SUB: Database Management System

Unit 1 : Introduction To Database Management System And Data Models

Prof: Morade D.S.

1.1 INTRODUCTION

- In business, it is very important to get the **right information at the right time** with less effort.
- A **database** is a place where data (information) is stored, changed, and managed.
- Users can work with databases either:
  - **Online** (through terminals/computers)
  - **Batch processing** (by running programs written in a programming language).

Database Management System (DBMS)

- A **Database Management System (DBMS)** is **software** that helps us to **store, organize, and manage data** in a database. Many people can use a DBMS to **search, add, or change data** at the same time.

Example in Real Life:

- Think of a **library**
  - The **books** = data
  - The **cupboards/shelves** = database (where books are kept)
  - The **librarian** = DBMS (manages the books: keeps them in order, helps you find the right book, adds new books, removes old ones).

1.2 APPLICATIONS OF DBMS

A DBMS is used in almost every field where data needs to be stored, managed, and accessed.

1. **Banking**
   - To store account details, transactions, loans, ATM details, etc.
   - Example: Checking your account balance at an ATM.
2. **Railways & Airlines ✈**
   - For ticket booking, cancellations, and schedules.
   - Example: When you book a train or flight online, DBMS manages your seat information.
3. **Education**
   - To keep student records, marks, courses, fees, etc.

- Example: Your school/college database stores your report card and fees status.

4. **Hospitals**
   - To maintain patient records, doctors' schedules, medicines, bills, etc.
   - Example: When a doctor searches for your past reports on a computer.
5. **Online Shopping / E-commerce**
   - To manage products, prices, customers, and orders.
   - Example: When you buy something on Amazon/Flipkart, the DBMS handles stock, payments, and delivery details.
6. **Telecommunications**
   - For call records, recharge, billing, and customer details.
   - Example: Your mobile bill details are stored in a telecom database.
7. **Government / Public Sector**
   - For citizen records, tax collection, voter ID, Aadhaar, etc.

1.3 **Advantages of DBMS**

1. **Data Organization**
   - DBMS keeps data in a **systematic and organized way**.
   - Example: Like files arranged neatly in a cupboard instead of scattered papers.
2. **Easy Data Access**
   - You can **quickly find information** using queries.
   - Example: Searching a student's marks in a school database within seconds.
3. **Data Security**
   - Provides **passwords and access control** so only authorized people can use the data.
   - Example: Only bank staff can see customer details, not everyone.
4. **Data Sharing**
   - Many users can use the **same data at the same time**.
   - Example: Railway ticket booking – many people can book tickets online simultaneously.
5. **Data Consistency**
   - Keeps data **accurate and up-to-date** for all users.
   - Example: If you withdraw money, your balance is updated everywhere instantly.
6. **Backup and Recovery**
   - Automatically takes backups and can **recover data if there is a crash**.
   - Example: If a system fails, your banking records are still safe.
7. **Reduced Data Redundancy**
   - Avoids storing the **same data multiple times**.
   - Example: Instead of saving a customer's details in many places, it is stored only once and used everywhere.
8. **Improved Decision Making**
   - Provides **reports and analysis** to help in planning and decision-making.
   - Example: Companies analyze sales data to decide which product sells best.

Disadvantages of DBMS

1. **High Cost**
   - Buying and maintaining DBMS software is expensive.
   - Example: Oracle, SQL Server licenses cost a lot for big companies.
2. **Complexity**
   - DBMS is a **complex system**; requires trained staff to handle it.
   - Example: A normal computer user cannot directly manage databases.
3. **Hardware & Software Requirements**
   - Needs **powerful computers and storage** to run smoothly.
4. **Regular Maintenance**
   - Database needs continuous updates, backups, and monitoring.
5. **Large Size**
   - DBMS software itself takes a lot of **disk space and memory**.
6. **Performance Issues**
   - If the database becomes very large and many users use it at the same time, it may become **slow**.
7. **Risk of Failure**
   - If the central database server fails, the **whole system stops working** until recovery.
8. **Security Risks**
   - If hackers break the security, **a lot of sensitive data** can be stolen.

1.4 USERS OF DBMS

Different people use a **DBMS (Database Management System)** in different ways, depending on their knowledge and job.
There are **4 main groups of users**:

1. **Database Designers**
   - They design and plan how the database will look and work.
   - Example: Deciding what tables are needed (like Student, Teacher, Courses in a school database).
2. **Application Programmers**
   - They write programs (applications/software) that allow other people to use the database easily.
   - Example: A programmer creating a railway ticket booking system.
3. **Sophisticated Users**
   - They directly use the database by writing queries (commands) in a database language like SQL.
   - Example: A data analyst running SQL queries to find sales reports.
4. **End Users**

- They are normal users who just use applications to get their work done, without knowing how the database works inside.
- Example: A student checking exam results online.

1.4.1 Database Designers

There are **two main types of database designers**:

1. **Database Manager** – Makes overall plans, rules, and structure of the database.
2. **Database Administrator (DBA)** – Takes care of the database daily, keeps it safe, updates it, and makes sure it runs smoothly.

1.4.1.1 Database Manager

- A **database manager** is like a supervisor.
- They manage and update the database but don't need deep technical knowledge like a DBA.
- The actual **Database Administrator (DBA)** does the technical work (create, delete, update).

Example: senior IT manager in a bank

- **What they do:** sets rules, priorities, and budgets for the database work.
- **Daily life:** has meetings with the DBA team, decides which new features to build first, checks reports like "Is the system fast? Is it safe?"
- **Goal:** make sure the bank's data work supports business needs (new branches, new loan product, etc.).
- **They usually don't type SQL;** they manage people and plans.

1.4.1.2 Database Administrator (DBA)

- A **DBA** is an expert (or a team) with good technical knowledge.
- They make sure the database works smoothly and securely.
- They also plan for future needs.

Main Responsibilities of a DBA:

1. Decide what data should be stored and how it is organized.
2. Create and manage the **data dictionary** (so users can search data).
3. Decide who can access what data (authorization).
4. Create **backup and recovery** plans.
5. Maintain **security** and check data integrity (accuracy).

6. Monitor the **performance** of the database.

Example: person who maintains a hospital's patient database

- **What they do:** technical care and safety of the data.
- **Daily life:**
  - creates tables like `Patients`, `Doctors`, `Appointments`.
  - gives permissions (reception can see basic info, doctors can see medical history).
  - sets **backups** (so nothing is lost if a server fails).
  - watches performance (if it's slow, they tune indexes).
  - fixes issues (e.g., disk full, query too slow).
- **Simple picture:** if the database were a hospital, the DBA is the doctor who keeps it healthy.

1.4.2 Application Programmers

- They are software developers.
- They create applications that allow others to use the database easily.
- They use programming languages like **C, COBOL, Pascal** (or modern ones like Java, Python, etc.).
- Example: A programmer developing an **ATM system** for a bank.

Example: developer of the IRCTC ticket booking system

- **What they do:** write the software that normal people use.
- **Daily life:**
  - designs screens: search trains, choose seats, pay.
  - writes code that turns button clicks into database actions:
    - "Search trains" → runs a query in the database.
    - "Book seat" → creates a booking record and marks that seat as taken.
- **Without them**, you'd have to write SQL by hand; with them, you just click buttons.

1.4.3 Sophisticated Users

- They are computer professionals but don't write programs.
- They directly use the database with **query languages** (like SQL).
- Example: A **data analyst** writing queries to generate sales reports.

Example: data analyst at an e-commerce company

- **What they do:** ask the database questions directly using **SQL** (no app building).
- **Daily life:**

    - runs queries like:
        - `SELECT month, SUM(sales) FROM Orders GROUP BY month;`
    - makes reports: “Which product sells best?” “Which city has fewer returns?”
- **They know SQL**, but they don’t create full applications.

1.4.4 Specialized Users

- They write **special database applications** for specific tasks.
- Example:
    - **CAD system** (Computer-Aided Design)
    - **Knowledge base** systems
    - **Expert systems** (AI-based).

Example: engineer building a CAD or an expert/AI system

- **What they do:** build **special** apps that use a database for a focused field.
- **Daily life:**
    - in CAD: stores designs, versions, parts.
    - in expert systems: stores rules/knowledge the AI uses.
- **It’s programming,** but for very specific, advanced tasks.

1.4.5 End Users

- Normal users who don’t know how the database works.
- They use pre-made applications to get their work done.
- Also called **naive users** or **unsophisticated users**.
- Example: A student checking exam results online, or a customer booking a ticket.

Examples: student checking results, customer using Paytm/PhonePe, passenger booking tickets

- **What they do:** just use the app.
- **Daily life:** type, click, read results.
- **They don’t see the database at all.** Everything happens behind the scenes.

Views of Data in DBMS

A **Database Management System (DBMS)** is software that helps us store, organize, and use data easily.
But the same data can look different to different people depending on their needs.
This is called **Views of Data**.

Why views are needed?

- Users don't need to know the technical details of how data is stored.
- They only see the part of data they need.
- DBMS hides the complexity and gives a simple view to the user.
  This hiding process is called **Data Abstraction**.

Levels of Data Abstraction

There are **3 levels**:

1. **Physical Level** (lowest level)
   - Shows **how data is stored** inside the computer (on hard disk or memory).
   - Example: Is the data stored in files, linked lists, or indexes?
   - Users don't see this. Only the system cares.
2. **Conceptual Level** (middle level)
   - Shows **what data is stored** and **how it is related**.
   - It describes the database as a whole.
   - Example: A table called `Employee` with attributes like `emp_id`, `name`, `salary`.
   - This is decided by the **DBA (Database Administrator)**.

```
Struct Emp {
    int emp_id;
    char name[20];
    float salary;
   };
```

 This means each record (row) in the Employee table will have 3 attributes: id, name, salary.

3. **View Level** (top level)
   - This is what **users actually see**.
   - Different users can have different views.
   - Example:
     - HR staff might only see `emp_id` and `name`.
     - Accountant might only see `emp_id` and `salary`.

- In a **multi-user system** (many people using the same database), not everyone needs to see the *whole* database.
- So, DBMS allows users to see only the part they need.
- This limited view is called a **View**.

♦ Why View Level is Useful?

1. **Security** → Sensitive data (like salary) can be hidden from some users.

2. **Convenience** → Each user sees only the part they care about.
3. **No duplication** → Database stays the same; only the "window" changes.

 Real-life example:

- In a **bank ATM**, you only see *your balance and transactions*. You cannot see other people's accounts or internal bank employee data.

♦ Data Independence

This means:
We can **change the way data is stored or organized** without affecting how users use it.

There are two types:

(i) Physical Data Independence

- Ability to change the **physical storage** of data without affecting users.
- Example:
  - Suppose employee records are stored in `.txt` files.
  - Later, the DBA changes them to `.csv` or stores them on SSD instead of HDD for speed.
  - Users (HR, Accounts) don't even notice — they still run the same queries like `SELECT name FROM employee`.

 In short: Users don't care how data is stored physically.

(ii) Logical Data Independence

- Ability to change the **logical structure** of data without affecting user views.
- Example:
  - Suppose a new column `email` is added in Employee table.
  - Old applications (that only use name and salary) will still work fine without needing changes.

 In short: Users don't care if new fields or relations are added, as long as their required fields remain.

Data Models

What is a Data Model?

- A **data model** is simply a **plan or design** of how data will be stored, organized, and connected in a database.
- It tells us **what data is stored** and **how it relates** to other data.

 Think of it like a **blueprint of a house**:

- The blueprint shows where each room (data) will be.
- It also shows how rooms are connected (relationships).
- Similarly, a data model shows where and how data is arranged in a database.

Why Data Models are Needed?

- To **organize data** properly.
- To **hide unnecessary details** and only show what users need.
- To make it **easier for developers** to design and maintain the database.

Types of Data Models

1. Object-Based Logical Models

These focus on describing data using **real-world objects**.

- **Entity-Relationship (E-R) Model**
  - Represents data as entities (things) and relationships between them.
  - Example: *Students* and *Courses* are entities. “Enrolls in” is a relationship.
- **Object-Oriented Model**
  - Uses objects (like in programming). Each object has **data** and **methods**.
  - Example: An *Employee* object may have data (name, salary) and methods (calculate bonus).
- **Semantic Data Model**
  - Focuses on the **meaning of data** and relationships.
  - Example: "Doctor treats Patient" shows a meaningful relationship.
- **Functional Data Model**
  - Uses **functions** to show how data is transformed.
  - Example: A function "TotalSalary(Dept)" calculates the total salary of employees in a department.

---

2. Record-Based Logical Models

These describe data in terms of **records (rows)** and **fields (columns)**.

- **Relational Model**
  - Data is stored in **tables** (rows and columns).
  - □ Example: Student table with columns (Roll_No, Name, Age).
- **Network Model**
  - Data is stored as records connected by **pointers/links** (like a graph).
  - □ Example: A student linked to many courses through a network structure.
- **Hierarchical Model**
  - Data is stored in a **tree structure** (parent-child).
  - □ Example: Company database → Department (parent) → Employees (child).

---

3. Physical Data Models

These describe **how data is actually stored** in the computer (hardware level).

- **Unifying Model** → Focuses on general organization of physical data.
- **Frame Memory Model** → Focuses on storing data in memory efficiently.

□ Example: Whether data is stored in files, blocks, or memory addresses.

Relational Model (1.6.1)

- The relational model organizes data in the form of **tables (relations)**.
- Each table has:
  - **Rows (records/tuples)** → represent individual data items.
  - **Columns (fields/attributes)** → represent properties of data.
- Every table must have a **unique key (Primary Key)** to identify each row.

□ Think of it like an **Excel sheet**:

- Each **sheet** = one table.
- Each **row** = one entry.
- Each **column** = one type of information.

Example from your book

1. Customer Table

custid	name	city
01	Millar	London
02	Roger	Mansfield
03	Brown	Blackburn

- Here, `custid` is the **primary key** (unique ID for each customer).

---

2. Account Table

account_no	Amt
1224	10,000
3424	5,000

- Here, `account_no` is the **primary key**.

---

3. Customer_Amount Table (Relationship Table)

This is used to connect **customers** with their **accounts**.

custid	account_no
01	1224
02	3424

- This table links the `custid` from **Customer Table** and `account_no` from **Account Table**.
- This shows **which customer owns which account**.

How the Relationship Works

- **Millar (custid 01)** → has account 1224 with amount 10,000.
- **Roger (custid 02)** → has account 3424 with amount 5,000.
- **Brown (custid 03)** → not listed in Customer_Amount table → means he has no account.

Why Relational Model is Useful?

1. **Easy to understand** → Data looks like simple tables.
2. **Flexible** → New data can be added easily.
3. **Relationships** → Multiple tables can be connected with keys.
4. **Security** → We can control which table or column a user can access.

---

2. Network Model (1.6.2)

- Data is stored as **records**.
- Relationships are shown using **links (pointers)**.
- It looks like a **graph** where nodes are data and links are relationships.

 Example from your book:

Customer Table

cust_id	name	city	link
01	Millar	London	→ Account 1224
02	Roger	Mansfield	→ Account 3424
03	Brown	Blackburn	(no account)

Account Table

Account No	Amt
1224	10,000
3424	5,000

 Meaning:

- `cust_id 01` (Millar) is linked to Account `1224` (10,000).
- `cust_id 02` (Roger) is linked to Account `3424` (5,000).
- `cust_id 03` (Brown) has no linked account.

✔ **In short**: Network model uses **pointers/links** to show the relationship between customers and accounts.

---

3. Hierarchical Model (1.6.3)

- Similar to network model, but data is arranged in a **tree structure**.
- One record is the **root (parent)** and others are **children**.
- Relationship is **one-to-many** (like a family tree).

 Example from your book (Fig 1.7):

```
Root Node
 ├── 01 Mina Pune
 │      └── 1224 10000
 ├── 02 Rina Satara
 │      └── 3424 5000
 └── 03 Bina Baramati
```

 Meaning:

- **Mina (cust_id 01)** is parent → linked to account `1224` with amount 10000.
- **Rina (cust_id 02)** is parent → linked to account `3424` with amount 5000.
- **Bina (cust_id 03)** is parent → no account (only customer data).

✔ **In short**: Hierarchical model is like a **tree**, where each parent (customer) can have one or more children (accounts).

KEYS

? What are Keys in DBMS?

In **DBMS (Database Management System)**:

- A **Key** is a column (attribute) or a group of columns that is used to **uniquely identify a row (tuple)** in a table.
- Keys are very important because they:
    1. **Uniquely identify records** in a table.

2. **Link two or more tables** together (relationships).

Example: STUDENT Table

```
+-----------+---------+------+
| Roll_no   | Name    | Age  |
+-----------+---------+------+
| 1         | Rahul   | 20   |
| 2         | Priya   | 19   |
| 3         | Raj     | 21   |
```

Here:

- **Roll_no** can uniquely identify each student → it is a **Key**.
- **Name** cannot be a key because more than one student may have the same name.

Types of Keys in DBMS

1. **Super Key** → Any column (or set of columns) that can uniquely identify a row.
    - Example: {Roll_no}, {Roll_no + Name}
2. **Candidate Key** → Minimal Super Key (only the required columns).
    - Example: {Roll_no}
3. **Primary Key** → One Candidate Key chosen as the main key.
    - Example: Roll_no
4. **Alternate Key** → The Candidate Keys that are **not chosen** as Primary Key.
    - Example: If both Roll_no and Email are Candidate Keys, and Roll_no is Primary → Email becomes Alternate Key.
5. **Composite Key** → A key made up of **two or more columns**.
    - Example: (Student_ID + Subject)
6. **Foreign Key** → A column in one table that refers to the **Primary Key of another table** (used to link tables).
    - Example: Student_ID in a "Marks" table refers to Student_ID in a "Student" table.

---

1. Super Key

- A **Super Key** is **any set of columns that can uniquely identify a row in a table**.
- A **super key** is a combination of all possible attributes that can uniquely identify the rows (or tuple) in the given relation.
    - Super key is a superset of a candidate key.
    - A table can have many super keys.
    - A super key may have additional attribute that are not needed for unique identity.

Emp_Id	Name	Aadhar_No	Email_Id	Dept_Id
01	Aman	775762540011	aa@gmail.com	1
02	Neha	876834788522	nn@gmail.com	2
03	Neha	996677898677	ss@gmail.com	2
04	Vimal	796454638800	vv@gmail.com	3

Super Keys:

1. {Emp_Id}
2. {Aadhar_No}
3. {Email_Id}
4. {Emp_Id, Aadhar_No}
5. {Aadhar_No, Email_Id}
6. {Emp_Id, Email_Id}
7. {Emp_Id, Aadhar_No, Email_Id}
8. {Emp_Id, Name}
9. {Emp_Id, Name, Dep_Id}
10. {Emp_Id, Name, Aadhar_No, Email_Id, Dept_Id}

 {Emp_Id} unique → Super Key

 {Aadhar_No} unique → Super Key

 {Email_Id} unique → Super Key

 {Emp_Id, Aadhar_No} → still unique, but redundant because Emp_Id alone was enough → Super Key

 {Aadhar_No, Email_Id} → unique, but redundant → Super Key

 {Emp_Id, Email_Id} → unique, but redundant → Super Key

 {Emp_Id, Aadhar_No, Email_Id} → still unique, but redundant → Super Key

 {Emp_Id, Name} → unique (because Emp_Id already unique), so Super Key

 {Emp_Id, Name, Dept_Id} → still unique, so Super Key

 {Emp_Id, Name, Aadhar_No, Email_Id, Dept_Id} → includes everything, still unique → Super Key

2. Candidate Key

- A **Candidate Key** is a **minimal Super Key**.
- That means it is the **smallest set of columns (attributes) that can uniquely identify each row** in a table.
- "Minimal" means → if you remove any column from it, it will **not remain unique**.

Candidate Keys

Emp_Id	Name	Aadhar_No	Email_Id	Dept_Id
01	Aman	775762540011	aa@gmail.com	1
02	Neha	876834788522	nn@gmail.com	2
03	Neha	996677898677	ss@gmail.com	2
04	Vimal	796454638800	vv@gmail.com	3

Candidate Keys:

- **Emp_Id** → Every employee has a unique Emp_Id (01, 02, 03, 04). So, it can uniquely identify rows.

- **Aadhar_No** → Every person's Aadhaar number is unique. So, it can also uniquely identify rows.

- **Email_Id** → Every employee has a unique email ID. So, this too can uniquely identify rows.

- **Name** → Not unique (two "Neha" exist). Cannot identify rows uniquely.

- **Dept_Id** → Not unique (Dept 2 has two employees). Cannot identify rows uniquely.

3. Primary Key

- A **Primary Key** is a **special Candidate Key** that the database designer chooses to uniquely identify rows in a table.
- Out of all **Candidate Keys**, only **one** is selected as the **Primary Key**.

 Rules of Primary Key:

1. It must be **unique** (no two rows can have the same value).
2. It must **not be NULL**.
3. It should have only the **minimum required columns**.

Example 1: CUSTOMER Table

Cust_name	SSN	BASIC
John	001-256	14000$
Jack	005-123	2000$
Smith	008-200	1000$
Jones	101-401	18000$
Mac	102-303	36000$

- **SSN** (Social Security Number) is **unique for each customer** → it can be the **Primary Key**.
- Even though we could also use **(Cust_name + SSN)** as a Candidate Key, it's not minimal.
- So the **Primary Key** chosen = **SSN**.

Example 2: ADVISOR Table

Advisor_ID	Name
A101	Rajesh
A102	Priya

- **Advisor_ID** is unique for each advisor → so it is the **Primary Key**.

Example 3: STUDENT Table (Composite Primary Key)

```
+-----------+----------+-------+
| Student_ID| Subject  | Marks |
+-----------+----------+-------+
| 101       | Math     | 85    |
| 101       | Science  | 78    |
| 102       | Math     | 92    |
```

- Here, **Student_ID alone is not enough** (because a student can take multiple subjects).
- **Subject alone is also not enough** (because many students take the same subject).
- But **(Student_ID + Subject)** together is unique → so this pair becomes the **Primary Key**.
  (This is called a **Composite Primary Key**).

Foreign Key

- A **Foreign Key** is a column (or set of columns) in one table that refers to the **Primary Key** of another table.
- It is used to **connect two tables** and maintain **referential integrity** (which means the data must be consistent between the related tables).

Example 1: Student & Marks Tables

STUDENT Table

```
+-----------+---------+------+
| Student_ID| Name    | Age  |
+-----------+---------+------+
| 101       | Rahul   | 20   |
| 102       | Priya   | 19   |
| 103       | Raj     | 21   |
```

- **Primary Key** = Student_ID

MARKS Table

```
+-----------+----------+-------+
| Student_ID| Subject  | Marks |
+-----------+----------+-------+
| 101       | Math     | 85    |
| 101       | Science  | 78    |
| 102       | Math     | 92    |
```

- Here, **Student_ID** in the MARKS table is a **Foreign Key**, because it refers to **Student_ID** in the STUDENT table.

- This ensures that we cannot insert marks for a student who does not exist in the STUDENT table.

□ Example 2: Orders & Customers

CUSTOMER Table

```
+-----------+---------+
| Cust_ID   | Name    |
+-----------+---------+
| C1        | John    |
| C2        | Priya   |
```

Primary Key = Cust_ID

ORDER Table

```
+-----------+----------+
| Order_ID  | Cust_ID  |
+-----------+----------+
| O101      | C1       |
| O102      | C2       |
```

- **Cust_ID** in ORDER is a **Foreign Key** (it connects each order to a customer).

□ Key Points about Foreign Key

1. A **Foreign Key** in one table refers to a **Primary Key** in another table.
2. It creates a **relationship** between the two tables.
3. Helps maintain **data integrity** → You cannot add an order for a customer that does not exist.
4. A table can have **multiple Foreign Keys**.

□ Alternate Key

- Out of all candidate keys, only one gets selected as **primary key**, remaining keys are known as alternate keys.
- In the Employee table:
  - **Emp_Id** is best suited for the **primary key**.
  - Rest of the attributes like **Aadhar_No, Email_Id** are considered as **alternate keys**.

□ Example 1: Student Table

```
+-----------+-----------+---------+
| Roll_no   | Email     | Name    |
+-----------+-----------+---------+
| 1         | rahul@x.com | Rahul |
| 2         | priya@x.com | Priya |
| 3         | raj@x.com   | Raj   |
```

- **Candidate Keys** = {Roll_no}, {Email}
- Suppose we choose **Roll_no** as the **Primary Key**.
- Then **Email** becomes the **Alternate Key**.

□ Example 2: Customer Table

```
+-----------+------------+---------+
| Cust_ID   | Phone_no   | Name    |
+-----------+------------+---------+
| C1        | 9991112222 | John    |
| C2        | 8883334444 | Priya   |
```

- **Candidate Keys** = {Cust_ID}, {Phone_no}
- If **Cust_ID** is chosen as the **Primary Key**,
- Then **Phone_no** becomes the **Alternate Key**.

♦ Surrogate Key

□ A **Surrogate Key** is an **artificial (system-generated) key** used to uniquely identify each row in a table.

- It has **no business meaning**.
- Usually created by the **DBMS** (like Auto-Increment numbers, UUIDs).
- It's used **only for identification**, not for carrying information.

□ Example

Customer Table

Cust_Id (Surrogate Key) Aadhar_No Name Email

Cust_Id (Surrogate Key)	Aadhar_No	Name	Email
1	775762540011	Aman	aa@gmail.com
2	876834788522	Neha	nn@gmail.com
3	996677898677	Neha	ss@gmail.com
4	796454638800	Vimal	vv@gmail.com

- Here, `Cust_Id` is a **Surrogate Key**.
- It's **just a number** (1, 2, 3, 4), automatically generated.
- It doesn't mean anything in real life (unlike Aadhar_No or Email).

♦ Composite (Compound) Key

 A **composite key** is when we need **more than one attribute together** to uniquely identify a row in a table.

- A single column (like `Cust_Id` or `Product_Code`) **alone** cannot uniquely identify records.
- But if we **combine them**, they form a **unique identifier**.

Example Table

Cust_Id	Order_Id	Product_Code	Product_Count
C01	001	P111	5
C02	012	P111	8
C02	012	P222	6
C01	001	P333	9

Why not single columns?

- `Cust_Id` alone □ (C01 appears multiple times).
- `Product_Code` alone □ (P111 appears multiple times).
- `Order_Id` alone □ (001 and 012 repeat with different products).

None of them is unique by itself.

□ Composite Key in this case

□ **{Cust_Id, Product_Code}** → Together they make each row unique.

- Row 1 → (C01, P111)
- Row 2 → (C02, P111)
- Row 3 → (C02, P222)
- Row 4 → (C01, P333)

No duplicates when we use both attributes together.

Primary Key vs Foreign Key

Aspect	Primary Key	Foreign Key
Definition	A column (or set of columns) that **uniquely identifies each row** in a table.	A column (or set of columns) in one table that refers to the **Primary Key in another table**.
Uniqueness	Must be **unique** (no two rows can have the same value).	Can have **duplicate values** (because many rows can refer to the same record in another table).
NULL values	Cannot be NULL.	Can be NULL (if the relationship is optional).
Existence	Only **one Primary Key** per table.	A table can have **multiple Foreign Keys**.
Purpose	To **identify records** uniquely in the same table.	To **link two tables** and maintain referential integrity.
Where defined?	Defined **within the same table**.	Defined in the **child table**, referencing the parent table.

Entity Relationship Diagram

- The **ER model** was introduced by **Peter Chen** in **1976**.
- The ER model defines the **conceptual (or logical) view of a database**.
- It is used for **designing databases**.
- It works around **real-world entities** and the **relationships** among them.
- A database schema in the ER Model can be represented pictorially as **ER diagrams**.
- An ER diagram maps well into a **relational schema**.

➢ Why ER Model is Useful?

The ER (Entity-Relationship) model is useful for several reasons:

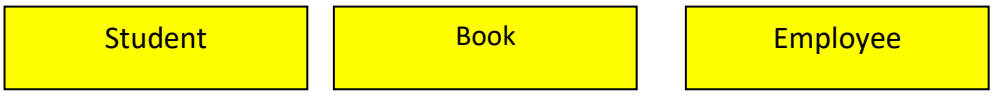
- An **ER Model maps well to the relational model**, meaning its constructs can be easily converted into relational tables.
- It serves as a communication tool, allowing **database designers to communicate the database design to users**.
- It functions as a **design plan**, helping **database developers implement a data model** in specific DBMS software.

➢ Elements of the E–R model:

1. Entities
2. Attributes
3. Entity sets
4. Relationships
5. Relationship sets

1. Entities

- An **entity** is just a **real-world object** that we store information about in a database.
- Each entity is different from others.
- Entities are represented by means of **rectangles**.

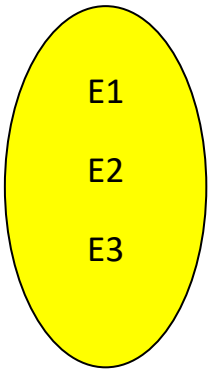


Examples:

- Student, Employee, Doctor, Book, Car, etc.
- A **Student** has properties like Roll Number, Name, Address, etc.
- An **Employee** has properties like Employee_ID, Name, Salary, etc.

Entity Set

- An **Entity Set** is a collection of similar types of entities that share the same attributes.
  - **Example:** A Students set may contain all the students of a school; a Teachers set may contain all the teachers of a school from all faculties.



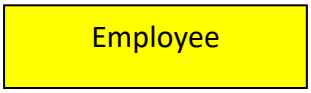
Entity Set

- Entity sets need not be **disjoint**.
  - **Example:** The entity set Employee (all employees of a bank) and the entity set Customer (all customers of the bank) may have members in common.

Strong Entity (Strong Entity Set)

A **strong entity** is an entity that:

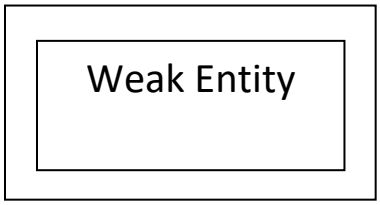
- **Always has a primary key**.
- Its existence is **not dependent** on any other entity; it is independent.
- A set of strong entities is known as a **strong entity set**.
- A strong entity is represented by a **single rectangle** in an ER diagram.



Weak Entity (Weak Entity Set)

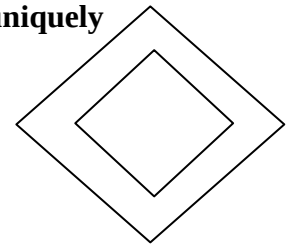
A **weak entity** is an entity that:

- Does not have enough attributes to form a **primary key**.
- Is **dependent on a strong entity** for its existence.
- A set of weak entities is known as a **weak entity set**.
- A weak entity is represented by a **double rectangle** in an ER diagram.



➢ Weak Relationship

 A **weak relationship** exists when one entity (called a **weak entity**) **cannot be uniquely identified by its own attributes alone**.



- It depends on another entity (called a **strong entity**) for identification.
- Represented in **ER diagrams** by a **double diamond**.

Example 1

ER Diagram:

- The ER diagram shows a relationship between `loan` and `payment` entities.
- The `loan` entity is a **Strong Entity (Identifying Entity Set)**. It has attributes `loan-number` and `amount`.
- The `loan` entity is represented by a single rectangle.
- The `payment` entity is a **Weak Entity**. It has attributes `payment-number`, `payment-date`, and `payment-amount`.
- The `payment` entity is represented by a double rectangle.
- The relationship between `loan` and `payment` is `loan-payment`, which is a **Weak Relation (Identifying Relation)**. It is represented by a double diamond.

Table Data:

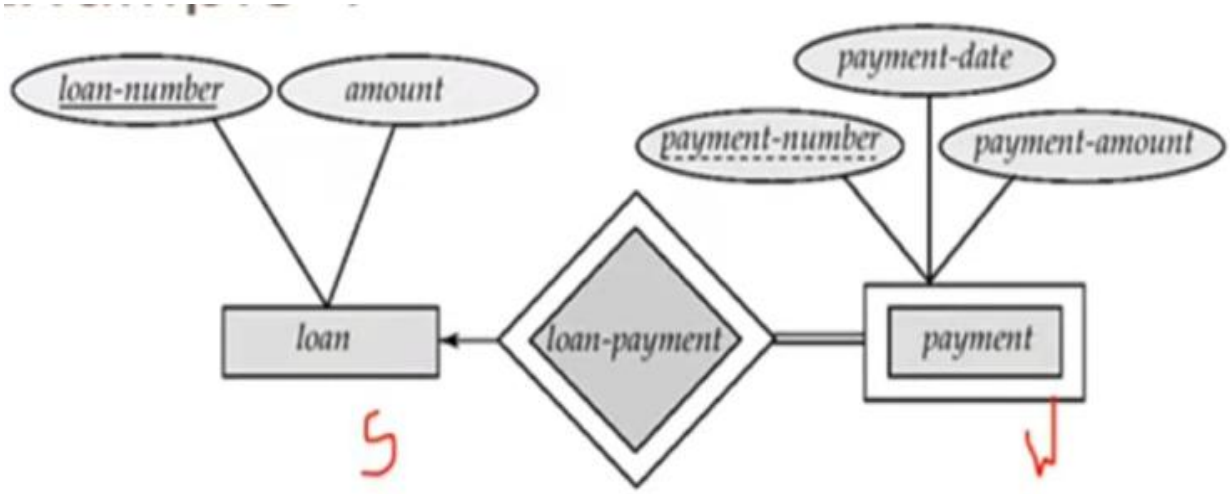
Loan Table

Loan_no	Amount
L1	1,00,000
L2	2,00,000
L3	3,00,000

Payment Table

Payment_no	Payment_date	Payment_amount

Payment_no	Payment_date	Payment_amount
1	05-06-2020	5000
1	08-07-2020	10000
1	10-08-2020	15000
2	05-07-2020	5000
2	08-08-2020	10000
2	10-09-2020	15000

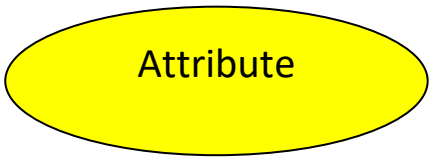


Representation

S.NO	STRONG ENTITY	WEAK ENTITY
1.	Always has a **primary key**.	Has a partial key or **discriminator key**.
2.	Existence is **not dependent** on any other entity.	Existence is **dependent on a strong entity**.
3.	Represented by a **single rectangle**. ▭	Represented by a **double rectangle**.
4.	The relationship between two strong entities is represented by a **single diamond**. ◇	The relationship between a strong and a weak entity is represented by a **double diamond**.
5.	Can have either total or partial participation in a relationship.	Always has **total participation** in its identifying relationship.

2. Attributes

- **Attributes** are the **properties/details** of an entity.
- They give more information about the entity.
- Each entity must have at least one attribute that **uniquely identifies it** → this is called the **Primary Key**.
- Attributes take their values from a **domain** (the set of possible values).
- Attributes are represented by ellipse



Examples:

- For Student → Roll Number, Name, Address, Age.
- For Employee → Employee_ID, Name, Department, Salary.
- For Disease → Disease Name, Causing Organism.

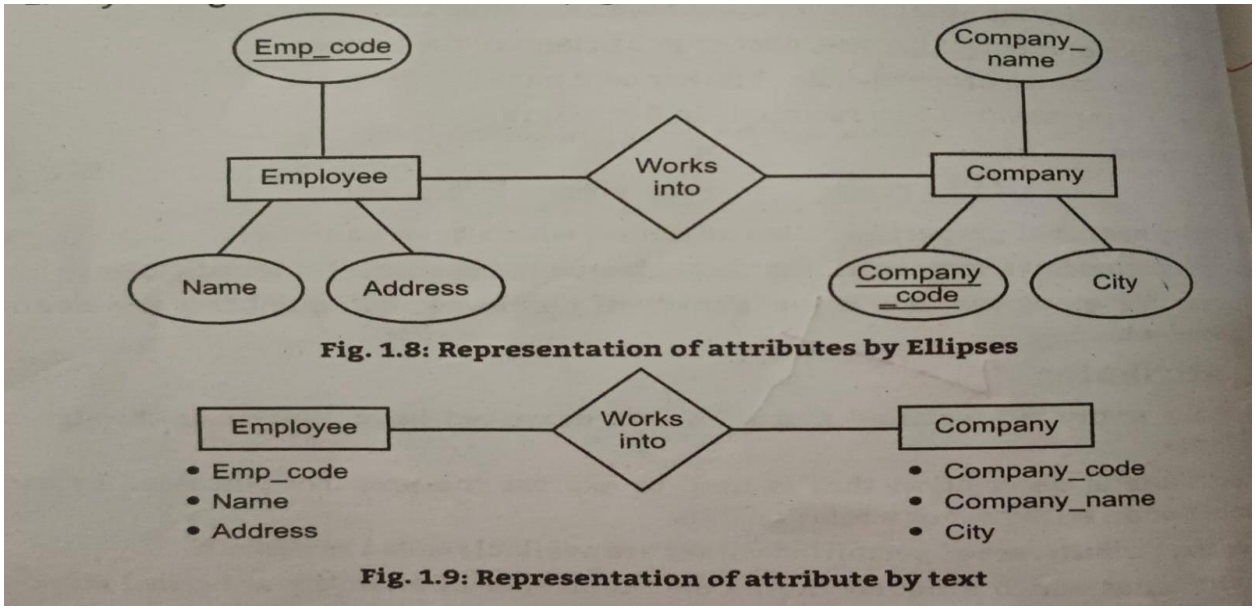


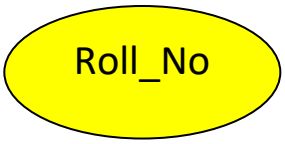
Fig. 1.8: Representation of attributes by Ellipses

Fig. 1.9: Representation of attribute by text

- Types of attributes

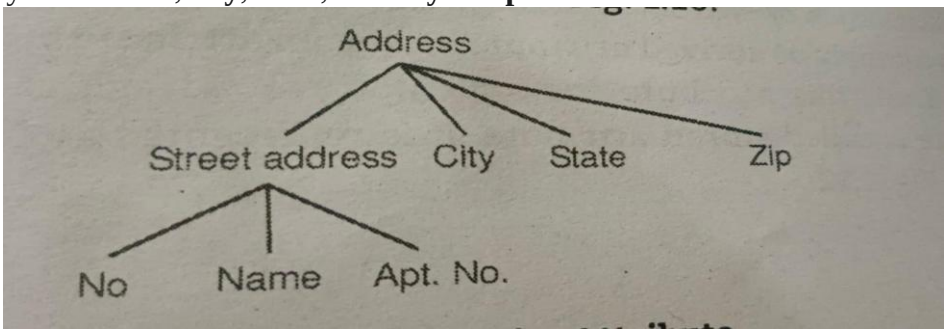
1. Simple Attributes

- **Simple attributes** are atomic values that cannot be divided further.
- **For example:**
  - A student's mobile number is an atomic value of 10 digits.
  - A Birth Date is an atomic value of date-month-year.



2. Composite Attributes

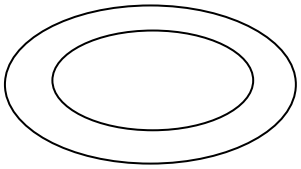
- **Composite attributes** are made of more than one simple attribute.
- A composite attribute is divided in a tree-like structure.
- **For example:**
  - A student's complete **name** may have **first_name** and **last_name**.
  - An **address** may have **street**, **city**, **state**, **country** and **pin code**.



3. Single-valued Attribute

- **Single-value attributes** contain a single value.
- **For example:**
  - Social_Security_Number, Aadhar_card_no, Roll_no

4. Multi-valued Attribute



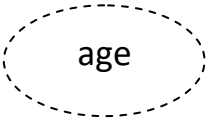
- **Multi-valued attributes** may contain more than one value.
- They are represented by a **double-ellipse**.

For example:

- A person can have more than one phone number, email address, etc.

5. Derived Attribute

- **Derived attributes** are attributes that do not exist in the physical database, but their values are derived from other attributes present in the database.
- Derived attributes are depicted by a **dashed ellipse**.



For example:

- **age** can be derived from `date_of_birth`.
- **average_salary** in a department should not be saved directly in the database; instead, it can be derived.

6. Null Attribute

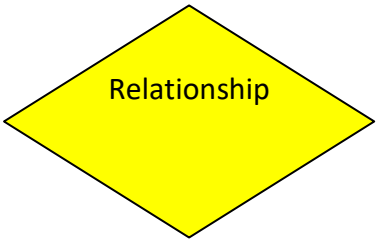
- **Null attributes** are attributes in a database that do not have a value assigned for a particular record.
- Null means **"no value"** or **"unknown"**, not zero or blank.
- Null is used when:
  - The value is **not known** at the time of data entry.
  - The value is **not applicable** for that particular record.

Examples:

1. **Phone_Number** of an employee → If an employee hasn't provided a phone number yet, the field will be `NULL`.
2. **Passport_No** in a student database → Many students may not have a passport, so it will be `NULL`.

3. Relationships

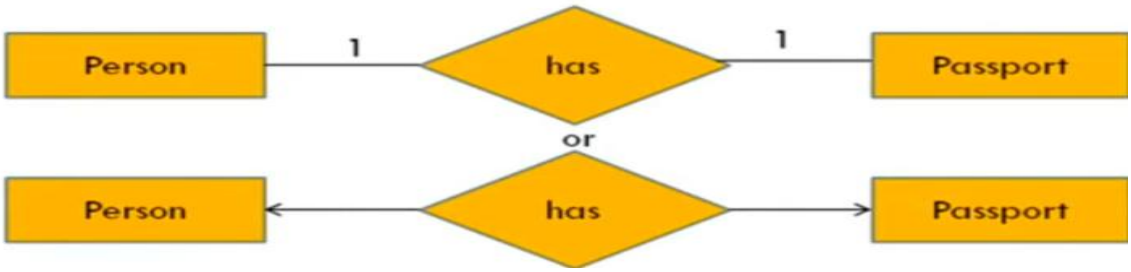
- A **Relationship** is an association among entities.
- **For example:**
  - an employee **works_at** a department.
  - a student **enrolls** in a course.
  - Here, **Works_at** and **Enrolls** are called **relationships**.
- Relationships are represented by a **diamond-shaped box**.



➢ **Types of Relationship(depends on cardinality)**

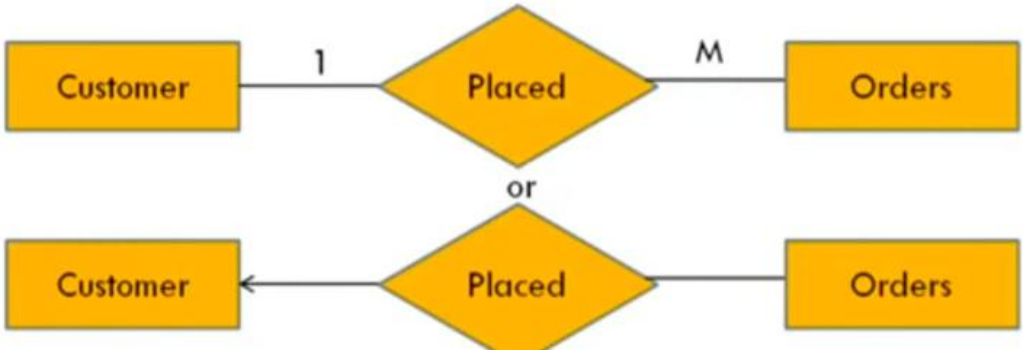
I. One-to-One (1-1) Relationship

- One entity from entity set A can be associated with at most one entity of entity set B and vice versa.
- **For example:** A person has only one passport, and a passport is given to one person.



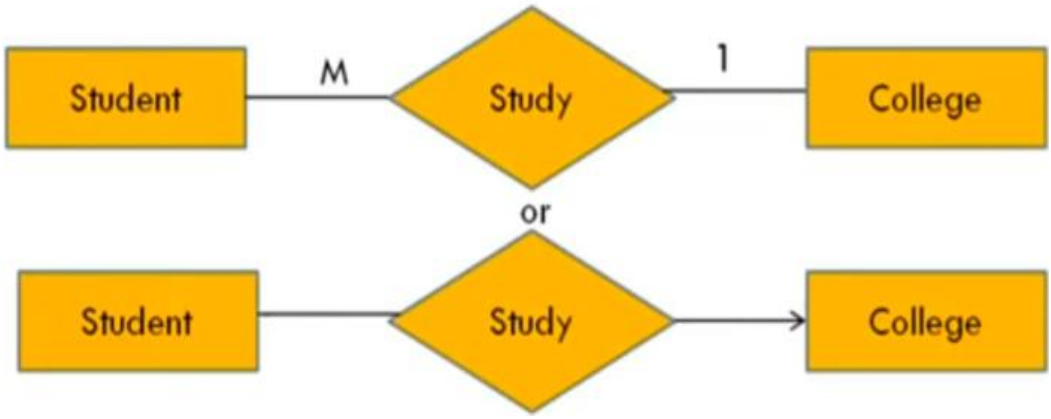
II. One-to-Many (1-M) Relationship

- One entity from entity set A can be associated with more than one entities of entity set B, however an entity from entity set B can be associated with at most one entity.
- **For example:** A customer can place many orders, but an order cannot be placed by many customers.



III. Many-to-One (M-1) Relationship

- More than one entities from entity set A can be associated with **at most** one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.
- For example - many students can study in a single college but a student cannot study in many colleges at the same time.

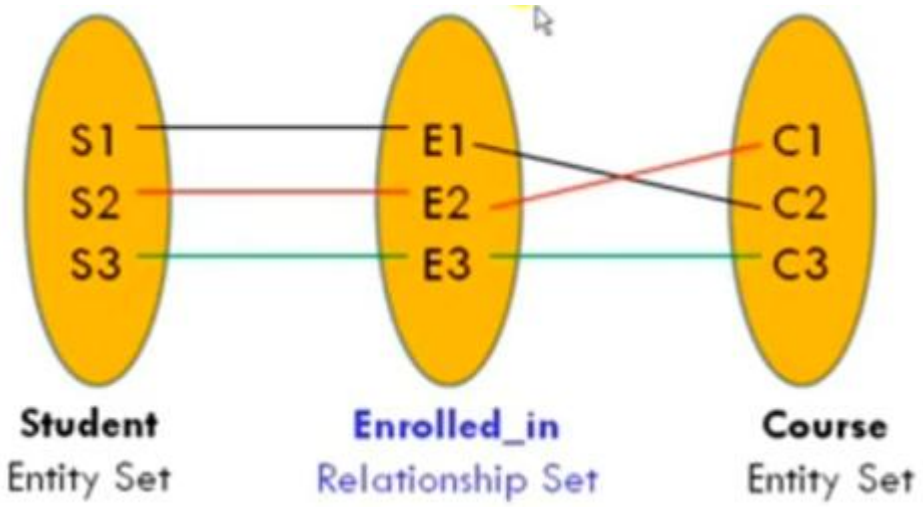


IV. Many-to-Many (M-N) Relationship

- One entity from A can be associated with more than one entity from B and vice versa.
- **For example,** an Employee can be assigned to many Projects and a Project can have many Employee.

➢ **Relationship Set**

- A set of relationships of a similar type is called a **relationship set**.
- The following relationship set **Enrolls (E1, E2, E3)** depicts:
  - **S1** is enrolled in **C2**
  - **S2** is enrolled in **C1**
  - **S3** is enrolled in **C3**



ER Diagrams

Case Study 3: A movie studio wishes to institute a database to manage their files of movies, actors and directors. The following facts are relevant:

(i) Each actor has appeared in many movies.

(ii) Each director has directed many movies.

(iii) Each movie has one director and one or more actors.

(iv) Each actor and director may have several addresses.

Draw E-R diagram.

Solution:

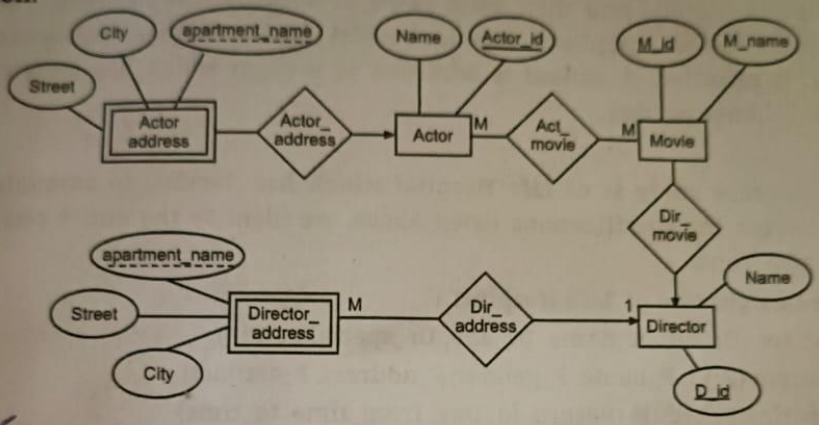


Fig. 1.38: Film Industry System

Case Study 4: In a nursery, the plants are sold to the customers. These plants are Flowering and Non-flowering only. Nutrients are given to the plant with some quantity. Nutrients include Pesticides, Watering and Manure.

Solution:

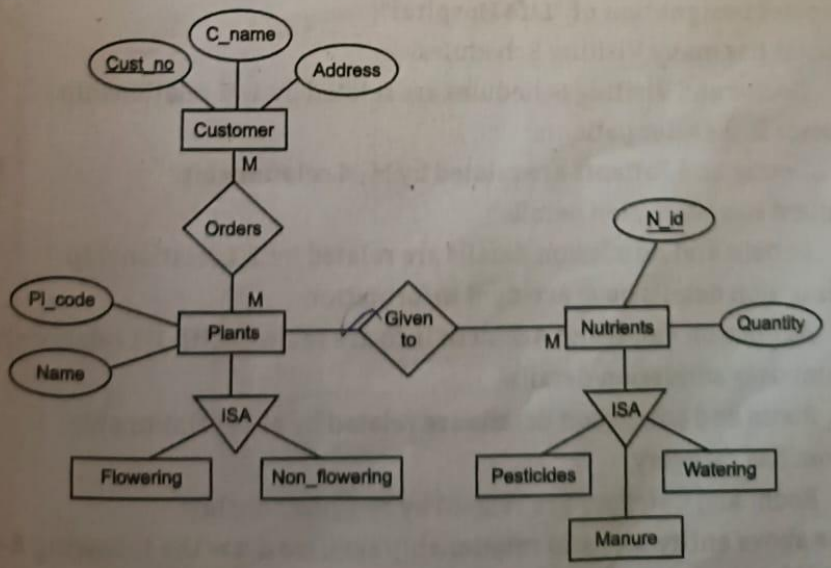


Fig. 1.39: Plant Nursery System

1.9 CASE STUDIES ON ER MODEL

- Solved case studies are included to show how E-R model is used for conceptual de
- To learn the effective way of drawing and designing E-R diagram.

Case Study 1: Construct an E-R diagram for a car insurance company that has a se customers. Each customer owns one or more cars. Each car is associated with zero to number of recorded accidents.

Solution:

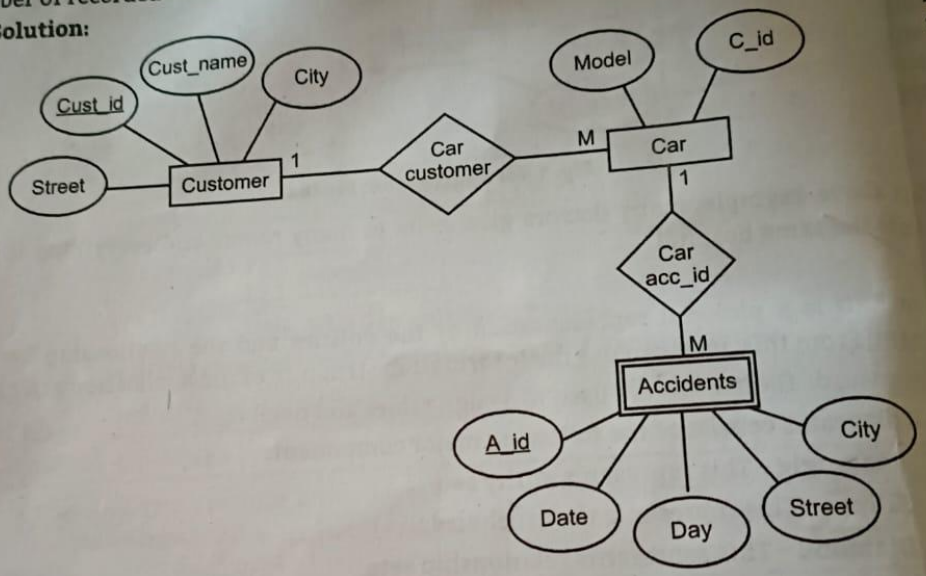


Fig. 1.36: Car Insurance System

Case Study 2: Draw an E-R diagram for order processing system where a person c give order for many items by specifying its quantity. And also an item can be ordered many persons.

Solution:

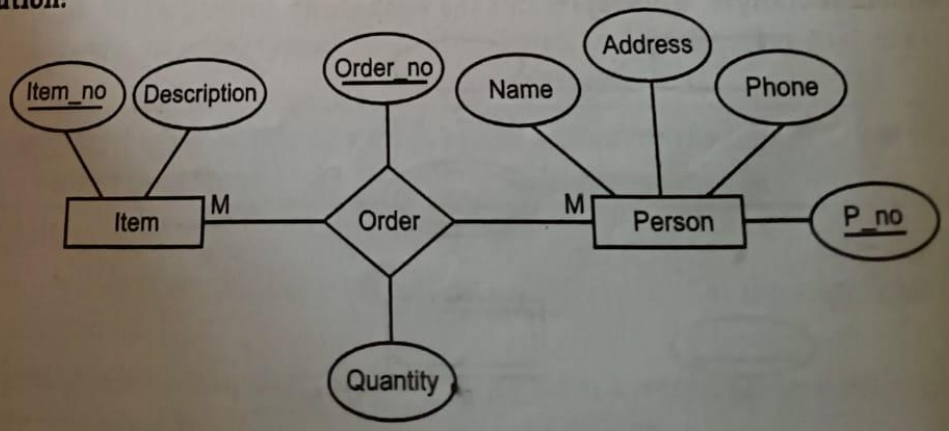
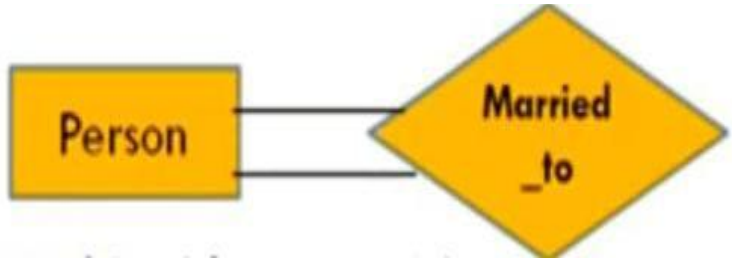


Fig. 1.37: Order Processing System

Types of relationships:

1. Unary Relationship (Degree = 1)

- Only one entity is involved in the relationship.
- It means the relationship happens within the same entity.



2. Binary Relationship (Degree = 2)

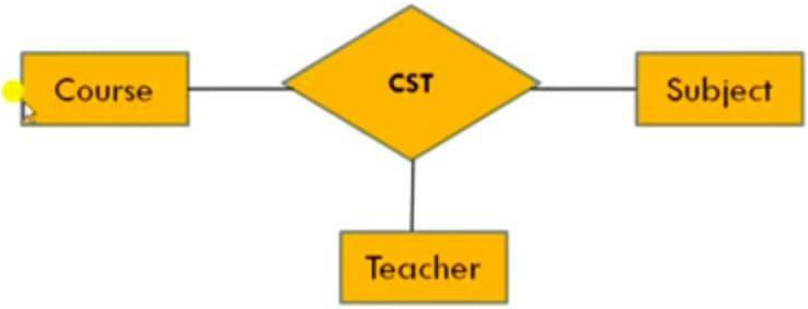
- Here, two different entities are involved in the relationship.
- It is the most common type of relationship.

- For example, Student is enrolled in Course.

3. Ternary Relationship (Degree = 3)

- When three entities take part in a single relationship, it is called a Ternary Relationship.
- It means that three different things (entities) are connected through one relationship**.**

- For example, The University might need to record which teachers taught which subjects in which courses.



Features of ERD

Extended Entity Relationship (ER) Features

- In the **1980s**, data became more **complicated and large**.
- The **old ER model** (Entity Relationship Model) was not enough to handle such complex databases.
- So, some **new features** were added to make the ER model stronger and more useful.These new features are called the **Extended ER Model**.
  It includes three important concepts:

Generalization

- **Meaning:**
  Generalization means **combining many small/specific entities into one bigger/general entity**.
  Example: "Car" and "Bike" can be combined into "Vehicle".
  It is a **bottom-up approach**.

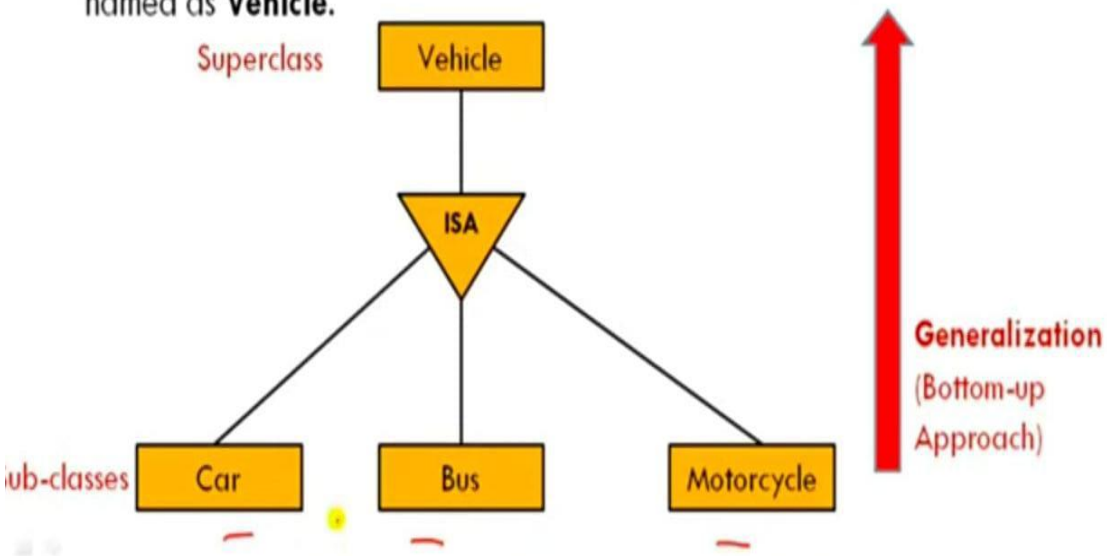
  Example:

- We have **three entities**:
  - **Car**
  - **Bus**
  - **Motorcycle**
- Instead of treating them separately, we notice they all have **common properties** (like Vehicle Number, Model, Company, etc.).
- So, we **combine them into a higher-level entity** called **Vehicle**.

 This new **Vehicle** entity is the **Superclass**,
and **Car, Bus, Motorcycle** are **Subclasses**.

- The **triangle with "ISA"** means:
  - Car **is a** Vehicle
  - Bus **is a** Vehicle
  - Motorcycle **is a** Vehicle

- Consider we have 3 sub entities Car, Bus and Motorcycle. Now these three entities can be generalized into one higher-level entity (or super class) named as **Vehicle.**



Specialization

- **Meaning:**
  Specialization means **dividing one large/general entity into smaller, more specific entities** based on their characteristics.
- **It is a "Top-down approach"**
  This means we **start from a higher-level entity** (like Employee) and **break it into lower-level entities** (like Teacher, Clerk, etc.).

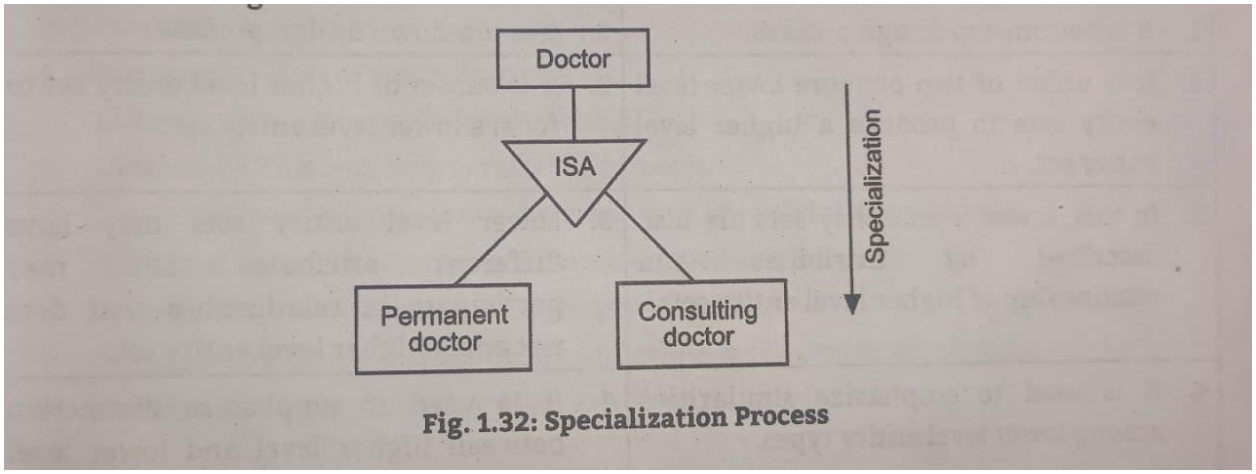


Fig. 1.32: Specialization Process

We have **one main entity: Doctor**

Now, doctors can have **different specializations** — for example:

- **Surgeon**
- **Physician**
- **Dentist**

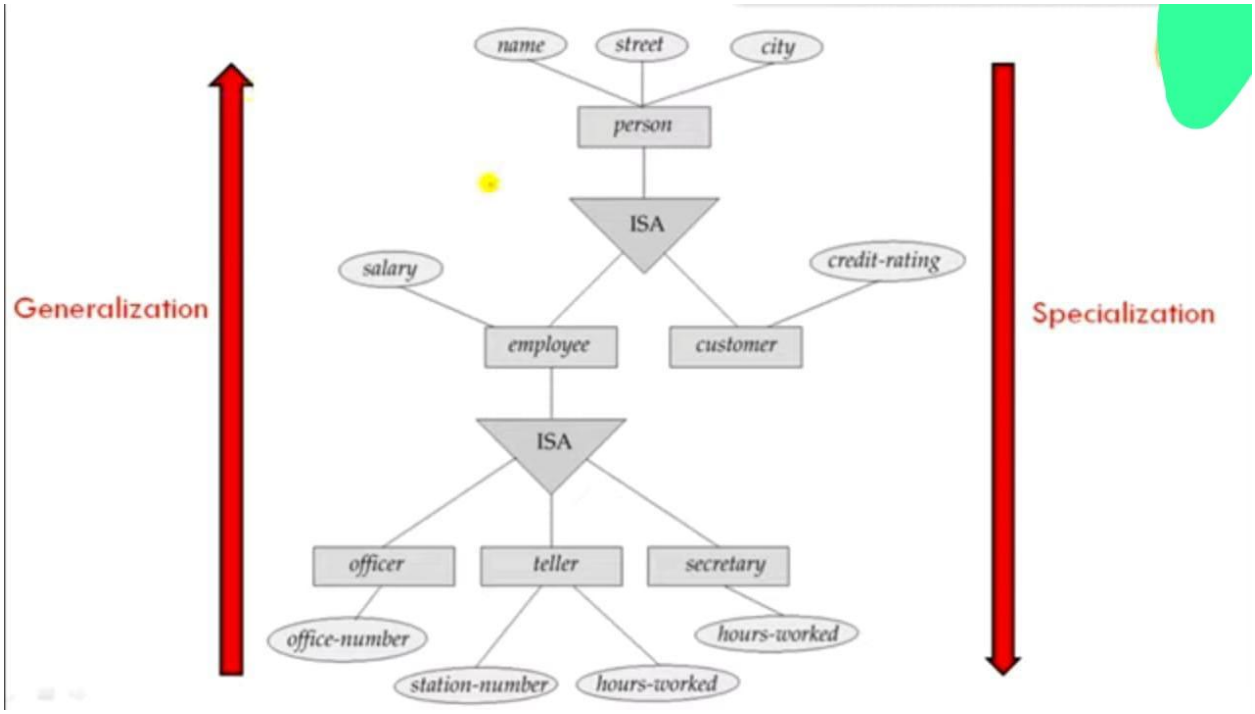
Even though all are doctors, they perform **different types of work**.

So, we **divide** the main entity **Doctor** into these **smaller, more specific entities** based on their special skills.

 The main entity **Doctor** is the **Superclass**,
and **Surgeon, Physician, Dentist** are the **Subclasses**.

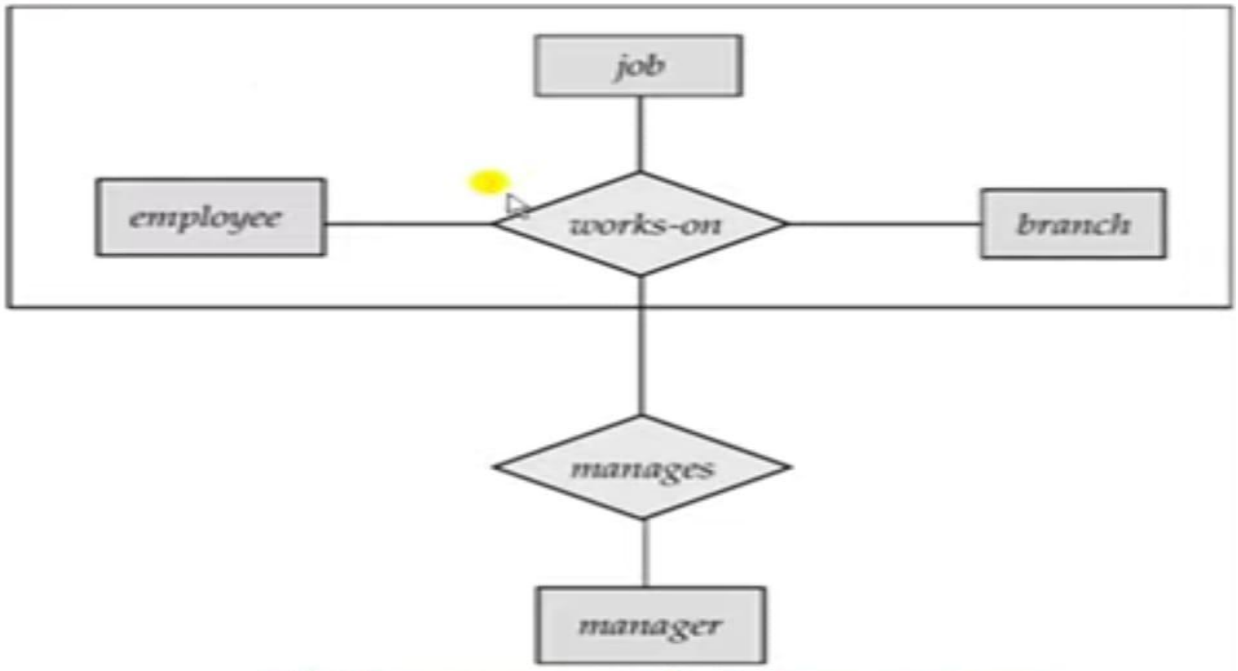
The **triangle with "ISA"** means:

- Surgeon **is a** Doctor
- Physician **is a** Doctor
- Dentist **is a** Doctor



Aggregation

- **Meaning:**
  Aggregation is used **when a relationship itself needs to be treated as an entity**.
  It is an **"association of relationships"** — that means we make a relationship act like an entity so that it can have its own attributes or take part in another relationship.



ER Diagram with Aggregation

Example Explanation

Let's take the entities:

- **Employee**
- **Job**
- **Branch**
- **Manager**

□ Step 1: Normal relationship

An **Employee works on a Job**.
□ This is a **relationship** between **Employee** and **Job**.

□ Step 2: Add another entity — Branch

Now, this **Job** is **assigned in a Branch**, and each **Branch has a Manager** who oversees the Job.

But here's the problem:
The relationship **"Employee works on Job"** is connected to **Branch (through Manager)** — not just Employee or Job alone.

So, we need a way to **link the relationship itself ("works on") with another entity (Branch/Manager)**.

□ Step 3: Treat the relationship as an entity

We treat the relationship **"Employee works on Job"** as a **new higher-level entity** (say, "Work_Assignment").

Now, we can relate this new entity **Work_Assignment** with **Manager** (or **Branch**).

What is a Relational Model?

A **Relational Data Model (RDM)** is a simple way of storing data in the form of **tables**.

Each **table** contains **rows and columns** — just like an Excel sheet.

- **Rows** represent individual records (data entries)
- **Columns** represent different fields (attributes)

Example

Let's take an example of a **bank** database

1Customer Table

cust_id	name	city
01	Millar	London
02	Roger	Mansfield

cust_id	name	city
03	Brown	Blackburn

This table stores **customer information**.

---

2 Account Table

account_no	Amt
1224	10,000
3424	5,000

This table stores **account information**.

3Customer_Amt Table (Relationship Table)

cust_id	account_no
01	1224
02	3424

This table **connects** customers to their accounts.
It shows **which customer owns which account**.

□ How it Works

- Each table has its **own data**.
- When we want to know, for example, *"What is the amount in Millar's account?"*
  → We can **link the tables** using `cust_id` and `account_no`.

This connection between tables is what makes it a **Relational Model**.

Advantages of Relational Model

1. **Simple and Easy:**
   Data is stored in tables, which is easy to understand.
2. **Flexible:**
   We can easily add, delete, or update data.
3. **Relationships are Clear:**
   The use of foreign keys (like `cust_id` and `account_no`) helps connect data from multiple tables.
4. **Less Memory Needed:**
   Works well even on computers with small memory or limited processing power.

Disadvantages

1. **Slower for Very Big Data:**
   When too many tables and relationships exist, it may slow down.
2. **Requires Structured Data:**
   Works best only if data fits properly into rows and columns.

- BASIC CONCEPTS: RELATION, TUPLE, ATTRIBUTE

1. Relation

A **Relation** means a **table** that stores data about a particular thing (like Student, Employee, or Product).

Each **relation** has:

- A **name** (e.g., Student)
- **Columns (attributes)**
- **Rows (tuples)**

 Example:

Relation name: Student

RollNo	Name	Address
1	Ravina	Aundh
2	Nikita	Kothrud
3	Rupali	Sangvi

RollNo	Name	Address
4	Ashmit	Kothrud
5	Unmesh	Katraj
6	Omkar	Sangvi

So, this **Student table** is a **relation**.

---

2. Attribute

An **Attribute** is a **column** in the table.
It shows one **property or characteristic** of the data.

In the **Student** table:

- `RollNo`, `Name`, and `Address` are **attributes**.

 Think of attributes as “headings” of the columns.

---

3. Tuple

A **Tuple** is a **single row (record)** in the table.
It shows all the information about one particular item/person.

For example:

RollNo	Name	Address
1	Ravina	Aundh

This one row (Ravina’s data) is **one tuple**.

If there are 6 rows → there are **6 tuples**.

4. Cardinality

Cardinality = Number of **rows (tuples)** in a table.

 In our **Student** table, there are 6 rows,
so **Cardinality = 6**.

---

5. Degree

Degree = Number of **columns (attributes)** in a table.

 In our **Student** table, there are 3 columns,
so **Degree = 3**.

---

5. Domain

A **Domain** means the **set of possible values** that an attribute can take.

In simple words,
 Domain = the range of valid values for a column.

Each attribute has its own domain, which defines:

- What **type of data** can be stored (number, text, date, etc.)
- What **values are allowed**

Example

Attribute	Example Values	Domain (Allowed Values / Data Type)
RollNo	1, 2, 3, 4, 5, 6	Integer (numeric values only)
Name	Ravina, Nikita, Rupali	Text / String
Address	Aundh, Kothrud, Sangvi	Text / City names

So here:

- The **domain of RollNo** is “numbers only”.
- The **domain of Name** is “alphabetic names”.
- The **domain of Address** is “city names”.