PARVATHA REDDY BABUL REDDY
VISVODAYA INSTITUTE OF TECHNOLOGY AND SCIENCE's
VITSPACE

For more Materials
☺ Click the link given below ☺

vitspace.netlify.com

Scroll to read your material

UNIT- III

Database Design and the E-R Model: Overview of the Design Process, The Entity Relationship Model, Constraints, Removing Redundant Attributes in Entity Sets, Entity Relationship Diagrams, Reduction to Relational Schemas, Entity-Relationship Design Issues. Relational Database Design: Features of Good Relational Designs, Atomic Domains and First Normal Form, Decomposition Using Functional Dependencies, Functional-Dependency Theory, Algorithms for Decomposition, Decomposition Using Multivalued Dependencies, More Normal Forms.

Database Design and the E-R Model

3.1 Overview of the Design Process

Design of the Database Schema, design of the programs that access and update the data, design of a security scheme for controlling access to the data are all involved in the creation of a Database. The central role in the designing process involves the requirements of the user. For designing a database, there are four phases, namely:

Initial Phase : The Initial Phase of database design is to fully describe the data needs of the potential database users The database designer needs to interact extensively with domain experts and users to carry out this task. The outcome of this phase is a specification of user requirements.

Conceptual Design Phase : The designer chooses a data model, by applying the concepts of the data model that is chosen and translates these requirements into a conceptual schema of the database. The **entity-relationship model** is typically used to represent the conceptual design. The conceptual schema specifies the entities that are represented in the database, the attributes of the entities, the relationships among the entities, and constraints on the entities and relationships. Typically, the conceptual-design phase results in the creation of an entity-relationship diagram that provides a graphic representation of the schema.

A fully developed conceptual schema also indicates the functional requirements of the enterprise. In a **specification of functional requirements**, users describe the kinds of operations (or transactions) that will be performed on the data. The operations can include updating,

searching, retrieving and deleting data. At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements.

Logical Design Phase : The database designer plots the high-level conceptual schema onto the implementation of the data model of the database system that has to be used. The implementation data model is also called the relational data model.

Physical Design Phase : Here, the designer uses the resulting system-specific database schema. In this phase, the physical features of the database are specified.

The following **two** issues are to be avoided in designing a database schema,

- Redundancy

- Incompleteness

3.2 The Entity-Relationship Model

The **entity-relationship** (**E-R**) data model was developed to facilitate database design by allowing specification of an *enterprise schema* that represents the overall logical structure of a database. The E-R data model employs **three** basic concepts: **entity sets, relationship sets**, and **attributes**.

Entity : An entity is an object that exists and which is distinguishable from other objects. An entity can be a person, a place, an object, an event, or a concept about which an organization wishes to maintain data. The following are some examples of entities:

Person: STUDENT, EMPLOYEE, CLIENT
Object: COUCH, AIRPLANE, MACHINE
Place: CITY, NATIONAL PARK, ROOM, WAREHOUSE
Event: WAR, MARRIAGE, LEASE
Concept: PROJECT, ACCOUNT, COURSE

All these entities have some attributes or properties that give them their identity.

It is important to understand the distinction between an **entity type**, an **entity instance**, and an **entity set.** An **entity type** defines a collection of entities that have same attributes. An **entity instance** is a single item in this collection. An **entity set** is a set of entity instances.

Strong Entity

The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity. Strong Entity is represented by a single rectangle.

Weak Entity

The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities. Weak Entity is represented by double rectangle
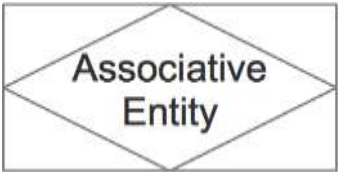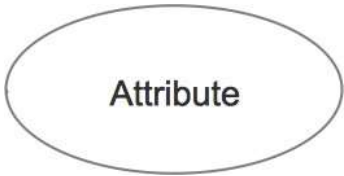
Attributes

Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.
There exists a domain 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, a 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 the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, age can be derived from date of birth.

- **Single-value attribute** − Single-value attributes contain single value. For example – Adhar Number.

- **Multi-value attribute** − Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email, address, etc.

Symbol	Shape Name	Symbol Description
Entity	Entity	An entity is represented by a rectangle which contains the entity's name.
Entities		
Entity	Entity	An entity is represented by a rectangle which contains the entity's name.
Weak Entity	Weak Entity	An entity that cannot be uniquely identified by its attributes alone. The existence of a weak entity is dependent upon another entity called the owner entity. The weak entity's identifier is a combination of the identifier of the owner entity and the partial key of the weak entity.
Associative Entity	Associative Entity	An entity used in a many-to-many relationship. All relationships for the associative entity should be many
Attributes		
Attribute	Attribute	In the Chen notation, each attribute is represented by an oval containing atributte's name
Key attribute	Key attribute	An attribute that uniquely identifies a particular entity. The name of a key attribute is underscored.

Multivalue attribute	Multivalued attribute	An attribute that can have many values (there are many distinct values entered for it in the same column of the table). Multivalued attribute is depicted by a dual oval.
Derived attribute	Derived attribute	An attribute whose value is calculated (derived) from other attributes. The derived attribute may or may not be physically stored in the database. In the Chen notation, this attribute is represented by dashed oval.
	Relationships	
Relationship	Strong relationship	A relationship where entity is existence-independent of other entities, and PK of Child doesn't contain PK component of Parent Entity. A strong relationship is represented by a single rhombus
Relationship	Weak (identifying) relationship	A relationship where Child entity is existence-dependent on parent, and PK of Child Entity contains PK component of Parent Entity. This relationship is represented by a double rhombus.
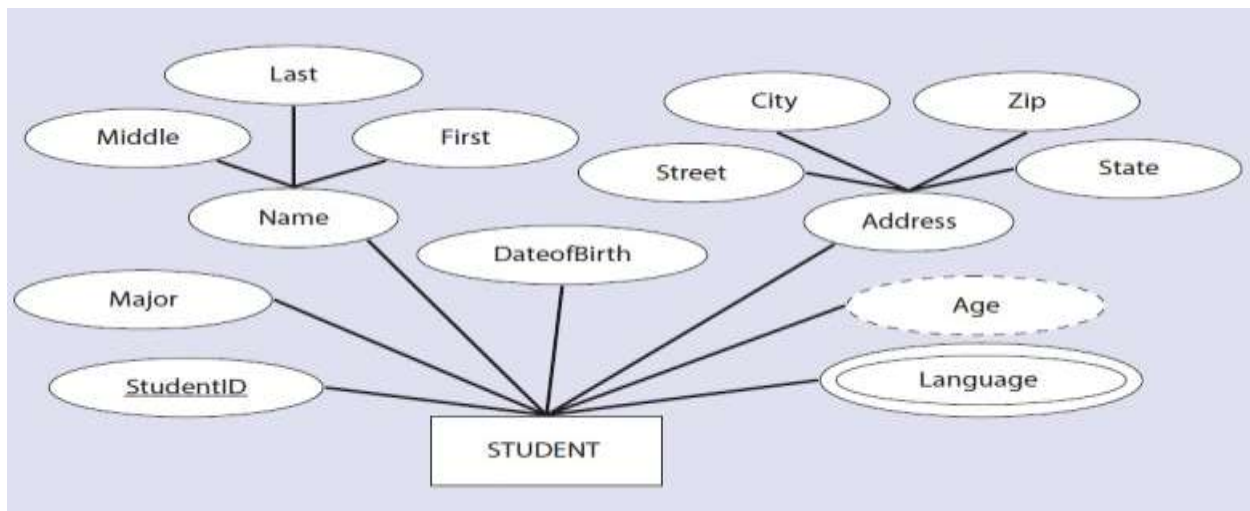
Attributes of the STUDENT entity type.

RELATIONSHIP AND RELATIONSHIP SETS

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

Relationship Set

A set of relationships of similar type is called a **relationship set**. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**. Descriptive attributes are used to record information about the relationship, rather than about any one of the participating entities

Degree of Relationship

The *degree* of a relationship is the number of entity types that participate in the relationship. The three most common relationships in ER models are *Binary, Unary* and *Ternary*

A **binary relationship,** or an association between two entity types, is the most common form of a relationship expressed by an E-R diagram.

For Example:
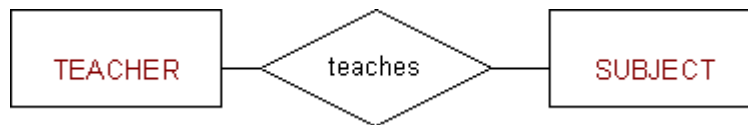
A **unary relationship** is when both participants in the **relationship** are the same entity
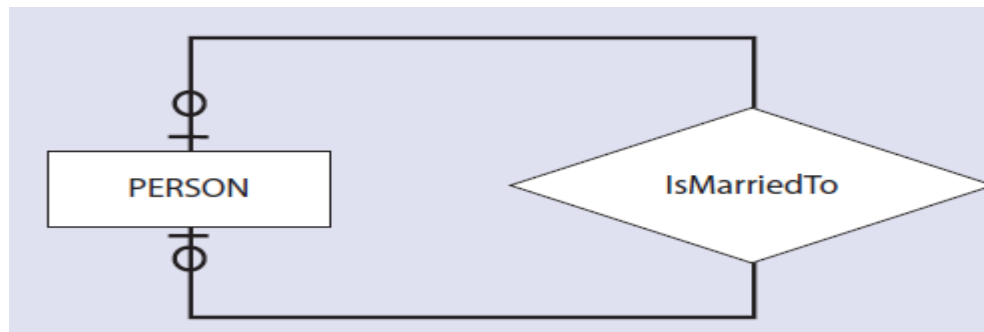
A **ternary relationship** is when three entities participate in the **relationship**
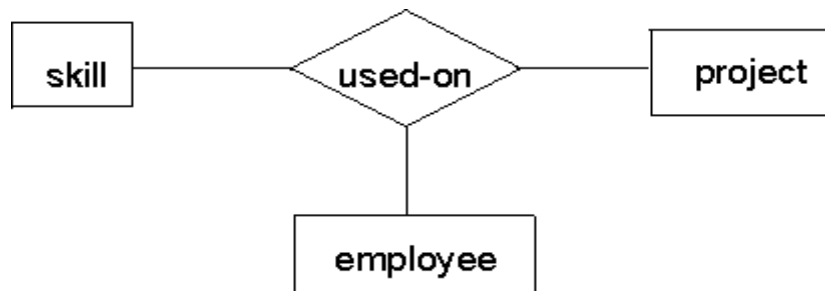
The **degree of relationship** (also known as cardinality) is the number of occurrences in one entity which are associated (or linked) to the number of occurrences in another.

Cardinality can be of different types:

1. One to one – When each entity in each entity set can take part **only once   in   the relationship**, the cardinality is one to one. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.

Using Sets, it can be represented as:
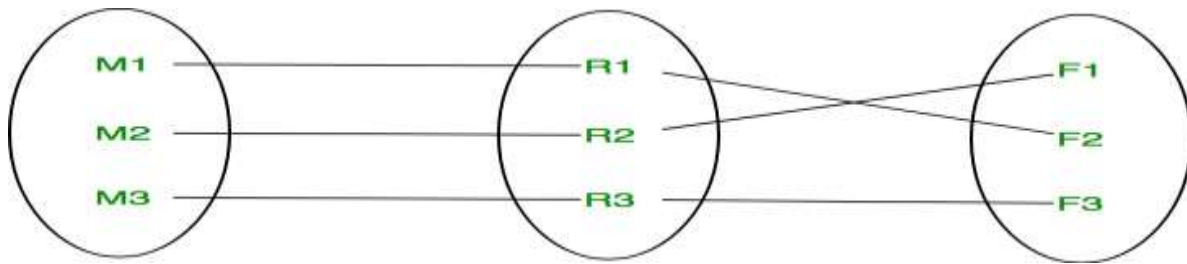
2. Many to one – When entities in one entity set **can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set,** cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.

Using Sets, it can be represented as:
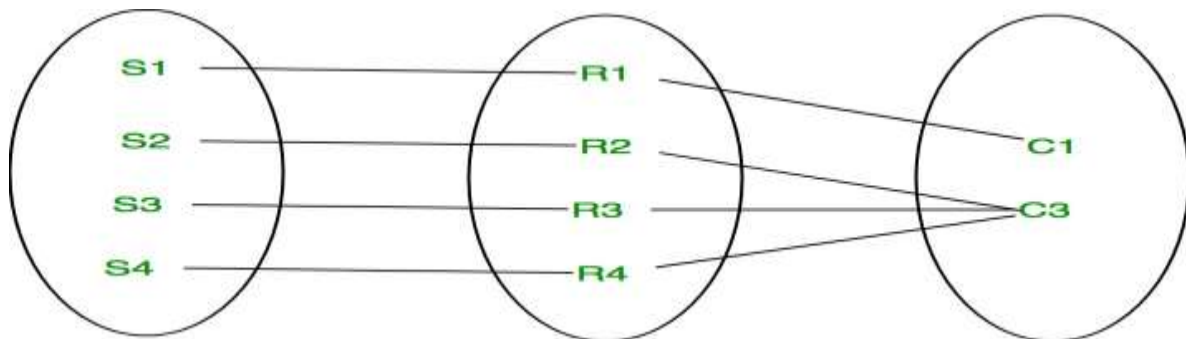
In this case, each student is taking only 1 course but 1 course has been taken by many students.

3. Many to many – When entities in all entity sets can **take part more than once in the relationship** cardinality is many to many. Let us assume that a student can take more than one

course and one course can be taken by many students. So the relationship will be many to many.

Using Sets, it can be represented as:
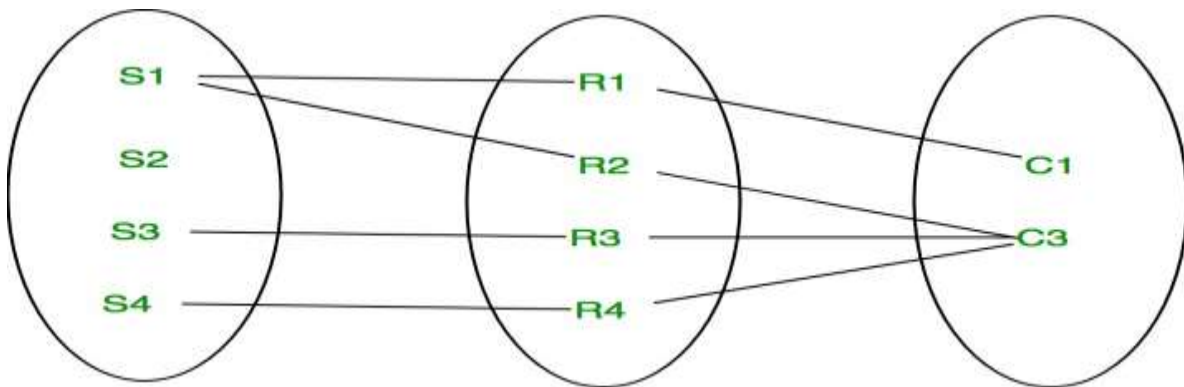
In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3 and S4. So it is many to many relationships.

Participation Constraint:

Participation Constraint is applied on the entity participating in the relationship set.

1.Total Participation – Each entity in the entity set **must participate** in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.

2.Partial Participation – The entity in the entity set **may or may NOT participat**e in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.

The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.

If the participation of an entity set in a relationship set is total, then a thick line connects the two. The presence of an arrow indicates a key constraint.
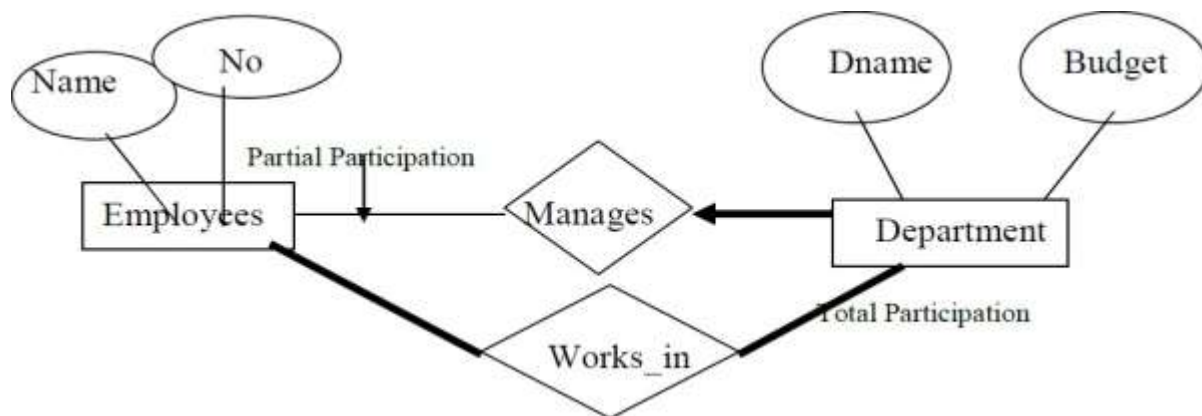
Weak Entity Type and Identifying Relationship:

An entity type has a key attribute which uniquely identifies each entity in the entity set. But there exists **some entity type for which key attribute can't be defined**. These are called Weak Entity type.

For example, A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents don't have existence without the employee. So Dependent will be weak entity type and Employee will be Identifying Entity type for Dependent.

A weak entity type is represented by a double rectangle. The participation of weak entity type is always total. The relationship between weak entity type and its identifying strong entity type is called identifying relationship and it is represented by double diamond. A weak entity type always has a **'total participation constraint'.**
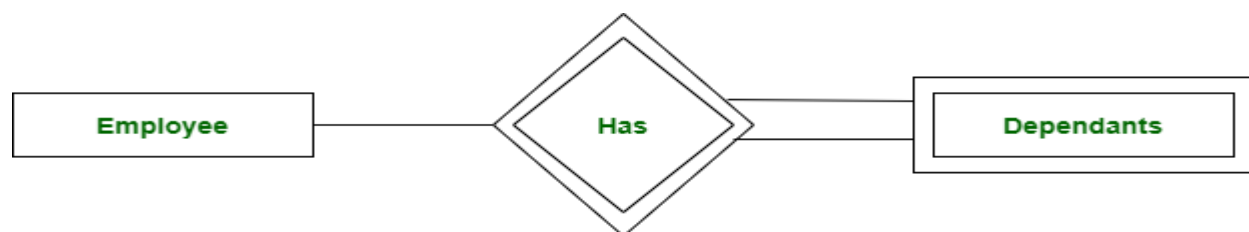
Class Hierarchies

Class hierarchy can be viewed one of two ways

- Specialization (Top Down Approach)
- Generalization (Bottom Up Approach)

Specialization

Specialization is a process of identifying subsets of an entity that shares different characteristics. It breaks an entity into multiple entities from higher level (super class) to lower level (subclass).

The class vehicle can be specialized into Car, Truck and Motorcycle (Top Down Approach)

Hence, vehicle is the **superclass** and Car, Truck, Motorcycle are **subclasses**. All three of these inherit attributes from vehicle. Moreover, these three share those attributes among themselves while containing some other attributes which make them different.
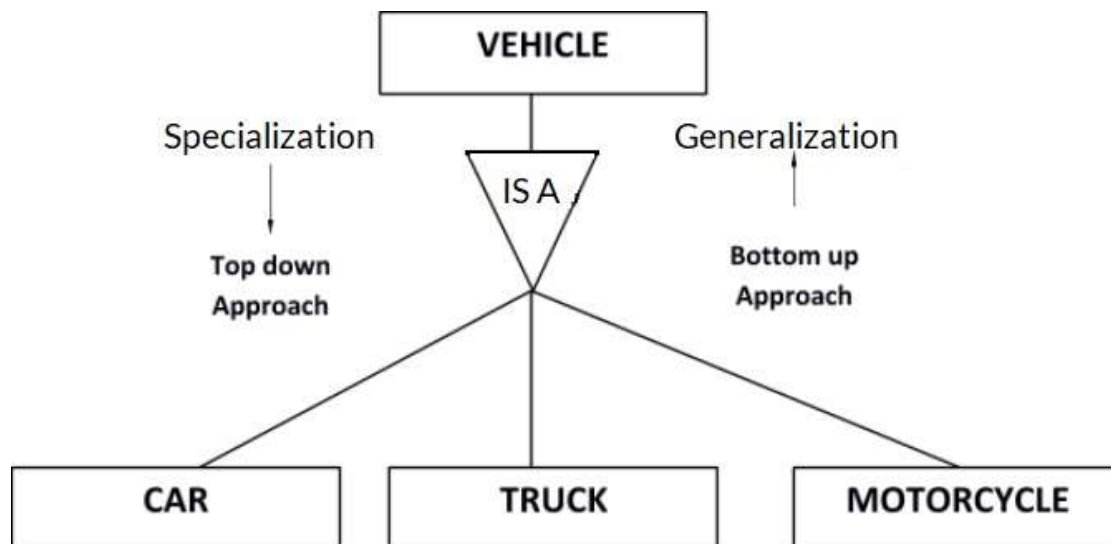
Generalization

Generalization is a process of generalizing an entity which contains generalized attributes or properties of generalized entities. The entity that is created will contain the common features. Generalization is a Bottom up process.

The classes Car, Truck and motorcycle can be generalised into Vehicle. (Bottom Up Approach). Car, Truck and Motorcycle are subclasses while vehicle is the superclass.

Basically, Vehicle contains the common attributes that were shared between Car, Truck and Motorcycle.

Aggregation

- Aggregation is an abstraction for building composite objects from their component objects.
- Aggregation is used to represent a relationship between a whole object and its component parts.
- Aggregation allows us to indicate that a relationship set (identified through a dashed box) participates in another relationship set.
- This is illustrated with a dashed box around sponsors.
- If we need to express a relationship among relationships, then we should use aggregation.
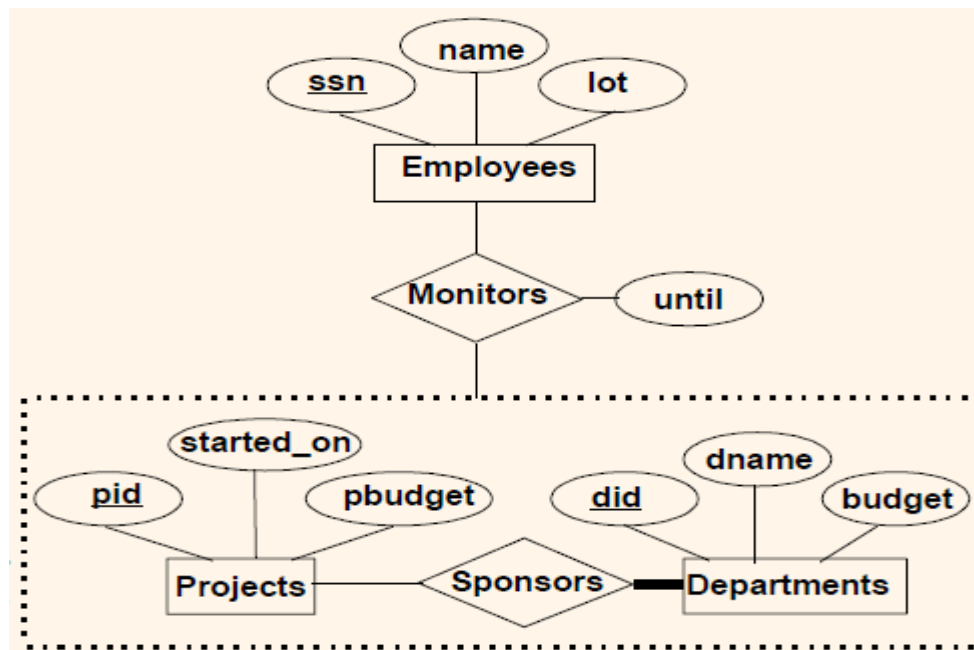
According to the above diagram,

1. A project can be sponsored by any number of departments.
2. A department can sponsor 1 or more projects.
3. 1 or more employees monitor each sponsorship.

(Many to Many Relationship)

Keys

o   Keys play an important role in the relational database.

o   It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

o   Keys also help to identify relationships uniquely, and thus distinguish relationships from each other.

o   The primary key of an entity set allows us to distinguish among the various entities of the set.

o   The structure of the primary key for the relationship set depends on the mapping cardinality of the   relationship set.

Types of Keys

1. **Superkey**: An attribute (or combination of attributes) that uniquely identifies each row in a table. It is a super set of candidate key.

2. **Candidate key** : An attribute (or set of attributes) that uniquely identifies a row. Let K be a set of attributes of relation R. Then K is a candidate key for R if it possess the following properties:

    1. Uniqueness – No legal value of R ever contains two distinct tuples with the same value of K

    2. Irreducibility- No proper subset of K has the uniqueness property

3. **Primary key** : is the candidate key which is selected as the principal unique identifier. It is a key that Primary Key   ıll entries.

s_id	S_name	age	course	address

4. **Composite Key:** Key that consist of two or more attributes that uniquely identifies an entity occurrence is called composite key.

5. **Foreign Key**: A foreign key is generally a primary key from one table that appears as a field in another where the first table has a relationship to the second.

In other words, if we had a table A with a primary key X that linked to a table B where X was a field in B, then X would be a foreign key in B.

studentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

Foreign Keys

Relationship

Primary Keys

courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course
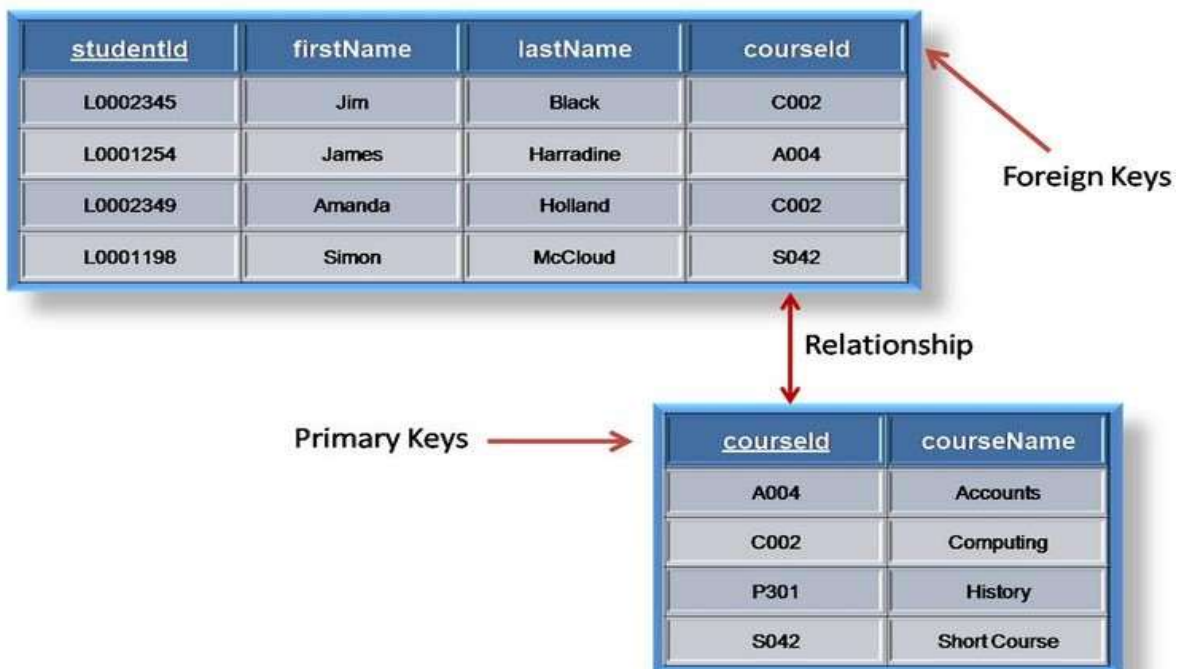
4. Removing Redundant Attributes in Entity Sets

- Once the entities and their corresponding attributes are chosen, the relationship sets among the various entities are formed.

- These relationship sets may result in a situation where attributes in the various entity sets are redundant and need to be removed .

- Consider the entity sets *instructor* and *department*
    - Instructor includes the attributes ID, name, de*pt_name, and salary with ID as primary key*
    - *Department includes the attributes dept_name, building, and budget with dept_name as primary key*

- Attribute dept_name appears in both entity sets. It is primary key for entity department, it is redundant in the entity set instructor and needs to be removed.

- If both entities have one to one relationship then we can remove it from instructor table as it will get added up in the relational schema of department.

- But if an instructor has more than one associated department, the relationship between the entities is recorded in a separate relation inst_dept

A good entity-relationship design does not contain redundant attributes. For example, the following are the entity sets and their attributes below, with primary keys underlined:

- **classroom**: with attributes (*building*, *room number*, *capacity*).
- **course**: with attributes (*course id*, *title*, *credits*).
- **instructor**: with attributes (*ID*, *name*, *salary*).

3.5   Reduction to Relational Schemas

For E-R model to be useful, need to be able to convert diagrams into an implementation schema

- Three components of conversion process:
    - Specify schema of the relation itself
    - Specify primary key on the relation
    - Specify any foreign key references to other relations
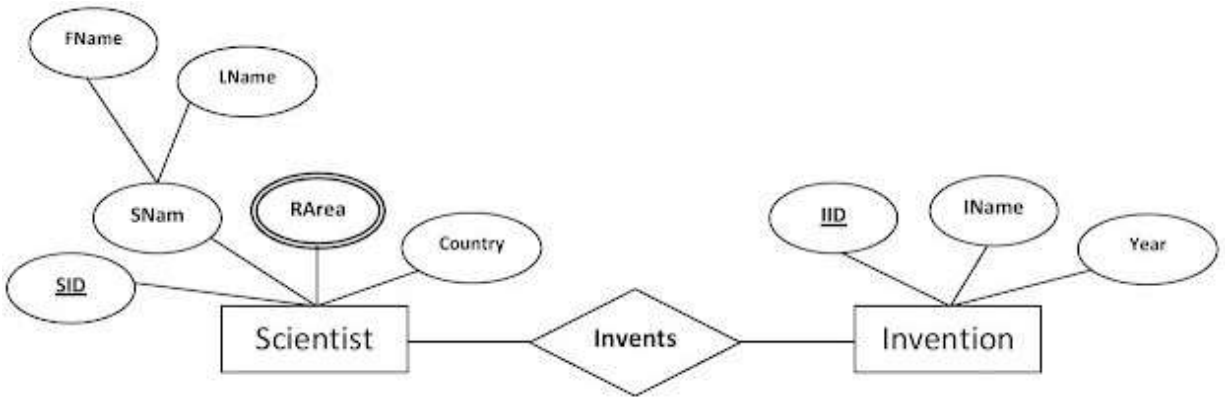
Representation of Strong Entity Sets with Simple Attributes
Given in the figure;

Entity sets and relationship sets

Name	Entity set / Relationship set	Type
Scientist	Entity set	Strong entity set
Invention	Entity set	Strong entity set
Invents	Relationship set	Many-to-Many relationship

Entity set *Scientist*

Attributes	Attribute Type	Description
SID	Simple and Primary key	Scientist ID
SNam	Composite	Scientist Name
RArea	Multi-valued	Research Area
Country	Simple	Country

Entity set *Invention*

Attributes	Attribute Type	Description
IID	Simple and Primary key	Invention ID
IName	Simple	Name of the invention
Year	Simple	Year of invention

Reduction into relational schema

Strong entity sets – *Entity set that has a primary key to uniquely represent each entity is Strong entity set.*

Strong entity sets can be converted into relational schema by having the entity set name as the relation schema name and the attributes of that entity set as the attributes of relation schema.

Then we have,

Scientist (SID, SName, RArea, Country)

Invention (IID, IName, Year)

1. After converting strong entity sets into relation schema
Scientist (SID, SName, RArea, Country)   Invention (IID, IName, Year)

Composite attributes – *If an attribute can be further divided into two or more component attributes, that attribute is called composite attribute.*

While converting into relation schemas, component attributes can be part of the strong entity sets' relation schema. No need to retain the composite attribute.

In our case, SNam becomes FName, and LName as follows;

Scientist (SID, FName, LName, RArea, Country)

2. After converting composite attributes into relation schema
Scientist (SID, FName, LName, RArea, Country)   Invention (IID, IName, Year)

Multi-valued attributes – *Attributes that may have multiple values are referred as multi-valued attributes.*

In our ER diagram, RArea is a multi-valued attribute. That means, a scientist may have one or more areas as their research areas.

To reduce a multi-valued attribute into a relation schema, we have to create a separate table for each multi-valued attribute. Also, we need to include the primary key of strong entity set (parent entity set where the multi-valued attribute belongs) as a foreign key attribute to establish link.

In our case, the strong entity set Scientist will be further divided as follows;

Scientist (<u>SID</u>, FName, LName, RArea, Country)

Scientist_Area (<u>SID, RArea</u>)

3. After converting multi-valued attributes into relation schema
Scientist (<u>SID</u>, FName, LName, Country)
Scientist_Area (<u>SID</u>, RArea)
Invention (<u>IID</u>, IName, Year)

Relationship set – *The association between two or more entity sets is termed as relationship set.* A relationship may be either converted into a separate table or not. That can be decided based on the type of the relationship. Only many-to-many relationship needs to be created as a separate table.

Here, we are given a **many-to-many relationship**. That means,

•**one entity** (record/row) of *Scientist* is related to **one or more** entities (records/rows) of *Invention* entity set (that is, one scientist may have one or more inventions) and,

•**one entity** (record/row) of *Invention* is related to **one or more** entities (records/rows) of *Scientist* entity set. (that is, one or more scientists may have invented one thing collectively).

To reduce the relationship Invents into relational schema, we need to create a separate table for *Invents*, because *Invents* is a **many-to-many** relationship set. Hence, create a table *Invents* with the primary keys of participating entity sets (both, Scientist and Invention) as the attributes.

Then we have,

<center>Invents (<u>SID, IID</u>)</center>

Here, SID and IID are both foreign keys and collectively forms the primary key of *Invents* table.

Finally, we have the following relation schemas;

4. After converting relationship sets into relation schema
Scientist (<u>SID</u>, FName, LName, Country)
Scientist_Area (<u>SID, RArea</u>)
Invention (<u>IID</u>, IName, Year)
Invents (<u>SID, IID</u>)

6.   Entity-Relationship Design Issues

The following are the ER Model design issues:

1. Use of Entity Sets versus Attributes
2. Use of Entity Sets versus Relationship Sets
3. Binary versus Entry Relationship Sets
4. Aggregation versus Ternary Relationship

Use of Entity Sets versus Attributes

Suppose the entity set employee with attributes employee-name and cell phone number. It can be dispute that a cell phone is an entity in its own right with attributes cell phone-number and location (the department where the cell phone is located). If we take this factor of view, the employee entity set has to be redefined as follows:

1. An employee entity set with attribute employee-name.

2. The phone entity set with attributes cell phone-number and location.

3. The relationship set emp-telephone, which indicates the relationship among employees and the cell phones that they have.

The first distinction between these two definitions of an employee is that each employee has exactly one cell phone number related to him. In the second case, however, the definition states that employees may additionally have various cell phone numbers (containing zero) connected with them.

Use of Entity Sets versus Relationship Sets

To simplify whether an item is first-class expressed by using an entity set or a relationship set, consider that a bank loan is modelled as an entity. An opportunity is to [model](#) a loan as a relationship between clients and departments, with loan-number and amount as descriptive attributes. Each loan is defined by a relationship between a client and a department.

If every loan is held via exactly one client and client is related with exactly one branch, the layout where a loan is defined as a relationship, perhaps satisfactory. However, with this design, we cannot describe conveniently a situation in which various clients keep a loan jointly. We have

to represent a separate relationship for each holder of the joint loan. Then, we must reflect the values for the definitive attributes loan-number and amount in each such relationship. Each such relationship must have a similar value for the specific attributes loan-number and amount.

Two problems appear as a result of the replication:

1. The information is saved multiple times, losing storage space.
2. Updates potentially leave the date in an inconsistent state, wherein the values differ in two relationships for attributes which can be speculated to have equal value.

Binary versus n-ary Relationship Sets

It is generally applicable to follow a number binary (n-ary, for n > 2) association set through various specific binary relationship sets. For integrity, assume the abstract ternary (n = 3) relationship set R, combining entity sets A, B and C. We replace the relation set R by an entity set E, and generate three relationship sets:

- R_(A) connecting E and A
- R_(B) connecting E and B
- R_(C) connecting E and C

If the relationship set R had any attributes, these are created to entity set E; otherwise, a special identifying attribute is developed for E (since each entity set should have at least one attribute to distinguish participants of the set). For each relationship (a_i, b_i, c_i) in the relationship set R, we develop a new entity? e?_i in an entity set E. Then, in every of the three new relationship sets, we add a relationship as follows:

- (e_i, a_i) in R_A
- (e_i, b_i) in R_B
- (e_i, c_i) in R_C

We can accurately achieve this technique n-ary relationship sets. Thus, theoretically, we can limit the E-R model to involve only binary relationship sets.

Aggregation versus Ternary Relationships

The option among utilizing aggregation a ternary relationship is primarily distinct by the existence of a relationship that associates a relationship set to an entity set (or second relationship set). The option may also be guided using certain integrity constraints to we need to define.

Consider the constraint that every sponsorship (of a task by a department) be monitored by at maximum one employee. We cannot define this constraint in phrases of the Sponsors2 relationship set. Also, we can get explicit the constraint by drawing an arrow from the aggregated relationship. Sponsors to the relationship Monitors. Thus, the display of such a constraint serves as another purpose for the usage of aggregation as opposed to a ternary relationship set.

Relational Database Design

Relational databases differ from other databases in their approach to organizing data and performing transactions. In an RDD, the data are organized into tables and all types of data access are carried out via controlled transactions. Relational database design satisfies the ACID (atomicity, consistency, integrity and durability) properties required from a database design. Relational database design mandates the use of a database server in applications for dealing with data management problems.

The four stages of an RDD are as follows:

- **Relations and attributes**: The various tables and attributes related to each table are identified. The tables represent entities, and the attributes represent the properties of the respective entities.

- **Primary keys**: The attribute or set of attributes that help in uniquely identifying a record is identified and assigned as the primary key

- **Relationships**: The relationships between the various tables are established with the help of foreign keys. Foreign keys are attributes occurring in a table that are primary keys of another table. The types of relationships that can exist between the relations (tables) are:

    o One to one

    o One to many

    o Many to many

An entity-relationship diagram can be used to depict the entities, their attributes and the relationship between the entities in a diagrammatic way.

- **Normalization:** This is the process of optimizing the database structure. Normalization simplifies the database design to avoid redundancy and confusion. The different normal forms are as follows:

    o First normal form
    o Second normal form
    o Third normal form
    o Boyce-Codd normal form
    o Fifth normal form

By applying a set of rules, a table is normalized into the above normal forms in a linearly progressive fashion. The efficiency of the design gets better with each higher degree of normalization.

7. Features of good relational design

Good Database Design is what everyone wants to achieve to avoid the consequences of dealing with a bad design. The following are the objectives of a good database design –

Avoid Redundant Data
The table in the database should be constructed following standards and with utmost dedication. It should have different fields and minimize redundant data. The table should always have a Primary Key that would be a unique id.

Faultless Information
The database should follow the standards and conventions and provide meaningful information useful to the organization.

Data Integrity
Integrity assists in guaranteeing that the values are valid and faultless. Data Integrity is set to tables, relationships, etc.

Modify
The database developed should be worked upon with the conventions and standards, so that it can be easily modified whenever the need arise.

8. Functional Dependency

Functional dependency is a constraint between two sets of attributes in a relation from a database. In other words, functional dependency is a constraint that describes the relationship between attributes in a relation.

A functional dependency (FD) is a kind of IC that generalizes the concept of a key. 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 an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

In other words, a dependency FD: $X \rightarrow Y$ means that the values of Y are determined by the values of X. Two tuples sharing the same values of X will necessarily have the same values of Y.

Let R be a relation schema and let X and Y be nonempty sets of attributes in R. We say that an instance r of R satisfies the FD X → Y if the following holds for every pair of tuples **t1** and **t2** in r

If t1:X = t2:X,   then   t1:Y = t2:Y .

The following table illustrates A → B

A	B
1	1
2	4
3	9
4	16
2	4
7	9

Since for each value of A there is associated one and only one value of B.

The following table illustrates that A does not functionally determine B:

A	B
1	1
2	4
3	9
4	16
3	10

The following illustrates the meaning of the FD AB → C by showing an instance that satisfies this dependency.

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

An Instance that Satisfies $AB \rightarrow C$

Ex : <Department> table with two attributes − **DeptId** and **DeptName**.

 The **DeptId** is our primary key. Here, **DeptId** uniquely identifies the **DeptName** attribute. This is because if you want to know the department name, then at first you need to have the **DeptId**.

DeptId	DeptName
001	Finance
002	Marketing
003	HR

Therefore, the above functional dependency between **DeptId** and **DeptName** can be determined as **DeptId** is functionally dependent on **DeptName**. i.e **DeptId -> DeptName**

Types of Functional Dependency

Functional Dependency has **three** forms −

- **Trivial Functional Dependency**
- **Non-Trivial Functional Dependency**
- **Completely Non-Trivial Functional Dependency**

Trivial Functional Dependency

Every dependent in trivial functional Dependency is a subset of the determinant. To put it another way, a functional relationship is said to be simple if its right-side characteristics are a subset of its left-side attributes. Right-side characteristics are a subset of its left-side attributes.

If Y is a subset of X, the functional relationship X->Y is referred to as trivial.

Example

Consider **<Department>** table with two attributes to understand the concept of trivial dependency.

The following is a trivial functional dependency since **DeptId** is a subset of **DeptId** and **DeptName**

{ DeptId, DeptName } -> Dept Id

Non –Trivial Functional Dependency

The trivial functional Dependency in DBMS is opposed by it. Formally speaking, dependent if not a subset of the determinant in Non-Trivial functional Dependency.

If Y is not a subset of X, the relationship between X and Y is said to be non-trivial functional.

Example

DeptId -> DeptName

The above is a non-trivial functional dependency since DeptName is a not a subset of DeptId.

Completely Non - Trivial Functional Dependency

It occurs when X intersection Y is null in − **X ->Y**

Armstrong's Axioms Property of Functional Dependency

Armstrong axioms are a complete set of inference rules or axioms, introduced and developed by [William W. Armstrong](#) in 1974. The inference rules are sound which is used to test **logical inferences of functional dependencies**. The Axioms are a set of rules, that when applied to a specific set, generates a closure of functional dependencies. They are called **RAT** rules.

A. Primary Rules

Rule 1	**Reflexivity** If A is a set of attributes and B is a subset of A, then A holds B. { A → B } *For example*, **{ Employee_Id, Name } → Name** is valid.
Rule 2	**Augmentation** If A hold B and C is a set of attributes, then AC holds BC. {AC → BC} It means that attribute in dependencies does not change the basic dependencies. *For example,* **X → Y** holds true then, **ZX → ZY** also holds true. *For example,* if **{ Employee_Id, Name } → { Name }** holds true then, **{ Employee_Id, Name, Age } → { Name, Age }**
Rule 3	**Transitivity** If A holds B and B holds C, then A holds C. If {A → B} and {B → C}, then {A → C} A holds B {A → B} means that A functionally determines B. *For example,* if **{ Employee_Id } → { Name }** holds true and **{ Name } → { Department }** holds true, then **{ Employee_Id } → { Department }** also holds true.

B. Secondary Rules

Rule 1	**Union** If A holds B and A holds C, then A holds BC. If{A → B} and {A → C}, then {A → BC}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If{A → BC} and {A → B}, then {A → C}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If{A → B} and {BC → D}, then {AC → D}

Example:
Consider relation E = (P, Q, R, S, T, U) having set of Functional Dependencies (FD).

$P \rightarrow Q$ $P \rightarrow R$
$QR \rightarrow S$ $Q \rightarrow T$
$QR \rightarrow U$ $PR \rightarrow U$

Calculate some members of Axioms are as follows,
1. $P \rightarrow T$
2. $PR \rightarrow S$
3. $QR \rightarrow SU$
4. $PR \rightarrow SU$

Solution:

1. P → T

In the above FD set, $P \rightarrow Q$ and $Q \rightarrow T$

So, Using Transitive Rule: If {A → B} and {B → C}, then {A → C}

∴ If $P \rightarrow Q$ and $Q \rightarrow T$, then **P → T.**

$$\boxed{P \rightarrow T}$$

2. PR → S

In the above FD set, $P \rightarrow Q$

As, $QR \rightarrow S$

So, Using Pseudo Transitivity Rule: If{A → B} and {BC → D}, then {AC → D}

∴ If $P \rightarrow Q$ and $QR \rightarrow S$, **then PR → S.**

PR → S

3. QR → SU

In above FD set, QR → S and QR → U

So, Using Union Rule: If{A → B} and {A → C}, then {A → BC}

∴ If QR → S and QR → U, **then QR → SU.**

<mark>QR → SU</mark>

4. PR → SU

So, Using Pseudo Transitivity Rule: If{A → B} and {BC → D}, then {AC → D}

∴ If PR → S and PR → U, **then PR → SU.**

<mark>PR → SU</mark>

3.9 Closure of an Attribute Set

- The set of all those attributes which can be functionally determined from an attribute set

  is called as a closure of that attribute set.

- Closure of attribute set {X} is denoted as {X}$^+$.

Steps to Find Closure of an Attribute Set-

Following steps are followed to find the closure of an attribute set-

Step-01:

Add the attributes contained in the attribute set for which closure is being calculated to the result set.

Step-02:

Recursively add the attributes to the result set which can be functionally determined from the attributes already contained in the result set.

Example-

Consider a relation R (A , B , C , D , E , F , G) with the functional dependencies-

$$A → BC$$

$$BC → DE$$

$$D → F$$

$$CF → G$$

Now, let us find the closure of some attributes and attribute sets-

Closure of attribute A-

$A^+ = \{\ A\ \}$

$= \{\ A\ ,\ B\ ,\ C\ \}$ (Using $A \rightarrow BC$)

$= \{\ A\ ,\ B\ ,\ C\ ,\ D\ ,\ E\ \}$ (Using $BC \rightarrow DE$)

$= \{\ A\ ,\ B\ ,\ C\ ,\ D\ ,\ E\ ,\ F\ \}$ (Using $D \rightarrow F$)

$= \{\ A\ ,\ B\ ,\ C\ ,\ D\ ,\ E\ ,\ F\ ,\ G\ \}$ (Using $CF \rightarrow G$)

Thus, $\quad A^+ = \{\ A\ ,\ B\ ,\ C\ ,\ D\ ,\ E\ ,\ F\ ,\ G\ \}$

Closure of attribute D-

$D+ = \{\ D\ \}$

$= \{\ D\ ,\ F\ \}$ (Using $D \rightarrow F$)

We can not determine any other attribute using attributes D and F contained in the result set.

Thus, $\quad D^+ = \{\ D\ ,\ F\ \}$

Closure of attribute set {B, C}-

$\{\ B\ ,\ C\ \}+ = \{\ B\ ,\ C\ \}$

$= \{\ B\ ,\ C\ ,\ D\ ,\ E\ \}$ (Using $BC \rightarrow DE$)

$= \{\ B\ ,\ C\ ,\ D\ ,\ E\ ,\ F\ \}$ (Using $D \rightarrow F$)

$= \{\ B\ ,\ C\ ,\ D\ ,\ E\ ,\ F\ ,\ G\ \}$ (Using $CF \rightarrow G$)

Thus, $\{\ B\ ,\ C\ \}^+ = \{\ B\ ,\ C\ ,\ D\ ,\ E\ ,\ F\ ,\ G\ \}$

Problem-

Consider the given functional dependencies-

$$AB \rightarrow CD$$

$$AF \rightarrow D$$

$$DE \rightarrow F$$

$$C \rightarrow G$$

$$F \rightarrow E$$

$$G \rightarrow A$$

Which of the following options is false?

(A) { CF }$^+$ = { A , C , D , E , F , G }

(B) { BG }$^+$ = { A , B , C , D , G }

(C) { AF }$^+$ = { A , C , D , E , F , G }

(D) { AB }$^+$ = { A , C , D , F ,G }

Solution-

Let us check each option one by one-

Option-(A):

{ CF }$^+$ = { C , F }

   = { C , F , G } (Using C → G)

   = { C , E , F , G } (Using F → E)

   = { A , C , E , E , F } (Using G → A)

   = { A , C , D , E , F , G } (Using AF → D)

Since, our obtained result set is same as the given result set, so, it means it is correctly given.

Option-(B):

{ BG }$^+$ = { B , G }

= { A , B , G } (Using G → A)

= { A , B , C , D , G } (Using AB → CD)

Since, our obtained result set is same as the given result set, so, it means it is correctly given.

Option-(C):

{ AF }$^+$ = { A , F }

= { A , D , F } (Using AF → D)

= { A , D , E , F } (Using F → E)

Since, our obtained result set is different from the given result set, so,it means it is not correctly given.

Option-(D):

{ AB }$^+$ = { A , B }   = { A , B , C , D } (Using AB → CD)

   = { A , B , C , D , G } (Using C → G)

Problem
Given R(<u>E-ID</u>, E-NAME, E-CITY, E-STATE)
FDs = { E-ID->E-NAME, E-ID->E-CITY, E-ID->E-STATE, E-CITY->E-STATE }
The attribute closure of E-ID can be calculated as:

1. Add E-ID to the set {E-ID}
2. Add Attributes which can be derived from any attribute of set.
   In this case, E-NAME and E-CITY, E-STATE can be derived from E-ID. So these are also a part of closure.
3. As there is one other attribute remaining in relation to be derived from E-ID. So result is:

$(E\text{-}ID)^+$ = {E-ID, E-NAME, E-CITY, E-STATE }
Similarly,

$(E\text{-}NAME)^+$ = {E-NAME}
$(E\text{-}CITY)^+$ = {E-CITY, E_STATE}

Finding the Keys Using Closure-
Super Key-

If the closure result of an attribute set contains all the attributes of the relation, then that attribute set is called as a super key of that relation.

Thus, we can say - **"The closure of a super key is the entire relation schema."**

Example-

In the above example, The closure of attribute A is the entire relation schema.

- Thus, attribute A is a super key for that relation.

Candidate Key-

If there exists no subset of an attribute set whose closure contains all the attributes of the relation, then that attribute set is called as a candidate key of that relation.

Example-

In the above example,

- No subset of attribute A contains all the attributes of the relation.
- Thus, attribute A is also a candidate key for that relation.

Canonical Cover

In any relational model, there exists a set of functional dependencies. These functional dependencies when closely observed might contain redundant attributes. The ability of removing these redundant attributes without affecting the capabilities of the functional dependency is known as "**Canonical Cover of Functional Dependency**".

- Canonical cover of functional dependency is sometimes also referred to as "**Minimal Cover**". There are **three** steps to calculate the canonical cover for a relational schema having set of functional dependencies.

  Step-1 : Decompose the functional dependencies using Decomposition rule(Armstrong's Axiom) i.e. single attribute on right hand side.
  Step-2 : Remove extraneous attributes from LHS of functional dependencies by calculating the closure of FD's having two or more attributes on LHS.

  Step-3 : Remove FD's having transitivity

- Canonical cover of functional dependency is denoted using "M_c".

EX : Consider a relation R(A,B,C,D) having some attributes and below are mentioned functional dependencies.

        FD1 : B A

        FD2 : AD C

        FD3 : C ABD

Step-1 : Decompose the functional dependencies using Decomposition rule(Armstrong's Axiom) i.e. single attribute on right hand side.

        FD1 : B A

        FD2 : AD C

        FD3 : C A

        FD4 : C B

        FD5 : C D

Step-2 : Remove extraneous attributes from LHS of functional dependencies by calculating the closure of FD's having two or more attributes on LHS.

Here, only one FD has two or more attributes of LHS i.e. AD C.

$$\{A\}+ = \{A\}$$

$$\{D\}+ = \{D\}$$

In this case, attribute "A" can only determine "A" and "D" can only determine "D". Hence, no extraneous attributes are present and the FD will remain the same and will not be removed.

Step-3 : Remove FD's having transitivity.

FD1 : B A

FD2 : C A

FD3 : C B

FD4 : AD C

FD5 : C D

Above FD1, FD2 and FD3 are forming transitive pair. Hence, using Armstrong's law of transitivity i.e. if X Y, Y X then X Z should be removed. Therefore we will have the following FD's left :

FD1 : B A

FD2 : C B

FD3 : AD C

FD4 : C D

Also, FD2 & FD4 can be clubbed together now. Hence, the canonical cover of the relation R(A,B,C,D) will be:

Mc {R(ABCD)} = {B A , C BD, AD C}

NORMAL **FORMS**

Normalization is a process of evaluating and organizing the data in database to avoid data redundancy and inconsistency of the data.

It is the process of decomposing the relations with anomaly to produce smaller, well structure relations

It is a formal process for describing which attributes should be group together in a relation based on the concept of determination

WHY WE NEED NORMALIZATION?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure. Data modification anomalies can be categorized into **three types:**

- o **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- o **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- o **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

ADVANTAGES OF NORMALIZATION

- o Normalization helps to minimize data redundancy.
- o Greater overall database organization.
- o Data consistency within the database.
- o Much more flexible database design.
- o Enforces the concept of relational integrity.

DISADVANTAGES OF NORMALIZATION

- You cannot start building the database before you know what the user needs.
- On Normalizing the relations to higher normal forms i.e. 4NF, 5NF the performance degrades.
- It is very time consuming and difficult process in normalizing relations of higher degree.
- Careless decomposition may leads to bad design of database which may leads to serious problems.

Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

Following are the various types of Normal forms:

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.
- No Partial dependency exists between non key attributes and key attributes

An attribute that is not part of any candidate key is known as non-prime attribute.

Note : Every Non key attribute is fully functional dependant on key attribute.

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}   **Non prime attribute**: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "**no** non-prime attribute is dependent on the proper subset of any candidate key of the table".

To make the table complies with 2NF we can break it in two tables like this:
teacher_details table:

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables completely with Second normal form (2NF).

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on
Candidate Keys: {emp_id}
Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Boyce Codd normal form (BCNF)

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:
emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.