Conceptual Design

- E-R model: entities, attributes and its types, Relationship, Relationship sets, Generalization, Specialization, Aggregation
- Relational Model: Relation, Domain, Tuple, Degree, cardinality
- Relational Algebra operations: Select, Project, Cartesian Product, Union, Set difference, join

****

Entity-relationship (E-R) model

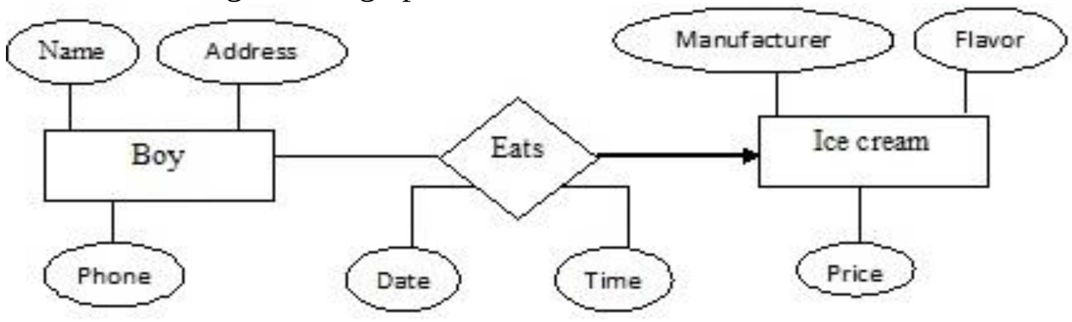
ER Model is a high-level data model, developed by Chen in 1976. This model defines the entities, relationships and their association with attributes for a specified system. It is useful in developing a conceptual design for the database & is very simple and easy to design logical view of data. The E-R model is mainly used for communication between database designers and end users during the analysis phase of database development.

Importance of ER Model:

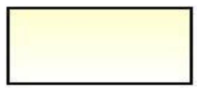
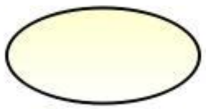
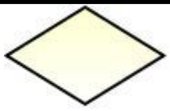
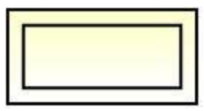
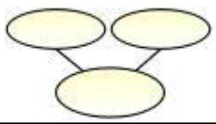
- ER Model is plain and simple for designing the structure.
- It saves time.
- Without ER diagrams you cannot make a database structure & write production code.
- It displays the clear picture of the database structure.

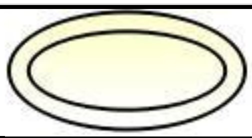
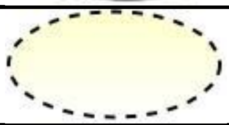
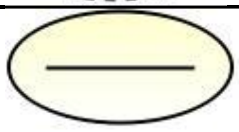
ER Diagrams

- ERD stands for Entity Relationship diagram.
- It is a graphical representation of an information system.
- ER diagram shows the relationship between objects, places, people, events etc. within system.
- It is a data modeling technique which helps in defining the business process.
- It used for solving the design problems.



Notations/Symbols used in E-R diagrams

Notations	Representation	Description
	Rectangle	It represents the Entity.
	Ellipse	It represents the Attribute.
	Diamond	It represents the Relationship.
	Line	It represents the link between attribute and entity set to relationship set.
	Double Rectangle	It represents the weak entity.
	Composite Attribute	It represents composite attribute which can be divided into subparts.

	Multi valued Attribute	It represents multi valued attribute which can have many values for a particular entity. For eg. Mobile Number.
	Derived Attribute	It represents the derived attribute which can be derived from the value of related attribute.
	Key Attribute	It represents key attribute of an entity which have a unique value in a table. eg. Employee →EmpId (Employee Id is Unique).

Entities and Entity Sets

- <u>An entity is a real world object that exists and it is distinguishable from other entities.</u> A database can be modeled as a collection of entities. Example-Person, Bank, company, Department, Customer, Employee, Student, event, plant etc.
- An entity has a set of properties and some properties have the values. It may uniquely identify an entity. Example, a *Customer* with *Customer_id* property with *value C101* uniquely identifies that person.
- An entity may be concrete (real) such as Customer, book or it may be abstract (Conceptual) such as a loan, or a holiday.
- <u>A set of entities of the same type that share the same properties or attributes is called as **Entity set.**</u> The set of all customers at a given bank can be defined as the entity set *Customer*. Two entity sets:
  - *Customer,* with properties Customer_id, Customer_name, City
  - *Account* with properties Account_no and Balance.

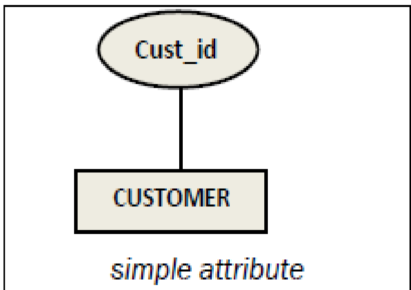
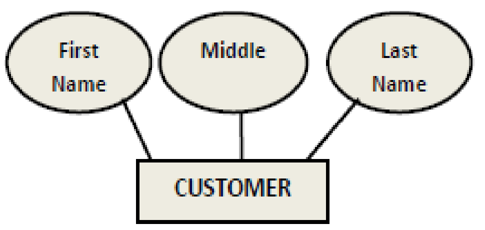
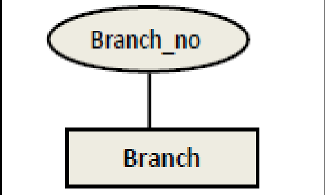
Attributes and its types

<u>All the entities in the data model have associated with it a set of properties of entities is called as **attributes**.</u> Example: *Customer* has Cust_*id,Cust_name and address.*
<u>There are types of attributes has been classified Such as</u>

Simple attribute:
- A simple attribute is an attribute collectedfrom a single component with an independent existence. Simple attributes cannot be further subdivided.
- Example*:* Customer's Cust_id, Students Roll_No etc.

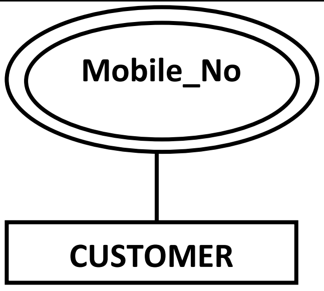
Composite attribute:
- Composite attributes are composed of more than one simple attribute; each attribute with its own independent existence is called a composite attribute.
- Example*:* A *Cust_name* attributes of a Customer entity that can be further divided in to subparts i.e. *first name, middle name and last name*.

Single-valued attribute:
- Single valued attribute is an attribute which as single value (atomic) for each entityOR An attribute that holds a single value for each single entity type
- Example*:* Each *branch* has only single valued attributes is known as *branch_no*

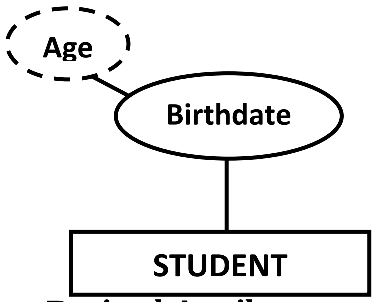
 simple attribute	 Composite attributes	 Single valued attribute

Multi-valued attribute:
- Multi-value attribute may contain more than one value. OR An attribute that holds multiple

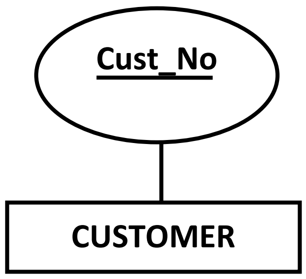
values for each single entity type
- Example: a Customer can have more than one phone numbers, hobbies etc.



Multi-Valued Attribute



Derived Attribute



Key Attribute

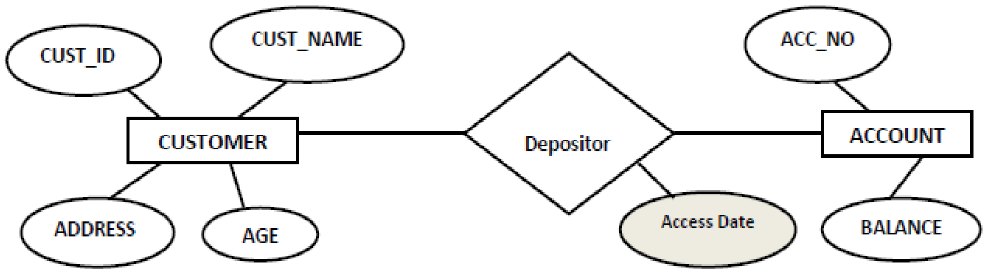
⊙ **Derived attribute:**
- In Derived attributes the value is derived from the other stored attribute. Derived attributes are attributes, which do not exist physical in the database, but there values are derived from other attributes presented in the database.
- *Example:* Age can be derived from Date of Birth(DOB).

⊙ **Stored attribute:**
- An attribute whose value cannot be derived from the values of other attributes is called a stored attribute. *Example:* Date of Birth (DOB).

⊙ **Null attribute**
- The attribute which take NULL value when entity does not have the value to it. The Null attribute is an attribute their value is unknown, unassigned and missing information.

⊙ **Key Attributes**
- This attribute has the unique value for an entity which is used to identified given row in the table is called as key attribute of an entity
- Example : Cust_id is an key attribute which has an unique value which is used to identifies given row in the table

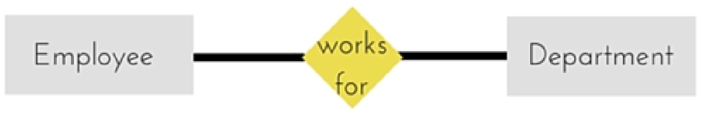
⊙ **Descriptive attributes**
- A relationship may also have attributes called descriptive attributes.
- Consider a relationship set *Depositor* with entity sets *Customer and Account*. We can associate attribute *access-date* to the *depositor* relationship to specify the most recent date on which a customer accessed an account as shown in Fig.



Relationship , Relationship sets, Degree and Cardinality

➢ **Relationships**

The association among entities is called relationship. A set of meaningful relationships between several entities. We used *diamond symbol* for indicate to Relationships among the several entities; it could read from left to right.
Example: Employee entity has relation *works for* with *department.*



➢ **Degree of relationship :** *Total number of entity sets participate in a relationship set is known as degree of relationship.*

Types of relationships

i) Unary relationship: *A unary relationship exists when an association is maintained within a single entity. Example,* boss and worker distinguish the two employees participating in the manage association as shown in following Fig.

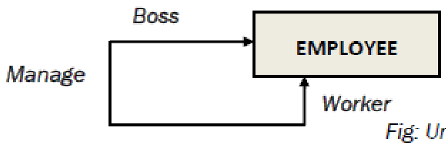


Fig: Unary Relationship

ii) Binary relationship: *A binary relationship exists when two entities are associated. Example:* the Book-Publisher relationship shown in Fig.

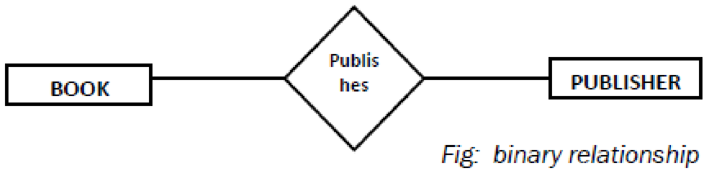
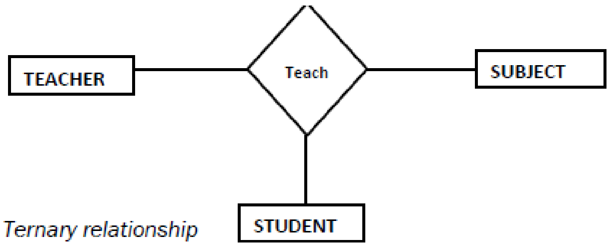


Fig: binary relationship

iii) Ternary relationship: *A ternary relationship exists when there are three entities associated. For example*, the entities *Teacher*, *Subject* and *Student* are related using a ternary relationship called '*Teaches*' as shown in Fig.



Ternary relationship

iv) Quaternary relationship: *A quaternary relationship exists when there are four entities associated.* Example : *'studies'* where four entities are involved *Student, Teacher, Subject and Course-material.* It is shown in Fig.

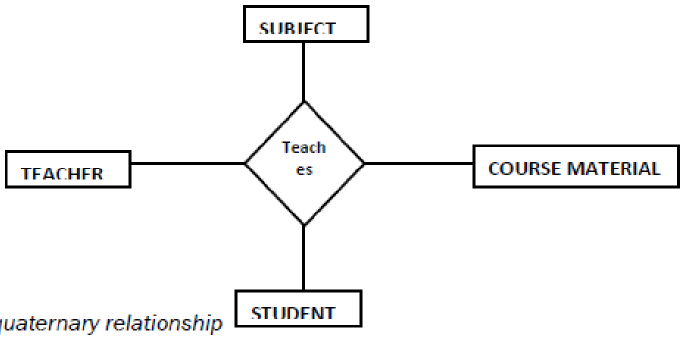
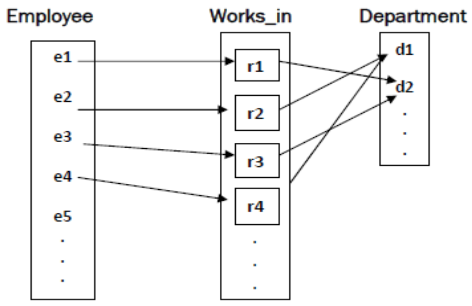


Fig.: quaternary relationship

- **Relationship Set:** *A relationship set is a set of relationships of the same type. Consider an example*, employees work in different departments. Then *relationship* exists between *employees* and *departments* because each employee must belong to some department. Relation of all *employees* with *department* when combined makes the relationship set because each employee has same kind of relation with departments.



Here, Employee and Department are two entity sets, *r stands for relationship* between *Employee and Department*. *Works_in* is the *relationship set* as shown in Figure.

- **Mapping Constraints**

There are *two types* of mapping constraints:
(a) Mapping cardinalities
(b) Participation constraints

(a) Mapping Cardinalities (Cardinality Ratios)

- *The numbers of entities of an entity set that are associated with entities of another entity set through a relationship set.*
- *Cardinalities indicates that a specific number of entity occurrence of related entity* .Types of Relationship Mapping

Following are the types of Relationship Mapping,
1. One - to - One Relationship
2. One - to - Many Relationships
3. Many - to - One Relationships
4. Many - to - Many Relationships

1. One - to - One Relationship

- In One - to - One Relationship, one entity is related with only one other entity.
- One row in a table is linked with only one row in another table and vice versa.
  Example: A Country can have only one Capital City.

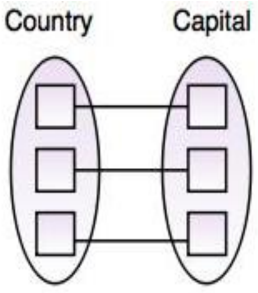


Fig. One to One Mapping

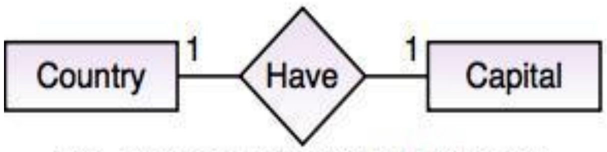


Fig. Representation in ER Diagram

2. One - to - Many Relationship

- In One - to - Many Relationship, one entity is related to many other entities.
- One row in a table A is linked to many rows in a table B, but one row in a table B is linked to only one row in table A. **Example:** One Department has many Employees.

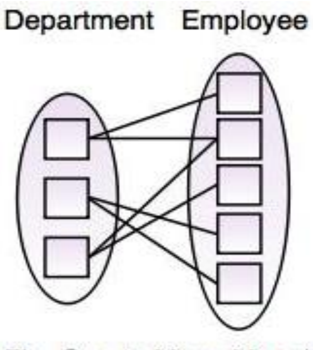


Fig. One to Many Mapping

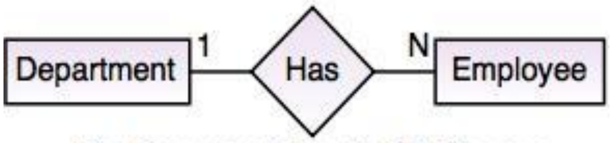


Fig. Representation in ER Diagram

3. Many - to - One Relationship

- In Many - to - One Relationship, many entities can be related with only one other entity.
  For example: No. of Employee works for Department.
- Multiple rows in Employee table are related with only one row in Department table.

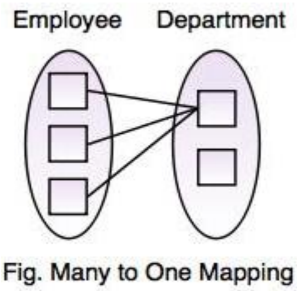


Fig. Many to One Mapping

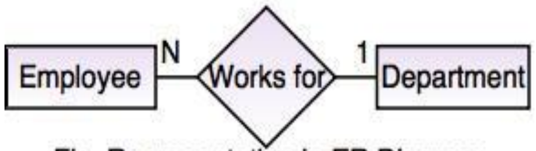


Fig. Representation in ER Diagram

4. Many - to - Many Relationship

- In Many - to - Many Relationship, many entities are related with the multiple other entities.
- This relationship is a type of cardinality which refers the relation between two entities. **Example:** Various Books in a Library are issued by many Students.

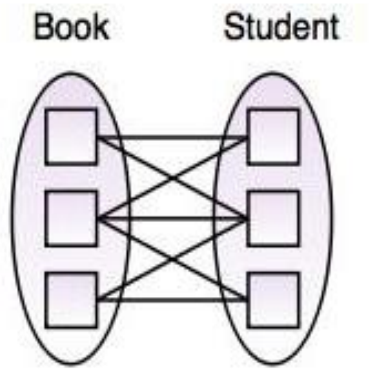


Fig. Many to Many Mapping

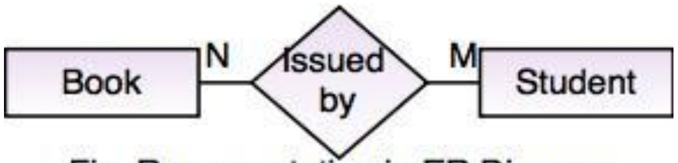


Fig. Representation in ER Diagram

(b)Participation Constraints

Participation concerns with the involvement of entities in a relationship. It specifies whether the existence of an entity depends on another entity. There are two types of Participation Constraints –

1. Total Participation
2. Partial Participation

1. Total Participation

- In Total Participation, every entity in the set is involved in some association of the relationship.
- It is indicated by a double line (════) between entity and relationship.

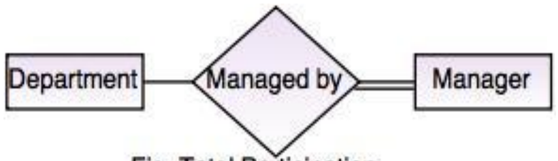


Fig. Total Participation

Example: Every Department must have a Manager.

2. Partial Participation

- In Partial Participation, not all entities in the set are involved in association of the relationship.
- It is indicated by a single line (────) between entity and relationship.

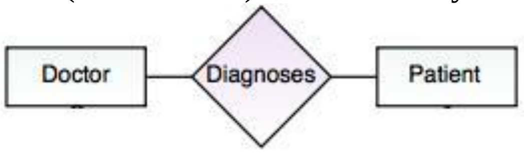


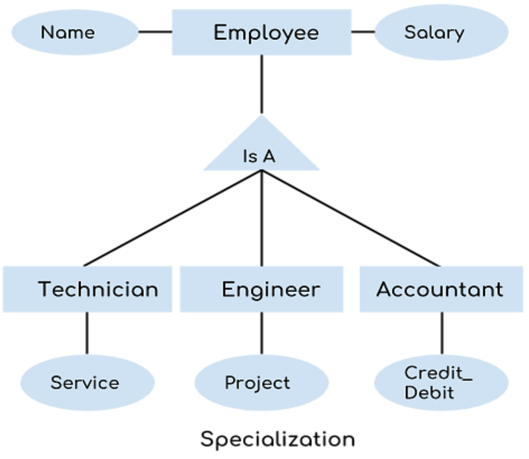
Fig. Partial Participation

Extended Entity Relationship Model (EER Model)

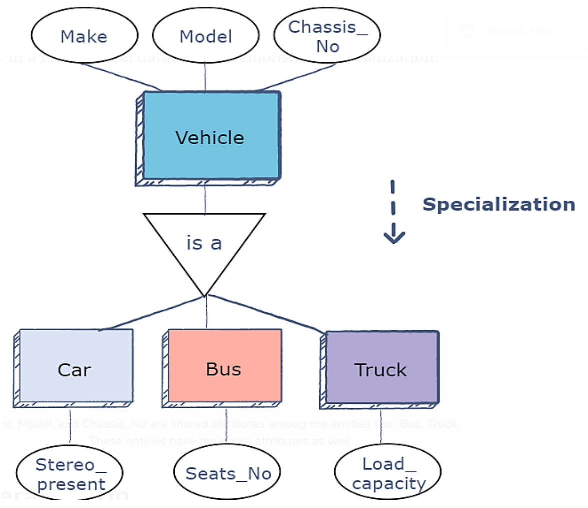
- EER is a high-level data model that incorporates the extensions to the original ER model. It includes all modeling concepts of the ER model.
- EER creates a design more accurate to database schemas.
- It reflects the data properties and constraints more precisely.
- Diagrammatic technique helps for displaying the EER schema.
- It includes the concept of specialization and generalization.
- It is used to represent a collection of objects that is union of objects of different of different entity types.

➢ Specialization

Specialization is a top-down approach in which a higher-level entity is divided into multiple *specialized* lower-level entities. In addition to sharing the attributes of the higher-level entity, these lower-level entities have *specific* attributes of their own. Specialization is used to find subsets of an entity that has a few different or additional attributes.



Specialization

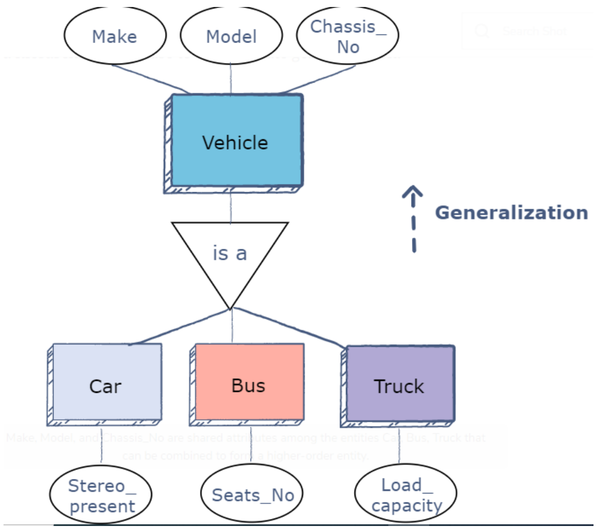


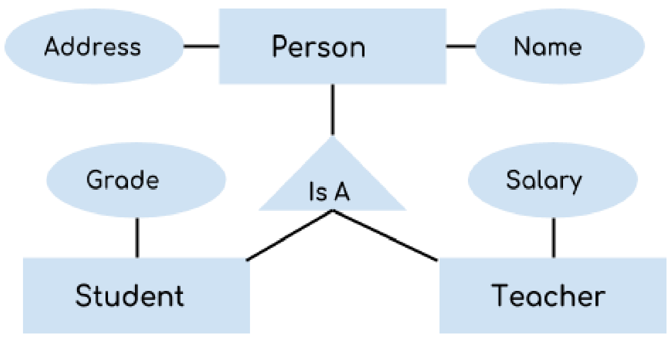
Example – Consider an entity employee which can be further classified as sub-entities Technician, Engineer & Accountant because these sub entities have some distinguish attributes. We have a higher level entity "Employee" which we have divided in sub entities "Technician", "Engineer" & "Accountant". I have shown that Technician handles service requests, Engineer works on a project and Accountant handles the credit & debit details. All of these three employee types have few attributes common such as name & salary which we had left associated with the parent entity "Employee" as shown in the above diagram.

➢ Generalization

Generalization is a bottom-up approach in which multiple lower-level entities are combined to form a single higher-level entity. Generalization is usually used to find common attributes among entities to form a generalized entity. It can also be thought of as the opposite of specialization.

1. Generalization uses bottom-up approach where two or more lower level entities combine together to form a higher level new entity.
2. The new generalized entity can further combine together with lower level entity to create a further higher level generalized entity.





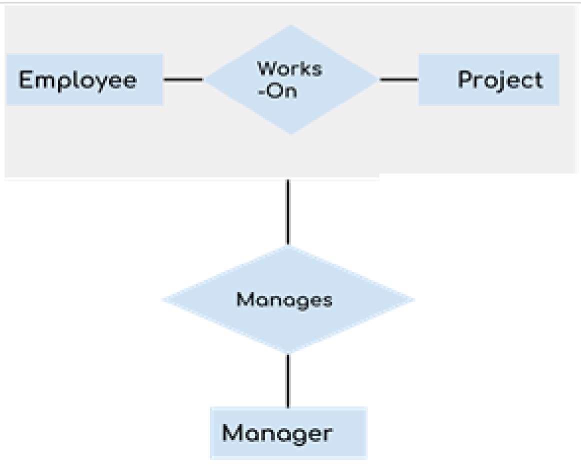
Generalization

Example: Generalization process the entities Student and Teacher only has the specialized attributes Grade and Salary respectively and their common attributes (Name & Address) are now associated with a new entity Person which is in the relationship with both the entities (Student & Teacher).

➢ Aggregation

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

Example: A manager not only manages the employee working as well as manages the project. In such scenario if entity "Manager" makes a "manages" relationship with either "Employee" or "Project" entity only then it will not make any sense because he has to manage both.



In these cases the relationship of two entities acts as one entity. In our example, the relationship "Works-On" between "Employee" & "Project" acts as one entity that has a relationship "Manages" with the entity "Manager".

Relational Database Concepts:

Introduction to relational databases

A relational database is based on the relational model and uses a collection of tables to represent both data and the relationship among those data.

It also includes DDL and DML

Relational database was originally defined by Edger Codd at IBM Research center in 1970.

In relational database, user only needs to understand logical structure of data, not how it is physically stored.

Data is represented using tables consists of rows and columns.

A relational database simply a collection of tables.

Relational database basic concepts

i) Relations or table

 A relation is defined as set of tuples that have the same attribute

 A relation is usually described as table, which is organized into rows and columns.

ii) Base and derived relation

 In relational database, all data are stored and accessed using relation.

 Relation / table which store data are called base relations.

 Relations which do not store data, but are computed by applying relational operator are called Derived relation.

iii) Tuple / Row / Record

 It holds all information about one item

 Example: all information like roll, name, age, address, age, mark etc., of a particular student.

iv) Field / Column

 A field holds one piece of information about an item.

 Field is column in database table.

 Example: age of all the student.

v) Constraints

 Condition specified for a particular data.

 Constraints restrict data that can be stored in relations.

 Example: User can set constraints that a given integer attribute should be between 1 & 10.

vi) Data type

 Every field in a database table is assigned a data types, which describe the kind of data that can be stored in the field.

vii) Stored procedure

 A stored procedure is a high-end database tool that adds programming power into database.
 Stored procedure is executable code generally stored in database.
 DBA will often create stored procedures to handle insert, edit and update of records.
 Front end programmer calls the stored procedure to utilize its functions.
 It makes programming code easier.

viii) Indices

 An index is one way of providing quicker access to data.
 It can be created on any combination of attributes on a relation.
 Relational database typically support multiple indexing technique.
 Indexing technique used are B Tree, B+ Tree.

ix) Normalization

 Normalization is used to eliminate the duplication of data.
 It is an integral part of the relational model.
 It prevents data manipulation anomalies and loss of data integrity.

Features of a relational database
Relational databases need ACID characteristics.

ACID refers to four essential properties: Atomicity, Consistency, Isolation, and Durability.

These features are the key difference between a relational database and a non-relational database.

Atomicity

Atomicity keeps data accurate. It makes sure all data is compliant with the rules, regulations, and policies of the business.

It also requires all tasks to succeed, or the transaction will roll back.

Atomicity defines all the elements in a complete database transaction.

Consistency

The state of the database must remain consistent throughout the transaction.
Consistency defines the rules for maintaining data points. This ensures they remain in a correct state after a transaction.
Relational databases have data consistency because the information is updated across applications and database copies (also known as ‘instances’). This means multiple instances always have the same data.

Isolation

With a relational database, each transaction is separate and not dependent on others. This is made possible by isolation.
Isolation keeps the effect of a transaction invisible until it is committed. This reduces the risk of confusion.

Durability

Durability means that you can recover data from a failed transaction.
It also ensures that data changes are permanent.

Difference between DBMS and RDBMS

DBMS	RDBMS
DBMS implies for Database Management System.	RDBMS implies for Relational Database Management System.
A File-System method is used to store the data.	Data is stored in tabular format that consists of rows and columns.
Data is represented and accessed in hierarchical form.	Tables are used to represent and access the data.
No relationship between the data is represented in DBMS.	Relationship between the data is presented through tables.
Normalization is not supported in DBMS.	Normalization can be applied in RDBMS.
It is limited to handling small amount of data.	It can handle large and complex set of data.
It supports distributed database.	It does not support distributed database.
Only single user at a time can access the data.	Multiple users are allowed to access data at a time.
Example: File System	Example: SQL, MySQL, Oracle, etc.

12 Codd's Rules

Rule 0: The Foundation Rule

The database must be in relational form. So that the system can handle the database through its relational capabilities.

Rule 1: Information Rule

A database contains various information, and this information must be stored in each cell of a table in the form of rows and columns.

Rule 2: Guaranteed Access Rule

Every single or precise data (atomic value) may be accessed logically from a relational database using the combination of primary key value, table name, and column name.

Rule 3: Systematic Treatment of Null Values

This rule defines the systematic treatment of Null values in database records. The null value has various meanings in the database, like missing the data, no value in a cell, inappropriate information, unknown data and the primary key should not be null.

Rule 4: Active/Dynamic Online Catalog based on the relational model

It represents the entire logical structure of the descriptive database that must be stored online and is known as a database dictionary. It authorizes users to access the database and implement a similar query language to access the database.

Rule 5: Comprehensive Data SubLanguage Rule

The relational database supports various languages, and if we want to access the database, the language must be the explicit, linear or well-defined syntax, character strings and supports the comprehensive: data definition, view definition, data manipulation, integrity constraints, and limit transaction management operations. If the database allows access to the data without any language, it is considered a violation of the database.

Rule 6: View Updating Rule

All views table can be theoretically updated and must be practically updated by the database systems.

Rule 7: Relational Level Operation (High-Level Insert, Update and delete) Rule

A database system should follow high-level relational operations such as insert, update, and delete in each level or a single row. It also supports union, intersection and minus operation in the database system.

Rule 8: Physical Data Independence Rule

All stored data in a database or an application must be physically independent to access the database. Each data should not depend on other data or an application. If data is updated or the physical structure of the database is changed, it will not show any effect on external applications that are accessing the data from the database.

Rule 9: Logical Data Independence Rule

It is similar to physical data independence. It means, if any changes occurred to the logical level (table structures), it should not affect the user's view (application). For example, suppose a table either split into two tables, or two table joins to create a single table, these changes should not be impacted on the user view application.

Rule 10: Integrity Independence Rule

A database must maintain integrity independence when inserting data into table's cells using the SQL query language. All entered values should not be changed or rely on any external factor or application to maintain integrity. It is also helpful in making the database-independent for each front-end application.

Rule 11: Distribution Independence Rule

The distribution independence rule represents a database that must work properly, even if it is stored in different locations and used by different end-users. Suppose a user accesses the database through an application; in that case, they should not be aware that another user uses particular data, and the data they always get is only located on one site. The end users can access the database, and these access data should be independent for every user to perform the SQL queries.

Rule 12: Non Subversion Rule

The non-submersion rule defines RDBMS as a SQL language to store and manipulate the data in the database. If a system has a low-level or separate language other than SQL to access the database system, it should not subvert or bypass integrity to transform data.