UNIT-1

Introduction: Database system, Characteristics (Database Vs File System), Database Users, Advantages of Database systems, Database applications. Brief introduction of different Data Models; Concepts of Schema, Instance and data independence; Three tier schema architecture for data independence; Database system structure, environment, Centralized and Client Server architecture for the database.

Entity Relationship Model: Introduction, Representation of entities, attributes, entity set, relationship, relationship set, constraints, sub classes, super class, inheritance, specialization, generalization using ER Diagrams.

INTRODUCTION TO DATABASE SYSTEM:

Database is a collection of related data and data is a collection of facts and figures that can be processed to produce information.

Mostly data represents recordable facts. Data aids in producing information, which is based on facts. For example, if we have data about marks obtained by all students, we can then conclude about toppers and average marks.

A ***database management system*** stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

Functions of a Database System:

- **Data Definition:** Defines the structure and organization of the data. This includes creating tables, setting data types, and defining relationships between data.
- **Data Manipulation:** Allows users to insert, update, delete, and query data. SQL is the most commonly used language for data manipulation.
- **Data Retrieval:** The DBMS enables users to retrieve data based on specific queries. This is typically done using SQL SELECT statements to fetch relevant data from the database.
- **Data Control:** Manages the security of the database and ensures that only authorized users can access or manipulate data. It also manages permissions for different users or roles.
- **Database Administration:** Database administrators (DBAs) are responsible for managing and overseeing the overall database system, ensuring its availability, performance, security, and integrity.

CHARACTERISTICS: DATABASE APPROACH VS. FILE PROCESSING APPROACH

- **Database Systems** are designed for managing large volumes of data with high integrity, flexibility, security, and efficient querying capabilities. They are best suited for applications where data relationships, concurrency control, and consistency are crucial.
- **File Systems**, on the other hand, are simple storage systems for organizing files on disk and are appropriate for applications that don't require complex data relationships or advanced querying features. However, for large-scale, relational, and transactional data needs, database systems are far more robust and efficient.

Characteristic	Database System	File System
Data Organization	Data is stored in a structured format (tables, records, fields) with relationships.	Data is stored as files (unstructured or semi-structured).
Data Redundancy	Minimizes redundancy by enforcing normalization and relationships.	Data redundancy is common because each file can store duplicate information.
Data Integrity	Ensures data integrity through constraints (primary keys, foreign keys, etc.).	No built-in integrity checks; integrity depends on how data is managed in files.
Data Retrieval	Efficient querying using SQL, with indexing and optimization.	Data retrieval is often slow and requires manual programming to read

		data from files.
Concurrency Control	Supports multiple users accessing the data simultaneously while ensuring consistency (via ACID properties).	Typically does not support concurrent access; multiple processes accessing files can cause conflicts.
Security	Provides built-in security features (user roles, access control, encryption).	Security is not inherently provided; file access control is usually handled by the operating system.
Backup and Recovery	Offers automated backup and recovery mechanisms. Data can be restored to a consistent state.	File systems usually require manual backup procedures and might not offer advanced recovery features.
Scalability	Scales well for large datasets, handling vast amounts of data efficiently.	Generally less scalable; as data grows, performance may degrade without special handling.
Data Independence	Supports logical and physical data independence, meaning changes to data structure don't affect applications.	Changes to the file structure often require changes to application logic.
Data Sharing	Allows multiple users and applications to share data securely and efficiently.	Sharing data between applications can be cumbersome, often requiring file management or conversion.
Querying and Reporting	Advanced querying and reporting through SQL (Structured Query Language).	File systems typically don't support complex querying; applications need to read and process data manually.
Transaction Management	Supports **ACID** (Atomicity, Consistency, Isolation, Durability) properties to ensure reliable transactions.	Does not natively support transaction management, making it prone to inconsistency if crashes occur.
Data Relationships	Supports relationships between different data sets (e.g., one-to-many, many-to-many).	No built-in relationship management; each file is independent.
Normalization	Data is normalized to reduce redundancy and improve efficiency.	Data is typically not normalized, leading to possible redundancy and inconsistencies.
Flexibility	Allows complex queries, joins, and operations on data, offering more flexibility for data manipulation.	Data manipulation is less flexible; manual programming is needed to process and combine data.
Performance Optimization	Supports indexing, query optimization, and caching to improve performance.	Performance depends on the file management and manual optimization techniques.
Data Access Method	Access to data is done through the DBMS, with a structured query language (SQL) interface.	Access to data is done through the file system, typically via file I/O operations.
Example	MySQL, Oracle, PostgreSQL, Microsoft SQL Server.	FAT, NTFS, EXT, HDFS, etc. (file systems used in operating systems).

DATABASE USERS

1) Database Administrator (DBA): chief administrator, who oversees and manages the database system (including the data and software). Duties include authorizing users to access the database, coordinating/monitoring its use, acquiring hardware/software for upgrades, etc. The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA might have a support staff.

2) Database Designers: responsible for identifying the data to be stored and for choosing an appropriate way to organize it. Database designers typically interact with each potential group of users and develop views of the database that meet the data and processing requirements of

these groups. The final database design must be capable of supporting the requirements of all user groups.

3) End Users: These are persons who access the database for querying, updating, and report generation. There are several categories of end users:

- **Casual end users:** use database occasionally, needing different information each time; use query language to specify their requests; typically middle- or high-level managers.
- **Naive/Parametric end users:** biggest group of users; frequently query/update the database using standard canned transactions that have been carefully programmed and tested in advance.
  Examples:
  - ✓ Bank tellers check account balances, post withdrawals/deposits
  - ✓ Reservation clerks for airlines, hotels, etc., check availability of seats/rooms and make reservations.
- **Sophisticated end users:** Include engineer scientists business analysts and others, who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.
- **Stand-alone users:** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces.
  Example: user of a tax package that stores a variety of personal financial data for tax purposes

ADVANTAGES OF USING THE DBMS APPROACH:

1. Data Redundancy Control

- Reduction in Data Duplication: In a DBMS, data is stored in a central repository and can be shared across multiple applications. Unlike file systems, where data may be duplicated in several files, DBMS reduces data redundancy (duplicate data) through techniques like normalization.
- Consistency: Reducing redundancy helps maintain consistency across the database, ensuring that the same data is not stored in multiple places and avoids potential conflicts.

2. Data Integrity

- Accuracy and Consistency: DBMS ensures data integrity by enforcing integrity constraints such as primary keys, foreign keys, and unique constraints. These rules maintain the accuracy and consistency of data across the system.
- Error Prevention: Constraints like NOT NULL, CHECK, and DEFAULT ensure that data follows predefined rules, reducing data errors.

3. Data Security

- Access Control: DBMS provides sophisticated security features such as user authentication, access controls, and encryption. It allows you to define which users or applications can access or modify specific data.
- Authorization: Users are granted access rights based on roles, ensuring sensitive data is protected and only accessible to authorized users.

4. Data Sharing

- Centralized Data Access: A DBMS allows data to be shared between different users, applications, and systems within an organization. It provides a centralized platform to store and manage data, making it easily accessible to all authorized parties.
- Multiple User Access: It supports concurrent access, meaning multiple users can access and manipulate the data simultaneously without compromising data integrity.

5. Improved Data Retrieval

- Efficient Querying: With DBMS, data retrieval becomes much easier and faster compared to file-based systems. Users can query data using SQL (Structured Query Language), a powerful language for retrieving and manipulating data.
- Indexes and Optimization: DBMS uses indexing and other optimization techniques to speed up query processing, ensuring fast retrieval of large datasets.

6. Data Independence

- Logical and Physical Independence: A DBMS provides logical data independence (changes in the logical schema don't affect applications) and physical data independence (changes in how data is physically stored don't affect the logical structure or applications).

- Ease of Modification: Data structures can be modified or expanded without disrupting the database application, offering greater flexibility.

7. Backup and Recovery

- Automated Backup: DBMS offers automatic backups of the entire database, ensuring that critical data is preserved in case of system failure or data corruption.
- Recovery Mechanisms: In the event of data loss or system crashes, a DBMS can restore the database to a consistent state using transaction logs and rollback features, minimizing downtime and data loss.

8. Concurrency Control

- Simultaneous Data Access: DBMS allows multiple users to access and update the database at the same time, ensuring that they do not interfere with each other's operations. It uses locking mechanisms and isolation levels to prevent data conflicts (e.g., lost updates or uncommitted transactions).
- ACID Properties: DBMS ensures that database transactions adhere to the ACID properties (Atomicity, Consistency, Isolation, Durability), guaranteeing that transactions are processed reliably and consistently.

9. Reduced Data Redundancy

- By organizing data into tables and eliminating the need for storing the same information in multiple places, a DBMS reduces unnecessary data duplication and storage space requirements.

10. Transaction Management

- ACID Compliance: DBMS ensures atomicity, meaning that a transaction is either fully completed or not executed at all. It ensures consistency, isolation, and durability, meaning that transactions will leave the database in a valid state and will be safely stored.
- Transaction Logs: DBMS uses transaction logs to track every change made to the database. This enables rollback of changes if an operation is incomplete or an error occurs, preventing data corruption.

11. Data Sharing and Integration

- Multi-application Support: DBMS allows multiple applications to share a common data source, ensuring data consistency across applications and systems.
- Data Integration: It enables integration of data from various sources and ensures that data is presented in a unified and consistent format, reducing the complexity of data management across systems.

12. Scalability

- Handling Large Data Volumes: DBMS is designed to handle large volumes of data, and it can scale to accommodate growing data needs. It supports techniques like partitioning, sharding, and distributed databases to ensure smooth handling of large datasets.
- Horizontal and Vertical Scaling: A DBMS can scale both horizontally (across multiple machines) and vertically (by adding resources like CPU and RAM) to handle more users and larger datasets.

13. Cost Efficiency

- Reduced Operational Costs: By improving data sharing, security, and retrieval efficiency, a DBMS reduces the operational costs of data management. It also reduces the time spent on manual data entry and ensures that resources are optimally used.
- Avoiding Data Redundancy: By eliminating duplicate data storage, DBMS reduces storage requirements and leads to cost savings, especially in large-scale systems.

DATABASE APPLICATIONS:

1. **Enterprise Resource Planning (ERP) Systems**
   Description: ERP systems are integrated software solutions used by organizations to manage and automate core business processes, such as accounting, human resources, inventory management, sales, and procurement.
   Database Use: These systems rely heavily on databases to store data across different departments and to provide real-time updates and reporting.
   Examples: SAP ERP, Oracle ERP, Microsoft Dynamics.
2. **Customer Relationship Management (CRM) Systems**

Description: CRM systems help businesses manage and analyze customer interactions and data throughout the customer lifecycle. These applications are used to improve customer relationships, retention, and sales.
Database Use: A CRM system stores customer data, sales information, interaction logs, and marketing campaigns in databases to enable detailed analysis and reporting.
Examples: Salesforce, HubSpot, Zoho CRM.

3. **Banking and Financial Applications**
Description: These applications are used by financial institutions like banks, investment firms, and insurance companies to manage financial transactions, accounts, and customer data.
Database Use: Databases store sensitive financial data such as account details, transaction histories, balances, and audit logs. They also manage data for generating financial reports and regulatory compliance.
Examples: Core banking systems, online banking applications, investment portfolio management systems.
4. **E-commerce Applications**
Description: E-commerce platforms are used by businesses to sell goods and services online. These applications manage customer orders, product inventories, payment processing, and order fulfillment.
Database Use: Databases store information on products, customers, orders, inventory levels, shipping details, and payment history.
Examples: Amazon, eBay, Shopify.
5. **Healthcare Management Systems**
Description: Healthcare management systems are used by hospitals, clinics, and other healthcare institutions to manage patient records, appointments, billing, and inventory of medical supplies.
Database Use: Databases store electronic health records (EHR), patient history, prescriptions, billing information, and medical reports.
Examples: Epic Systems, Cerner, Allscripts.
6. **Inventory Management Systems**
Description: These systems are used by businesses to manage inventory, track stock levels, handle reordering, and maintain accurate records of product movements.
Database Use: Databases track products, suppliers, inventory levels, sales orders, and purchase orders.
Examples: TradeGecko, Zoho Inventory, NetSuite.
7. **Educational Management Systems**
Description: These applications are used by schools, universities, and educational institutions to manage student records, grades, class schedules, and faculty data.
Database Use: Databases store student information, grades, course schedules, attendance records, and other academic data.
Examples: Blackboard, Moodle, PowerSchool.
8. **Social Media Platforms**
Description: Social media platforms allow users to create profiles, connect with others, and share content. These platforms manage user-generated content and interactions.
Database Use: Databases store user profiles, posts, comments, likes, messages, and connections between users (e.g., friendships, followers).
Examples: Facebook, Twitter, Instagram.
9. **Supply Chain Management (SCM) Systems**
Description: SCM systems are used by organizations to manage the flow of goods and services, including inventory, production, procurement, and logistics.
Database Use: Databases store information on suppliers, products, shipping schedules, procurement orders, and stock levels.
Examples: Oracle SCM Cloud, SAP SCM, Microsoft Dynamics 365.
10. **Library Management Systems**
Description: These systems are used by libraries to manage book inventories, checkouts, returns, and memberships.

Database Use: Databases store information on books, members, due dates, and transaction histories (book loans).
Examples: Koha, Libsys, Sierra.

11. **Point of Sale (POS) Systems**
Description: POS systems are used by retail businesses to manage sales transactions, process payments, and track inventory in real-time.
Database Use: Databases store transaction data, inventory levels, customer purchase history, and sales reports.
Examples: Square POS, Shopify POS, Lightspeed POS.
12. **Data Warehouses and Business Intelligence (BI) Applications**
Description: BI applications are used to collect, store, and analyze large volumes of data for decision-making. Data warehouses aggregate data from different sources for analytical purposes.
Database Use: Databases store historical data, metrics, and other business performance indicators. These databases enable reporting, data mining, and trend analysis.
Examples: Tableau, Power BI, Amazon Redshift.
13. **Content Management Systems (CMS)**
Description: A CMS allows users to create, manage, and modify digital content (such as websites or blogs) without needing specialized technical knowledge.
Database Use: Databases store content such as text, images, videos, articles, and metadata for easy retrieval and modification.
Examples: WordPress, Joomla, Drupal.
14. **Travel and Booking Systems**
Description: These applications are used by travel agencies, airlines, hotels, and car rental companies to manage bookings, reservations, and customer data.
Database Use: Databases store flight or hotel availability, reservation details, customer preferences, and transaction history.
Examples: Expedia, Booking.com, Airbnb.

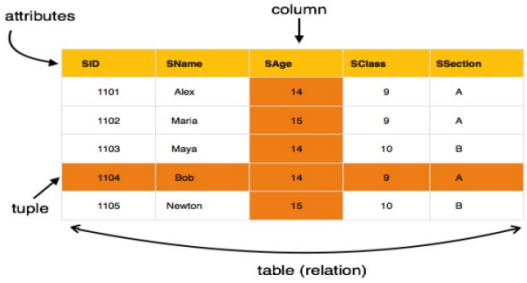
DATA MODELS:

A **data model** in a Database Management System (DBMS) defines how data is structured, stored, and manipulated. It serves as a blueprint for designing databases, determining how data is related, how it can be accessed, and how it will be processed.

Relational Data Model

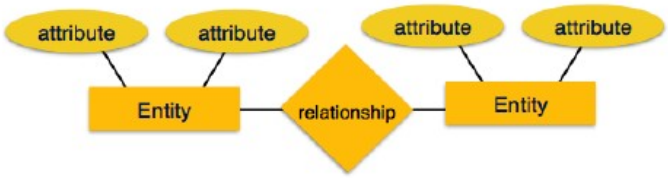
- **Description**: The **Relational Data Model** is the most widely used model in modern databases, represented in tables (also called **relations**). Each table consists of rows (records) and columns (attributes).
- **Structure**: Data is organized into tables, where each table contains **tuples (rows)** and **attributes (columns)**. Tables are related to each other via **foreign keys**, which are references to the primary key of another table.
- **Example**: A relational model for a library system:
  - **Books** table: (BookID, Title, AuthorID)
  - **Authors** table: (AuthorID, AuthorName)
  - **Loans** table: (LoanID, BookID, BorrowerID, LoanDate)

The **Books** and **Authors** tables are related by the AuthorID field, and the **Books** and **Loans** tables are related by the BookID field.



Entity-Relationship (ER) Model

- **Description**: The **Entity-Relationship (ER) Model** is used for conceptual database design. It uses **entities**, **attributes**, and **relationships** to model the data requirements of a system.
- **Structure**: Entities represent real-world objects (e.g., Student, Course), and relationships represent associations between entities (e.g., enrollment). Attributes provide additional information about entities (e.g., Student Name, Course Code).



Object-Oriented Data Model

- **Description**: The **Object-Oriented Data Model** combines the principles of object-oriented programming (OOP) with database management. Data is stored as objects, which can have both **attributes** (data) and **methods** (functions).
  Example: A multimedia database where media files (like images, videos, or documents) are represented as objects containing data (e.g., file size, format) and methods (e.g., display, edit).

Document Data Model (NoSQL)

- **Description**: The **Document Data Model** is part of **NoSQL databases** and is used to store and manage semi-structured data. Data is stored in **documents**, typically in formats like **JSON** or **BSON** (binary JSON), which are easy to read and write.
  Example: MongoDB is a document-based NoSQL database that stores data as JSON-like documents.

SCHEMAS, INSTANCES AND DATA INDEPENDENCE:

Database schema

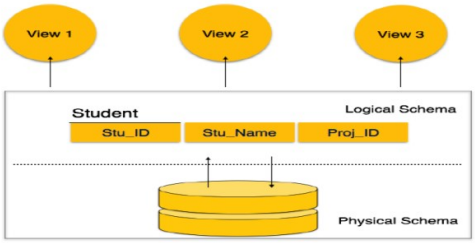
Schema is the logical design or structure of the database that defines how data is organized and how the relations between data are associated. It includes the **tables**, **views**, **indexes**, and **constraints** that describe the structure of the database.

Types of Schema:

- **Internal Schema (Physical Schema)**: Describes the physical storage of data, including how the data is stored on the disk (e.g., storage structures, file formats, indexes). This is concerned with the low-level storage details.
- **Conceptual Schema**: Describes the logical structure of the entire database, independent of any application or user. It defines the entities, their attributes, and the relationships between them (e.g., a collection of tables).
- **External Schema (View Schema)**: Represents the individual user views or different perspectives of the database. It defines what data is visible to different users and how it is presented to them (e.g., user-specific reports or access control).

Example:

- **Internal schema** may define how the Employee table is stored physically.
- **Conceptual schema** will define the Employee table's structure, such as columns like EmpID, Name, Salary, and how it's related to other tables like Department.
- **External schema** may present only the Name and Salary of employees to the HR department while restricting access to the EmpID and other sensitive data.



Instance in DBMS:

An instance is a snapshot of the data stored in the database at a specific point in time. It refers to the actual content or the data that is stored in the database, which changes dynamically as operations (inserts, updates, deletes) are performed.

- Schema is the structure (design) of the database.
- Instance is the actual data stored according to the schema at a given time.
- If the schema defines a table Employee (EmpID, Name, Salary), an instance of the database might be:

EmpID	Name	Salary
101	John Doe	50000
102	Jane Smith	55000

Data Independence in DBMS

Data independence is the capacity to change the schema at one level without having to change the schema at the next higher level. In simpler terms, it allows for the separation of data from the applications that use the data, providing flexibility to alter the data structure without affecting the application logic.

THREE TIER SCHEMA ARCHITECTURE FOR DATA INDEPENDENCE:

The goal of the three-schema architecture is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

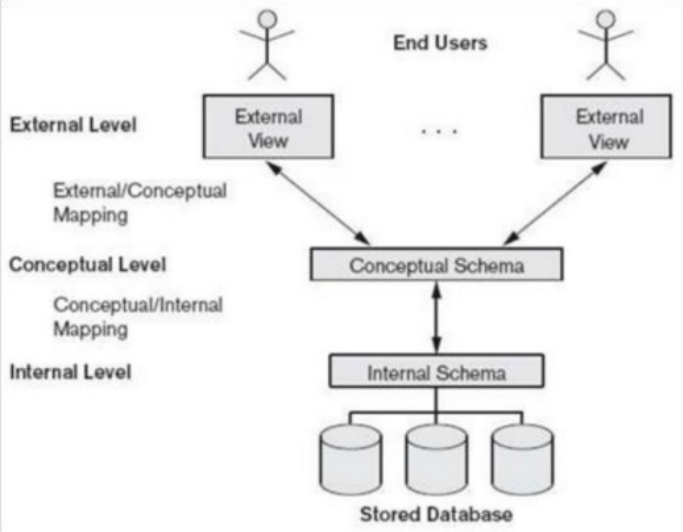


Figure: The three-schema architecture

1) **The internal level** has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

2) **The conceptual level** has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented.

3) **The external or view level** includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. Each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

In a DBMS based on the three-schema architecture, each user group refers to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema

for processing over the stored database. If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view.
The processes of transforming requests and results between levels are called mappings.

Data Independence

Data independence can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1) **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database, to change constraints, or to reduce the database. Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence.

2) **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized for example, by creating additional access structures to improve the performance of retrieval or update. Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the mapping between the two levels is changed.

CENTRALIZED AND CLIENT-SERVER ARCHITECTURE FOR DBMS:

Centralized DBMSs Architecture

All DBMS functionality, application program execution, and user interface processing carried out on one machine.

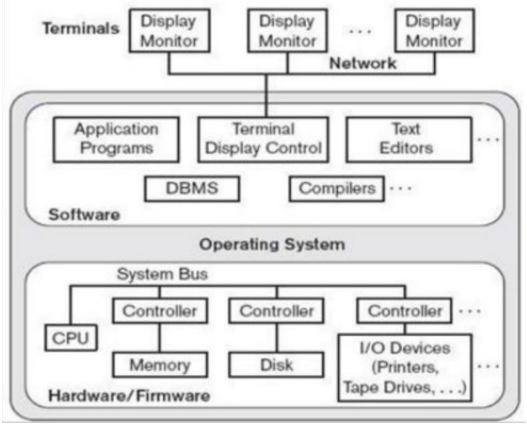


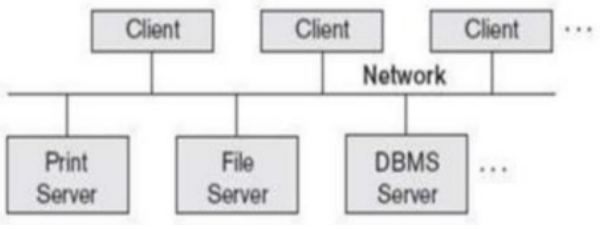
Figure: A physical centralized architecture

Basic Client/Server Architectures

The client/server architecture was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, e-mail servers, and other software and equipment are connected via a network.

Define specialized servers with specific functionalities.
For example file server that maintains the files of the client machines.
The resources provided by specialized servers can be accessed by many client machines.
The client machines provide the user with the appropriate interfaces to utilize these servers and local processing power to run local applications

Figure: Logical two-tier client/server architecture



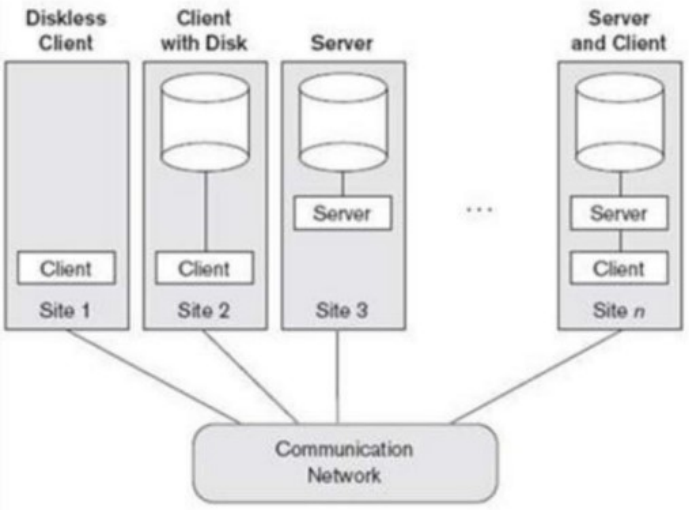


Figure: Physical two-tier client/server architecture

The concept of client/server architecture assumes an underlying framework that consists of many PCs and workstations as well as a smaller number of mainframe machines, connected via LANs and other types of computer networks.
A client is a user machine that provides user interface capabilities and local processing.
When a client requires access to additional functionality such as database access that does not exist at that machine, it connects to a server that provides the needed functionality.
A server is a system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access. The software components are distributed over two systems: client and server

- ✓ **Server handles:** Query and transaction functionality related to SQL
- ✓ **Processing Client handles:** User interface programs and application programs

The user interface programs and application programs can run on the client side. When DBMS access is required, the program establishes a connection to the DBMS (which is on the server side) once the connection is created, the client program can communicate with the DBMS.

A client program can actually connect to several RDBMSs and send query and transaction requests using the ODBC API, which are then processed at the server sites. Any query results are sent back to the client program, which can process and display the results as needed. A related standard for the Java programming language, called JDBC, has also been defined to allow Java client programs to access one or more DBMSs through a standard interface.

Object-oriented DBMSs

The different approach to two-tier client/server architecture was taken by some object-oriented DBMSs, where the software modules of the DBMS were divided between client and server in a more integrated way.

Server level: May include the part of the DBMS software responsible for handling data storage on disk pages, local concurrency control and recovery, buffering and caching of disk pages.

Client level: May handle the user interface, data dictionary functions, DBMS interactions with programming language compilers, global query optimization, concurrency control, and recovery across multiple servers, structuring of complex objects from the data in the buffers.

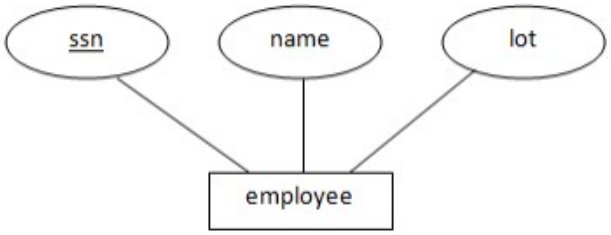
In this approach, the client/server interaction is more tightly coupled and is done internally by the DBMS modules some of which reside on the client and some on the server rather than by the users/programmers.

Entity Relationship Model

INTRODUCTION:

Entity: An entity is something which is described in the database by storing its data, it may be a concrete entity a conceptual entity.

Entity set: An entity set is a collection of similar entities. The Employees entity set with attributes ssn, name, and lot is shown in the following figure.



Attribute: An attribute describes a property associated with entities. Attribute will have a name and a value for each entity.

Domain: A domain defines a set of permitted values for a attribute

Entity Relationship Model: An ERM is a theoretical and conceptual way of showing data **relationships** in software development. It is a database **modeling** technique that generates an abstract diagram or visual representation of a system's data that can be helpful in designing a relational database.

ER model allows us to describe the data involved in a real-world enterprise in terms of objects and their relationships and is widely used to develop an initial database design.

REPRESENTATION OF ENTITIES:

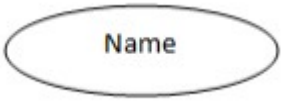
1. ENTITIES:

Entities are represented by using rectangular boxes. These are named with the entity name that they represent.

Fig: Student and Employee entities

2. ATTRIBUTES:

Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity.



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.

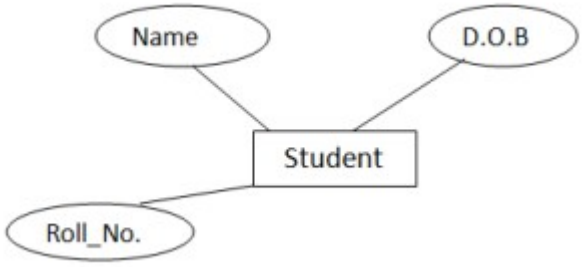
- **Derived attribute** − Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.

- **Single-value attribute** − Single-value attributes contain single value. For example Social_Security_Number.

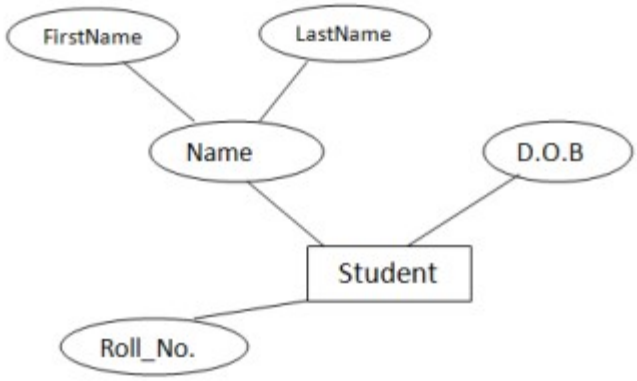
- **Multi-value attribute** − Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

ER Representation for Attributes:

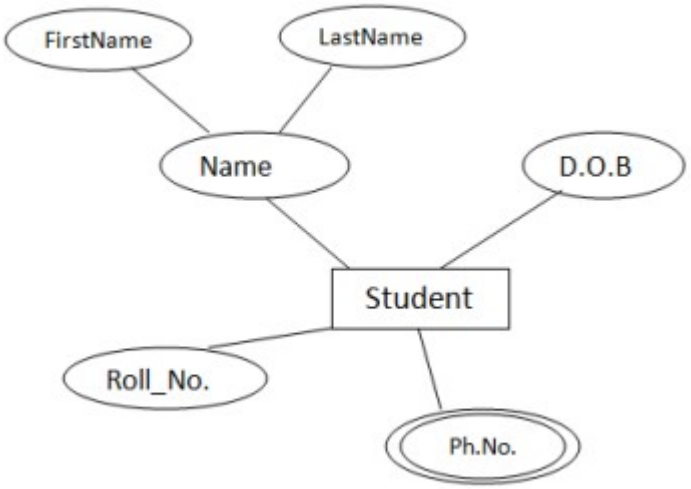
Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



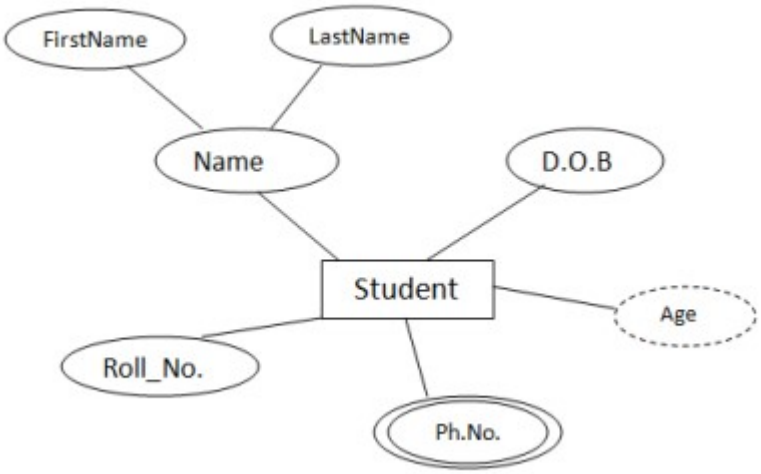
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.



Multi valued attributes are depicted by double ellipse.



Derived attributes are depicted by dashed ellipse.



3. RELATIONSHIP:

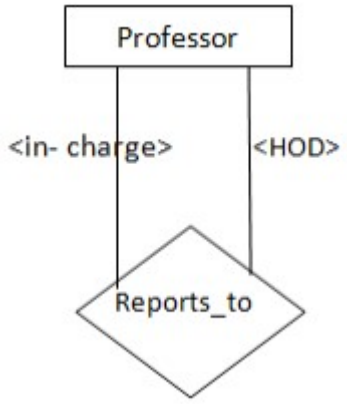
Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

Types of relationships:

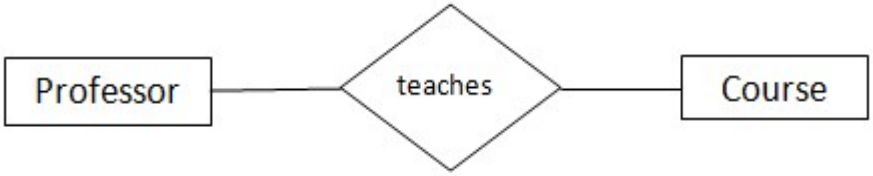
Degree of Relationship is the number of participating entities in a relationship defines the degree of the relationship. Based on degree the relationships are categorized as

- Unary = degree 1
- Binary = degree 2
- Ternary = degree 3
- n-ary = degree

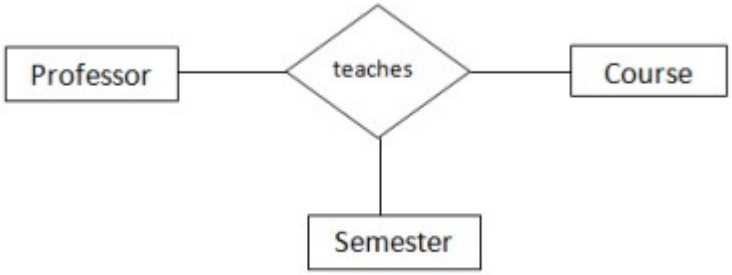
Unary Relationship: A relationship with one entity set. It is like a relationship among 2 entities of same entity set. Example: A professor (in-charge) reports to another professor (Head Of the Dept).



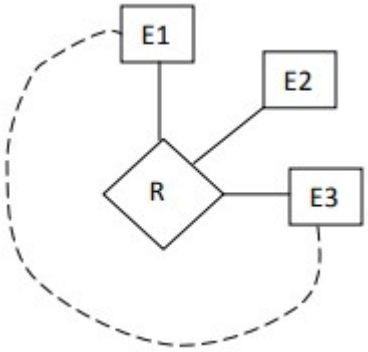
Binary Relationship: A relationship among 2 entity sets. Example: A professor teaches a course and a course is taught by a professor.



Ternary Relationship: A relationship among 3 entity sets. Example: A professor teaches a course in so and so semester.

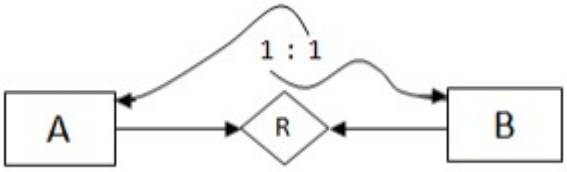


n-array Relationship: A relationship among n entity sets.

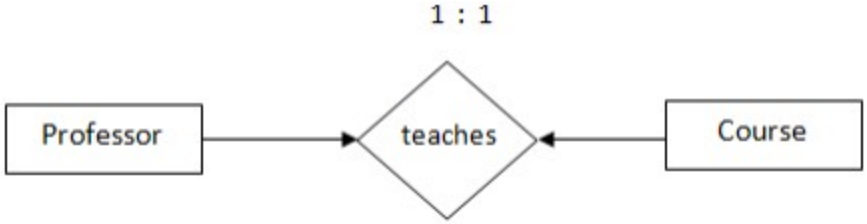


Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set. Cardinality ratios are categorized into 4. They are.

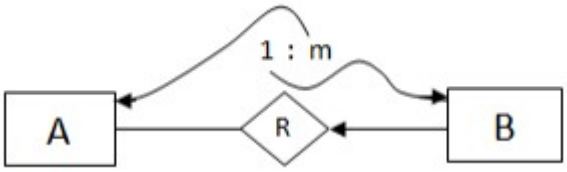
1. **<u>One-to-One relationship:</u>** When only one instance of an entities are associated with the relationship, then the relationship is *one-to-one relationship.* Each entity in A is associated with at most one entity in B and each entity in B is associated with at most one entity in A.



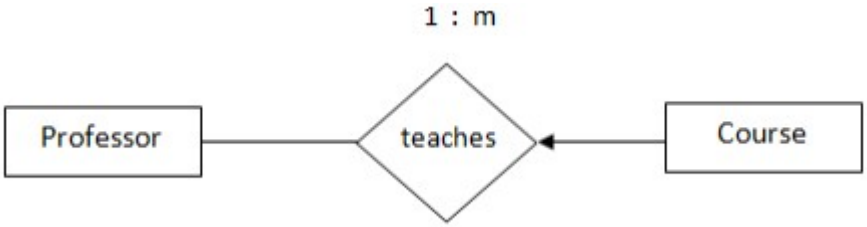
Each professor teaches one course and each course is taught by one professor.



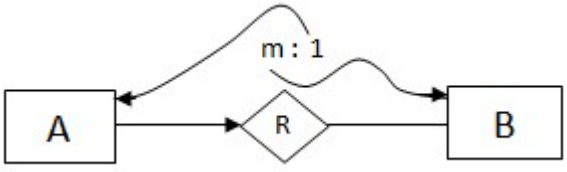
2. **<u>One-to-many relationship</u>**: When more than one instance of an entity is associated with a relationship, then the relationship is *one-to-many relationship.* Each entity in A is associated with zero or more entities in B and each entity in B is associated with at most one entity in A.



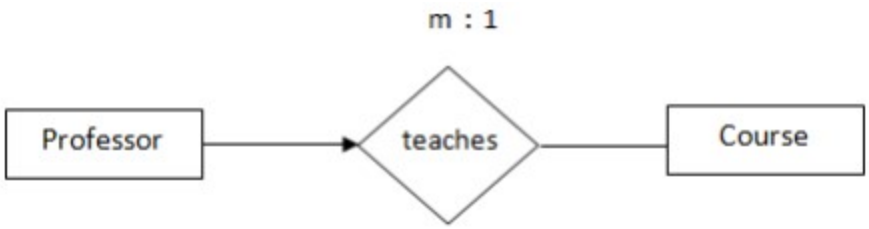
Each professor teaches 0 (or) more courses and each course is taught by at most one professor.



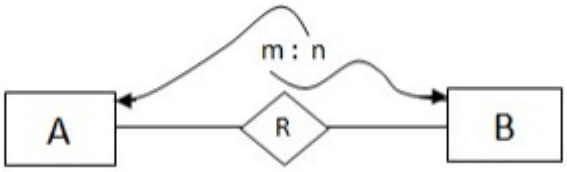
3. **<u>Many-to-one relationship</u>**: When more than one instance of entity is associated with the relationship, then the relationship is *many-to-one relationship.* Each entity in A is associated with at most one entity in B and each entity in B is associated with 0 (or) more entities in A.



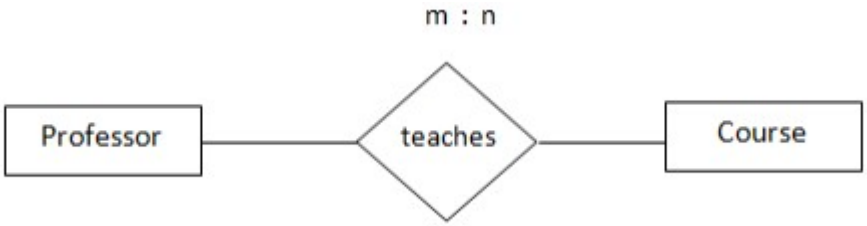
Each professor teaches at most one course and each course is taught by 0 (or) more professors.



4. **Many-to-Many relationship**: If more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship, then it depicts many-to-many relationship. Each entity in A is associated with 0 (or) more entities in B and each entity in B is associated with 0 (or) more entities in A.



Each professor teaches 0 (or) more courses and each course is taught by 0 (or) more professors.

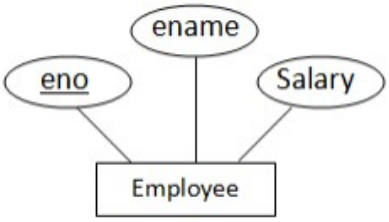


4. RELATIONSHIP SET:

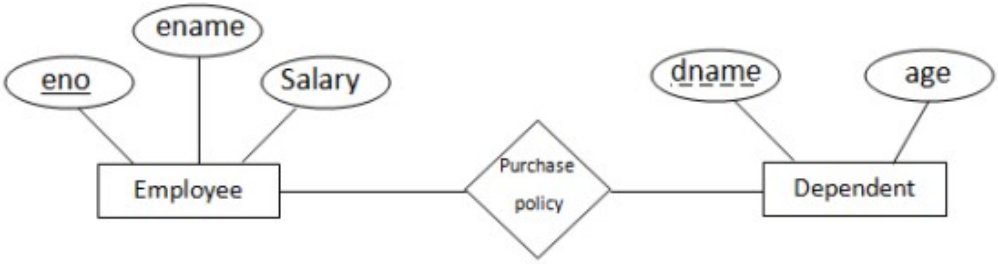
A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

STRONG AND WEAK ENTITY SETS:

Strong Entity set: If each entity in the entity set is distinguishable or it has a key then such an entity set is known as strong entity set.



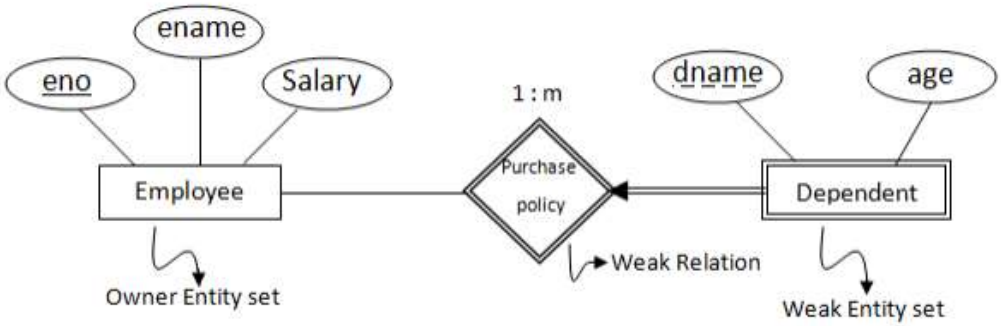
Weak Entity set: If each entity in the entity set is not distinguishable or it doesn't has a key then such an entity set is known as strong entity set.



eno is key so it is represented by solid underline. dname is partial key. It can't distinguish the tuples in the Dependent entity set. so dname is represented by dashed underline.

Weak entity set is always in total participation with the relation. If entity set is weak then the relationship is also known as weak relationship, since the dependent relation is no longer needed when the owner left.

Ex: policy dependent details are not needed when the owner (employee) of fired from that policy left or company or expired. The detailed ER Diagram is as follows.



The cardinality of the owner entity set is with weak relationship is 1 : m. Weak entity set is uniquely identifiable by partial key and key of the owner entity set.

Dependent entity set is key to the relation because the all the tuples of weak entity set are associated with the owner entity set tuples.

PARTICIPATION CONSTRAINTS:

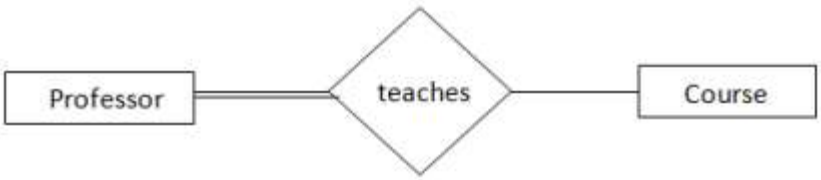
- **Total Participation** – If Each entity in the entity set is involved in the relationship then the participation of the entity set is said to be total. Total participation is represented by double lines.

- **Partial participation** – If, Not all entities of the entity set are involved in the relationship then such a participation is said to be partial. Partial participation is represented by single lines.

Example: Participation Constraints can be explained easily with some examples. They are as follows.

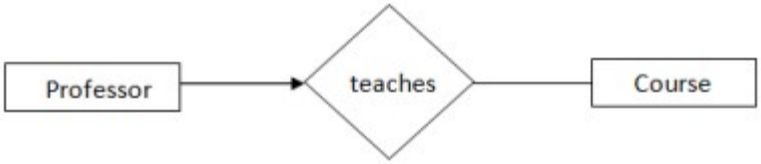
1.Each Professor teaches at least one course.

min=1 (Total Participation)
max=many (No key)



2. Each Professor teaches at most one course.

min=0 (Partial participation)
max=many (Key)

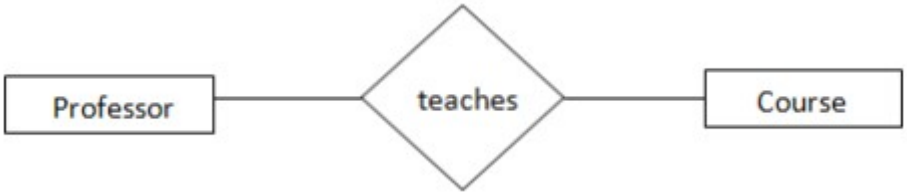


3. Each Professor teaches Exactly one course.

min=1 (Total Participation)
max=1 (Key)

4. Each Professor teaches course.

min=0 (Partial Participation)
max=many (no Key)



Note: Partial Participation is the default participation.

Constraints in Class Hierarchies:

Constraints that can be applied for Class Hierarchies are: 1. Condition Constraints
2. User Defined Constraints

A **Condition Defined Constraint** is imposed, while classifying the entities of a higher level entity set to be part of (or) a member of lower level entity sets based on a specified defined constraints.

Example: Every higher level entity in the entity set "Account" is checked using the attribute "acc_type" to be assigned either to the "SavingsAccount" or to the "CurrentAccount". SavingsAccount and CurrentAccount are lower level entity sets.

If no condition is specified during the process of designing the lower level entity sets, then it is called **user defined constraint.**

Disjoint Constraint: This constraint checks whether an entity belongs to only one lower level entity set or not.

Overlapping Constraint: This constraint ensures by testing out that an entity in the higher level entity set belong to more than one lower level entity sets.

Completeness Constraint: This is also called total constraint which specifies whether or not and entity in the higher level entity set must belong if at least one lower level entity set in generalization or specialization.

When we consider the completeness constraint, we come across total and partial constraints. i.e., Total Participation constraint and Partial Participation Constraint.

Total Participation forces that a higher level entity set 's entity(Every entity) must belong to at least one lower level entity set mandatorily.

Ex: An account entity set's entity set must be belong to either savings account entity set or current account entity set.

Partial Participation is rarely found with an entity set because sometimes an entity set in the higher level entity set beside being a member of that higher level entity set, doesn't belong to any of the lower level entity sets immediately until the stipulated period.

Ex: A new employer listed in the higher level entity set but not designated to any one of the available teams that belong to the lower level entity set.

GENERALIZATION AND SPECIALIZATION

One entity type might be a subtype of another, very similar to subclasses in OO programming. Relationship that is existed between these entities is known as IsA relationship. The two entities related by IsA are always descriptions of the same real-world object. These are typically used in databases to be implemented as Object Oriented Models. The upper entity type is the more abstract entity type (super type) from which the lower entities inherit its attributes.

Properties of Is A:

Inheritance:

- All attributes of the super type apply to the subtype.
- The subtype inherits all attributes of its super type.
- The key of the super type is also the key of the subtype.

Transitivity:

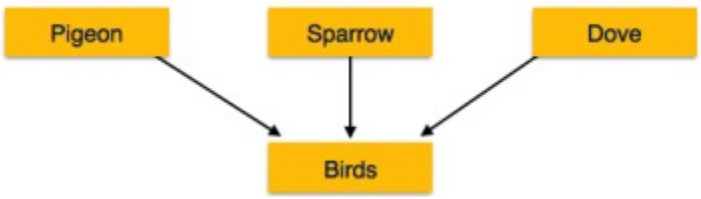
- This property creates a hierarchy of IsA relationships.

Advantages:

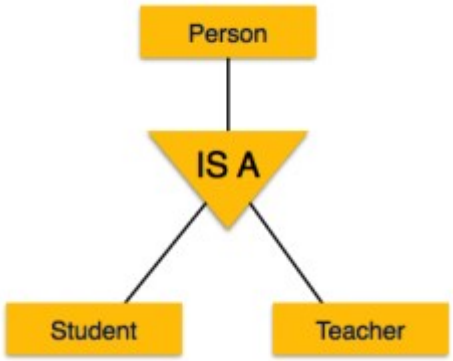
- Used to create a more concise and readable E-R diagram.
- It best maps to object oriented approaches either to databases or related applications.
- Attributes common to different entity sets need not be repeated.
- They can be grouped in one place as attributes of the supertype.
- Attributes of (sibling) subtypes are likely to be different.

The process of sub grouping with in a entity set is known as specialization or generalization. Specialization follows top down approach and generalization follows bottom-up approach. Both the speculation and generalization are depicted using a triangle component labeled as IS A.

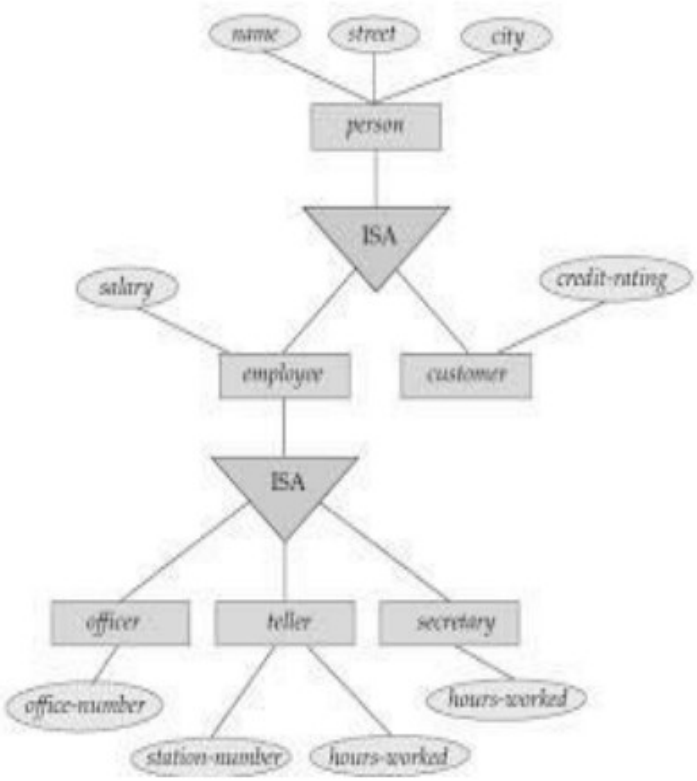
Generalization: is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entity to make further higher level entity. In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For example, pigeon, house sparrow, crow and dove can all be generalized as Birds.



Specialization: is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, some higher level entities may not have lower-level entity sets at all. 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, persons can be identified as employee, employer, customer, or vendor, based on what role they play in the company.



Inheritance: We use all the above features of ER-Model in order to create classes of objects in object-oriented programming. The details of entities are generally hidden from the user; this process known as **abstraction**. Inheritance is an important feature of Generalization and Specialization. It allows lower-level entities to inherit the attributes of higher-level entities.



Attribute inheritance is a crucial property where a subclass entity set inherits all the attributes of its super class entity set. Attributes can be additionally specified which is used to give a clear representation though that same attribute is found nowhere in the hierarchy.

Employee and customer can inherit the attributes of Person entity and they have their own attributes like salary for employee and credit_rating for customer. similarly, the entities officer, teller and secretary inherit alll the attributes of employee and they can have their own attributes like office_member for officer, station_number & hours_worked for teller and hours_worked for secretary.

If an entity set has one single higher level entity set then it is termed as single inheritance. If it has multiple higher level entity sets then we can term it as multiple inheritance.

Sub Class and Super Class

- Sub class and Super class relationship leads the concept of Inheritance.
- The relationship between sub class and super class is denoted with symbol. (d)

1. Super Class
   - Super class is an entity type that has a relationship with one or more subtypes.
   - An entity cannot exist in database merely by being member of any super class.
     For example: Shape super class is having sub groups as Square, Circle, Triangle.
2. Sub Class
   - Sub class is a group of entities with unique attributes.
   - Sub class inherits properties and attributes from its super class.
     For example: Square, Circle, Triangle are the sub class of Shape super class.

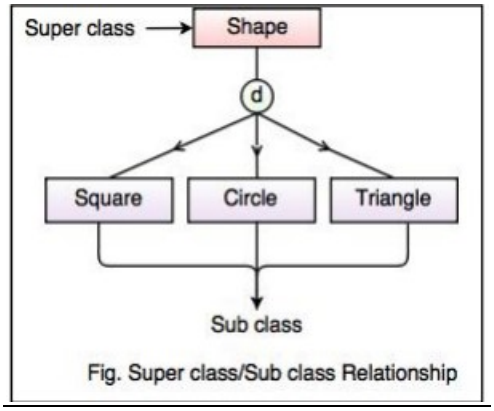
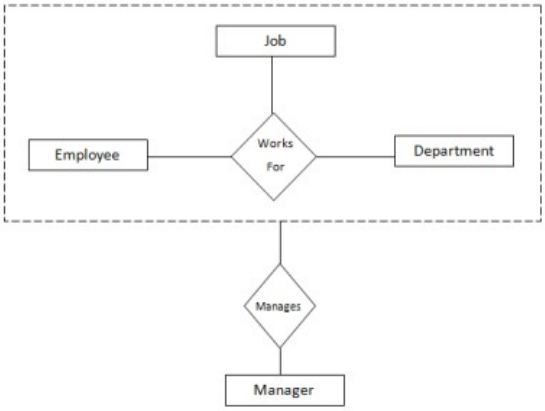


Fig. Super class/Sub class Relationship

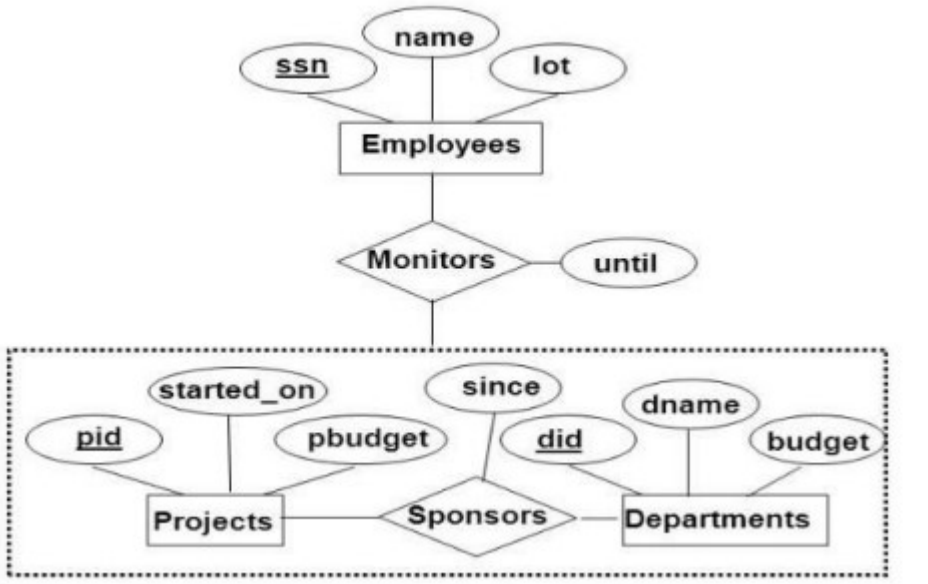
AGGREGATION:

An aggregation is not a ternary relationship but is an attempt to establish the relationship with another relationship set. It is also termed as relationship with in a relationship. Aggregation can be used over a binary, ternary or a quaternary relationship set. Aggregation is denoted using a dashed rectangle.

Aggregation over ternary relationship:



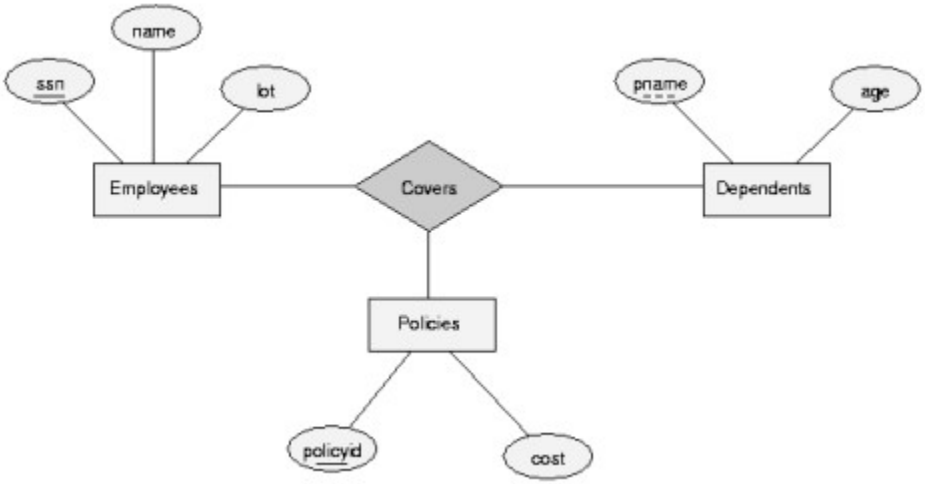
Aggregation over binary Relationships:



In the examples shown above, we treated the already existed relationship sets "WorksFor" and "Sponsors" as an entity set for defining the new relationship sets "Manages" and "Monitors". A relationship set is participating in another relationship. So it can be termed as aggregation.

TERNARY RELATIONSHIP DECOMPOSED INTO BINARY:

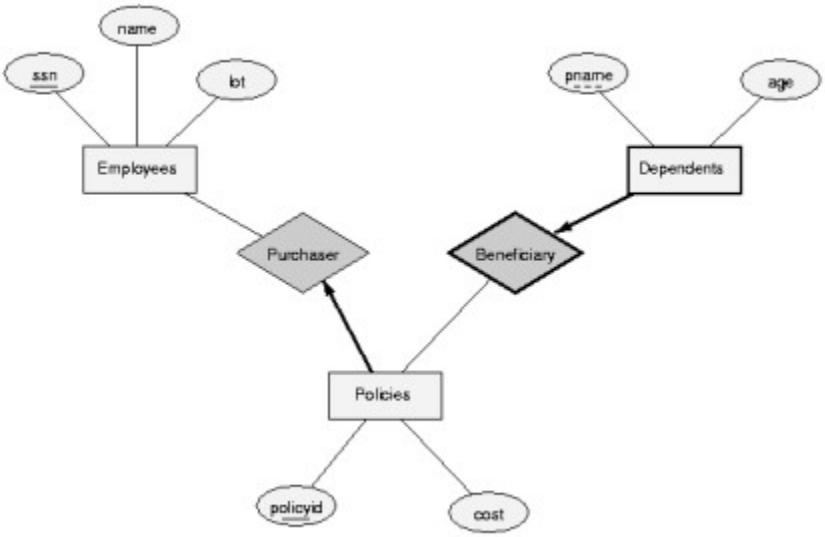
Consider the following ER diagram, representing insurance policies owned by employees at a company. It depicts 3 entity sets Employee, policy and Dependents. The 3 entity sets are associated with a ternary relationship set called Covers. Each employee can own several polices, each policy can be owned by several employees, and each dependent can be covered by several policies.



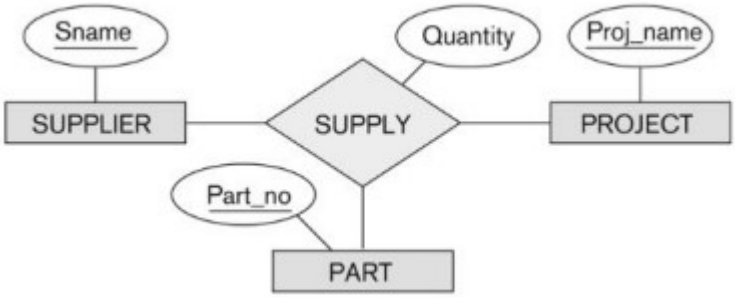
if we wish to model the following additional requirements:

- A policy cannot be owned by 2 or more employees.
- Every policy must be owned by some employee
- Dependents is a weak entity set and each dependent is uniquely identified by taking pname in conjunction with the policyid of a policy entity (which, intuitively, covers the given dependent).

The best way to model this is to switch away from the ternary relationship set, and instead use two distinct binary relationship sets.

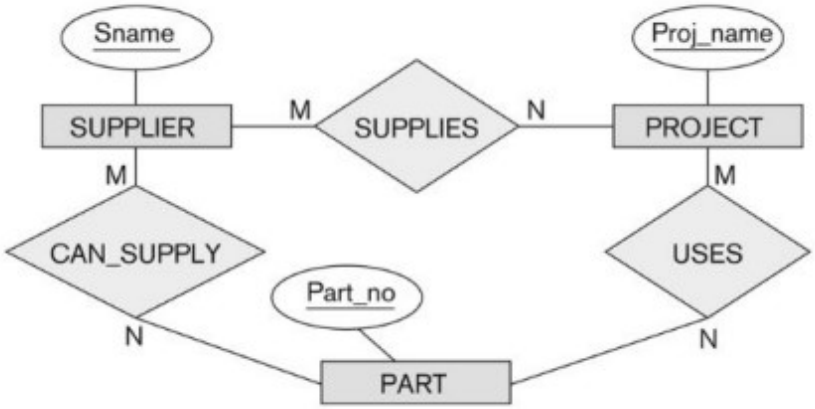


TERNARY VS BINARY: Generally the degree of a relationship set is assessed by counting the no. of nodes or edges that emanate from that relationship set.



Supply in the ternary relationship set from the first figure, which has a set of relationship instances (s,j,p) which means 's' is a supplier who is supplying part 'p' to a project 'j'.

A ternary relationship represent different information than 3 binary relationship sets do. Here the relationship sets canSupply, uses and supplies substitute the ternary relationship set "supply".



"CANSUPPLY", "USES" and "SUPPLIES" are the three binary relationship sets established where

- Supplier and part which have "CANSUPPLY" binary relationship include an instance (s,p) which says supplier 's' can supply part 'p' to any project.
- "USES" relationship between project and part includes an instance (j,p) which says project 'j' uses part 'p'.
- "SUPPLIES" binary relationship between supplier and project includes an instance (s,j) which says supplier 's' supplies some part to project 'j'.

No combination of binary relationships is an adequate substitute. Because there is question "where to add quantity attribute?". Is it to the can-supply or to the uses or to the supplies??

The solution for this is to maintain the same ternary relationship with a weak entity set Supply which has attribute Qty.