DBMS CONTENT BY MURALI KRISHNA SENAPATY

BASED ON DATABASE MANAGEMENT SYSTEM BY ELMASRI & NAVATHE

UNIT:1 **(6 Hours)**

Introduction to database Systems: advantages of database system over traditional file system, Basic concepts & Definitions, Database users, Database Language, Database System Architecture,Schemas, Sub Schemas, & Instances, database constraints, 3-level database architecture, Data Abstraction, Data Independence, Mappings, Structure, Components & functions of DBMS, Data models.

Entity-Relationship model: Basic concepts, Design process, Constraints, Keys, Design issues, E-R Diagrams, weak entity sets, Extended E-R features – generalization, specialization, aggregation, Reduction to E-R database schema.

CHAPTER1: Introduction to DBMS

Databases and database systems are an essential component of life in modern society: most of us encounter several activities every day that involve some interaction with a database.

For example:

> if we go to the bank to deposit or withdraw funds, if we make a hotel or airline reservation,

> if we access a computerized library catalog to search for a bibliographic item, or

> if we purchase something online—such as a book, toy, or computer—chances are that our activities will involve someone or some computer program accessing a database.

> Even purchasing items at a supermarket often automatically updates the database that holds the inventory of grocery items.

Data: These are the known facts that can be recorded and have an implicit meaning.

It is a raw collection of facts, figures, symbols, text etc.

The data in the new media technology has made it possible to store images, audio clips, and video streams digitally. These types of files are becoming an important component of **multimedia databases**.

Geographic information systems (GIS) can store and analyze maps, weather data, and satellite images.

Data warehouses and **online analytical processing (OLAP)** systems are used in many companies to extract and analyze useful business information from very large databases to support decision making.

Real-time and **active database technology** is used to control industrial and manufacturing processes.

Database:

A collection of logically related data together is called database.

Databases and database technology have a major impact on the growing use of computers. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, genetics, law, education, and library science.

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 hard drive, using a personal computer and software such as Microsoft Access or Excel. This collection of related data with an implicit meaning is a database.

A database has the following implicit properties:

Mini-World:

A database represents some aspect of the real world, sometimes called the **mini-world**. Changes to the mini-world are reflected in the database.

For example: Case study of a student information system
Data : some tables about students address, academic marks, subjects allotted, fees etc.
Operations on them like : insert, update, delete, new entry, searching

A random assortment of data cannot correctly be referred to as a database. A database is designed, built, and given input with data for a specific purpose. It has a group of users and some features which these users can use.

In other words, a database has some source from which data is derived, some degree of interaction with events in the real world.

In order for a database to be accurate and reliable at all times, if any changes in mini world observed then it must be reflected in database soon.

A database size can be complex. For example, the computerized catalog of a large library may contain half a million entries organized under different categories—by author's name, by subject, by book title and each category organized alphabetically.

DBMS:

When a database is computerized then it is called database management system.

It is a software package/ system to facilitate the creation and maintenance of a computerized database.

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database.

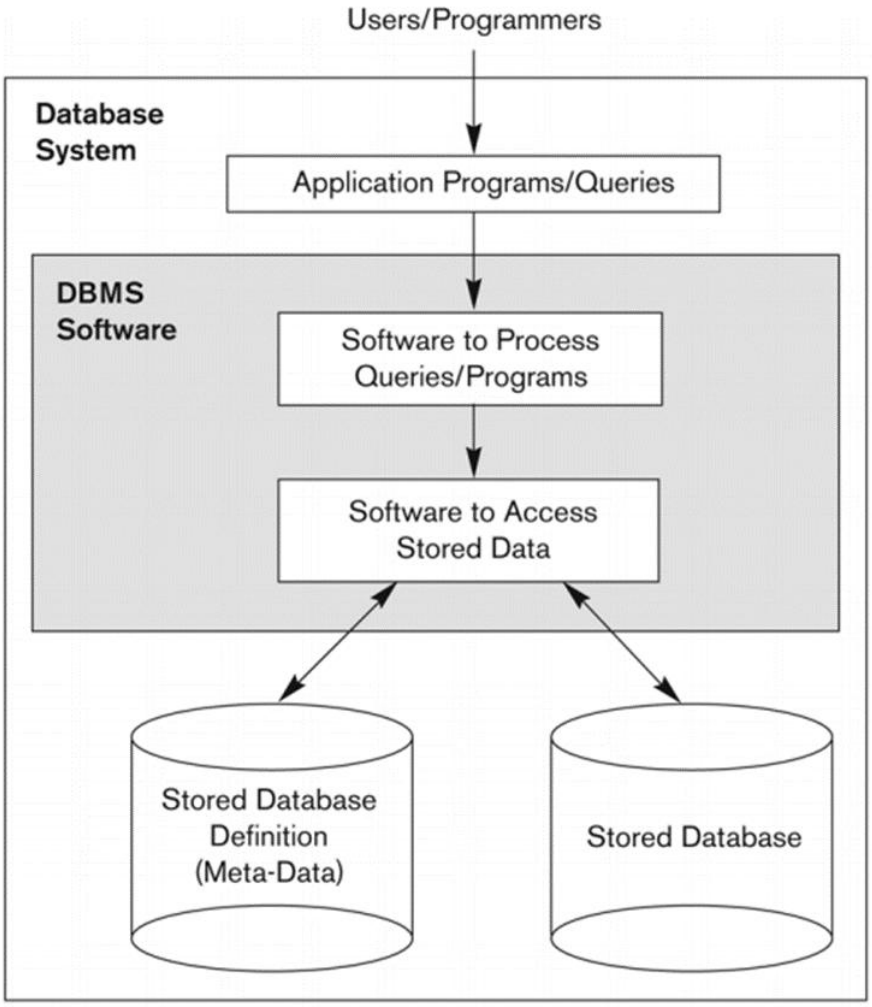
The DBMS is a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications.

Therefore the primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

Applications of DBMS:

- Banking systems
- Airline reservation systems
- University records
- Telecommunication systems
- Online shopping systems
- Social media platforms

Figure below represents a simple database system:



DBMS Functionality / Facilities:

It involves Defining, Constructing, Manipulating, Sharing, Protecting, Maintaining.

Defining: Defining a database involves specifying the data types, structures, and constraints of the data. The database descriptive information is stored in the form of a database catalog or dictionary; it is called **meta-data**.

Constructing: It is the process of storing the data on some storage medium that is controlled by the DBMS.

Manipulating: It includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in mini world, and generating reports from the data.

An application program accesses the database by sending queries or requests for data to the DBMS. A query typically causes some data to be retrieved; a transaction may cause some data to be read and some data to be written into the database.

Sharing: A database allows multiple users and programs to access the database simultaneously.

Protecting: It includes system protection against hardware or software malfunction (or crashes) and security protection against unauthorized or malicious access.

Maintaining: A typical large database may have a life cycle of many years, so the DBMS must be able to maintain the database system by allowing the system to evolve as requirements change over time.

CASE study on problems with manual system:

Case Study of University Database:

A UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment.
The database is organized as five files, each of which stores data records of the same type.
1. The STUDENT file stores data on each student,
2. the COURSE file stores data on each course,
3. the SECTION file stores data on each section of a course,
4. the GRADE_REPORT file stores the grades that students receive in the various sections they have completed, and the
5. PREREQUISITE file stores the prerequisites of each course.

To define this database, we must specify the structure of the records of each file by specifying the different types of data elements to be stored in each record.

To construct the UNIVERSITY database, Notice that records in the various files may be related. For example, the record for Smith in the STUDENT file is related to two records in the GRADE_REPORT file that specify Smith's grades in two sections.

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Database manipulation involves querying and updating.

Examples of queries are as follows:
■ *Retrieve the transcript—a list of all courses and grades—of 'Smith'*
■ *List the names of students who took the section of the 'Database' course offered in fall 2008 and their grades in that section*

Examples of updates include the following:
■ *Change the class of 'Smith' to sophomore*
■ *Create a new section for the 'Database' course for this semester*

Case study of Manual work at Govt. Offices (Importance of E-Governance).

File System:

Characteristics of file systems:

- Data is stored in a specific format.
- Each file is independent and has its own content.
- Data security may not exist in file content.
- Files have limited flexibility and can be difficult to maintain.

Drawbacks of file systems:

- Data Redundancy: Duplicate data stored in various locations.
- Data Inconsistency: Different values for same data across files.
- Difficulty in retrieving data when stored in multiple places.
- When a file structure need to be changed then each program associated with it need to be changed.
- Atomicity Problem: If a transaction fails during its process, the system should return to its previous consistent state. This is hard to manage in file-based systems.

- Concurrent Access Issue: When multiple users access the same file at the same time, it may lead to inconsistent data.

Characteristics of the Database Approach

In the database approach, a single repository maintains data that is defined once and then accessed by various users. In a database, the names or labels of data are defined once, and used repeatedly by queries, transactions, and applications.

Database Characteristics are:
1. Self-describing nature of a database system
2. Insulation between programs and data, and data abstraction
3. Support of multiple views of the data
4. Sharing of data and multi-user transaction processing

1. Self-Describing Nature of a Database System:
The database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called meta-data, and it describes the structure of the primary database.

2. Insulation between Programs and Data, and Data Abstraction:

program-data independence:
The structure of data files is stored in the DBMS catalog separately and the programs are stored separately. So the programs will not interact directly.

DBMS allows changing data structures and storage organization without having to change the DBMS access programs.

An operation is specified in two parts.

- The interface of an operation
- The method of the operation

program-operation independence:
User application programs can operate on the data by invoking the operations through their names and arguments, regardless of how the operations are implemented.

Data Abstraction: The characteristic that allows program-data independence and program-operation independence is called data abstraction.

A data model is used to hide storage details and present the users with a conceptual view of the database.

3. Support of Multiple Views of the Data:
Each user may see a different view of the database, which describes only the data of interest to that user. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. Example: bank account data is visualized in YONO app, online website, ATM, phonepe all are of different views as per the need.

4. Sharing of data and multi-user transaction processing:
It allows multiple users to access the database at the same time.
The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.

For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger.

These types of applications are generally called online transaction processing (OLTP) applications. A fundamental role of multi-user DBMS software is to ensure that concurrent transactions operate correctly and efficiently.

Database Users:

They are

1. Actors on the scene

2. Workers behind the scene

1. Actors on the Scene:

In large organizations, many people are involved in the design, use, and maintenance of a large database with hundreds of users. We identify the people whose jobs involve the day-to-day use of a large database; we call them the actors on the scene.

These people actually use, control the database content and design database applications.

They are Database Administrators, Database Designers, End users, System Analysts and Application Programmers.

1.1 Database Administrators(DBA): The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time.

1.2 Database Designers: They are responsible for designing the databases such as choosing appropriate structure, naming the fields, defining relationships and communicating to end users.

They identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data. They must communicate with the end-users and understand their needs.

1.3 End users: They are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use.
Further end users are categorized as : Casual End users, Naive end users, Sophisticated end users and Standalone users.

They are:
> Casual end users: They occasionally access the database, but they may need different information each time.
> Naive or parametric end users: They make up a large portion of database end users. They constantly querying and updating the database, using standard types of queries and updates. Example: Bank Tellers, Reservation Agents etc.
>Sophisticated end users: They include engineers, scientists, business analysts, and others who implement their own applications to meet their complex requirements.
>Standalone users: They maintain personal databases by using menu-based or graphics-based interfaces. Example: the user of a tax package stores a variety of personal financial data for tax purposes.

1.4 System Analysts: They handle the requirements of end users, especially naive and parametric end users, and develop specifications.

1.5 Application Programmers: **They implement the end user specifications in programs; then they test, debug, document, and maintain these** canned **transactions.**

2. Workers behind the Scene:

They are system designer and implementer, Tool developers, Operators and maintenance personnel.

2.1 System Designer: They design and implement the DBMS modules and interfaces as a software package.

2.2 Tool developers: They develop the tools which are optional packages and included in database design and performance monitoring.

2.3 Operators and maintenance personnel: They are responsible for the actual running and maintenance of the hardware and software environment for the database system.

Advantages of Using the DBMS Approach:

1) Reduces Data Redundancy:

In traditional software development utilizing file processing, every user group maintains its own files for handling its data-processing applications. This redundancy in storing the same data multiple times leads to several problems such as data inconsistency, repeated operations, waste of storage space.

2) Eliminates Data Inconsistency:

Since data is not duplicated and centrally stored at one location ans shared so the mismatching or conflicting data values are eliminated.

3) Data Independence:

The Data and the programs that use it are independent of each other, making changes easier.

4) Enforcing Data Integrity Constraints:
A DBMS allows **to follows certain rules such as integrity constraints like data types, valid ranges, and uniqueness.**
Ex: a number in range between 1 and 5,
Ex: a link between different tables.
Ex: catching wrong inputs like age, section in single character between A to G etc.

5) Data Security / Restricting Unauthorized Access:
When multiple users share a large database, it is likely that most **users will not be authorized to access all information in the database.** A DBMS should provide a security and authorization subsystem which the DBA uses to create accounts and to specify account restrictions as per the users. Example: bank account information accessed by customer, teller, branch manager, database administrator etc.

6) Concurrent Access:
Multiple users can access data at the same time in a network environment.

7) Providing Backup and Crash Recovery:
DBMS provides facilty for recovering data during hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery.

8) Atomic Transactions:
If a transaction fails or crashes, DBMS restores the database to its previous safe state.

9) Providing Persistent Storage for Program Objects:

In the traditional approach, data had to be converted to a suitable format before saving to files, and then converted back when reading — which takes time and effort.
But with Object-Oriented Programming (OOP) and **Object-Oriented Databases, this conversion is not needed**, because the data is stored directly as objects. This makes the process faster and easier, and keeps the data structure intact.

10) Providing Storage Structures and Search Techniques for Efficient Query Processing:
The DBMS provides facility to execute queries and Updates efficiently. Because the database is typically stored on disk, the DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records.

11) Providing Multiple User Interfaces:
Because many types of users with varying levels of technical knowledge use a database, **a DBMS should provide a variety of user interfaces.** Example: different interfaces to access data of a bank account holder in different platforms such as: ATM centre, online web site, mobile application, interface for a teller, interface for a manager, interface for a loan issuer, interface for a DBA etc.

12) Representing Complex Relationships among Data:
A database may include numerous varieties of data that are interrelated in many complex ways. Example: one-one, one-many, many-one, many-many, derived values, etc.

13) Permitting Inferencing and Actions Using Rules:
Some databases can use rules to find new information and do tasks automatically. For example, a rule can check which students are on probation without writing long programs. Databases can also have triggers that run

actions (like sending messages or updating data) when something changes.

14) Additional Implications of Using the Database Approach: **Additional facilities such as Setting Standards(allows the rules such as how data is named, stored, and shown)**

15) **Rapid Application Development (Creating reports is quicker way)**, Flexible (easier to update the structure of a new data), Real-Time Updates (keep updating instantly in background), Cost Savings(reduces duplicate work).

Functions of a Database Administrator (DBA)

1) Database Schema Definition:

- The DBA uses DDL (Data Definition Language) to define the logical structure of the database, including tables, relationships, constraints, and views.
- This schema acts as a blueprint for the entire database.

2) Storage Structure and Access Method Definition

- The DBA decides how data should be stored physically and what access methods (e.g., indexing, hashing) are best suited to improve performance.

3) Schema and Physical Organization Modification

- As user requirements evolve, the DBA modifies the schema or reorganizes data without disrupting ongoing operations or applications.

4) Granting User Authorization for Data Access

- Manages user roles and permissions, ensuring users can only access data relevant to their roles.
- Implements data privacy and access control through GRANT and REVOKE operations.

5) Ensuring Data Integrity and Security

- Enforces integrity constraints (e.g., foreign keys, domain constraints) to maintain accuracy and consistency.
- Implements security policies to protect data from unauthorized access or threats.

6) Database Recovery and Backup Management

- Develops and manages recovery strategies to restore the database in case of failure (e.g., system crash, power loss).
- Schedules and automates backups, ensuring minimal data loss and downtime.

7) Monitoring and Performance Tuning

- Continuously monitors database workloads, query execution, and system resources to identify bottlenecks.
- Tunes system parameters and SQL queries to improve throughput and response time.

8) Concurrency Control and Transaction Management

- Manages concurrent data access by multiple users using locking protocols or timestamp ordering.
- Ensures **ACID properties (Atomicity, Consistency, Isolation, Durability) for transactions.**

9) Coordination with Application Developers and End Users

- Works closely with developers to support application development.
- Provides technical support, data modeling assistance, and helps users understand how to interact with the database effectively.

CHAPTER 2: DATABASE SYSTEM CONCEPTS AND ARCHITECTURE

Data Models , Schemas and Instances:

A Data Model is a collection of concepts used to describe the structure of a database, operations and the constraints.

Category	Description	Examples
1.High-Level (Conceptual)	Provide concepts on in early phases of database design.	ER Model
2. Representational (Logical)	Represent data in formats understandable to computers but independent of physical storage.	Relational Model, Network Model, Hierarchical Model
3. Low-Level (Physical)	Describe how data is stored in computer memory or on disk.	Record-based storage models, File systems

- ER model is often used to design databases in conceptual phase.
- Relational model is the most popular logical model.
- Physical models are DBMS-specific and are not usually exposed to end-users.

Database Schema: A schema is the structure or blueprint of a database.

A schema is the overall logical structure of the entire database. **It defines the design and organization of data:**

what tables exist, what fields they contain, and the relationships among them. It is defined during database design and rarely changes. It is a collection of meta data.

Subschema:
A subschema is a subset of the schema that describes the view of the database for a particular user or application. It defines which part of the database a user or program can access.

Different users can have different subschemas for security, simplicity, or efficiency. A subschema is a personalized view of the database for each user.

Example:
A professor may only access COURSE and STUDENT name fields but not SID or personal data.
A student may only see their own records.

Instances: The actual content of the database at a particular point is known as instance of a database. It changes frequently as new data is added or modified. Multiple instances occur over time while the schema remains mostly constant.

Types of Database Architecture:

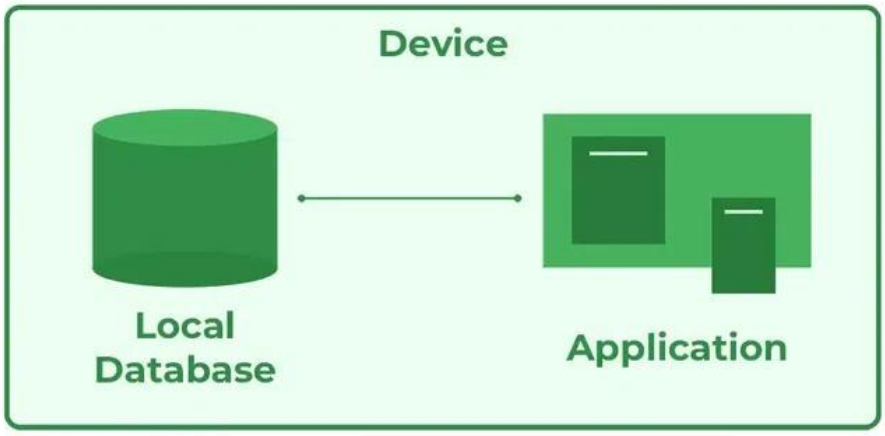
DBMS architecture is the design structure that defines how data is stored, managed, and accessed in a database system. It organizes the database into layers or tiers to ensure efficient data processing, better scalability, and easier management.

There are three types of DBMS Architecture that we use. They are:
1-Tier Architecture

2-Tier Architecture
3-Tier Architecture

1-Tier Architecture

In 1-Tier Architecture, the user works directly with the database on the same system. This means the client, server, and database are all located on one machine. The user can open the application, interact with the data, and perform tasks without needing a separate server or network connection.
A common example is Microsoft Excel.



Advantages of 1-Tier Architecture

1) Simple Architecture
2) Cost-Effective
3) Easy to Implement

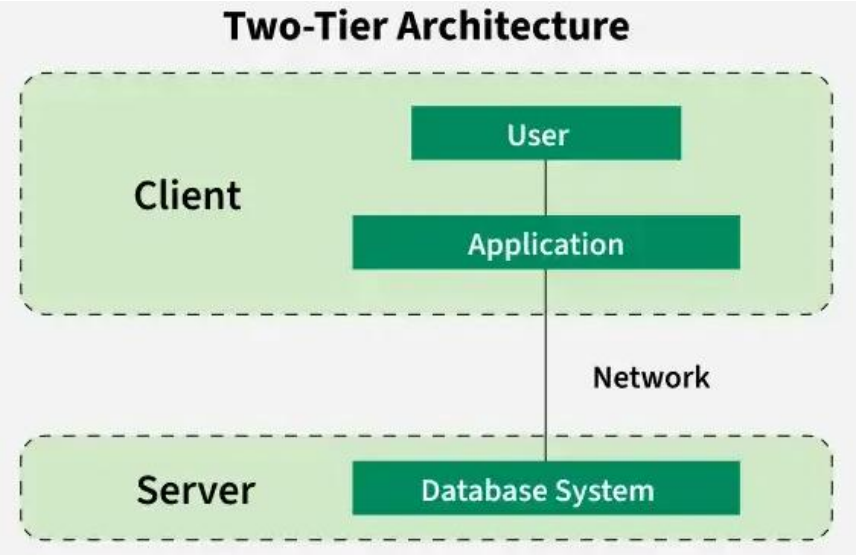
Disadvantages of 1-Tier Architecture

1) Limited to Single system
2) Poor Security
3) No Centralized Control
4) Hard to Share Data

2-Tier Architecture:

The 2-tier architecture is similar to a basic client-server model. The application at the client end directly communicates with the database on the server side. APIs like ODBC and JDBC are used for this interaction. The

server side is responsible for providing query processing and transaction management functionalities.



Advantages of 2-Tier Architecture

1) Easy to Use

2) Scalable: easily upgradable by adding clients or upgrading hardware.

3) Low Cost: It is cheaper than 3-Tier Architecture.

4) Simple and easily understandable

Disadvantages of 2-Tier Architecture

1) Limited Scalability compared to 3 tier. When the number of users increases, the system performance can slow down.

2) Security is limited

3) Tight Coupling: The client and the server are closely linked.

3. Three Tier Architecture / Three Schema Architecture / Three-Level Architecture

==========

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

1. External Schema:

The external or view level includes a number of external schemas or user views. **Each external schema shows the part of the database that a particular user group is interested in and hides the rest of the database** from that user group.

Here the online users and naive users are accessing the data. The DBMS having a powerful security mechanism which hides the unnecessary data from these users.

In 3-tier architecture we can observes that the database has exactly one conceptual schema, one physical schema but it have more than one external; schema connected with a group of users.

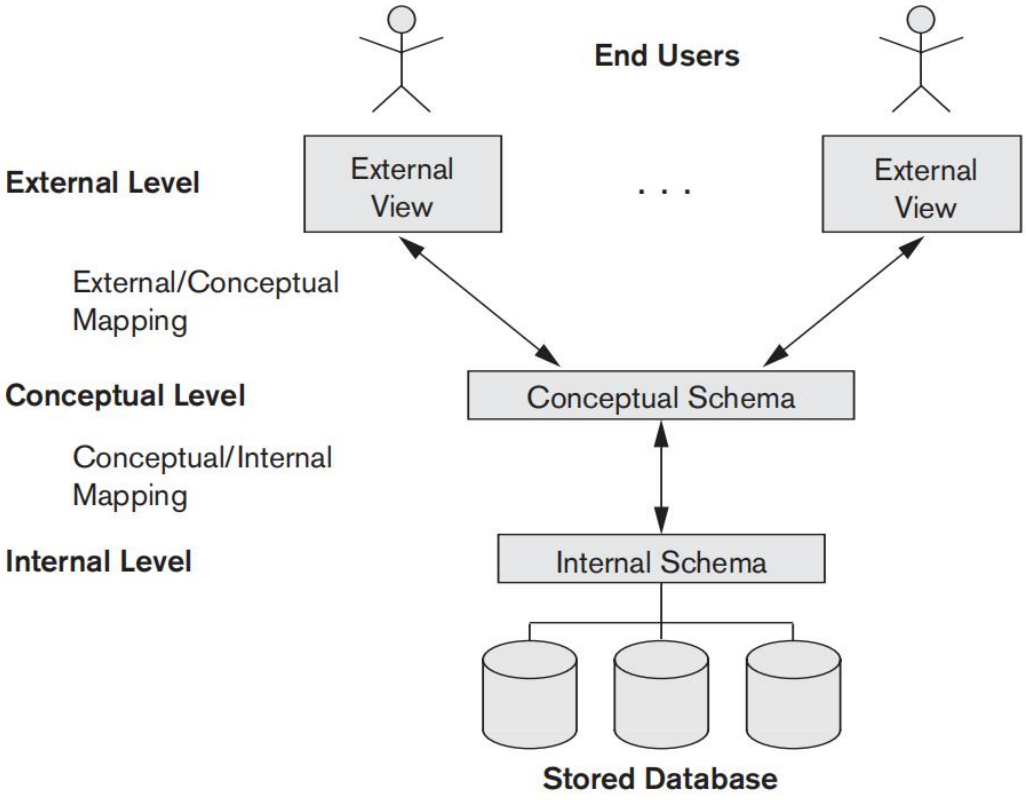
2. Conceptual Schema:

The conceptual level has a conceptual schema, which describes **the logical structure of the whole database** for a community of users. **The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.**

A database administrator can see the data, relationships among the data, constraints used, security mechanism of the database.

3. Internal Schema:

The internal level has an internal schema, which **describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.** It shows how the data is stored in memory, **data compression, record description for storage, indexes available, low level data structures used, data availability in tracks, sectors, cylinders etc.**



Advantages of 3-Tier Architecture

1) **Enhanced scalability**: Scalability is enhanced due to the distributed deployment of application servers.

2) Data Integrity: 3-Tier Architecture maintains Data Integrity. Since there is a middle layer between the client and the server. So **the data corruption can be avoided.**

3) **Security**: **It Improves Security**. It prevents direct interaction of the client with the server thereby reducing access to unauthorized data.

Disadvantages of 3-Tier Architecture

1) **More Complex**: 3-Tier Architecture is more complex in comparison to 2-Tier Architecture.

2) **Difficult to Interact**: It is difficult in comparison o 2-Tier Architecture.

3) **Slower Response Time**: Due to extra layer it may take more time to get a response compared to 2-Tier systems.

4) **Higher Cost**: Setting up and maintaining three separate layers requires more hardware, software, and skilled people.

Data Independence:

It is the capacity to change a schema or view at one level of a database system without affecting a schema at next higher level.

It is of two types:
1) Logical Data Independence
2) Physical Data Independence

Logical Data Independence:

It is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database by adding a record type or data item, to change constraints, or to reduce the database by removing a record type or data item.
In this case the external schema must not be affected.

Physical data independence:

It is the capacity to change the internal schema without having to change the conceptual schema.
Sometimes the physical files may be reorganized, data compression applied, splitting and merging of records in file are done. These changes must bot affect at conceptual level.

Database Languages :

A DBMS must **provides** appropriate **languages** and interfaces **to support different categories of users.**
For the DBMS, these languages are used for schema definition and data manipulation. **They are:**

1) Data Definition Language (DDL):

Used by database designers and DBAs to define the conceptual and internal schemas. It stores schema descriptions in the catalog. ***The DDL defines structures such as tables, attributes, domains, and constraints etc.***

2) Storage Definition Language (SDL):
It is used to define Internal Schema. ***It specifies how data is stored physically on storage media, including file structure, page size, access paths, and indexes etc.***

3) View Definition Language (VDL):
Used to define external views and mappings to the conceptual schema. In most relational DBMSs, SQL handles DDL, VDL, and DML together.
The VDL supports creating different logical views tailored for user groups***, allowing data abstraction and security.*** Views can be defined using SQL CREATE VIEW statements.

4) Data Manipulation Language (DML) / Query Language:
Used for data ***operations like retrieval, insertion, deletion, and update. It allows users*** and applications ***to perform transactions and query*** data stored in the database.

There are two types:

4.1) **Low-level (procedural) DML**: Procedural DML requires the user to specify both what data is needed and how to get it. It follows a step-by-step process to manipulate data.

4.2) **High Level (Non-procedural) DML**: Non-procedural DML requires the user to specify what data is needed, without specifying how to retrieve it.

DBMS Interfaces:

To support a range of users, DBMS provides several user-friendly interfaces:

1) Menu-Based Interfaces:

It guides users through query formulation ***using menu based interaction***. It is popular on web platforms.

2) Forms-Based Interfaces:

It displays forms for data input and querying. Here the Naive ***users fill out certain entries of form to retrieve matching records.***

3) Graphical User Interfaces (GUI):

Visual schema representation ***with drag-and-drop query building***. Often includes menus and forms.

4) Natural Language Interfaces:

Accept requests in English or other languages. ***They try to interpret user intent and convert it to queries.*** Limited capability, though keyword-based querying is emerging.

5) Speech Interfaces:

Allow spoken input and output for applications like flight info or banking inquiries using predefined vocabulary.

6) Parametric User Interfaces:

Designed for repetitive operations (e.g., bank tellers) with simplified commands or function keys for quick execution.

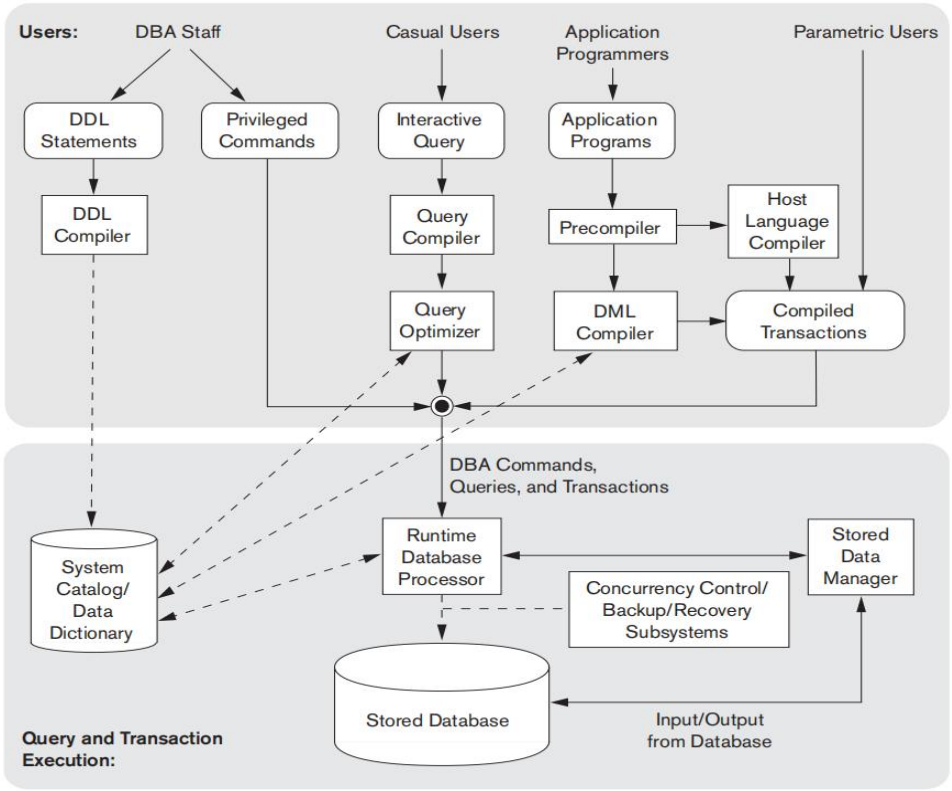
7) DBA Interfaces:

The interface provides privileged commands for administrators (e.g., account creation, schema modification, storage reorganization).

The Database System Environment

DBMS Component Modules

Figure below is a simplified form, the typical DBMS components.



The figure is divided into two parts.

1) The top part of the figure ***refers to the various users* of the database environment and their interfaces.**

1) The lower part shows ***the internals of the DBMS*** responsible for storage of data and processing of transactions.

Top Half: User Interaction & Compilation Layer

Types of Users: DBA, Casual users, application programmers, parametric users

1. DBA
   DDL Statements → DDL Compiler → Updates System Catalog/Data Dictionary
   Privileged Commands → Directly interact with the Runtime DB Processor
2. Casual Users

Issue Interactive Queries → Query Compiler → Query Optimizer

3. Application Programmers
   Write Application Programs → Use a Precompiler to extract DML commands
   Pass through DML Compiler → Query Optimizer
4. Parametric Users
   Use Compiled Transactions (pre-written programs)

1. DDL Compiler: Converts DDL to metadata stored in System Catalog.
2. Query Compiler + Optimizer: Parses and optimizes user queries.
3. DML Compiler: Processes DML commands into low-level instructions.
4. Compiled Transactions: Generated by compiling host-language programs (e.g., C/Java with embedded SQL).

Bottom Half: Query & Transaction Execution

1. Runtime Database Processor
   Executes commands, queries, and transactions
   Interacts with the System Catalog for metadata
   Sends results or further instructions to:
   Stored Data Manager (handles low-level data I/O)
   Concurrency Control / Backup / Recovery Subsystems
2. Stored Data Manager
   Performs actual data storage/retrieval from the Stored Database
3. System Catalog / Data Dictionary
   Holds metadata, schema definitions, user permissions, etc.
   Accessed during compilation and execution

The top part:

It shows interfaces for the DBA staff, casual users, application programmers and parametric users.

> DBA works for commands related to database administration.

> casual users work with interactive interfaces to formulate queries

> application programmers who create programs using some host programming languages, and

> parametric users who do data entry work by supplying parameters to predefined transactions.

> **The DDL compiler:** It processes schema definitions, stores meta-data in the DBMS catalog such as the names and sizes of files, names and data types of data items, storage details of each file etc.

> **The Query Compiler:** Whenever a query generated, it is validated for correctness of the query syntax. The query compiler compiles them into an internal form. It rearranges and reordering of operations, elimination of redundancies during the query execution.

> **The precompiler:** It extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access. The rest of the program is sent to the host language compiler. The object codes for the DML commands and the rest of the program are linked, forming a canned transaction. These canned transactions are executed repeatedly by parametric users, who simply supply the parameters to the transactions.
An example is a bank withdrawal transaction where the account number and the amount may be supplied as parameters.

The lower part:

> **The runtime database processor:**
It executes
(1) the privileged commands,
(2) the executable query plans, and
(3) the canned transactions with runtime parameters.

It handles buffers management in the main memory.
In this figure it is shown that the concurrency control and backup and recovery systems separately as module.
They are integrated into the working of the runtime database processor.

Classification of Database Management Systems:

Several criteria are normally used to classify DBMS. They are:

1) Based on data model
2) Based on no. of users
3) Based on no. of sites

1) Based on data model:

==========

The data model expresses data and relationships.
The data model defines how data is logically structured, how it is related, and how it can be manipulated.
They divided as:

1.1 Object-Based Logical Models:
1.2. Record-Based Logical Models
1.3. Physical Data Models

1.1) Object-Based Logical Models:

These models focus on **describing the data and its relationships in a form that is close to how users understand the real world.**

ER MODEL: The most widely used object-based model is the **Entity-Relationship (ER) model, which represents data using entities , attributes , and relationships.**

It is a diagram which represents a set of entities and the relationships exist among them.

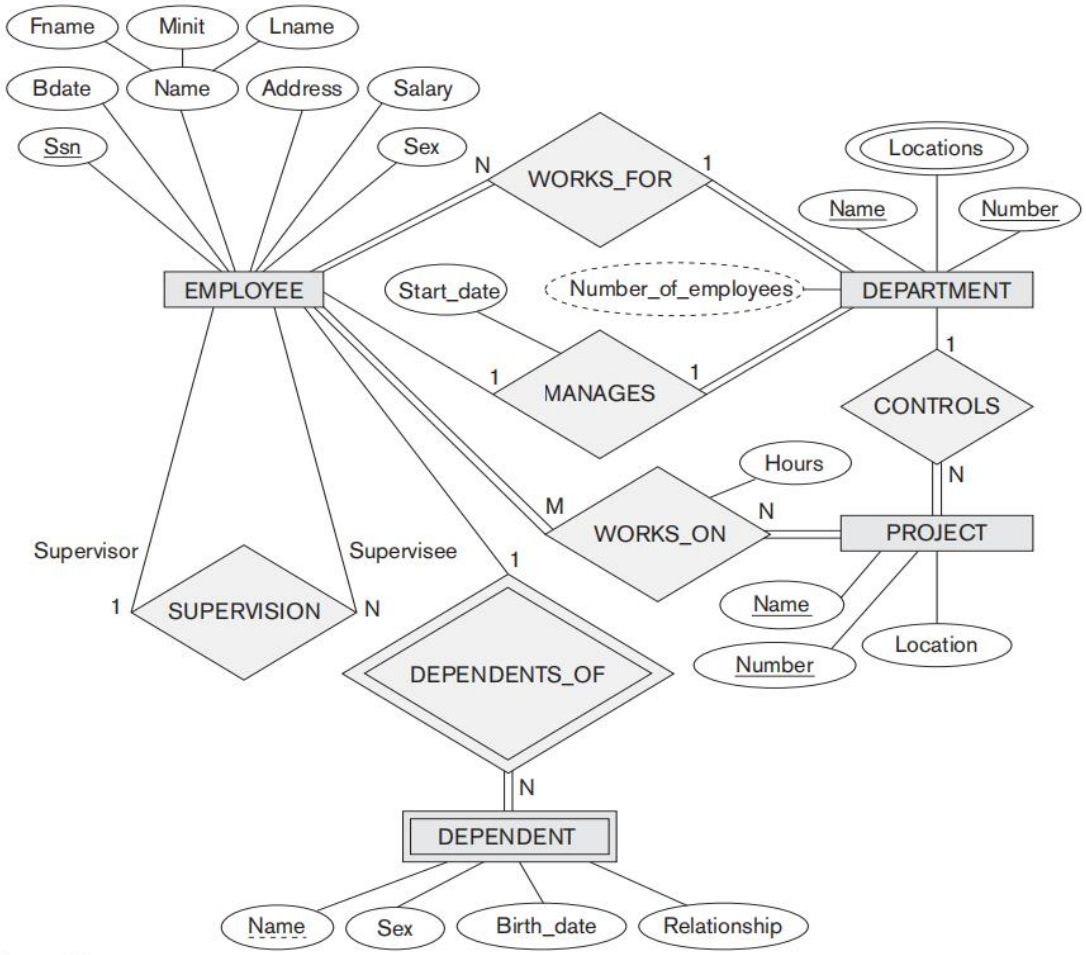
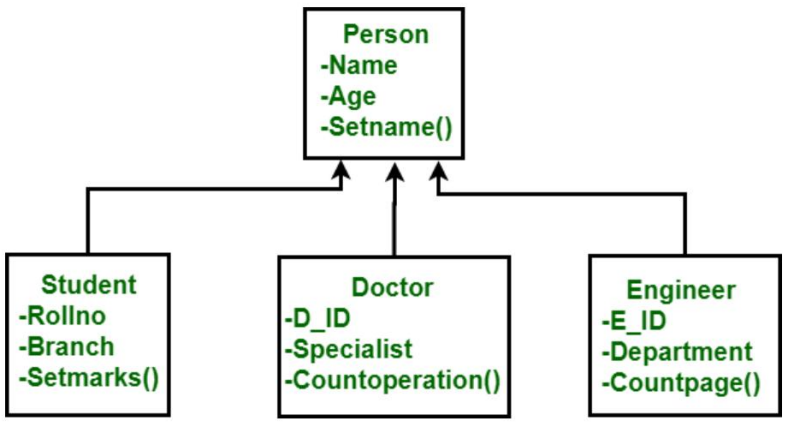


Figure 7.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 7.14.

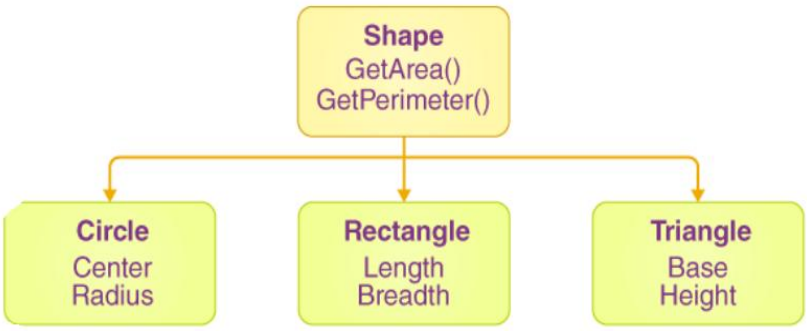
Object Oriented Model: It includes the object-oriented programming features included such as encapsulation, inheritance, and polymorphism, enabling the database to store complex data types and methods.

Example1:

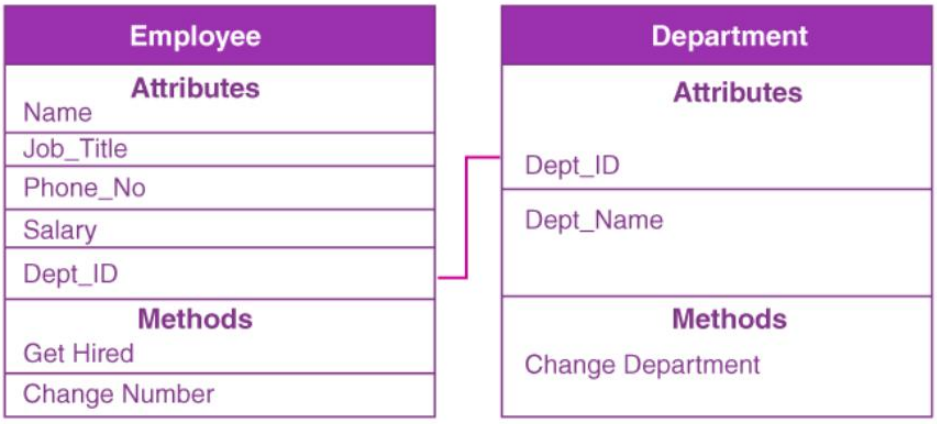


Basic Object Oriented Data Model

Example2:



Example3:



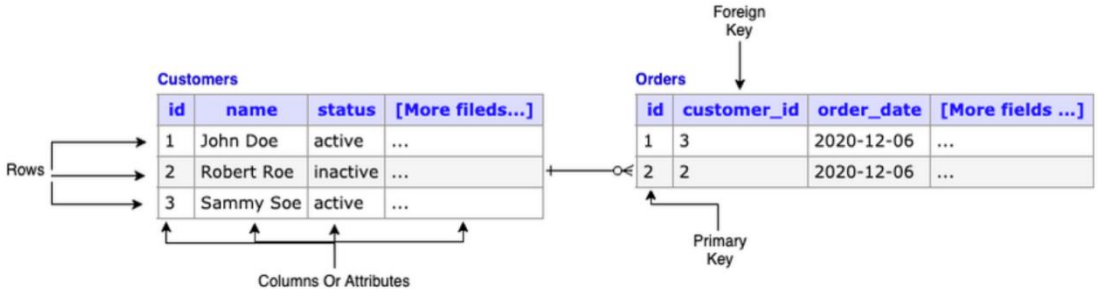
Object Oriented Model

1.2.) Record-Based Logical Models:

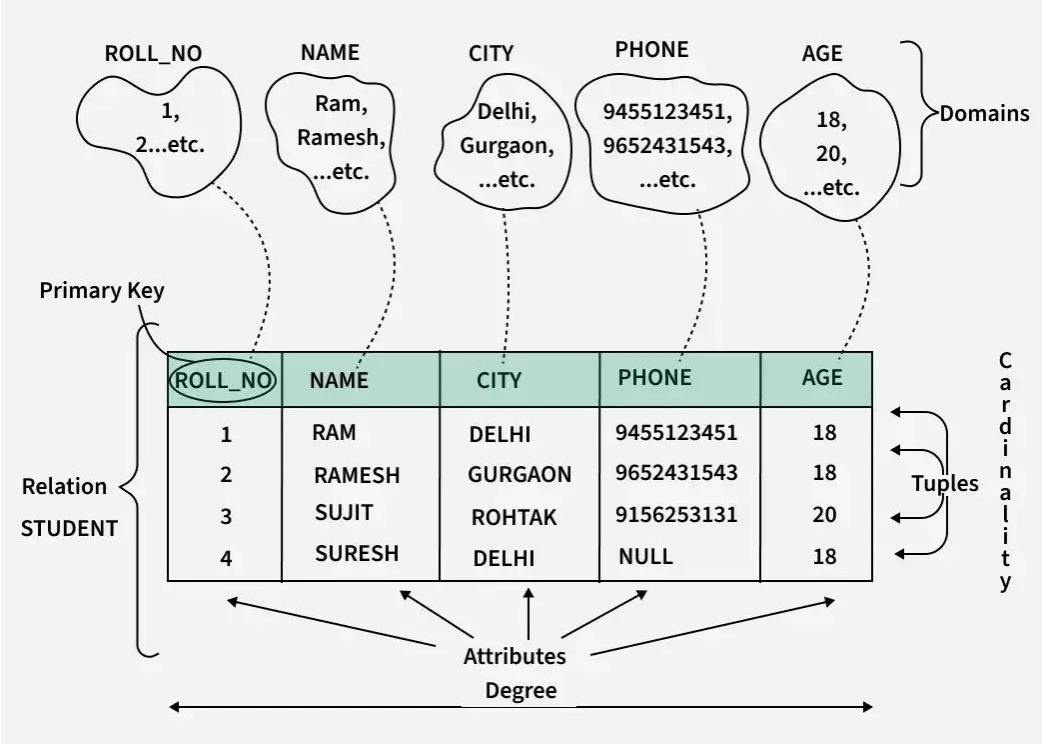
Record-based logical models provide a lower-level representation of data, **focusing more on how data is logically structured** and accessed within the DBMS.

The relational model: It is the most popular record-based model, **representing data in the form of relations or tables, where each row is a tuple and each column is an attribute.** This model is highly flexible and supports querying through relational algebra and SQL.

Example1:



Example2:

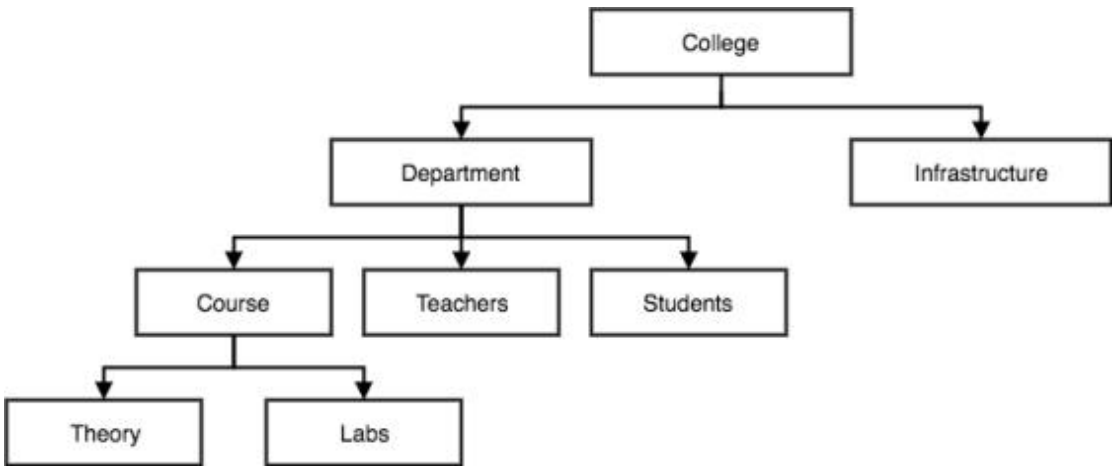


Relation: It is the description of data in terms of table.

Hierarchical Model:

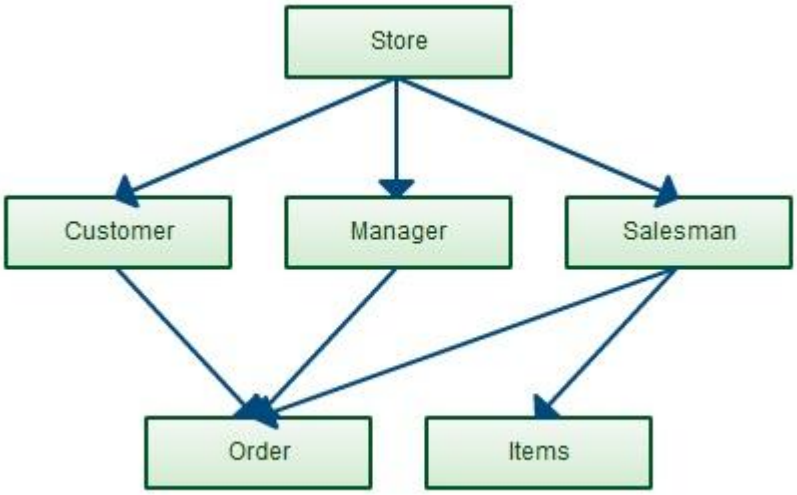
It organizes data in a tree-like structure with records having a parent-child relationship; it is efficient for representing data with a strict 1:N hierarchy.

When the entities and their relationships are in hierarchical order like a tree then it is called hierarchical model.



Network Model:

It arranges data using a graph structure, allowing complex many-to-many relationships through the use of pointers and sets. **When relationship among different entities are represented like a network, then it is called network model.**



1.3.) Physical Data Models:

Physical data models **describe the lowest level of data abstraction, focusing on how the data is actually stored in the storage medium such as disks**. These models deal with internal storage details like file organization, indexing techniques, record placement, page layout, and access paths.

The goal of physical models is to optimize performance, storage efficiency, and data retrieval speed. Designers use physical data models to determine how physically data to be stored like sequential files, B+ trees, hashing, and clustering techniques.

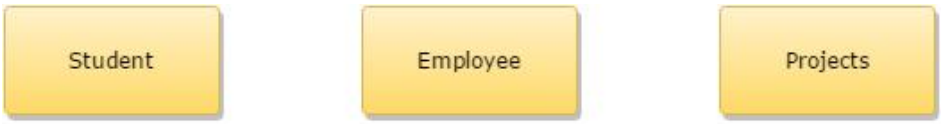
ER MODEL

ER-Diagram is a pictorial representation of data that describes how data is communicated and related to each other.

Any object, such as entities, attributes of an entity, sets of relationship, and other attributes of relationship, can be characterized with the help of the ER diagram.

Entity: Any real world object is called entity/ Each entity will have a set of properties. They are represented using the rectangle-shaped box.

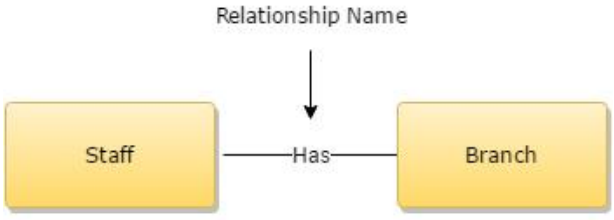
Ex: student, employee, bus, car, cow, goat, every person etc.



Entities

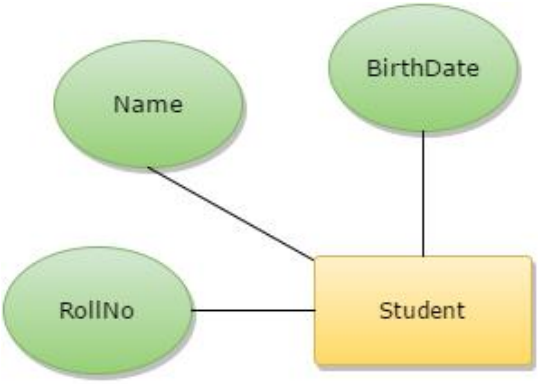
Entity Set: A collection of similar entities called an entity set.

Relationship: It is a connection or association between two or more entities.



Branch has staff

Attributes: Each property of an entity is called an attribute. It can be written using ellipse.



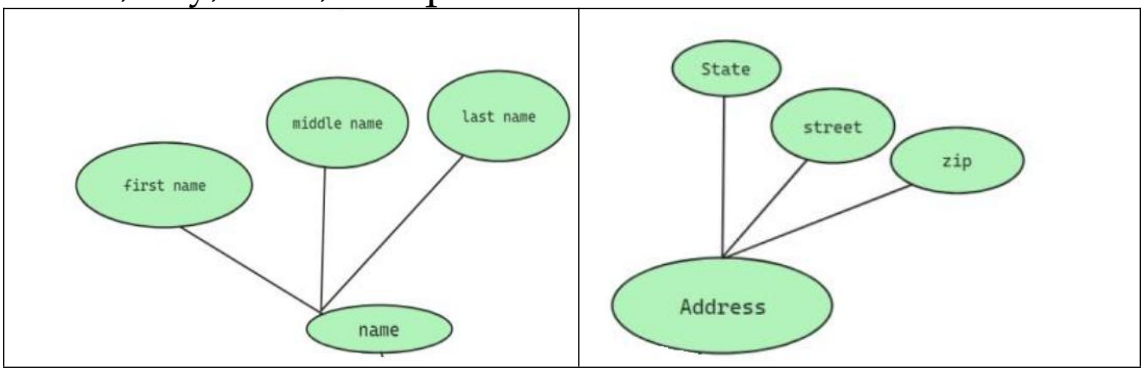
Attributes

Types of attributes:

They are:
1) Simple Attribute (Atomic Attribute)
2) Composite Attribute
3) Single Valued Attribute
4) Multivalued Attribute
5) NULL Attribute
6) Derived Attribute
7) Key Attribute

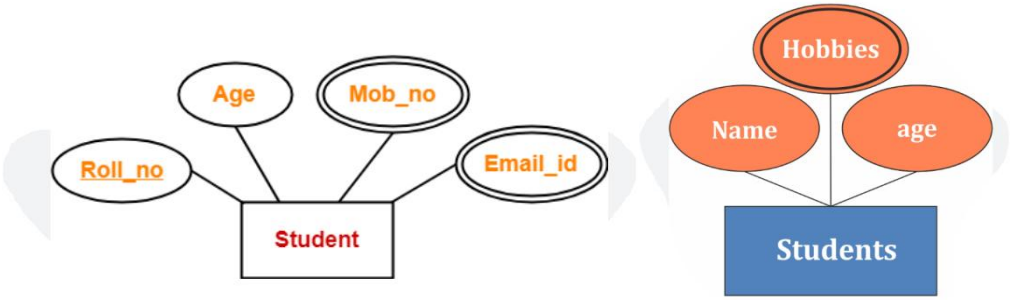
Simple Attributes: These attributes cannot be broken down further into smaller components. They hold a single atomic value.
Example: A student's roll number, gender, marks,

Composite Attributes: In contrast to simple attributes, composite attributes can be divided into smaller, meaningful parts or sub-attributes.
Example: An address attribute can be broken down into street, city, state, and pin code.

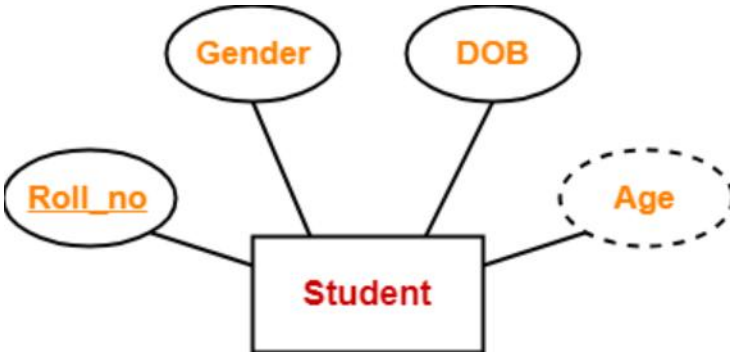


Single-valued Attributes: These attributes hold only one value for a particular entity instance.
Example: An employee's date of birth or a student's gender.

Multi-valued Attributes: Multi-valued attributes can store multiple values for a single entity instance.
Example: A student might have multiple phone numbers or a person might have several email addresses.



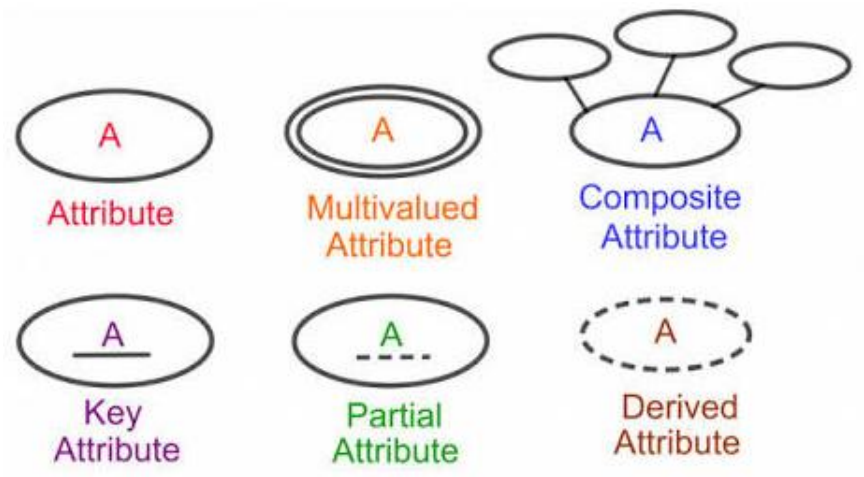
Derived Attributes: Derived attributes are not directly stored in the database but are computed or derived from other stored attributes.
Example: An employee's age can be calculated from their date of birth.



Key Attributes: These are special attributes that uniquely identify each record (or entity instance) within a table (or entity set).
Example: A student's roll number or an employee's ID, product number, pan id, adhaar id, bill no etc.

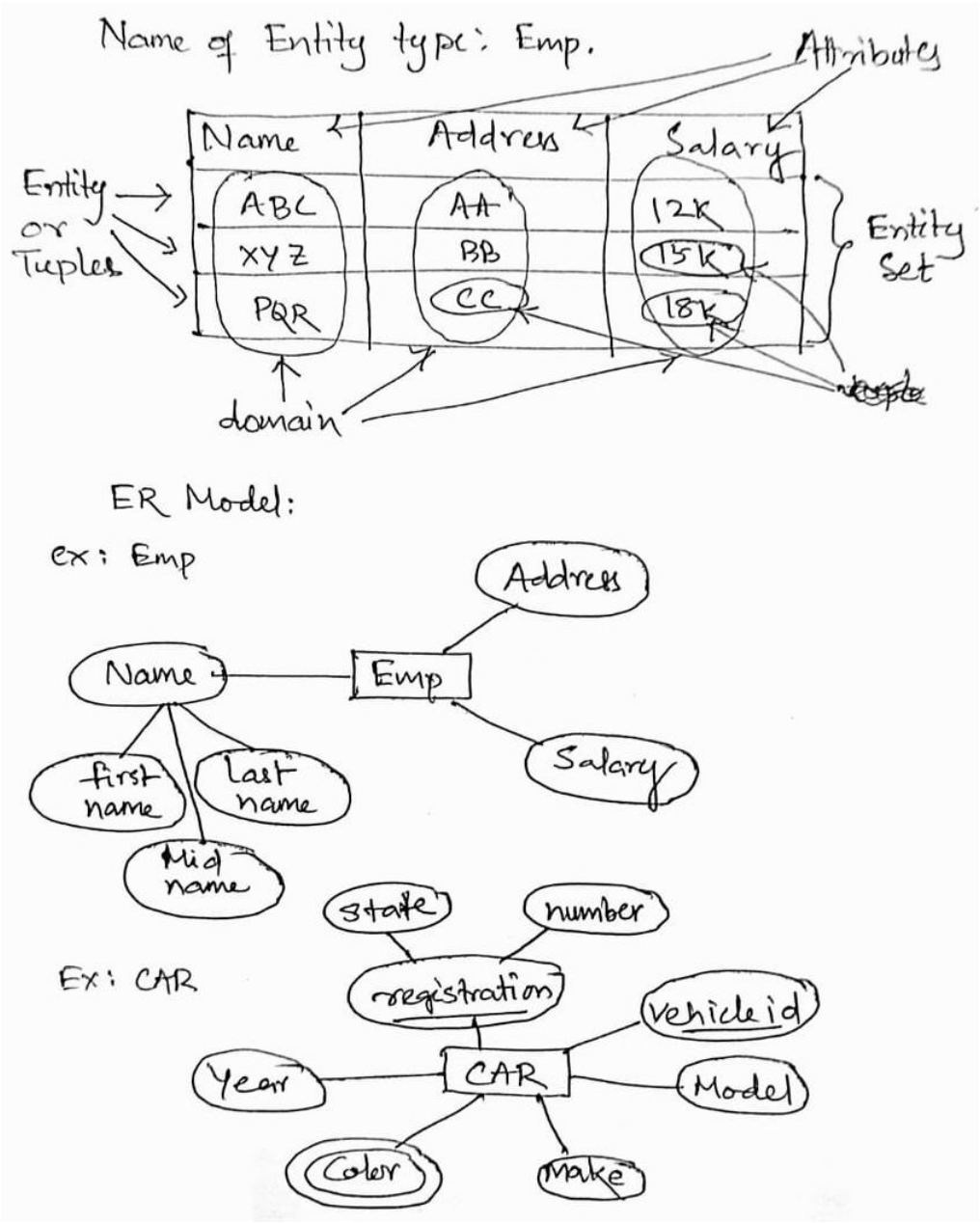
Null Attribute: This attribute can have a NULL value, indicating that the value is either unknown or not applicable for a particular entity.
Example: An employee's middle name, if they don't have one, About me, remarks etc.

General symbols for attributes:

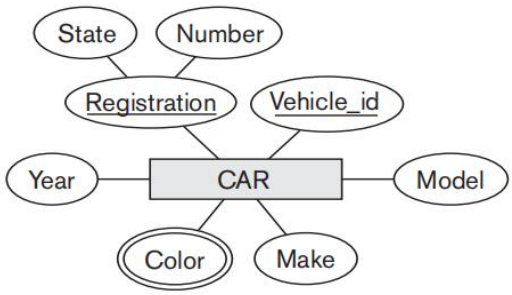


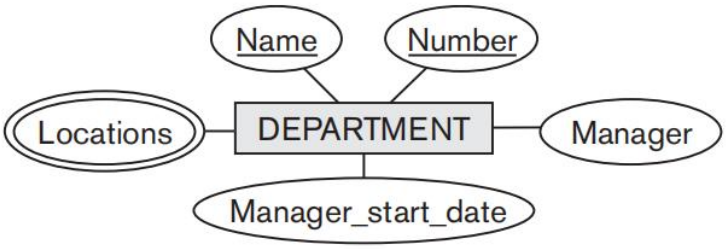
ENTITY RELATIONSHIP MODEL:

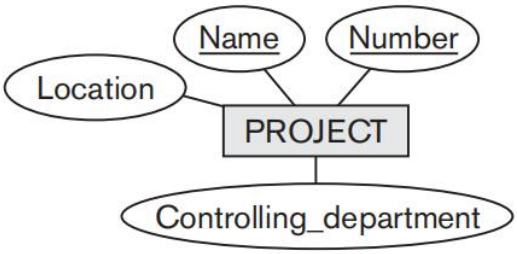
EX: EMPLOYEE ENTITIES:

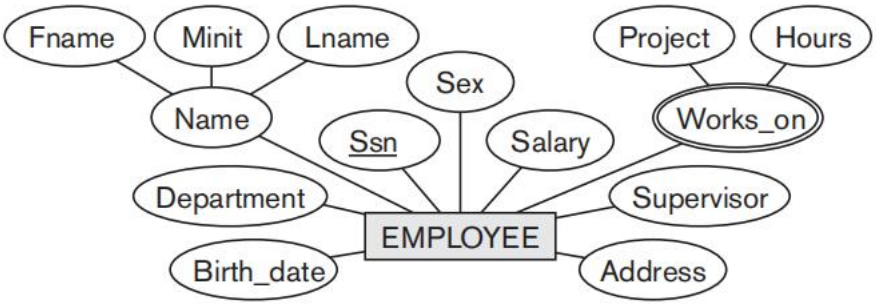

Name of Entity type: Emp.
Attributes
Name
Address
Salary
Entity or Tuples
ABC
XYZ
PQR
AA
BB
CC
12K
15K
18K
Entity Set
domain
ER Model:
ex: Emp
Address
Name
Emp
Salary
first name
Last name
Mid name
state
number
Ex: CAR
registration
Vehicle id
Year
CAR
Model
Color
Make

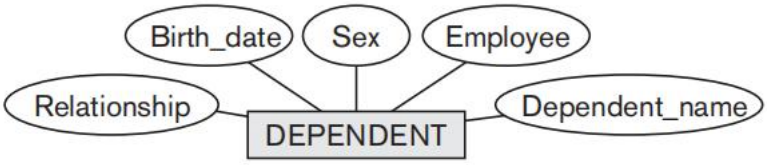
Ex: CAR


State
Number
Registration
Vehicle_id
Year
CAR
Model
Color
Make

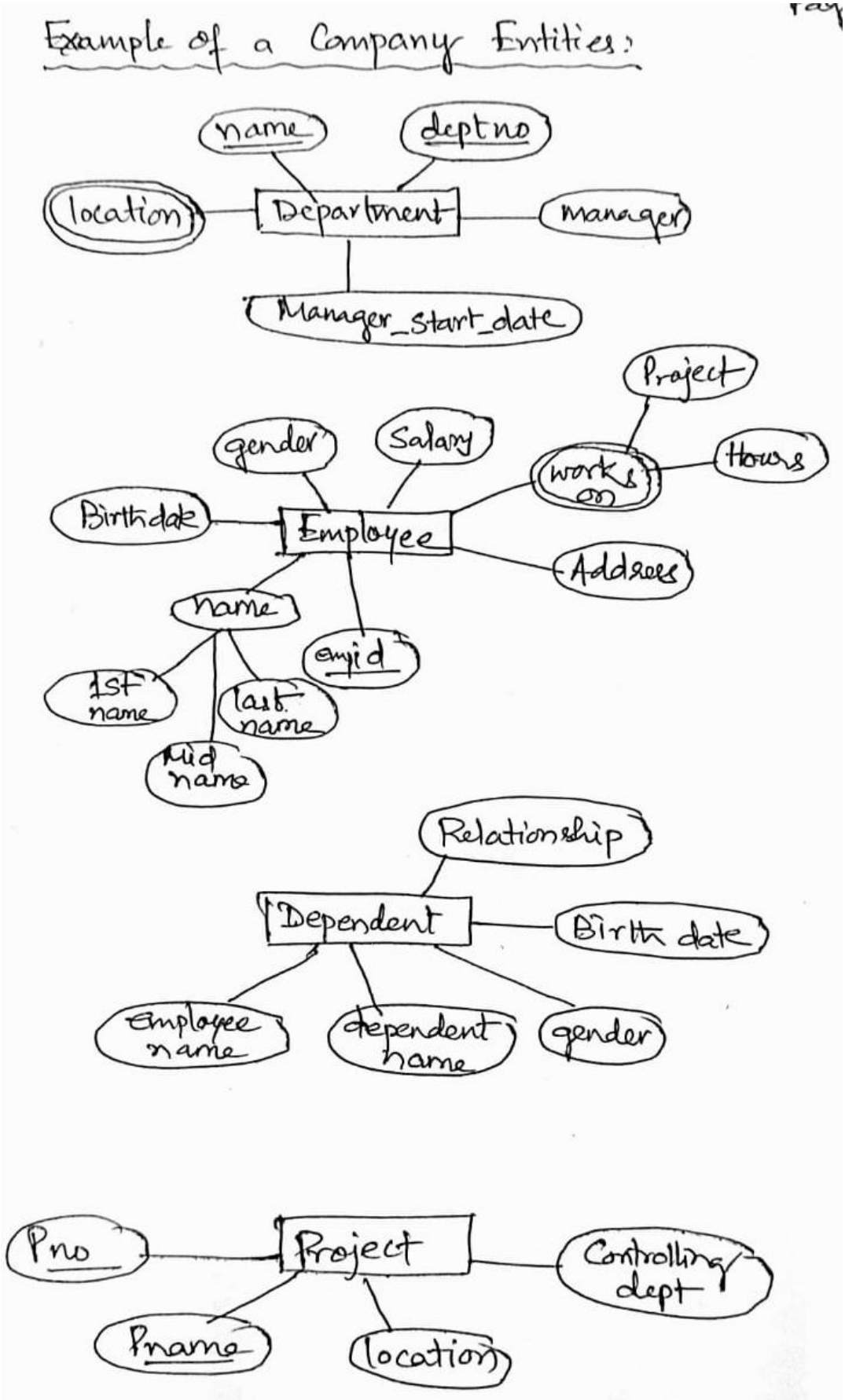








EXAMPLE: COMPANY ENTITIES



CASE STUDIES ON ER DIAGRAM:

1) Design a case study for identifying the Entities, Attributes, Relationships for preparing ER Diagram of a **University Database System.**

2) Design a case study for identifying the Entities, Attributes, Relationships for preparing ER Diagram of a **Banking System.**
3) Design a case study for identifying the Entities, Attributes, Relationships for preparing ER Diagram of a **Super Market Management System.**

DATABASE CONSTRAINTS / STURCTURAL CONSTRAINTS:

A database must define certain constraints for the contents of database. They are:
1) Mapping Cardinalities
2) Participation constraints

Mapping Cardinalities:
==========
It defines the type of relationships exist between two or more participating entities.
Mapping cardinalities mostly useful for describing the binary relationship sets. (i.e: relationship between two entities)

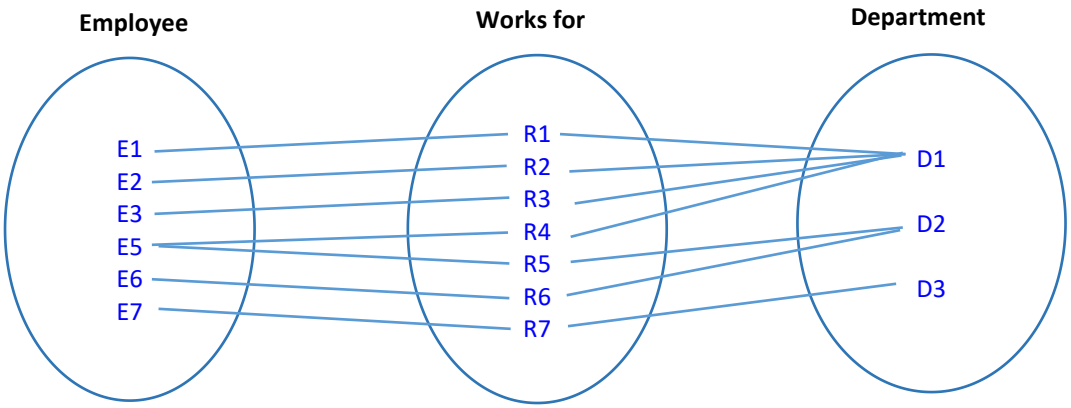
Participation Constraints:
==========
It is all about how much participating by an entity for a relationship.
The participation constraints explains about the type of relationship such as :
Total participation, partial participation.

When every entity of an entity set having relationship, then it is called total participation. Otherwise it is called partial participation.

Ex: Total participation: Every customer having relationship with Account entity.
Ex: Partial participation: Few customer having relationship with loan entity.

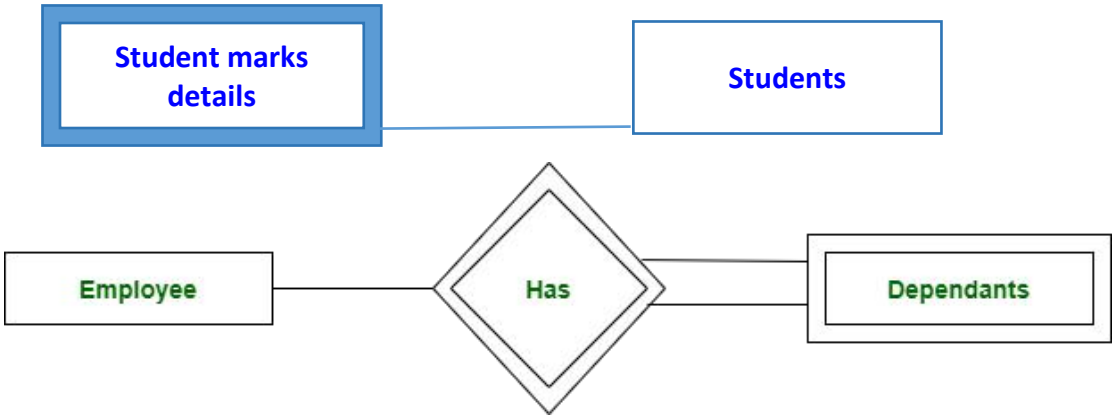
Relationship Set:
==========

It is a set of relationship instances r_i :



Strong Entity:
A Strong Entity is a type of entity that has a key Attribute that can uniquely identify each instance of the entity. A Strong Entity does not depend on any other Entity in the Schema for its identification.

Weak Entity :
A Weak Entity cannot be uniquely identified by its own attributes alone. It depends on a strong entity to be identified.



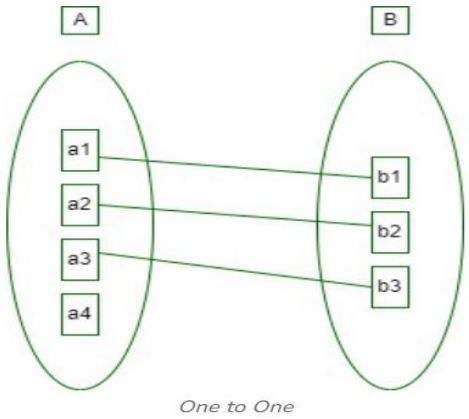
Mapping Cardinalities:

==========

For a binary relationship between entity sets A and B, the mapping cardinality must be one of the following types:

1. One to one mapping (1 : 1)
2. One to many mapping (1 : N)
3. Many to one mapping (N : 1)
4. Many to many mapping (M : N)

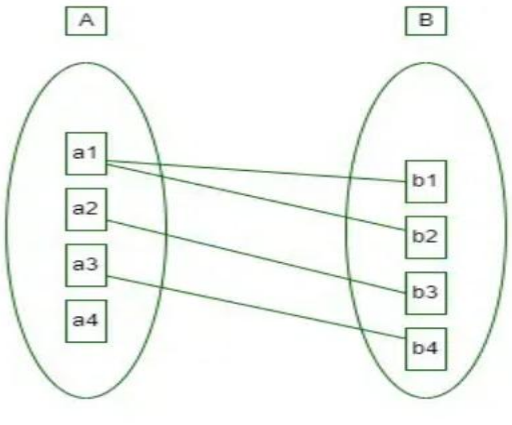
One to one mapping: (1:1 binary relationship)
==========
When an entity in a relation A is associated with atmost one entity in B and similarly an entity in a relation B is associated with atmost one entity in a relation A then it is One to One mapping.



One to One

Examples:
1) One HOD having "managing" relationship with one department.
2) One professional having "issue" relationship with pan card issuers.
3) In a particular hospital, the surgeon department has one head of department. They both serve one-to-one relationships.

One to many mapping (1 : N)
==========
An entity in A is associated with any number of entities in B.
But an entity in B can be connected to at most one entity in A.

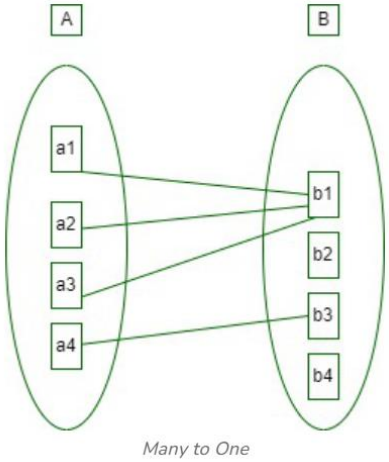


One to Many

Examples:
1) One to many phone calling using conference calling

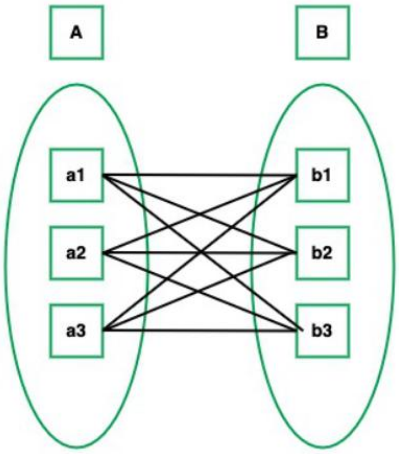
2) One department admits many students
3) One faculty interacts with many students in class.
4) In a particular hospital, the surgeon department has multiple doctors. They serve one-to-many relationships.

Many to one mapping (N : 1)
==========
An entity in A is connected to at most one entity in B. Or we can say a unit or item in B can be associated with any number of entities or items in A.



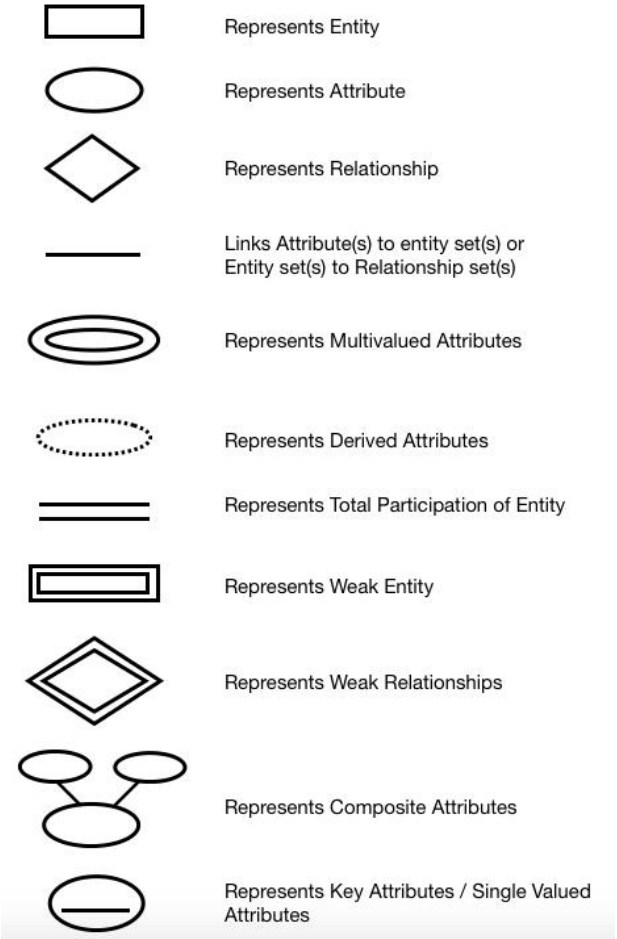
Example:
1) Many audience listen to one seminar
2) Many students do one course

Many to Many mapping: (M: N)
==========
An entity in A is associated with any number of entities in B, and an entity in B is associated with any number of entities in A.



Example1: Relationship **joined** between **Entity: Students** and **Entity: Courses**
Example2: Relationship **work on** between **Entity: Employee** and **Entity: projects**. They serve many-to-many relationships.

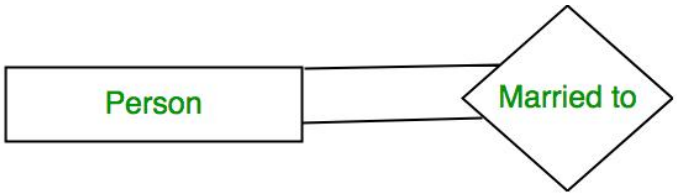
Notations of ER Diagram:



Degree of a Relationship Set

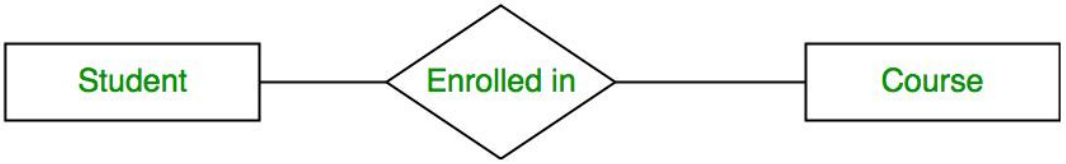
The number of different entity sets participating in a relationship set is called the degree of a relationship set.

1. Unary Relationship: When there is only ONE entity set participating in a relation, the relationship is called a unary relationship. For example, one person is married to only one person.



Unary Relationship

2. Binary Relationship: When there are TWO entities set participating in a relationship, the relationship is called a binary relationship. For example, a Student is enrolled in a Course.



Binary Relationship

3. Ternary Relationship: When there are three entity sets participating in a relationship, the relationship is called a ternary relationship.

4. N-ary Relationship: When there are n entities set participating in a relationship, the relationship is called an n-ary relationship.

Designing ER Model:

The steps for generating the ER Model are:

1) **Identify the Entities**
2) **Find the relationships among different entities**
3) **Identify the attributes of entities**
4) **Identify the key attributes of entities**
5) **Design the ER Model.**

Example1:
CASE STUDY of Company:
==========

A company having different departments with employees and undergoes different projects.

1) **Identifying Entities:**

Employees, Departments, Projects, Dependents.

2) Identifying the Attributes:

==========

Entity	Attributes
Employee	**Empno**, Name, DateofJoin, Address, Salary, Gender
Department	**Deptno,DeptName**, Location, Deptsummary, no. of employees
Project	**ProjectNo**, Project_name, Location
Dependents	**Name**, Gender, dateofbirth, relationship

3) Identifying Relationships:

2.1) Relationship: Works for

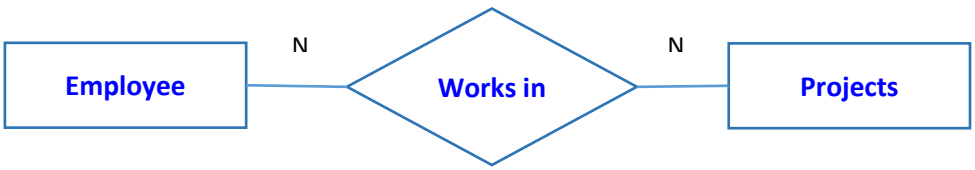
Entity: Employee and Department

2.2) Relationship: Manages

Entity: Employee and Department

2.3) Relationship: Works in

Entity: Employee and Projects



2.4) Relationship: Supervises

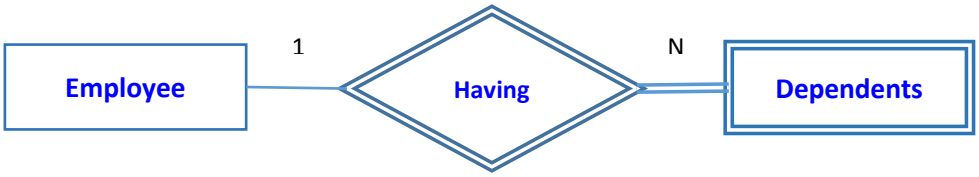
Entity: Employee and Employee

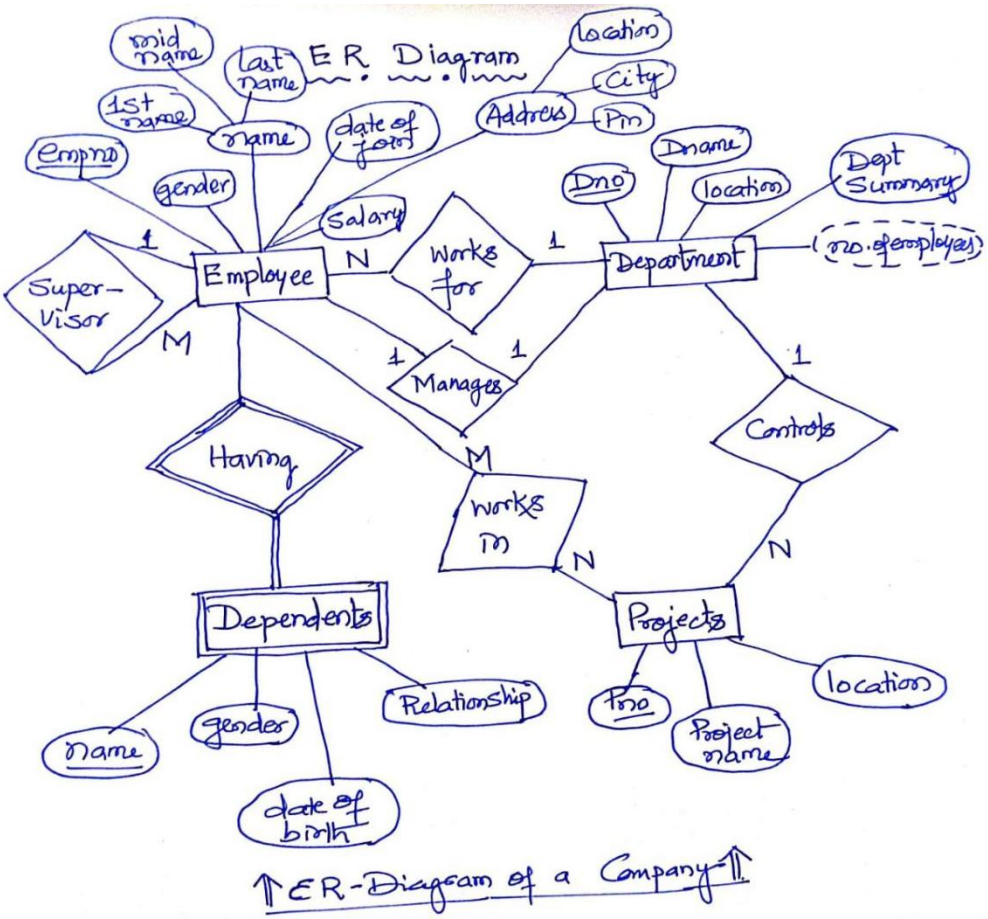
2.5) Relationship: Controls

Entity: Department and Projects

2.6) Relationship: Having

Entity: Employee and Dependents





↑ER-Diagram of a Company↑

Question) Design an ER diagram for representing University database system and explain suitably.

Identify the Entities:
1) Students
2) Faculties
3) Courses
4) Departments or Branch
5) Examinations
6) Library

Identify attributes:
1) Students (**Rollno**, Name, gender, date of birth, branchID, course, address, phone, email, adhaar no)
2) Faculties (**EmpID**, Name, gender, designation, branchID, Qualification, phone, email)
3) Courses(**CourseID**, Cname, Credits, semester, branchID)
4) Branches(**BranchID**, Bname, location, Head)
5) Examinations(**ExamID**, CourseID, Exam type, date, time, roomno)
6) Library(**BookID**, Bname, author, publisher, quantity)

Identifying relationships:
1) Branches - Offers - Courses
One ----------> many
2) Courses - opted by Students
One ----------> many
3) Faculties - working for - Branches
Many ----------> one
4) Faculties - headed by Branches
One ----------> one
5) Faculties - teaching - Students
Many ----------> many
6) Students - attending - examinations

Many ----------> many

7) Students - issues - Books

One ----------> many

8) Faculties - issues - Books

One ----------> many

Example 2: Case Study of UNIVERSITY

Identifying Entities: Depts, Courses, Instructors, Students

Identifying Relationships:

① Relationship: Offered By

Entities: Depts and Courses

② Relationship: enrolled.

Entities: Courses and Students

③ Relationship: works_for

Entities: instructors and Depts

④ Relationship: Headed By

Entities: Depts and Instructors

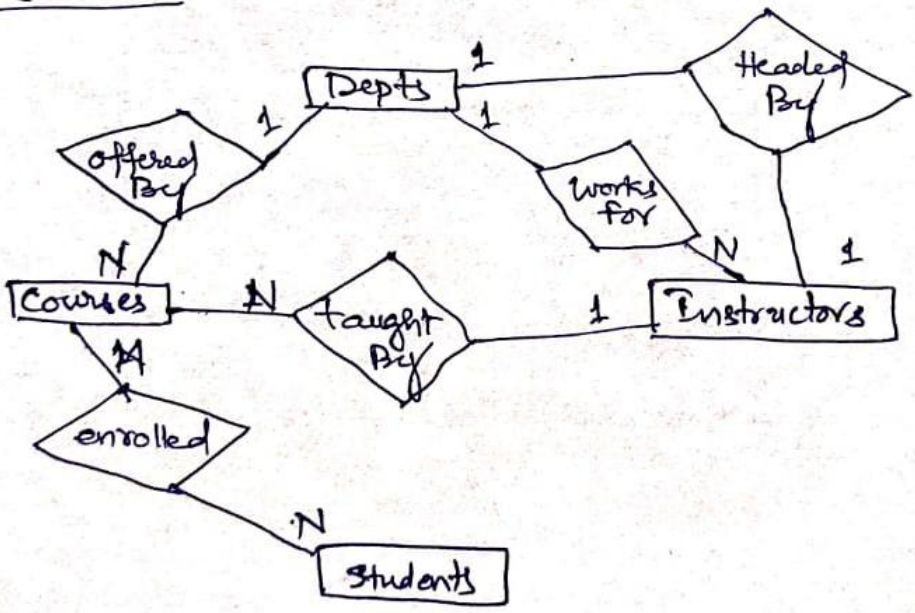
(5) Relationship: taught by

Entities: Courses and Instructors

Identifying the Attributes:

Entity	Attributes
Depts	Deptno, Deptname, location, Summary
Courses	courseid, name, fees, duration
instructors	ins_id, name, address, phone, doj
students	studid, name, dateofbirth, doj, address

ER Model:



Example 3: Case Study of a Banking System

Assume the each branch maintains loans and accounts of customers.

Identifying Entities: Bank, branch, Accounts, loans, Customer

~~Ende~~ Identifying Relationships:

① Relationship: branches
Entities: Bank and Branch

② Relationship: offers
entities: Branch and loans

③ Relationship: maintains
entities: Branch and Accounts

④ Relationship: Avails
entities: Loans and customers

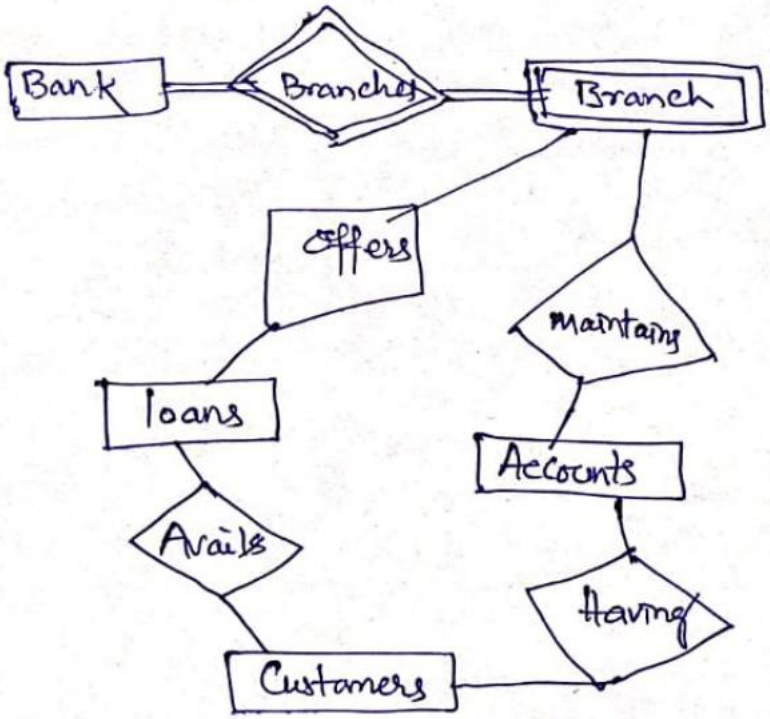
(5) Relationship: having

Entities: Customers and Accounts

Identifying the attributes:

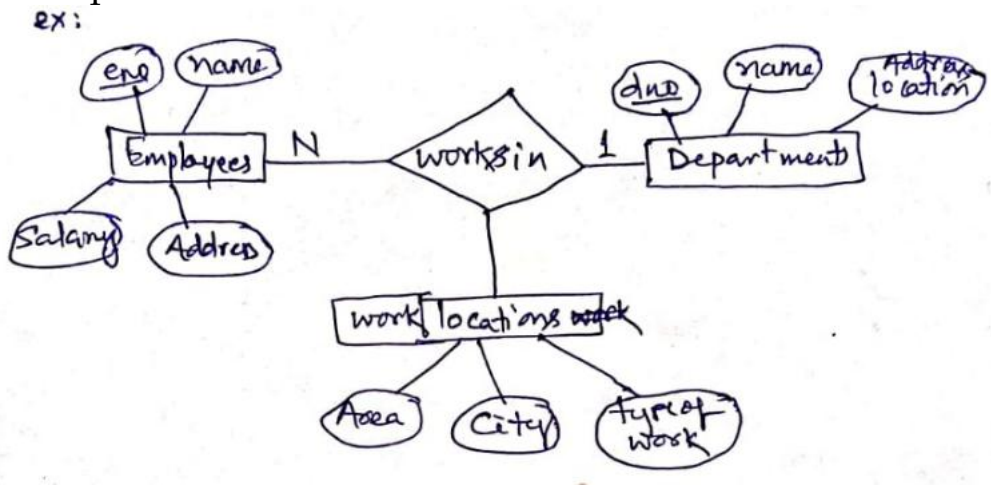
	Entity		Attributes
(1)	Bank	——	Bankcode, name, address
(2)	Branches	——	Branchno, address
(3)	Customers	——	custno, name, address, dob, age, phone
(4)	loans	——	loanno, loantype, Amount, mode
(5)	Accounts	——	Accno, type, balance

E R Model:

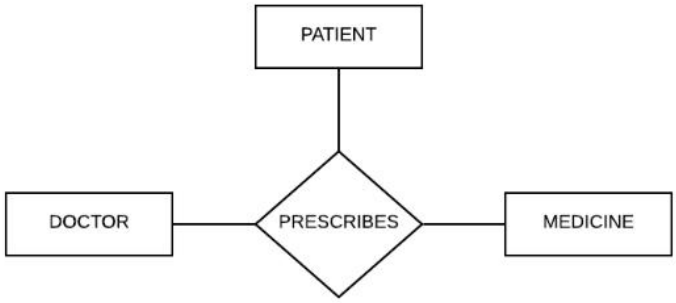


Extended Features of ER Models:

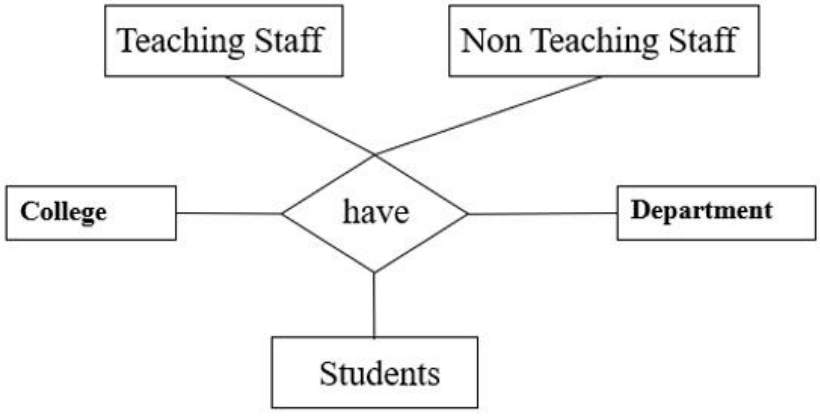
Ternary Relationships:
It is an association between three entity sets.
Example 1:



Example 2:

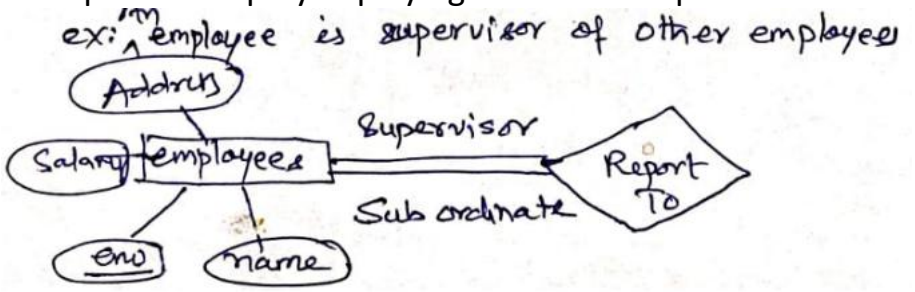


N-Ary Relationships:
It is an association between N no. of entity sets.
Example 1:



Report - To Relationship:
In an entity seat, if relationship exist between themselves then it is called report-to relationship.

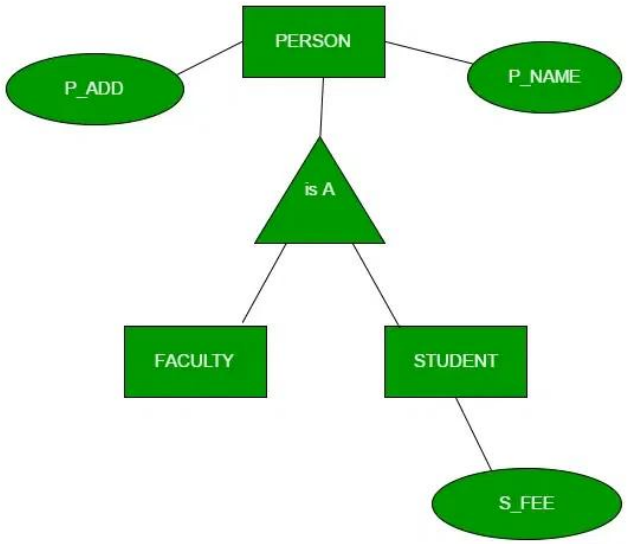
Example: An employee playing the role of supervisor of other entities.



Generalization:

Generalization is the process of identifying some common characteristics and generalize them into a single class. It is a bottom-up approach. It is a reverse of specialization.

Example:



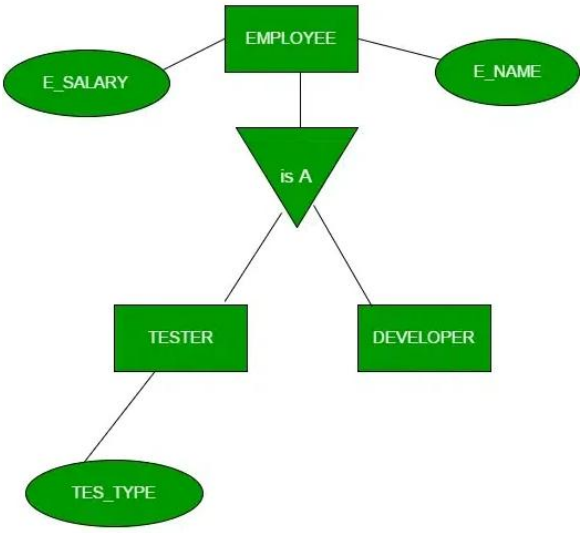
Example: STUDENT and FACULTY can be generalized to a higher-level entity called PERSON as shown in diagram above.

In this case, common attributes like P_NAME and P_ADD become part of a higher entity (PERSON) and specialized attributes like S_FEE become part of a specialized entity (STUDENT).

Specialization:

In specialization, an entity is divided into sub-entities based on its characteristics. It is a top-down approach where the higher-level entity is specialized into two or more lower-level entities.

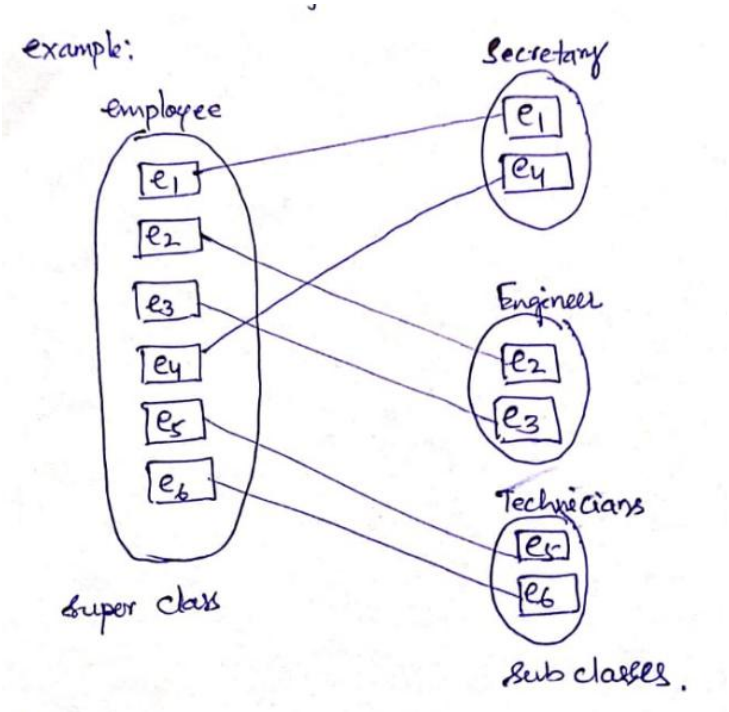
Example 1:



Example: an EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER, etc. as shown in figure above.

In this case, common attributes like E_NAME, E_SAL, etc. become part of a higher entity (EMPLOYEE) and specialized attributes like TES_TYPE become part of a specialized entity (TESTER).

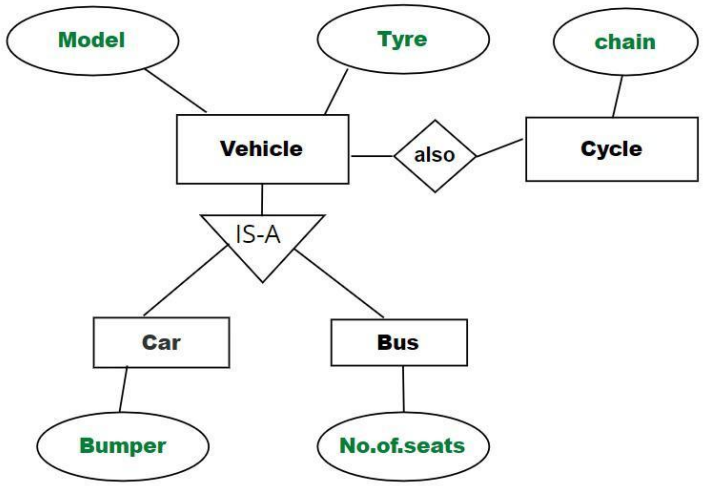
Example 2:



Inheritance

It is an important feature of generalization and specialization. In specialization, a higher-level entity is divided into lower-level sub-entities that inherit its attributes. In generalization, similar lower-level entities are combined into a higher-level entity that holds common attributes.

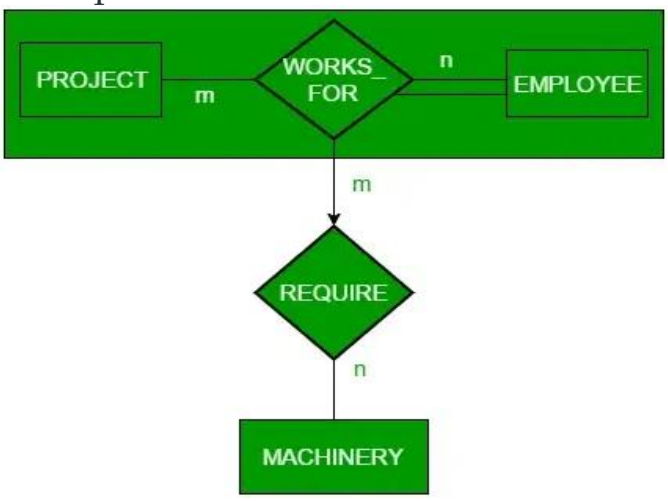
Example 1:



Aggregation:

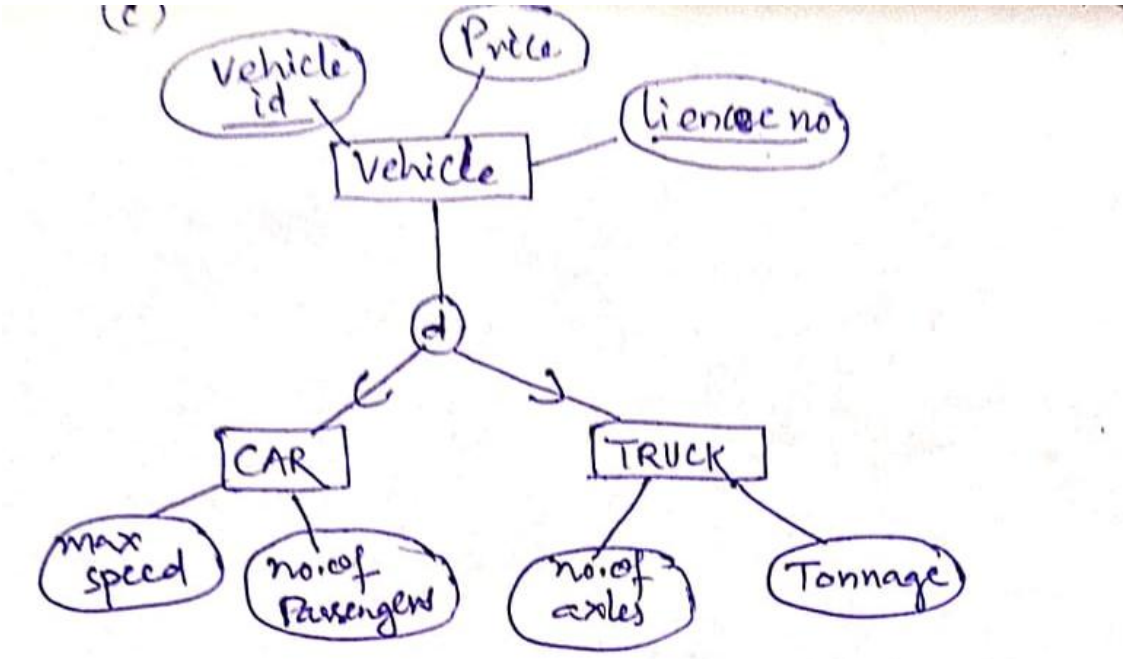
Aggregation refers to the process of combining multiple entities or relationships into a single, higher-level entity. It's a way to simplify complex relationships and represent them as a single, more meaningful entity

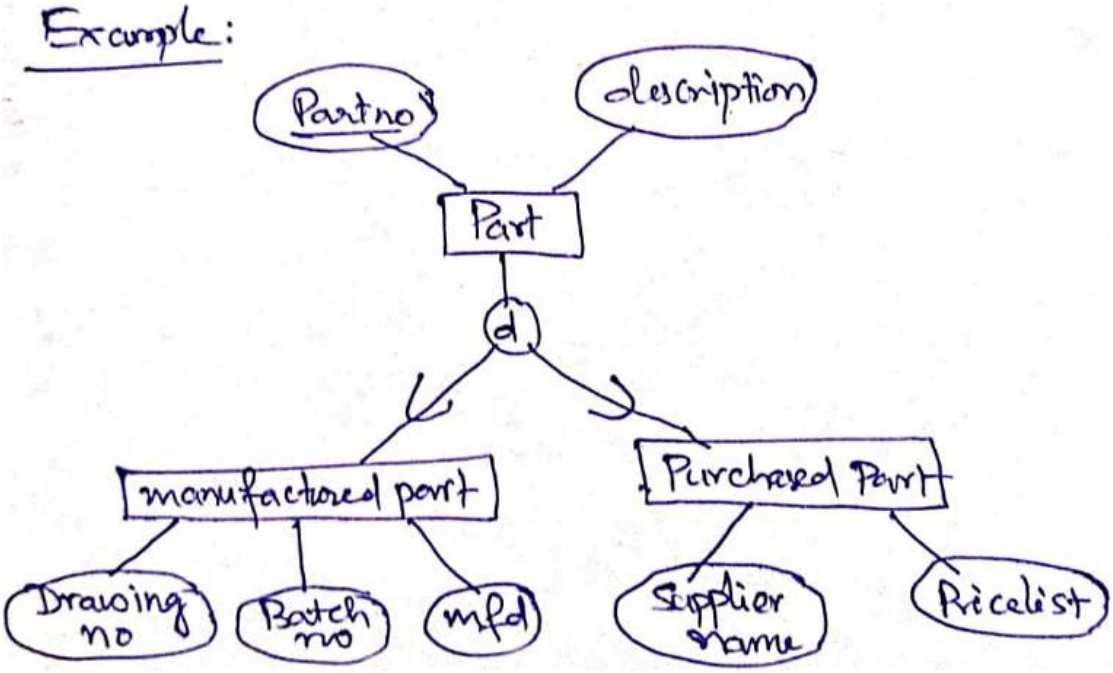
Example 1:

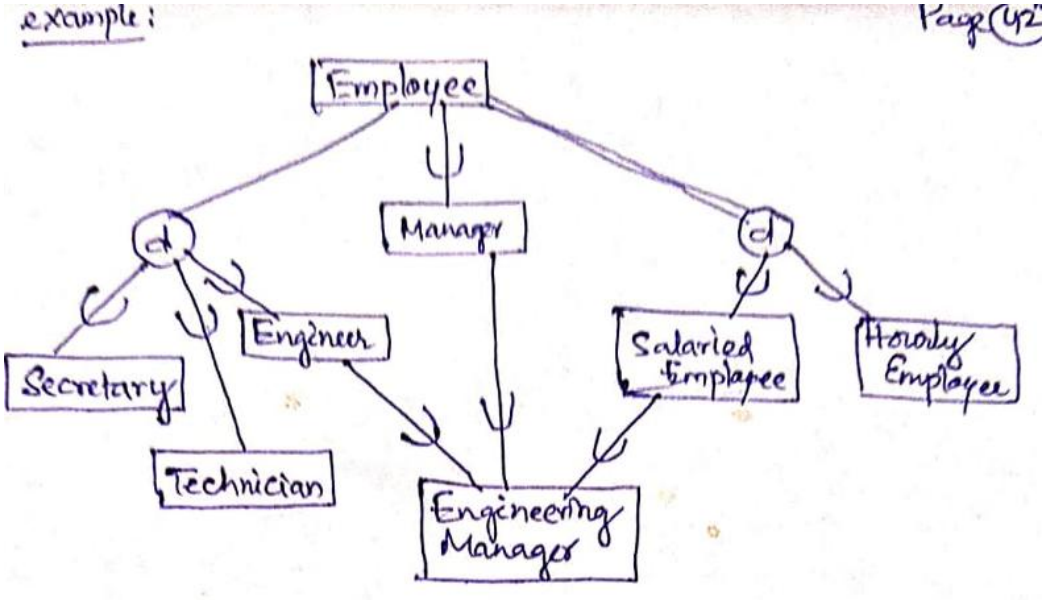


Example: an Employee working on a project may require some machinery. So, REQUIRE relationship is needed between the relationship WORKS_FOR and entity MACHINERY.

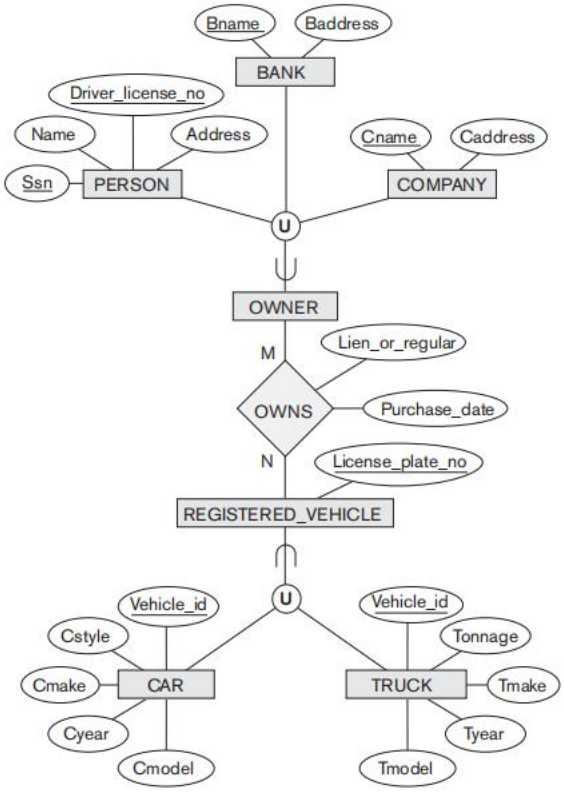
Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into a single entity and relationship REQUIRE is created.







Example:



Here are some common ER model examples:
==========

1) University Database:
Entities:
Student, Course, Professor, Department.

Attributes:
Student: Student ID (PK), Name, Address, Major.

Course: Course ID (PK), Title, Credits.
Professor: Professor ID (PK), Name, Specialization.
Department: Department ID (PK), Name, Location.

Relationships:
Student enrolls in Course (Many-to-Many).
Professor teaches Course (One-to-Many).
Professor belongs to Department (One-to-Many).

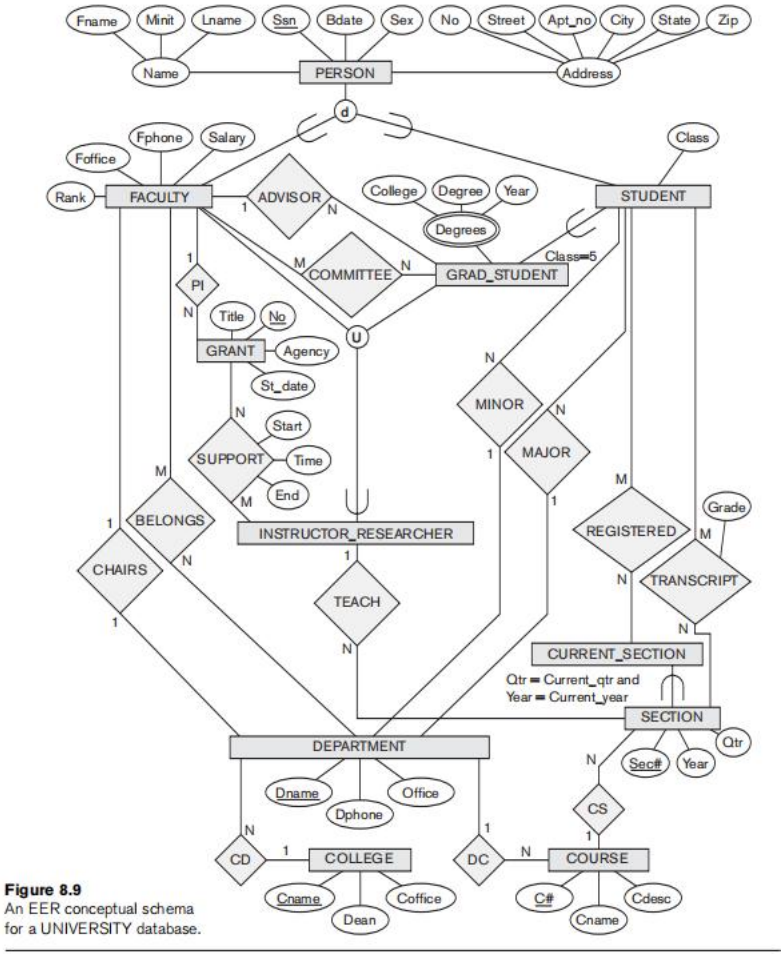


Figure 8.9
An EER conceptual schema for a UNIVERSITY database.

2) Hospital Management System:
==========

Entities:
Patient, Doctor, Appointment, Treatment, Ward.
Attributes:
Patient: Patient ID (PK), Name, Age, Address, Phone.
Doctor: Doctor ID (PK), Name, Specialization.
Appointment: Appointment ID (PK), Date, Time.
Treatment: Treatment ID (PK), Description, Cost.
Ward: Ward ID (PK), Name, Capacity.

Relationships:
Patient has Appointment (One-to-Many).

Doctor conducts Appointment (One-to-Many).
Patient receives Treatment (One-to-Many).
Patient is assigned to Ward (Many-to-One).

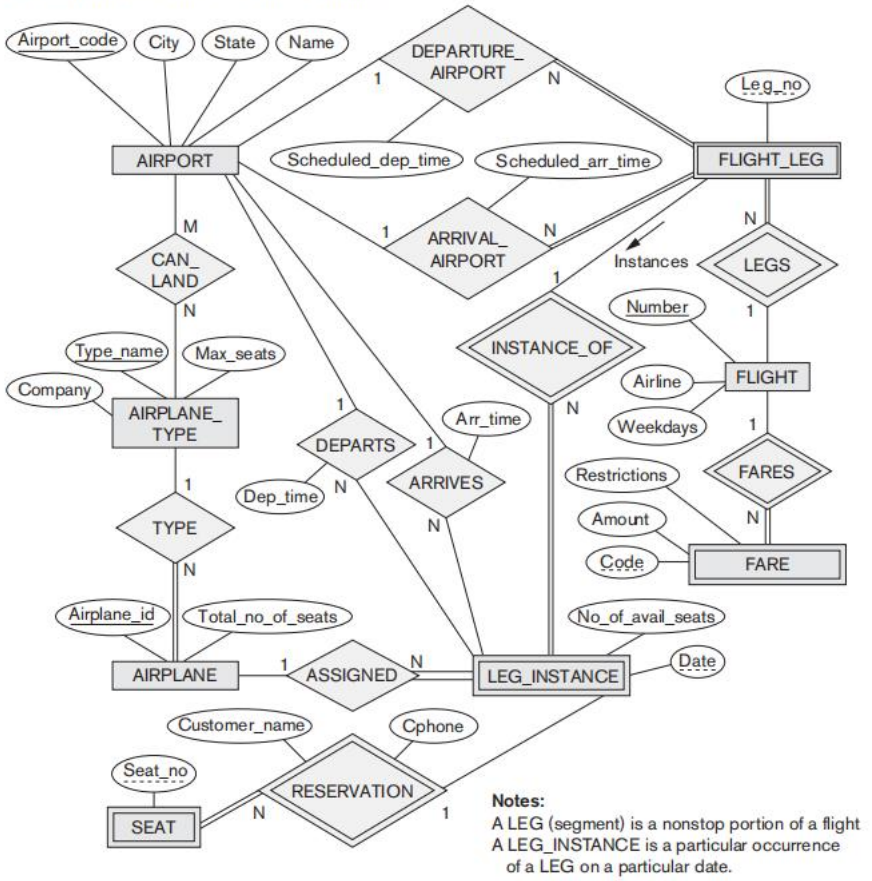
3) Online Shopping System:
==========

Entities:
Customer, Product, Order, Category.
Attributes:
Customer: Customer ID (PK), Name, Email, Address.
Product: Product ID (PK), Name, Price, Description.
Order: Order ID (PK), Order Date, Total Amount.
Category: Category ID (PK), Name.

Relationships:
Customer places Order (One-to-Many).
Order contains Product (Many-to-Many).
Product belongs to Category (Many-to-One).

4) Air Line System:

Figure 7.20
An ER diagram for an AIRLINE database schema.



UNIT:2

Relational Query Languages: Relational Operations. Relational Algebra, Selection and projection set operations, renaming, Joins, Division, Examples of Algebra overviews, Relational calculus, Tuple relational Calculus, Domain relational calculus, Extended relation algebra operation

RELATIONAL MODEL CONCEPTS:

In 1970s the relational model was first introduces by Ted Codd in IBM Research. The model uses the concepts of mathematical relation.

The relational model represents the database as a collection of relations where each relation is nothing but a table with rows and columns.

Relation: A set of records in the form of a table is called a relation.

A relation contains 2 things:

1) relation schema 2) relation instance.

Relation Schema: It contains the basic information of a table. This information includes the name of the table, names of the attributes, datatypes, ranges, primary key etc.
Ex: student (studid: string primary key, name: string, age: integer)

Relation Instance:
At a particular instance of time, all the available records or tuples of a table is called relation instance.

Domain:
A domain D is a set of atomic values. For each attribute there is a set of permitted values called the domain of that attribute.

Relation Cardinality:
The no. of records or rows available in a relation is called relation cardinality.

Relation Degree:
The no. of attributes in a relation is called Relation Degree.

Tuple:
Each row or record of a table called Tuple.

Relational Database Schema: It a collection of relation schemas.

Relational Model Constraints:

These are the restrictions or sets of rules imposed on the database contents. It validates the quality of the database. Mainly Constraints on the relational database are of 4 types

1) Domain constraints
2) Key constraints
3) Entity Integrity constraints
4) Referential integrity constraints

1. Domain Constraints

Every domain must contain atomic values(smallest indivisible units) which means composite and multi-valued attributes are not allowed.

Example:

EMP ID	Name	Phone
01	Alok kumar shah	1234567890 2344566780

In the above relation, Name is a composite attribute and Phone is a multi-values attribute, so it is violating domain constraint.

Correct way is:

EMP ID	First Name	Mid Name	Last Name	Phone1	Phone2
01	Alok	kumar	shah	1234567890	2344566780

2. Key Constraints or Uniqueness Constraints

- These are called uniqueness constraints since it ensures that every tuple in the relation should be unique.
- A relation can have multiple keys or superkeys, out of which we choose one of the keys as the primary key.
- Null values are not allowed in the primary key, hence Not Null constraint is also part of the key constraint.

Example:

EID	Name	Phone	DL	PAN
01	Bikash	6000000009	Kjkl4564	Qqqq1111
02	Paul	9000090009	Qwe12656	Wwww2222
04	Tuhin	9234567892	ASD87612	Eeee4444

In the above table, EID is the primary key, and the first and the last tuple have the same value in EID ie 01, so it is violating the key constraint.

T1[SuperKey] != T2[Superkey]

Where T1 and T2 are Tuples.

Super Key:

Every relation must have atleast one default super key.

A super key is a set of one or more attributes that uniquely identifies each tuple in a relation.

So every relation has one default super key ie: set of all its attributes.

Example:

Consider a relation: Students

Attributes: Students(rollno, name, age, DL)

Key attribute is rollno

Possible super keys are:

1) (rollno, name, age, DL)

2) (rollno, age, DL)

3) (rollno, name, DL)

4) (rollno, DL)

5) (DL)

6) (rollno)

Candidate Keys:

Every key constraint of a relation can be called a candidate key.

Candidate key is an attribute that can uniquely identify a tuple in a relation.

Ex: rollno, adhaarno, panno, driving licence no, registration no, passport no, voter id, bank account no etc.

Primary Key:

In a relation one of the candidate keys is used as a primary key.

A primary key is used to uniquely identify each tuple in a relation and it cannot have NULL value.

Constraints for NULL value:

It is a constraints on attribute which specifies whether NULL values are permitted or not.

Entity Integrity Constraints:

==========

It states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples.

Example:

EID	Name	Phone
01	Bikash	9000900099
02	Paul	600000009
NULL	Sony	9234567892

Referential Integrity:

==========

The Referential integrity constraint is specified between two relations or tables and used to maintain the consistency among the tuples in two relations.

This constraint is seen when a primary key attribute of a relation R1 is used for referring in another relation R2. Here that attribute in R2 is called as Foreign key.

Example:

relation : Products, primary key : PNO

PNO	PNAME	QTY
12	LUX	100
13	MAGGI	200
14	THUMSUP	150

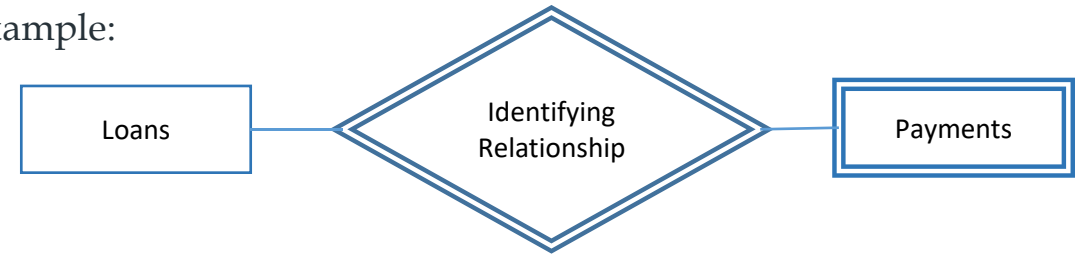
Relation : Sales, primary key : EID, foreign key : ProdNO

EID	Cust Code	Date	ProdNO	Qty
01	C1	1/1/25	12	10
02	C2	2/1/25	13	20
03	C1	2/1/25	13	10
04	C3	3/2/25	14	20
05	C2	4/3/25	12	40
06	C4	4/4/25	17	50

Partial Key:

The attribute of a weak entity set, that is associated with the primary key of a strong entity set to form the primary key of weak entity set, then it is known as partial key.

Example:



Relation: Loans, Primary Key : Loan No

Loan No	Loan Type	Amount	Customer ID

Relation: Loan Payments, Primary Key: Loan No + Pay SL No

Loan No	Pay SL No	Date	Amount

ER MODEL TO RELATIONAL MODEL MAPPING:

To convert ER Model to relational model we can follow the steps as:
1) Mapping of regular entities
2) Mapping of weak entity set
3) Mapping of 1:1 Binary relationship
4) Mapping of 1:N relationship
5) Mapping of Multivalued attributes
6) Mapping of M:N relationship
7) Mapping of N-Ary relationship

1) Mapping of regular entities:
==========
For each regular entity type in the ER diagram, create a corresponding table or relation. The table should include all the simple attributes of the entity type, and one of the key attributes should be designated as the primary key.

Example:
Consider a banking database system having entities: branches, customers, employees, accounts and loans.

Let us create tables for each entities:
Relation: branches

Bcode	Name	City	assets

Relation: Customers

Custid	Cname	address

Empid	Name	Dno	Street	City	Pin	doj	phone

Relation: employees

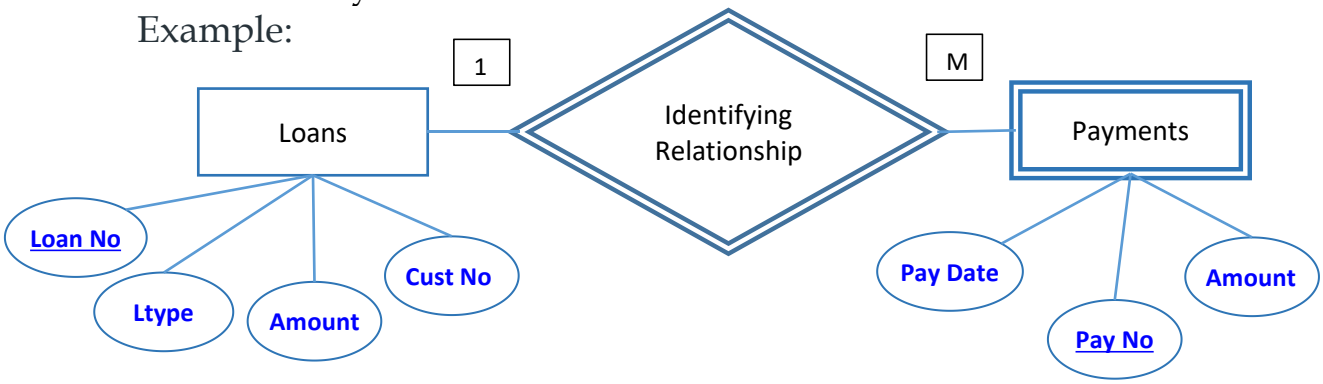
Relation: Accounts

Acno	Ac type	balance

Relation: loans

Loan no	Ltype	amount

2) Mapping of weak entity set:
==========
We need to include the primary key of strong entity as foreign key in weak entity.
Example:



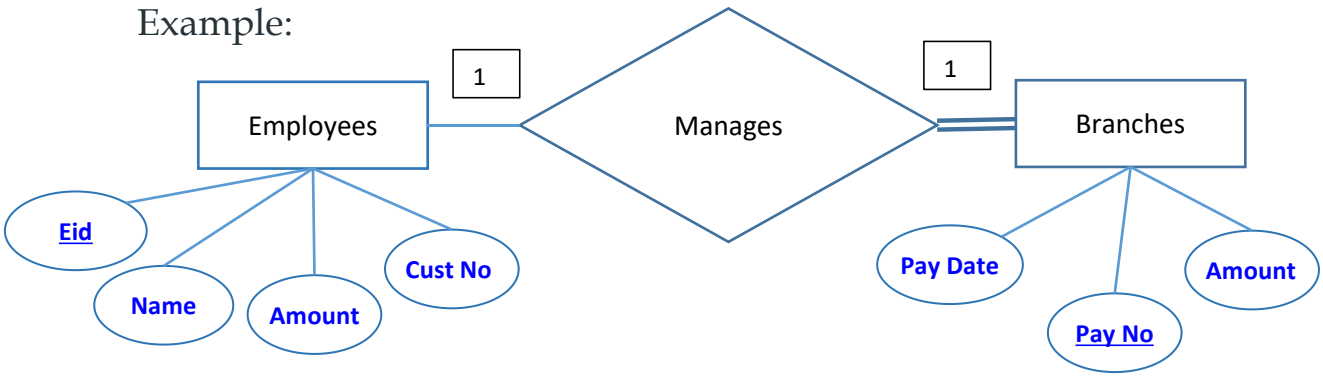
Relation: Loans

Loan No (PK)	Ltype	Amount	Cust No

Relation: Payments

Pay No (PK)	Pay Date	Loan No (FK)	Amount

3) Mapping of 1:1 Binary relationship:
==========
In 1:1 mapping, if an entity having partial participation with another entity then the entity of total participation needs to include foreign key.
Example:
Example:



Relation: Employees

Eid	Name	Address	Phone
1	Abc		
2	QWE		
3	ZXC		

Relation: Branches

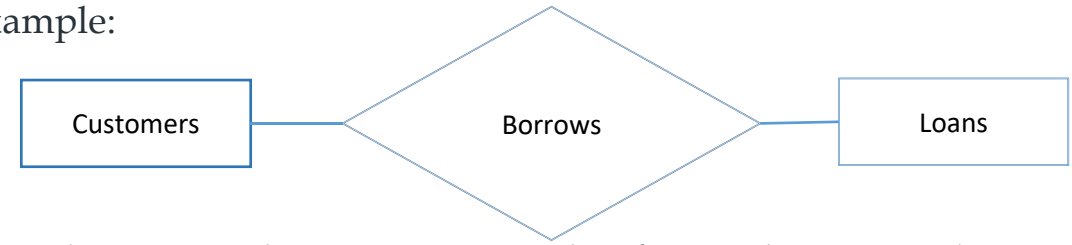
Bcode	Bname	Location	**ManagerID**
B1	CSE		2

4) Mapping of 1:N relationship:

==========

In 1:N relationship, the primary key of 1:relationship to be used as foreign key of N:relationship.

Example:



Here the primary key CustID is used as foreign key CID in loans realtion.

Relation: Customer, Primary Key : CustID

CustID	Name	Address

Relation: Loans, Primary Key: LoanNo

LoanNo	Ltype	Amount	CID

5) Mapping of Multi valued attributes:

==========

Either create a separate relation for the multi valued attribute along with primary key or divide the multi valued attribute into different partitions.

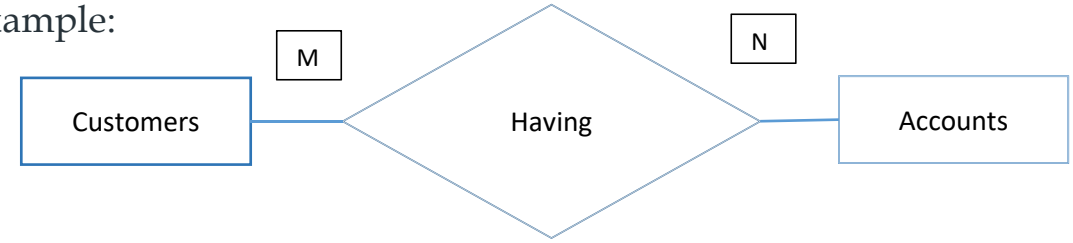
Example1:
Relation: Employees

SLNo	EmpID	Phone
1	E1	111
2	E1	222
3	E1	333

Example2:

EmpID	Phone1	Phone2	Phone3
E1	111	222	333

6) Mapping M:N Relationship:
==========
Create a separate relation which will contain only primary keys of both entities as attributes
Example:



‘

Relation 1: Customers

CustID	Name	Address

Relation 2: Accounts

AC No	AC Type	Balance

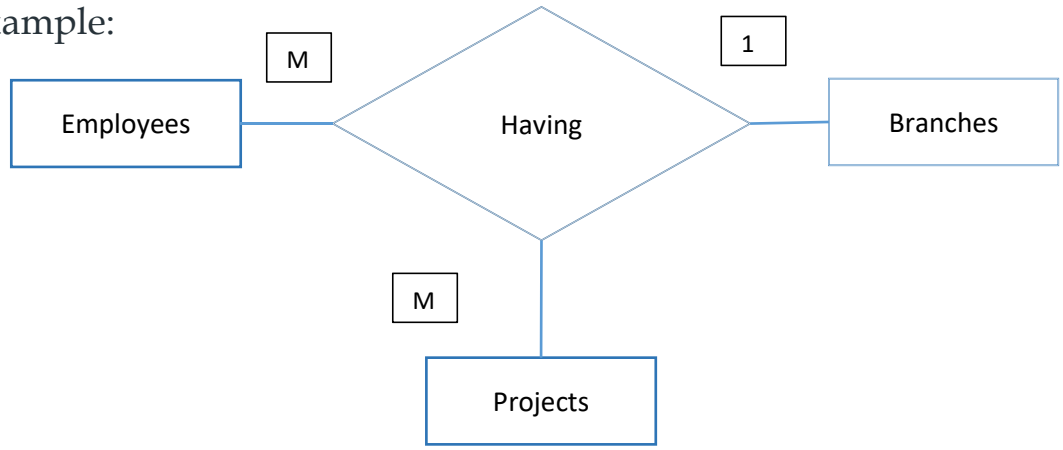
Relation 3: Cust_AC

SLNo	CustID	ACNo

7) Mapping N-Ary Relationship:
Create a separate relation that includes all the primary keys of entity sets.

Example:

Example:



,

SLNo	EmpID	BranchID	ProjectID

==========

CASE STUDY: A SCHEMA DIAGRAM OF A COMPANY

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 3.5
Schema diagram for the COMPANY relational database schema.

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

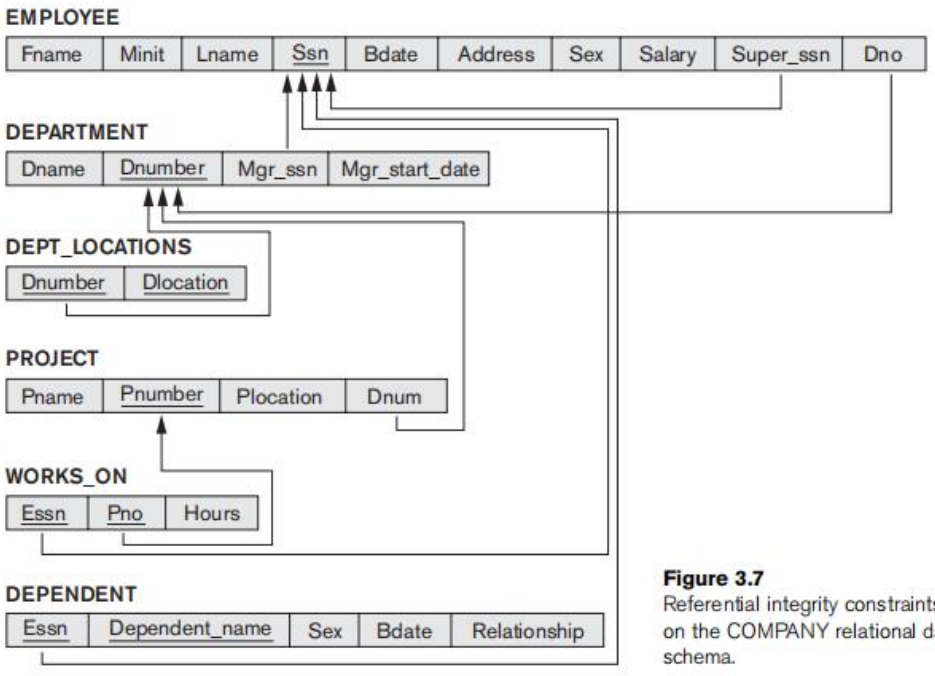


Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

**These shall be followed in Lab for SQL Query Implementation.

==========

CHAPTER : QUERY Languages:

A language which allows users to send a request or query to the database for retrieving some information.

Query language types:

1) **Procedural**
2) **Non-Procedural**

Procedural Language:

Here we have to specify the way of retrieving information from the database. Example: **Relational Algebra.**

Non-Procedural Language:

Here we can retrieve the information without specifying the procedure to retrieve. Example: **Relational Calculus.**

The query language explains the process of querying based on which the different programming languages operates such as:
MY SQL, PL SQL, ISQL etc.

Relational Algebra:

A data model must include a set of operations to manipulate the database. **The basic set of operations for a relational model is called relational algebra.**

Relational Algebra is a formal language used to query and manipulate relational databases, consisting of a set of operations like selection, projection, union, and join. It provides a mathematical framework for querying databases, ensuring efficient data retrieval and manipulation. Relational algebra serves as the mathematical foundation for query SQL.

A sequence of relational algebra operations forms a relational algebra expression.
The fundamental operations in the relational algebra are:
1) Unary operations
2) Binary operations

Unary operations:

The operations are called unary when they operate on one relation.
They are:
i) Select operation
ii) Project operation
iii) Rename operation

Select Operation:

Here we use the lower case greek letter sigma (σ) to denote selection. The selection operation is used to select a subset of tuples from a relation based on a condition.

Syntax:

$$\sigma_{selection\_condition}(relation\_name)$$

The selection_condition formula may use connectors such as or, and, and not. Also, these terms may make use of relational operators such as – =, ≠, ≥, < , >, ≤.

For Example: Consider a Table Customers having attributes Custcode, Name and Location.

Relation: customers

Custcode	Name	location
01	sameer	bbsr
02	kiran	gnpr
03	tapan	kolkata
04	sushil	bbsr
05	kaustav	rgda
06	soumya	bbsr

Ex1: Find the customer details who are living in "BBSR" from the relation "customers"

Query:

$$\sigma_{\text{location = "bbsr"}}(\text{customers})$$

Ex2: to retrieve all the details of a relation:

Query:

$$\sigma(\text{customers})$$

The selection condition may use the relational operators I.e: >, <, >=, <=, !=, =

Also we can join multiple conditions using logical operators:
and operator is AND
Or operator is OR
Not operator is NOT

For Example2: consider a students relation having attributes: Studid, Name, City, Phone, Branch, percentage

Relation: students

Studid	Name	City	Phone	Branch	percentage

1	soumya	bbsr	111	ece	85
2	kiran	gnpr	222	ece	95
3	sourabh	gnpr	333	mech	85
4	abhi	bbsr	444	ece	65
5	srikant	bbsr	555	mech	80
6	kanthi	rgda	666	ece	82
7	sunita	rgda	777	cse	93
8	tarun	bbsr	888	mech	70
9	kanha	bbsr	999	ece	75
10	santi	gnpr	000	cse	95

Now consider the following queries on it:

Query1: Find the details from relation students where branch is ece.

σ branch = "ece" (students)

Query2: Find the details where percentage>80

σ percentage>80 (students)

Query3:
Find the details where branch is ece and belongs to bbsr

σ branch="ece" AND city="bbsr" (students)

Query4:
Find the details where branch is mech or percentage >=85

σ branch="mech" OR percentage>=85 (students)

Query5: Find the details who are not belongs to bbsr.

σ NOT (city="bbsr) (students)

(OR)

σ city !=bbsr (students)

Some sample queries for practice:

==========

Q1) Find the customer details who lives at bbsr and having balance in account>1000 rupees from relation "customers"

σ city = "bbsr" AND balance>1000 (customers)

Q2) Find the employees belongs to maintenance dept having salary more than 20000 from relation "EMP"

σ dept="maintenance" AND salary>20000 (EMP)

It can also be written as:

σ dept = "maintenance" (σ salary>20000 (EMP))

Its syntax is:
σ condition1 (σ condition2 (R))
OR
σ condition2 (σ condition1 (R))
OR
σ condition1 AND condition2 ®

Projection Operation:

==========

The projection operation selects certain columns from the table and discards the other columns.
The symbol used is upper case greek letter pi i.e: Π

Syntax :

Π <attributes list > (relation name)

The projection is nothing but a vertical partition of the relation.

Sample Queries:

==========

Q1) display only the studentID and percentage from the relation "students"
Query:

$$\Pi_{\text{studentID, percenatge}}(\text{students})$$

Q2) Display name and city from relation "students"
Query:

$$\Pi_{\text{name, city}}(\text{students})$$

Composition of selection and projection:
==========
We can combine both selection and projections in queries.

Sample queries:
Q1) Find the studentID and name where branch is ECE from relation students.
Query:

$$\Pi_{\text{studentID, Name}}(\sigma_{\text{branch="ece"}}(\text{students}))$$

Q2) Find the rollno and name from student relation where branch is IT and age>=21

$$\Pi_{\text{rollno, name}}(\sigma_{\text{branch="IT" AND age>=21}}(\text{students}))$$

$$\Pi_{\text{rollno, name}}(\sigma_{\text{branch="IT"}}(\sigma_{\text{age>=21}}(\text{students})))$$

Q3) Find the students name where city is bbsr and branch is ECE.

$$\Pi_{\text{name}}(\sigma_{\text{branch="ECE"AND city="BBSR"}}(\text{students}))$$

Q4:Find name, phone number where branch is ECE and percentage ≥ 85 from relation: students.

$$\Pi_{\text{name,phoneno}}(\sigma_{\text{branch="ECE"AND percent≥85}}(\text{students}))$$

RENAME OPERATION:

It is used to assigned a new name to a table or relation. It is denoted by using a lowe case greek letter rho. **(ρ)**.

Syntax:

ρ<newName> (RelationExpression)

Alternative syntax:

newName←(RelationExpression)

Ex:

ρ employee (σ (emp))

Ex:

ρSales_emp (σ dept=”sales” (emp))

Rename is also used for renaming attributes of a relation.

Example: We can rename an attribute **B** in relation **R** to **D**

ρ(D/B) R

will rename the attribute 'B' of the relation by 'D".

:- Sample Queries :-

Q1) Create an alternate table for all the tuples of relation Emp.

Ans:

ρ staff (σ (emp))

(OR) staff $\leftarrow$ (σ (emp))

Q2) Assign a new table for students rollno, name where branch is EEE.

ρ ECE_list (Π rollno, name (σ branch = "EEE" (student

(OR)

ECE_list $\leftarrow$ (Π rollno, name (σ branch = "EEE" (students)))

Q3) Assign new table for the students name, address who are belongs to BBSR and branch is ECE.

$$Stud\_ece\_bbsr\left(\prod_{name,\,address}\left(\sigma_{City="BBSR"\ AND\ Branch="ECE"}(Students)\right)\right)$$

Q4) Create a relation called stud_data that Contains Studid, name where branch is ECE OR percentage >=85

$$Stud\_data \leftarrow \left(\prod_{studid,\,name}\left(\sigma_{branch="ECE"\ OR\ Percent>=85}(Students)\right)\right)$$

we can break the complex sequence of operations by specifying temporary names for intermediate results.

ex5) Create a relation called stud_data that contains studid, name where branch is ECE OR percentage >=85

Answer:

$$temp \leftarrow \left(\sigma_{branch="ECE"\ OR\ Per>=85}(Students)\right)$$

$Stud\_data \leftarrow (\Pi_{studid, name}(temp))$

We can even use rename operation for renaming the attributes of alternate relation created.

Example:

Create a relation called Stud having attributes: rollno, loc. It should contain data studid, city from students relation whose branch is Mech and percent >=80.

$temp \leftarrow (\sigma_{branch="Mech" \text{ AND } Per>=80}(Students))$

$Stud(rollno, loc) \leftarrow (\Pi_{studid, city}(temp))$

Here studid is renamed as rollno
City is renamed as loc.

Binary Operations:

The operations are known as binary when they operates on a pair of relations.

They are:

i) **Union operations**
ii) **Intersection operations**
iii) **Set différence**
iv) **Cartesian product**
v) **Natural join**
vi) **Division**
vii) **Assignment**

Before we apply the binary operations on two relations **here we need to check for** the compatibility known as **Tuple compatibility.**

Two relations R(a1,a2,a3,…an) and S(b1, b2, b3, …bn) are union compatible,if they have same degree and if domain(Ai) = domain (Bi)

That means the no. of attributes and type of attributes must match in both the relations.

Union operations:

==========

It is an operation from set theory denoted by U. It is used to combine two relations ex: R U S.

The result of this operation contains all the tuples that are either in R or in S or in both. Here the duplicate tuples are eliminated.

Q=RUS then, Q will contain all the tuples of R and S by excluding duplicacy.

Example:

Relation: Java(rollno, name, address)

Relation: Python(rollno, name, address)

Relation: Java(rollno, name, address)

rollno	name	address
j01	Soumya	bbsr
j02	Kiran	Gnpr
j03	Shivani	bbsr

Relation: Python(rollno, name, address)

rollno	name	address
p01	Tina	bbsr
p02	Meena	bbsr
j03	Shivani	bbsr
p03	Kishan	Gnpr

Operation:

Query1: Java ∪ Python

Result (Union of Java and Python):

rollno	name	address
j01	Soumya	bbsr
j02	Kiran	Gnpr
j03	Shivani	bbsr
p01	Tina	bbsr
p02	Meena	bbsr
p03	Kishan	Gnpr

*** Here duplicate is eliminated.**

Example2:
Query2

πrollno(Java) ∪ πrollno (Python)

rollno
j01
j02
j03

rollno
p01
p02
j03
p03

Result:
rollno
j01
j02
j03
p01
p02

rollno

p03

*** Duplicate j03 is eliminated.**

Example3:

Query3:

πaddress(Java) $\cup$ πaddress(Python)

πaddress(Java)

address

bbsr

Gnpr

bbsr

πaddress(Python)

address

bbsr

bbsr

bbsr

Gnpr

Result: $\cup$

address

bbsr

Gnpr

Only two unique addresses remain.

Ex4:

Suppose we have two different relations that do not directly match:

Students: (studid, name, address, phone, age)

Results: (studid, name, percent, grade)

We can only apply union after projecting common attributes:

Query4:

$\pi_{studid,name}(Students) \cup \pi_{studid,name}(Results)$

INTERSECTION OPERATION:

==========

It takes two relations as input and produces a resultant relation that contains common tuples from both tables. It is denoted by ∩.

Suppose Q=R∩S, then Q will contains the common tuples of both R and S.

Example1:

Query1:

Java ∩ Python

Display the common list of address in both the relations:

rollno	name	address
j03	Shivani	bbsr

Example2:

Query2:

$\pi_{address}(Java) \cap \pi_{address}(Python)$

Result:

address

bbsr

Gnpr

Set difference operation (Minus)

==========

It is a binary operation which takes two relations as an input and It finds the tuples that are available in one relation but not in another relation.

Ex1:
Suppose R = {1,2,3,4,5} and S={3,5,9,11}
If Q=R-S Then Q={1,2,4}
If Q=S-R Then Q= {9,11}

Ex:
Java – Python → Tuples in Java but not in Python
Python – Java → Tuples in Python but not in Java

Query1:

Java–Python

Result:

rollno	name	address
j01	Soumya	bbsr
j02	Kiran	Gnpr

Ex:
Python–Java
Result:

rollno	name	address
p01	Tina	bbsr
p02	Meena	bbsr
p03	Kishan	Gnpr

Ex: Find the address from relation: java which is not exist in relation : python

Table: java

rollno	name	address
j01	Soumya	bbsr
j02	Kiran	Gnpr
j03	Shivani	bbsr
j04	Sam	rgda

Table : python

rollno	name	address
p01	Tina	bbsr
p02	Meena	bbsr

rollno	name	address
j03	Shivani	bbsr
p03	Kishan	Gnpr

Query2:

π address(Java) – π address(Python)

steps:

Step 1: Projection on address

From Java → { bbsr, Gnpr, rgda }

From Python → { bbsr, Gnpr }

Step2:

{bbsr,Gnpr,rgda}–{bbsr,Gnpr}

Step3:

{rgda}

Result:

address

rgda

Problem:

Consider relation:

Students(studid, name, branch, address, age, percent, phone).
Find the student details where branch is not ECE.

Step 1: Extract ECE students

stud_ece ← (σbranch="ECE" (Students))

Step 2: Students NOT in ECE

NONECE= Students–stud_ece

Find only the IDs and names of students who are not in ECE.

Query:

stud_ece ← (σbranch="ECE" (Students))

NONECEdata←(πstudid,name(Students))−(πstudid,name(stud_ece))

Cartesian Product (X)

==========

We already are aware of the fact that relations are nothing but a set of tuples, and here we will have 2 sets of tuples.

On applying CARTESIAN PRODUCT on two relations that is on two sets of tuples, it will take every tuple one by one from the left set(relation) and will pair it up with all the tuples in the right set(relation).

If we have two relations:

the **CROSS PRODUCT (X)** of two relation **R(R1, R2, R3, ..., Rp)** with degree p, and **S(S1, S2, S3, ..., Sn)** with degree n, is a relation **T(R1, R2, R3, ..., Rp, S1, S2, S3, ..., Sn)** with degree p + n attributes.

Notation:

T=R × S

where R and S are the relations,
the symbol '×' is used to denote the CROSS PRODUCT operator.

R(R1, R2, R3, ..., Rp) and S(S1, S2, S3, ..., Sn)

Schema of T: **T(R1,R2,R3,...,Rp,S1,S2,...,Sn)**

Degree of T: **p+n(sum of attributes of both relations)**

Number of tuples: **| R | × | S |**

Example

Let's say:

Relation R (Students):

R1 (sid)	R2 (sname)
S1	Soumya
S2	Kiran
S3	Tina

Relation S (Courses)

S1 (cid)	S2 (cname)	S3 (credits)
C1	Java	3

S1 (cid)	S2 (cname)	S3 (credits)
C2	Python	4
C3	DBMS	3
C4	OS	4

Cartesian Product: T = R × S

sid	sname	cid	cname	credits
S1	Soumya	C1	Java	3
S1	Soumya	C2	Python	4
S1	Soumya	C3	DBMS	3
S1	Soumya	C4	OS	4
S2	Kiran	C1	Java	3
S2	Kiran	C2	Python	4
S2	Kiran	C3	DBMS	3
S2	Kiran	C4	OS	4
S3	Tina	C1	Java	3
S3	Tina	C2	Python	4
S3	Tina	C3	DBMS	3
S3	Tina	C4	OS	4

5 attributes (sid, sname, cid, cname, credits)

12 tuples (3 × 4)

EXAMPLE2:

Step1: Suppose given two relations:customers and borrowers

customers

Cid	name	Address
101	a	gnpr
102	b	rgda
104	c	bbsr
108	d	gnpr
110	e	berh

borrowers

loanno	Custid
L01	102
L02	104
L03	108

Step 2: Cartesian Product (Customer × Borrower)

The result will have attributes:
(Customer.Cid, Customer.name, Customer.Address, Borrower.loanno, Borrower.Custid)

Since **Customer has 5 rows** and **Borrower has 3 rows**, the Cartesian product has **5 × 3 = 15 rows**.
This includes many tuples where Customer.Cid ≠ Borrower.Custid.

Step 3: Selection (Join Condition)

We apply condition:

$$\sigma_{Customer.Cid=Borrower.Custid}\ (Customer \times Borrower)$$

Results is:

Customer.Cid	Customer.name	Customer.Address	Borrower.Custid	Borrower.loanno
102	b	rgda	102	L01
104	c	bbsr	104	L02
108	d	gnpr	108	L03

Step 4: Projection (Getting only desired attributes)

If we want only Customer.name and Borrower.loanno, apply projection:

$$\pi_{Customer.name,Borrower.loanno}\ (\sigma_{Customer.Cid=Borrower.Custid}\ (Customer \times Borrower))$$

Final Result:

Customer.name	Borrower.loanno
b	L01

Customer.name	Borrower.loanno
c	L02
d	L03

Example 2: Retrieve a list of names of each female employee's dependents

Given Relations:

EMP(eno, ename, age, sex)

Dependent(dname, eeno, age, relation)

Step 1: Select Female Employees
emp_name←πename,eno (σsex='F' (EMP))

This gives only **female employees** with their names and employee numbers.

Step 2: Cartesian Product + Selection

We match female employees with their dependents:

Act_depend←σeno=eeno(emp_name×Dependent)

This ensures that we only keep dependents related to those female employees.

Join Operation:

==========

The join operation is **denoted by** ⋈

It is used to combine two relations by **matching key attributes**.

A join can be written as:

$R \bowtie_{\text{<join condition>}} S$

The join condition will be denoted as: $Ai\ \theta\ Bj$

Where Ai is the attribute of R.

Bj is the attribute of S

And θ is the comparison operator

Example1: $\text{Customers} \bowtie_{\text{(Customers.CustID = Loans.CustID)}} \text{Loans}$

Example2:

$\text{Temp} \leftarrow (\text{student}) \bowtie_{\text{student.roll=Exam.roll}} (\text{Exam})$

Example3:
Relation1: EMP(eno, ename, salary, dept)
Relation2: Dependents(depno, dname, eeno)

emp_name <- (Πeno,ename (EMP)
Emp_name (eno,ename)
Dependents(depno, dname, eeno)
(Emp.eno,Emp.ename, dependent.depno, dependent.dname, dependent.eeno)

$\text{Result} \leftarrow \Pi_{\text{dname,ename}} (\text{emp\_name} \bowtie_{\text{eno=eeno}} \text{Dependent})$

Example 3:
Given relations:
Emp(eno, ename, address, salary)
Dept(dno, dname, summary, mgrno)
Task: Find employees who are managers of departments.

Relational Algebra Expression:

1. **Join operation:**

$$X \leftarrow Dept \bowtie_{mgrno=eno} Emp$$

(Join Dept and Emp where the manager number = employee number)

2. **Projection:**

$$\Pi_{dname,ename}(X)$$

(Select only department name and employee name from the result)

We join the Dept table with the Emp table based on the condition that the department's mgrno (manager number) matches the employee's eno (employee number).

From the result of this join, we project (Π) only the department name (dname) and employee name (ename) to get the required output: employees who are managers of departments, along with the department names.

Types of Join

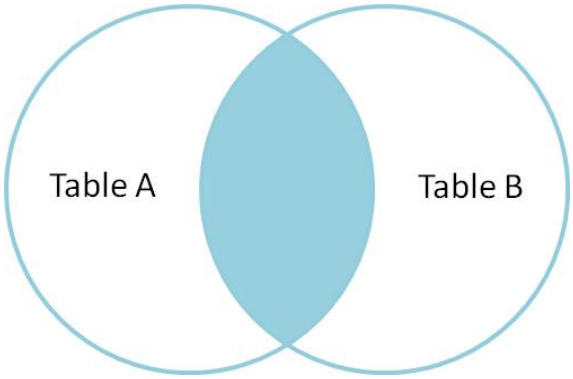
There are many types of Joins in SQL. Here are the frequently used SQL JOIN types:

1. Inner Join

Inner Join is a join operation in DBMS that combines two or more tables based on related columns and returns only rows that have matching values among tables.

Inner join has three types.

1) Conditional join
2) Equi Join
3) Natural Join



(a) Conditional Join

Conditional join or Theta join is a type of inner join in which tables are combined based on the specified condition.

In conditional join, the join condition can include <, >, <=, >=, ≠ operators in addition to the '=' operator.

Example: Suppose two tables A and B

Table A

R	S
10	5
7	20

Table B

T	U
10	12
17	6

$$A \bowtie_{S<T} B$$

Output

R	S	T	U
10	5	10	12

R	S	T	U
10	5	17	6

(b) Equi Join

Equi Join is a type of inner join where the join condition uses the equality operator ('=') between columns.

Example: Suppose there are two tables Table A and Table C

$$A \bowtie_{A.ColumnB = C.ColB} C$$

(c) Natural Join

Natural join is a type of inner join in which we do not need any comparison operators.

In natural join, columns should have the same name and domain. There should be at least one common attribute between the two tables.

Ex:

$$A \bowtie_{A.ColB = B.ColB} B$$

It can also be written as :

$$A \bowtie B$$

2. Outer Join

Outer join is a type of join that retrieves matching as well as non-matching records from related tables. There are three types of outer join

- Left outer join
- Right outer join
- Full outer join

(a) Left Outer Join

This type of outer join retrieves all records from the left table and retrieves matching records from the right table.

Example: Suppose there are two tables Table A and Table B

Table A

Number	Square
2	4
3	9
4	16

Table B

Number	Cube
2	8
3	27
5	125

Query: $A \bowtie B$

Output

Number	Square	Cube
2	4	8
3	9	27
4	16	NULL

(b) Right Outer Join

This type of outer join retrieves all records from the right table and retrieves matching records from the left table.

Right Outer Join
Example: Suppose there are two tables Table A and Table B

Output:

Number	Square	Cube
2	4	8
3	9	27
5	NULL	125

(c) Full Outer Join

FULL JOIN creates the result set by combining the results of both LEFT JOIN and RIGHT JOIN. The result set will contain all the rows from both tables.

Example: Table A and Table B are the same as in the left outer join

Output:

Number	Square	Cube
2	4	8
3	9	27
4	16	NULL
5	NULL	125

DIVISION OPERATOR (%)

==========

Division in SQL is typically required when you want to find out entities that are interacting with **all entities** of a set of different types of entities.

The division operator is used when we have to evaluate queries that contain the keyword 'all'.

Here are a few common examples:

- Identifying people who have accounts in every bank within a particular city.
- Determining students who have enrolled in all necessary courses to qualify for graduation.

Example:

Emp(empid, ename, age, salary)

Find the **employeeid** of employee table who works on **all the projects** that "*Mahesh*" works on.

Given relation:

Emp(empid, name, age, salary)

Relations

- Emp(empid, name, age, salary)
- Projects(pno, pname, location)
- WorksOn(empid, pno, hours)

This models a many-to-many relationship between employees and projects.

Table: EMP

empid	name	age	salary
101	Mahesh	21	1000
102	Kanti	31	2000

empid	name	age	salary
103	Kiran	41	3000
104	Soma	51	4000
105	Naina	61	5000

Table: Projects

pno	pname	location
P1	SS	gnpr
P2	PP	gnpr
P3	QQ	rgda
P4	RR	bbsr

Table: WorksOn

empid	pno	hours
101	P1	11
101	P3	2
101	P4	10
102	P1	5
102	P4	10
103	P1	10
103	P3	5
103	P4	10
104	P2	5
105	P4	15

QUERY:

Step 1:

$$mah \leftarrow \sigma_{fname="Mahesh"}(Emp)$$

- This selects tuples from **Emp** (Employee table) where the first name is *Mahesh*.
- Result: `mah` contains details of employee(s) named Mahesh.

O/P:

The relation `mah` contains Mahesh's details:

empid	name	age	salary
101	Mahesh	21	1000

Step 2:

$$mah\_pno \leftarrow \pi_{pno}\Big((works\_on \bowtie_{empid=empid} mah)\Big)$$

- Join **works_on** with `mah` (on empid).
- Project `pno` (project numbers).
- Result: `mah_pno` contains all projects on which Mahesh works.

O/P:

The projects that Mahesh (empid = 101) works on are:

pno
P1
P3
P4

Step 3:

$$Eno\_pno \leftarrow \pi_{empid,pno}(works\_on)$$

- Projects only employee IDs and project numbers from **works_on**.
- Result: `Eno_pno` is a relation showing which employee works on which project.

O/P:

The full employee-project assignments are:

empid	pno
101	P1
101	P3
101	P4
102	P1
102	P4
103	P1
103	P3
103	P4
104	P2
105	P4

Step 4:

$$Enos(empid) \leftarrow Eno\_pno \div mah\_pno$$

- Division operation: Find all `empid` in `Eno_pno` that are associated with **all the projects Mahesh works on**.
- Result: `Enos` contains IDs of employees who work on **all projects that Mahesh works on**.

O/P:

- **mah_pno = {P1, P3, P4}**
- We now find employees in `Eno_pno` who are assigned to **all three projects {P1, P3, P4}**.

Checking employee by employee:

- **101 (Mahesh)** → Works on {P1, P3, P4} ✅
- **102** → Works on {P1, P4}, but **missing P3** ❌
- **103** → Works on {P1, P3, P4} ✅
- **104** → Works on {P2} only ❌
- **105** → Works on {P4} only ❌

Employees who work on **all projects that Mahesh works on**:

empid
101
103

Assignment Operation (←)

- **Definition:**

  It is used to assign the result of a **relation expression** to a **new variable**.

- **Example:**

$$Temp \leftarrow \pi_{empid,name}(Emp)$$

  - Here, we project only `empid` and `name` from the relation **Emp**.
  - The result is assigned to a new relation (temporary variable) called **Temp**.

—: Relational Calculus :—

Relational Calculus is a non-procedural language. It specifies what is to be retrieved rather than how to be retrieved.

A calculus expression does not contain how a query to be evaluated.

The Relational Calculus is of 2 types:

1. Tuple Relational Calculus (TRC)
   (based on Tuple variables)
2. Domain Relational Calculus (DRC)
   (based on domain variables)

Tuple Relational Calculus (TRC) in DBMS

Definition:

TRC is based on specifying a number of tuple variables.

A tuple variable may take any individual tuple from a relation as its value.

Syntax: The basic syntax of TRC is as follows:

$$\{ t \mid cond(t) \}$$

where: t is a tuple variable that represents a set of all tuples, cond(t) is a conditional expression involving t.

Result: The set of all tuples t that satisfy cond(t)

Example (Q1)

Query:
Find all the employees whose salary is above 50000.

TRC Expression:

$$\{ t \mid Emp(t) \land t.sal > 50000 \}$$

Here:
t is a tuple variable representing tuples from the relation Emp.
Condition: t.sal > 50000.

Meaning: The TRC variable ***t*** represents a tuple such that:

t ∈ Emp (belongs to Emp relation), and t.sal > 50000.
So the result = all employees with salary > 50000.

Query 2

Retrieve first name and last name of employees whose salary is above ₹50,000.

TRC Expression:

$$\{\langle t.fname, t.lname\rangle \mid emp(t) \land t.sal > 50000\}$$

✔ Meaning: Select `fname` and `lname` from `Emp` where `sal > 50000`.

Query 3

Retrieve the date of birth and address of the employee whose name is "Mahesh".

TRC Expression:

$$\{\langle t.dob, t.address\rangle \mid emp(t) \ \wedge \ t.fname = "Mahesh"\}$$

✔ Meaning: From `Emp`, pick `dob` and `address` of employee(s) whose first name is "Mahesh".

Query 4

Find the names of male students who belong to the CSE branch and are older than 20.

TRC Expression:

$$\{\langle t.name\rangle \mid student(t) \ \wedge \ t.gender = "M" \ \wedge \ t.branch = "CSE" \ \wedge \ t.age > 20\}$$

✔ Meaning: From `Student`, return `name` of students who are male, in CSE branch, and older than 20.

Query 5

Find the name and phone number of customers who belong to "Delhi".

TRC Expression:

$$\{\langle t.name, t.phno\rangle \mid cust(t) \ \wedge \ t.city = "Delhi"\}$$

✔ Meaning: From `Cust`, retrieve `name` and `phno` of customers living in Delhi.

Aggregate Functions in Relational Algebra

These are special functions used for mathematical calculations on attributes of tuples.

Functions:
SUM → total of values
AVERAGE (AVG) → mean value
MAXIMUM (MAX) → largest value
MINIMUM (MIN) → smallest value
COUNT → number of tuples

Grouping in Relational Algebra

Sometimes we need to:

- Group tuples in a relation based on some attribute(s).
- Apply an aggregate function on each group.

This is similar to GROUP BY in SQL.

Notation

It is denoted by **F** (script F):

$$\gamma_{\text{grouping attributes}} \text{ Functions List } (R)$$

- **grouping attributes** → attributes on which tuples are grouped.
- **Functions list** → set of aggregate functions applied.
- **R** → relation.

Example

Suppose relation:

`Emp(empid, dept, salary)`

👉 To find the **average salary in each department**:

$$\gamma_{\text{dept}}\ AVG(salary)(Emp)$$

👉 To find the **count of employees in each department**:

$$\gamma_{\text{dept}}\ COUNT(empid)(Emp)$$

Query 1

Retrieve employees by grouping on deptno and their average salary.

Relational Algebra:

$$\gamma_{Dno}\ \text{Count}(Eno),\ \text{Average}(sal)\ (EMP)$$

- Group by `Dno` (department number).
- Count number of employees (`Eno`).
- Compute average salary (`sal`).

✔ Equivalent SQL:

```sql
SELECT Dno, COUNT(Eno) AS Count_Eno, AVG(Sal) AS Average_sal
FROM Emp
GROUP BY Dno;
```

Query 2

Count employees and find total amount of salary by grouping on deptno.

Relational Algebra:

$$\gamma_{Dno}\ \mathrm{Count}(Eno),\ \mathrm{Sum}(sal)\ (EMP)$$

- Group by `Dno`.
- Count employees.
- Find sum of salaries.

✔ Equivalent SQL:

```sql
SELECT Dno, COUNT(Eno) AS Count_Eno, SUM(Sal) AS Sum_sal
FROM Emp
GROUP BY Dno;
```

Domain Relational Calculus (DRC)

A variable that ranges over the domain of attributes is called a domain variable.
The general form of a DRC expression is:

$$\{x_1, x_2, \ldots, x_n \mid \mathrm{Condition}(x_1, x_2, \ldots, x_n)\}$$

Example 1
Relation:
Students(rollno, name, age, gender, deptname)

Query: Find rollno, name of students whose deptname is "eee".

Domain variables:
a = rollno, b = name, c = age, d = gender, e = deptname

Expression:

$$\{a, b \mid Students(a, b, c, d, e) \text{ AND } e = "eee"\}$$

Example 2
Query: Find name, deptname of all male students.
Expression:

$$\{b, e \mid Students(a, b, c, d, e) \text{ AND } d = "M"\}$$

Example 3
Relation:
Emp(eno, name, address, salary, deptno)
Query: Find name, address of all employees who either:
Work in Deptno = 4 and salary > 10000
OR
Work in Deptno = 5 and salary > 15000

Domain variables:
a = eno, b = name, c = address, d = salary, e = deptno

Expression:

$$\{b, c \mid Emp(a, b, c, d, e) \text{ AND } ((e = 4 \text{ AND } d > 10000) \text{ OR } (e = 5 \text{ AND } d > 15000))\}$$

Database Development Life Cycle (DDLC)

Similar to the Software Development Life Cycle (SDLC), it generally consists of:
Feasibility Study
Requirement Analysis…….SRS document
Designing Phase……..DD
Coding & Unit Testing……..CODING
Integration & System Testing
Deployment & Maintenance

The Designing Phase is based on the requirements analysis, which uses tools like:
DFD (Data Flow Diagrams)
ER Diagram,
Use case diagrams
Class diagrams
Object diagrams
Decision tree
Decision table
Data dictionary

Types of requirements:

1) Functional Requirements
2) Data Requirements

Functional Requirements: It specifies different operations required on data.

Data Requirements: It specifies the data, its decomposition and management.

Database Development Life Cycle: It is a process of designing, implementing, and maintaining a database system to meet the operational needs of an organization.

Database Requirements Analysis: The process of collecting the details by analysing the requirements related to database.

Conceptual Database Design: Based on overall data requirements constructing a model of information including relationship.

DBMS Selection: Selecting a particular DBMS S/W to meet the current needs and for future expansions.

Prototyping: Quickly building an interactive working model of the database applications.

Normalization Concepts:

It is a formal way to decompose the database relations so that **redundancy & inconsistency can be minimized.**

The process of normalization follows the process of drawing ER diagrams.

Normalization is required to achieve:

- **To reduce redundancy.**
- **To minimize errors during insertion, updation, deletion.**
- **To use the memory efficiently.**
- **To eliminate NULL values.**

***Normalization** is applied on relation schemas based on their functional dependencies and primary keys.*

Functional Dependency:

A functional dependency occurs when the value of one attribute (or a set of attributes) **uniquely determines the value of another attribute.**

Example:

Consider a table named Students with the following attributes:

- StudentID
- StudentName
- StudentAge

If each student has a unique StudentID, and this ID determines the student's name, we can express this functional dependency as:

StudentID → StudentName

Let us consider a relation schema
R = { A_1, A_2, A_3, ... An }
For two sets of attributes X and Y that are subsets of R,
the functional dependency can be denoted by (X → Y)

Here, **X** is the determinant, and **Y** is the dependent attribute.

This means that for each unique value of X, there is precisely one corresponding value of Y.

Here the constraint is that,
for any two tuples t_1 and t_2 in R that have
$t_1[X] = t_2[X]$, then they also must have
$t_1[Y] = t_2[Y]$.

It means Y component of a tuple depends on X component.
Y is functionally dependent (FD) on X.

Example 1:
Students (rollno, name, address, age, branch)

Let us consider X represents rollno and Y represents age.
Here {rollno} -> {Age}

So the relation r of R can be shown as:

TRANSACTION	Rollno (X)	Age (Y)
T1	**101**	**21**
T2	102	31
T3	103	41
T4	104	21
T5	**101**	**21**
T6	102	31

Now, here we can observe the functional dependency between tuples t_1 and t_5 .

Here $t_1[X]$ is 101 and also $t_5[X]$ is 101,
so here $t_1[X] = t_5[X]$.
Then the $t_1[Y]$ must be equal to $t_5[Y]$ i.e: 21

It means Y component of a tuple is dependent on X component.
So Y is functionally dependent on X.

It means in any tuple if the rollno is 101 then the age must be 21. So rollno determines the age or age is functionally dependent in rollno.

Note:

1) A primary key can determine all other attributes in a relation.

2) if X determines Y, it does not say that Y can determine X.

Ex:
Products(PID, pname, price, mfd, expdate, qty)
Here primary key is PID
Possible keys:
(PID, pname, price, mfd, expdate, qty)
(PID, pname, price, mfd, expdate)
(PID, pname, price, mfd, qty)
(PID, pname, price)
(PID, pname)
(PID)

Example2:
Given relational schema:
Emp_Dept(empid, dno, hours, ename, dname, dept_loc)

Here the possible Functional Dependencies (FDs) are:

(a) {empid} → {ename}
Each employee ID uniquely determines the employee name.

(b) {dno} → {dname, dept_loc}
Each department number uniquely determines the department name and its location.

(c) {empid, dno} → {hours}
A combination of employee ID and department number uniquely determines the number of hours the employee works in that department.

Types of Functional Dependencies

There are 3 types:
- **Full Functional Dependency**
- **Partial Functional Dependency**
- **Transitive Dependency**

Key Terms

• Determinant: The attribute(s) on the left-hand side of FD (X in X → Y).
• Dependent: The attribute(s) on the right-hand side of FD (Y in X → Y).
• Candidate Key: A minimal attribute set whose closure contains all attributes of the relation.

FULL FUNCTIONAL DEPENDENCY:
==========
An attribute is fully functionally dependent on a key if it depends on the entire composite key and not on a subset of it.

EXAMPLE:

$$Student(\text{sno, name, addr, age})$$

- Suppose:
  $X = \{sno, name\}, Y = \{age\}$
- Here, "age" depends only on "sno" (student number), not on "name".
- Therefore, the "name" attribute is unnecessary in determining "age".
- So, we can refine:
  $X = \{sno\}, Y = \{age\}$

Thus, age is **fully functionally dependent** on sno.

Example2:
Suppose {EmpID, DeptNo} -> {Dname, loc}
But here to determine Department Name and Location the DeptNo is sufficient. So we can refine it by removing EmpID as:
{DeptNo} -> {Dname, loc}

PARTIAL FUNCTIONAL DEPENDENCY:
==========
A non-key attribute is partially dependent if it depends on only a part of the composite primary key.
Example1:
Suppose {EmpID, DeptNo} -> {Dname, loc}
Here we have EmpID and DeptNo as composite primary key.
But here {Dname, loc} can be determined using only DeptNo. So they are partially dependent on composite primary key.

Example2:
Student_Course(studentID, studentName, phoneNo, courseID, courseName, domain, no_of_hours, Classroom, grade, enrollment_date)

If we consider {studentID, courseID} as composite primary key then:

==========

The Functional Dependencies (FDs) are:

1. {studentID} → {studentName, phoneNo}

A student ID uniquely identifies a student's name and phone number.

2. {courseID} → {courseName, domain, classroom}
A course ID uniquely determines the course name and domain

3. {studentID,courseID} → {no_of_hours, grade, enrollment_date}
The number of hours, grade, enrollment_date depends on both the student and the course together.

TRANSITIVE DEPENDENCY:
==========
A non-key attribute depends on another non-key attribute, which in turn depends on the primary key.

In a relation R, if attributes X, Y, Z exist:

- If **Z is functionally dependent on Y**,
- And **Y is functionally dependent on X**,
- Then **Z is transitively dependent on X**.

👉 In short:

$$X \to Y \quad \text{and} \quad Y \to Z \quad \Rightarrow \quad X \to Z$$

Example1:
Consider relation: Student(studentID, deptID, deptName)
Functional Dependencies:
1. studentID→deptID
(each student belongs to a department)
2. deptID→deptName
(a department ID determines its name)
This is a transitive dependency, because deptName depends on deptID, and deptID depends on studentID

Example2:
Stud_course(rno, name, ccode, cname, duration)
Here {rno} -> {ccode}
and {ccode} -> {duration}
So, there is transitive dependency i.e: {rno} -> {duration}

Trivial FD:

X → Y is trivial if Y is a subset of X.

Examples:

{EmployeeID, EmployeeName} → EmployeeName

{RollNo, StudentName}→StudentName

(StudentName is included in the determinant).

{A, B, C}→{A}

(A is a subset of {A, B, C}).

Non-trivial FD: X → Y is non-trivial if Y is not part of X.

Example:

StudentID → StudentName

{CourseCode}→CourseName

(CourseName is not part of CourseCode).

A→B

(if B is not included in A).

Key concepts of functional dependency:

- **Notation:** An FD is represented as $X \rightarrow Y$, which means that the attribute set X functionally determines the attribute set Y.
- **Determinant and dependent:** In the expression $X \rightarrow Y$, the left-side attribute set, X, is called the **determinant**. The right-side attribute set, Y, is the **dependent**.
- **Rule:** The FD $X \rightarrow Y$ holds if two tuples (rows) that have the same value for X also have the same value for Y. In other words, knowing the value of X allows you to uniquely determine the value of Y.
- **Example:** In a `STUDENT` table, `StudentID` is a determinant for `StudentName` and `StudentAge`. This is represented as `StudentID` → `StudentName`, `StudentAge`.

Benefits of Functional Dependency in DBMS

1) helps in reducing redundancy and saving storage space.
2) organizing data efficiently and ensures that the data is reliable, consistent, and of high quality.
3) Reduces the chances of errors in records.
4) Allows for quicker and easier access

NORMALIZATION (UNIT-III)

Normalization is an important process in database design. Database normalization is the process of organizing the attributes of the database to reduce or eliminate data redundancy.

It is the step by step decomposition of complex records into simple records. Normalization reduces redundancy. The normalization process first proposed by Boyse Codd in 1972.
The normalization is used to achieve minimizing redundancy, errors during insertion, deletion and updation.

Types of Normalization:

1) 1st Normal Form (1NF)
2) 2nd Normal Form
3) 3rd Normal Form and Boyes Codd Normal Form (BCNF or 3.5NF)
4) 4th Normal Form
5) 5th Normal Form and Projection Join Normal Form

1st Normal Form:

A relation is in first normal form if every attribute in that relation is single-valued attribute or atomic attribute. The atomic attributives cannot be sub-divided. So, here Multi valued attributes and composite attributes need to be sub-divided and convert into single valued attributes.

Example1:

data(rollno, name, age, gender, phoneno)

Here:

To Handle composit attribute name:

name -> composite attribute:

We can split into first name, middle name, last name.

We must make all attributes atomic:

data(rollno, firstname, middlename, lastname, age, gender, phoneno)

To Handle multi-valued phoneno:

Phoneno -> multi valued attribute. Here we can split into multiple rows shown below:

Option1

rollno	firstname	middlename	lastname	age	gender	phoneno
101	Ravi	Kumar	Sharma	20	M	9876543210
101	Ravi	Kumar	Sharma	20	M	9123456789
102	Sita	Devi	Reddy	21	F	9876000001

Option2

Roll no	Firs tname	Middle name	Last name	age	gender	phoneno1	phoneno2
101	Ravi	Kumar	Sharma	20	M	9876543210	9123456789
102	Sita	Devi	Reddy	21	F	9876000001	NULL

Option3:

rollno	phoneno
101	9876543210
101	9123456789
102	9876000001

Example Relation (with multi-valued attribute):

dept(dname, dno, mgno, location)

ex2: Consider a relation with multivalued attribute:

dept (dname, dno, mgrno, location)

research	5	333	blore, Delhi, Hyd
Admin	4	222	blore, chennai
Sevile	1	777	hyd, chennai

There are 3 solutions to deal with multivalued attribute.

solution1: create separate tuples for each location. But here it will increase redundancy.

Dno	Dname	Mgrno	location
5	Research	333	Hyd
5	Research	333	Blore
5	Research	333	Delhi
	:		

solution2: If a limit is set for the multi valued attribute e.g., here 3. Then the location can be decomposed as location1, location2, location3. Here if a dept does not have 3 locations then there will be NULL.

Dno	Dname	mgrno	location1	location2	location3
5	Research	333	Hyd	Blore	Delhi
4	admin	222	blore	chennai	NULL
1	Sevile	777	Hyd	chennai	NULL.

solution3: Create a separate relation for the multivalued attribute & primary key,

ex

dno	dname	mgrno
5	Research	333
4	Admin	222
1	Sevile	777

dno	location
5	Hyd
5	Blore
5	Delhi
4	Chennai
4	Blore
1	Hyd
1	Chennai

2ND NORMAL FORM:

==========

A relation schema is in 2NF if every non-prime attribute A in R is fully functional dependent on its primary key. So in 2NF the partial functional dependencies are to be removed.

A relation R is in 2NF if it is in 1NF and there is no partial dependency exist.

ex1: Consider a relation:

emp-proj (eno, pno, hours, ename, pname, plocation)

here {eno, pno} is a composite Primary key.

The functional Dependencies are:

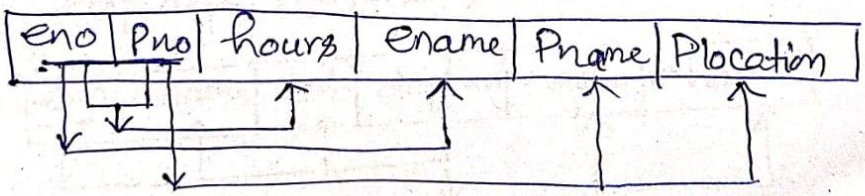
{eno, pno} → {hours}

Here hours is fully functional Dependent.

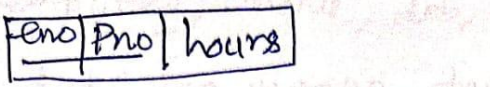
{eno, Pno} → {ename}

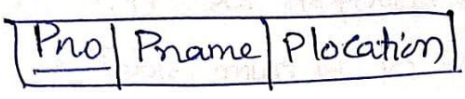
Here ename is partial functional Dependent.

{eno, Pno} → {Pname, Plocation} is also have Partial functional Dependency.



So the relation need to be decomposed in 2NF as follows:





3RD NORMAL FORM:

==========

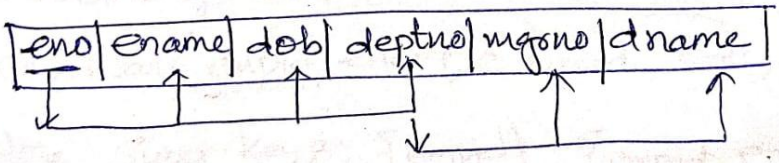
In the 3rd normal for the transitive dependencies has to be identified as it have to be removed.

If A-> B and B-> C then A -> C.

A relation R is in 3NF if it is in 2NF and there is no transitive dependency exist.

ex:

emp_dept (eno, ename, dob, deptno, mgrno, dname)



Here we can normalize by decomposing the relation into 2 relation as:

emp(eno, ename, dob, dno)

dept(dno, mgrno, dname)

A 3NF is defined as:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

Prime attribute: An attribute that is a part of one of the candidate keys is known as prime attribute.

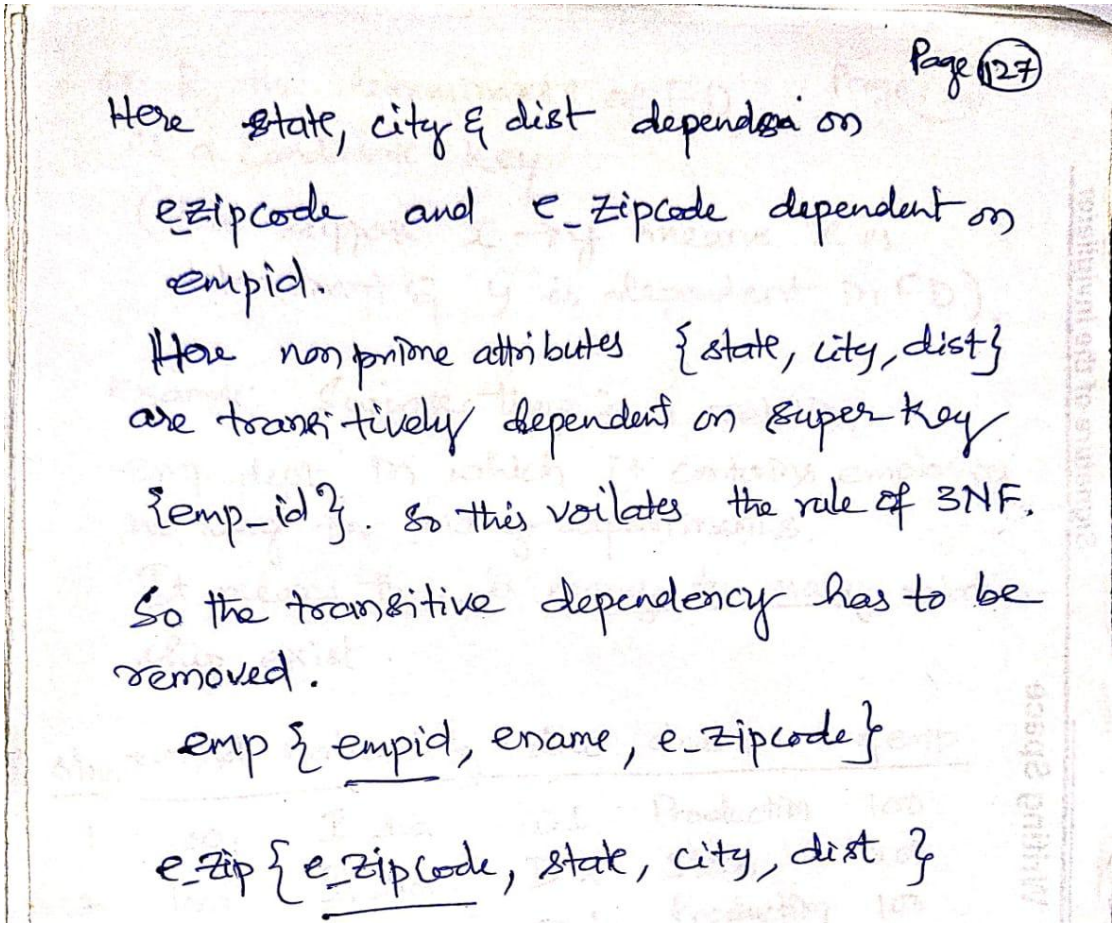
Non Prime attribute: An attribute that is not part of any candidate key is known as Non-prime attribute.

ex: Suppose a company wants to store the address of employees:

emp_detail(empid, ename, e_zipcode, state, dist, city)

Here Super keys: {empid}, {empid, ename}, {empid, ename, e_zipcode} and so on.

Candidate key: {empid}



BOYSE CODE NORMAL FORM:

==========

A stronger definition of 3NF is given by Boyse Code Normal Form or BCNF. So it is also known as 3.5NF.
A relation R is said to be in BCNF if and only if it is in 3NF and for all functional dependencies specified on R the determinant is a candidate key.

(Here suppose X->Y means X is determinant variable and Y is dependent variable.
Here we need to understand super key and candidate key first:
Super key:
A super key is any set of columns that can uniquely identify a row, but it is not necessarily minimal.

All candidate keys are super keys, but not all super keys are candidate keys. For instance, the combination {StudentID, Name} is a super key, but it is not a candidate

key because StudentID alone is sufficient for uniqueness, violating the minimality rule.

example: Suppose there is a relation emp_dept in which it contains employees working for many departments.
It means there is many to many relationship exist.

Slno	empid	nationality	dno	dname	no.of emp
1	1001	India	D1	Production	100
2	1001	India	D2	Stores	200
3	1002	America	D1	Production	100
4	1002	America	D3	Purchase	500

Here the functional dependencies:

{empid} → {nationality}

{dno} → {dname, no.of emp}

Candidate Key: {empid, dno}

Here table is not in BCNF because neither empid nor dno is alone keys.

To make the table in BCNF we need to divide into 3 tables.

R_1 {empid, nationality}

R_2 { dno, dname, no.of emp }

R_3 { empid, dno }

Now candidate keys are:

R_1 : empid

R_2 : dno

R_3 : empid, dno

Now it is in BCNF as because in FDs the left side part is a key.

What is a candidate key?

A candidate key is a column or set of columns in a table that can uniquely identify any database record without referring to any other data.

Each table may have one or more candidate keys, but one candidate key is unique, and it is called the primary key.

What is a non-key attribute?

It can not be used to identify a record uniquely. ex: name, age

Candidate key X Primary key

There can be multiple candidate keys in a table. Each candidate key can qualify as primary key.

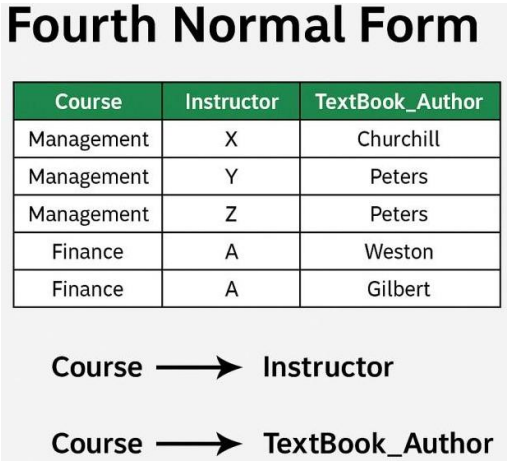
Primary key: It is a column or a combination of columns that uniquely identifies a record.

Super key X Candidate key

Super key: A super key is a set of attributes within a table whose values

4th Normal Form in DBMS:

Fourth Normal Form (4NF) is a higher level of normalization in relational database design, which deals with **multivalued dependencies** (MVDs).



Multivalued Dependency

A multivalued dependency occurs in a relation when one attribute determines multiple independent values of another attribute, independent of other attributes. A multivalued dependency always requires at least three attributes because it consists of at least two attributes that are dependent on a third.

For a dependency A -> B, if for a single value of A, multiple values of B exist, then the table may have a multi-valued dependency.

Example: A course can have multiple instructors, a course can also have multiple textbook authors but instructors and authors are independent of each other. This creates two independent multi-valued dependencies:

Course ->-> Instructor
Course ->-> TextBook_Author

If stored in the same table, this creates redundant combinations and data anomalies.

A Multi Valued Dependencies can be two types:
1) Trivial MVD
2) Non-Trivial MVD

1. Trivial MVD

Definition: An MVD $X \twoheadrightarrow Y$ is trivial if:
1. $Y \subseteq X$, or
2. $X \cup Y = R$ (i.e., together they cover all attributes of the relation).

Example Relation: R(A, B, C)
Case 1: $Y \subseteq X$
MVD: $A \twoheadrightarrow A$
Here, Y = A which is already part of X = A.
This is trivial.
Case 2: $X \cup Y = R$
MVD: $A \twoheadrightarrow (B, C)$
Here, $X \cup Y = (A, B, C) = R$.
This is trivial.

2. Non-Trivial MVD

Definition: An MVD $X \twoheadrightarrow Y$ is non-trivial if:
1. $Y \not\subseteq X$, and
2. $X \cup Y \neq R$

Example Relation: R(Student, Hobby, Skill)
Case: Student $\twoheadrightarrow$ Hobby
- Y = Hobby is not part of X = Student
- $X \cup Y$ = (Student, Hobby) $\neq R$

Hence, this is non-trivial.
Similarly, Student $\twoheadrightarrow$ Skill is also non-trivial.
These non-trivial MVDs cause redundancy, which is removed by decomposition into 4NF.

Example2: (Non-Trivial)
Example: A course can have multiple instructors, a course can also have multiple textbook authors but instructors and authors are independent of each other. This creates two independent multi-valued dependencies:

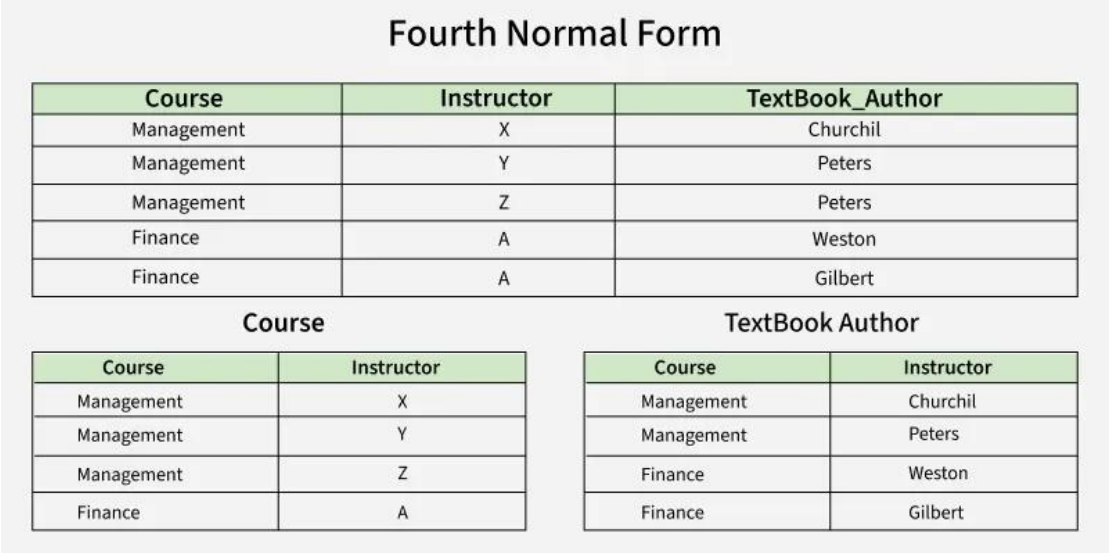
```
Course ->-> Instructor
Course ->-> TextBook_Author
```

If stored in the same table, this creates redundant combinations and data anomalies. A multi-valued dependency is a generalization of a functional dependency, but they are not the same.

Fourth Normal Form (4NF)
A relation R is in 4NF if it is already in 1NF,2NF,3NF, BCNF and there is no multi-valued dependencies.

The Fourth Normal Form (4NF) is a level of database normalization where there are no non-trivial multi-valued dependencies other than a candidate key.

It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multi-valued dependency. It is an extension of Boyce-Codd Normal Form (BCNF) and ensures that a relation does not contain multiple independent one-to-many relationships within a single table.



Course	Instructor	TextBook_Author
Management	X	Churchil
Management	Y	Peters
Management	Z	Peters
Finance	A	Weston
Finance	A	Gilbert

Course

Course	Instructor
Management	X
Management	Y
Management	Z
Finance	A

TextBook Author

Course	Instructor
Management	Churchil
Management	Peters
Finance	Weston
Finance	Gilbert

A relation R is in 4NF if and only if the following conditions are satisfied:

1. It should be in the Boyce-Codd Normal Form (BCNF).
2. The table should not have any Multi-valued Dependency.

Fifth Normal Form (5NF)

5NF is one of the highest levels of normalization, also known as Project-Join Normal Form (PJNF).

A table is said to be in 5NF if:

- It is already in 4NF
- It cannot be broken down into smaller tables without losing data.

In other words, there should be no join dependency left that can cause redundancy.

The main goal of 5NF is to break down a table into the smallest possible pieces while making sure that:
You can still reconstruct the original data without loss, and
No unnecessary repetition of data occurs.

Join Dependency (JD)

A join dependency, denoted as JD(R1, R2, ..., Rn), specifies that a relation R can be reconstructed by joining its projections R1, R2, ..., Rn.

- **Trivial JD: One of Ri = R**
- **Non-trivial JD: All Ri are proper subsets of R.**

EXAMPLE:
Consider relation R(Course, Instructor, Textbook):

- **A course can have multiple instructors.**
- **A course can require multiple textbooks.**
- **Each instructor can recommend different textbooks.**

If stored as one table, it may create redundant combinations (Cartesian product problem).
To solve this, we decompose the relation into three projections:
1. R1(Course, Instructor)
2. R2(Course, Textbook)
3. R3(Instructor, Textbook)

By joining R1, R2, and R3, we can reconstruct the original relation without redundancy. This decomposition is an example of achieving 5NF.

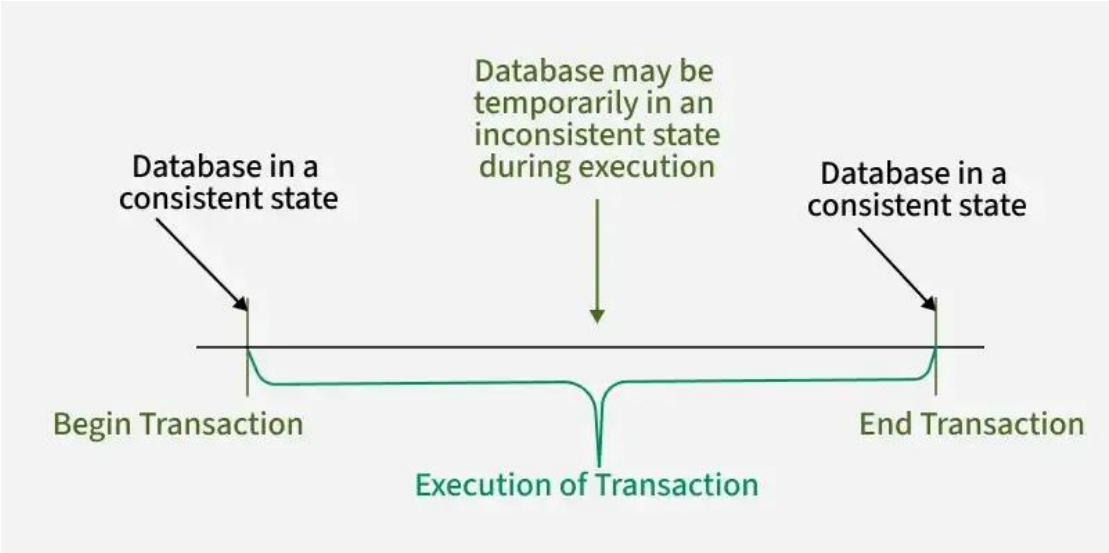
Transaction Concept (UNIT-IV)

A transaction in a Database Management System (DBMS) is a single logical unit of work that accesses and possibly modifies the contents of a database.

A transaction refers to a sequence of one or more operations (such as read, write, update, or delete) performed on the database as a single logical unit of work.

- A transaction ensures that either all the operations are successfully executed (committed) or none of them take effect (rolled back).

- Transactions are designed to maintain the integrity, consistency and reliability of the database, even in the case of system failures or concurrent access.



Example: Let's consider an online banking application:

Transaction: When a user performs a **money transfer**, several operations occur, such as:

- **Reading** the account balance of the sender.
- **Writing** the deducted amount from the sender's account.
- **Writing** the added amount to the recipient's account.

In a **transaction**, all these steps should either complete successfully or, if any error occurs, the database should **rollback** to its previous state, ensuring no partial data is written to the system.

Transaction Properties (ACID)

ACID properties are fundamental guarantees provided by a DBMS for reliable transaction processing:

Atomicity: Ensures that a transaction is treated as an indivisible unit. Either all operations within the transaction are completed successfully, or none are. If any part fails, the entire transaction is rolled back.

Consistency: **Guarantees that a transaction brings the database from one valid state to another valid state.** It ensures that all defined rules and constraints are maintained.

Isolation: **Ensures that concurrent transactions appear to execute in isolation from each other.** The effects of one transaction are not visible to other concurrent transactions until the first transaction is committed.

Durability: **Guarantees that once a transaction is committed, its changes are permanently stored in the database** and will survive any subsequent system failures.

Transaction State

A transaction progresses through several states during its lifecycle:
Active: The initial state where the transaction is executing its operations.
Partially Committed: The s**tate** after the final operation of the transaction has been executed, but before the changes are permanently saved to the database.
Committed: The state where all changes made by the transaction are permanently saved to the database.
Failed: The state where the transaction cannot proceed normally due to an error or a system failure.
Aborted: The state after a failed transaction has been rolled back, undoing all its changes to restore the database to its state before the transaction began.

Transaction Schedules

When multiple transaction requests are made at the same time, we need to decide their order of execution. Thus, a transaction schedule can be defined as a chronological order of execution of multiple transactions. Example: After a successful transfer, the updated balance remains safe despite a power failure.
There are broadly two types of transaction schedules discussed as follows:
i) Serial Schedule
In a serial schedule, transactions execute one at a time, ensuring database consistency but increasing waiting time and reducing system throughput. To improve throughput while maintaining consistency, concurrent schedules with strict rules are used, allowing safe simultaneous execution of transactions.
ii) Non-Serial Schedule
Non-serial schedule is a type of transaction schedule where multiple transactions are executed concurrently, interleaving their operations, instead of running one after another. It improves system efficiency but requires concurrency control to maintain database consistency.