

Database Management System


Unit:1

Introduction To Database And Transaction

Data Models

Database Design

Chapter1: Introduction to Database and Transaction

- ❖ **What is database system**
- ❖ **Purpose of database system**
 - Data redundancy and inconsistency
 - Difficulty in accessing data
 - Data isolation
 - Integrity problems
 - Atomicity problems
 - Security problems
- ❖ **View of data**
 - Data Abstraction
 - Instances and Schema
- ❖ **Relational Database**
 - Tables
- ❖ **Data Manipulation Language (DML)**
- ❖ **Data Definition Language (DDL)**
- ❖ **Database access from application programs**
- ❖ **Database design**
 - Design Process
- ❖ **Database Architecture**
- ❖ **Transaction Management**
 - Atomicity
 - Consistency
 - Isolation
 - Durability

ACID

 - Recovery manager
 - Failure recovery
 - Concurrency

Introduction to Database and Transaction

Database Management System(DBMS)

- DBMS is a collection of interrelated data and a set of programs to access those data.
- DBMS refers to the overall process of creating, maintaining, storing, retrieving and managing databases in an efficient way.
- Main purpose is to manage large bodies of information in a structured format.

DBMS Applications

- Banking
 - Airlines
 - Universities
 - Credit card transaction
 - Telecommunication
 - Finance
 - Sales
-

Purpose of Database System

Data redundancy

- Occurs when the same data is unnecessarily duplicated or repeated within a database.
- Redundancy can lead to wastage of storage space and can pose challenges in maintaining data consistency.

Data inconsistency

- Refers to contradictions in data stored in the database.
- Can arise when the same data is updated in one place but not in another, leading to conflicting information across the database.

Difficulty in accessing data

Data access challenges refer to difficulties in efficiently retrieving information from storage systems, necessitating more responsive and user friendly data-retrieval methods for widespread adoption.

Data isolation

Data isolation is the concept in database management where transactions are executed in a way that does not interfere with each other, maintaining the consistency of the data.

Integrity problems

The data values stored in the database must satisfy certain types of consistency constraints.

Atomicity problems

Refers to issues where database transactions fail to ensure that all their operations are either entirely executed or entirely rolled back, potentially resulting in data inconsistencies.

Security problems

Not every user of the database system should be able to access all the data.

View of Data

Data abstraction

It's a process that simplifies data complexity by encapsulating details, allowing users to interact with data at higher levels without needing to understand its underlying structure.

There are several types of level-

1. View level
2. Logical level
3. Physical level
4. External level

Physical level

The focus is on how data is physically stored, including complex low-level data structures and storage mechanisms.

Logical level

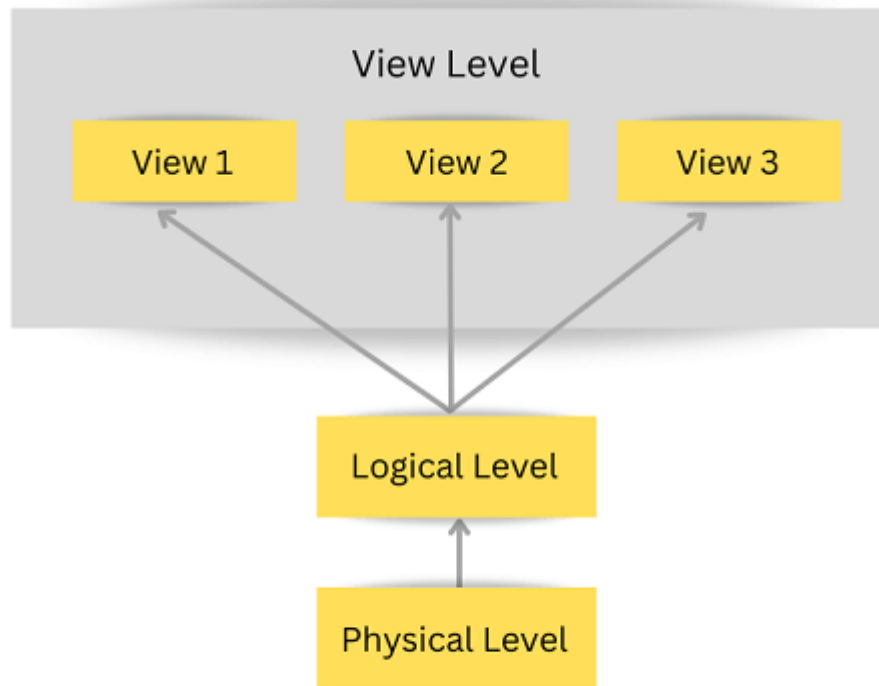
Describes what data is stored in the database and the relationships among that data, simplifying it into a small number of relatively simple structures.

View level / User level

Provides customised and user specific perspectives of data. Allows users to create virtual tables, specifying which data they want to see, while the underlying data structure remains unchanged.

External level / Highest level

Defines how end-users and applications interact with the data, including user interfaces, queries, and reports tailored to specific user needs.



Levels of Data Abstraction in DBMS

Instances and Schema

Instances

The collection of information stored in the database at a particular moment is called an instance of the database

Schema

The overall design of the database is called the database schema. Schemas are changed infrequently.

Relational Database

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 **Data Manipulation Language(DML)** and **Data Definition Language(DDL)**.

Table

A relational database table is a structured data repository with rows and columns, representing records and attributes, following a defined schema.

Data Manipulation Language

SQL is a nonprocedural language used for data retrieval. Queries specify desired data from tables, returning results as a single table.

Data Definition Language

SQL's DDL is used for defining tables and constraints.

Database access from application programs

- SQL lacks the computational power of universal Turing machines and cannot perform certain tasks of general-purpose programming languages.
 - SQL cannot handle user input, output to displays, or network communication.
 - Host languages like C, C++, or Java are used to perform these tasks with embedded SQL queries to access databases.
 - Application programs enable interaction with databases.
 - Accessing the database requires executing DML statements from the host language
 - Two methods exist for executing DML statements: API's that send DML and DDL statements to the database and retrieve results, and preprocessor-based embedding of DML calls within the host language.
 - Standards like ODBC (Open Database Connectivity) simplify database interaction in host languages.
-

Database Design

- Database systems manage extensive volumes of information within an enterprise.
 - These databases are integral to an enterprise's operations, either serving as a source of information or supporting various devices and services.
 - Database design primarily focuses on structuring the database schema.
 - Creating a comprehensive database applications environment to meet enterprise needs involves addressing broader considerations.
 - Database design is a critical aspect of optimising data management in practical contexts.
-

Design Process

- Establish a conceptual framework to outline data requirements and structure the database accordingly.
 - Interact with domain experts and users to comprehensively document data needs.
 - Translate user requirements into a high level schema, offering a detailed view of the enterprise's data structure.
 - Define the operations or transactions users will perform on the data, ensuring alignment with the conceptual schema.
 - Transform the conceptual schema into a system-specific data model, guiding the actual database implementation.
-

Database Architecture

- The architecture of a database system is significantly influenced by the underlying computer system on which it operates.
 - Database systems can be centralised or client-server, where one server serves multiple client machines.
 - Database users often connect to the system through a network, with client and server machines involved.
 - Database applications are typically divided into two or three parts
 - In a two-tier architecture, the application is at the client machine, invoking database functionality at the server through query language statements. API standards like ODBC and JDBC facilitate client-server interaction.
-

Transaction Management

Transaction management is a key component of database management systems (DBMS) that focuses on ensuring the consistency, integrity, and reliability of data within a database, particularly in the context of multiple, concurrent transactions. A transaction is a logical unit of work that consists of one or more operations, such as database queries or updates, that need to be executed as a single, indivisible unit.

Key concepts of Transaction Management are-

1. Atomicity
2. Consistency
3. Isolation
4. Durability



- Recovery manager
- Failure recovery
- Concurrency

Atomicity

Atomicity in database transactions guarantees that either all changes made within a transaction are applied successfully or none at all, avoiding partial updates and ensuring data integrity. It ensures that a transaction is treated as an all-or-nothing operation.

Consistency

Consistency in database transactions ensures that the execution of a transaction brings the database from one valid state to another, adhering to predefined integrity constraints. It prevents violations of database rules, maintaining data accuracy and reliability throughout the transaction's lifecycle.

Isolation

Isolation in database transactions ensures that each transaction is executed independently, unaffected by concurrent transactions. It prevents interference between transactions, maintaining their separate workspaces until completion. Isolation guarantees data integrity by preventing inconsistencies that might arise from the simultaneous execution of multiple transactions.

Durability

Durability in database transactions ensures that once a transaction is committed, its effects persist even in the face of system failures. The changes made by committed transactions are permanent and remain intact, providing a reliable and persistent state for the database, enhancing data integrity and recoverability.

Recovery manager

A recovery manager in a database system is responsible for ensuring the consistency and durability of data. It includes mechanisms for logging changes made during transactions, allowing for recovery after system failures. The recovery manager restores the database to a consistent state, maintaining data integrity.

Failure recovery

Failure recovery in a database system involves restoring the database to a consistent state after a system failure. It relies on recovery mechanisms like transaction logging to identify and redo or undo transactions. This process ensures data integrity by recovering from unexpected events, such as hardware failures or crashes.

Concurrency

Concurrency in a database system refers to the simultaneous execution of multiple transactions or processes. It allows multiple users to access and manipulate data concurrently. Concurrency control mechanisms, such as locking or timestamping, are employed to maintain data consistency and prevent conflicts during simultaneous transactions.

Chapter2: DATA MODELS

❖ Data Models

- Importance of Data Models
- Advantages of Data Models

❖ File Managements Systems

- Hierarchical Databases
- Network Databases

❖ Basic Building Blocks

- Entity
- Attributes
- Relationships
- Degree

❖ Types of Relationships

❖ Business Rules

- Characteristics of Business Rules
- Types of Business Rules

❖ Degrees of data Abstractions

DATA MODELS

Data Models

- Data models define how information is organised, identifying elements like names, numbers, and relationships.
- They use symbols and text to help business and IT professionals understand and discuss data structures effectively.
- Data models simplify complex real-world data, making it easier to manage and comprehend.
- By providing a clear structure, they contribute to the development of adaptable and stable applications.
- Data models improve communication within organisations, ensuring a shared understanding of data elements and relationships.

Importance of Data Models

- Data models provide a structured blueprint for organising and representing information in databases.
- Acting as a common language, data models facilitate effective communication among stakeholders.
- Data models serve as guidelines, directing developers in creating consistent and organised databases.
- They enable flexible systems, accommodating changes in data structures without major disruptions to operations.
- Enforcing relationships and constraints, data models contribute to maintaining accurate and reliable data.

Advantages of Data Models

- Data models prevent future system risks and failures by defining data structures in advance.
 - Early system visualisation reduces project costs through better planning and estimation.
 - Data models identify and eliminate data repetition, ensuring consistency and reliability.
 - Models aid in improving Graphical User Interfaces, aligning system design with user preferences.
 - Early visualisation facilitates stakeholder communication, aligning expectations and ensuring project success.
-

File Management Systems

- Initial systems stored diverse data, like payroll and accounting records, as individual files.
- File management systems lacked data models, treating files uniformly without internal content awareness.
- File content details were embedded in applications, as file management systems knew little about internal structures.
- Maintaining large file-based systems faced inefficiencies, leading to the development of database management systems.
- The evolution to database management systems centralised data definitions, offering efficient and structured management.

Hierarchical Databases

- Data in a hierarchical database is organised in a tree structure with a single root node and parent-child relationships.
- Nodes in the hierarchy have parent-child connections, where each node (except the root) has one parent and can have multiple children.
- Relationships in hierarchical databases are typically one-to-many, meaning a parent node can have multiple child nodes.
- Accessing data in a hierarchical database is done through navigational methods, moving through the tree structure to retrieve information.
- Hierarchical databases are less flexible compared to relational databases, making it challenging to represent complex relationships or accommodate changes easily.

<u>Advantages</u>	<u>Disadvantages</u>
<ol style="list-style-type: none"> 1. Hierarchical databases are efficient for certain types of queries, especially those that follow the natural tree structure, as navigation is straightforward. 2. The hierarchical structure enforces data integrity, as each child node must have a parent, ensuring a level of consistency in the data. 3. Retrieving data from a hierarchical database can be fast, especially when accessing a specific branch or subtree of the hierarchy. 4. The hierarchical model is relatively simple to understand and implement, making it suitable for specific applications with well-defined parent-child relationships. 	<ol style="list-style-type: none"> 1. The rigid structure of hierarchical databases limits flexibility, making it challenging to represent complex relationships that do not fit neatly into a tree structure. 2. Data redundancy can be an issue, especially when multiple occurrences of similar data are stored at different levels of the hierarchy. 3. As the size of the database grows, managing and navigating through the hierarchy can become complex, leading to scalability issues. 4. Modifying the structure of a hierarchical database can be challenging, as changes may require altering the entire hierarchy, impacting existing applications and data.

Network Database

- In a network database, data is organised using a graph-like structure, where nodes represent records, and edges define relationships between records.
- Records in a network database can have different types, and relationships between records are established through sets and pointers.
- Unlike hierarchical databases, network databases support many-to-many relationships, allowing records to be linked to multiple other records.
- Network databases facilitate complex queries by allowing navigational access through the interconnected records, offering flexibility in data retrieval.
- The network model provides more flexibility in schema design compared to hierarchical databases, making it easier to represent diverse relationships and adapt to changing requirements.

<u>Advantages</u>	<u>Disadvantages</u>
<ol style="list-style-type: none"> 1. Network databases excel in representing and managing many-to-many relationships, providing flexibility in modelling complex data structures. 2. The network model enforces data integrity through its interconnected structure, as each record type is related to others through explicit relationships. 3. Network databases offer greater flexibility in schema design compared to hierarchical databases, making it easier to adapt to changing data requirements. 4. Navigational access in network databases allows for the execution of complex queries, providing efficient retrieval of interconnected data. 	<ol style="list-style-type: none"> 1. Designing and implementing a network database can be complex, requiring a thorough understanding of the structure and relationships, which may pose challenges for developers. 2. As the database size increases, navigating through interconnected records can become complex, potentially leading to scalability issues. 3. Network databases lack a standardised query language, unlike relational databases with SQL, reducing consistency and interoperability. 4. Altering the structure of a network database is challenging, especially with large datasets or significant schema changes, requiring substantial effort.

Basic Building Block

The foundational elements for any model include entities, attributes, relationships, and constraints.

Entity

A fundamental component with independent existence in the real world, such as a Student, Faculty, or Subject. Entities may have physical or logical existence (e.g., Department, Section, or conditions like age > 18).

Attributes

Properties of entities that describe them, such as name, age, or phone for an Employee. Each entity has specific values for its attributes.

Relationships

Associations between entities, like "Employee works for Department." The degree of a relationship is the number of participating entity types.

Degree

The degree of a relationship type is the number of participating entity types in a specific relation within the data model.

Types of Relationships

Types of relationships in the context of database modelling refer to the nature and connectivity between entities. Here are common types:

1. **One-to-One (1:1):**
 - 1.1. Each record in the first entity corresponds to exactly one record in the second entity, and vice versa.
 2. **One-to-Many (1:N):**
 - 2.1. Each record in the first entity can relate to multiple records in the second entity, but each record in the second entity corresponds to only one record in the first entity.
 3. **Many-to-One (N:1):**
 - 3.1. Similar to one-to-many, but reversed. Multiple records in the first entity can relate to a single record in the second entity.
 4. **Many-to-Many (N:N):**
 - 4.1. Multiple records in the first entity can relate to multiple records in the second entity, and vice versa.
-

Business Rules

- Business rules are explicit statements defining operational policies or practices within an organisation.
- They impose constraints on business behaviour, influencing and controlling how certain activities are conducted.
- Business rules serve as guidelines for system behaviour, helping manage business change and ensuring consistency in operations.

Characteristics of Business Rules

1. Atomicity:

- 1.1. Business rules should address a singular aspect of the system environment, focusing on specific and distinct operational elements.

2. Business Format:

- 2.1. Expressing rules in a business format ensures they are understandable to business people, utilising formats like ER diagrams or object diagrams.

3. Business Ownership:

- 3.1. Each rule is assigned to a businessperson who is responsible for verifying, enforcing, and monitoring its compliance, fostering accountability.

4. Classification:

- 4.1. Business rules can be classified based on data and constraints, providing a structured approach to understanding and categorising them.

5. Business Formalism:

- 5.1. Business rules should be implementable in the related information system, maintaining consistency and avoiding redundancy in their application.

Types of Business Rules

1. Definitions:

- a. Define business terms, incorporating them into the system's data dictionary. Example: "A professor is someone who teaches students."

2. Facts:

- a. Connect business terms in meaningful ways, implemented as relationships between various data entities. Example: "A professor may have students."

3. Constraints:

- a. Demonstrate connections between business rules and terms, often specifying limitations on the relationships between data entities. Example: "Each professor may teach up to four subjects."

4. Derivations:

- a. Enable new knowledge or actions, typically implemented as formulas and triggers. Example: "A student's pending fees equal fees paid minus total fees."

Degrees of data Abstractions

To ensure system usability and efficiency, data representation in a database involves several levels of abstraction, each serving a specific purpose:

Physical Level:

Describes the lowest level of abstraction, focusing on how data is stored in detail, including the use of complex data structures.

Logical Level:

Represents a higher level of abstraction, describing what data is stored in the database and the relationships among them. It simplifies the database structure for users by using relatively simple structures.

View Level:

Represents the highest level of abstraction, focusing on only part of the entire database. Views exist to simplify user interactions, providing tailored perspectives and hiding unnecessary complexity. Multiple views may be defined for the same database.

Chapter3: DATABASE DESIGN

ER Relationship Model

- Entity
- Entity Set
- Attributes
- Relationship
- Simple and Composite attributes
- Single-valued and multivalued attributes
- Derived attribute

Constraints

- Mapping Cardinalities
- Participation Constraints
- Keys

ER Diagram

- Mapping Cardinality
- Strong Entity Set
- Weak entity set

ERD issues

Codd's Rules

Codd's Rules

Relational Schema

- Representation of Strong Entity Sets with Simple Attributes
- Representation of Strong Entity Sets with Complex Attributes
- Representation of Weak Entity Sets

Introduction to UML

- Class diagram
- Use case diagram
- Activity Diagram
- Implementation diagram
- Advantages of UML Diagrams
- Disadvantages of UML Diagrams

ER Relationship Model

The Entity-Relationship (ER) model is a blueprint for creating databases. It involves:

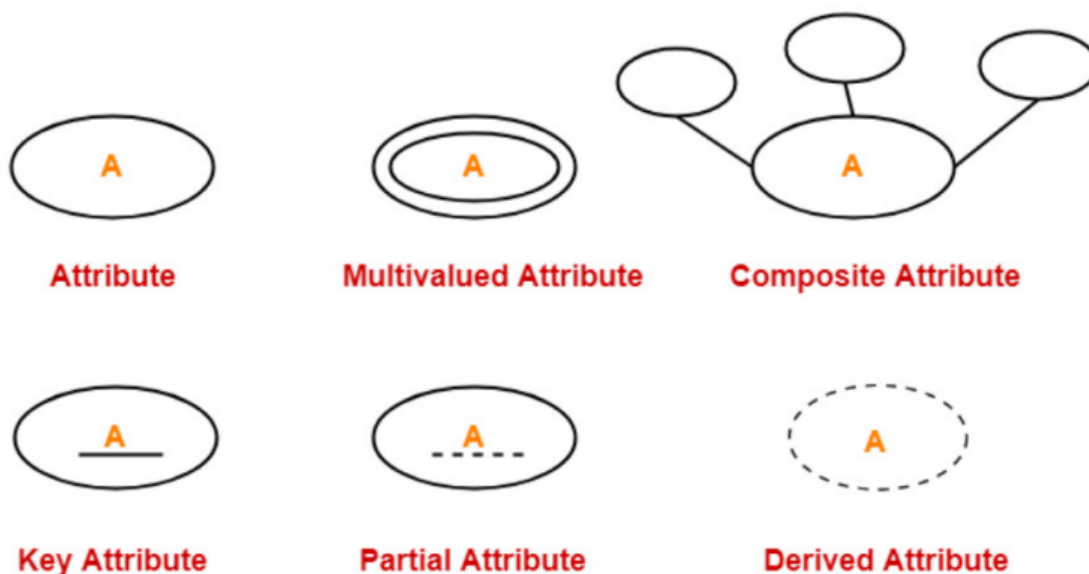
- **Entity:** A real-world object, like a person or product.
- **Entity Set:** A collection of similar entities, like a set of all customers.
- **Attributes:** Characteristics describing entities, such as a person's name.
- **Relationship:** Association between entities, e.g., a customer buying a product.
- **Simple and Composite Attributes:** Simple attributes are basic, like a person's age. Composite attributes are made up of smaller parts, like a person's full address.
- **Single-valued and Multivalued Attributes:** Single-valued attributes have one value, like a person's birthdate. Multivalued attributes can have multiple values, like phone numbers for a person.
- **Derived Attribute:** An attribute whose value can be calculated from other attributes, such as age derived from a birthdate.

Understanding these concepts helps in designing effective databases.

- **Entity**
 - In the context of the Entity-Relationship model, an entity is a distinct real-world object or concept with identifiable properties, such as a person, place, or product.
- **Entity Set**
 - An entity set is a collection of similar entities in the Entity-Relationship model, grouping instances that share common characteristics, like a set of all customers or products.
- **Attributes**
 - Attributes are characteristics or properties of entities in the Entity-Relationship model, providing details about real-world objects, such as a person's name or an item's price.
- **Relationship**
 - In the Entity-Relationship model, a relationship signifies an association between entities. It describes how entities interact or are connected, capturing the connections between different real-world objects or concepts.
- **Simple and Composite Attributes**
 - Simple attributes are basic, indivisible characteristics of entities (e.g., age). Composite attributes are made up of smaller parts, representing more complex details (e.g., full address).
- **Single-valued and Multivalued Attributes**
 - Single-valued attributes have one value, like a person's birthdate. Multivalued attributes can have multiple values, such as phone numbers for a person, representing varied information.

• Derived Attributes

- Derived attributes in the Entity-Relationship model are those whose values can be calculated or derived from other attributes, providing information that is not stored explicitly but inferred.



Constraints

Constraints are rules and limits that ensure data accuracy and integrity. They dictate how data in tables should behave, covering aspects such as uniqueness, relationships between tables, and the validity of data entered. Constraints prevent errors, maintain consistency, and uphold the overall quality of the database. Key types include Primary Key, Unique, Foreign Key, Check, Default, and NotNull constraints.

Mapping Cardinalities

Mapping Cardinalities define how instances of one entity relate to instances of another in a database, specifying one-to-one, one-to-many, many-to-one, or many-to-many relationships.

• One-to-One (1:1):

- Each instance in one group is associated with at most one instance in another group, and vice versa.

• One-to-Many (1:N):

- Each instance in one group can be associated with multiple instances in another group, but each instance in the second group is associated with at most one instance in the first group.

• Many-to-One (N:1):

- The reverse of One-to-Many, where each instance in the second group can be associated with at most one instance in the first group, but each instance in the first group can be associated with multiple instances in the second group.

- **Many-to-Many (M:N or N:M):**

- Each instance in one group can be associated with multiple instances in another group, and vice versa.

Participation Constraints

- **Total Participation:**

- Every entity in a certain group must participate in at least one relationship.

- **Partial Participation:**

- Only some entities in a group participate in relationships; participation is optional for others.

Keys

A key in a database is a column or set of columns that uniquely identifies each record in a table. It serves as a crucial identifier for ensuring data integrity and efficient data retrieval.

Keys can be classified into various types:

- **Primary Key:**

- A candidate key selected as the main reference key for the table.

- **Foreign Key:**

- A field in a table that refers to the primary key in another table, establishing relationships.

- **Unique Key:**

- Ensures uniqueness of values across a column or set of columns.

- **Super Key:**

- Any combination of fields within a table that uniquely identifies each record.

- **Candidate Key:**

- A minimal super key, representing the least combination of fields uniquely identifying records.

- **Composite Key:**

- A key consisting of multiple attributes, providing unique identification.

- **Simple Key:**

- A key consisting of a single attribute.

ER Diagram (Entity Relationship Diagram)

An ER (Entity-Relationship) diagram visually represents a database structure, illustrating entities, attributes, and relationships between them. It employs standardized symbols to convey the logical organization of data, aiding in database design by depicting how entities interact and how data is organized within a relational database.

Entity:

An entity represents a real-world object or concept, such as a person, place, thing, or event, that can be uniquely identified and distinguished from other entities in the database.

Attribute:

An attribute is a property or characteristic of an entity. It provides more information about the entity and is depicted as ovals in an ER diagram.

Relationship:

A relationship describes how two or more entities are connected or associated with each other. Relationships can have a degree (binary, ternary, etc.) based on the number of participating entities.

Cardinality:

Cardinality defines the numerical relationship between entities in a relationship. It specifies how many instances of one entity can be related to instances in another entity. Common cardinality types include "one-to-one," "one-to-many," and "many-to-many."

Weak Entity:

A weak entity is an entity that cannot be uniquely identified by its own attributes and relies on a related entity, known as a "strong entity," for identification.

Subtype and Supertype:

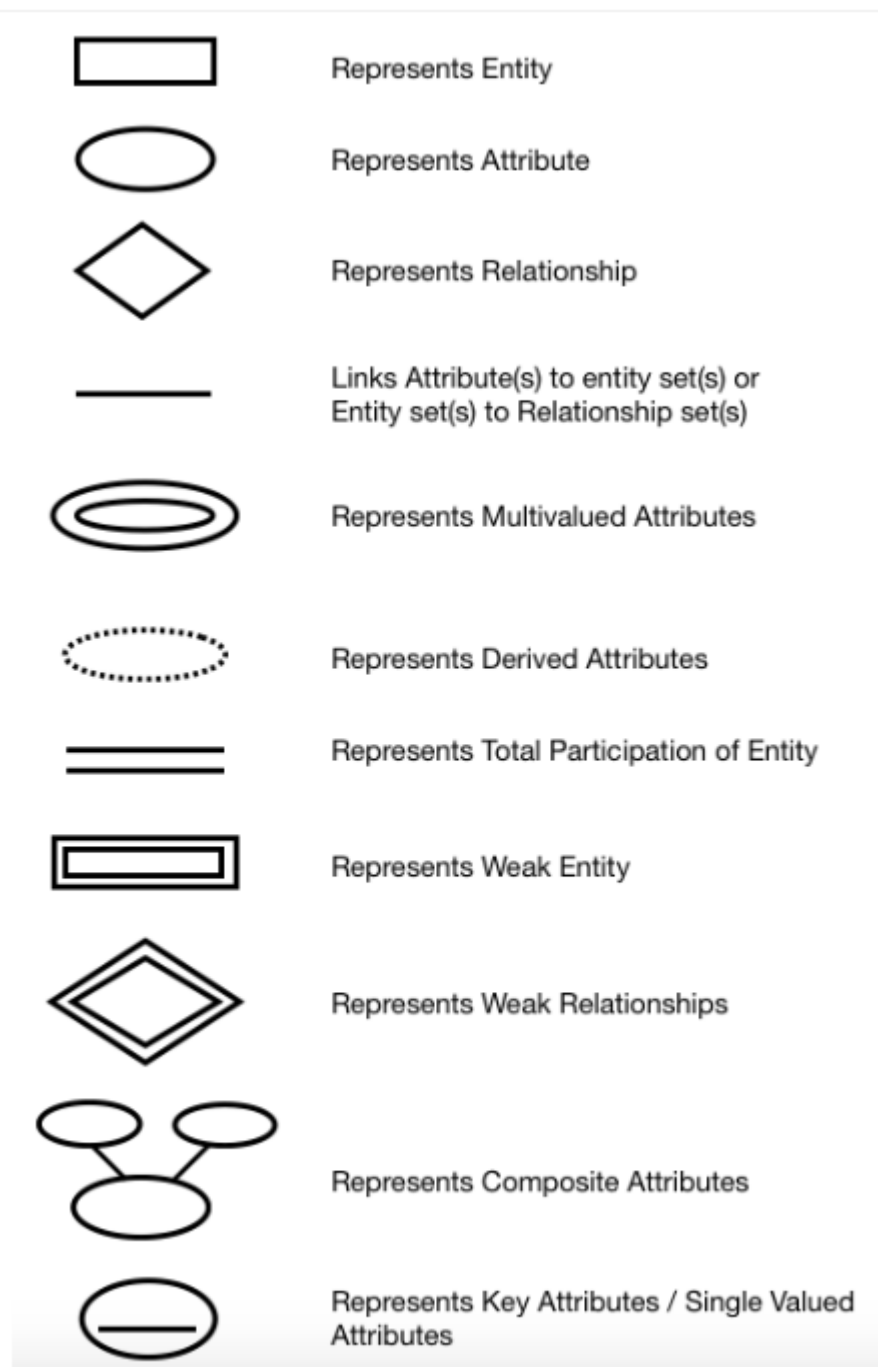
Subtype and supertype relationships represent generalization and specialization in an entity hierarchy. A supertype is a generalized entity, while a subtype is a more specific entity.

Recursive Relationship:

A recursive relationship occurs when an entity is related to itself through a relationship. This is common in scenarios where entities have relationships with other instances of the same entity.

Symbols used in ER diagram

- **Rectangle (Entity):** Represents an entity.
- **Ellipse (Attribute):** Represents an attribute.
- **Diamond (Relationship):** Denotes a relationship between entities.
- **Line (Relationship Link):** Connects entities and indicates relationships.
- **Double Ellipse (Multivalued Attribute):** Represents a multivalued attribute.
- **Double Diamond (Weak Entity):** Denotes a Weak Entity.
- **Double Rectangle (Subtype/Supertype):** Represents a subtype/supertype relationship.
- **Straight Line with a Dash :** Denotes a total participation constraint.
- **Crow's Foot (three lines):** Represents "many" in a one-to-many relationship.
- **Oval with Line underlined (Derived Attribute):** Denotes a derived attribute.



Components of ER Diagram in dbms

Strong entity set

- **Representation:** A strong entity set is symbolized by a single rectangle.
- **Primary Key:** It contains enough attributes to autonomously form a primary key for unique identification.
- **Relationship Representation:** The relationship between two strong entity sets is indicated by a diamond symbol.
- **Connection:** A single line connects the strong entity set to the relationship.
- **Total Participation:** The connection may or may not involve total participation, signifying if every entity in the set must participate in the relationship.

Weak entity set

- **Representation:** A weak entity set is represented by a double rectangle.
 - **Primary Key:** Lacking sufficient attributes to autonomously form a primary key, it relies on the attributes inherited from the related strong entity set, known as the "owning" or "parent" entity.
 - **Identifying Relationship Representation:** The identifying relationship between the weak entity set and the strong entity set is symbolised by a double diamond.
 - **Connection:** A double line visually depicts the connection between the weak entity set and the relationship set.
 - **Total Participation:** Total participation always exists in the identifying relationship, signifying that every weak entity must participate in the relationship for proper identification.
-

ERD Issues

- **Primary Key Misuse:**
 - Using primary keys as attributes in other entity sets instead of establishing relationships.
 - **Ambiguity in Object Representation:**
 - Difficulty in determining whether an object is best expressed as an entity set or a relationship.
 - **Binary vs. Non-Binary Relationships:**
 - Common confusion in representing relationships, with some non-binary relationships better represented by multiple binary relationships.
-

Codd's Rules

- **Information Rule:**
 - All available data in a system should be represented as relations or tables.
- **Guaranteed Access Rule:**
 - Each data item must be accessible without ambiguity, specifying the table name, primary key, and column name.
- **Systematic Treatment of Null Values:**
 - Null values, distinct from blank or zero, are unknown and should be treated appropriately.
- **Dynamic Online Catalog (Self-Describing Database):**
 - A dynamic online catalog, based on the relational model, should provide information about tables and data in the database.

- **Comprehensive Data Sublanguage:**
 - The data access language (SQL) must be the sole means of accessing data, supporting Data Manipulation Language (DML) and Data Definition Language (DDL).
 - **View Updating Rule:**
 - All views of data should be theoretically updatable, allowing updates through the system.
 - **High-Level Insert, Update, and Delete:**
 - The query language must manipulate sets of rows in a table, facilitating high-level operations like INSERT, UPDATE, and DELETE.
 - **Physical Data Independence:**
 - Changes in the physical storage structure should not affect applications accessing data.
 - **Logical Data Independence:**
 - Changes to the database design should be transparent to users.
 - **Integrity Independence:**
 - Data integrity constraints, definable in the language, must be stored in the database catalog, not in application programs.
 - **Distribution Independence:**
 - Data can be stored centrally or distributed across multiple systems.
 - **Non-Subversion Rule:**
 - Constraints should not be bypassed by other languages; the relational database management system (RDBMS) rules should not be subverted.
-

Relational Schema

A relational schema acts as a logical blueprint for a relational database, outlining the structure and organization of data. It converts abstract designs, especially from the E-R model, into a relational format, providing a framework for comprehensive data representation within the database.

Representation of Strong Entity Sets with Simple Attributes:

When dealing with a strong entity set having simple descriptive attributes, the representation involves creating a schema for that entity set. Each attribute is distinctly listed in the schema, and every tuple in the corresponding relation represents a unique entity from the entity set. For instance, the "student" entity set with attributes ID, name, and tot_cred would be represented as: `student (ID, name, tot_cred).`

Representation of Strong Entity Sets with Complex Attributes:

In cases where a strong entity set possesses non-simple (composite) attributes, each component attribute is handled individually in the schema. For instance, for an entity set like "instructor" with a composite attribute "name," the schema would include separate attributes for first name, middle name, and last name.

Representation of Weak Entity Sets:

When dealing with a weak entity set "A" depending on a strong entity set "B," the representation involves creating a relation schema "A". This schema includes attributes from both sets, ensuring a comprehensive representation. If "A" has attributes a_1, a_2, \dots, a_m and "B" has a primary key $\{b_1, b_2, \dots, b_n\}$, the schema for "A" would be defined as: $\text{A}(a_1, a_2, \dots, a_m, b_1, b_2, \dots, b_n)$.

INTRODUCTION TO Unified Modeling Language (UML)

Unified Modeling Language (UML) is a standardized language for specifying, visualizing, and documenting software system artifacts. Proposed as a standard, UML is used to create specifications for various components within complex software systems, serving as a specification language in the field of software engineering.

Class Diagram in UML:

A vital UML diagram, the class diagram portrays static system structure, featuring classes, relationships, and key attributes, providing insights into software organization.

Use case diagram:

Illustrates user-system interactions in task steps, capturing system dynamics and gathering requirements. Identifies external/internal factors influencing system behavior for effective visualization in Unified Modeling Language (UML).

Activity Diagram:

A UML flowchart illustrating dynamic system activity flow. Supports executable system construction using forward and reverse engineering techniques.

Implementation diagram:

Illustrates system components and interconnections, detailing both software and hardware levels, offering a concise overview of the system's structural implementation.

Advantages of UML Diagrams:

- Most useful for visualizing and documenting software system designs.
- Effective for modeling large, complex software systems.
- Simple to learn, yet offers advanced features for analysts, engineers, designers, and architects.
- Specifies systems in an implementation-independent manner.
- Provides a flexible skeleton that can be refined and extended with additional features.
- Specifies functional requirements in an object-oriented manner.

Disadvantages of UML Diagrams:

- UML can be complex, especially for small projects, with a steep learning curve for beginners.
- Effectiveness relies on specialized tools.
- Developing detailed diagrams can be time-consuming.
- Risk of over-abstraction leading to confusion; some diagrams may introduce ambiguity.
- Lacks GUI modeling specification; inadequate for formally specifying serialization and object persistence in distributed systems.