COURSE CODE: CS2304

DATABASE MANAGEMENT SYSTEMS

Unit 1

Course Objectives

1. To introduce the **core principles and architectures** of modern database systems.
2. To apply **data modelling** techniques using **Entity-Relationship** and relational models.
3. To design and **normalize relational schemas** for efficient data storage.
4. To write effective **SQL and PL/SQL** programs for data manipulation and transaction control.
5. To understand the concepts of **query processing, optimization, and indexing.**
6. To explore the role of **NoSQL databases, Big Data systems, and distributed databases** in modern applications.

Course Outcomes

1. Design and construct conceptual database models using **ER and EER diagrams for real-life applications**.
2. Transform **high-level data models** into normalized relational schemas **using functional dependencies and synthesis techniques**.
3. Apply the concepts of normalization **to develop the quality relational data model**
4. Formulate and execute queries using **relational algebra, SQL, and develop procedural constructs using PL/SQL.**
5. Explore and implement modern database technologies such as **NoSQL and Big Data frameworks like MongoDB and Hadoop.**
6. Demonstrate understanding of **physical database structures, indexing mechanisms, and query optimization techniques**

Books

- **Text Books:**
- 1. **Abraham Silberschatz, Henry F. Korth, S. Sudarshan; ―Database System Concepts‖; 6th Edition, McGraw-Hill Education**
- 2. Ramez Elmasri, Shamkant B. Navathe; ―Fundamentals of Database Systems‖; 7th Edition, Pearson
- **Reference Books:**
- 1. Thomas M. Connolly, Carolyn E. Begg, "Database Systems: A Practical Approach to Design", Implementation, and Management, 6th Edition ;Pearson
- 2. Raghu Ramakrishnan, Johannes Gehrke; Database Management Systems‖, 3rd Edition; McGraw Hill Education
- 3. Kristina Chodorow, "MongoDB The definitive guide", O'Reilly Publications, ISBN: 978-93-5110-269-4, 2nd Edition.
- 4. Dr. P. S. Deshpande, "SQL and PL/SQL for Oracle 10g Black Book", DreamTech.
- 5. Ivan Bayross, SQL, PL/SQL: The Programming Language of Oracle, BPB Publication.
- 6. Reese G., Yarger R., King T., "Williums H, Managing and Using MySQL", Shroff Publishers and Distributors Pvt. Ltd., ISBN: 81 - 7366 - 465 – X, 2nd Edition.
- 7. Dalton Patrik, "SQL Server – Black Book", DreamTech Press.
- 8. Eric Redmond, Jim Wilson, "Seven databases in seven weeks", SPD, ISBN: 978-93-5023-918-6.
- 9. Jay Kreibich, Using SQLite, SPD, ISBN: 978-93-5110-934-1, 1st edition.

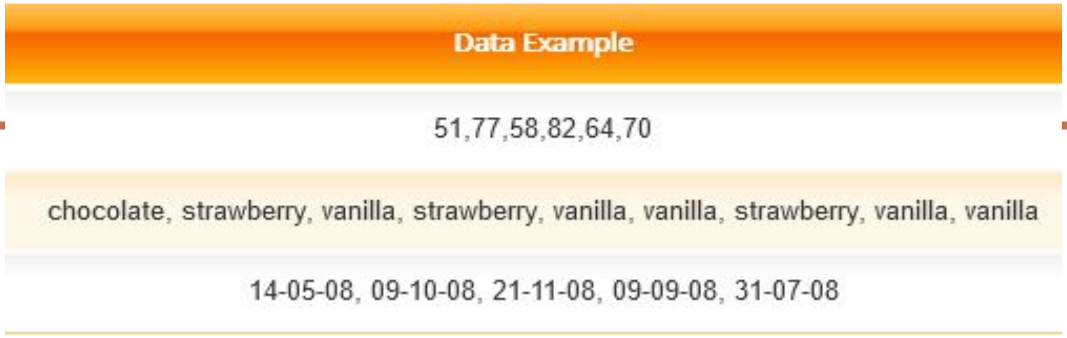
Assessment Scheme

Unit 1 Content

Introduction (4 Hours)

Introduction: Need of Database Management Systems, Evolution, Database System Concepts and Architecture, Database Design Process Data Modeling: Entity Relationship (ER) Model, Extended ER Model, Relational Model, Codd's Rules;

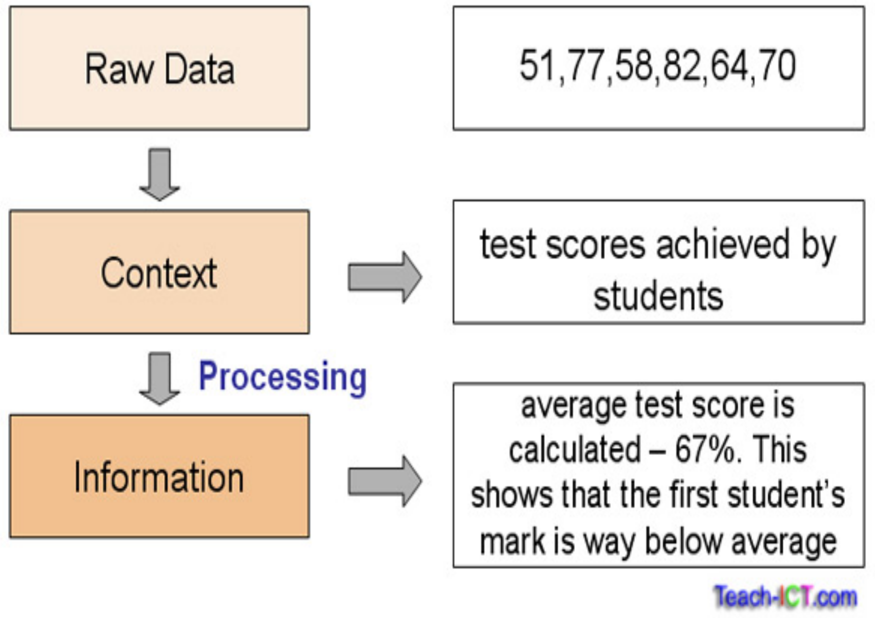
What is Data?


Data Example
51,77,58,82,64,70
chocolate, strawberry, vanilla, strawberry, vanilla, vanilla, strawberry, vanilla, vanilla
14-05-08, 09-10-08, 21-11-08, 09-09-08, 31-07-08

- Data is a collection of a distinct unit of information. This "data" is used in a **variety of forms of text, numbers, media and many more.**
- It **can be stored in pieces of paper or electronic memory, etc.**
- Data is basically information that can be translated into a particular form **for efficient movement and processing.**
- **Data consists of raw facts and figures - it does not have any meaning until it is processed and given a context.**
- **Data comes in many forms.** Although it is generally alphanumeric (text, numbers and symbols) it can also consist of images or sound.

What is Data?

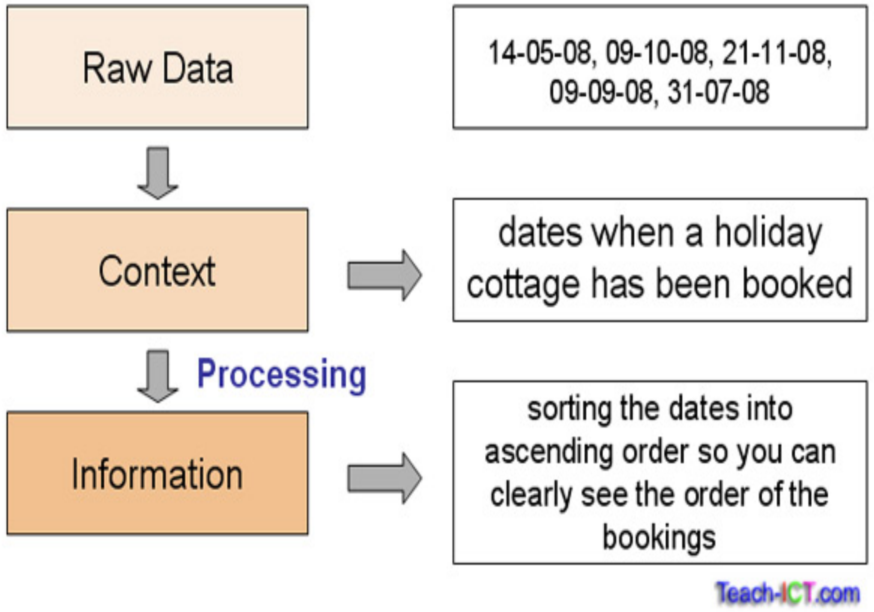
- In order to **turn data into information**, an organization needs to process the data.
- It then needs to present the **processed data in a context** which will be meaningful to the person who receives it.
- **INFORMATION = DATA + CONTEXT + MEANING**
- **Information is data which has been processed within a context in order to give it meaning**



51,77,58,82,64,70

test scores achieved by students

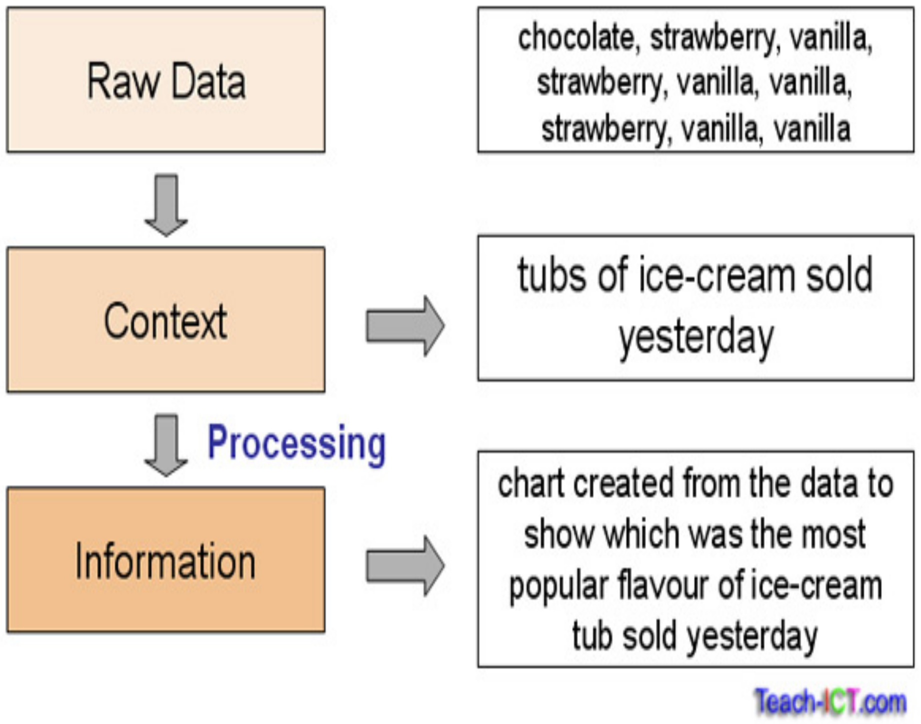
average test score is calculated – 67%. This shows that the first student's mark is way below average



14-05-08, 09-10-08, 21-11-08, 09-09-08, 31-07-08

dates when a holiday cottage has been booked

sorting the dates into ascending order so you can clearly see the order of the bookings



chocolate, strawberry, vanilla, strawberry, vanilla, vanilla, strawberry, vanilla, vanilla

tubs of ice-cream sold yesterday

chart created from the data to show which was the most popular flavour of ice-cream tub sold yesterday

Data Vs Information

Data

Data is a collection of facts, numbers, words, or observations that can be used to learn about something

- Example:
  - "25, 30, 35" (numbers with no context).
  - "**Red**, **Blue**, **Green**" (colors with no explanation of their relevance).
- Purpose: **Serves as the input for processing to generate information.**

Information

Processed, organized, or structured data that provides context, meaning, and relevance.

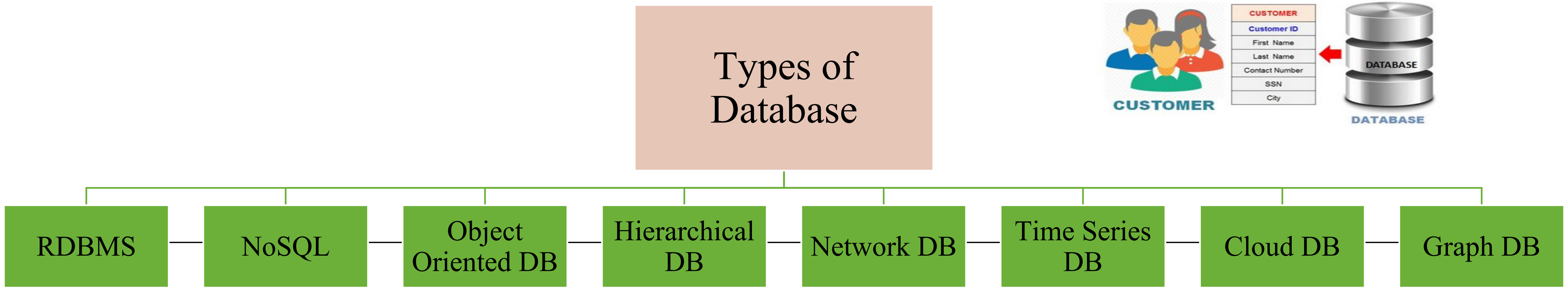
- Example:
  - "The average age of participants is 30 years." (derived from the data above).
  - "The most popular color among users is **Blue**." (interpreted from the colors data).
- Purpose: **Helps in understanding, making decisions, and communicating insights.**

Data Base

A **database** is an **organized collection of data that is stored and managed to ensure easy access, retrieval, and updating.** Databases are designed to store large amounts of information efficiently and allow users or applications to query and manipulate the data.

For example,

- consider **the names, telephone numbers**, and addresses of the people you know. You may have recorded this data in an indexed address book, or you may have stored it on a diskette, using a personal computer and software such as DBASE IV or V, Microsoft ACCESS, or EXCEL.
- The **best analogy is the library.** The library contains a huge **collection of books of different genres**, here the **library is database and books are the data.**
- In layman terms, **consider your school registry**. All the details of the students are entered in a single file. You get the details regarding the students in this file. This is called a Database where you can access the information of any student.



Why Database?

- The **main purpose** of the database **is to operate a large amount of information by storing, retrieving, and managing data.**
- Databases are useful in many different scenarios for storing data. It is typical to use a database when different sets of data needs to be linked together, such as:
    - **Students in a school and their grades**
    - **Customer records and sales information**
    - **Patients' and doctors' records**
    - **Transactions between different bank accounts**
    - **Taxpayers and income tax payments**
- The various reasons a database is important are
- **Manages large amounts of data**
- A database stores and manages a **large amount of data on a daily basis.** This would not be possible using any other tool such as a spreadsheet as they would simply not work.
- **Easy to update data**
- In a database, it is easy to update data using various **Data Manipulation languages (DML)** available. One of these languages is SQL.

Why Database?

- **Access**
- Access is about **making data available to users.**
- Databases support good data access because: **Large volumes of data can be stored in one place. Multiple users can read and modify the data at the same time**
- Databases are **searchable and sortable**, so the data you need can be **found quick and easily**
- **Security of data**
- Databases have various methods to ensure security of data. There are **user logins required before accessing a database and various access specifiers.**
- Databases allow access to be controlled, allowing **users to have different privileges**: for example, some users may be able to read data, but not to write it.
- **Easy to research data**
- It is very easy to access and research data in a database. This is done using **Data Query Languages (DQL) which allow searching of any data in the database and performing computations on it.**

Data Base Management System (DBMS)

A **Database Management System (DBMS)** is a **software system that enables users to create, manage, and interact with databases efficiently.** It **serves as an interface between the database and the end-users or applications**, ensuring that data is organized, stored, and retrieved effectively while maintaining security and consistency.

Advantage of DBMS

- **Controls redundancy :** It stores all the data in a single database file, so it can control data redundancy.
- **Data sharing :** An authorized user can share the data among multiple users.
- **Backup:** It provides Backup and recovery subsystem. This recovery system creates automatic data from system failure and restores data if required.
- **Multiple user interfaces :** It provides a different type of user interfaces like GUI, application interfaces.

Disadvantage of DBMS

- **Size :** It occupies large disk space and large memory to run efficiently.
- **Cost :** DBMS requires a high-speed data processor and larger memory to run DBMS software, so it is costly.
- **Complexity :** DBMS creates additional complexity and requirements.

Features of a DBMS

- **Data Storage:**
  Organizes data in structured formats (e.g., tables, files, or documents).
- **Data Manipulation:**
  Allows CRUD operations: Create, Read, Update, Delete.
- **Querying:**
  Provides query languages like SQL (Structured Query Language) to retrieve and manipulate data.
- **Data Integrity:**
  Ensures data accuracy and consistency.
- **Security:**
  Controls access through authentication and permissions.
- **Concurrency Control:**
  Manages multiple user access simultaneously without conflicts.
- **Backup and Recovery:**
  Protects data through regular backups and restores in case of failures.
- **Data Independence:**
  Separates data from applications, allowing modifications without impacting programs.

Problem With File System

1. **Data redundancy and inconsistency.** When files and programs are created by different programmers over time, they may have varying formats and use different programming languages. This often results in duplicate data being stored in multiple locations, such as a customer's address appearing in both savings and checking account records. **Such redundancy increases storage and access costs and can cause inconsistency, where updates in one file (e.g., a changed address) are not reflected in others, leading to errors and mismatched information.**

Example: Telephone Number update in One file may not update in other file of Customer Information

2. **Difficulty in Accessing Data:** Accessing specific data in file-processing systems is challenging. For example, if a bank officer needs a list of customers within a specific postal code, there might be no program to generate it. **The officer must either manually filter all customer records or request a custom program.** If later, the officer needs only customers with account balances above $10,000, the same issue arises. Such systems lack flexibility for efficient data retrieval.

Problems in File System (Cont…)

3. Data isolation. Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

4. Integrity problems: Integrity problems arise when consistency constraints, **like a minimum bank balance, are hard to enforce and update across multiple programs and files.**

5. Atomicity problems: Atomicity problems occur when system failures cause inconsistent data. For example, if transferring $50 from account A to account B, a failure may remove the $50 from account A but not credit it to account B, leaving the database in an inconsistent state. Ensuring atomicity means the transaction must either complete fully or not occur at all, which is hard to manage in conventional file-processing systems.

6. Concurrent-access anomalies occur when multiple users update data simultaneously, leading to inconsistencies. For example, if two customers withdraw $50 and $100 from the same bank account, both may read the initial balance of $500 and update it, resulting in an incorrect final balance (either $450 or $400 instead of $350). Ensuring correct data requires supervision, but managing concurrent access is challenging in uncoordinated systems.

7. Security problems occur when users have unauthorized access to data. For example, in a banking system, payroll personnel should only access employee information, not customer accounts. However, due to the ad hoc addition of application programs, enforcing such security constraints becomes challenging.

Need of DBMS

1. **Efficient Data Storage and Retrieval:**
   A DBMS provides structured storage for large volumes of data and enables fast retrieval through indexing and querying techniques
2. **Data Integrity and Accuracy:**
   Ensures that data is accurate, consistent, and follows defined rules and constraints, reducing errors and inconsistencies.
3. **Data Security:**
   Controls access to sensitive data through authentication, authorization, and encryption, protecting it from unauthorized users.
4. **Minimized Data Redundancy:**
   Reduces duplicate data storage by centralizing data and preventing unnecessary copies, which helps optimize space and storage costs.
5. **Concurrent Access:**
   Supports multiple users accessing and modifying the database simultaneously, ensuring that concurrent operations do not cause conflicts or data inconsistency.
6. **Data Backup and Recovery:**
   Provides mechanisms for data backup and restoration, ensuring that data can be recovered in case of system failures or crashes.

Evolution of DBMS

Early File Systems -1950 -1960

Hierarchical (1960-1970) and Network Models – 1970 to 1980

Relational Model

1970 to Present

Object Oriented DB

1980 to 1990

and

NO-SQL DBMS

2000 to Present

NewSQL & Cloud Based DB

2010 to Present

Database System Concepts and Architecture

Database System Concepts and Architecture refer to the **fundamental principles, components, and structures that define how a database system is designed, implemented, and operated.** The **architecture outlines the different levels at which the database system operates and how users interact with the system**. Below is an overview of key database system concepts and architecture.

Database System Concepts

- Database
- DBMS (Database Management System)
- View of Data: Data Abstraction, Instances and Schemas
- Data Model

Relational Database Model

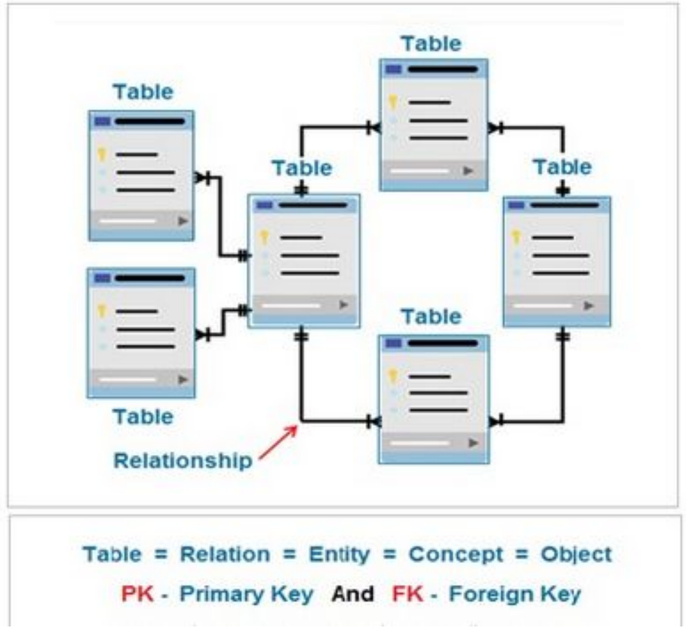
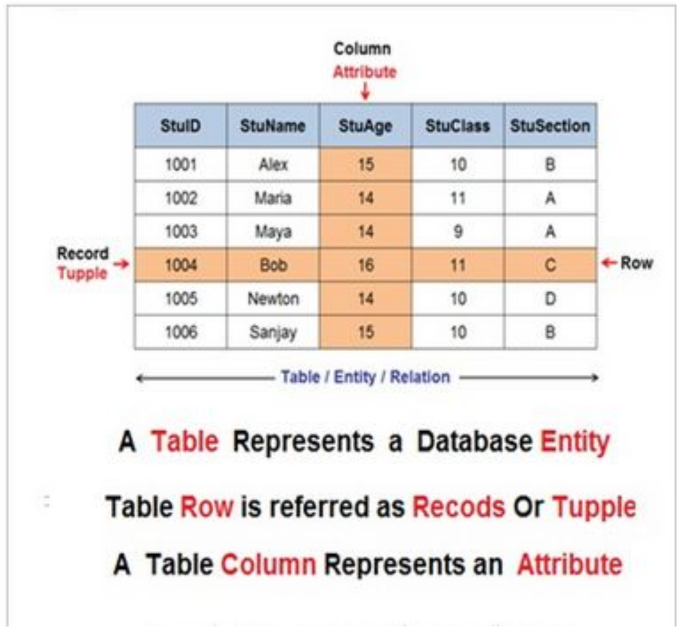


Table = Relation = Entity = Concept = Object

PK - Primary Key And FK - Foreign Key

RDBMS - Relational Table Example



StuID	StuName	StuAge	StuClass	StuSection
1001	Alex	15	10	B
1002	Maria	14	11	A
1003	Maya	14	9	A
1004	Bob	16	11	C
1005	Newton	14	10	D
1006	Sanjay	15	10	B

A Table Represents a Database Entity

Table Row is referred as Recods Or Tupple

A Table Column Represents an Attribute

View of Data

1. Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-systems users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

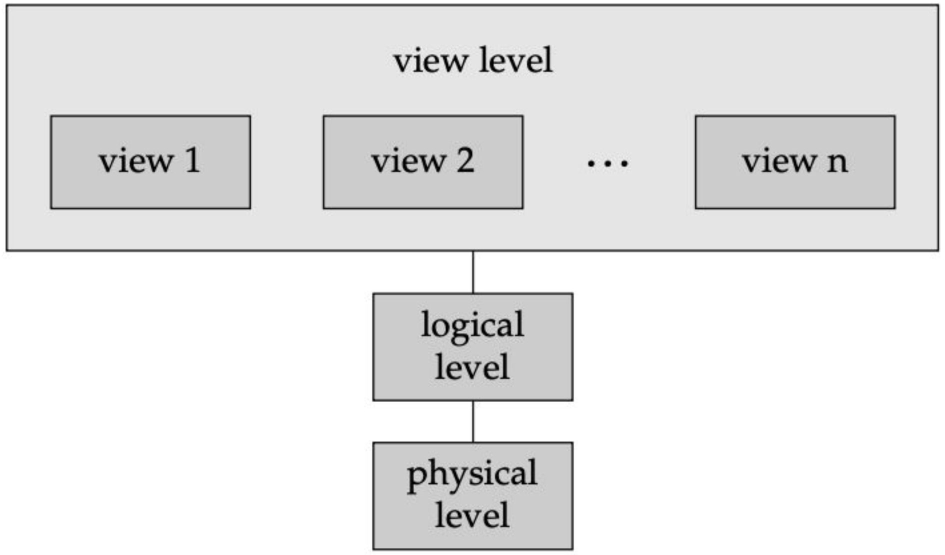


Fig: The three levels of data abstraction

- **Physical level.** The lowest level of abstraction **describes how the data are actually stored**. The physical level describes complex low-level data structures in detail.
- **Logical level.** The next-higher level of abstraction **describes what data are stored in the database,** and **what relationships exist among those data**. The logical level thus describes the entire database in terms of a small number of relatively simple structures
- **View level**. The highest level of abstraction **describes only part of the entire database**. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system.

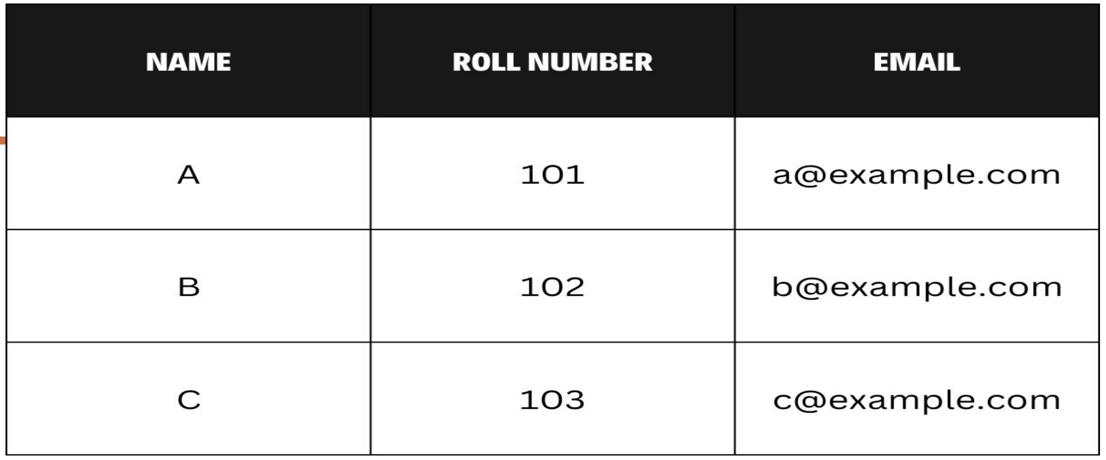
View of Data (Cont…)

2. Instances and Schema

•**Instances**: An instance **refers to the actual data stored in the database at any given point in time.** It is the content of the database as it exists for a specific moment.

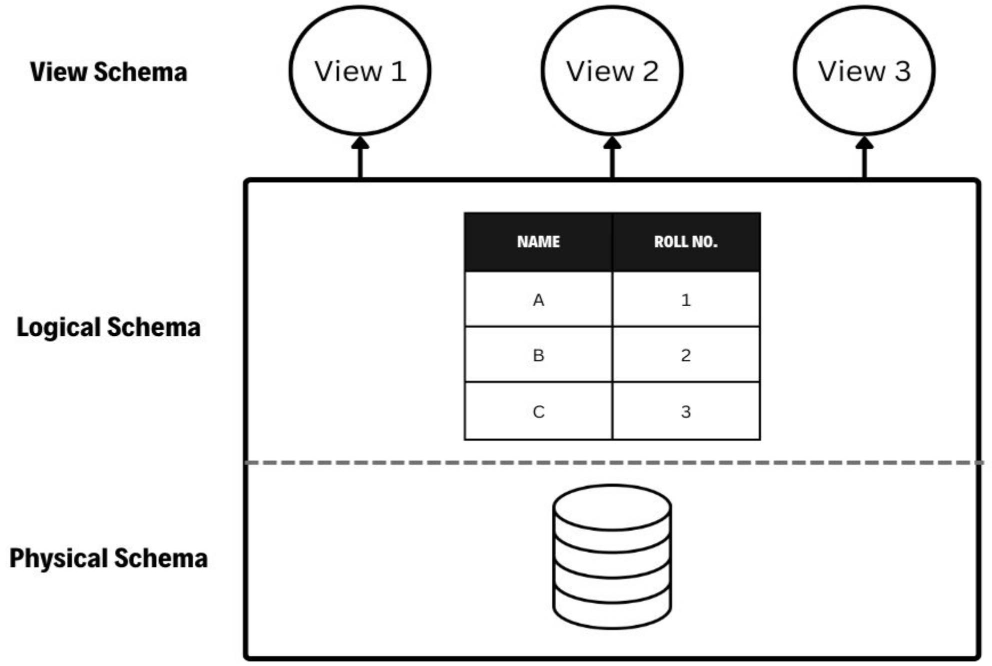
•**Schema**: The schema **defines the logical structure of the database, including the tables, relationships, and constraints**. It is a blueprint of how data is organized.

- **Physical Schema**: The physical schema describes the database design at the physical level. **Describes how data is stored on storage media (e.g., disk).**
- **Logical Schema**: Describes the logical view of data (e.g., **tables, views**).



NAME	ROLL NUMBER	EMAIL
A	101	a@example.com
B	102	b@example.com
C	103	c@example.com

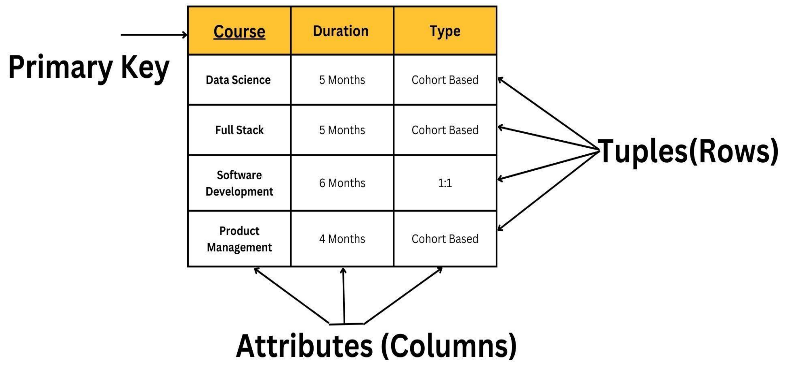
Instances



Schema

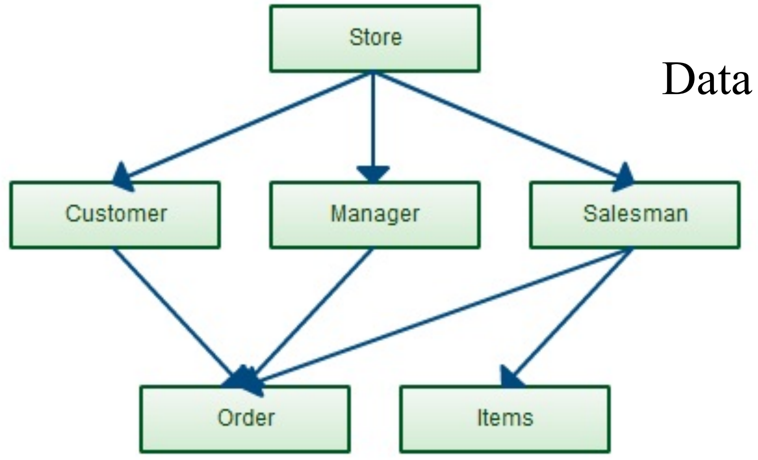
Data Model

A **data model** defines how data is represented, stored, and organized in a database system. Common data models include:

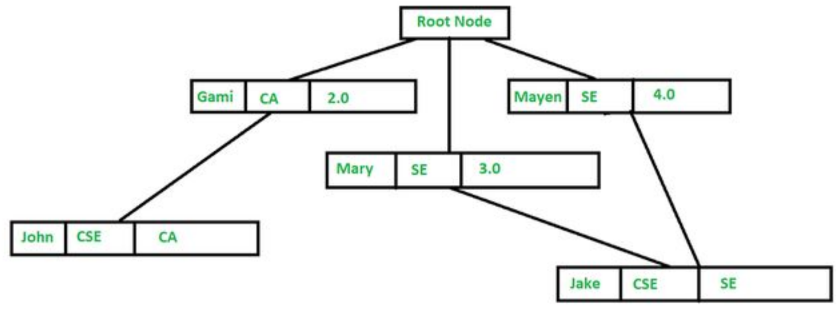

Primary Key
Tuples(Rows)
Attributes (Columns)

Course	Duration	Type
Data Science	5 Months	Cohort Based
Full Stack	5 Months	Cohort Based
Software Development	6 Months	1:1
Product Management	4 Months	Cohort Based

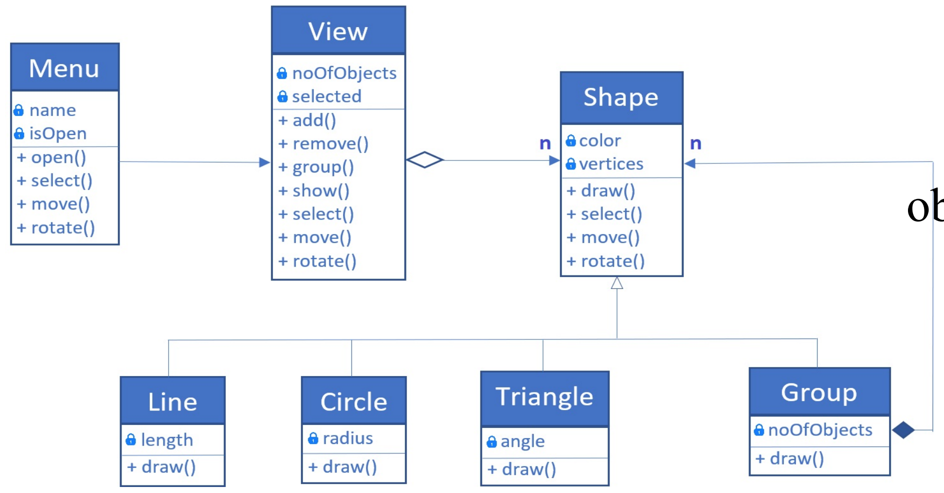
Relational Model
Data is stored in tables (relations).



Network Model
Data is organized in graph-like structures.



Hierarchical Model
Data is organized in a tree-like structure.

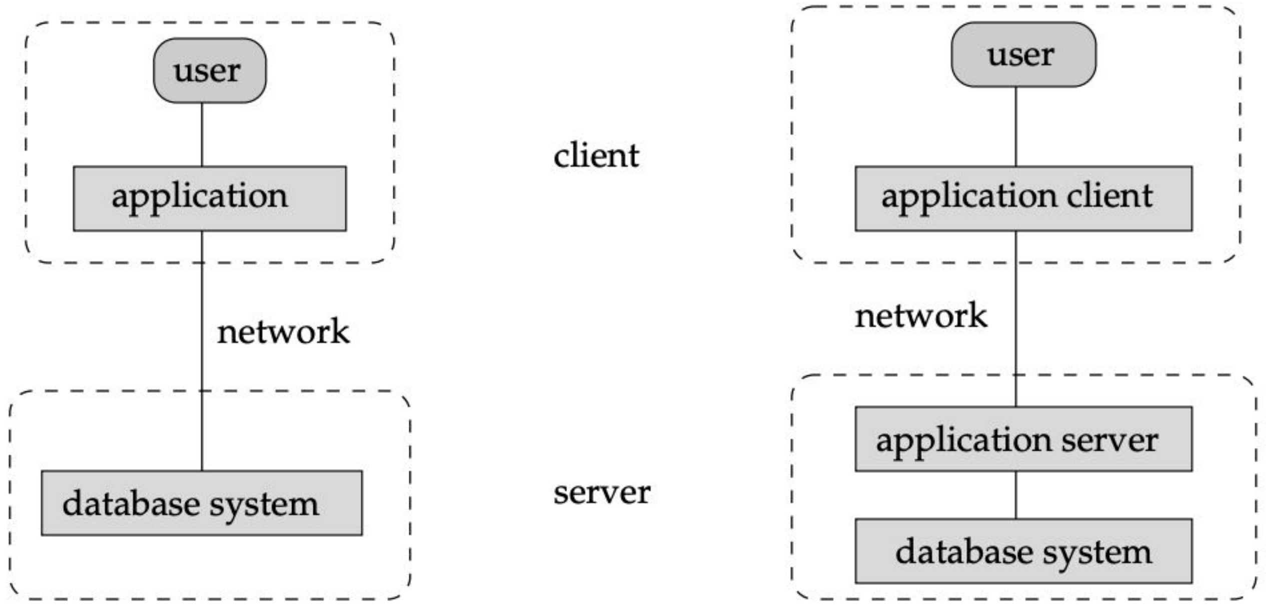


Object-Oriented Model
Data is represented as objects, similar to object-oriented programming

Client Server Architecture

In modern database systems, **client-server architecture** is commonly used, where:

- **Client**: The client application sends queries to the server and receives the results. The client interacts with the user and requests data from the database.
- **Server**: The database server is responsible for handling requests, processing queries, and interacting with the actual database to retrieve or modify data.
  - The **DBMS** runs on the server, and it manages the database, handles concurrency control, ensures security, and provides other essential services.
  - A client-server architecture can be classified into three tiers:

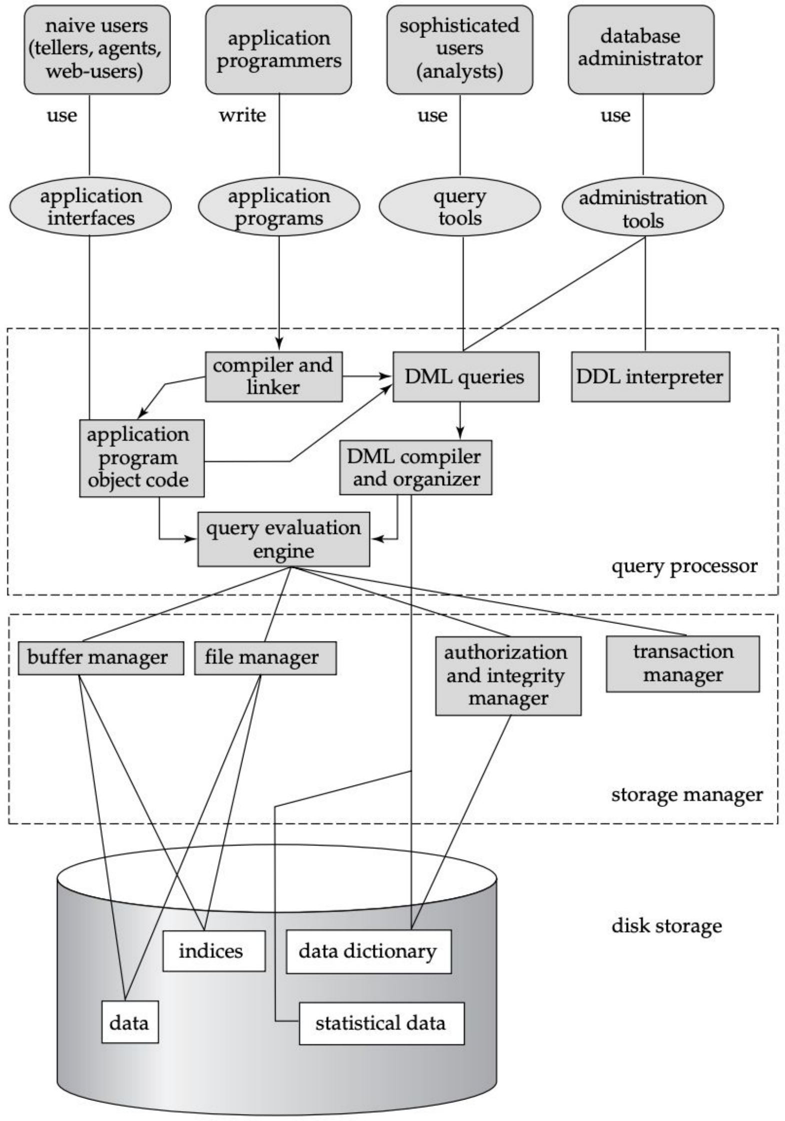


a. two-tier architecture b. three-tier architecture

Two-Tier: Direct communication between client and server (client requests data from the database server).

Three-Tier: An additional middle layer (application server) processes logic before interacting with the database server.

Detailed System Structure



A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components.

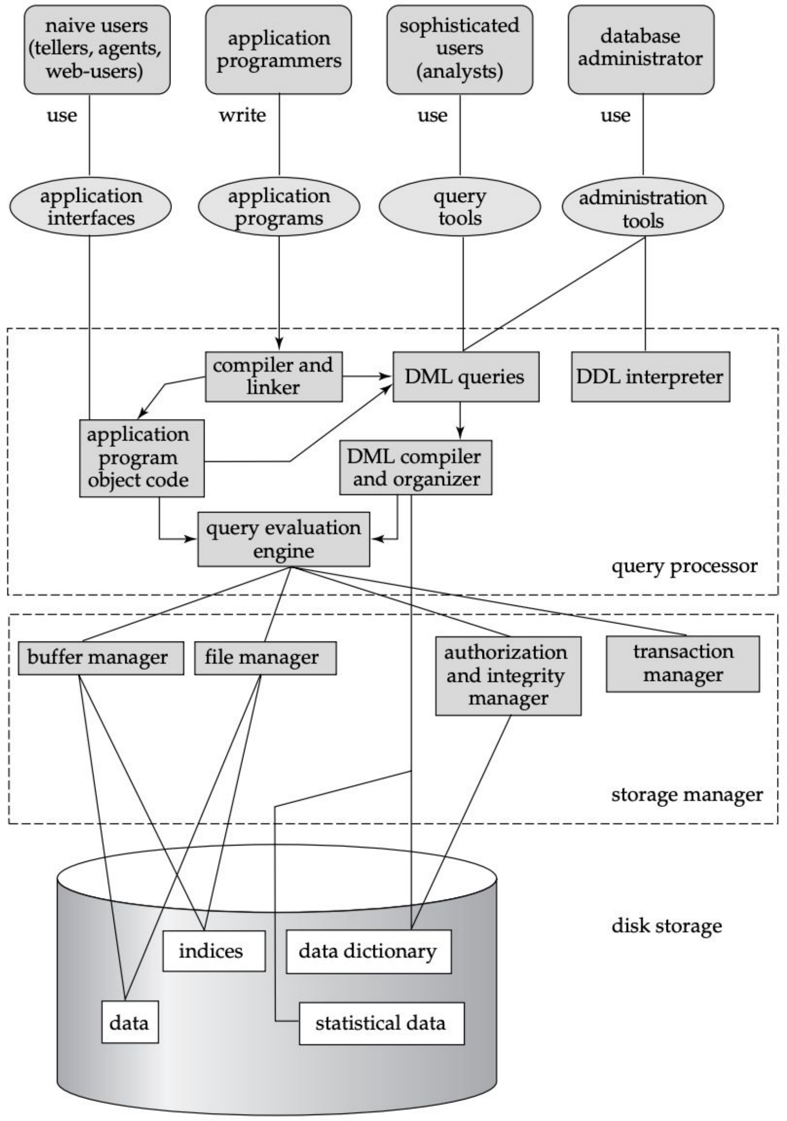
Storage Manager

A storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.

Storage Manager Components

1. **Authorization and Integrity Manager:** Ensures user access permissions and checks integrity constraints.
2. **Transaction Manager:** Maintains database consistency during system failures and manages concurrent transactions without conflict.
3. **File Manager:** Allocates disk storage space and manages data structures for stored information.
4. **Buffer Manager:** Fetches data from disk to main memory and decides caching strategy to handle large data sizes efficiently.
5. **Data Files:** Store the actual database.
6. **Data Dictionary:** Holds metadata about the database structure, including its schema.
7. **Indices:** Provide fast access to specific data values.

Detailed System Structure (Cont…)



The Query Processor

The query processor components include

- DDL interpreter: which interprets DDL statements and records the definitions in the data dictionary.
- DML compiler, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands. A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs query optimization, that is, it picks the lowest cost evaluation plan from among the alternatives.
- Query evaluation engine, which executes low-level instructions generated by the DML compiler.

Database Design Process
Data Modeling

Entity Relationship (ER) Model, keys, Extended ER Model, Relational Model, Codd's Rules;

The Database Design Process

Database Design:

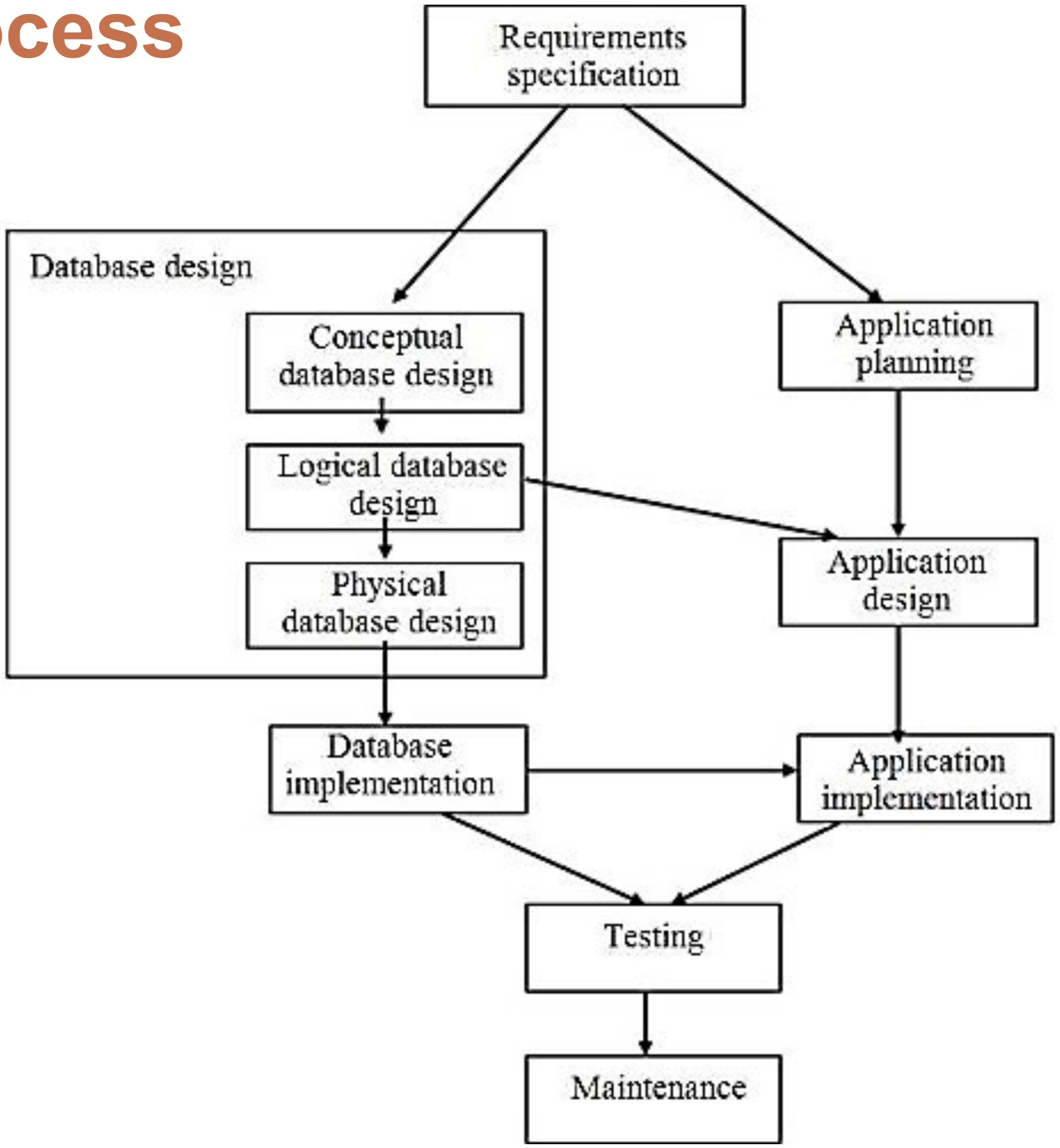
- **Database design can be generally defined as a collection of tasks or processes that enhance the designing, development, implementation, and maintenance of enterprise data management system**
- **It simply means mapping of conceptual model to implementation model.**

- **Phase 1:** Requirements Collection and Analysis
- **Phase 2:** Conceptual Database Design
- **Phase 3:** Choice of a DBMS
- **Phase 4:** Data Model Mapping (Logical Database Design)
- **Phase 5:** Physical Database Design
- **Phase 6:** Database System Implementation and Tuning

Why is Database Design important?

1. Database designs provide the blueprints of how the data is going to be stored in a system. A proper design of a database highly affects the overall performance of any application.
2. The designing principles defined for a database give a clear idea of the behavior of any application and how the requests are processed.
3. Another instance to emphasize the database design is that a proper database design meets all the requirements of users.
4. Lastly, the processing time of an application is greatly reduced if the constraints of designing a highly efficient database are properly implemented.

The Database Design Process



Phases of Database Design (contd.)

- Requirements Collections and Analysis
  - Identifying Users
  - Interacting with users to gather requirements
  - Time consuming BUT very important
- Conceptual Database Design
  - Produce a conceptual schema for the database that is independent of a specific DBMS
  - Involves two parallel activities
    - Conceptual Schema Design
    - Transaction and Application Design

Phases of Database Design (contd.)

- Approaches to Conceptual Schema Design
- Centralized Schema Design Approach
  - Also known as one-shot approach
  - Requirements of different applications and user groups are merged into a single set of requirements and a single schema is designed
  - Time consuming, places the burden on DBA to reconcile conflicts
- View Integration Approach
  - Schema is designed for each user group or application
  - These schemas are then merged into a global conceptual schema during the view integration phase
  - More practical

Phases of Database Design (contd.)

Strategies for Schema Design

- Top Down Strategy
  - Start with a schema containing high-level abstractions and then apply successive top-down refinements

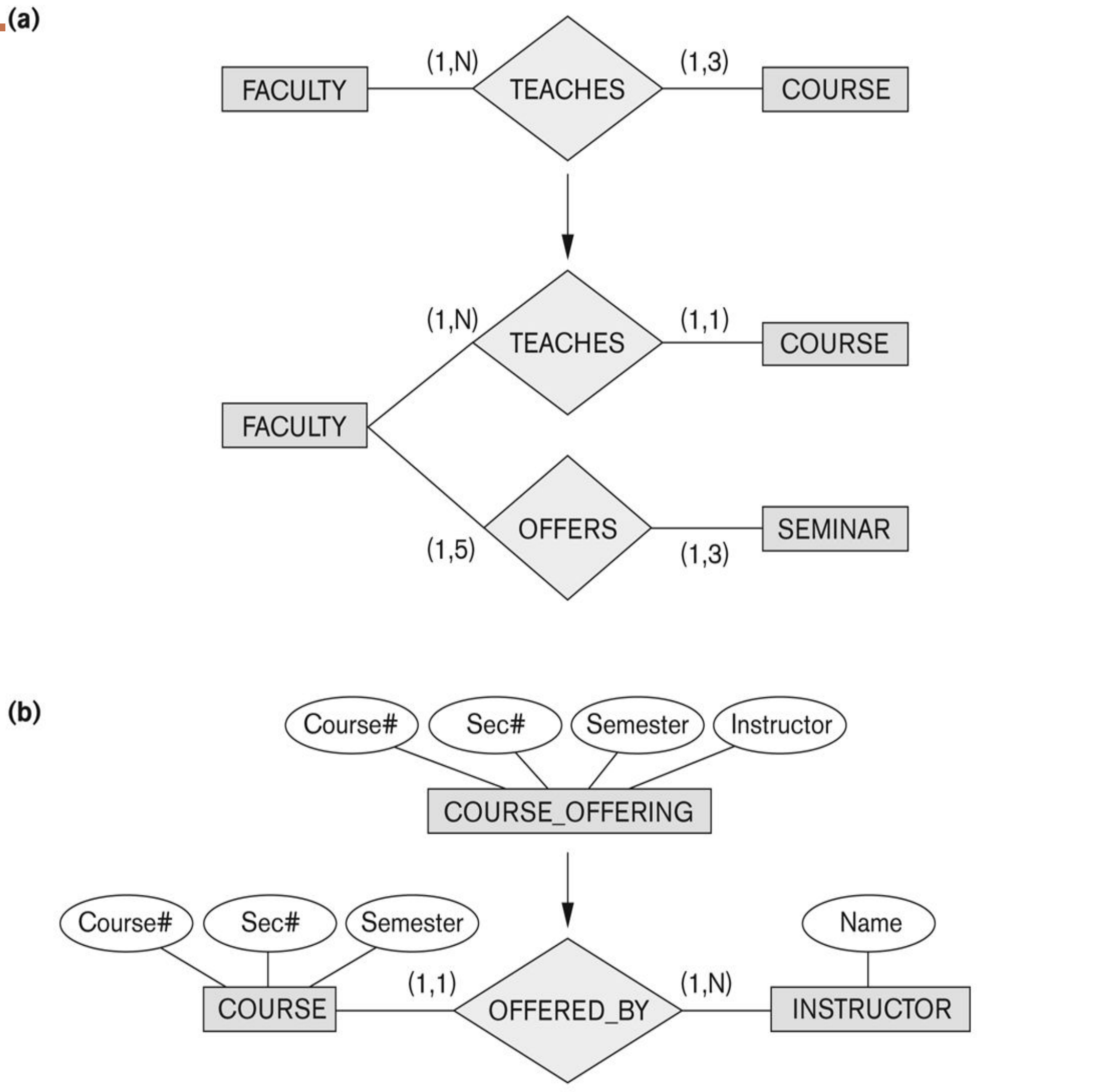


Figure 12.2
Examples of top-down refinement. (a) Generating a new entity type.
(b) Decomposing an entity type into two entity types and a relationship type.

Phases of Database Design (contd.)

Strategies for Schema Design (contd.)

- Bottom-Up Strategy
  - Start with a schema containing basics abstractions and then combine or add to these abstractions

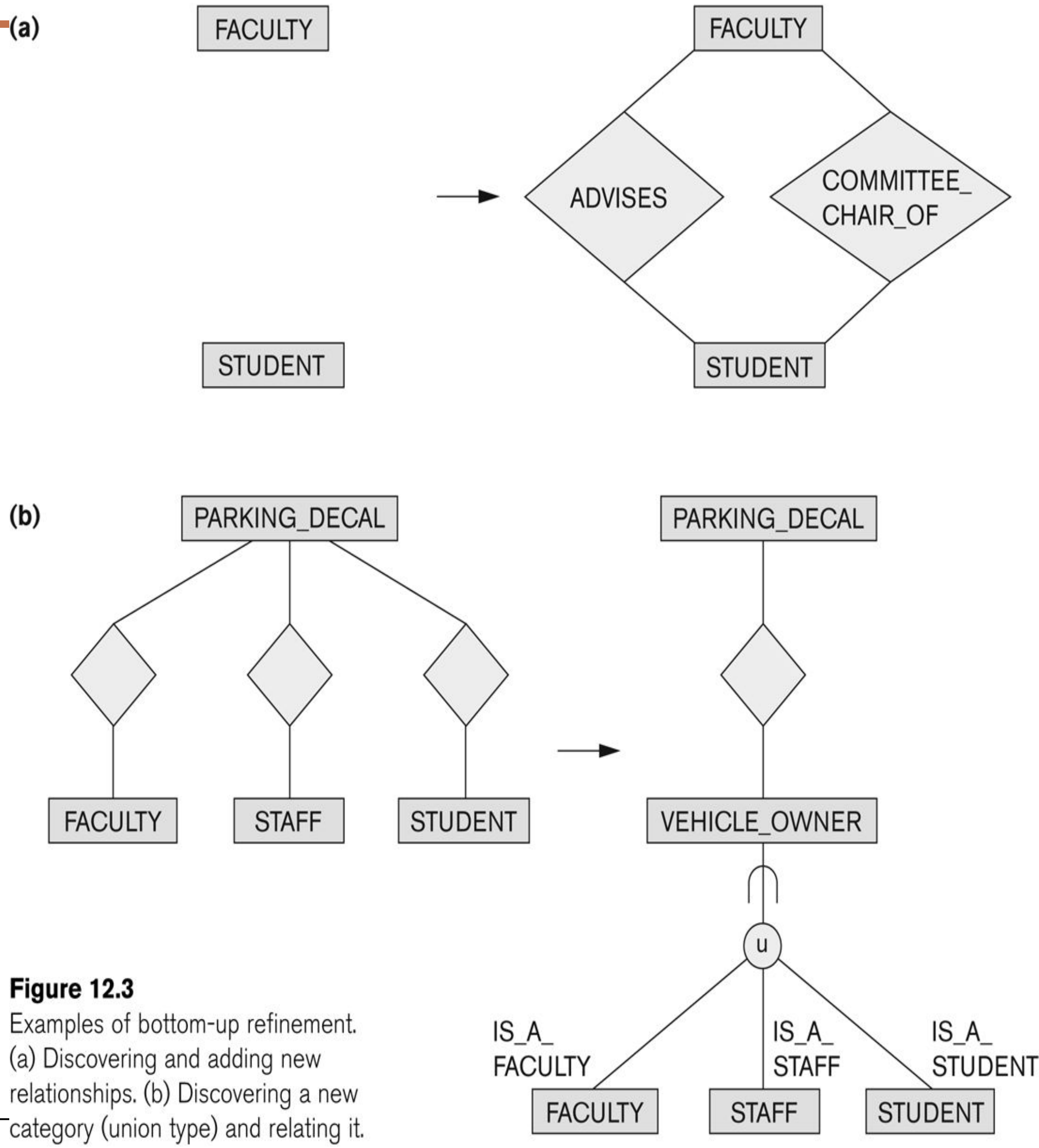


Figure 12.3
Examples of bottom-up refinement. (a) Discovering and adding new relationships. (b) Discovering a new category (union type) and relating it.

Phases of Database Design

- Choice of DBMS(Phase 3)
- Many factors to consider for choice of DBMS
- Technical Factors
    - Type of DBMS: Relational, object-relational, object etc.
    - Storage Structures
    - Architectural options
  - Economic Factors
    - Acquisition, maintenance, training and operating costs
    - Database creation and conversion cost
  - Organizational Factors
    - Organizational philosophy
      - Relational or Object Oriented
      - Vendor Preference
    - Familiarity of staff with the system
    - Availability of vendor services

Phases of Database Design (contd.)

- *Data model mapping (Phase 4):* During this phase, which is also called logical database design, we map (or transform) the conceptual schema from the high-level data model used in Phase 2 into the data model of the chosen DBMS. We can start this phase after choosing a specific type of DBMS.
- *Physical database design (Phase 5):* During this phase, we design the specifications for the stored database in terms of physical storage structures, record placement, and indexes. This corresponds to designing the *internal schema* in the terminology of the three-level DBMS architecture.

Phases of Database Design (contd.)

- *Database system implementation and tuning (Phase 6):*

During this phase, the database and application programs are implemented, tested, and eventually deployed for service. Various transactions and applications are tested individually and then in conjunction with each other. This typically reveals opportunities for physical design changes, data indexing, reorganization, and different placement of data is referred as database tuning. Tuning is an ongoing activity a part of system maintenance that continues for the life cycle of a database as long as the database and applications keep evolving and performance problems are detected.

Entity-Relationship Model

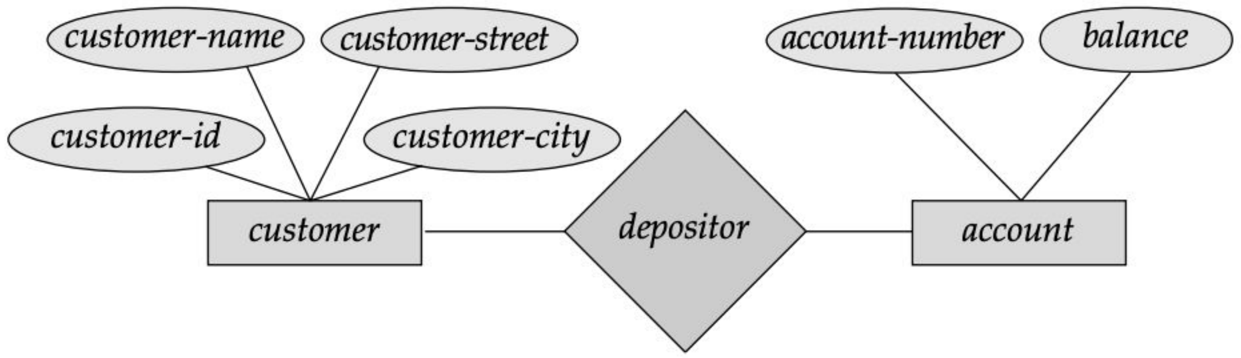
- Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.
- **ER Model is best used for the conceptual design of a database.**
- ER Model is based on −
- **Entities** and their *attributes.*
- **Relationships** among entities
- In ER modeling, the database structure is portrayed as a diagram called an **Entity-relationship diagram**

Entity → real-world object (Student, Course, Professor).

Attributes → properties of entities (Name, Age, CourseName).

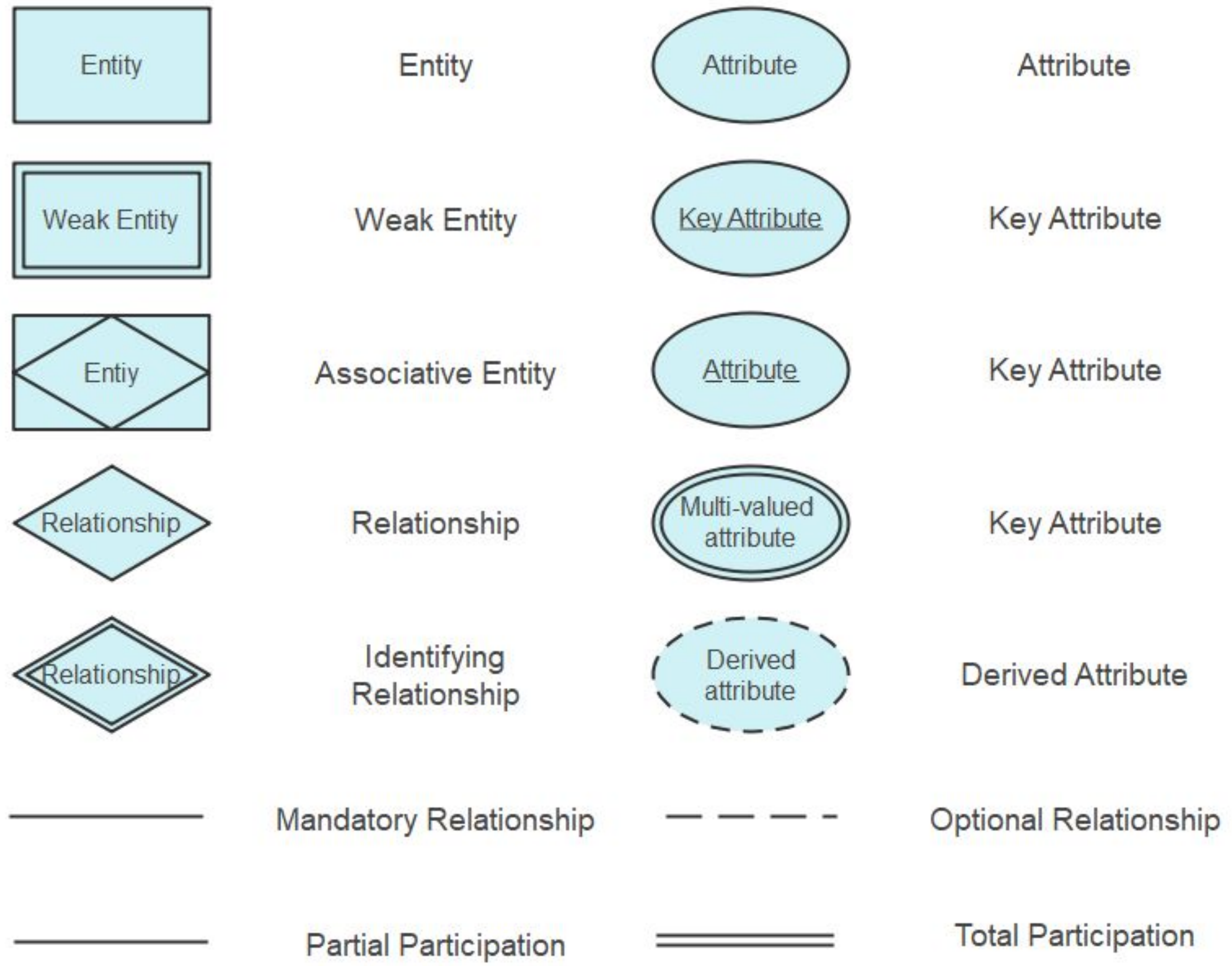
Relationships → how entities connect (Student enrolls in Course).

ER Diagram → visual blueprint showing entities, attributes, and relationships.



A sample E-R diagram

Symbol and Notation



Entity

- An entity in an ER Model is a real-world object.
- An **entity** is a “thing” or “object” in the real world that is distinguishable from all other objects.

  For example, each person in an enterprise is an entity.
- Each entity has **attributes**—the particular properties that describe it. Or Attributes are descriptive properties possessed by each member of an entity set

Examples of Entities

1. Banking System

- **Customer**:
  Attributes: Name, Customer_ID, Address, Phone_Number.
- **Account**:
  Attributes: Account_Number, Account_Type, Balance, Branch.
- **Transaction**:
  Attributes: Transaction_ID, Amount, Date, Account_Number.

2. E-commerce

- **Product**:
  Attributes: Product_ID, Name, Price, Category, Stock.
- **Customer**:
  Attributes: Customer_ID, Name, Email, Address, Phone_Number.
- **Order**:
  Attributes: Order_ID, Order_Date, Customer_ID, Total_Amount.

3. University Management System

- **Student**:
  Attributes: Student_ID, Name, Age, Program, Year.
- **Course**:
  Attributes: Course_Code, Course_Name, Credits, Department.
- **Instructor**:
  Attributes: Instructor_ID, Name, Department, Designation.

Attributes

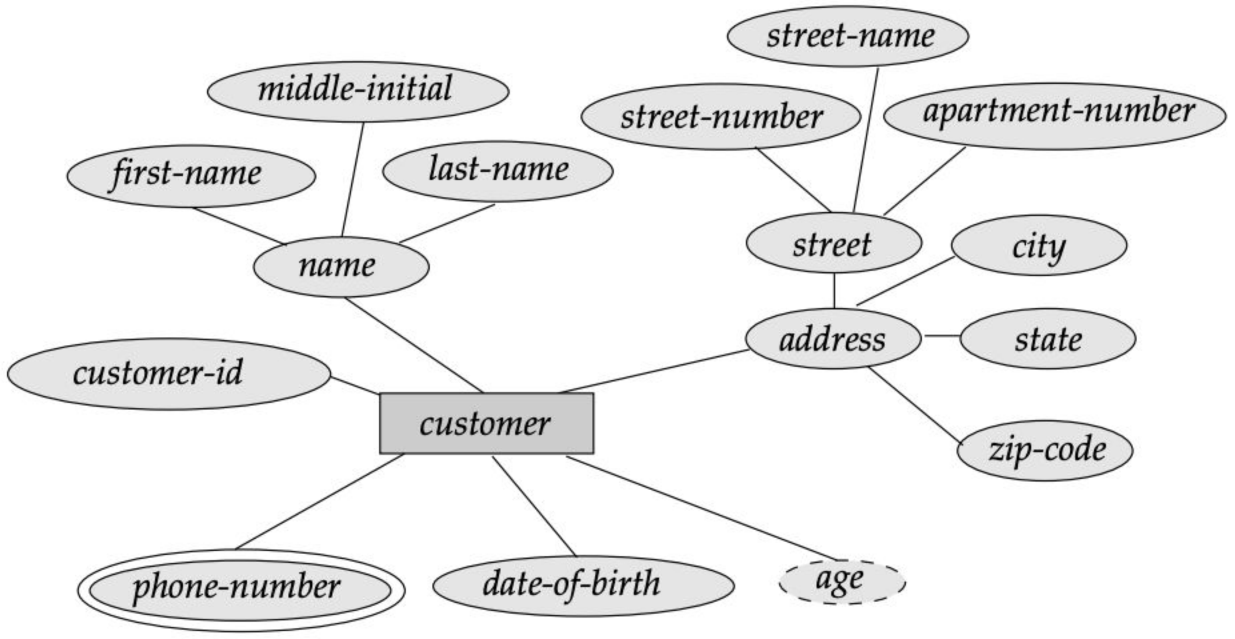
In an **Entity-Relationship (ER) model**, **attributes** are properties or characteristics that describe an entity or a relationship. Attributes provide detailed information about the entity or relationship in a database.

Types of Attributes in ER Models

1. Simple Attribute: Cannot be divided into smaller components.
Example:
Cutomer: Gender, dob.

2. Composite Attribute: Can be divided into smaller sub-parts, each with its own meaning.
Example:
Address can be divided into Street, City, State, Zip_Code.
Name can be split into First_Name, Middle_Name, Last_Name.

3. Single-Valued Attribute: Holds only one value for each entity instance.
Example:
custid



E-R diagram with composite, multivalued, and derived attributes.

Attrubtes (Cont…)

4. Multi-Valued Attribute: Can hold multiple values for a single entity instance.

Example: a person can have more than one phone number

Person: Phone_Numbers, Email_Addresses.

Car: Colors (e.g., a car can have multiple colors).

5. **Derived Attribute:** Values are derived from other attributes.

Example:

Age can be derived from Date_of_Birth.

Total_Salary can be derived from Basic_Salary and Allowances.

6. **Key Attribute**: A unique attribute that identifies an entity instance uniquely.

Example:

Employee_ID for an Employee entity.

ISBN for a Book entity.

7. **Stored Attribute:** An attribute whose value is stored in the database, not derived from other attributes.

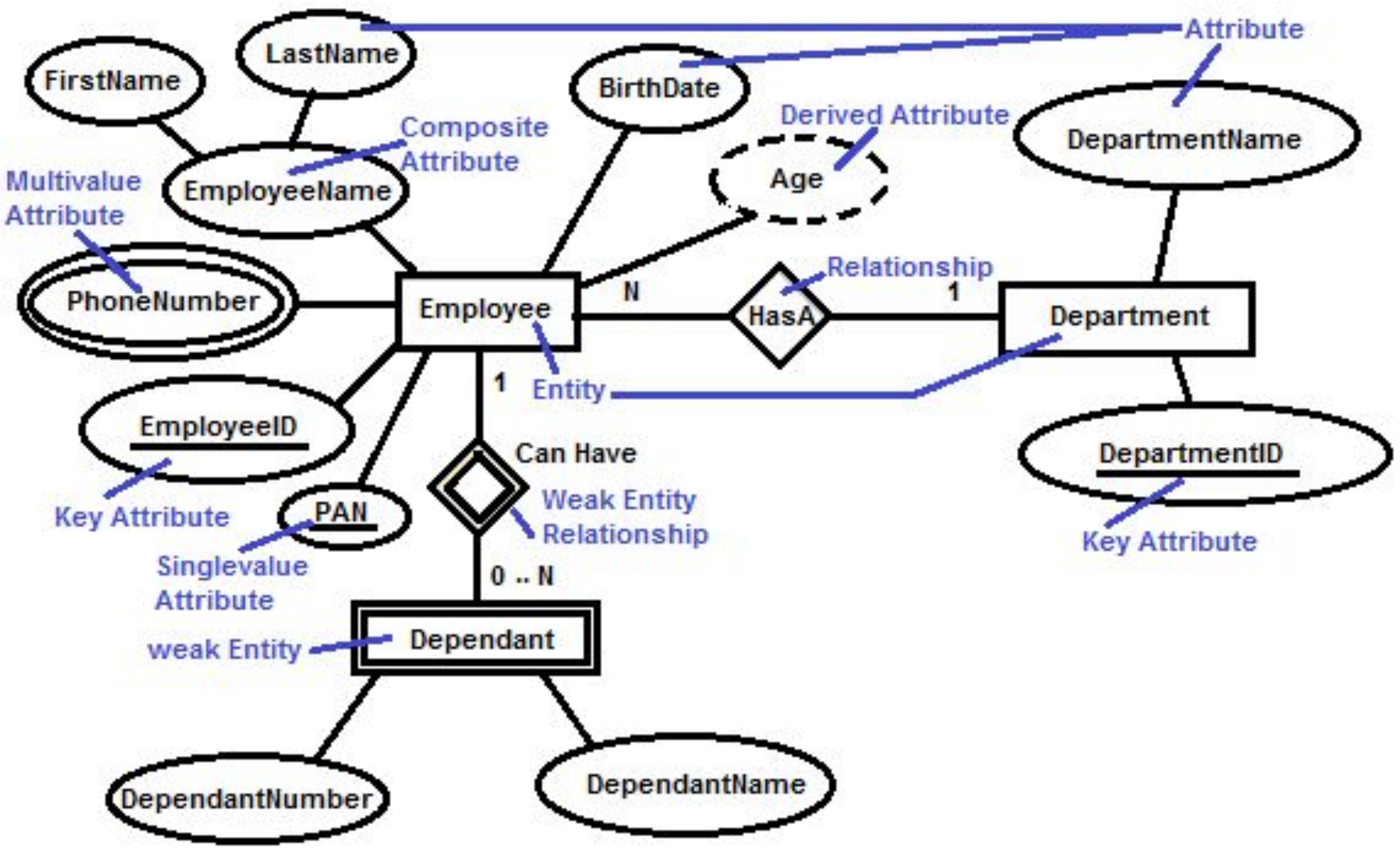
Example:

Date_of_Birth, Salary.

8. **Complex Attribute**: A combination of multi-valued and composite attributes.

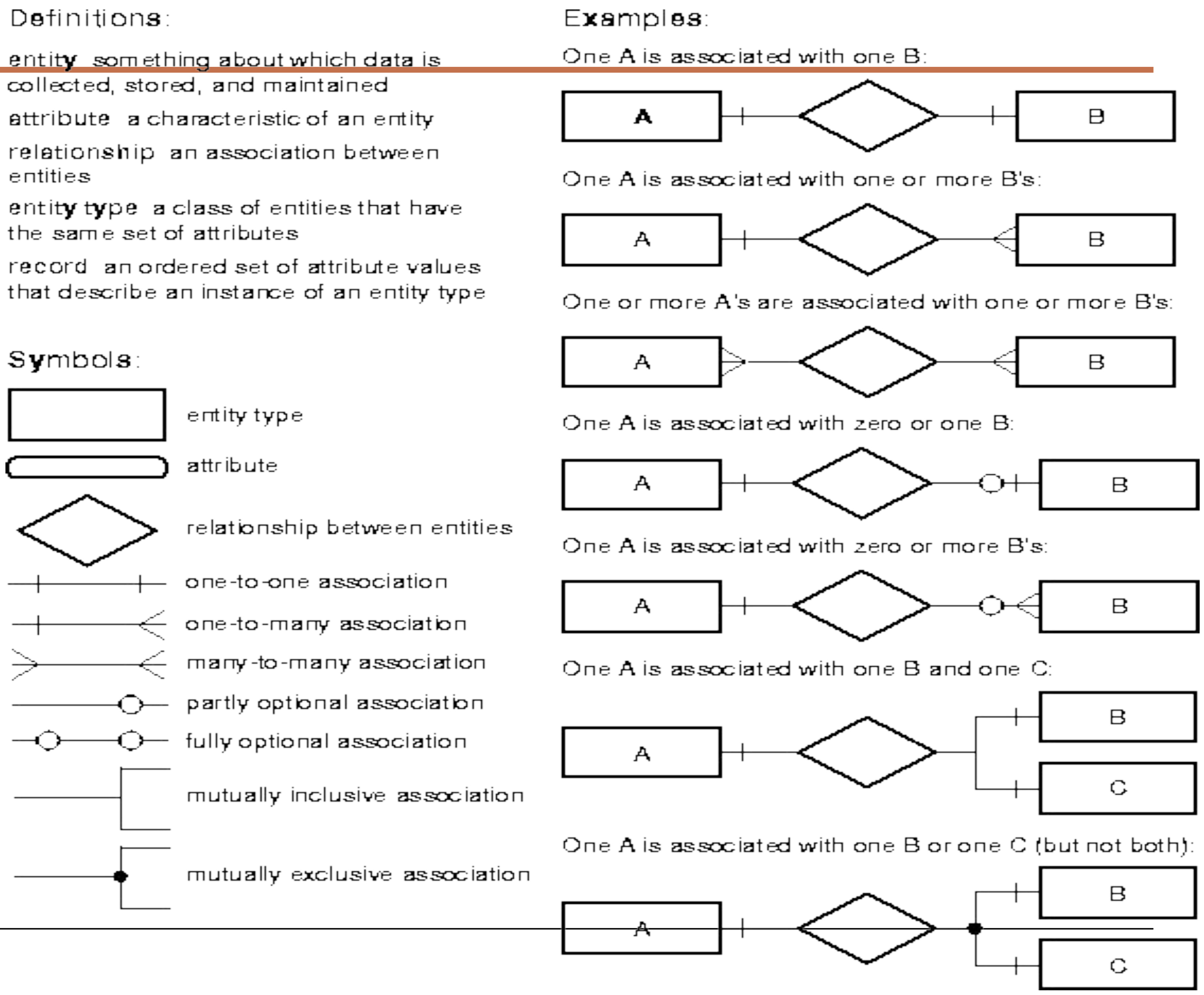
Example:

Educational_Qualifications: Degree (Bachelor's, Master's) with Year and University.



Relations

In the **Entity-Relationship (ER) model**, **relations** describe the associations or interactions between entities. These relationships are crucial in defining how entities are connected and interact with one another in a database system.



Definitions:

entity something about which data is collected, stored, and maintained

attribute a characteristic of an entity

relationship an association between entities

entity type a class of entities that have the same set of attributes

record an ordered set of attribute values that describe an instance of an entity type

Symbols:

entity type

attribute

relationship between entities

one-to-one association

one-to-many association

many-to-many association

partly optional association

fully optional association

mutually inclusive association

mutually exclusive association

Examples:

One A is associated with one B:

One A is associated with one or more B's:

One or more A's are associated with one or more B's:

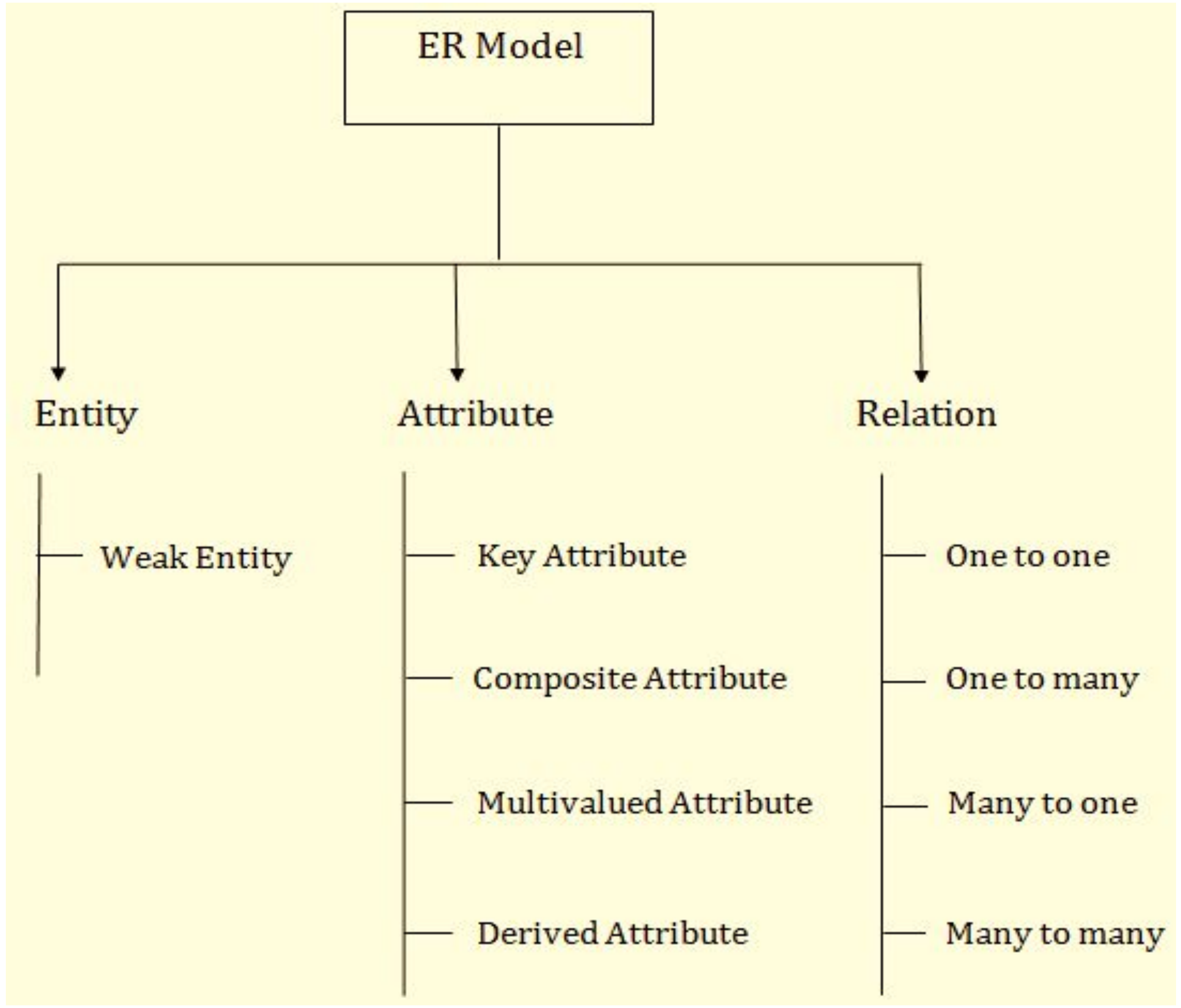
One A is associated with zero or one B:

One A is associated with zero or more B's:

One A is associated with one B and one C:

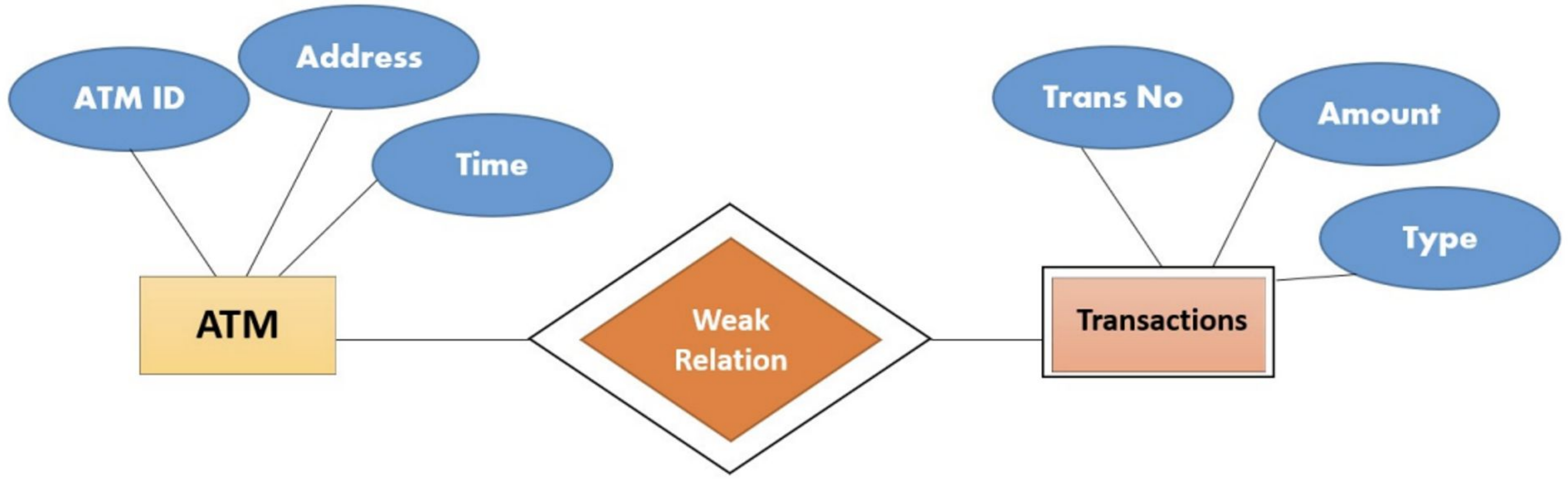
One A is associated with one B or one C (but not both):

Quick Reference



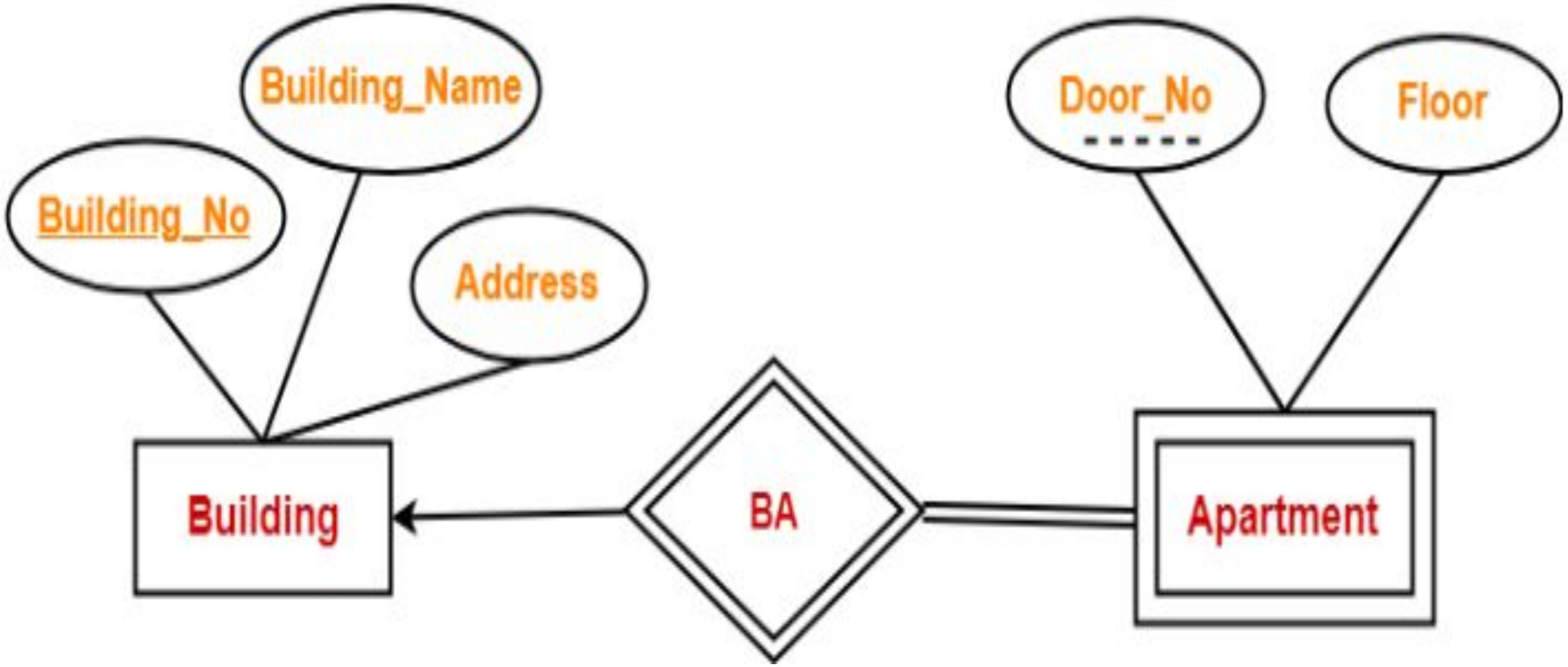
Weak Entity

- An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



In above example, "Trans No" is a discriminator within a group of transactions in an ATM.Let's learn more about a weak entity by comparing it with a Strong Entity

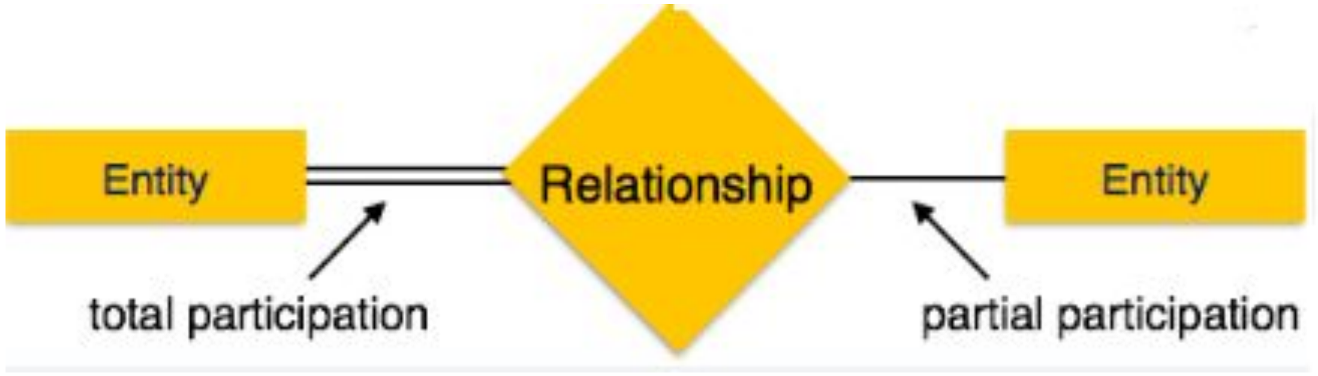
Weak entity example



Strong entity set	Weak entity set
A single rectangle is used for the representation of a strong entity set.	A double rectangle is used for the representation of a weak entity set.
It contains sufficient attributes to form its primary key.	It does not contain sufficient attributes to form its primary key.
A diamond symbol is used for the representation of the relationship that exists between the two strong entity sets.	A double diamond symbol is used for the representation of the identifying relationship that exists between the strong and weak entity set.
A single line is used for the representation of the connection between the strong entity set and the relationship.	A double line is used for the representation of the connection between the weak entity set and the relationship set.
Total participation may or may not exist in the relationship.	Total participation always exists in the identifying relationship.

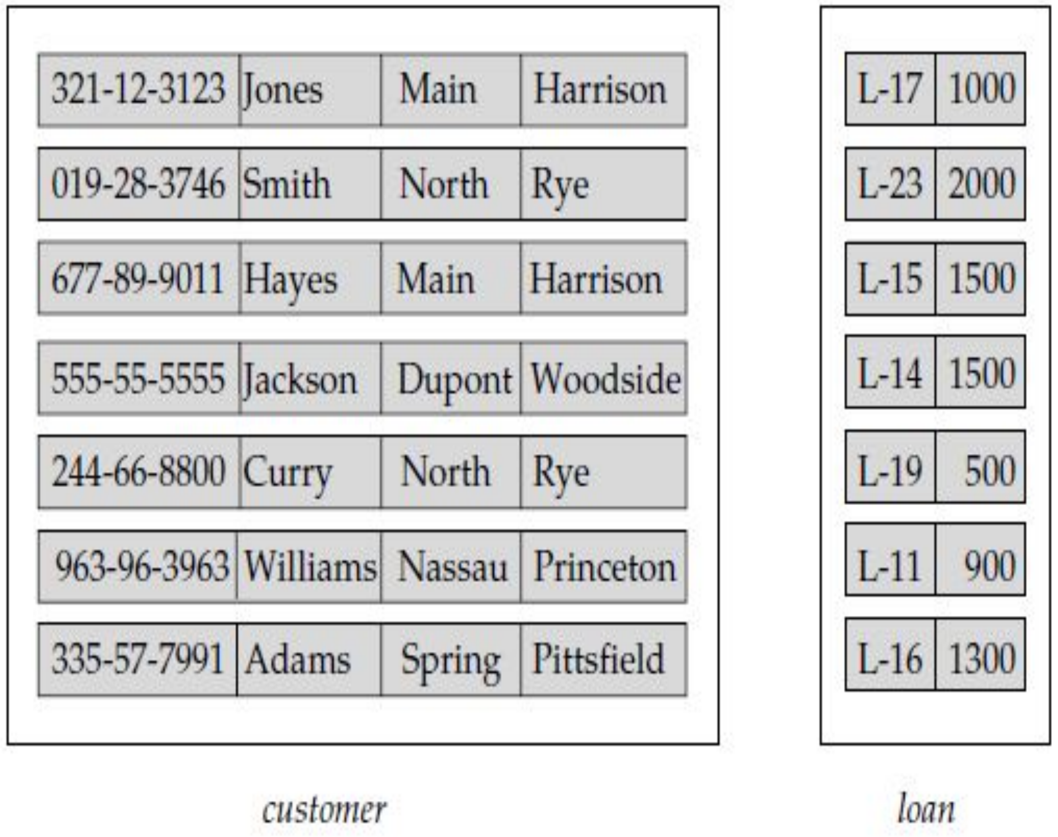
Total and Partial Participation

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



Entity set

- An entity set is a set of entities of the same type that share the same properties, or attributes. The set of all persons who are customers at a given bank, for example, can be defined as the entity set customer.
- The individual entities that constitute a set are said to be the extension of the entity set. Thus, all the individual bank customers are the extension of the entity set customer.



321-12-3123	Jones	Main	Harrison
019-28-3746	Smith	North	Rye
677-89-9011	Hayes	Main	Harrison
555-55-5555	Jackson	Dupont	Woodside
244-66-8800	Curry	North	Rye
963-96-3963	Williams	Nassau	Princeton
335-57-7991	Adams	Spring	Pittsfield

customer

L-17	1000
L-23	2000
L-15	1500
L-14	1500
L-19	500
L-11	900
L-16	1300

loan

Entity sets *customer* and *loan*.

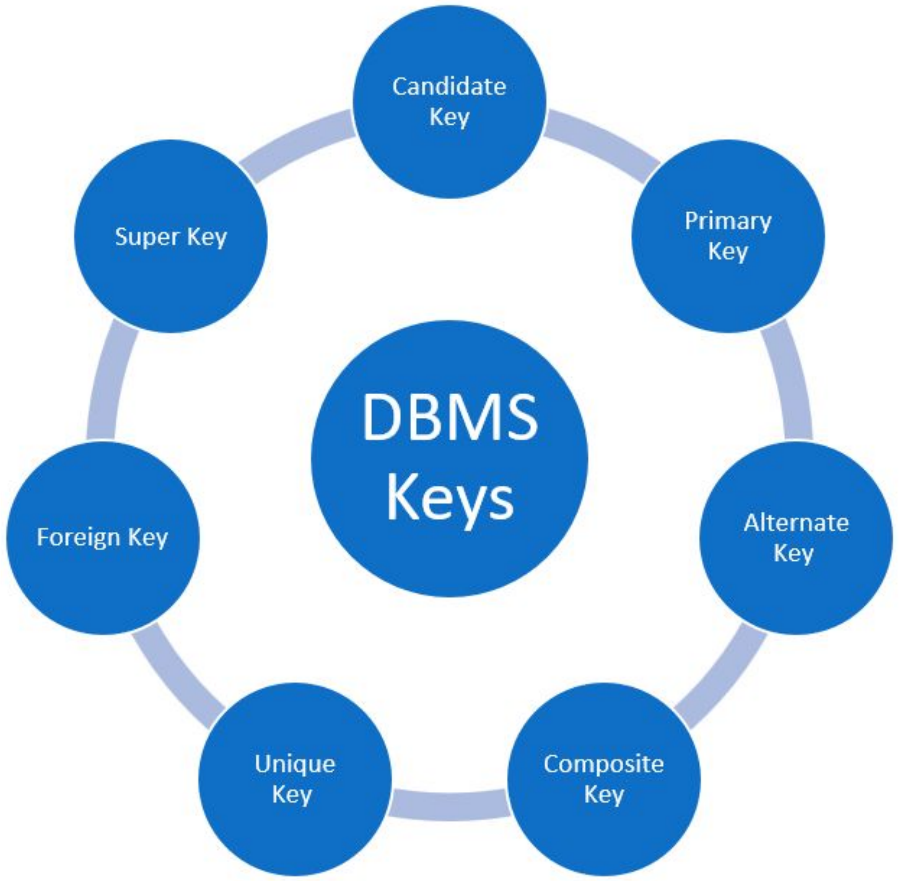
Why ER Diagram

- **Helps you to define terms related to entity relationship modeling**
- **Provide a preview of how all your tables should connect**, **what fields are going to be on each table**
- Helps to **describe entities, attributes, relationships**
- **ER diagrams are translatable into relational tables** which allows you to build databases quickly
- ER diagrams can be used by **database designers as a blueprint** for implementing data in specific software applications
- The **database designer gains a better understanding of the information to be contained in the database** with the help of ERP diagram
- ERD **Diagram allows you to communicate with the logical structure of the database to users**

Types of keys

A key in the Entity-Relationship (ER) model is an attribute or a set of attributes that uniquely identifies an entity or relationship instance in a database. It ensures that each instance can be uniquely distinguished from others.

Key Type	Unique	Allows Nulls?	Purpose
Super Key	Yes	Yes	Uniquely identifies records (may have redundancy).
Candidate Key	Yes	No	Minimal super key without redundancy.
Primary Key	Yes	No	Main unique identifier for records.
Alternate Key	Yes	No	Backup key, not chosen as primary key.
Foreign Key	No	Yes	Establishes relationships between tables.
Composite Key	Yes	No	Uniquely identifies records using multiple attributes.
Unique Key	Yes	Yes (one null)	Ensures uniqueness but permits one null.
Surrogate Key	Yes	No	System-generated unique identifier.
Secondary Key	No	Yes	Used for indexing or searching.



Candidate key

A candidate key represents a **unique combination of one or more columns within a table that can uniquely identify each row** and can potentially become the primary key. Each candidate key ensures that no two rows in the table can have the same combination of values for the candidate key columns.

Designing candidate keys is a fundamental step in database schema design, as they lay the groundwork for establishing the primary means of identifying records within a table.

For instance, in a **Students** table, possible candidate keys could be **StudentID**, **SocialSecurityNumber**, and a combination of **FirstName** and **LastName**.

StudentID	FirstName	LastName	DateOfBirth	Social Security Number (SSN)
1001	John	Doe	2000-05-15	123-45-6789
1002	Jane	Smith	2001-03-22	987-65-4321
1003	Mike	Johnson	2000-11-07	456-78-9123

In this table:

- "SocialSecurityNumber" is a candidate key as it uniquely identifies each student.
- The combination of "FirstName" and "LastName" can also be a candidate key if it ensures uniqueness throughout the information.

Primary key

The primary key is the chosen candidate key that uniquely identifies each record within a table.

It cannot contain duplicate values or NULL values. The primary key is typically indexed, allowing for faster search and retrieval operations. In relational database systems, the primary key plays a pivotal role in establishing relationships between tables through foreign keys.

For example, the below **Students** table has **StudentID** as the primary key since it uniquely identifies each student record.

In this example, "StudentID" is the primary key. It has these characteristics:
1. Unique: Each StudentID is different for every student.
2. Not null: Every student must have a StudentID.
3. Unchanging: The StudentID doesn't change over time.

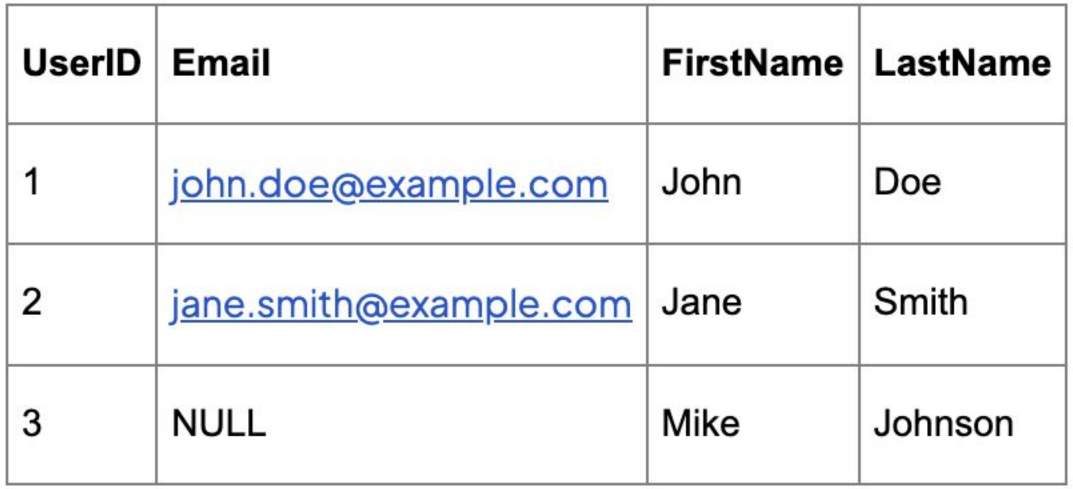
StudentID	FirstName	LastName	DateOfBirth
1001	John	Doe	2000-05-15
1002	Jane	Smith	2001-03-22
1003	Mike	Johnson	2000-11-07

Unique Key

A unique key constraint ensures that all values in a column or a combination of columns are unique within a table. Similar to a primary key, a unique key guarantees uniqueness but may allow NULL values. Unique keys provide additional data integrity by preventing duplicate values in specified columns.

While distinct from primary keys, unique keys are critical in database design for enforcing uniqueness constraints and supporting various data modeling requirements. They offer flexibility in ensuring data consistency while accommodating different business rules and constraints.

For example, in the **Users** table given below, the **Email** column is assigned as a unique key constraint to ensure each user's email address is unique. However, some users might not provide an email address (NULL value).



UserID	Email	FirstName	LastName
1	john.doe@example.com	John	Doe
2	jane.smith@example.com	Jane	Smith
3	NULL	Mike	Johnson

In this example, the "Email" column has a unique constraint, ensuring every email copes with is precise, although it allows NULL values.

Foreign Key

A foreign key is a column or set of columns within a table that establishes a relationship with the primary key or a unique key in another table. It serves to maintain referential integrity by enforcing a link between related tables.

The values in the foreign key columns must match the values in the referenced primary key or unique key column(s). This relationship allows for the creation of relational databases, where data across multiple tables can be interconnected and queried in meaningful ways. Foreign keys are instrumental in maintaining consistency and coherence in database systems.

For example, in an **Orders** table, **CustomerID** is a foreign key referencing the **CustomerID** primary key column in a **Customers** table, establishing a relationship between orders and customers.

Customers Table:

CustomerID	FirstName	LastName
1	Alice	Brown
2	Bob	White
3	Charlie	Black

Orders Table:

OrderID	CustomerID	OrderDate
101	1	2024-06-15
102	2	2024-06-16
103	1	2024-06-17

In this example, "CustomerID" inside the Orders table is a foreign key that references "CustomerID" inside the Customers table.

Super Key

The super key is a set of one or more columns whose combined values can uniquely identify each record within a table. It represents a broader concept than a primary key and can include more columns than necessary for unique identification.

Every super key guarantees uniqueness, but not all super keys are selected as primary keys. While primary keys are specifically chosen as the main means of identifying records, super keys provide a foundation for understanding the various potential identifiers within a table.

For example, in a **Students** table, a super key could be a combination of **StudentID**, **FirstName**, and **LastName**.

StudentID	FirstName	LastName	DateOfBirth
1001	John	Doe	2000-05-15
1002	Jane	Smith	2001-03-22
1003	Mike	Johnson	2000-11-07

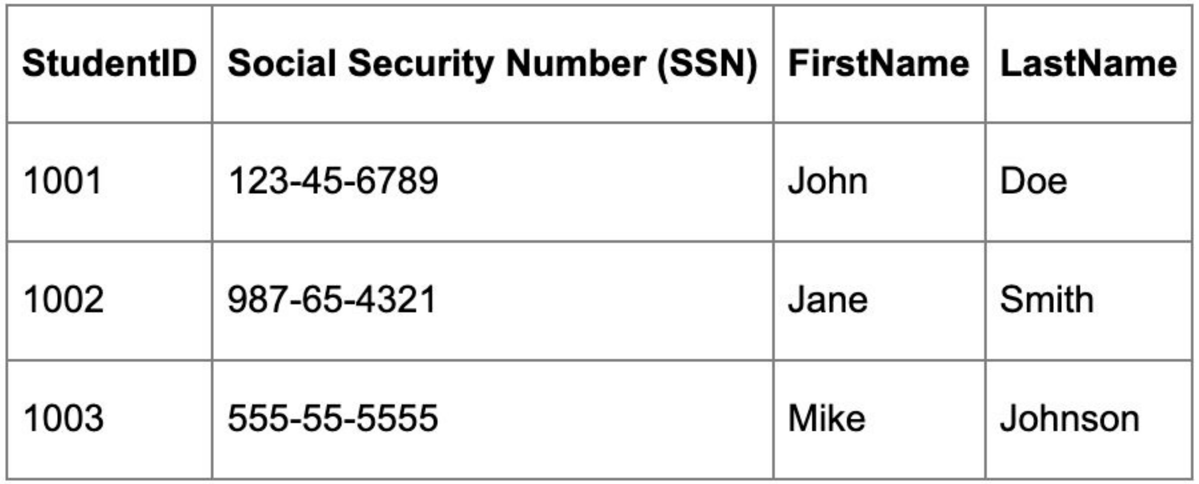
In this example, a super key may be a combination of "StudentID," "FirstName," and "LastName."

Alternate Key

An alternate key is a candidate key within a table that is not chosen as the primary key. It represents an alternative means of uniquely identifying records but is not designated as the primary means of identification. An alternate key is useful when you need to enforce unique constraints on multiple tables or when you need to reference a table from multiple other tables using different columns.

In database design, considering alternate keys alongside primary keys ensures flexibility and accommodates different querying and indexing requirements.

For instance, if **StudentID** is chosen as the primary key in a **Students** table, then **SocialSecurityNumber** could be an alternate key. Because it meets the criteria of uniqueness, identification, and potential significance within the context of the Students table.



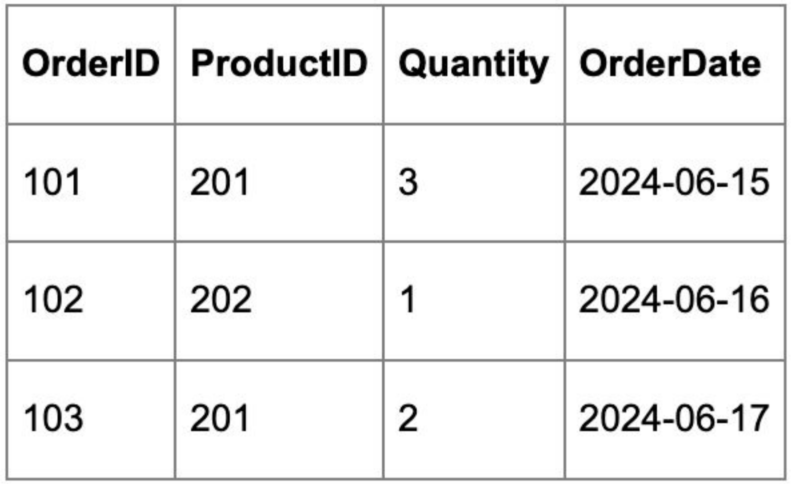
StudentID	Social Security Number (SSN)	FirstName	LastName
1001	123-45-6789	John	Doe
1002	987-65-4321	Jane	Smith
1003	555-55-5555	Mike	Johnson

In this example, "SocialSecurityNumber" is an alternate key because it uniquely identifies every student however is not the primary key.

Composite Key

The composite key is a combination of two or more columns within a table that uniquely identifies each record. While individual columns may not be unique on their own, their combination ensures uniqueness.

For instance, in an **Orders** table, a composite key might consist of **OrderID** and **ProductID** to identify each order item uniquely.



OrderID	ProductID	Quantity	OrderDate
101	201	3	2024-06-15
102	202	1	2024-06-16
103	201	2	2024-06-17

In this example, the combination of "OrderID" and "ProductID" serves as a composite key to uniquely pick out every order item.

Surrogate Key & Secondary Key

- Surrogate Key
- A surrogate key is a system-generated, artificial key (like an auto-increment number) used as a primary key when no natural key is suitable.
- Secondary Key
- A secondary key is any attribute (or set of attributes) used for searching or querying but is not unique and not the primary key.

Problem 1

For the Student table:

StudentID	Name	Email	Phone
101	Alice Johnson	alice@example.com	1234567890
102	Bob Smith	bob@example.com	9876543210

1. Identify Super Key
2. Identify Candidate key
3. Identify Primary Key
4. Identify Alternate Key

Solution 1

For the Student table:

StudentID	Name	Email	Phone
101	Alice Johnson	alice@example.com	1234567890
102	Bob Smith	bob@example.com	9876543210

Identify Super Key : A set of one or more attributes that can uniquely identify rows in a table.

{StudentID}
{Email}
{Phone}
{StudentID, Email} {StudentID, Phone}
{Email, Phone}

Identify Candidate key: A minimal super key—a super key with no redundant attributes.

{StudentID}
{Email}
{Phone}

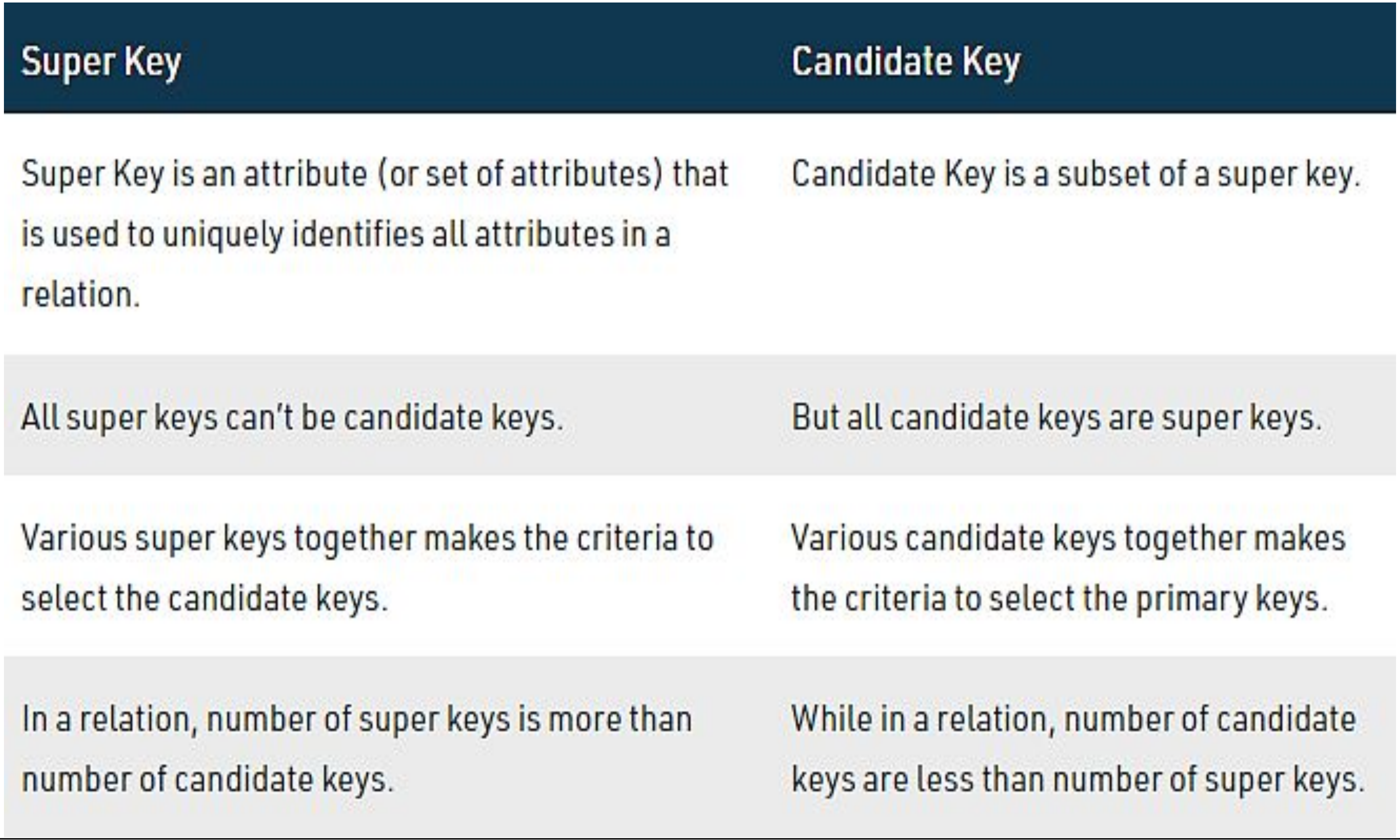
Identify Primary Key: A chosen candidate key to uniquely identify rows in the table.(Must not have NULL or Duplicate value.

{StudentID}

Identify Alternate Key Candidate keys not chosen as the primary key.

{Email}
{Phone}

Candidate key Vs Super key



Super Key	Candidate Key
Super Key is an attribute (or set of attributes) that is used to uniquely identifies all attributes in a relation.	Candidate Key is a subset of a super key.
All super keys can't be candidate keys.	But all candidate keys are super keys.
Various super keys together makes the criteria to select the candidate keys.	Various candidate keys together makes the criteria to select the primary keys.
In a relation, number of super keys is more than number of candidate keys.	While in a relation, number of candidate keys are less than number of super keys.

Problem 2

StudentID	CourseID	Grade
101	CSE101	A
101	CSE102	B
102	CSE101	A

Name the key which you will apply to uniquely identify the record? Then identify Attribute for the Key?

Solution 2

StudentID	CourseID	Grade
101	CSE101	A
101	CSE102	B
102	CSE101	A

Name the key which you will apply to uniquely identify the record? Then identify Attribute for the Key?

The key will be Composite key:

{StudentID, CourseID} uniquely identifies rows

Problem 3

Student Table

StudentID (PK)	Email (AK)	Phone (AK)
101	alice@example.com	1234567890
102	bob@example.com	9876543210

Registration Table

StudentID (FK)	CourseID (FK)	Grade
101	CSE101	A
101	CSE102	B
102	CSE101	A

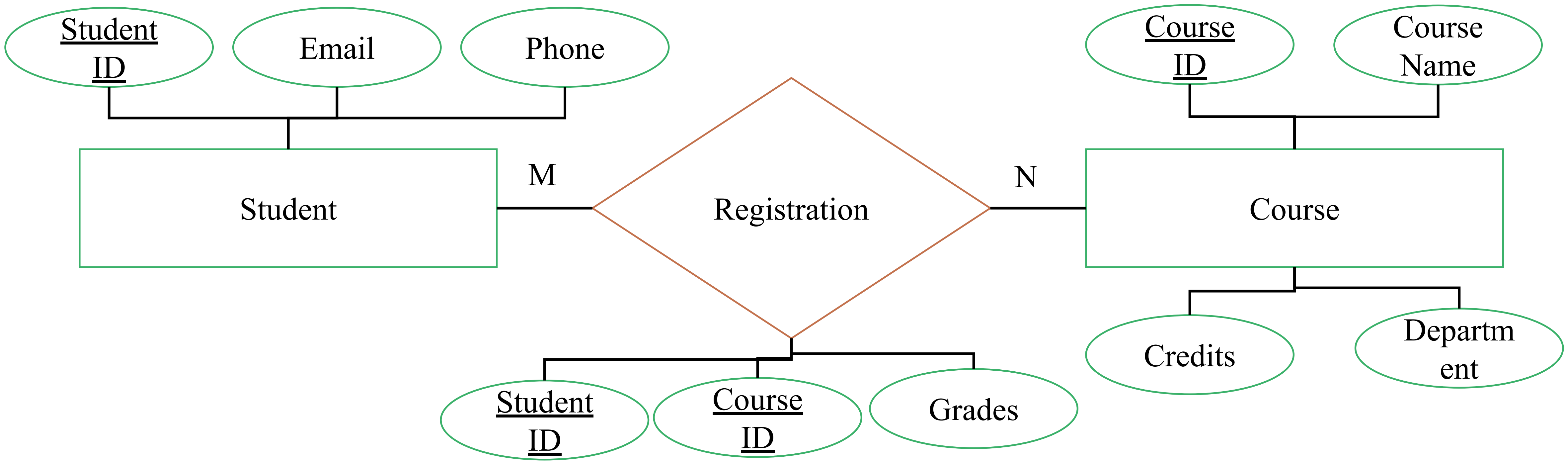
Course Table

CourseID (PK)	CourseName	Credits	Department
CSE101	Data Structures	4	Computer Science
CSE102	Algorithms	4	Computer Science
MAT101	Calculus I	3	Mathematics

Can you Identify the following and draw an ER Model

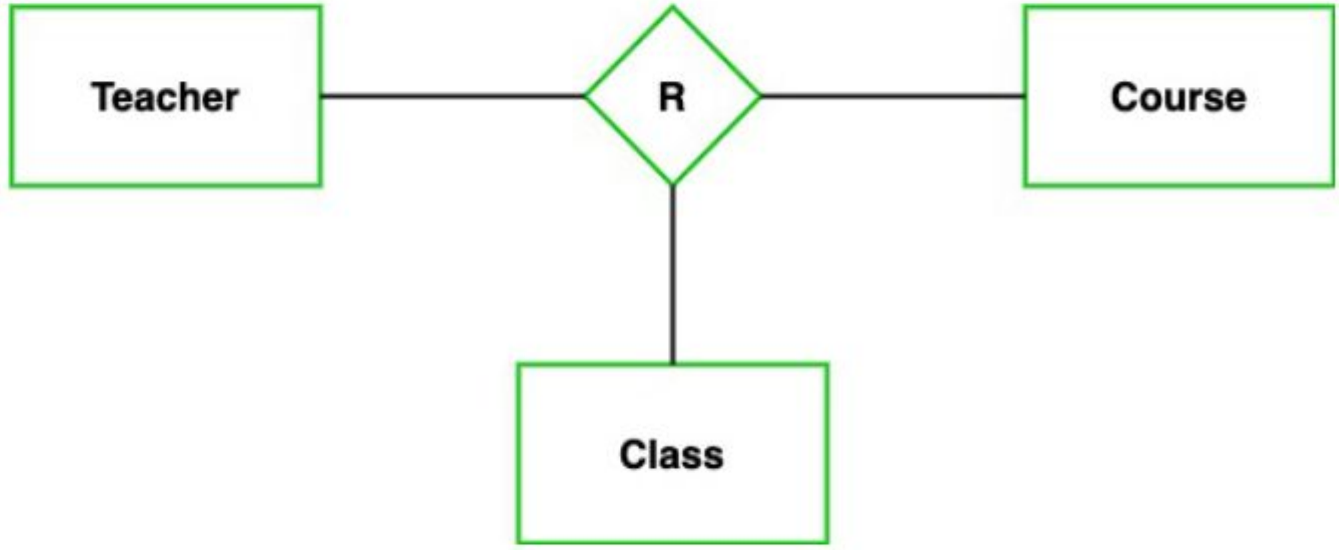
1. Name the Entity
2. Name the Relation
3. Draw the ER Diagram
4. Mapping cardinalities on ER Diagram

Solution 3


Student ID
Email
Phone
Course ID
Course Name
M
N
Student
Registration
Course
Credits
Departm ent
Student ID
Course ID
Grades

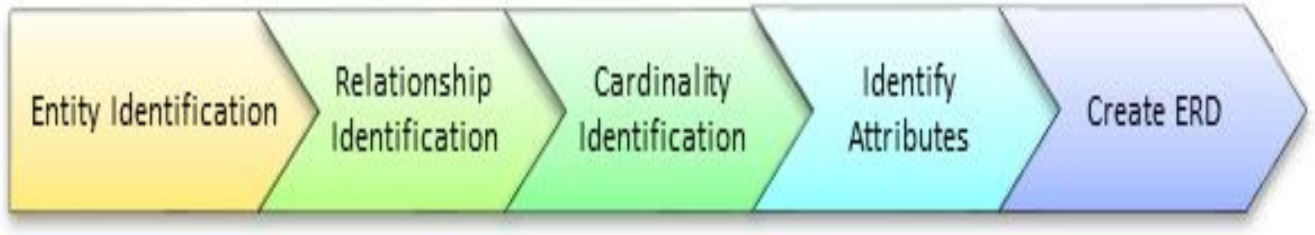
Degree of Relationships

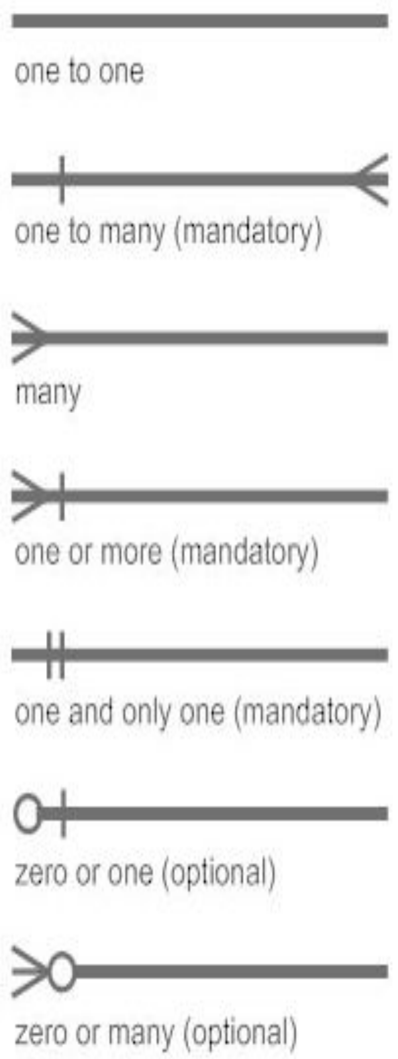
- In DBMS, a degree of relationship **represents the number of entity types that associate in a relationship**.
- For Ex. Binary relationship type, Ternary relationship type.



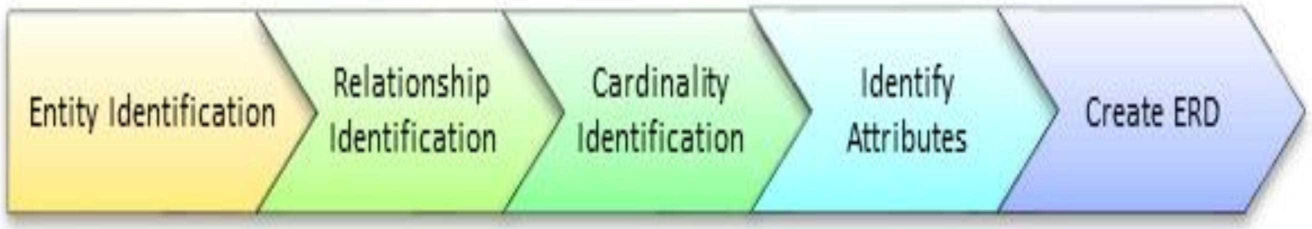
How to Create an Entity Relationship Diagram (ERD)

In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course.





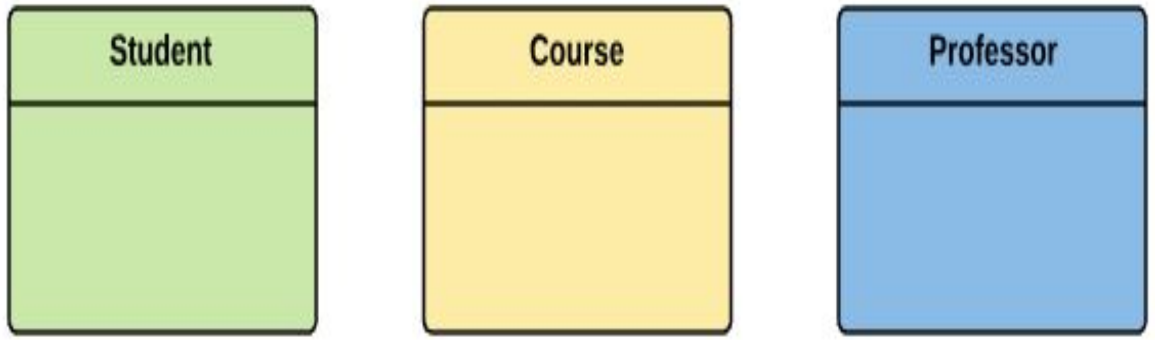
How to Create an Entity Relationship Diagram (ERD)



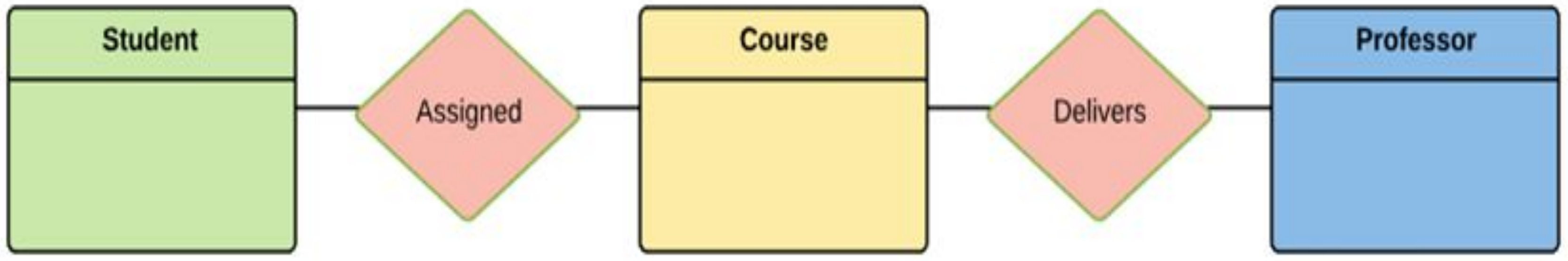
Step 1) Entity Identification

We have three entities

- Student
- Course
- Professor



How to Create an Entity Relationship Diagram



Step 2) Relationship Identification

We have the following two relationships

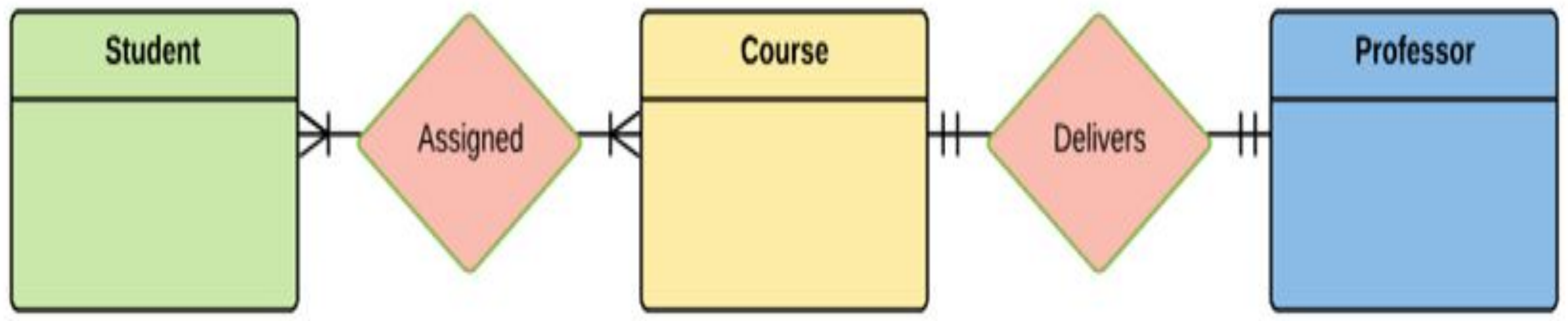
- The student is **assigned** a course
- Professor **delivers** a course

How to Create an Entity Relationship Diagram

Step 3) Cardinality Identification

For them problem statement we know that,

- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course



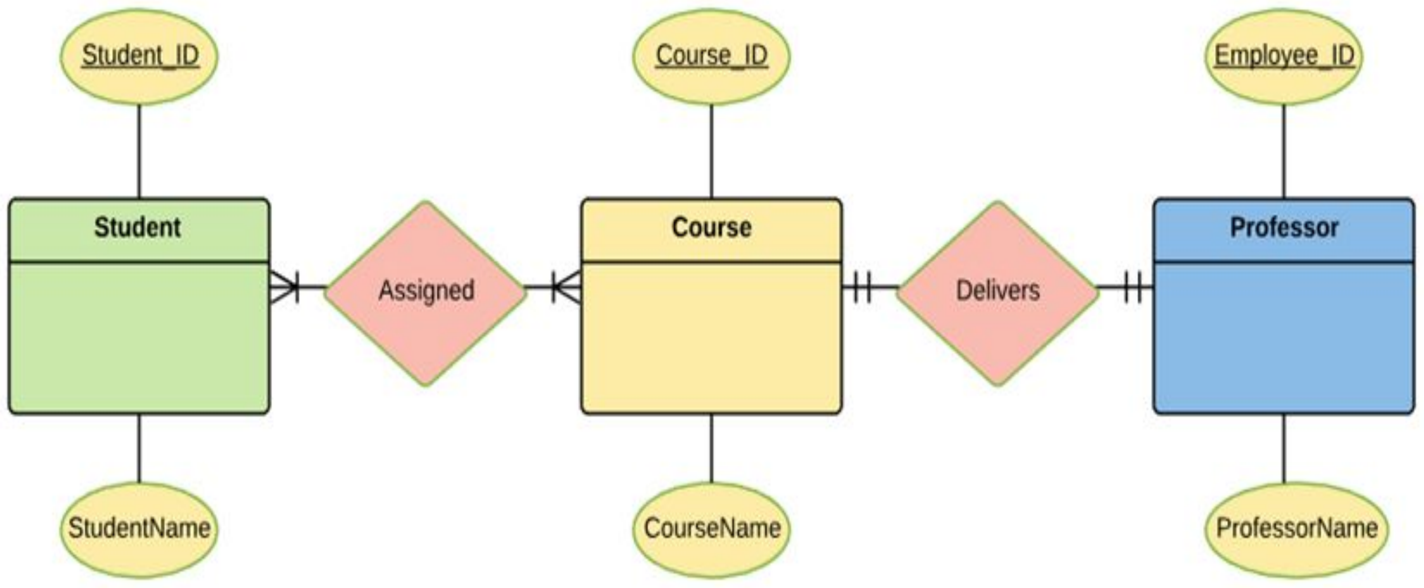
How to Create an Entity Relationship Diagram

Step 4) Identify Attributes

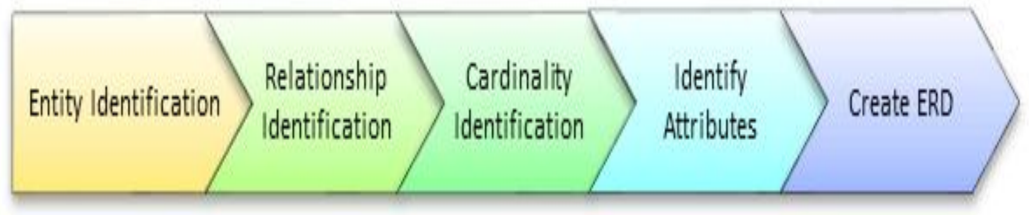
You need to study the files, forms, reports, data currently maintained by the organization to identify attributes. You can also conduct interviews with various stakeholders to identify entities. Initially, it's important to identify the attributes without mapping them to a particular entity.

Once, you have a list of Attributes, you need to map them to the identified entities. Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorNam e
Course	Course_ID	CourseName

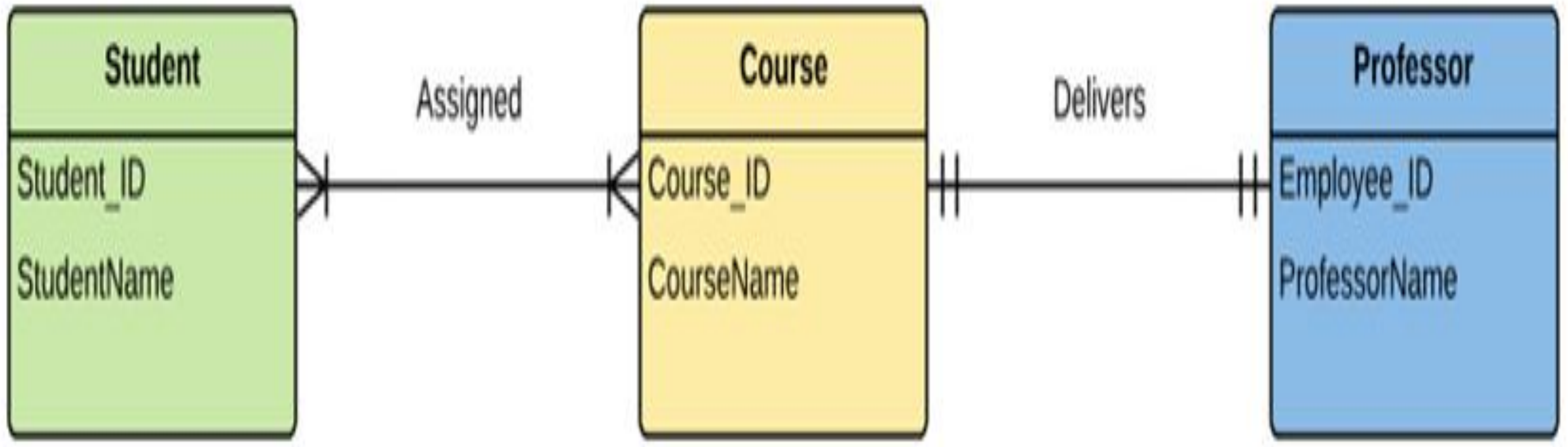


How to Create an Entity Relationship Diagram (ERD)

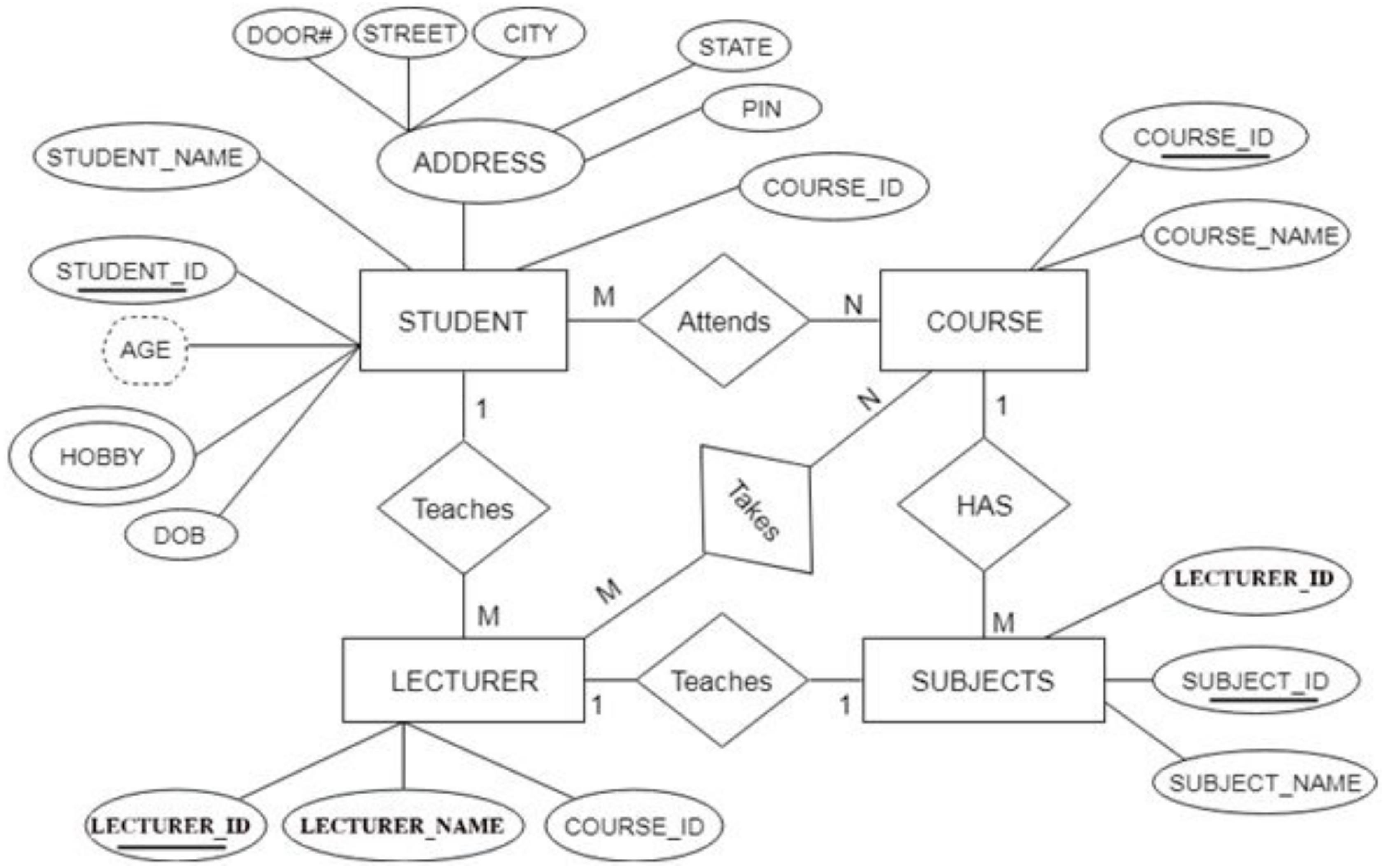


Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName

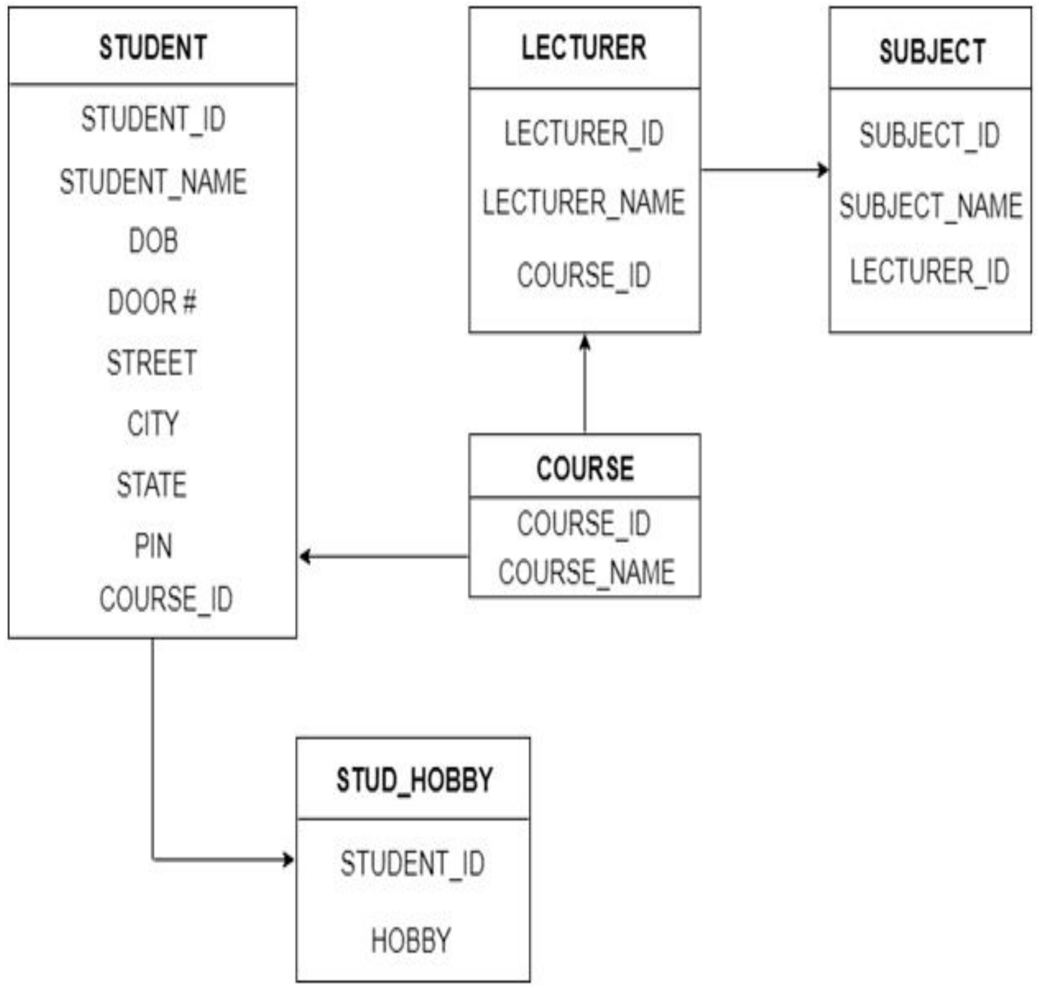
Step 5) Create the ERD Diagram



Sample ER Diagram for Student Database



ER to Table Conversion



Advantages of ER Model

- Conceptually it is very simple: ER model is very simple because if we know relationship between entities and attributes, then we can easily draw an ER diagram.
- Better visual representation: ER model is a diagrammatic representation of any logical structure of database. By seeing ER diagram, we can easily understand relationship among entities and relationship.
- Effective communication tool: It is an effective communication tool for database designer.
- Highly integrated with relational model: ER model can be easily converted into relational model by simply converting ER model into tables.
- Easy conversion to any data model: ER model can be easily converted into another data model like hierarchical data model, network data model and so on.

Extended ER Model

- ER Diagram can be used to represent only basic features of data model.
- Enhanced entity-relationship diagrams are advanced database diagrams very similar to regular ER diagrams which represent requirements and complexities of complex databases.

In addition to ER model concepts EE-R includes –

- **Subclasses and Super classes.**
- **Specialization and Generalization.**
- **Category or union type.**
- **Aggregation**

Super classes and Subclasses- Inheritance relationship

- Super class and sub class relationship leads to concept of inheritance.
- The relationship between sub class and super class is denoted with d symbol.

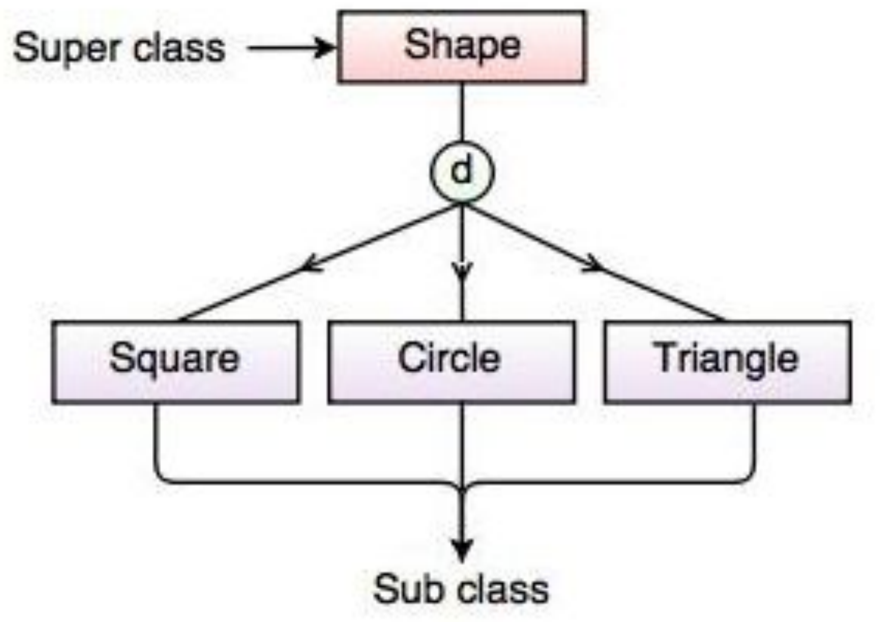
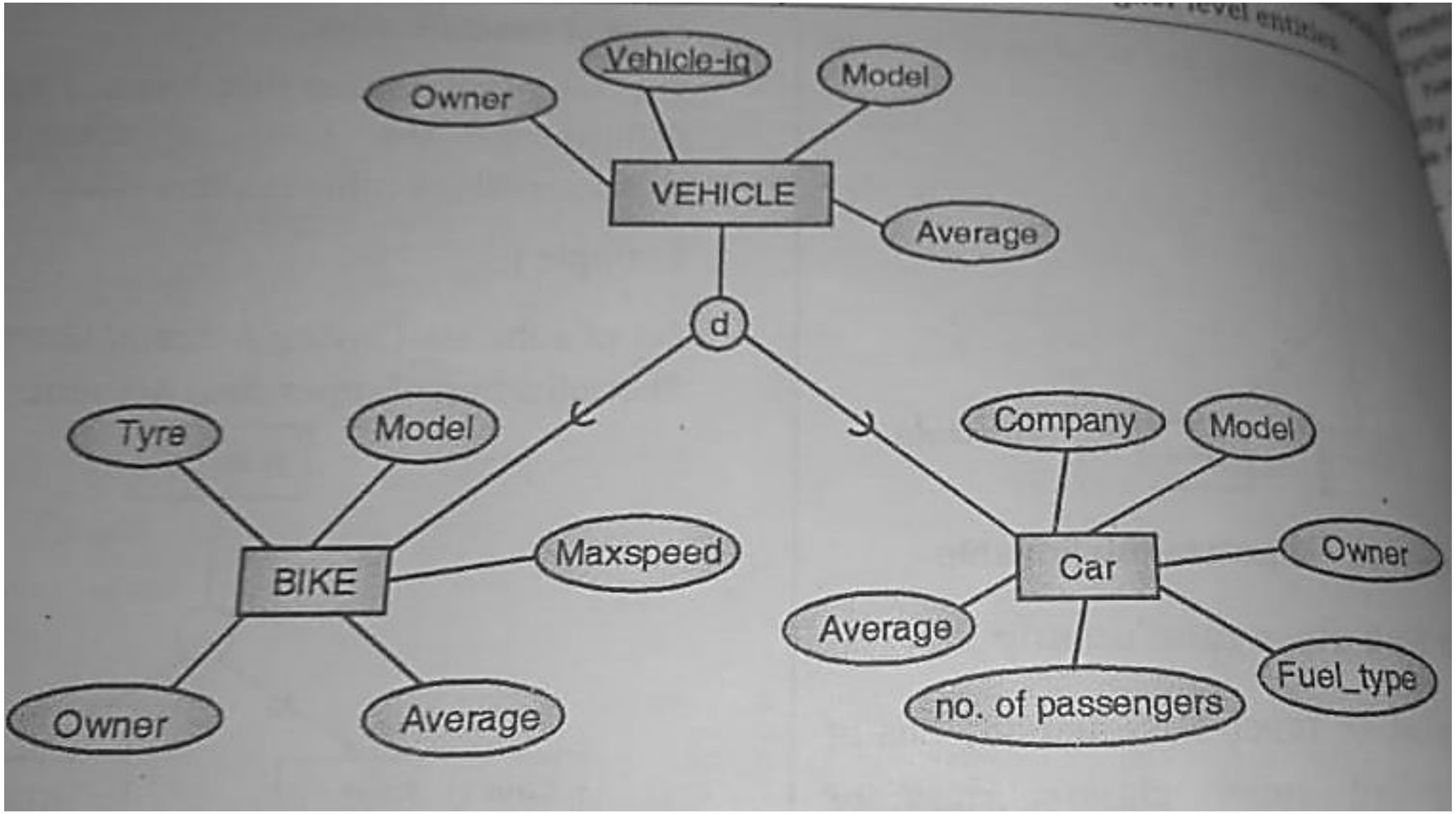


Fig. Super class/Sub class Relationship

Example


Owner
Vehicle-id
Model
VEHICLE
Average
d
Tyre
Model
BIKE
Maxspeed
Owner
Average
Company
Model
Car
Owner
Average
no. of passengers
Fuel_type

Specialization

- Specialization is a process that defines a entities which is divided into sub groups based on their characteristic.
- It is a **top down approach**, in which one higher entity can be broken down into two lower level entity.

For example

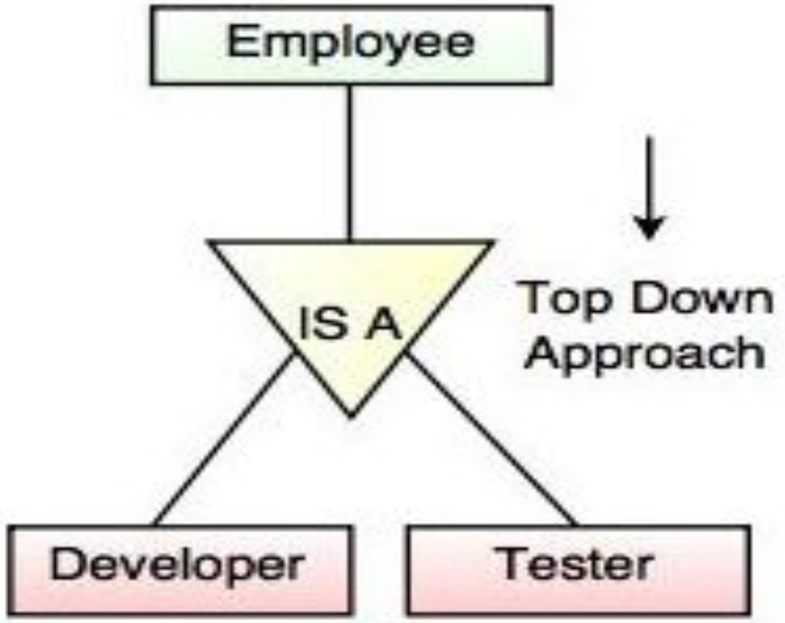


Fig. Specialization

Generalization

- Generalization is the process of generalizing the entities which contain the properties of all the generalized entities, i.e., subclasses are combined to make a superclass.
- It is a **bottom approach**, in which two lower level entities combine to form a higher level entity.

For example:

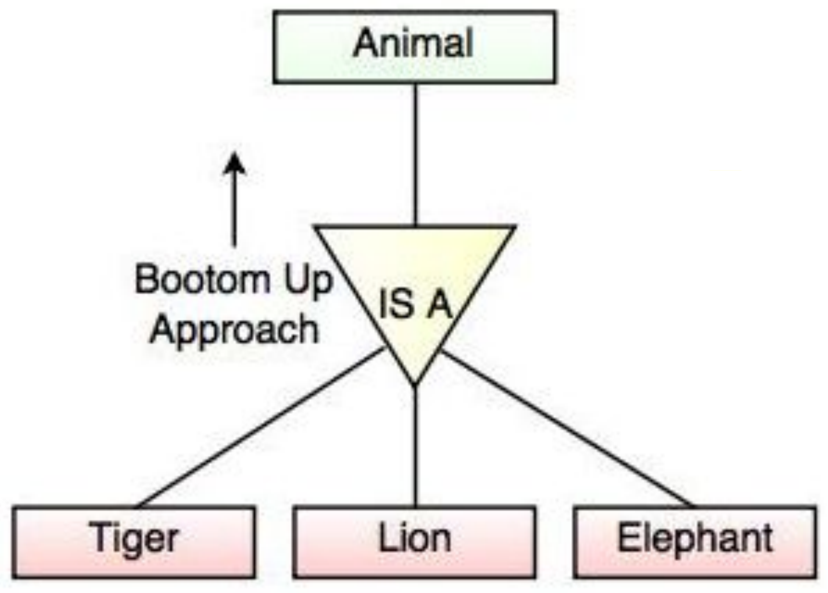


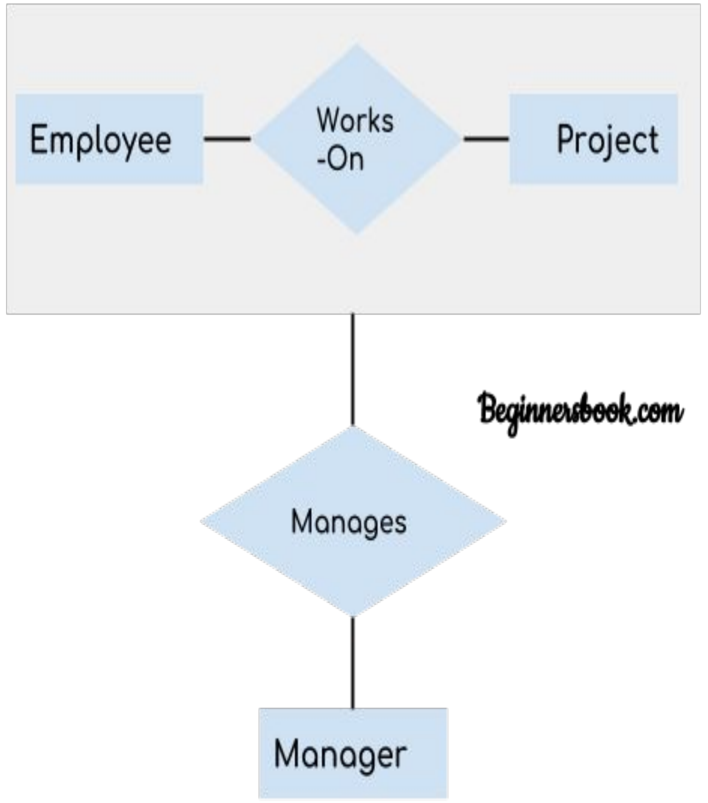
Fig. Generalization

Generalization Vs Specialization

GENERALIZATION	SPECIALIZATION
Generalization works in Bottom-Up approach.	Specialization works in top-down approach.
In Generalization, size of schema gets reduced.	In Specialization, size of schema gets increased.
Generalization is normally applied to group of entities.	We can apply Specialization to a single entity.
Generalization can be defined as a process of creating groupings from various entity sets	Specialization can be defined as process of creating subgrouping within an entity set

Aggregation

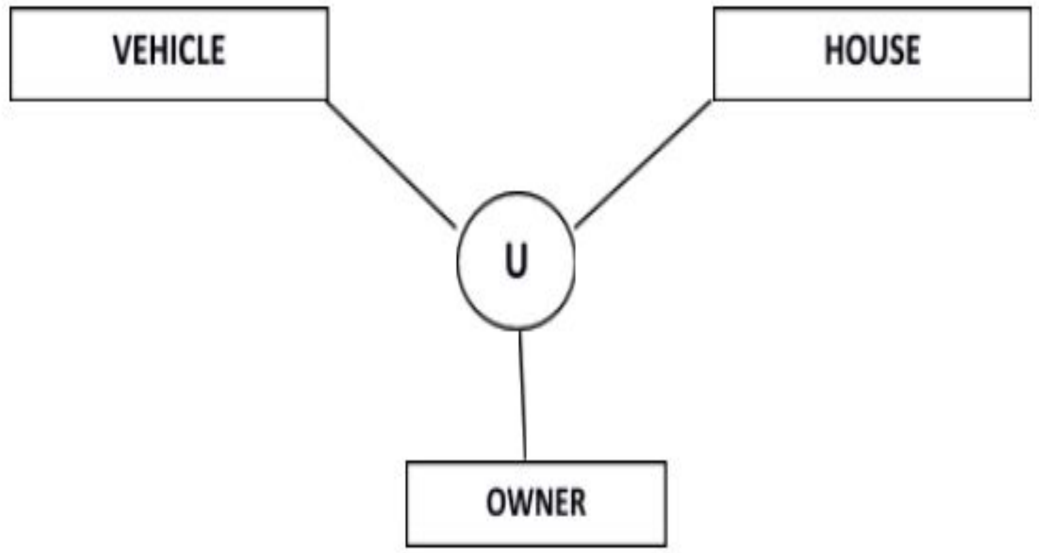
- **Aggregation** is a process in which a **single entity alone is not able to make sense in a relationship so the relationship of two entities acts as one entity**.
- Aggregation can be defined as a procedure for **combining multiple entities into a single one.** In database management, it is a design system performed to model relationships between a group of entities and another relationship. Its main motive is treating these relationships as a single one.



- In real world, we know that a manager not only manages the employee working under them but he has to manage the project as well. In such scenario if entity “Manager” makes a “manages” relationship with either “Employee” or “Project” entity alone then it will not make any sense because he has to manage both. In these cases the relationship of two entities acts as one entity. In our example, the relationship “Works-On” between “Employee” & “Project” acts as one entity that has a relationship “Manages” with the entity “Manager”.

Extended ER Model

- Category or Union
- Relationship of one super or sub class with more than one super class.
- Owner is the subset of two super class: Vehicle and House.



Hospital Management System

Scenario: A hospital needs to track patients, doctors, and the treatments provided.

Entities:
??

Relationships:
??

Hospital Management System

Scenario: A hospital needs to track patients, doctors, and the treatments provided.

Entities:

1. **Patient**: PatientID (PK), Name, Age, Gender, ContactInfo
2. **Doctor**: DoctorID (PK), Name, Specialization, Department
3. **Treatment**: TreatmentID (PK), TreatmentName, Cost

Relationships:

1. **Receives**: Between Patient and Treatment (Many-to-Many).
    1. Attributes: DateOfTreatment, Remarks
2. **Provides**: Between Doctor and Treatment (One-to-Many).

ER Diagram practice example - Online Shopping System

Scenario: An e-commerce platform wants to keep track of customers, the products they buy, and the vendors selling these products.

Entities:
??
Relationships:
??

ER Diagram practice example - Online Shopping System

Scenario: An e-commerce platform wants to keep track of customers, the products they buy, and the vendors selling these products.

Entities:

1. **Customer**: CustomerID (PK), Name, Email, Phone, Address
2. **Product**: ProductID (PK), ProductName, Price, Category
3. **Vendor**: VendorID (PK), VendorName, ContactInfo

Relationships:

1. **Orders**: Between Customer and Product (Many-to-Many).
    1. Attributes: OrderID (PK), OrderDate, Quantity
2. **Supplies**: Between Vendor and Product (One-to-Many).

Library Management System

Scenario: A library keeps track of its members, the books they borrow, and the authors of those books.

Entities:

1. ??

Relationships:??

Library Management System

Scenario: A library keeps track of its members, the books they borrow, and the authors of those books.

Entities:

1. **Member**: MemberID (PK), Name, MembershipDate, Phone
2. **Book**: BookID (PK), Title, Genre, PublicationYear
3. **Author**: AuthorID (PK), AuthorName, Nationality

Relationships:

1. **Borrows**: Between Member and Book (Many-to-Many).
    1. Attributes: BorrowDate, ReturnDate
2. **Writes**: Between Author and Book (One-to-Many).

University Management System

Scenario: A university manages students, professors, courses, and departments.

Entities:
??

Relationships:
1. **??**

University Management System

Scenario: A university manages students, professors, courses, and departments.

Entities:

1. **Student**: StudentID (PK), Name, Email, Major
2. **Professor**: ProfessorID (PK), Name, Department
3. **Course**: CourseID (PK), CourseName, Credits
4. **Department**: DepartmentID (PK), DepartmentName

Relationships:

1. **Enrolls**: Between Student and Course (Many-to-Many).
    1. Attributes: EnrollmentDate, Grade
2. **Teaches**: Between Professor and Course (One-to-Many).
3. **Belongs To**: Between Professor and Department (Many-to-One).

Hotel Booking System

Scenario: A hotel wants to manage its customers, rooms, and bookings.

Entities:

1. **Customer**: CustomerID (PK), Name, Phone, Email
2. **Room**: RoomID (PK), RoomType, PricePerNight, Status
3. **Booking**: BookingID (PK), BookingDate, CheckInDate, CheckOutDate

Relationships:

1. **Books**: Between Customer and Room (Many-to-Many).
    1. Attributes: BookingDate, Duration, AmountPaid

Airline Reservation System

Scenario: An airline wants to track passengers, flights, and bookings.

Entities:

1. Passenger: PassengerID (PK), Name, PassportNumber, Phone
2. Flight: FlightID (PK), Source, Destination, DepartureTime, ArrivalTime
3. Ticket: TicketID (PK), SeatNumber, Price

Relationships:

1. Books: Between Passenger and Flight (Many-to-Many).
    1. Attributes: BookingDate, Class
2. Issued For: Between Flight and Ticket (One-to-Many).

Movie Streaming Platform

A movie streaming platform wants to manage its operations, tracking details about users, movies, subscriptions, and reviews. The platform must support multiple languages, genres, and ratings while ensuring detailed tracking of user interactions.

Complexity and Notes:

- A **User** can watch multiple **Movies**, but a **Movie** can also be watched by multiple **Users**.
- A **Movie** can belong to multiple **Genres**, and a **Genre** can categorize multiple **Movies**.
- A **User** can write multiple **Reviews**, and each **Review** belongs to a specific **Movie**.
- A **Movie** can have multiple **Actors** in the cast, and an **Actor** can act in multiple **Movies**.
- Each **Movie** is directed by only one **Director**, but a **Director** can direct multiple **Movies**.

Possible Solution for Movie Streaming

Relationships

- Watches:
  Relationship between User and Movie (Many-to-Many).
  Attributes: WatchDate, DurationWatched
- HasSubscription:
  Relationship between User and Subscription (One-to-Many).
- BelongsTo:
  Relationship between Movie and Genre (Many-to-Many).
- Reviews:
  Relationship between User and Review (One-to-Many).
- HasReview:
  Relationship between Review and Movie (Many-to-One).
- Casts:
  Relationship between Actor and Movie (Many-to-Many).
- Directs:
  Relationship between Director and Movie (One-to-Many).

Relational Model

- Relational Model was **proposed by E.F. Codd to model data in the form of relations or tables.**
- After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDBMS languages like Oracle SQL, MySQL etc.
- **Relational Model represents how data is stored in Relational Databases**. A relational database stores data in the form of relations (tables).

Relational Model

- **Constraints in Relational Model**

- **While designing Relational Model, we define some conditions which must hold for data present in database are called Constraints. These constraints are checked before performing any operation (insertion, deletion and updation) in database. If there is a violation in any of constrains, operation will fail.**
- **Domain Constraints:** **These are attribute level constraints.** An attribute can only take values which lie inside the domain range. e.g,; If a constrains AGE>0 is applied on STUDENT relation, **inserting negative value of AGE will result in failure.**
- **Key Integrity:** Every relation in the database should have at least one set of attributes which defines a tuple uniquely. Those set of attributes is called key. e.g.; **ROLL_NO in STUDENT is a key. No two students can have same roll number.** So a key has two properties:
- **It should be unique for all tuples.**
- **It can't have NULL values.**

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

Relational Model

• Constraints in Relational Model

- **Referential Integrity:** When one attribute of a relation can only take values from other attribute of any other relation, it is called referential integrity. Let us suppose we have 2 relations

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

- **BRANCH_CODE of STUDENT can only take the values which are present in BRANCH_CODE of BRANCH which is called referential integrity constraint.**
- **The relation which is referencing to other relation is called REFERENCING RELATION (STUDENT in this case)** and **the relation to which other relations refer is called REFERENCED RELATION (BRANCH in this case).**

Relational Model

- **ANOMALIES**

- An anomaly is an irregularity, or something which deviates from the expected or normal state. When designing databases, we identify three types of anomalies: Insert, Update and Delete.
- **Insertion Anomaly in Referencing Relation:**
- We can't insert a row in **REFERENCING RELATION if referencing attribute's value is not present in referenced attribute value**. **e.g.; Insertion of a student with BRANCH_CODE 'ME' in STUDENT relation will result in error because 'ME' is not present in BRANCH_CODE of BRANCH.**
- **Deletion/ Updation Anomaly in Referenced Relation:**
- **We can't delete or update a row from REFERENCED RELATION if value of REFERENCED ATTRIBUTE is used in value of REFERENCING ATTRIBUTE. e.g; if we try to delete tuple from BRANCH having BRANCH_CODE 'CS', it will result in error because 'CS' is referenced by BRANCH_CODE of STUDENT, but if we try to delete the row from BRANCH with BRANCH_CODE CV, it will be deleted as the value is not been used by referencing relation.** It can be handled by following method:
- **ON DELETE CASCADE:** It will delete the tuples from REFERENCING RELATION if value used by REFERENCING ATTRIBUTE is deleted from REFERENCED RELATION. **e.g;, if we delete a row from BRANCH with BRANCH_CODE 'CS', the rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be deleted.**
- **ON UPDATE CASCADE:** It will update the REFERENCING ATTRIBUTE in REFERENCING RELATION if attribute value used by REFERENCING ATTRIBUTE is updated in REFERENCED RELATION. **e.g;, if we update a row from BRANCH with BRANCH_CODE 'CS' to 'CSE', the rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be updated with BRANCH_CODE 'CSE'.**

STUDENT

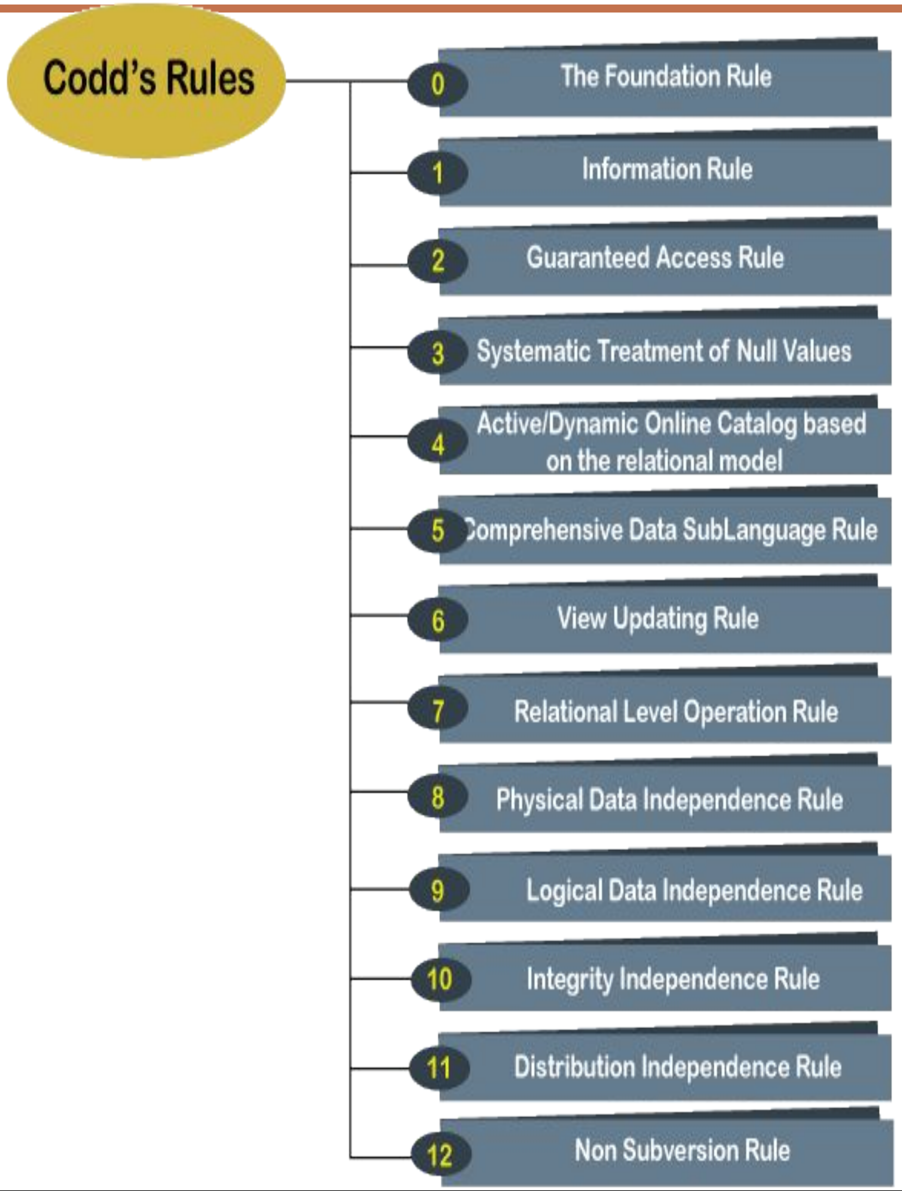
ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

Codd's Rules

- E.F Codd was a Computer Scientist who invented the **Relational model** for Database management. Based on relational model, the **Relational database** was created.
- Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with **twelve rules** of his own, which according to him, **a database must obey in order to be regarded as a true relational database.**
- These rules can be **applied on any database system that manages stored data using only its relational capabilities.**
- This is a foundation rule, which acts as a **base for all the other rules.**


Codd's Rules
0 The Foundation Rule
1 Information Rule
2 Guaranteed Access Rule
3 Systematic Treatment of Null Values
4 Active/Dynamic Online Catalog based on the relational model
5 Comprehensive Data SubLanguage Rule
6 View Updating Rule
7 Relational Level Operation Rule
8 Physical Data Independence Rule
9 Logical Data Independence Rule
10 Integrity Independence Rule
11 Distribution Independence Rule
12 Non Subversion Rule

Codd's Rules

- **Rule 0: The Foundation Rule**

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

- **Rule 1: Information Rule**

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

  e.g-Student data is in a table format

- **Rule 2: Guaranteed Access Rule**

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

  e.g-Marks of StudentID = 102

- **Rule 3: Systematic Treatment of Null Values**

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

  e.g-If a student's marks are not entered (null), the database should know how to handle it properly — not crash or misbehave.

Codd's Rules

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

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

  e.g. database details like "What tables exist?" should be stored inside the database and easy to look up.(select query)

- **Rule 5: Comprehensive Data Sub Language Rule**

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

  -The system must support at least one full language like SQL for all operations(insert, update, delete).

- **Rule 6: View Updating Rule**

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

  e.g-Views (virtual tables) should be updatable if possible.

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

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

  SQL should allow operations on multiple rows at once, not just one.

Codd's Rules

- **Rule 8: Physical Data Independence Rule**

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

  Changes in how data is stored physically should not affect how you access it

- **Rule 9: Logical Data Independence Rule**

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

  If a "Phone Number" column is added to the Student table, your old queries should still work.

- **Rule 10: Integrity Independence Rule**

  The database should be able to enforce its own integrity rather than using other programs. Key and Check constraints, trigger etc, should be stored in Data Dictionary. **All entered values should not be changed or rely on any external factor or application to maintain integrity.** This also make **RDBMS** independent of front-end.

  Rules like “Marks cannot be more than 100” should be defined in the database, not in apps.

- **Rule 11: Distribution Independence Rule**

  The distribution independence rule represents a **database that must work properly, even if it is stored in different locations and used by different end-users.** Suppose a user accesses the database through an application; in that case, they should not be aware that another user uses particular data, and the data they always get is only located on one site. This lays the foundation of **distributed database**.

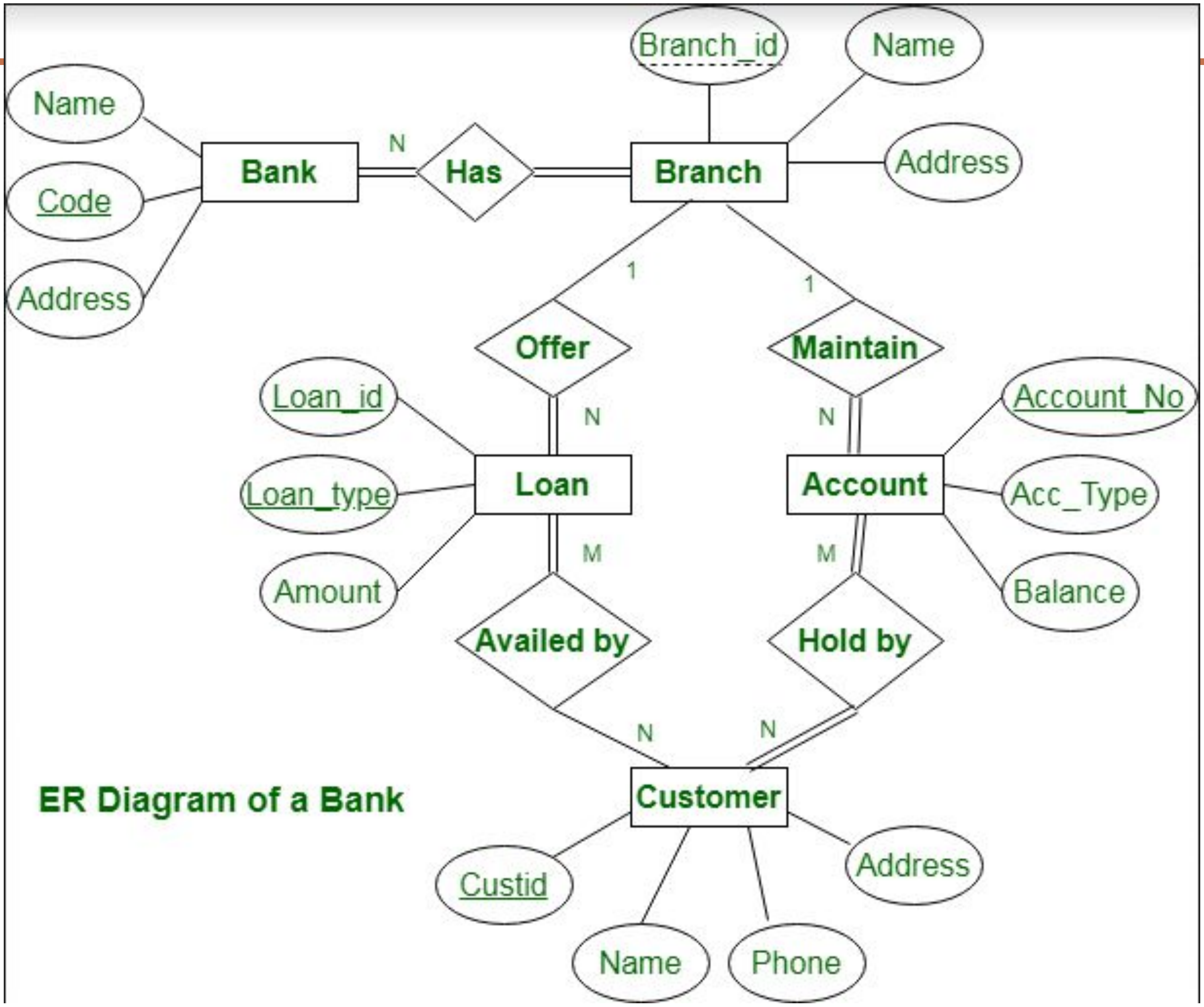
  Even if student data is stored on multiple servers, your queries should work the same.

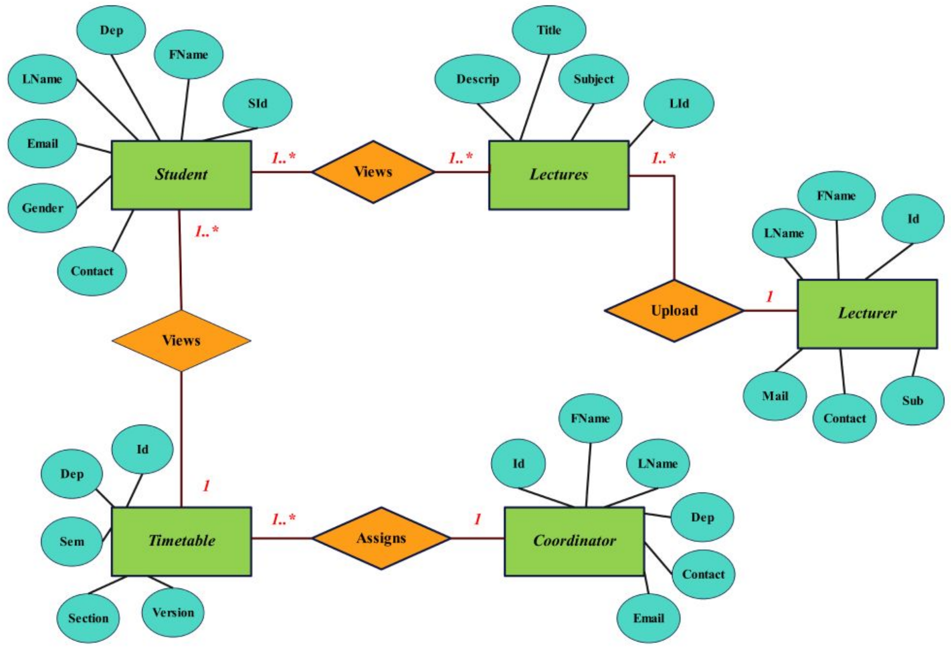
- **Rule 12: Non Subversion Rule**

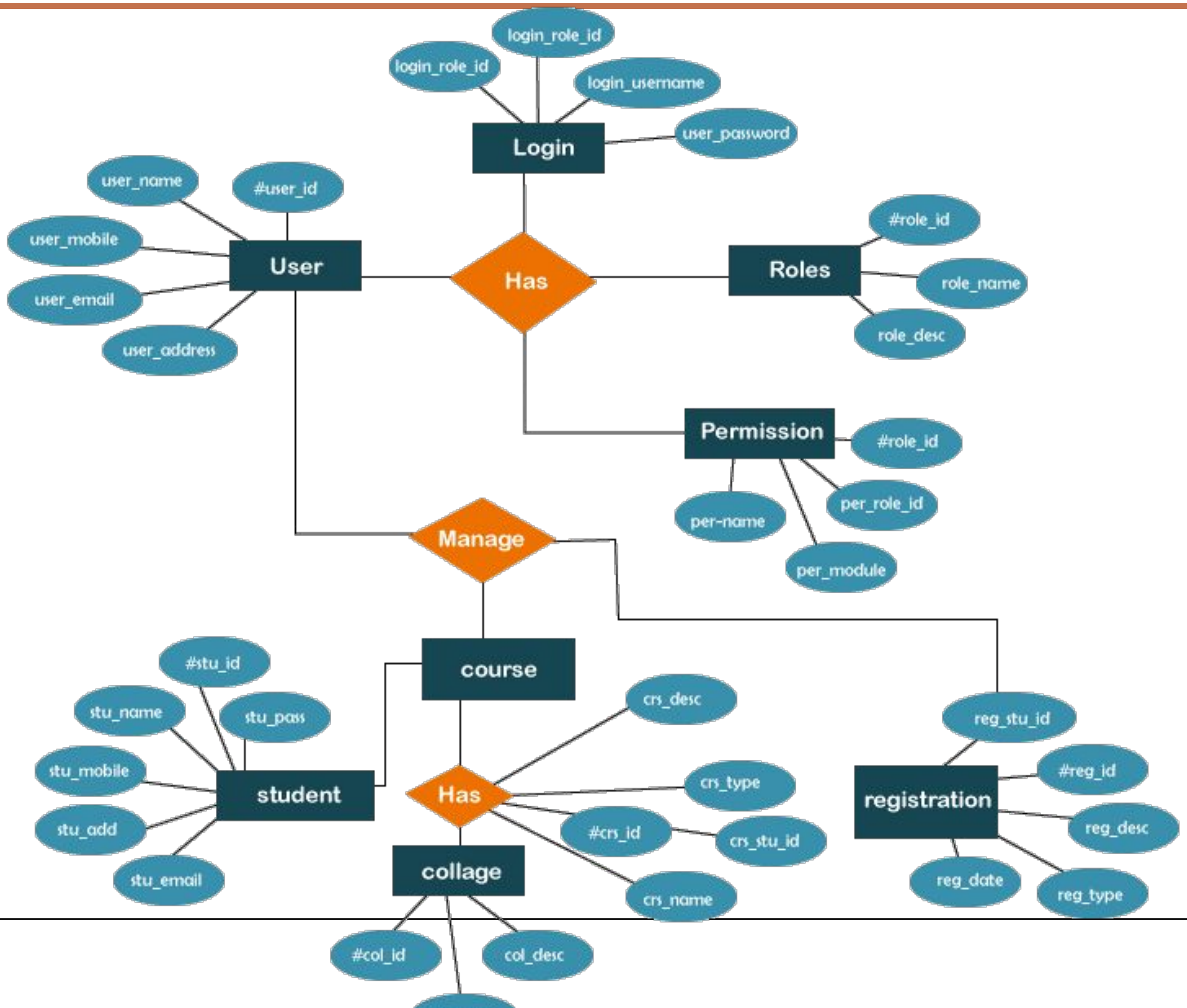
  The non-subversion rule defines RDBMS as a SQL language to store and manipulate the data in the database. **If a system has a low-level or separate language other than SQL to access the database system, it should not subvert or bypass integrity to transform data. That is** If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints

  You shouldn't be able to bypass rules even using advanced or hidden tricks.

Rule No.	Name	In Simple Words
0	Foundation Rule	Must follow all other rules
1	Information Rule	Store everything in tables
2	Guaranteed Access	Every piece of data is easily accessible
3	Nulls	Handle missing values properly
4	Catalog	Database structure also stored in tables
5	Language	SQL or similar must support all operations
6	View Update	Should allow updating through views
7	Set-level Ops	Allow bulk operations
8	Physical Data Independence	Internal storage changes shouldn't affect users
9	Logical Data Independence	Adding fields shouldn't break apps
10	Integrity Independence	Constraints defined in DB, not apps
11	Distribution Independence	Works same on one or many computers
12	Non-subversion	No bypassing the rules


Branch_id
Name
Name
Bank
N
Has
Branch
Address
Code
Address
1
1
Offer
Maintain
Loan_id
N
N
Account_No
Loan
Account
Loan_type
Acc_Type
M
M
Amount
Balance
Availed by
Hold by
N
N
ER Diagram of a Bank
Customer
Custid
Address
Name
Phone


Dep
FName
LName
SId
Email
Student
Gender
Contact
1..*
Views
1..*
Title
Descrip
Subject
LId
Lectures
1..*
FName
LName
Id
Upload
1
Lecturer
Mail
Contact
Sub
1..*
Views
Id
Dep
1
Sem
Timetable
Section
Version
1..*
Assigns
1
FName
Id
LName
Dep
Coordinator
Contact
Email


login_role_id
login_role_id
login_username
user_password
Login
user_name
#user_id
user_mobile
User
Has
Roles
#role_id
role_name
role_desc
user_email
user_address
Permission
#role_id
per_role_id
per-name
per_module
Manage
course
#stu_id
stu_name
stu_pass
stu_mobile
student
stu_add
stu_email
Has
crs_desc
crs_type
#crs_id
crs_stu_id
crs_name
collage
#col_id
col_desc
reg_stu_id
#reg_id
registration
reg_desc
reg_date
reg_type

- College Hostel Allocation System-Design an ER diagram for a college hostel system. Students apply for rooms. Each room belongs to a hostel. Each student is allocated one room, and a room can house one or more students.

Entities:

- **Student** (Student_ID, Name, Branch, Year)
- **Room** (Room_ID, Room_No, Capacity)
- **Hostel** (Hostel_ID, Hostel_Name, Warden_Name)

Relationships:

- **Applies_For**: Between Student and Room (Many-to-One)
- **Contains**: Between Hostel and Room (One-to-Many)

Thank You