DBMS: Introduction + ER Diagram + Functional Dependency

<u>Restaurant Case Study</u>: Let's say we own a restaurant and we receive orders from multiple customers every day that we can deliver to the customer using the information of their **name, contact number,** and **address**. Also, we plan to send their bills to their **email** address for customer reference.
We plan to **store this data** of the customer so that every time we receive the data from the same customer we don't ask for the same information.

Que: Is the use case of storing data clear?

In past (when technology was not used much) this data was stored in **registers/notebooks** which involved some inconveniences like:

1. Data is stored in an unorganized way so searching for the customer data was a manual process.
2. If the number of customers is high then maintaining multiple records was a pain.
3. If any customer changes his/her address of delivery then updating that in the record was a task that might not be possible due to less space in the page containing the data. Creating new data and deleting old records could be an option. And so on…

Que: Is the need of having a Database Management System (DBMS) clear?

Database: The database is a systematic collection of interrelated data which is used to:

a. Retrieve the data,
b. Insert the data, and
c. Delete the data efficiently.

Let's understand **Entity, Attributes** & **Relationships** using the Uber System:

1. **Entity:** Core identities wrt which the data needs to be stored. Eg (In Uber): users, drivers, trips, etc.
2. **Attributes:** Data of each of the entities. Eg (In Uber):
   a. Name, contact number, email id, etc. - of **Users**
   b. Name, contact number, vehicle number, etc. - of **Drivers**
   c. Start time, End Time, Start Location, Drop Location, etc. - of **Trips** and …
3. **Relationship** between the entities. Eg (In Uber): Driver drives a Cab. Here *drives* is a relationship between Driver and Cab.

Kishan | 9---- | KA ---
Kishan | 9-- | MR —

Types of Attributes

Simple Attribute: Attributes that are atomic in nature and cannot be divided any further. Eg: Contact Number, Age, etc.	**Composite Attribute:** Attributes that are not atomic and can be divided. Eg: Name of a person can be divided into First Name, Middle Name & Last Name, etc.
Derived Attribute: Attributes that are not stored in the database but their value can be derived from the other attributes that are stored in the database. Eg: The age of a person can be derived from the DOB.	**Multivalued Attribute:** Attributes that have more than one value. Eg: a user can have more than one contact number or email id, therefore contact number & email id are multivalued attributes.

Each entity can be stored like a **table** where each row will be one unique entity and columns will be attributes of that entity. Eg; User Table can have each row storing the data of a user and columns can be user name, contact number, email id, etc.

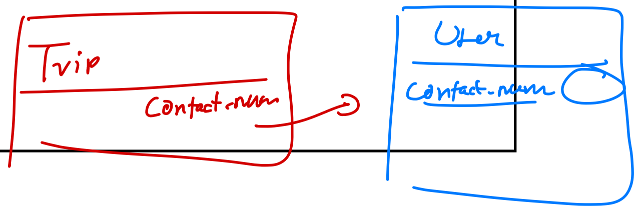
user_id	**name**	**contact_number**	**dob**	**email**
1234	Utkarsh	9876598765	07-10-2010	utkarsh@gmail.com
3456	Karan	9753186420	22-11-2010	karan@gmail.com

Keys

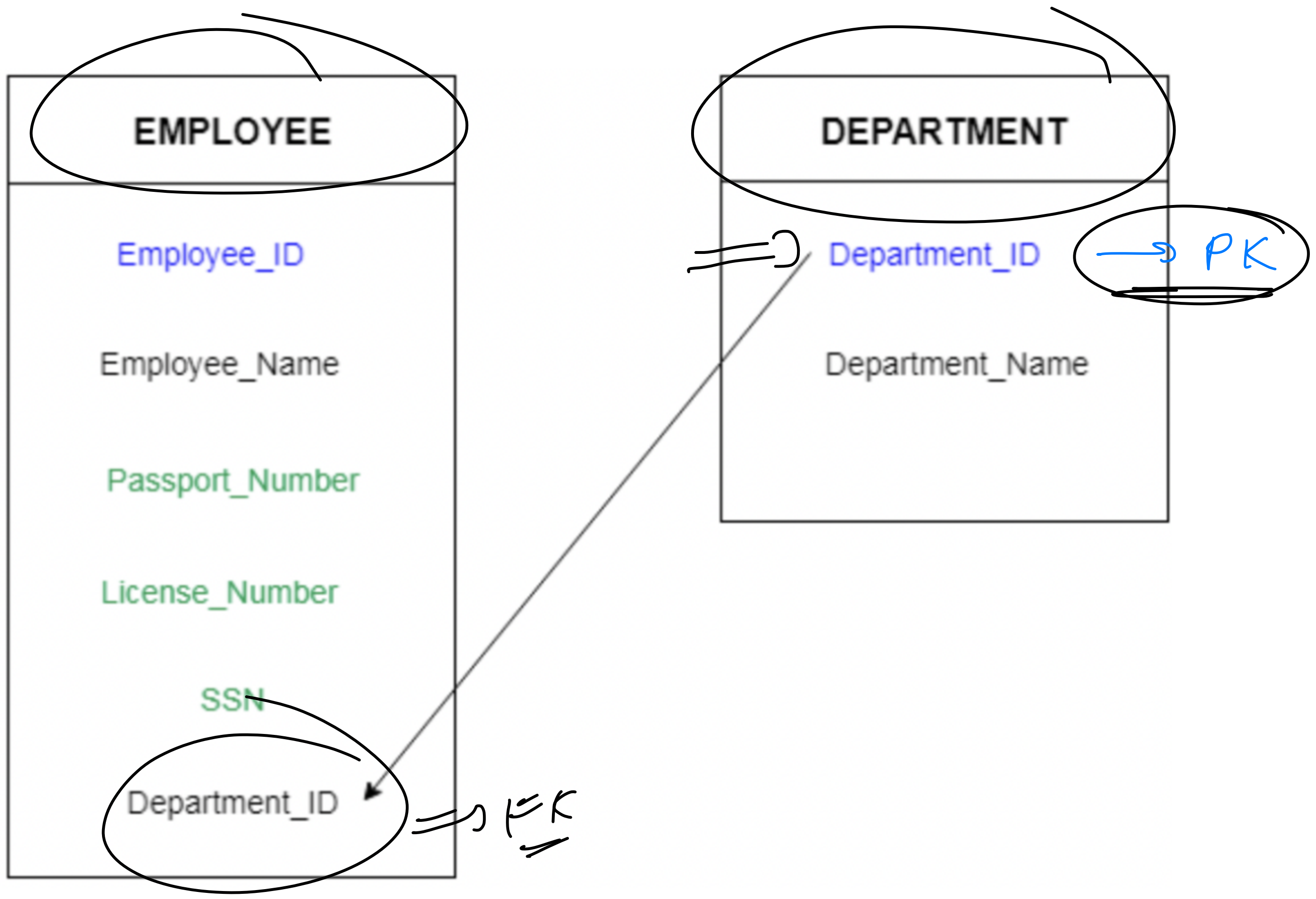
Super Key: An attribute or set of attributes that uniquely identifies a tuple/row within a table. Eg: a. Name & vehicle number can be used to uniquely identify a Driver. b. User name, contact number, and DOB can be used to uniquely identify a User.	**Candidate Key:** A super key such that no proper subset is a super key within the relation. Eg: c. Only vehicle numbers can be used to uniquely identify a Driver. d. Only contact numbers can be used to uniquely identify a User. e. User contact number and vehicle number can be used to uniquely identify a Trip.

SK → CK → PK

Primary Key: A candidate key that will be used as a standard to uniquely identify a tuple/row within a relation. The unselected candidate keys are known as **Alternate Keys.** Eg: Let's say a user contact number is selected as a standard in the Uber system to identify the user, then the contact number is the primary key. But user email id can also be used to identify a user uniquely therefore email id is an alternate Key.	**Foreign Key:** Attributes of one table that are used to point Primary Key of another table. Eg: Trip booking contact number can be a Foreign key in the Trips table but can be Primary Key in the User table. (the user who booked the trip can be uniquely identified with the contact number from which the trip was booked.)



Eg of **Foreign Key:**



Cardinality of a Relationship:

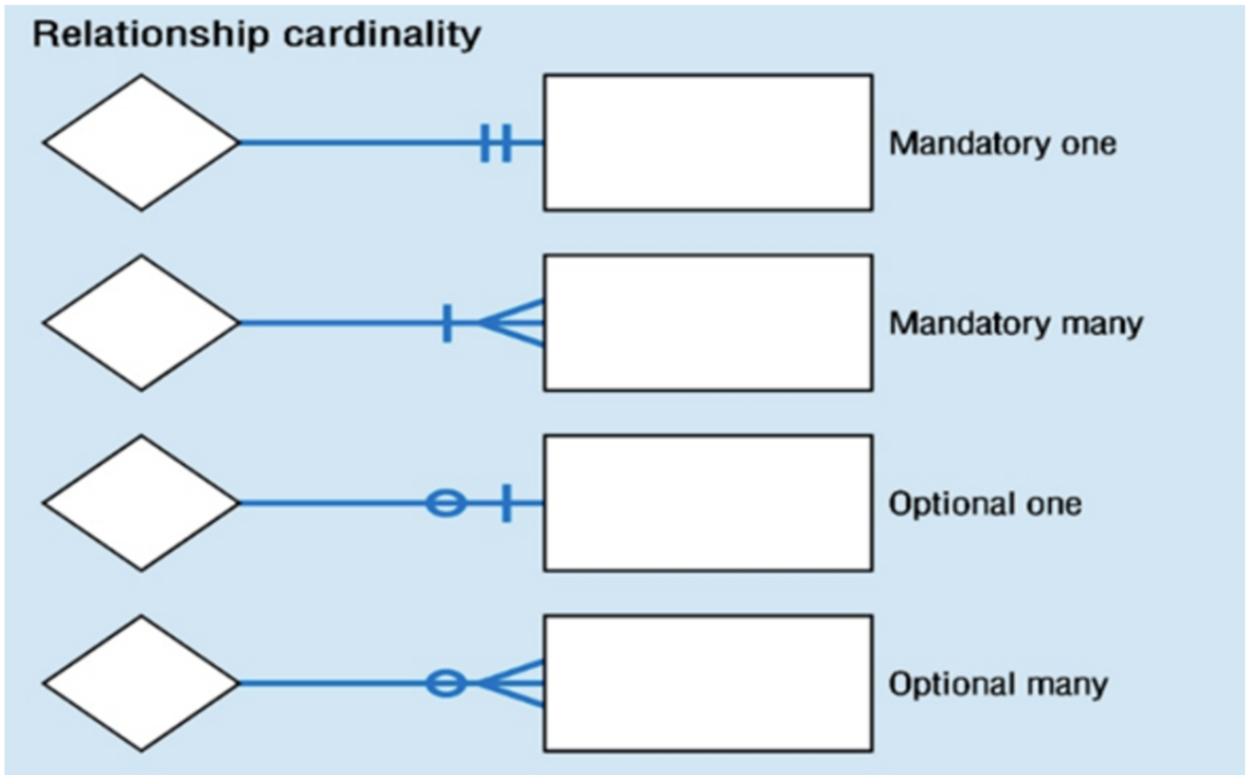
1. **One - to - One:**
   Eg: Driver - to - Cab → one driver can only drive one cab (i.e. registered to this driver) and one cab can only be driven by one driver.
2. **One - to - Many:**
   Eg: Driver - to - Trip → one driver can take multiple trips but one trip can only be driven by one driver.
3. **Many - to - One:**
   Eg: Trip - to - Driver → One trip can only be driven by one driver but one driver can take multiple trips.
4. **Many - to - Many:**
   Eg: User - to - Driver → Multiple users can ride with multiple drivers and multiple drivers can give rides to multiple users.

Entity Relationship Diagram / Model

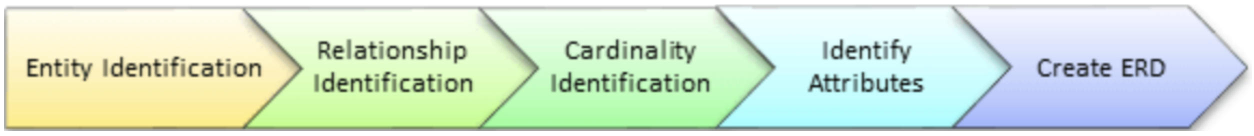
1. Before storing any data one should have a clear picture of what needs to be stored and how to organize it well for faster retrieval and update.
2. ER Diagram is the **first step** of designing any database.
3. It's a diagram that shows all the entities with their attributes and the relationship between the entities.

Symbols in ER Diagram:

1. Rectangle - Entity
2. Ellipse - Attribute
3. Double Ellipse - Multivalued Attribute
4. Underlined Ellipse - Primary Key Attribute
5. Diamond - Relationship
6. Lines - Link between: entities & attributes, entities & relationships, and relationship & attributes.

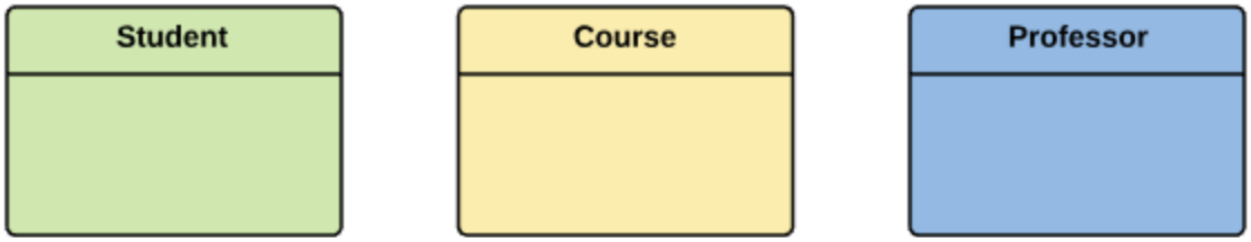


Steps to create ER Diagram:



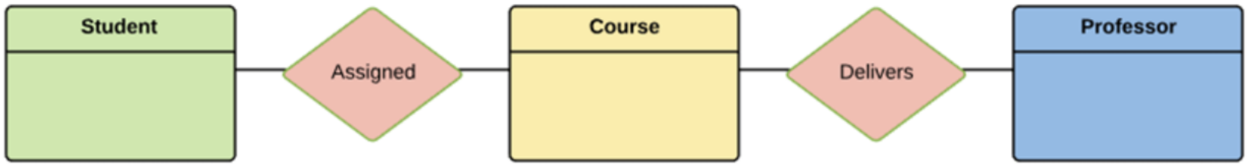
Problem Statement: Design a ER Diagram of a university where a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course. (Feel free to assume attributes of entities on your own.)

STEP 1: **Entity Identification**: Student, Course & Professor.



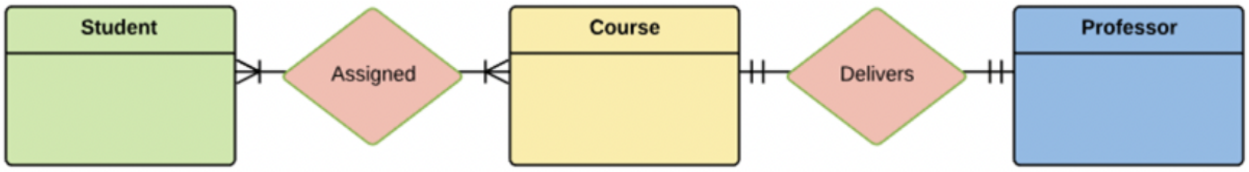
STEP 2: **Relationship Identification**:

a. Student is assigned a course
b. Professor delivers a course



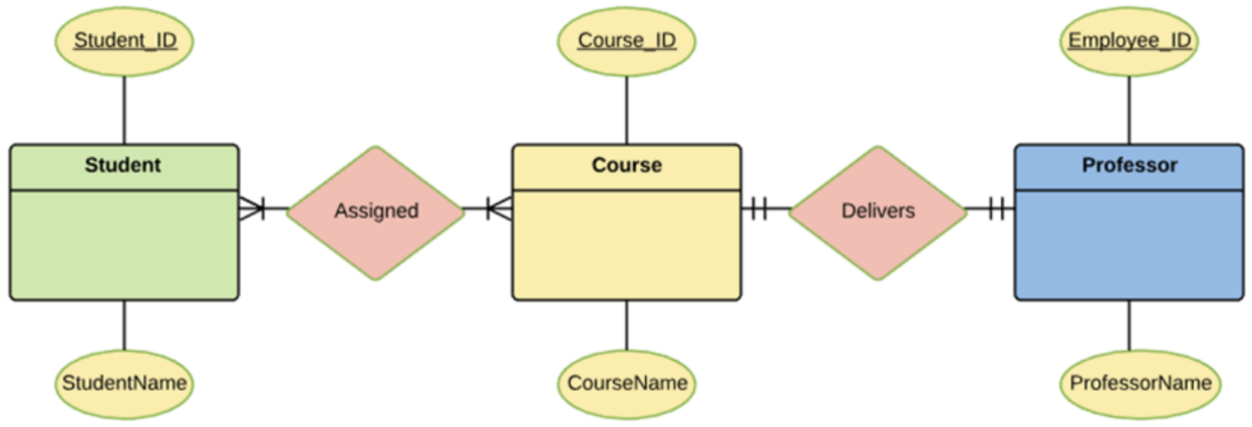
STEP 3: **Cardinality Identification:**

a. A student can be assigned multiple courses & a course can have multiple students.
b. A professor can only deliver one course & a course can only be delivered by one professor.

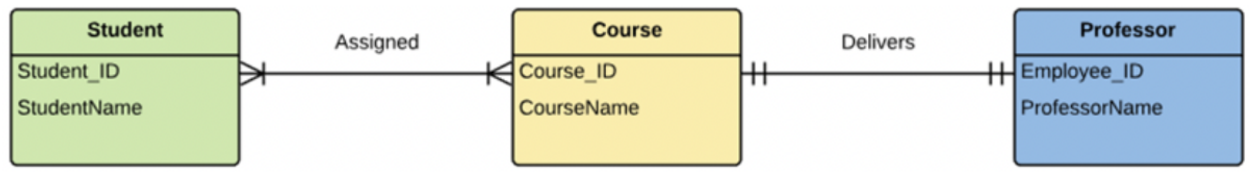


STEP 4: **Identify Attributes:**

a. Student: Student_ID, StudentName, etc.
b. Course: Course_ID, CourseName, etc.
c. Professor: Employee_ID, ProfessorName, etc.

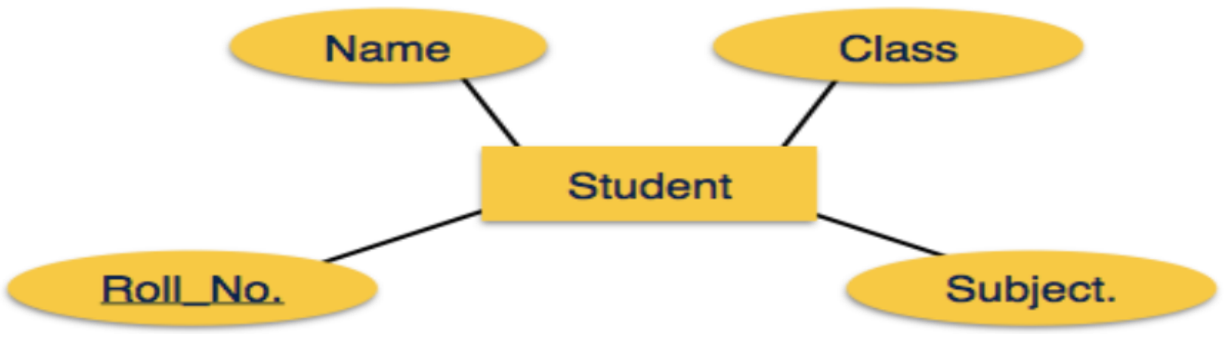


A modern way of representing the same ER Diagram is

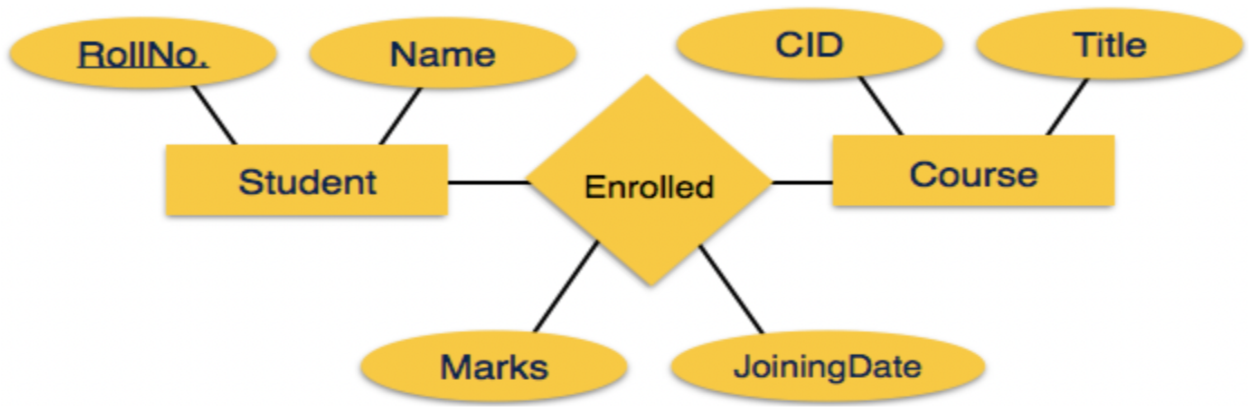


Mapping Attributes:

1. **Mapping on Entities:**



2. **Mapping on Relationships:**



<u>Functional Dependency</u>

Functional Dependency is a set of constraints between two attributes in a relation. Functional Dependency is represented by an arrow sign (→) i.e. X → Y, where X functionally determines Y. If the value of X is known we can uniquely identify value of Y. Here, X (left side) is **determinant** and Y (right side) is **dependent**.

Eg:

1. **Contact_Number → User_Name**, if we know the contact number of a user than we can uniquely identify the user name.
2. **User_Id → User_Name, Contact_Number, Email_Id** - if we know the user id of a user than we can uniquely identify the user name, contact number & email id.
3. **Pincode → City,** etc.

<u>Armstrong's Axioms</u>: If **F** is a set of functional dependencies then the closure of F, denoted as **F^{+}**, is the set of all functional dependencies logically implied by **F**. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

Eg: **F:** X → Y, X → Z, Z → A

This also implies that:

a. X → Y, Z
b. X → Y, Z, A
c. X → X
d. X → A
e. Z → Z, A… and so on.

<u>Rules for Functional Dependencies</u>:

1. **Reflexive Rule:** If **X1** is a subset of **X** then **X → X1.**
   Eg: X = {User_Name, Contact_Number, User_Email}
   X → Contact_Number (X1)
   X → User_Name, Contact_Number (X1) ... and so on.

2. **Augmentation Rule[redacted]:** If **X** → **Y** then **XZ** → **YZ** for any **Z.**
   Eg: X = {User_Id},   Y = {User_Name},   Z = {Contact_Number}
   If User_Id → User_Name then
   User_Id, Contact_Number → User_Name, Contact_Number

3. **Transitive Rule:** If **X → Y** & **Y → Z** then **X → Z.**
   Eg: X = {User_Id}, Y = {Contact_Number}, Z = {User_Name}
   If User_Id → Contact_Number & Contact_Number → User_Name then
   User_Id → User_Name.

4. **Union Rule:** If **X → Y** and **X → Z** then **X → Y, Z.**
   Eg: X = {User_Id}, Y = {Contact_Number}, Z = {User_Name}
   If User_Id → Contact_Number & User_Id → User_Name then
   User_Id → Contact_Name, User_Name.

5. **Decomposition Rule / Project Rule:** (Reverse of Union Rule).
   If **X → Y, Z** then **X → Y** and **X → Z.**
   Eg: X = {User_Id}, Y = {Contact_Number}, Z = {User_Name}
   If User_Id → Contact_Number, User_Name then
   User_Id → Contact_Number & User_Id → User_Name.

6. **Pseudo Transitive Rule:** If **X → Y** and **YZ → W** then **XZ → W.**
   Eg: X = {User_Id}, Y = {Contact_Number}, Z = {User_Name}, W = {User_Email}
   If User_Id → Contact_Number and
   Contact_Number, User_Name → User_Email then
   User_Id, User_Name → User_Email.

<u>**Types of Functional Dependencies:**</u>

Trivial: If in X → Y, Y is a subset of X then it's called a trivial functional dependency. Eg: User_Id, User_Name → User_Name.	**Non-Trivial:** If in X → Y, Y is not a subset of X then it's called a non-trivial functional dependency. Eg: User_Id, User_Name → User_Name, User_Email	**Completely Non-Trivial:** If in X → Y, X intersection Y = {} (empty) then it's called completely non-trivial functional dependency. Eg: User_Id → User_Email, User_Name