Course: IT13 Advanced DBMS
Chapter 1: Introduction to Relational Databases Language

DBMS Data Models

Data model tells how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in DBMS. Data models define how data is connected to each other and how it will be processed and stored inside the system.

The very first data model could be flat data-models where all the data used to be kept in same plane. Because earlier data models were not so scientific they were prone to introduce lots of duplication and update anomalies.

DBMS Data Models

Data model tells how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in DBMS. Data models define how data is connected to each other and how it will be processed and stored inside the system. The structure of a database is the data model; a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints . A data model provides a way to describe the design of a database at the physical, logical and view level.

The data models can be classified in four different categories:

- **Relational Model or Record based:** The relational model uses a collection of tables to represent both data and the relationship among those data. A relational model is an example of a record based model. Record based models are so named because the database is structured in fixed format records of several types.
- **The Entity –Relationship Model**: The ER data model is based on a perception of a real world that consist of a collection of of basic objects, called entities, and of relationships among these objects
- **Object Based Data Model**: The OO data model is another data model that has seen increasing attention. The OO model can be seen as extending the ER model with notions of encapsulation, methods and object identity.
- **Semi structured Data Model**: it permits the specification of data where individual data items of the same type may have different sets of attributes
- The **network data model** and the **hierarchical data model** preceded the relational data model. These models were tied closely to the underlying implementation and complicated the task of modeling data.

Entity-Relationship Model

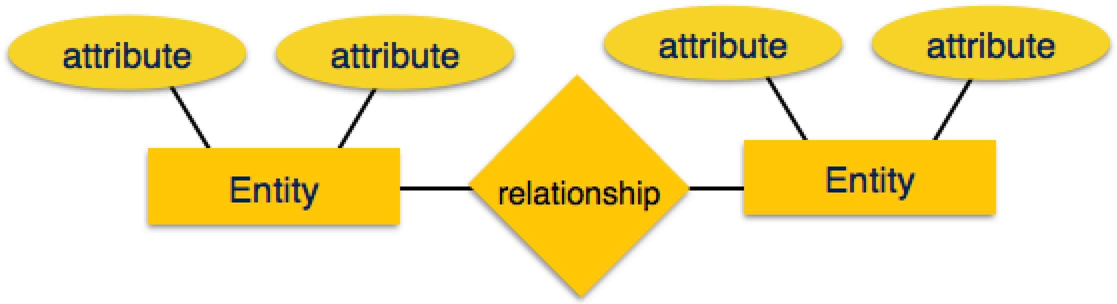
Entity-Relationship model is based on the notion of real world entities and relationship among them. While formulating real-world scenario into database model, ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of database.

ER Model is based on:

- **Entities** and their *attributes*
- **Relationships** among entities

  These concepts are explained below.



[*Image: ER Model*]

- **Entity**

  An entity in ER Model is real world entity, which has some properties called ***attributes***. Every attribute is defined by its set of values, called ***domain***.

  For example, in a school database, a student is considered as an entity. Student has various attributes like name, age and class etc.

- **Relationship**

  The logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

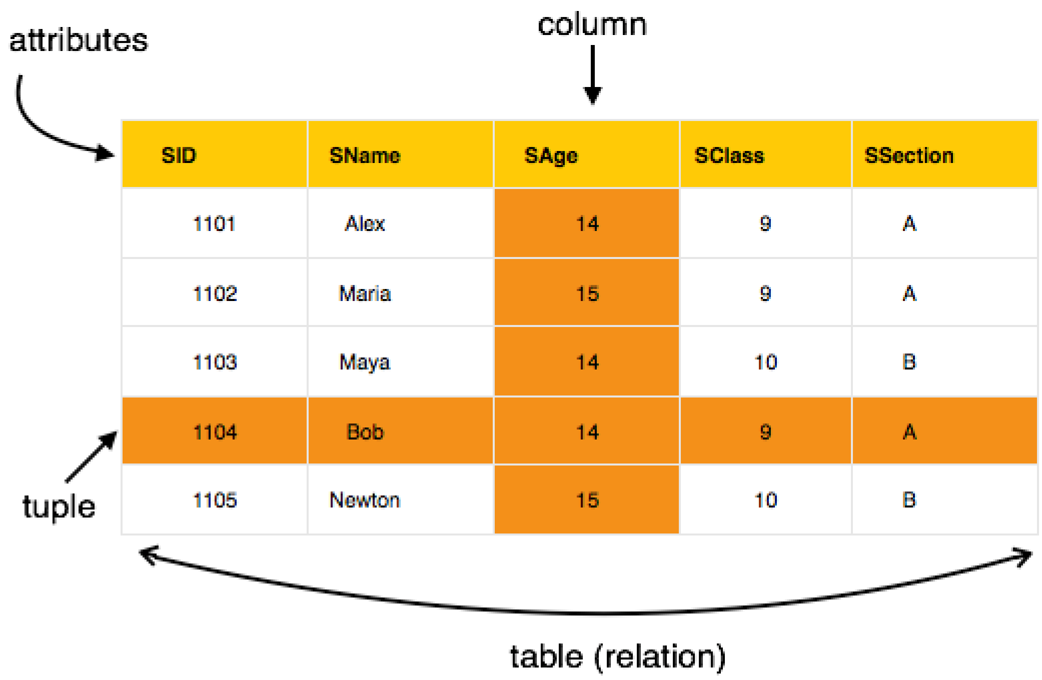
  Mapping cardinalities:

- one to one
- one to many
- many to one
- many to many

ER-Model is explained here.

Relational Model

The most popular data model in DBMS is Relational Model. It is more scientific model then others. This model is based on first-order predicate logic and defines table as an n-ary relation.



SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

[*Image: Table in relational Model*]

The main highlights of this model are:

- Data is stored in tables called relations.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in relation contains unique value
- Each column in relation contains values from a same domain.

ER Model: Basic Concepts

Entity relationship model defines the conceptual view of database. It works around real world entity and association among them. At view level, ER model is considered well for designing databases.

Entity

A real-world thing either animate or inanimate that can be easily identifiable and distinguishable. For example, in a school database, student, teachers, class and course offered can be considered as entities. All entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. Entity set may contain entities with attribute sharing similar values. For example, Students set may contain all the student of a school; likewise Teachers set may contain all the teachers of school from all faculties. Entities sets need not to be disjoint.

Attributes

Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, age as attributes.

There exist a domains or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of attributes:

- **Simple attribute:**

  Simple attributes are atomic values, which cannot be divided further. For example, student's phone-number is an atomic value of 10 digits.

- **Composite attribute:**

  Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.

- **Derived attribute:**

  Derived attributes are attributes, which do not exist physical in the database, but there values are derived from other attributes presented in the database. For example, average_salary in a department should be saved in database instead it can be derived. For another example, age can be derived from data_of_birth.

- **Single-valued attribute:**

  Single valued attributes contain on single value. For example: Social_Security_Number.

- **Multi-value attribute:**

  Multi-value attribute may contain more than one values. For example, a person can have more than one phone numbers, email_addresses etc.

These attribute types can come together in a way like:

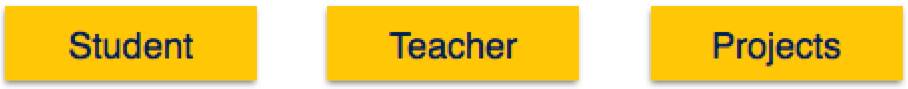
- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes
- composite multi-valued attributes

ER Diagram Representation

Now we shall learn how ER Model is represented by means of ER diagram. Every object like entity attributes of an entity, relationship set, and attributes of relationship set can be represented by tools of ER diagram.

Entity

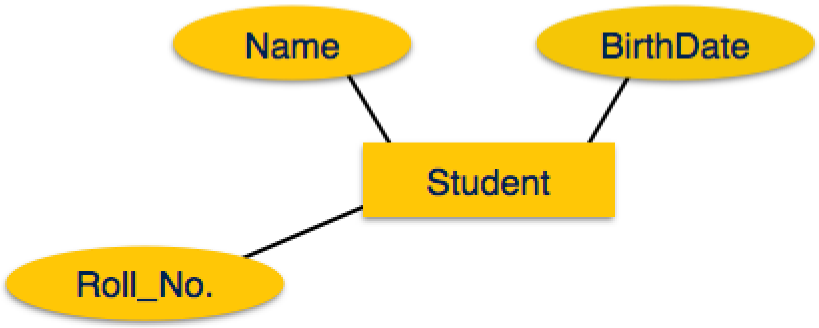
Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.



[*Image: Entities in a school database*]

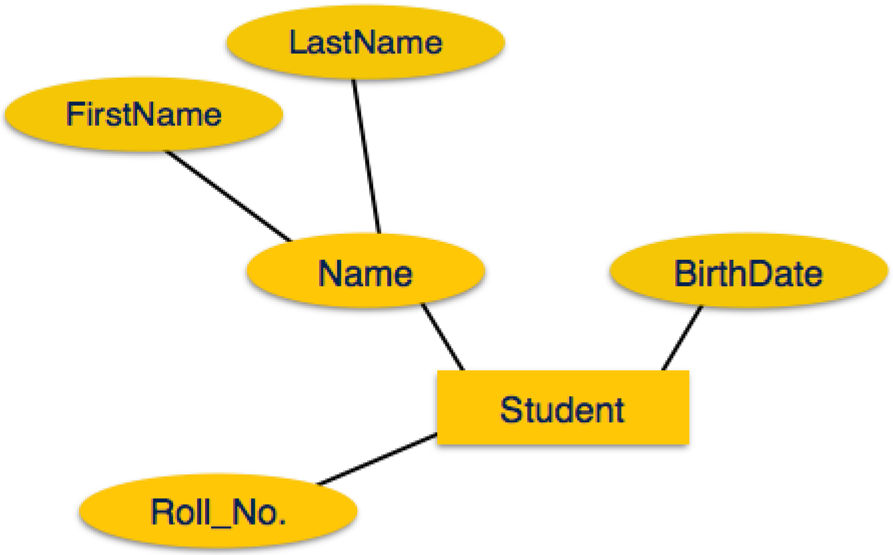
Attributes

Attributes are properties of entities. Attributes are represented by means of eclipses. Every eclipse represents one attribute and is directly connected to its entity (rectangle).



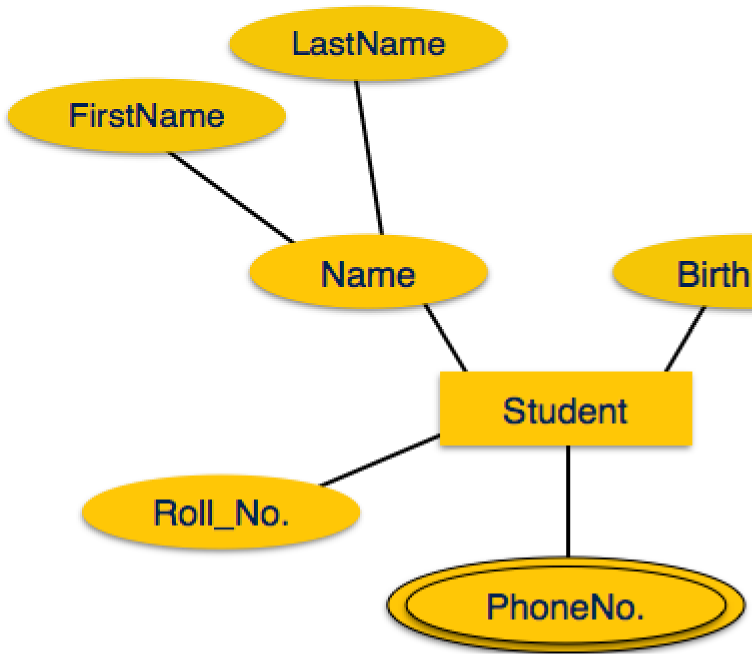
[*Image: Simple Attributes*]

If the attributes are ***composite***, they are further divided in a tree like structure. Every node is then connected to its attribute. That is composite attributes are represented by eclipses that are connected with an eclipse.



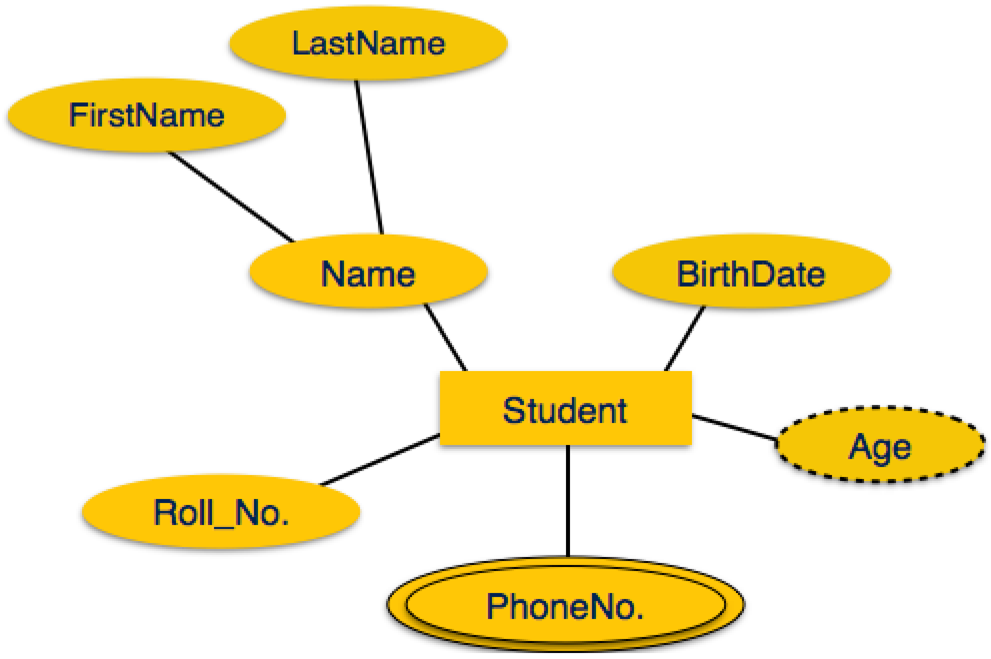
[*Image: Composite Attributes*]

Multivalued attributes are depicted by double eclipse.



[*Image: Multivalued Attributes*]

Derived attributes are depicted by dashed eclipse.



[*Image: Derived Attributes*]

Entity-set and Keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, roll_number of a student makes her/him identifiable among students.

Super Key: Set of attributes (one or more) that collectively identifies an entity in an entity set

For example, the *customer_id* attribute of the entity set customer is sufficient to distinguish one customer entity from another. Thus *customer_id* is a superkey. Similarly the combination of *customer_name* & *customer_id* is a superkey for the entity set customer. The *customer_name* attribute of customer is not a super key, bcz several people might have the same name.

Candidate Key: Minimal super key is called candidate key that is, supers keys for which no proper subset are a super key. An entity set may have more than one candidate key.

For example the combination of *customer_name* and c*ustomer_street* is sufficient to distinguish among members of customer entity set. Then, both{*customer_id}* and *{customer_name, customer_street}* are candidate keys. Although the attributes *customer_id* & *customer_name* together can distinguish customer entities, their combination does not form a candidate key, since the attribute *customer_id* alone is a candidate key.

Primary Key: This is one of the candidate key chosen by the database designer to uniquely identify the entity set. The primary key should be chosen such that its attributes are never or very rarely, changed

Relationship

The association among entities is called relationship. For example, employee entity has relation works_at with department. Another example is for student who enrolls in some course. Here, Works_at and Enrolls are called relationship.

Relationship Set:

Relationship of similar type is called relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

Degree of relationship

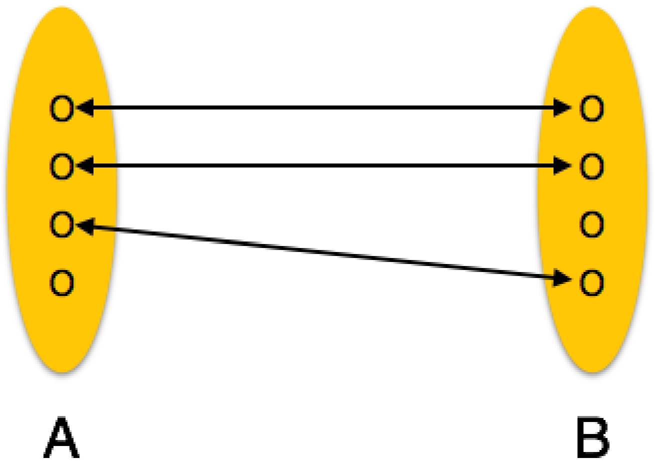
The number of participating entities in an relationship defines the degree of the relationship.

- Binary = degree 2
- Ternary = degree 3
- n-ary = degree

Mapping Cardinalities:

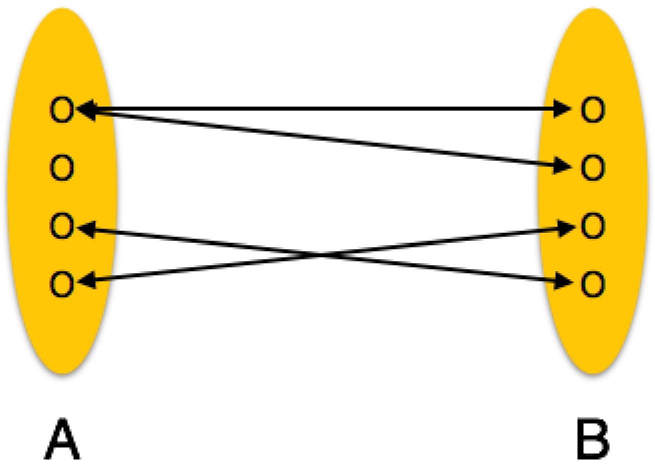
Cardinality defines the number of entities in one entity set which can be associated to the number of entities of other set via relationship set.

- **One-to-one:** one entity from entity set A can be associated with at most one entity of entity set B and vice versa.



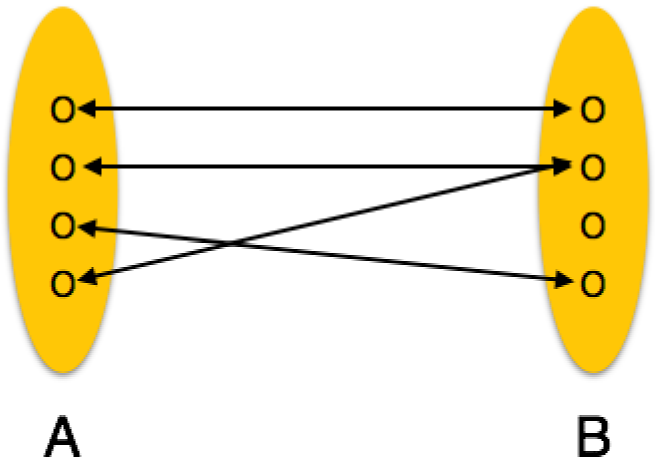
[*Image: One-to-one relation*]

- **One-to-many:** One entity from entity set A can be associated with more than one entities of entity set B but from entity set B one entity can be associated with at most one entity.



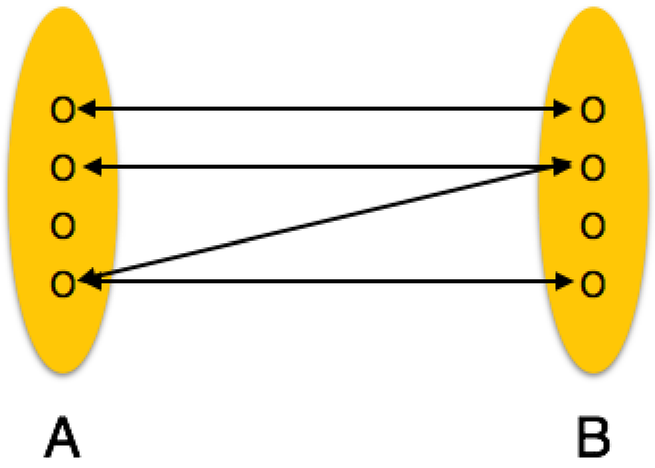
[*Image: One-to-many relation*]

- **Many-to-one:** More than one entities from entity set A can be associated with at most one entity of entity set B but one entity from entity set B can be associated with more than one entity from entity set A.



[*Image: Many-to-one relation*]

- **Many-to-many:** one entity from A can be associated with more than one entity from B and vice versa.



[*Image: Many-to-many relation*]

Relationship

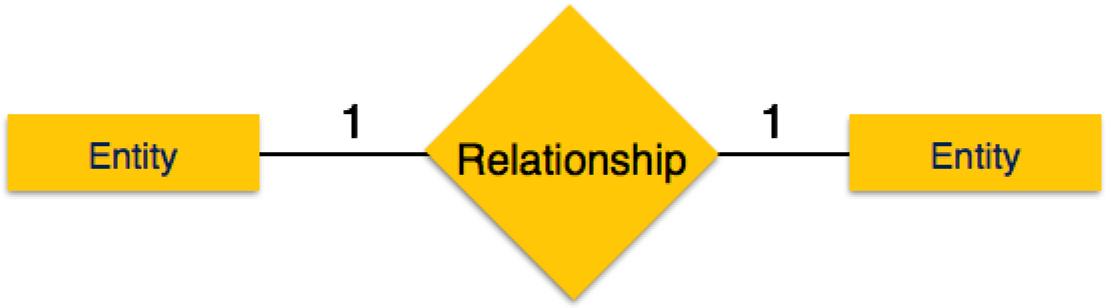
Relationships are represented by diamond shaped box. Name of the relationship is written in the diamond-box. All entities (rectangles), participating in relationship, are connected to it by a line.

Binary relationship and cardinality

A relationship where two entities are participating, is called a ***binary relationship***. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

- **One-to-one**

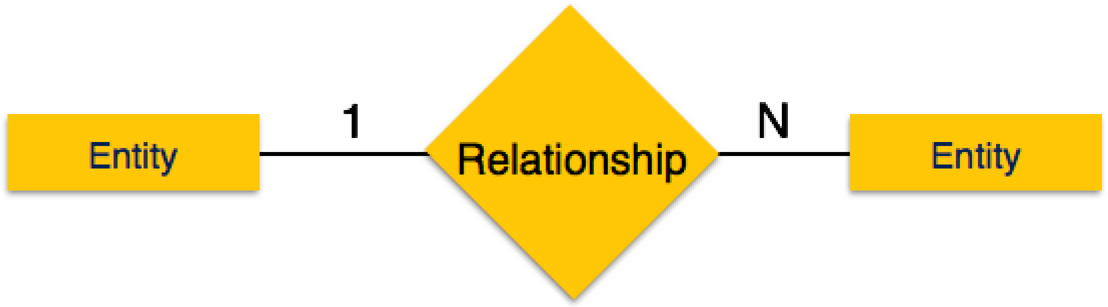
  When only one instance of entity is associated with the relationship, it is marked as '1'. This image below reflects that only 1 instance of each entity should be associated with the relationship. It depicts one-to-one relationship



[*Image: One-to-one*]

- **One-to-many**

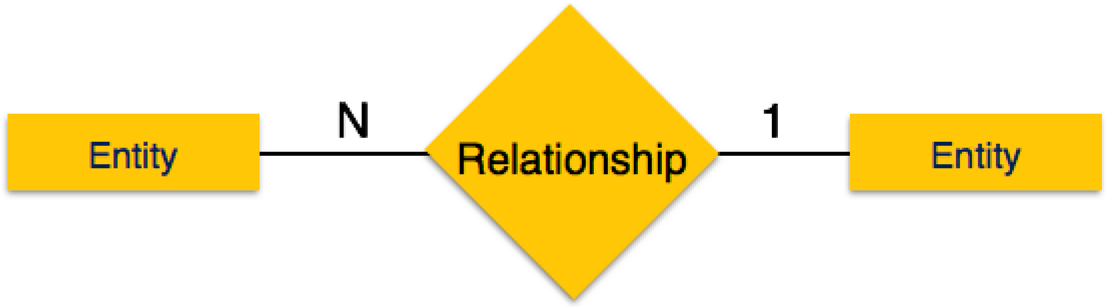
  When more than one instance of entity is associated with the relationship, it is marked as 'N'. This image below reflects that only 1 instance of entity on the left and more than one instance of entity on the right can be associated with the relationship. It depicts one-to-many relationship



[*Image: One-to-many*]

- **Many-to-one**

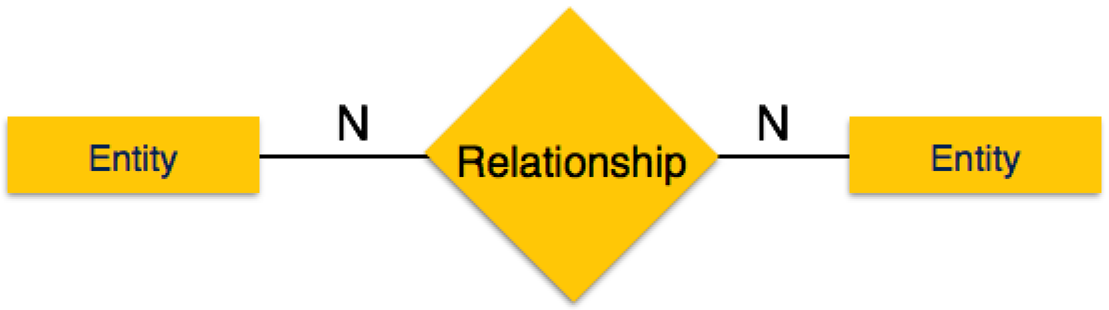
  When more than one instance of entity is associated with the relationship, it is marked as 'N'. This image below reflects that more than one instance of entity on the left and only one instance of entity on the right can be associated with the relationship. It depicts many-to-one relationship



[*Image: Many-to-one*]

- **Many-to-many**

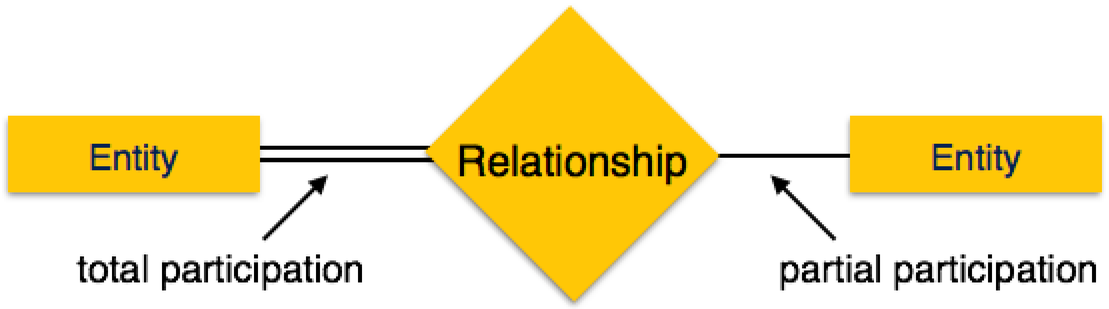
  This image below reflects that more than one instance of entity on the left and more than one instance of entity on the right can be associated with the relationship. It depicts many-to-many relationship



[*Image: Many-to-many*]

Participation Constraints

- **Total Participation:** Each entity in the entity is involved in the relationship. Total participation is represented by double lines.
- **Partial participation:** Not all entities are involved in the relation ship. Partial participation is represented by single line.



[*Image: Participation Constraints*]

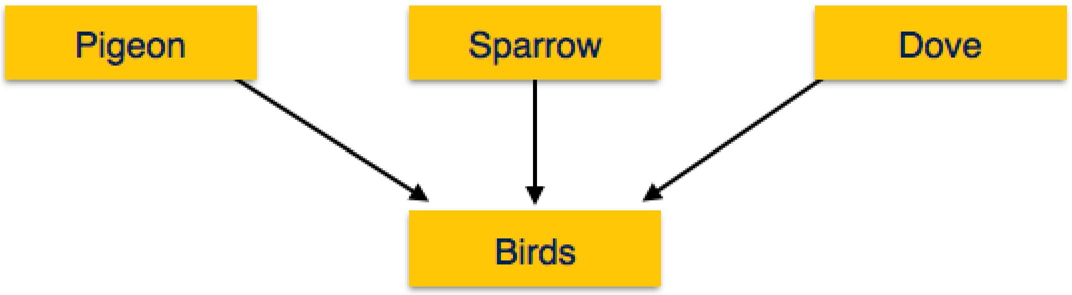
Generalization Aggregation

ER Model has the power of expressing database entities in conceptual hierarchical manner such that, as the hierarchical goes up it generalize the view of entities and as we go deep in the hierarchy it gives us detail of every entity included.

Going up in this structure is called generalization, where entities are clubbed together to represent a more generalized view. For example, a particular student named, Mira can be generalized along with all the students, the entity shall be student, and further a student is person. The reverse is called specialization where a person is student, and that student is Mira.

Generalization

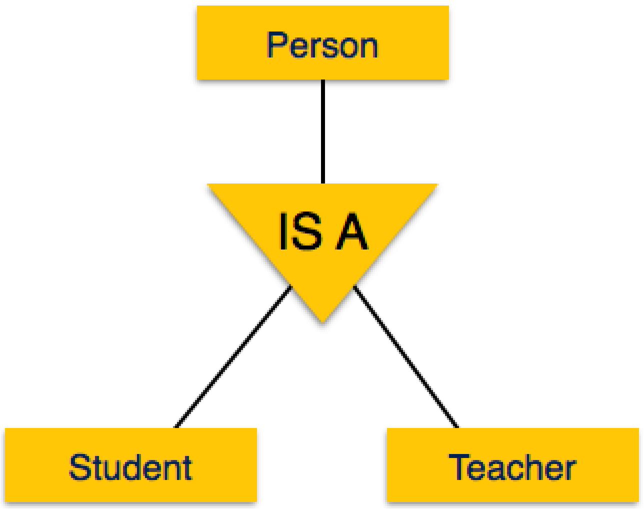
As mentioned above, the process of generalizing entities, where the generalized entities contain the properties of all the generalized entities is called Generalization. In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For an example, pigeon, house sparrow, crow and dove all can be generalized as Birds.



[*Image: Generalization*]

Specialization

Specialization is a process, which is opposite to generalization, as mentioned above. In specialization, a group of entities is divided into sub-groups based on their characteristics. Take a group Person for example. A person has name, date of birth, gender etc. These properties are common in all persons, human beings. But in a company, a person can be identified as employee, employer, customer or vendor based on what role do they play in company.



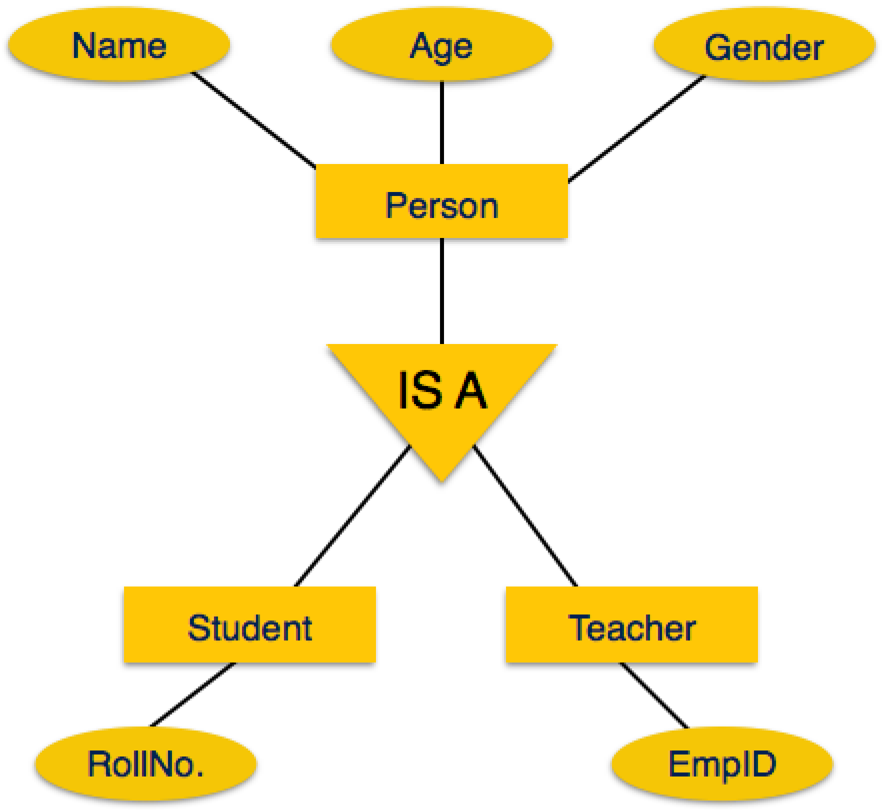
[*Image: Specialization*]

Similarly, in a school database, a person can be specialized as teacher, student or staff; based on what role do they play in school as entities.

Inheritance

We use all above features of ER-Model, in order to create classes of objects in object oriented programming. This makes it easier for the programmer to concentrate on what she is programming. Details of entities are generally hidden from the user, this process known as abstraction.

One of the important features of Generalization and Specialization, is inheritance, that is, the attributes of higher-level entities are inherited by the lower level entities.



[*Image: Inheritance*]

For example, attributes of a person like name, age, and gender can be inherited by lower level entities like student and teacher etc.

ER Model to Relational Model

ER Model when conceptualized into diagrams gives a good overview of entity-relationship, which is easier to understand. ER diagrams can be mapped to Relational schema that is, it is possible to create relational schema using ER diagram. Though we cannot import all the ER constraints into Relational model but an approximate schema can be generated.

There are more than one processes and algorithms available to convert ER Diagrams into Relational Schema. Some of them are automated and some of them are manual process. We may focus here on the mapping diagram contents to relational basics.

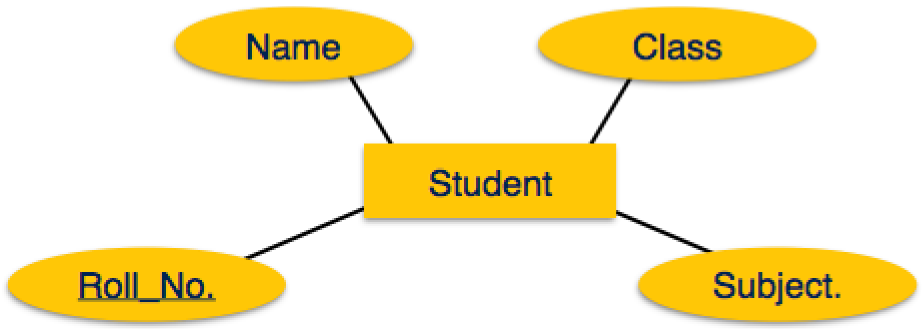
ER Diagrams mainly comprised of:

- Entity and its attributes
- Relationship, which is association among entities.

Mapping Entity

An entity is a real world object with some attributes.

Mapping Process (Algorithm):



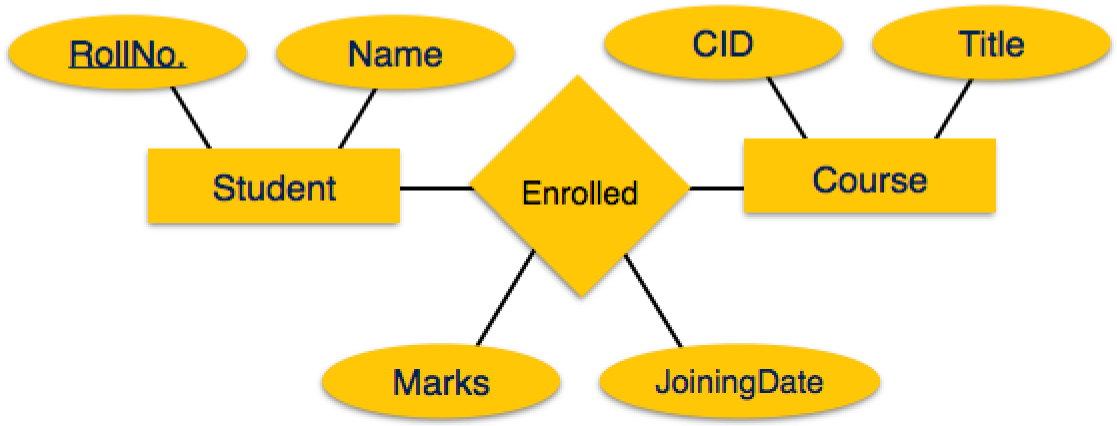
[*Image: Mapping Entity*]

- Create table for each entity
- Entity's attributes should become fields of tables with their respective data types.
- Declare primary key

Mapping relationship

A relationship is association among entities.

Mapping process (Algorithm):



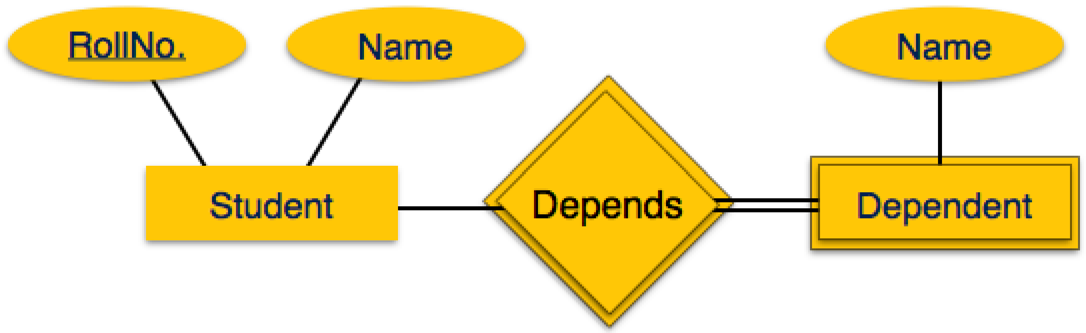
[*Image: Mapping relationship*]

- Create table for a relationship
- Add the primary keys of all participating Entities as fields of table with their respective data types.
- If relationship has any attribute, add each attribute as field of table.
- Declare a primary key composing all the primary keys of participating entities.
- Declare all foreign key constraints.

Mapping Weak Entity Sets

A weak entity sets is one which does not have any primary key associated with it.

Mapping process (Algorithm):



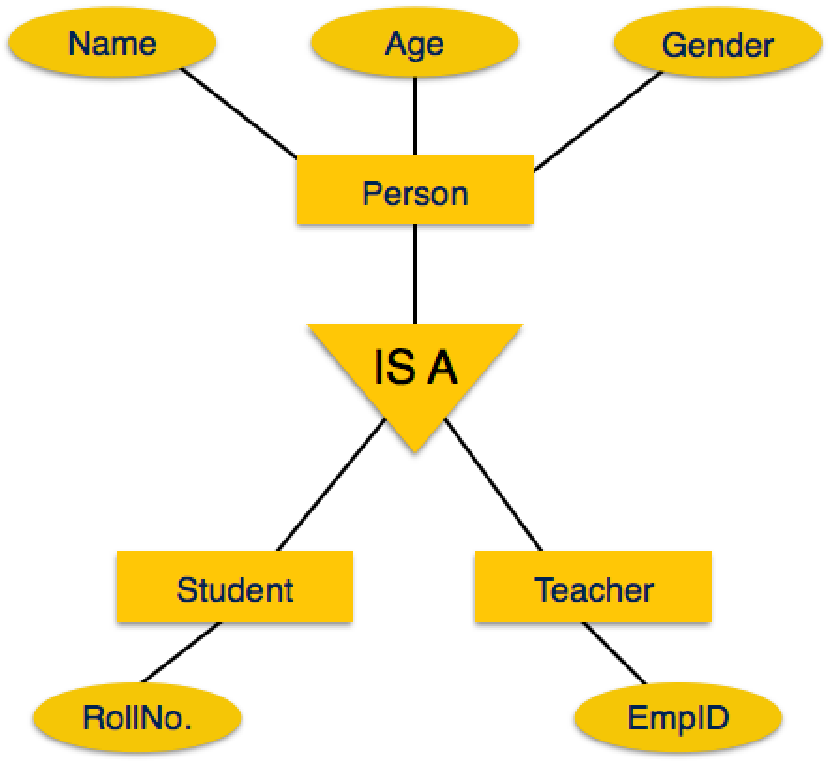
[*Image: Mapping Weak Entity Sets*]

- Create table for weak entity set
- Add all its attributes to table as field
- Add the primary key of identifying entity set
- Declare all foreign key constraints

Mapping hierarchical entities

ER specialization or generalization comes in the form of hierarchical entity sets.

Mapping process (Algorithm):



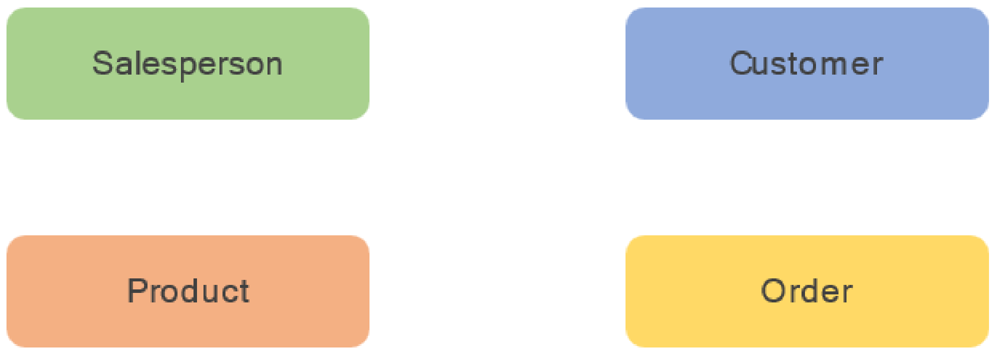
[*Image: Mapping hierarchical entities*]

Entity Relationship Diagram

Problem statement:
E-commerce is one of the emerging fields and is widely accepted across the globe. Design an ER model for a start-up named "Sell in the Sale" based on the below business rules so that it is easy for the management to understand the design of their company's database. A salesperson can manage many customers. A customer, however, is managed by only one salesperson. One customer can place multiple orders, but each order will always belong to single customer. An order lists many products, and a single product can be present in multiple orders. An order will have the columns orderID, orderDate, noOfProducts, and productName. Make sure to follow proper notations to represent the entities, attributes, and relationships with their types (1:1, 1:M, M:1, and M:M).
Objective:
Identify the entities, attributes, and relationships in order to solve this problem

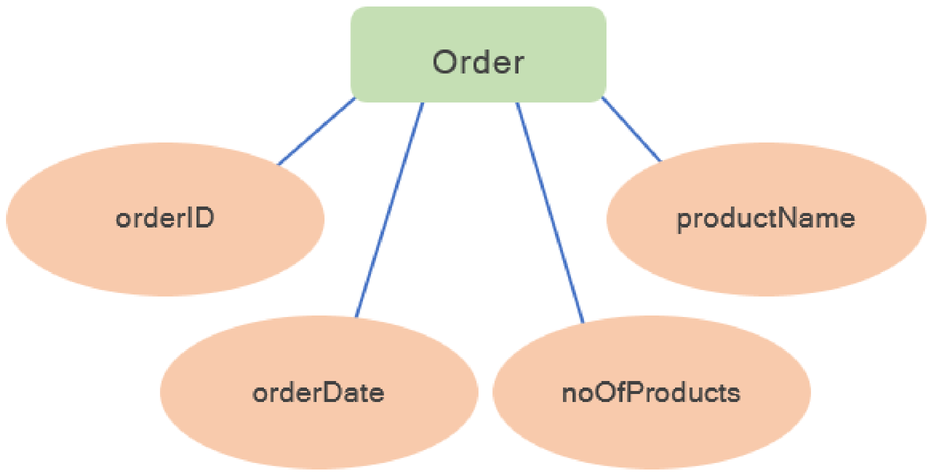
Steps to be performed:

Step 01: Identify the entities
An entity is a real-world object that represents the data. It is represented as a rectangle.
Here, Salesperson, Customer, Order, and Product represent the entities.



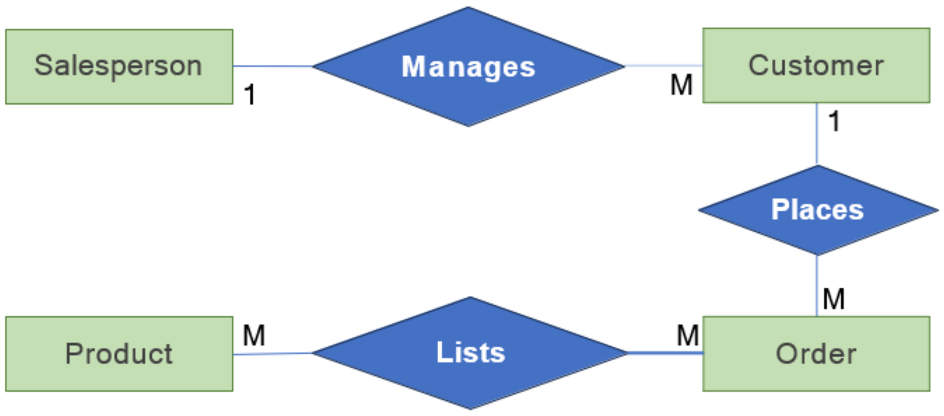
Step 02: Identify the attributes

An attribute is a property that describes an entity. It is represented as an ellipse. orderID, orderDate, noOfProducts, and productName describe the entity Order. Hence, they are attributes.

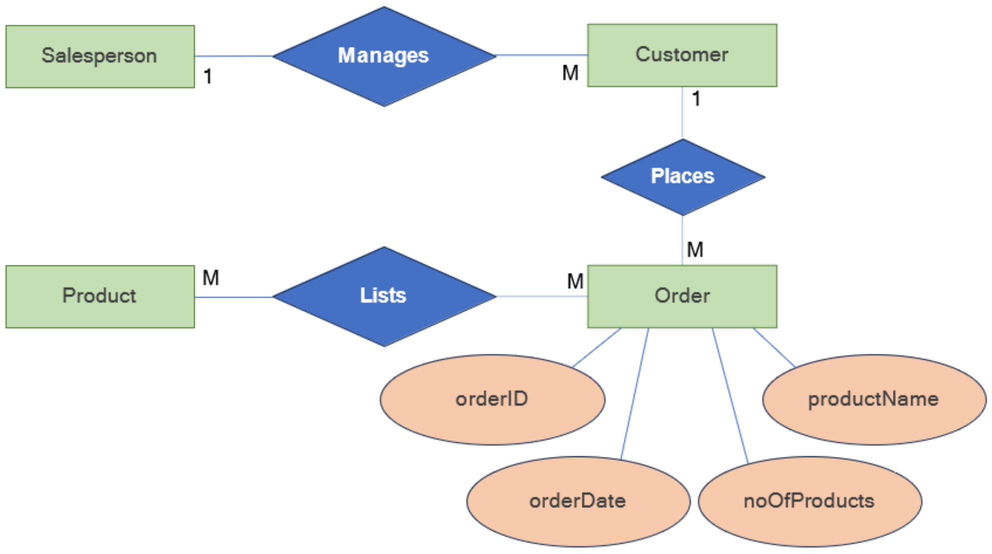


Step 03: Identify the relationships
Relationships are used to document the interaction between the entities. It is represented as a diamond.
A salesperson can manage many customers. A customer is managed by only one salesperson.

Hence, the relationship here is Manages and one-to-many (1:M).
Multiple orders can be placed by a customer, but an order will always belong to a single customer. Hence the relationship here is Places and one-to-many (1:M).
An order lists many products, and a single product can be present in multiple orders.
Hence the relationship here is Lists and many-to-many (M:M).



Step 04: Using the information gathered in steps 1, 2, and 3, create an ER diagram



Transaction property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction
Atomicity
Consistency
Isolation
Durability

Codd's 12 Rules

Dr Edgar F. Codd did some extensive research in Relational Model of database systems and came up with twelve rules of his own which according to him, a database must obey in order to be a true relational database.

Rule 0: The Foundation rule

A relational database management system must manage its stored data using only its relational capabilities. The system must qualify as relational, as a database, and as a management system. For a system to qualify as a relational database management system (RDBMS), that system must use its relational facilities (exclusively) to manage the database.

Rule 1: Information rule

This rule states that all information (data), which is stored in the database, must be a value of some table cell. Everything in a database must be stored in table formats. This information can be user data or meta-data.

Rule 2: Guaranteed Access rule

This rule states that every single data element (value) is guaranteed to be accessible logically with combination of table-name, primary-key (row value) and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL values

This rule states the NULL values in the database must be given a systematic treatment. As a NULL may have several meanings, i.e. NULL can be interpreted as one the following: data is missing, data is not known, data is not applicable etc.

Rule 4: Active online catalog

This rule states that the structure description of whole database must be stored in an online catalog, i.e. data dictionary, which can be accessed by the authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive data sub-language rule

This rule states that a database must have a support for a language which has linear syntax which is capable of data definition, data manipulation and transaction management operations. Database can be accessed by means of this language only, either directly or by means of some application. If the database can be accessed or manipulated in some way without any help of this language, it is then a violation.

Rule 6: View updating rule

This rule states that all views of database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-level insert, update and delete rule

This rule states the database must employ support high-level insertion, updation and deletion. This must not be limited to a single row that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical data independence

This rule states that the application should not have any concern about how the data is physically stored. Also, any change in its physical structure must not have any impact on application.

Rule 9: Logical data independence

This rule states that the logical data must be independent of its user’s view (application). Any change in logical data must not imply any change in the application using it. For example, if two tables are merged or one is split into two different tables, there should be no impact the change on user application. This is one of the most difficult rule to apply.

Rule 10: Integrity independence

This rule states that the database must be independent of the application using it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes database independent of the front-end application and its interface.

Rule 11: Distribution independence

This rule states that the end user must not be able to see that the data is distributed over various locations. User must also see that data is located at one site only. This rule has been proven as a foundation of distributed database systems.

Rule 12: Non-subversion rule

This rule states that if a system has an interface that provides access to low level records, this interface then must not be able to subvert the system and bypass security and integrity constraints.

Relation Data Model

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and have all the properties and capabilities required to process data with storage efficiency.

Concepts

Tables: In relation data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represent records and columns represents the attributes.

Tuple: A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance: A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema: This describes the relation name (table name), attributes and their names.

Relation key: Each row has one or more attributes which can identify the row in the relation (table) uniquely, is called the relation key.

Attribute domain: Every attribute has some pre-defined value scope, known as attribute domain.

DBMS Normalization

Functional Dependency

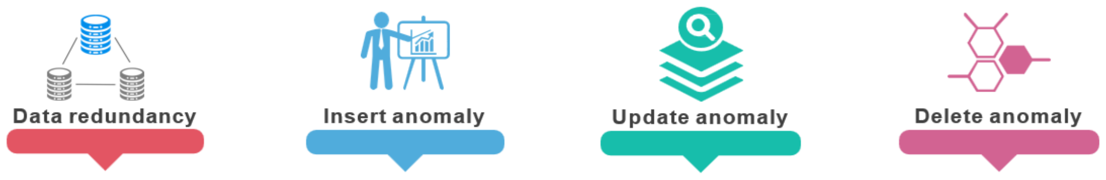
Functional dependency (FD) is set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by arrow sign (→), that is X→Y, where X functionally determines Y. The left hand side attributes determines the values of attributes at right hand side.

Normalization

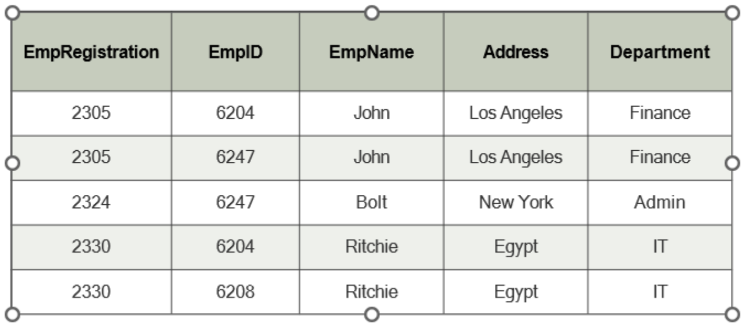
If a database design is not perfect it may contain anomalies, which are like a bad dream for database itself. Managing a database with anomalies is next to impossible.

- **Update anomalies:** if data items are scattered and are not linked to each other properly, then there may be instances when we try to update one data item that has copies of it scattered at several places, few instances of it get updated properly while few are left with there old values. This leaves database in an inconsistent state.
- **Deletion anomalies:** we tried to delete a record, but parts of it left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies:** we tried to insert data in a record that does not exist at all.



Data Redundancy

Data redundancy occurs when two or more rows or columns have the same or repeated value, causing the memory to be used inefficiently.

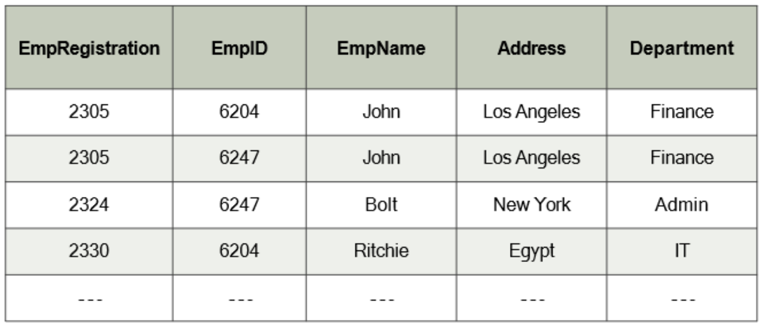


EmpRegistration	EmpID	EmpName	Address	Department
2305	6204	John	Los Angeles	Finance
2305	6247	John	Los Angeles	Finance
2324	6247	Bolt	New York	Admin
2330	6204	Ritchie	Egypt	IT
2330	6208	Ritchie	Egypt	IT

The records of the two employees, **John** and **Ritchie** are repetitive in the above table, which results in data redundancy.

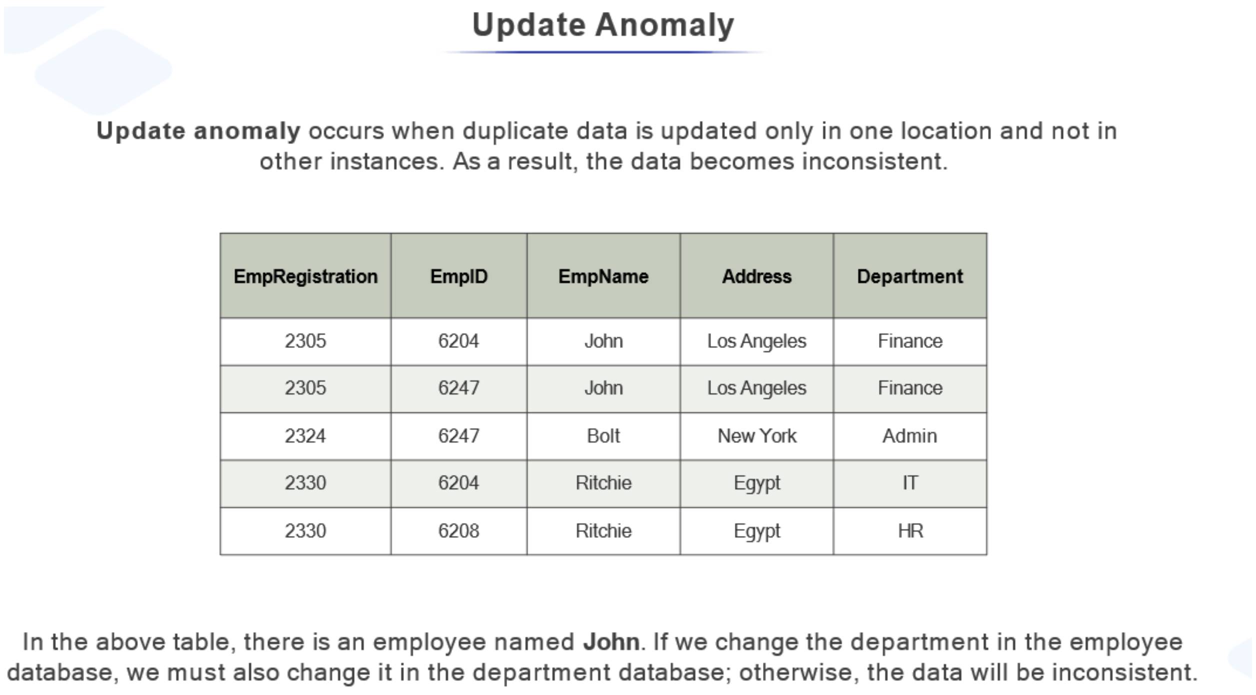
Insert Anomaly

Insert anomaly occurs when some attributes or data items are to be inserted into the database without the existence of other attributes.



EmpRegistration	EmpID	EmpName	Address	Department
2305	6204	John	Los Angeles	Finance
2305	6247	John	Los Angeles	Finance
2324	6247	Bolt	New York	Admin
2330	6204	Ritchie	Egypt	IT
---	---	---	---	---

If we want to enter a new EmpID into the employee table, we must wait till the employee joins the organization. Hence, it is called insertion anomalies.

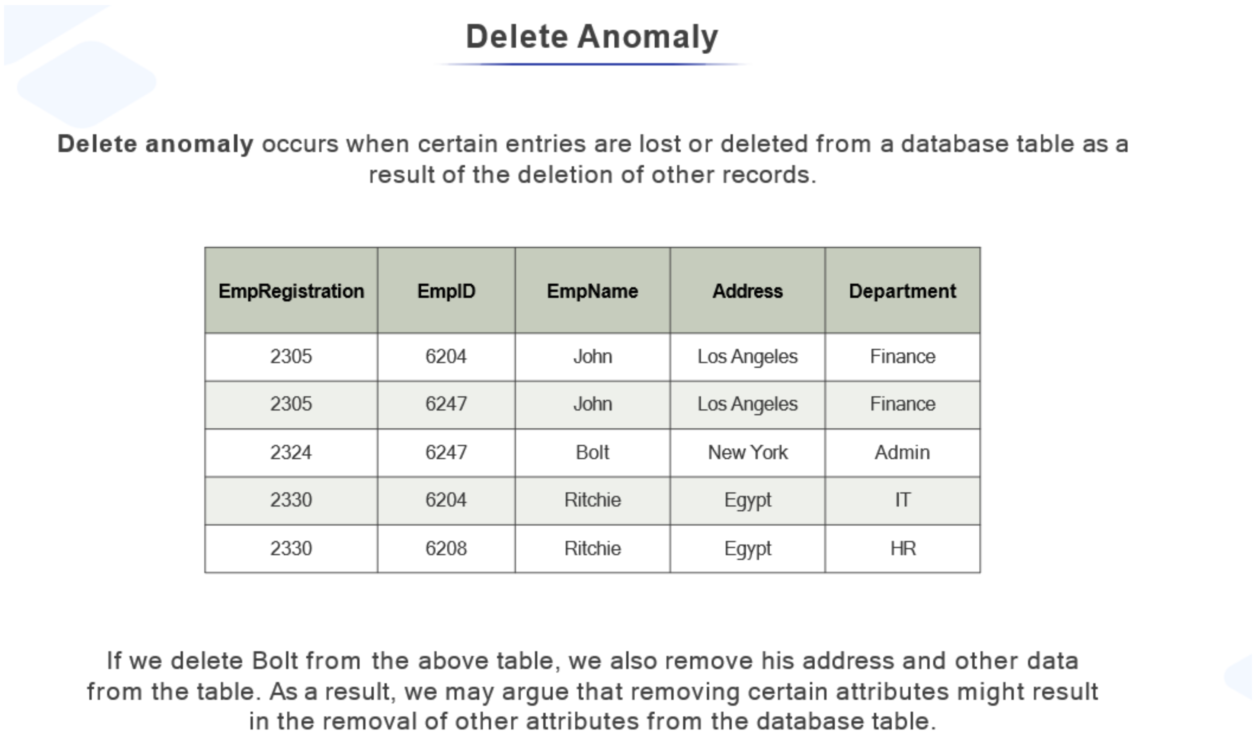


Update Anomaly

Update anomaly occurs when duplicate data is updated only in one location and not in other instances. As a result, the data becomes inconsistent.

EmpRegistration	EmpID	EmpName	Address	Department
2305	6204	John	Los Angeles	Finance
2305	6247	John	Los Angeles	Finance
2324	6247	Bolt	New York	Admin
2330	6204	Ritchie	Egypt	IT
2330	6208	Ritchie	Egypt	HR

In the above table, there is an employee named **John**. If we change the department in the employee database, we must also change it in the department database; otherwise, the data will be inconsistent.



Delete Anomaly

Delete anomaly occurs when certain entries are lost or deleted from a database table as a result of the deletion of other records.

EmpRegistration	EmpID	EmpName	Address	Department
2305	6204	John	Los Angeles	Finance
2305	6247	John	Los Angeles	Finance
2324	6247	Bolt	New York	Admin
2330	6204	Ritchie	Egypt	IT
2330	6208	Ritchie	Egypt	HR

If we delete Bolt from the above table, we also remove his address and other data from the table. As a result, we may argue that removing certain attributes might result in the removal of other attributes from the database table.

While designing a database out of an entity–relationship model, the main problem existing in that

"raw" database is redundancy. Redundancy is storing the same data item in more one place. A redundancy creates several problems like the following:

1. Extra storage space: storing the same data in many places takes large amount of disk space.

2. Entering same data more than once during data insertion.

3. Deleting data from more than one place during deletion.

4. Modifying data in more than one place.

5. Anomalies may occur in the database if insertion, deletion, modification etc are not done properly. It creates inconsistency and unreliability in the database.

To solve this problem, the "raw" database needs to be normalized. This is a step by step process of removing different kinds of redundancy and anomaly at each step. At each step a specific rule is followed to remove specific kind of impurity in order to give the database a slim and clean look.

Un-Normalized Form (UNF)

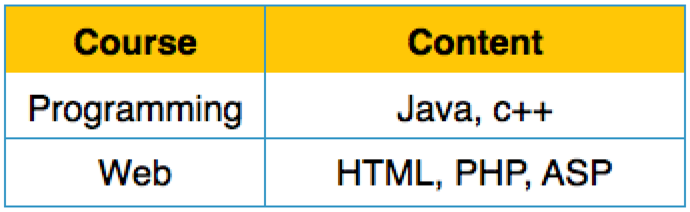
If a table contains non-atomic values at each row, it is said to be in UNF. An atomic value is something that can not be further decomposed. A non-atomic value, as the name suggests, can be further decomposed and simplified. Consider the following table:

Emp-Id	**Emp-Name**	**Month**	**Sales**	**Bank-Id**	**Bank-Name**
E01	AA	Jan	1000	B01	SBI
		Feb	1200		
		Mar	850		
E02	BB	Jan	2200	B02	UTI
		Feb	2500		
E03	CC	Jan	1700	B01	SBI

Feb	1800
Mar	1850
Apr	1725

First Normal Form:

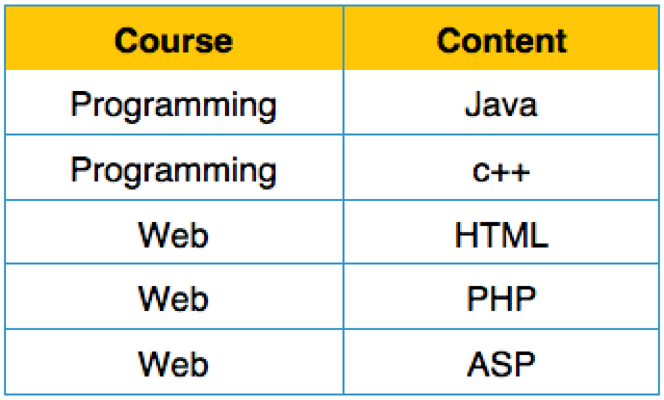
This is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. Values in atomic domain are indivisible units.



Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

[*Image: Unorganized relation*]

We re-arrange the relation (table) as below, to convert it to First Normal Form



Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

[*Image: Relation in 1NF*]

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

Example :-A relation is said to be in 1NF if it contains no non-atomic values and each row can provide a unique combination of values. The above table in UNF can be processed to create the following table in 1NF. And remove repeating and non repeating group of elements in two different tables

Emp-Id	**Emp-Name**	**Month**	**Sales**	**Bank-Id**	**Bank-Name**
E01	**AA**	**Jan**	**1000**	**B01**	**SBI**

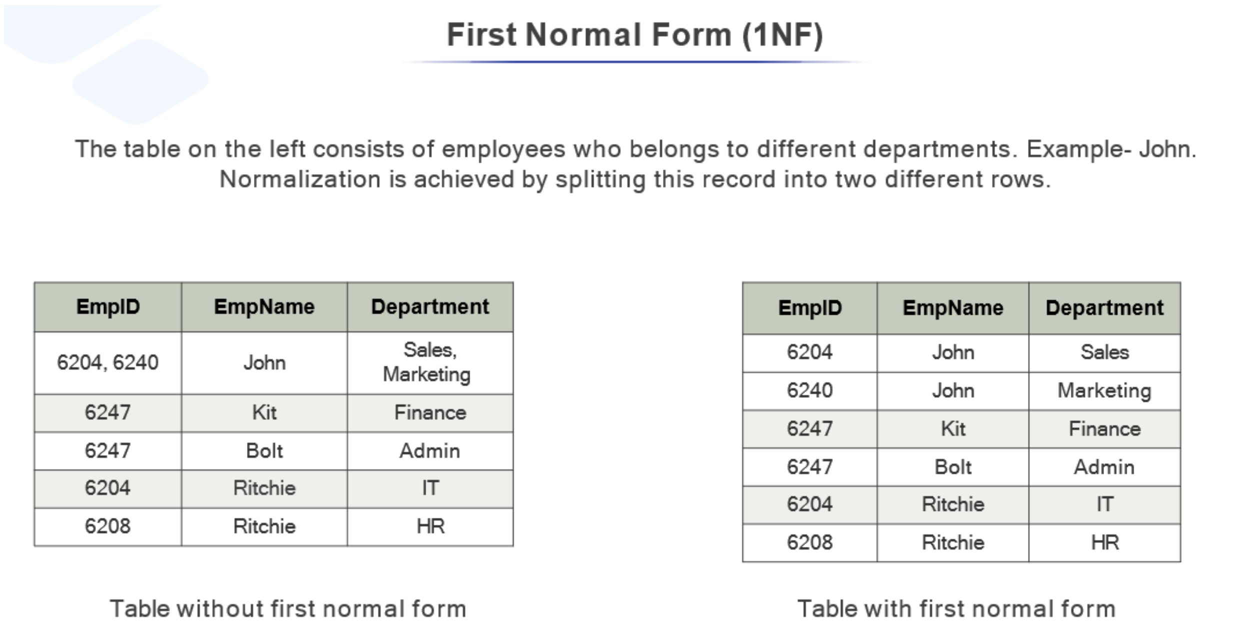
E01	**AA**	**Feb**	**1200**	**B01**	**SBI**
E01	**AA**	**Mar**	**850**	**B01**	**SBI**
E02	**BB**	**Jan**	**2200**	**B02**	**UTI**
E02	**BB**	**Feb**	**2500**	**B02**	**UTI**
E03	**CC**	**Jan**	**1700**	**B01**	**SBI**
E03	**CC**	**Feb**	**1800**	**B01**	**SBI**
E03	**CC**	**Mar**	**1850**	**B01**	**SBI**
E03	**CC**	**Apr**	**1725**	**B01**	**SBI**

As you can see now, each row contains a unique combination of values. Unlike in UNF, this relation contains only atomic values

First Normal Form (1NF)

- The 1NF states that all the attributes in a relation must have atomic domains.
- The 1NF specifies that a table attribute cannot have multiple values.

In 1NF, multivalued attribute, composite attribute, and their combination are not allowed



First Normal Form (1NF)

The table on the left consists of employees who belongs to different departments. Example- John. Normalization is achieved by splitting this record into two different rows.

EmpID	EmpName	Department
6204, 6240	John	Sales, Marketing
6247	Kit	Finance
6247	Bolt	Admin
6204	Ritchie	IT
6208	Ritchie	HR

Table without first normal form

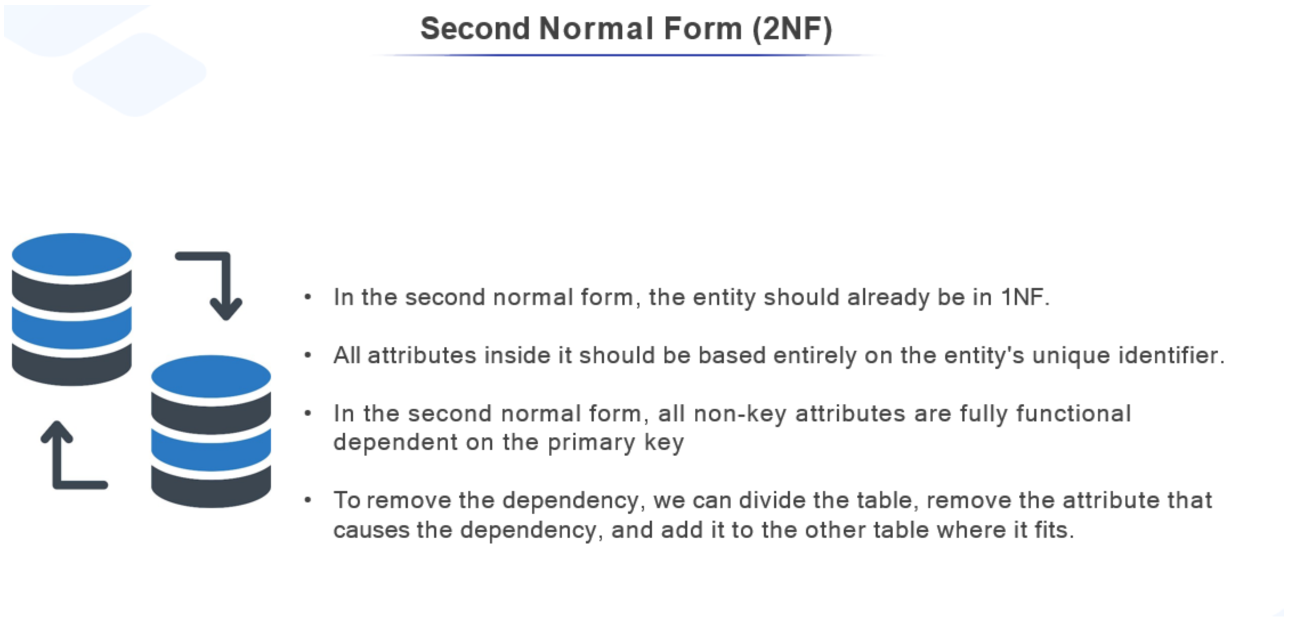
EmpID	EmpName	Department
6204	John	Sales
6240	John	Marketing
6247	Kit	Finance
6247	Bolt	Admin
6204	Ritchie	IT
6208	Ritchie	HR

Table with first normal form

Second Normal Form:

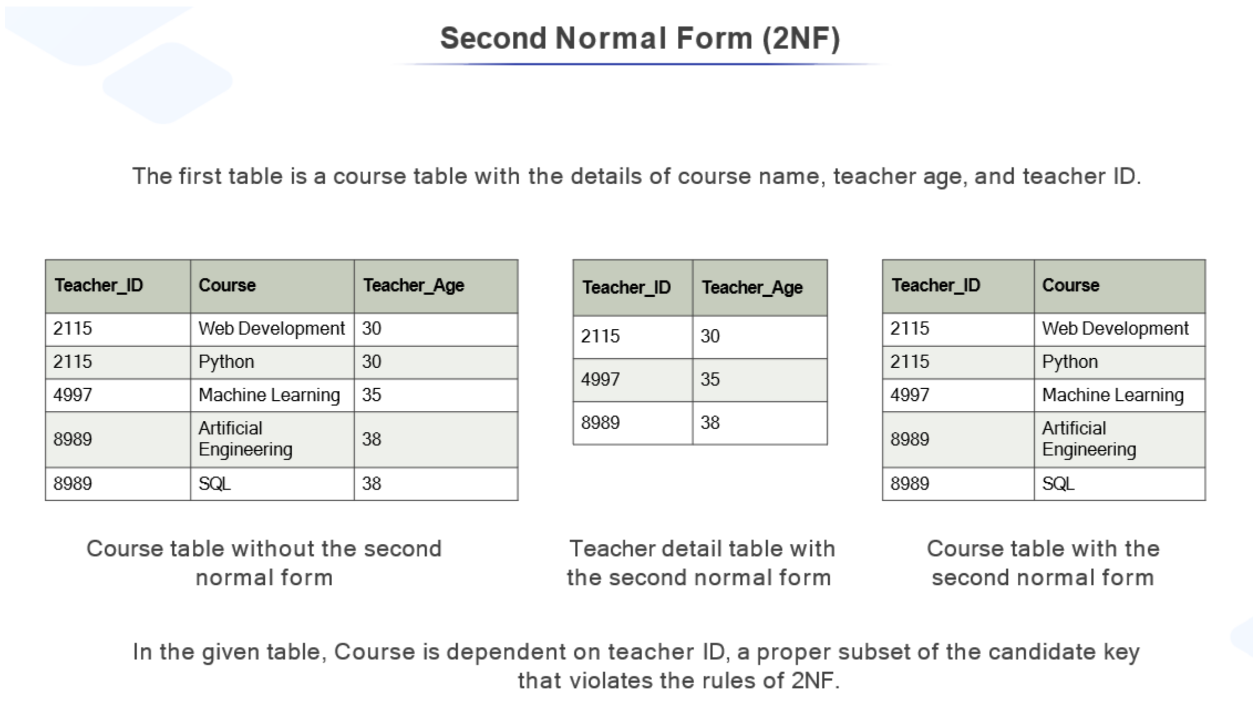
Before we learn about second normal form, we need to understand the following:

- **Prime attribute:** an attribute, which is part of prime-key, is prime attribute.
- **Non-prime attribute:** an attribute, which is not a part of prime-key, is said to be a non-prime attribute.



Second Normal Form (2NF)

- In the second normal form, the entity should already be in 1NF.
- All attributes inside it should be based entirely on the entity's unique identifier.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key
- To remove the dependency, we can divide the table, remove the attribute that causes the dependency, and add it to the other table where it fits.



Second Normal Form (2NF)

The first table is a course table with the details of course name, teacher age, and teacher ID.

Teacher_ID	Course	Teacher_Age
2115	Web Development	30
2115	Python	30
4997	Machine Learning	35
8989	Artificial Engineering	38
8989	SQL	38

Course table without the second normal form

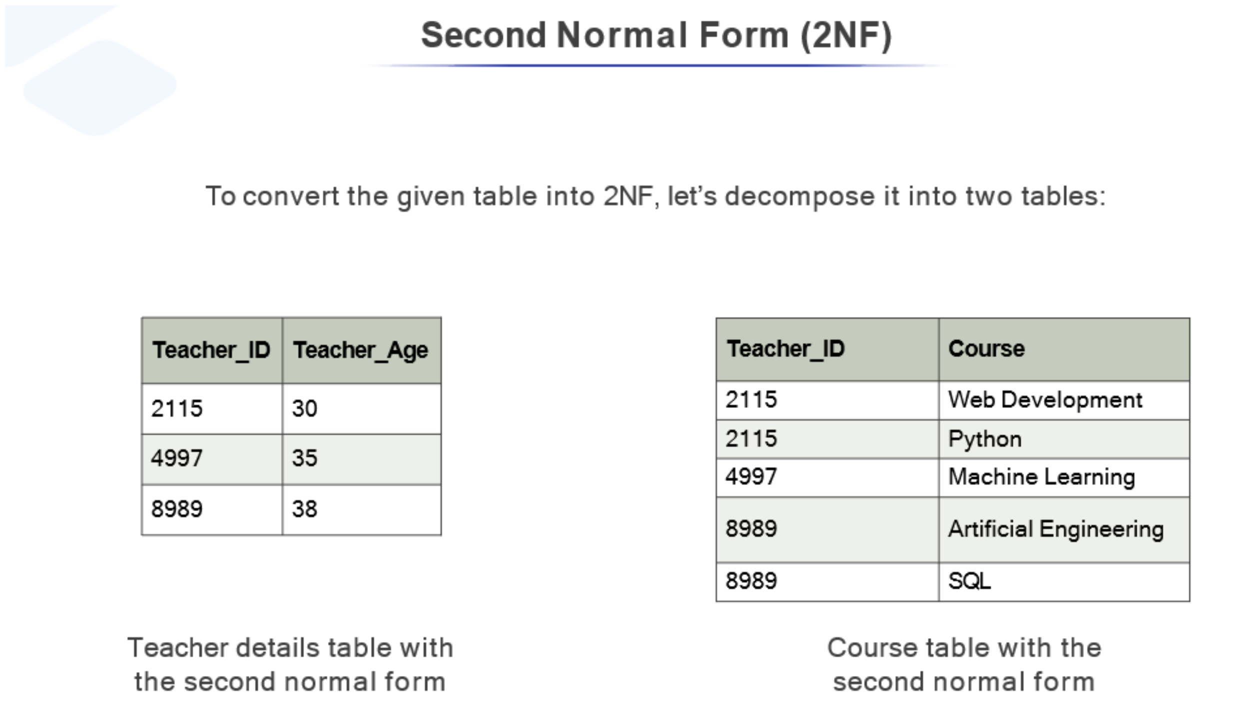
Teacher_ID	Teacher_Age
2115	30
4997	35
8989	38

Teacher detail table with the second normal form

Teacher_ID	Course
2115	Web Development
2115	Python
4997	Machine Learning
8989	Artificial Engineering
8989	SQL

Course table with the second normal form

In the given table, Course is dependent on teacher ID, a proper subset of the candidate key that violates the rules of 2NF.



Second Normal Form (2NF)

To convert the given table into 2NF, let's decompose it into two tables:

Teacher_ID	Teacher_Age
2115	30
4997	35
8989	38

Teacher details table with the second normal form

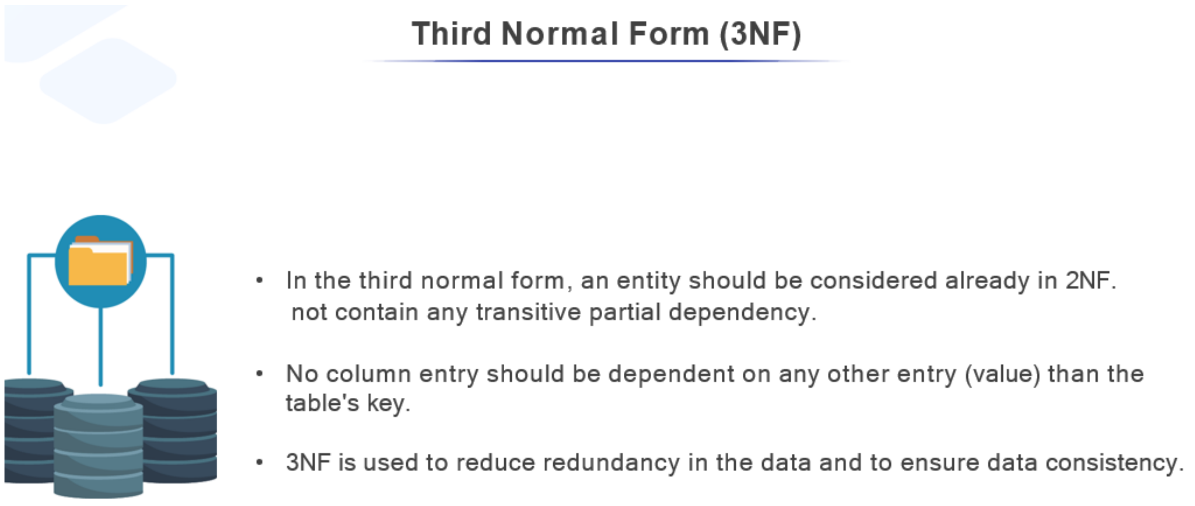
Teacher_ID	Course
2115	Web Development
2115	Python
4997	Machine Learning
8989	Artificial Engineering
8989	SQL

Course table with the second normal form

Third Normal Form:

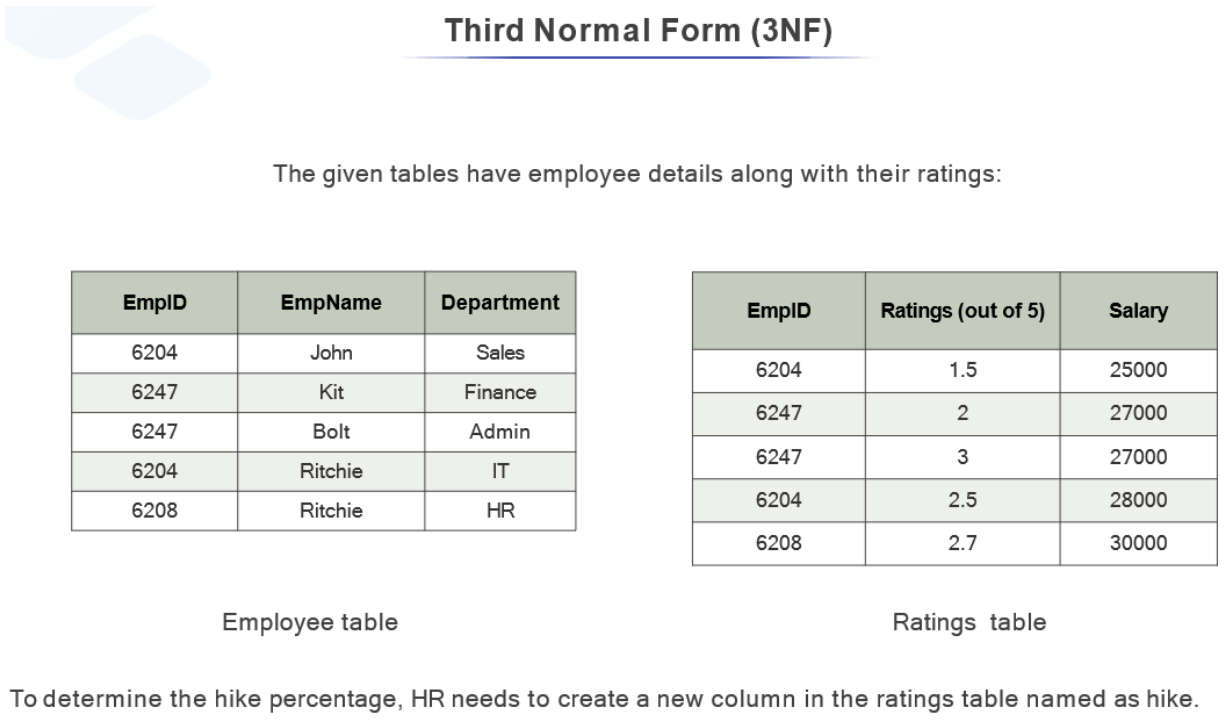
For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy:

- 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.



Third Normal Form (3NF)

- In the third normal form, an entity should be considered already in 2NF. not contain any transitive partial dependency.
- No column entry should be dependent on any other entry (value) than the table's key.
- 3NF is used to reduce redundancy in the data and to ensure data consistency.



Third Normal Form (3NF)

The given tables have employee details along with their ratings:

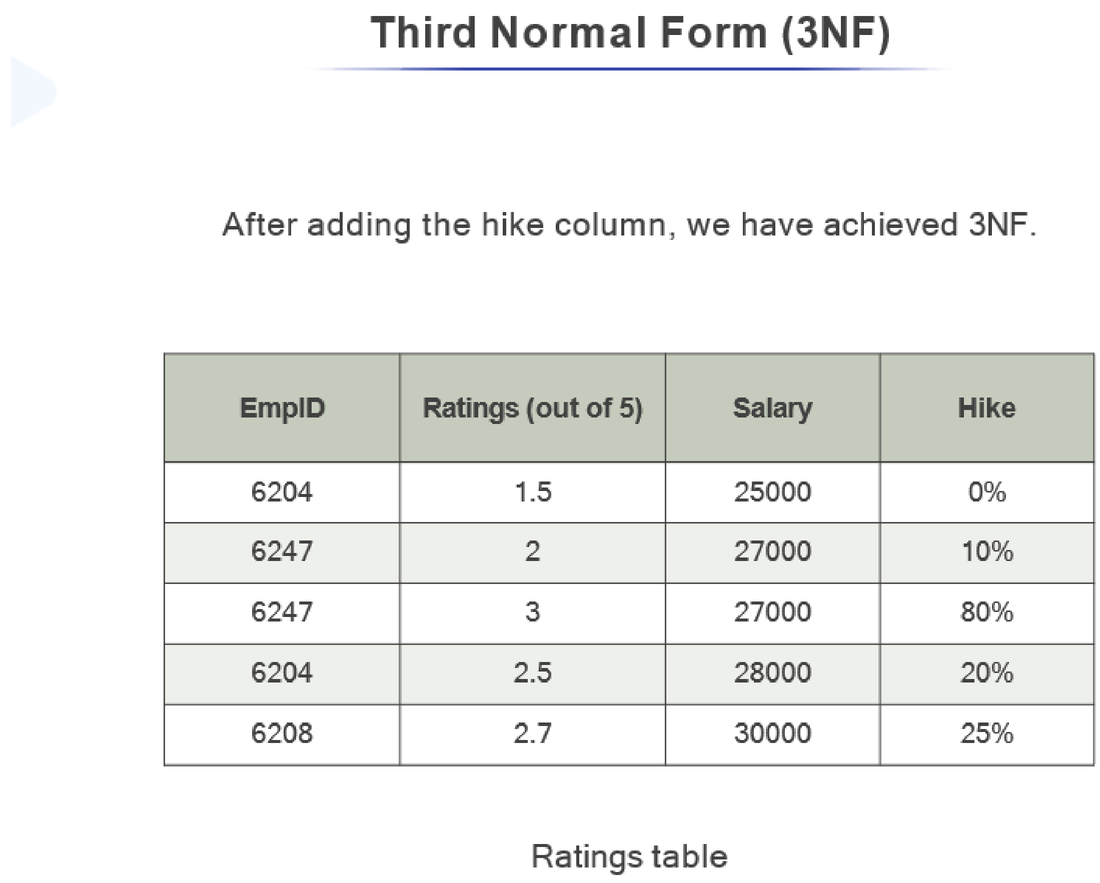
EmpID	EmpName	Department
6204	John	Sales
6247	Kit	Finance
6247	Bolt	Admin
6204	Ritchie	IT
6208	Ritchie	HR

Employee table

EmpID	Ratings (out of 5)	Salary
6204	1.5	25000
6247	2	27000
6247	3	27000
6204	2.5	28000
6208	2.7	30000

Ratings table

To determine the hike percentage, HR needs to create a new column in the ratings table named as hike.



Third Normal Form (3NF)

After adding the hike column, we have achieved 3NF.

EmpID	Ratings (out of 5)	Salary	Hike
6204	1.5	25000	0%
6247	2	27000	10%
6247	3	27000	80%
6204	2.5	28000	20%
6208	2.7	30000	25%

Ratings table