**Unit-3**

**Functional Dependency**

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y

X àY holds if whenever **two tuples have the same value for X, they *must have* the same value for Y**

***If t1[X]=t2[X], then t1[Y]=t2[Y]*** in any relation instance r(R)

**Armstrong's Axioms**

* Armstrong's Axioms are a set of rules, generates a closure of functional dependencies. **Closure** of a set F of FDs is the set F+ of all FDs that can be inferred from F
* **Reflexive rule** −

If Y subset-of X, then X à Y

* **Augmentation rule** −

If X à Y, then XZ à YZ

* **Transitivity rule** −

If X à Y and Y à Z, then X à Z

**Additional Useful Inference Rules**

* **Decomposition Rule:-**
  + If X à YZ, then X à Y and X à Z
* **Union Rule:-**
  + If X à Y and X à Z, then X à YZ
* **Psuedotransitivity Rule:-** 
  + If X à Y and WY à Z, then WX à Z

* **Trivial** − If a functional dependency (FD) X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.

Eg. ABàB

* **Non-trivial** − If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD. Eg. ABàBC
* **Completely non-trivial** − If an FD X → Y holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD. Eg. AB
* **Fully F.D.-** If ABàC then AàC/BC must not hold
* **Partial F.D.-** If AB is candidate key and AàC/BC is also there

Normalization

**Normalization** is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.

**Anomalies in DBMS**

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly.

**Example**: Suppose a table named **employee** having four attributes**: emp\_id, emp\_name, emp\_address and emp\_dept.**

|  |  |  |  |
| --- | --- | --- | --- |
| ***emp\_id*** | ***emp\_name*** | ***emp\_address*** | ***emp\_dept*** |
| 101 | Rick | Delhi | D001 |
| 101 | Rick | Delhi | D002 |
| 123 | Maggie | Agra | D890 |
| 166 | Glenn | Chennai | D900 |
| 166 | Glenn | Chennai | D004 |

If a database design is not perfect, it may contain anomalies. Managing a database with anomalies is next to impossible.

**Update anomaly**: If data items are scattered and are not linked to each other properly, then it could lead to strange situations. If we want to update **the address of Rick** then we have to **update the same in two rows or the data will become inconsistent**. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

**Insert anomaly**: We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else. Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp\_dept field doesn’t allow nulls.

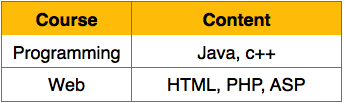
**Delete anomaly**: We tried to insert data in a record that does not exist at all. Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp\_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

**Normalization**

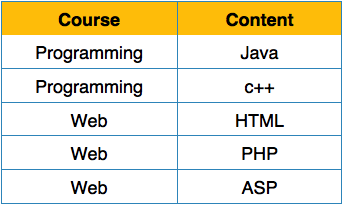
* Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data.
* It divides larger tables to smaller tables and links them using relationships.
* Normalization is used for mainly two purposes,
* Eliminating redundant (useless) data.
* Ensuring data dependencies make sense ie; data is logically stored.

**First Normal Form**

This rule defines that all the attributes in a relation must have **atomic domains**. The values in **an atomic domain are indivisible units**.



to convert it to First Normal Form.



Each attribute must contain only a single value from its pre-defined domain.

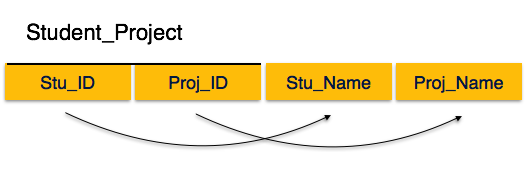
**Second Normal Form**

* **Prime attribute** − An attribute, which is a **part of the candidate-key,** is known as a prime attribute.
* **Non-prime attribute** − An attribute, **which is not a part of the prime-key,** is said to be a non-prime attribute.

*A database is in second normal form if it satisfies the following conditions:*

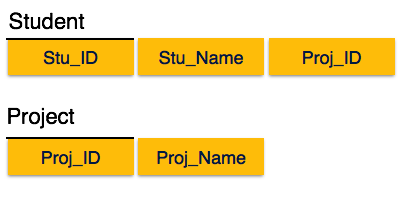
* *It is in first normal form*
* *All non-key attributes are fully functional dependent on the primary key*

That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.



In **Student\_Project** relation that the **prime key attributes are Stu\_ID and Proj\_ID**. **Non-prime attributes, Stu\_Name and Proj\_Name** must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu\_Name can be identified by Stu\_ID and Proj\_Name can be identified by Proj\_ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.

Convert it to Second Normal Form so that there exists no partial dependency.



**Third Normal Form**

For a relation to be in Third Normal Form, it must be in Second Normal form and the no non-prime attribute is transitively dependent on prime key attribute.

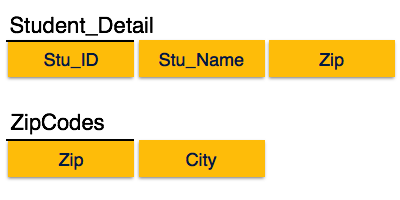
* *For any non-trivial functional dependency, X → A, then either −*
  + *X is a superkey or,*
  + *A is prime attribute.*

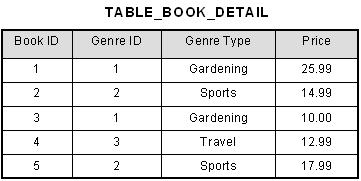


In the **Student\_detail relation,** **Stu\_ID** is the **key and only prime key** attribute. **City** can be **identified by Stu\_ID as well as** Zip itself. **Neither Zip is a superkey nor is City a prime attribute**.

Additionally, Stu\_ID → Zip → City, so there exists **transitive dependency**.

To bring this relation into third normal form break the relation into two relations as follows −

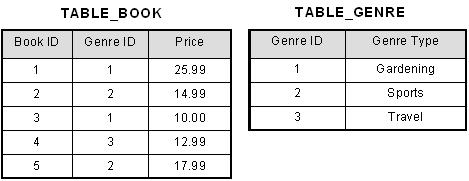




* Here [Book ID] [Genre ID] and [Genre ID]  [Genre Type].

[Book ID]  [Genre ID]  [Genre Type] and we have transitive functional dependency,

* Here [Genre Type] non-prime attribute is transitively dependent on primary key [Book ID]
* To bring this table to third normal form, we split the table into two as follows:







**Boyce-Codd Normal Form**

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. It states that −

* For any non-trivial functional dependency, X → A, X must be a super-key.

*A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F+ of the form*

*a ® b where a Í R and b Í R, at least one of the following holds:*

* *a ® b is trivial (i.e., b Í a)*
* *a is a superkey for R*

**Figure 14.13**

Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF due to the f.d. C → B.

**Fourth Normal Form ( 4NF )**

A relation R is in Fourth Normal Form (4NF) if and only if the following conditions are satisfied simultaneously:

1. R is already in 3NF or BCNF.
2. It contains no multi-valued dependencies.

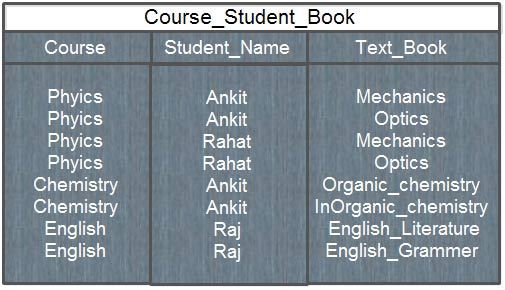
**MVD** can be defined informally as follows:

MVDs occur when two or more independent multi valued facts about the same attribute occur within the same table. It means that if in a relation R having A, B and C as attributes, B and Care multi-value facts about A, which is represented as AB and AC, then multi value dependency exist only if B and C are independent of each other.

There are two things to note about this definition.

Firstly, in order for a table to contain MVD, it must have three or more attributes.

Secondly, it is possible to have a table containing two or more attributes which are interdependent multi valued facts about another attribute.



Here, in above [database](http://ecomputernotes.com/fundamental/what-is-a-database/advantages-and-disadvantages-of-dbms) following MVDs exists:

Course --> --> Student\_name

Course --> --> Text book

Here, Student\_name and Text\_book are independent of each other.

**Anomalies of database with MVDs**

This form of the table is obviously full of anomalies. If a new student joins the physics, we have to make two insertions for that student in the database, which is equal to the number of physics textbooks. Consider the problem if there are hundred textbooks for a subject. Similarly, if a new textbook is introduced for a course, then again we have to make multiple insertions in the database, which is equal to number of students for that course. So, there is a high degree of redundancy in the database, which will lead to update problems.

The relation is also in BCNF, since all three attributes concatenated together constitute its key, yet it is clearly contained anomalies and requires decomposition with the help of fourth normal form.

**Rule to transform a relation into Fourth Normal Form**

A relation R having A, B, and C, as attributes can be non loss-decomposed into two projections R1(A,B) and R2(A,C) if and only if the MVD A  B|C hold in R.

Looking again at the un-decomposed COURSE\_STUDENT\_BOOK table, it contains a multi-valued dependency as shown below:

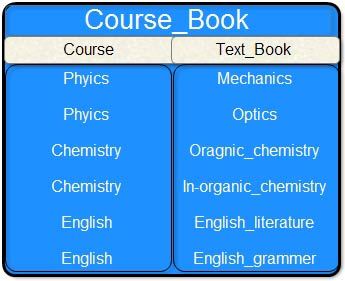
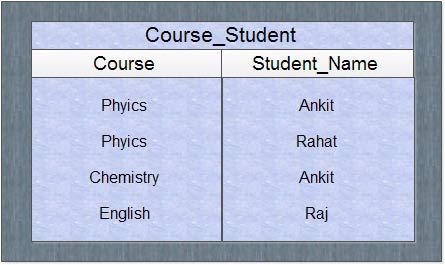
Course  Student\_name

Course Text\_book

To put it into 4NF, two separate tables are formed as shown below:

COURSE\_STUDENT (Course, Student\_name)

COURSE\_BOOK (Course, text\_book)



If now a new student joins a course then we have to make only one insertion in COURSE\_STUDENT table and if a new book introduced for a course then again we have to make a single entry in COURSE\_BOOK table, so this modified database eliminate the problem of redundancy which also solves the update problems.

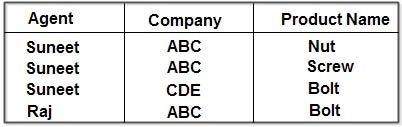
**Fifth Normal Form (5NF)**

A relation R is in Fifth Normal Form (5NF) if and only if the following conditions are satisfied simultaneously:

1. R is already in 4NF.
2. It cannot be further non-loss decomposed.

5NF is of little practical use to the database designer, but it is of interest from a theoretical point of view and a discussion of it is included here to complete the picture of the further normal forms.

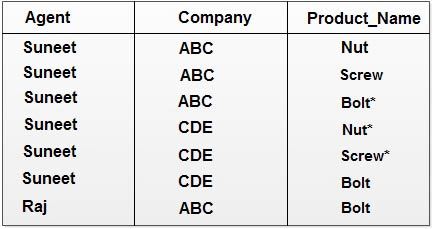
This table lists agents, the companies they work for and the products they sell for those companies.



The table is in 4NF because it contains no multi-valued dependency. It does, however, contain an element of redundancy in that it records the fact that Suneet is an agent for ABC twice. But there is no way of eliminating this redundancy without losing information. Suppose that the table is decomposed into its two projections, PI and P2.

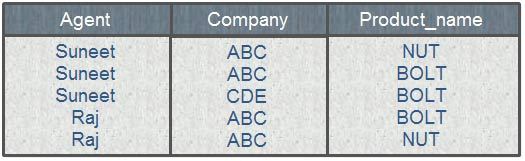


The redundancy has been eliminated, but the information about which companies make which products and which of these products they supply to which agents has been lost. The natural join of these projections over the 'agent' columns is:



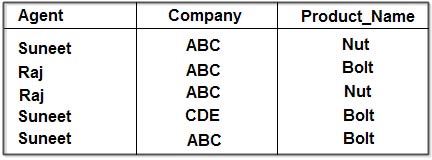
The table resulting from this join is spurious, since the asterisked row of the table contains incorrect information. Now suppose that the original table were to be decomposed into three tables, the two projections, P I and P2 which have already shown, and the final, possible projection, P3.

If a join is taken of all three projections, first of PI and P2 with the (spurious) result shown above, and then of this result with P3 over the 'Company' and 'Product name' column, the following table is obtained:

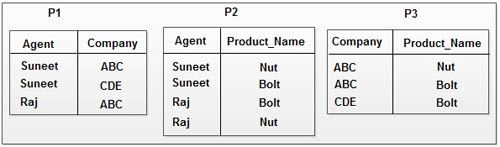


This still contains a spurious row. The order in which the joins are performed makes no difference to the final result. It is not simply possible of decompose the 'AGENT\_COMPANY\_PRODUCT' table, populated as shown, without losing information. Thus, it has to be accepted that it is not possible· to eliminate all redundancies using normalization techniques, because it cannot be assumed that all decompositions will be non-loss.

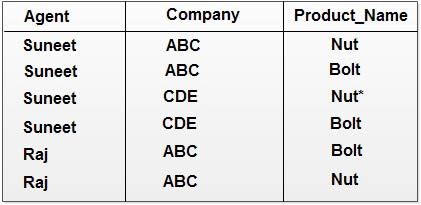
But now consider the different case where, if an agent is an agent for a company and that company makes a product, then he always sells that product for the company. Under these circumstances, the 'agent company product' table as shown below:



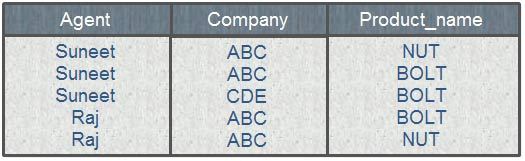
The assumption being that ABC makes both Nuts and Bolts and that CDE makes Bolts only. This table can be decomposed into its three projections without loss of information as demonstrated below:



All redundancy has been removed, if the natural join of PI and P2 IS taken, the result is:



The spurious row as asterisked. Now, if this result is joined with P3 over the column 'company 'product\_name' the following table is obtained:



This is a correct recomposition of the original table and no loss decomposition into the three projections was achieved. Again, the order in which the joins are performed does not affect the final result. The original table, therefore, violated 5NF simply because it was non-loss decomposable into its three projections.

In the first case exemplified above, non-loss decomposition of the 'agent\_company -product' table was not possible. In the second it was. If a table is nonloss decomposable as in the second case, it is said to be in violation of 5NF. The difference, of course, lay in certain semantic properties of the information being represented. These properties were not understandable simply by looking at the table, but had to be supplemented by further information about the relationship between products, agents and companies.

**Lossless And Lossy Decompossition**

Decomposition of a relation is done when a relation in relational model is not in appropriate normal form. Relation R is decomposed into two or more relations if decomposition is lossless join as well as dependency preserving.

**Lossless Join Decomposition**

If we decompose a relation R into relations R1 and R2,

* Decomposition is lossy if R1 ⋈ R2 ⊃ R
* Decomposition is lossless if R1 ⋈ R2 = R

**To check for lossless join decomposition using FD set, following conditions must hold:**

1. Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.

Att(R1) U Att(R2) = Att(R)

1. Intersection of Attributes of R1 and R2 must not be NULL.

Att(R1) ∩ Att(R2) ≠ Φ

1. Common attribute must be a key for at least one relation (R1 or R2)

Att(R1) ∩ Att(R2)  Att(R1) or Att(R1) ∩ Att(R2)  Att(R2)

For Example, A relation R (A, B, C, D) with FD set {A->BC} is decomposed into R1(ABC) and R2(AD) which is a lossless join decomposition as:

1. First condition holds true as Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).
2. Second condition holds true as Att(R1) ∩ Att(R2) = (ABC) ∩ (AD) ≠ Φ
3. Third condition holds true as Att(R1) ∩ Att(R2) = A is a key of R1(ABC) because A->BC is given.

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

The following are the types:

**Lossless Decomposition:**

Decomposition is lossless if it is feasible to reconstruct relation R from decomposed tables using Joins. This is the preferred choice. The information will not lose from the relation when decomposed. The join would result in the same original relation.

Let us see an example:

**<EmpInfo>**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp\_ID** | **Emp\_Name** | **Emp\_Age** | **Emp\_Location** | **Dept\_ID** | **Dept\_Name** |
| E001 | Jacob | 29 | Alabama | Dpt1 | Operations |
| E002 | Henry | 32 | Alabama | Dpt2 | HR |
| E003 | Tom | 22 | Texas | Dpt3 | Finance |

Decompose the above table into two tables:

**<EmpDetails>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp\_ID** | **Emp\_Name** | **Emp\_Age** | **Emp\_Location** |
| E001 | Jacob | 29 | Alabama |
| E002 | Henry | 32 | Alabama |
| E003 | Tom | 22 | Texas |

**<DeptDetails>**

|  |  |  |
| --- | --- | --- |
| **Dept\_ID** | **Emp\_ID** | **Dept\_Name** |
| Dpt1 | E001 | Operations |
| Dpt2 | E002 | HR |
| Dpt3 | E003 | Finance |

Now, Natural Join is applied on the above two tables:

The result will be:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp\_ID** | **Emp\_Name** | **Emp\_Age** | **Emp\_Location** | **Dept\_ID** | **Dept\_Name** |
| E001 | Jacob | 29 | Alabama | Dpt1 | Operations |
| E002 | Henry | 32 | Alabama | Dpt2 | HR |
| E003 | Tom | 22 | Texas | Dpt3 | Finance |

Therefore, the above relation had lossless decomposition i.e. no loss of information.

**Lossy Decomposition:**

As the name suggests, when a relation is decomposed into two or more relational schemas, the loss of information is unavoidable when the original relation is retrieved.

**<EmpInfo>**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp\_ID** | **Emp\_Name** | **Emp\_Age** | **Emp\_Location** | **Dept\_ID** | **Dept\_Name** |
| E001 | Jacob | 29 | Alabama | Dpt1 | Operations |
| E002 | Henry | 32 | Alabama | Dpt2 | HR |
| E003 | Tom | 22 | Texas | Dpt3 | Finance |

Decompose the above table into two tables:**<EmpDetails>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp\_ID** | **Emp\_Name** | **Emp\_Age** | **Emp\_Location** |
| E001 | Jacob | 29 | Alabama |
| E002 | Henry | 32 | Alabama |
| E003 | Tom | 22 | Texas |

**<DeptDetails>**

|  |  |
| --- | --- |
| **Dept\_ID** | **Dept\_Name** |
| Dpt1 | Operations |
| Dpt2 | HR |
| Dpt3 | Finance |

**Canonical cover in DBMS-**

* A canonical cover is a simplified and reduced version of the given set of functional dependencies whose closure is exactly same as that of the closure of given set of functional dependencies.
* A canonical cover is also called as **irreducible set** as it is a reduced version of the given set of functional dependencies and is free from all the unnecessary or extraneous functional dependencies.

**Need of a canonical cover-**

* The original set of functional dependencies may contain unnecessary or  extraneous functional dependencies which are useless and working with them unnecessarily increases the computation time.
* So, we try to reduce the given set of functional dependencies by eliminating the unnecessary functional dependencies which are not required so that working with them become easier and computation time is reduced.

**NOTE-** canonical cover is not at all unique and for a given set of functional dependencies, we may have more than one canonical cover.

**Steps to find canonical cover in DBMS-**

**Step-01:** Write the given set of functional dependencies in such a way that each functional dependency contains exactly one attribute on its right side.

**For example-** For a given functional dependency X → YZ, we will write it as-

X → Y , X → Z

**Step-02:** Now, consider each functional dependency one by one from the set obtained after performing step-01 and determine whether it is essential or non-essential.

To determine so, take the functional dependency and compute the closure of its left side according to the following cases-

**Case-01:** Compute the closure of left side by considering that particular functional dependency is present.

**Case-02:** Compute the closure of left side by considering that particular functional dependency is absent.

**If the results come out to be same,**

It means that the presence or absence of that functional dependency does not create any difference and it is non-essential.

So, Eliminate that functional dependency from our set of functional dependencies.

( Eliminate the functional dependency from the given set as soon as it is discovered that the functional dependency is non-essential and don’t consider it when you check the essentiality of other functional dependencies.)

**If the results come out to be different,**

It means that the absence of that functional dependency creates a difference and it is essential.

So, Do not eliminate that functional dependency from our set of functional dependencies and mark that particular functional dependency as essential.

Repeat this step until all the functional dependencies are checked.

**Step-03:** Now, consider the newly obtained set of functional dependencies after performing step-02 and check if there is any functional dependency that has more than one attribute on its left side.

* If no, then this newly obtained set of functional dependencies is the canonical cover.
* If yes, then take such functional dependencies under consideration and check if their left side can be reduced.

This is checked by computing the closure of all the possible subsets of the left side of each such functional dependency. If any of its subset produces the same closure result as produced by the entire left side of functional dependency, then replace the left side of that functional dependency with its subset producing the same result.

Repeat this for all such functional dependencies.

After it’s done, the set so obtained will be the canonical cover for the given set of functional dependencies.

**PRACTICE PROBLEM BASED ON FINDING THE CANONICAL COVER-**

The following functional dependencies hold true for the relational scheme R ( W , X , Y , Z ) –

X → W WZ → XY Y → WXZ

Write the irreducible equivalent for this set of functional dependencies.

**Solution-**

**Step-01:** First, we will write the given set of functional dependencies in such a way that each functional dependency contains exactly one attribute on its right side as shown-

X → W WZ → X WZ → YY → WY → XY → Z

**Step-02:**

Now, we will check the essentiality of each functional dependency one by one.

**For X → W:**

* Considering X → W, we have (X)+ = { X , W }
* Ignoring X → W, we have (X)+ = { X }

Since, the two results are different, so X → W is essential and can not be eliminated.

**For WZ → X:**

* Considering WZ → X, we have (WZ)+ = { W , X , Y , Z }
* Ignoring WZ → X, we have (WZ)+ = { W , X , Y , Z }

Since, the two results are same, so WZ → X is non-essential and can be eliminated.

So, eliminating WZ → X, our set of functional dependencies updates to-

X → W WZ → Y Y → W Y → X Y → Z

**For WZ → Y:**

* Considering WZ → Y, we have (WZ)+ = { W , X , Y , Z }
* Ignoring WZ → Y, we have (WZ)+ = { W , Z }

Since, the two results are different, so WZ → Y is essential and can not be eliminated.

**For Y → W:**

* Considering Y → W, we have (Y)+ = { W , X , Y , Z }
* Ignoring Y → W, we have (Y)+ = { W , X , Y , Z }

Since, the two results are same, so Y → W is non-essential and can be eliminated.

So, eliminating Y → W, our set of functional dependencies updates to-

X → W WZ → Y Y → X Y → Z

**For Y → X:**

* Considering Y → X, we have (Y)+ = { W , X , Y , Z }
* Ignoring Y → X, we have (Y)+ = { Y , Z }

Since, the two results are different, so Y → X is essential and can not be eliminated.

**For Y → Z:**

* Considering Y → Z, we have (Y)+ = { W , X , Y , Z }
* Ignoring Y → Z, we have (Y)+ = { W , X , Y }

Since, the two results are different, so Y → Z is essential and can not be eliminated.

Thus,

Our essential functional dependencies are-

X → W WZ → Y Y → X Y → Z

**Step-03:** Now, we will check examine those functional dependencies having more than one attribute on their left side and check if their left side can be reduced.

In our set of essential functional dependencies, only WZ → Y has two attributes on its left side and rest all other functional dependencies have only single attribute on their left side.

Now, closure of WZ is-

(WZ)+ = { W , X , Y , Z }

Now, consider all the possible subsets of WZ and check if their closure result matches with the closure result of WZ.

* (W)+ = { W }
* (Z)+ = { Z }

Since, none of the subset gives the same result, so we can not write WZ → Y as W → Y or Z → Y.

So, set of functional dependencies obtained in step-02 is the irreducible set or canonical cover.

Thus,

X → W WZ → Y Y → X Y → Z (**Irreducible set / Canonical Cover)**