : A → B is redundant or not.

Example

- Suppose a relation **R** is given with attributes **A, B, C, D, E**.
- Also, a set of functional dependencies F is given with following FDs.

F = {A →B, C →D,BD → E, AC → E}

1. Find out whether a FD **f: AC → E** is redundant or not.

Step-1: F'= { A→ B, C → D, BD → E} # F' = F – f

Step-2: T = AC # set T = determinant of AC → E

Step-3: T = AC + B = ACB # A → B is in F' and A ⊆ T

T = ACB+D=ACBD # C→D is in F' and C ⊆ T

T = ACBD+E=ACBDE # BD→E is in F' and BD ⊆ T

Step-4: f: AC → E is redundant. # E ⊆ T

2. Find out whether a FD **f: BD → E** is redundant or not.

Step-1: F'= { A→ B, C → D, AC→ E} #F' = F – f

Step-2: T = BD # set T = determinant of f: BD→ E

Step-3: Nothing can be added to T,

As there is no other FD: X → Y such that X ⊆ T

Step-4: f: BD→ E is not redundant. # E is not contained in T

Closure of a set of Function Dependencies

- A closure of a set of FDs is a set of all possible FDs that can be derived from a given set of FDs. It is also referred as a complete set of FDs.
- If F is used to denote the set of FDs for relation R, then a closure of a set of FDs implied by F is denoted by $\mathbf{F}^+$.

- **Algorithm:** Determining $\mathbf{X}^+$, the closure of **X** under **F**.

Input: Let **F** be a set of FDs for relation **R**.

Steps:

1. $\mathbf{X}^+ = \mathbf{X}$ # Initialize X^+ to X.
2. **Repeat** # Traverse loop
   a) **old $\mathbf{X}^+ = \mathbf{X}^+$** # Save X^+ to old X^+
   b) **For each FD: Y → Z in F Do**
      If **$Y \subseteq X^+$** Then # if Y is contained in X^+
         $X^+ = X^+ \cup Z$ # add Z to X^+
      End if
   End For

Until (X^+ = **old**X^+) # Loop through step-2 until no new attributes are found.

3. Return X^+ # Return closure of **X.**

Output: Closure X^+ of **X** under **F**.

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 , ano → { balance, bname } }
- Find out the closure of **F**.

Solution

- Determine each set of attributes **X** that appears as a left-hand side of FD in F.

  { cid, ano } and **ano**.

- Find out **{ cid, ano }**$^+$

Step-1 : { cid, ano }$^+$ = { cid, ano }

Step-2 : { cid, ano }$^+$ = { cid, ano, acess_date } # {cid, ano} ⊆ X^+

{ cid, ano }$^+$ = { cid, ano, acess_date, balance, bname } # ano⊆ X^+

Step-3 : { cid, ano }$^+$ = { cid, ano, acess_date, balance, bname }

- Find out **ano**$^+$

Step-1 : ano$^+$ = ano

Step-2 : ano$^+$ = { ano, balance, bname } # ano ⊆ X^+

Step-3 : ano$^+$ = { ano, balance, bname }

Combine all such sets of X^+ to form a closure of **F**.

{ cid, ano }$^+$ = { cid, ano, acess_date, balance, bname }

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

3. What is Decomposition? Explain types of Decomposition with example. OR What is Decomposition? Explain Lossy join and Lossless join Decomposition with example.

Decomposition

- Decomposition is the process of breaking down given relation into two or more relations.
- Here, a 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 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 problem 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. **Account (Ano, Balance, Bname)**
2. **Branch (Bname, Baddress)**

These two relations are shown in below figure:

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

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

- A decomposition of a relation can be either lossy decomposition or lossless join decomposition.

There are two types of decomposition

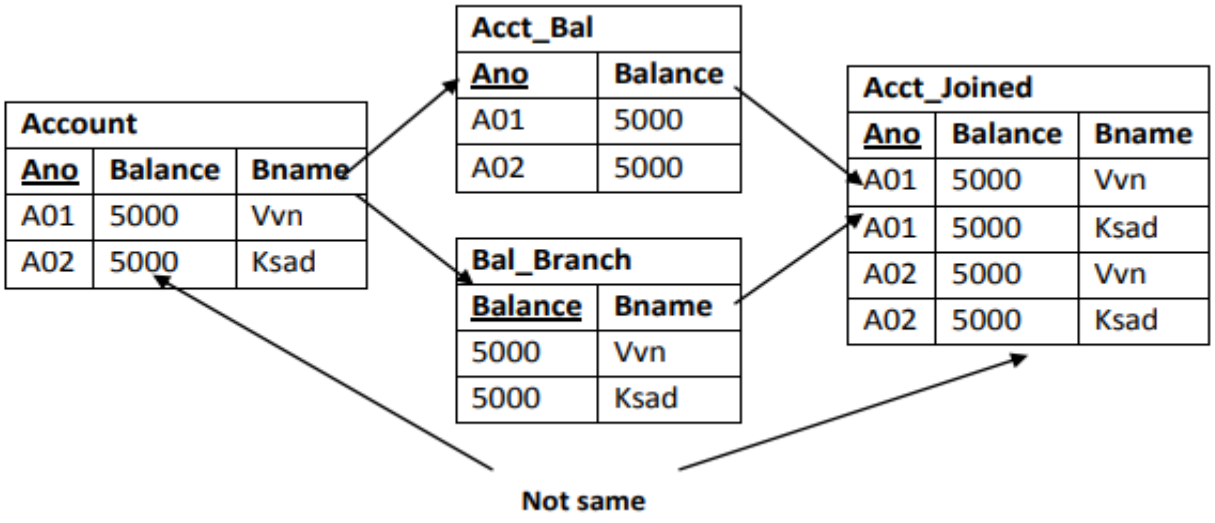
1. **Lossy Decomposition**
2. **Lossless Join Decomposition**

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

- 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 attribute Balance, 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.



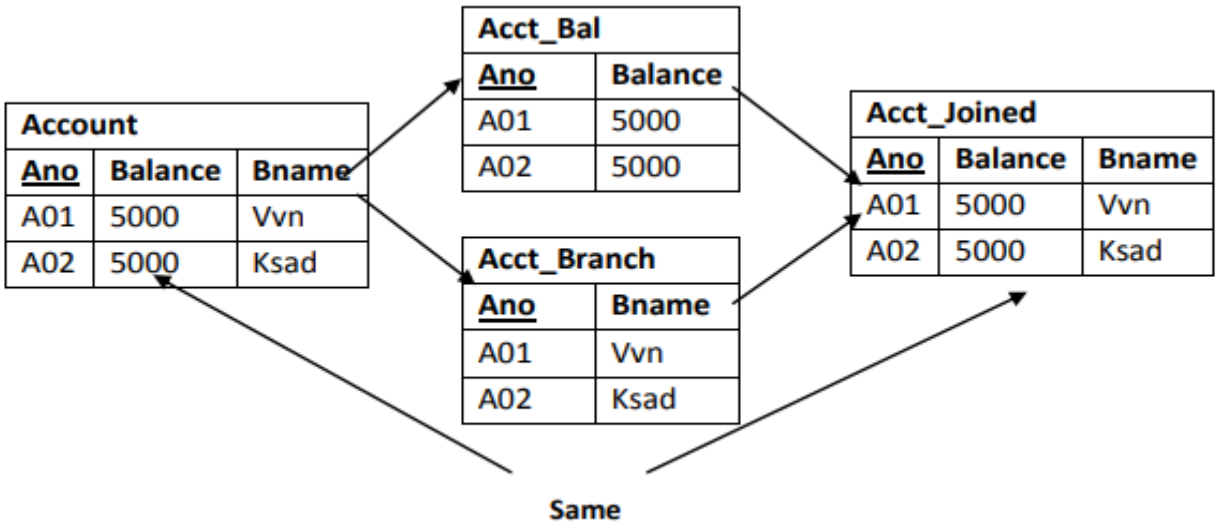
- In other words, decomposition is lossy if **R** decompose into **R1** and **R2** and again **combine (join)** **R1** and **R2** we don't get original table as **R**, over **X**, where **R** is an original relation, **R1** and **R2** are decomposed relations, and **X** is a common attribute between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

Lossless Join 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 non-additive decomposition. All decompositions must be lossless.

Example

- 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.



4. Explain dependency preserving decomposition with example.

- When any relation is decomposed, **illegal** relations should not be created. A relation is an **illegal** relation, if it does not preserve given functional dependencies.
- A relation **R** is decomposed into the relation schema **R1, R2, ... , Rn** with the functional dependencies **F1,F2, .. ,Fn**. Let **F' = F1 U F2 U... U Fn**.
- This decomposition is dependency preserving decomposition, if closure of **F'** is identical to F^+, i.e $F'^+ = F^+$.
- Here, closure is considered rather than simple set of FDs. Because even if **F' ≠ F**. It may be that, $F'^+ = F^+$.

Example

- Let a relation R (A,B,C,D)and a set of FDs **F = { A->B,A->C,C->D}** are given.
- A relation is decomposed into -

  R1 = (A, B, C) with FDs **F1 = {A->B, A->C}**.
  R2 = (C, D) with FDs **F2 = {C->D}**.

  F' = F1 U F2 = {A->B, A->C, C->D}
  So, **F'= F**.
  And so, $\mathbf{F'^{+} = F^{+}}$.

- Thus, the decomposition is dependency preserving decomposition.