



## Lecture Notes

Name of the Faculty	<b>KISHORE K</b>
Department	<b>CSE(Data Science)</b>
Year & Semester	<b>B.Tech-II &amp; I Sem</b>
Subject Name	<b>Data Base Management Systems</b>

**DEPARTMENT OF CSE (DATA SCIENCE)**

**VIDYA JYOTHI INSTITUTE OF TECHNOLOGY**

(Approved by AICTE, New Delhi, India & Affiliated to JNTUH, Hyderabad)

**AZIZ NAGAR, CB (PO), HYDERABAD-500 075**

***Introduction to Database System Concepts:***

*Database-System Applications, Purpose of Database Systems, View of data, Database Language, Database Architecture, Database Users and Administrators.*

***Introduction to the Relation Models and Database Design using ER Model:*** *Overview of the Design Process, The Entity-Relationship Model, Constraints, Entity-Relationship Diagrams, Reduction to Relational Schemas, Entity-Relationship Design Issues, Extended E-R Feature, Structure of relational databases , database schema.*

## Introduction

**Data:** data is a collection of raw facts and figures.

**DataBase:** database is a collection of interrelated data.

**DataBase-Management System (DBMS)** is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the **database**, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

## Database-System Applications

Databases are widely used. Here are some representative applications:

### ***1. Enterprise Information***

- ***Sales:*** For customer, product, and purchase information.
- ***Accounting:*** For payments, receipts, account balances, assets and other accounting information.

- **Human resources:** For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
- **Manufacturing:** For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.
- **Online retailers:** For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations.

## 2. Banking and Finance

- **Banking:** For customer information, accounts, loans, and banking transactions.
- **Credit card transactions:** For purchases on credit cards and generation of monthly statements.
- **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.

3. **Universities:** For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).

4. **Airlines:** For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.

5. **Telecommunication:** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

## Purpose of Database Systems

Database systems arose in response to early methods of computerized management of commercial data. As an example of such methods, typical of the 1960s, consider part of a university organization that, among other data, keeps information about all instructors, students, departments, and course offerings. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including programs to:

- Add new students, instructors, and courses
- Register students for courses and generate class rosters
- Assign grades to students, compute grade point averages (GPA), and generate transcripts

System programmers wrote these application programs to meet the needs of the university.

New application programs are added to the system as the need arises. For example, suppose that a university decides to create a new major (say, computer science). As a result, the university creates a new department and creates new permanent files (or adds information to existing files) to record information about all the instructors in the department, students in that major, course offerings, degree requirements, etc. The university may have to write new application programs to deal with rules specific to the new major. New application programs may also have to be written to handle new rules in the university. Thus, as time goes by, the system acquires more files and more application programs.

This typical **file-processing system** is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMSs) were introduced, organizations usually stored information in such systems.

Keeping organizational information in a file-processing system has a number of major disadvantages:

1. **Data Redundancy and Inconsistency.** Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). For example, if a student has a double major (say, music and mathematics) the address and telephone number of that student may appear in a file that consists of student records of students in the Music department and in a file that consists of student records of students in the Mathematics department. This redundancy leads to higher storage and access cost. In addition, it may lead to **data inconsistency**; that is, the various copies of the same data may no longer agree. For example, a changed student address may be reflected in the Music department records but not elsewhere in the system.
2. **Difficulty in Accessing Data.** Suppose that one of the university clerks needs to find out the names of all students who live within a particular postal-code area. The clerk asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of *all* students. The university clerk has now two choices: either obtain the list of all students and extract the needed information

manually or ask a programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same clerk needs to trim that list to include only those students who have taken at least 60 credit hours. As expected, a program to generate such a list does not exist. Again, the clerk has the preceding two options, neither of which is satisfactory. The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

3. **Data Isolation.** Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
4. **Integrity Problems.** The data values stored in the database must satisfy certain types of **consistency constraints**. Suppose the university maintains an account for each department, and records the balance amount in each account. Suppose also that the university requires that the account balance of a department may never fall below zero. Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.
5. **Atomicity Problems.** A computer system, like any other device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer \$500 from the account balance of department *A* to the account balance of department *B*. If a system failure occurs during the execution of the program, it is possible that the \$500 was removed from the balance of department *A* but was not credited to the balance of department *B*, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be *atomic*—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.
6. **Concurrent-access Anomalies.** For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. Indeed, today, the largest Internet retailers may have millions of accesses per day to their data by shoppers. In such an environment, interaction of concurrent updates is possible and may result in inconsistent data.

Consider department A, with an account balance of \$10,000. If two department clerks debit the account balance (by say \$500 and \$100, respectively) of department A at almost exactly the same time, the result of the concurrent executions may leave the budget in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value \$10,000, and write back \$9500 and \$9900, respectively. Depending on which one writes the value last, the account balance of department A may contain \$9500 or \$9900, rather than the correct value of \$9400. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

- 7. Security Problems.** Not every user of the database system should be able to access all the data. For example, in a university, payroll personnel need to see only that part of the database that has financial information. They do not need access to information about academic records. But, since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult.

#### Difference Between File System and DBMS

Basics	File System	DBMS
<b>Structure</b>	The file system is a way of arranging the files in a storage medium within a computer.	DBMS is software for managing the database.
<b>Data Redundancy</b>	Redundant data can be present in a file system.	In DBMS there is no redundant data.
<b>Backup and Recovery</b>	It doesn't provide Inbuilt mechanism for backup and recovery of data if it is lost.	It provides in house tools for backup and recovery of data even if it is lost.
<b>Query processing</b>	There is no efficient query processing in the file system.	Efficient query processing is there in DBMS.

<b>Basics</b>	<b>File System</b>	<b>DBMS</b>
<b>Consistency</b>	There is less data consistency in the file system.	There is more data consistency because of the process of normalization.
<b>Complexity</b>	It is less complex as compared to DBMS.	It has more complexity in handling as compared to the file system.
<b>Security Constraints</b>	File systems provide less security in comparison to DBMS.	DBMS has more security mechanisms as compared to file systems.
<b>Cost</b>	It is less expensive than DBMS.	It has a comparatively higher cost than a file system.
<b>Data Independence</b>	There is no data independence.	In DBMS data independence exists, mainly of two types: 1) Logical Data Independence. 2) Physical Data Independence.
<b>User Access</b>	Only one user can access data at a time.	Multiple users can access data at a time.
<b>Meaning</b>	The users are not required to write procedures.	The user has to write procedures for managing databases
<b>Sharing</b>	Data is distributed in many files. So, it is not easy to share data.	Due to centralized nature data sharing is easy
<b>Data Abstraction</b>	It give details of storage and representation of data	It hides the internal details of Database
<b>Integrity Constraints</b>	Integrity Constraints are difficult to implement	Integrity constraints are easy to implement



Basics	File System	DBMS
Attributes	To access data in a file , user requires attributes such as file name, file location.	No such attributes are required.
Example	Cobol, C++	Oracle, SQL Server, MySql

## View of Data

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an *abstract* view of the data. That is, the system hides certain details of how the data are stored and maintained.

### Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database.

Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

- 1. Physical level.** The lowest level of abstraction describes *how* the data are actually stored. The physical level describes complex low-level data structures in detail.
- 2. Logical level.** The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.
- 3. View level.** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to



simplify their interaction with the system. The system may provide many views for the same database.

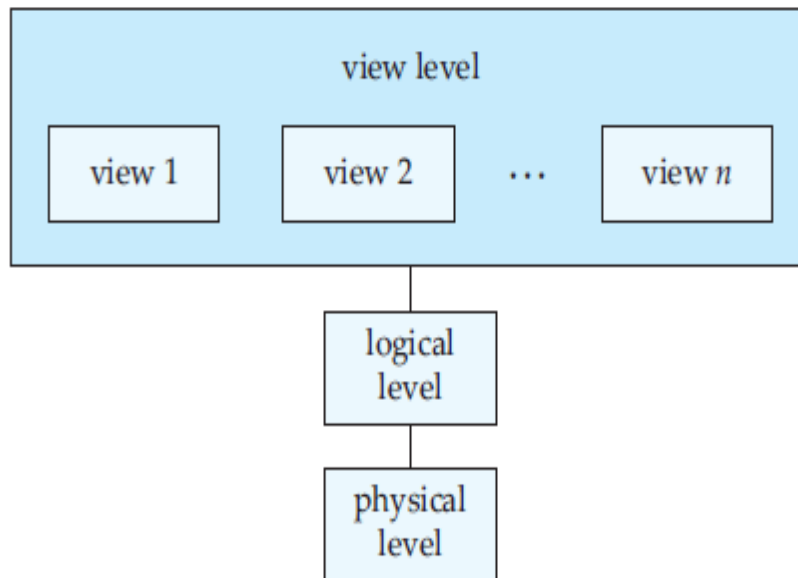


Figure 1.1 The three levels of data abstraction.

Figure 1.1 shows the relationship among the three levels of abstraction. An analogy to the concept of data types in programming languages may clarify the distinction among levels of abstraction. Many high-level programming languages support the notion of a structured type. For example, we may describe a record as follows:

```
type instructor = record  
  ID : char (5);  
  name : char (20);  
  dept name : char (20);  
  salary : numeric (8,2);  
end;
```

This code defines a new record type called *instructor* with four fields. Each field has a name and a type associated with it. A university organization may have several such record types, including

- *department*, with fields *dept name*, *building*, and *budget*
- *course*, with fields *course id*, *title*, *dept name*, and *credits*
- *student*, with fields *ID*, *name*, *dept name*, and *tot cred*

At the **physical level**, an *instructor*, *department*, or *student* record can be described as a block of consecutive storage locations. The compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the **logical level**, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

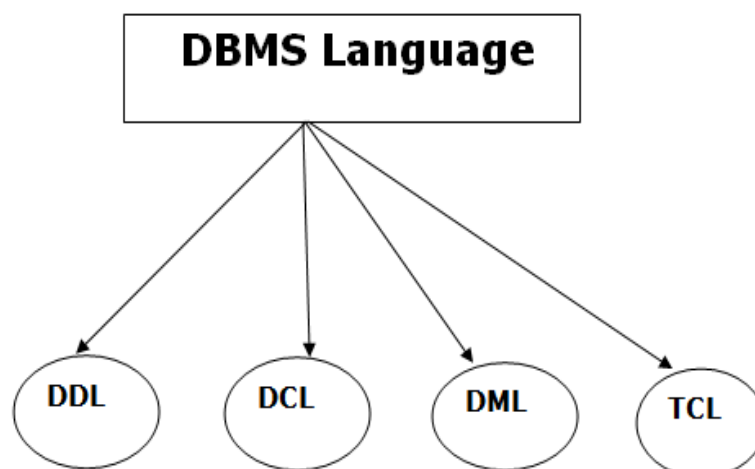
Finally, at the **view level**, computer users see a set of application programs that hide details of the data types. At the view level, several views of the database are defined, and a database user sees some or all of these views. In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database.

## Database Languages

A database system provides a

- Appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.

### Types of Database Languages



## 1. Data Definition Language (DDL)

- **DDL** stands for **Data Definition Language**. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

## 2. Data Manipulation Language (DML)

**DML** stands for **Data Manipulation Language**. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data.
- **Lock Table:** It controls concurrency.

### 3. Data Control Language (DCL)

- **DCL** stands for **Data Control Language**. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

### 4. Transaction Control Language (TCL)

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

## Database Architecture

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines. Database systems can also be designed to exploit parallel computer architectures. Distributed databases span multiple geographically separated machines.

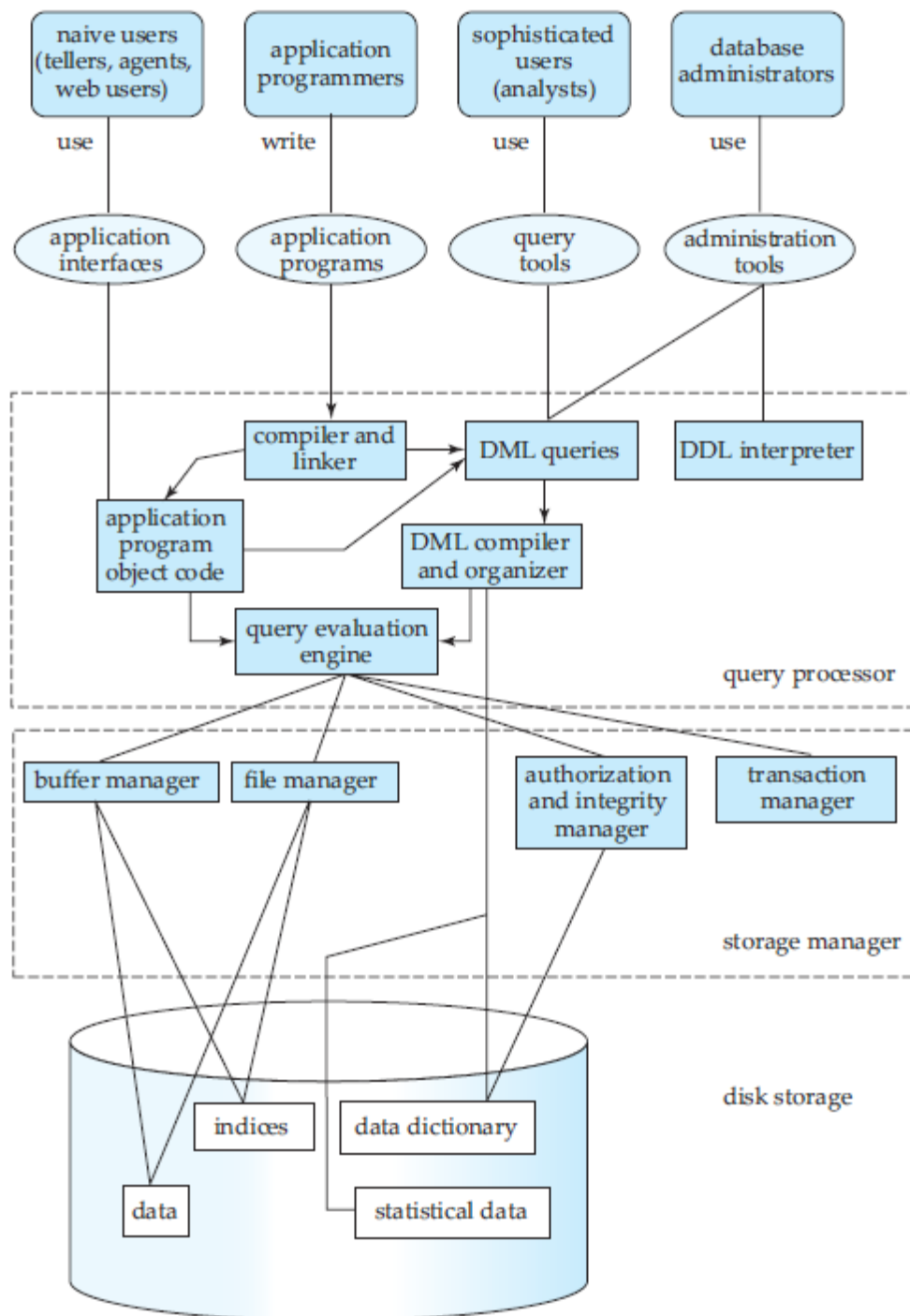


Figure 1.5 System structure.

**Three Parts that make up the Database System are:**

- Query Processor
- Storage Manager
- Disk Storage

The explanations for these are provided below:

## **1. Query Processor**

The query processing is handled by the query processor, as the name implies. It executes the user's query, to put it simply. In this way, the query processor aids the database system in making data access simple and easy. The query processor's primary duty is to successfully execute the query. The Query Processor transforms (or interprets) the user's application program-provided requests into instructions that a computer can understand.

### **Components of the Query Processor**

- **DDL Interpreter:**

Data Definition Language is what DDL stands for. As implied by the name, the DDL Interpreter interprets DDL statements like those used in schema definitions (such as create, remove, etc.). This interpretation yields a set of tables that include the meta-data (data of data) that is kept in the data dictionary. Metadata may be stored in a data dictionary. In essence, it is a part of the disc storage that will be covered in a later section of this article.

- **DML Compiler:**

Compiler for DML Data Manipulation Language is what DML stands for. In keeping with its name, the DML Compiler converts DML statements like select, update, and delete into low-level instructions or simply machine-readable object code, to enable execution. The optimization of queries is another function of the DML compiler. Since a single question can typically be translated into a number of evaluation plans. As a result, some optimization is needed to select the evaluation plan with the lowest cost out of all the options. This process, known as query optimization, is exclusively carried out by the DML compiler. Simply put, query optimization determines the most effective technique to carry out a query.

- **Compiler and Linker:**

Before the query evaluation, the embedded DML commands in the application program (such as SELECT, FROM, etc., in SQL) must be pre-compiled into standard procedural calls (program instructions that the host language can understand).

Therefore, the DML statements which are embedded in an application program must be converted into routine calls by the Embedded DML Pre-compiler.

- **Query Evaluation Engine:**

It starts by taking the evaluation plan for the question, runs it, and then returns the result. Simply said, the query evaluation engine evaluates the SQL commands used to access the database's contents before returning the result of the query. In a nutshell, it is in charge of analyzing the queries and running the object code that the DML Compiler produces. Apache Drill, Presto, and other Query Evaluation Engines are a few examples.

## **2. Storage Manager:**

An application called Storage Manager acts as a conduit between the queries made and the data kept in the database. Another name for it is Database Control System. By applying the restrictions and running the DCL instructions, it keeps the database's consistency and integrity. It is in charge of retrieving, storing, updating, and removing data from the database.

### **Components of Storage Manager**

Following are the components of Storage Manager:

- **Integrity Manager:**

Whenever there is any change in the database, the Integrity manager will manage the integrity constraints.

- **Authorization Manager:**

Authorization manager verifies the user that he is valid and authenticated for the specific query or request.

- **File Manager:**

All the files and data structure of the database are managed by this component.

- **Transaction Manager:**

It is responsible for making the database consistent before and after the transactions. Concurrent processes are generally controlled by this component.



- **Buffer Manager:**

The transfer of data between primary and main memory and managing the cache memory is done by the buffer manager.

### 3. Disk Storage

A DBMS can use various kinds of Data Structures as a part of physical system implementation in the form of disk storage.

#### Components of Disk Storage

Following are the components of Disk Manager:

- **Data Dictionary:**

It contains the metadata (data of data), which means each object of the database has some information about its structure. So, it creates a repository which contains the details about the structure of the database object.

- **Data Files:**

This component stores the data in the files.

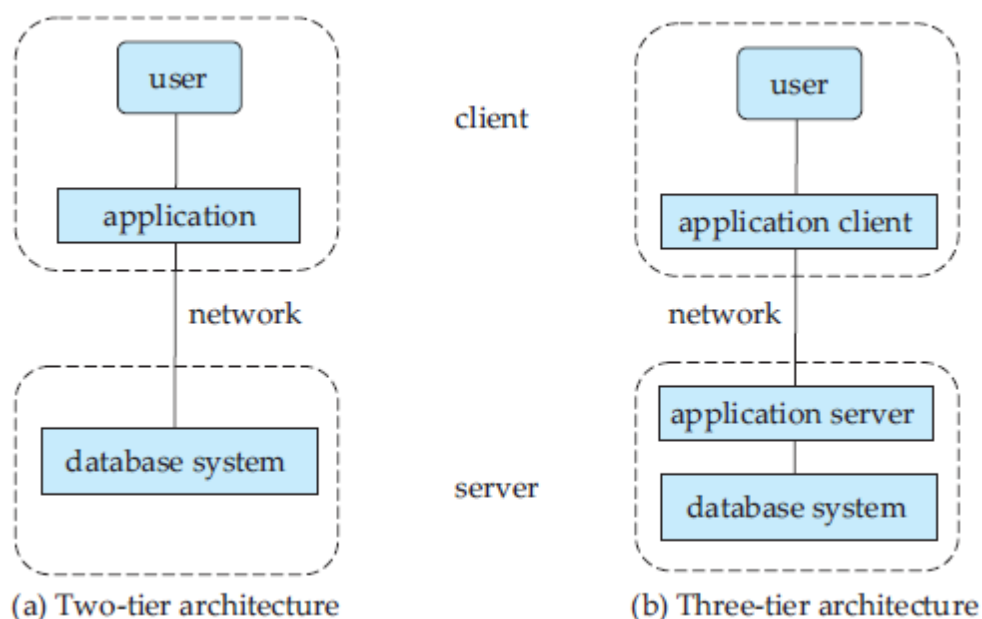
- **Indices:**

These indices are used to access and retrieve the data in a very fast and efficient way.

Database applications are usually partitioned into two or three parts

#### (a) Two-Tier Architecture

#### (b) Three-Tier Architecture



In a **two-tier architecture**, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.

In contrast, in a **three-tier architecture**, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an **application server**, usually through a forms interface.

The application server in turn communicates with a database system to access data. The **business logic** of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on the WorldWideWeb.

## Database Users and Administrators

A primary goal of a database system is to retrieve information from and store new information into the database. People who work with a database can be categorized as

1. **Database Users**
2. **Database Administrators.**

### 1. Database Users

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

- **Naïve users** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.
- **Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.
- **Sophisticated users** interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software. Analysts who submit queries to explore data in the database fall in this category.

- **Specialized users** are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

## 2. Database Administrator

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a **database administrator (DBA)**. The functions of a DBA include:

- **Schema definition.** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
- **Storage structure and access-method definition.**
- **Schema and physical-organization modification.** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.
- **Granting of authorization for data access.** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.
- **Routine maintenance.** Examples of the database administrator's routine maintenance activities are:
  - ✓ Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
  - ✓ Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
  - ✓ Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

*Introduction to the Relation Models and Database Design using ER Model: Overview of the Design Process, The Entity-Relationship Model, Constraints, Entity-Relationship Diagrams, Reduction to Relational Schemas, Entity-Relationship Design Issues, Extended E-R Feature, Structure of relational databases , database schema.*

## Introduction to the Relational Model

### Overview of the Design Process

The task of creating a database application is a complex one, involving design of the database schema, design of the programs that access and update the data, and design of a security scheme to control access to data. The needs of the users play a central role in the design process.

It involves a meticulous process that unfolds in three distinct phases:

1. **Conceptual,**
2. **Logical, and**
3. **Physical Database Design.**

These levels of design are crucial in creating a database that not only captures the essence of the data but also ensures its integrity, efficiency, and security.

### 1. Conceptual Database Design

Conceptual database design is the highest level of abstraction in the database design process. At this stage, designers focus on understanding the problem domain and defining the overall structure of the database without getting into technical implementation details. The primary goal is to create a clear and comprehensive representation of the data and its relationships.

**Problem Description:** Imagine a university wants to create a database to manage student information. In the conceptual design phase, the primary concern is identifying the main entities and their relationships within the university context. Key entities might include students, courses, instructors, and departments. Relationships could include a student enrolling in courses, instructors teaching courses, and departments managing courses.

#### **Example:**

- **Entities:** Student, Course, Instructor, Department
- **Relationships:** Student enrolls in Course, Instructor teaches Course, Department manages Course

## 2. Logical Database Design

Logical database design bridges the gap between the conceptual and physical levels. Here, designers translate the conceptual model into a more detailed representation, focusing on data structures, relationships, and constraints. The logical design is independent of any specific database management system (DBMS) and is often expressed using Entity-Relationship Diagrams (ERDs) or similar modeling techniques.

**Problem Description:** Continuing with our university example, in the logical design phase, you would define attributes for each entity and specify their data types, primary keys, and foreign keys. This stage also involves normalizing the data to eliminate redundancy and ensure data integrity.

**Example:**

- **Student Entity:**  
    **Attributes:** StudentID (Primary Key), FirstName, LastName, DateOfBirth
- **Course Entity:**  
    **Attributes:** CourseID (Primary Key), CourseName, Credits
- **Instructor Entity:**  
    **Attributes:** InstructorID (Primary Key), FirstName, LastName
- **Department Entity:**  
    **Attributes:** DepartmentID (Primary Key), DepartmentName

## 3. Physical Database Design

Physical database design is the most detailed and technical level of the database design process. At this stage, designers make decisions about how the logical design will be implemented on a specific DBMS. Considerations include indexing, storage, performance optimization, and security measures.

**Problem Description:** For our university database, in the physical design phase, you would determine which DBMS to use (e.g., MySQL, Oracle, PostgreSQL) and create the actual database schema. This involves specifying the exact table structures, data types, constraints, and indexes. It also includes decisions about data storage, partitioning, and access control.

**Example:**

- **Student Table** (MySQL Syntax):

```
CREATE TABLE Student (  
    StudentID INT PRIMARY KEY,  
    FirstName VARCHAR(50),  
    LastName VARCHAR(50),  
    DateOfBirth DATE  
);
```

- **Course Table:**

```
CREATE TABLE Course (  
    CourseID INT PRIMARY KEY,  
    CourseName VARCHAR(100),  
    Credits INT  
);
```

## The Entity-Relationship Model

The **Entity-Relationship (E-R)** data model was developed to facilitate database design by allowing specification of an *enterprise schema* that represents the overall logical structure of a database.

The E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema. Because of this usefulness, many database-design tools draw on concepts from the E-R model.

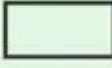




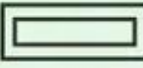
The Entity Relationship Diagram explains the relationship among the entities present in the database. ER models are used to model real-world objects like a person, a car, or a company and the relation between these real-world objects. In short, the ER Diagram is the structural format of the database.

### Why Use ER Diagrams In DBMS?

- ER diagrams represent the E-R model in a database, making them easy to convert into relations (tables).
- ER diagrams provide the purpose of real-world modeling of objects which makes them intently useful.
- ER diagrams require no technical knowledge and no hardware support.
- These diagrams are very easy to understand and easy to create even for a naive user.
- It gives a standard solution for visualizing the data logically.

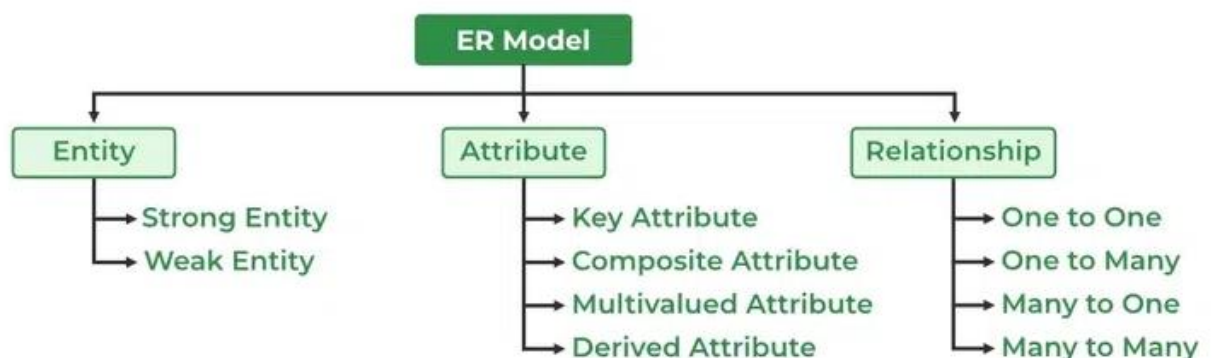
## Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which consists of these symbols:

Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

The E-R data model employs three basic concepts:

1. **Entity Sets,**
2. **Attributes and**
3. **Relationship Sets**





**Entity:** An **entity** is a “thing” or “object” in the real world that is distinguishable from all other objects.

For example, each person in a university is an entity. An entity has a set of properties, and the values for some set of properties may uniquely identify an entity. For instance, a person may have a **person id** property whose value uniquely identifies that person.

**Entity Set:** An **entity set** is a set of entities of the same type that share the same properties, or attributes.

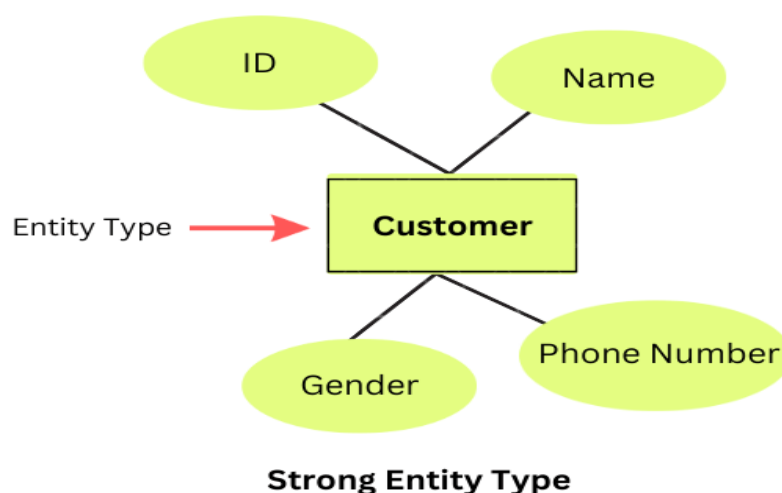
The set of all people who are instructors at a given university, for example, can be defined as the entity set **instructor**. Similarly, the entity set **student** might represent the set of all students in the university.

### Types of Entity

There are two types of entity:

#### 1. Strong Entity

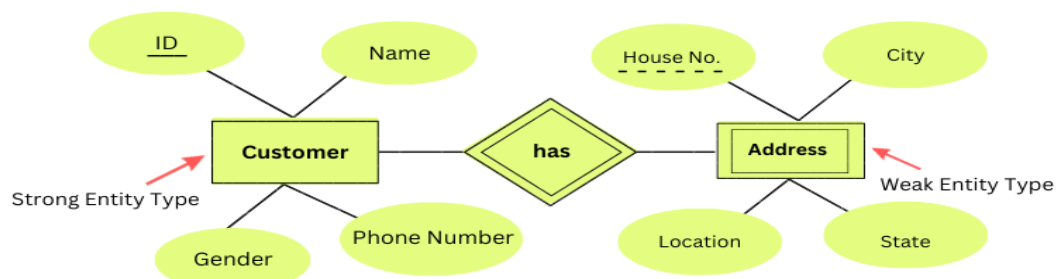
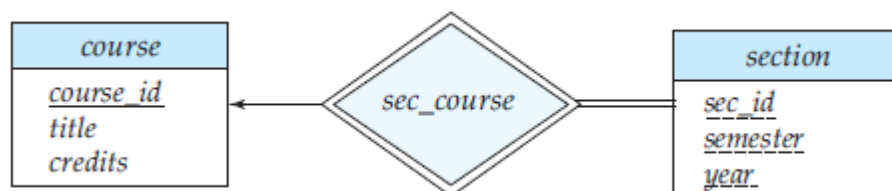
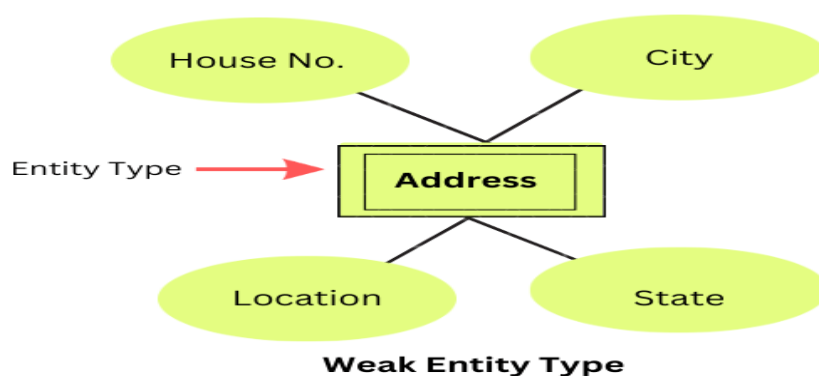
A Strong Entity is a type of entity that has a key Attribute. Strong Entity does not depend on other Entity in the Schema. It has a primary key that helps in identifying it uniquely and it is represented by a rectangle. These are called Strong Entity Types.



## 2. Weak Entity

A weak entity is an entity that cannot be uniquely identified by its own attributes alone; it depends on the existence of another entity, called the "owner" or "parent" entity.

A weak entity type is represented by a Double Rectangle. The participation of weak entity types is always total. The relationship between the weak entity type and its identifying strong entity type is called identifying relationship and it is represented by a double diamond.



Entity Relationship Diagram between Strong Entity Type & Weak Entity Type

### What is Attributes?

Attributes are the properties that define the entity type. For example, Roll\_No, Name, DOB, Age, Address, and Mobile\_No are the attributes that define entity type Student. In ER diagram, the attribute is represented by an oval.



### Types of Attributes

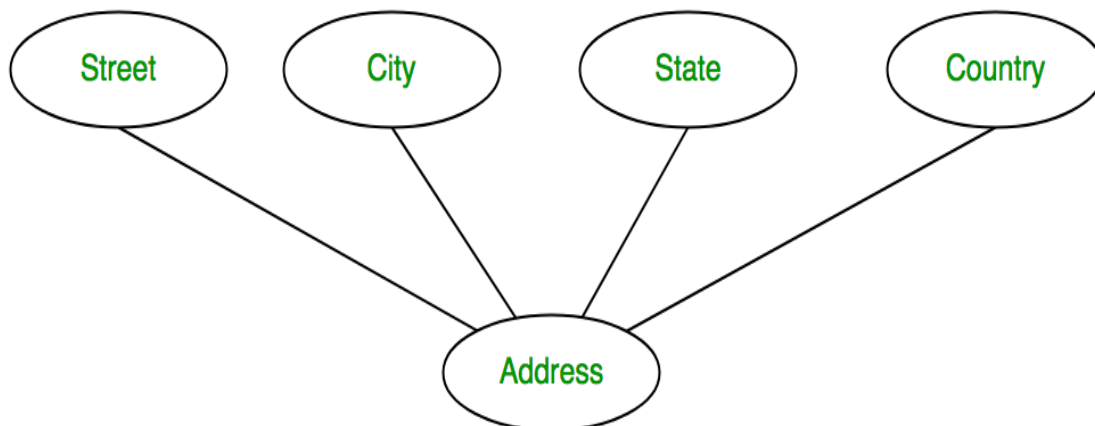
#### 1. Key Attribute

The attribute which **uniquely identifies each entity** in the entity set is called the key attribute. For example, Roll\_No will be unique for each student. In ER diagram, the key attribute is represented by an oval with underlying lines.



#### 2. Composite Attribute

An attribute **composed of many other attributes** is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country. In ER diagram, the composite attribute is represented by an oval comprising of ovals.



### 3. Multivalued Attribute

An attribute consisting of more than one value for a given entity. For example, Phone\_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval.

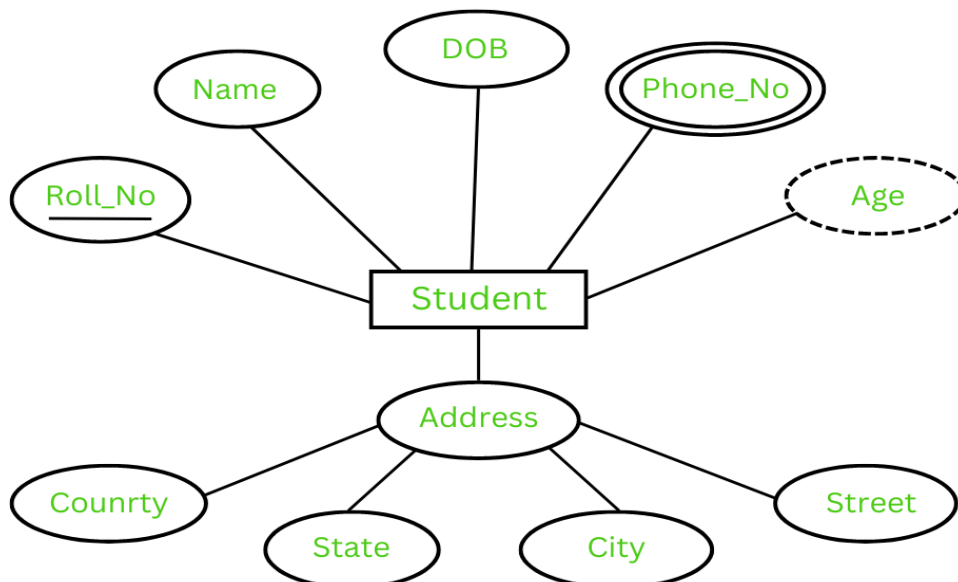


### 4. Derived Attribute

An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, the derived attribute is represented by a dashed oval.



The Complete Entity Type Student with its Attributes can be represented as:

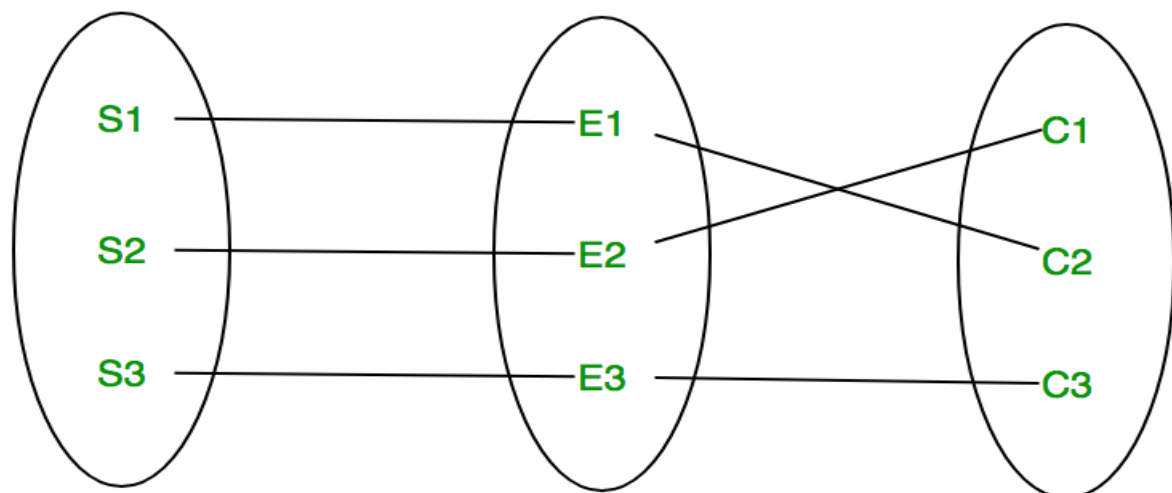


### Relationship Type and Relationship Set

A Relationship Type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, the relationship type is represented by a diamond and connecting the entities with lines.



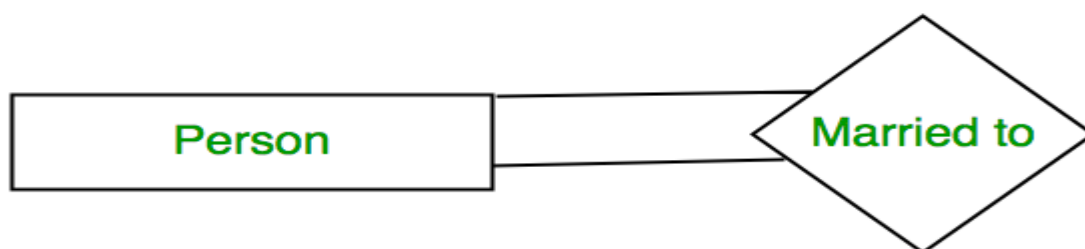
A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 as enrolled in C2, S2 as enrolled in C1, and S3 as registered in C3.



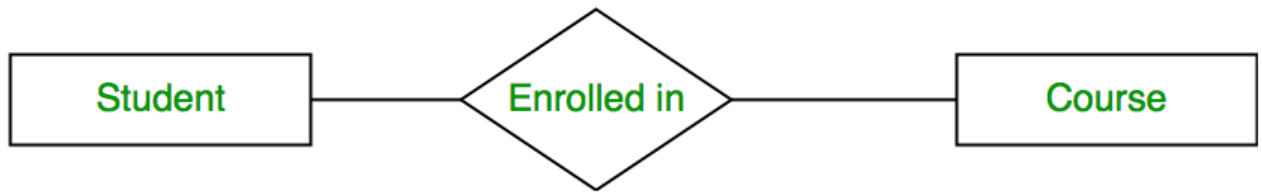
### Degree of a Relationship Set

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

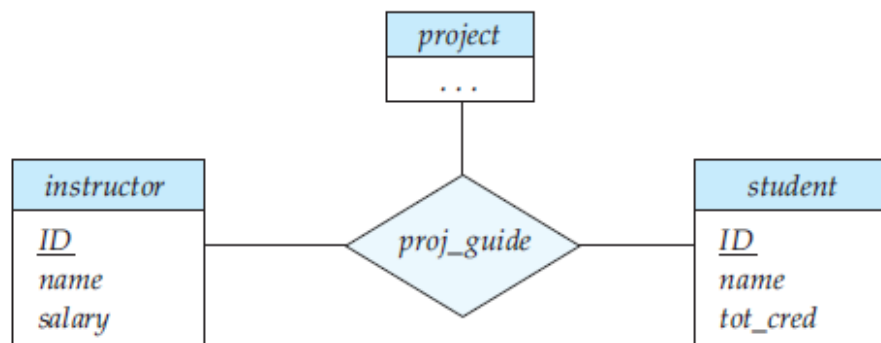
**1. Unary Relationship:** When there is only ONE entity set participating in a relation, the relationship is called a unary relationship. For example, one person is married to only one person.



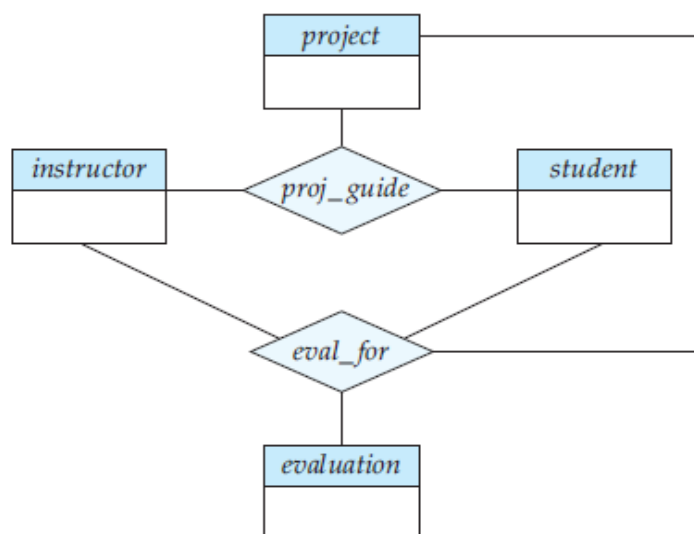
**2. Binary Relationship:** When there are TWO entities set participating in a relationship, the relationship is called a binary relationship. For example, a Student is enrolled in a Course.



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



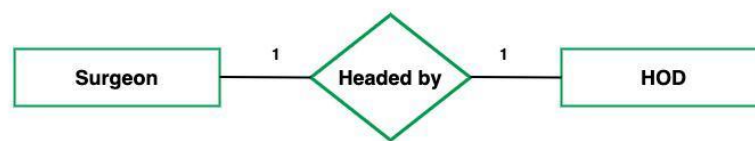
**4. N-ary Relationship:** When there are n entities set participating in a relationship, the relationship is called an n-ary relationship.



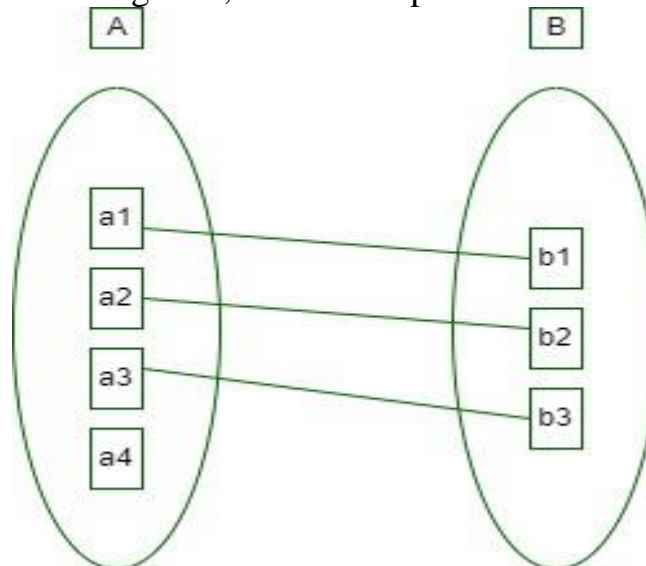
## What is Cardinality?

The number of times an entity of an entity set participates in a relationship set is known as cardinality. Cardinality can be of different types:

1. **One-to-One:** When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one. Let us assume that a male can marry one female and a female can marry one male. So the relationship will be one-to-one. The total number of tables that can be used in this is 2.

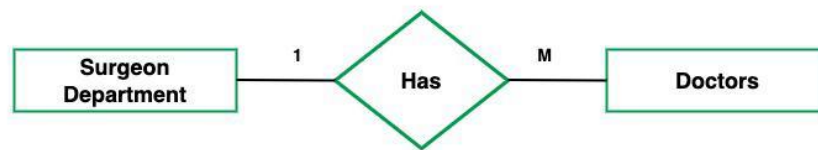


Using Sets, it can be represented as:

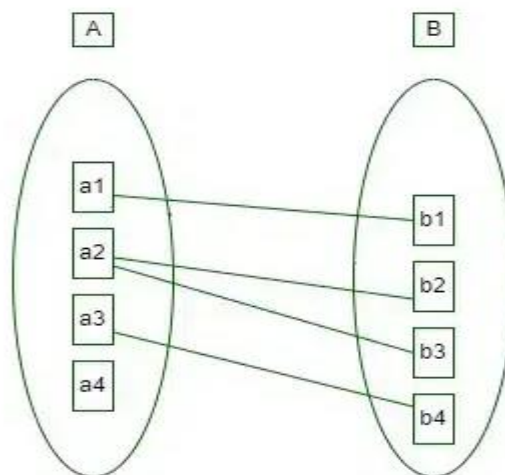


2. **One-to-Many:** In one-to-many mapping as well where each entity can be related to more than one entity and the total number of tables that can be used in this is 2. Let us assume that one surgeon department can accommodate many doctors. So the Cardinality will be 1 to M. It means one department has many Doctors. total number of tables that can used is 3.



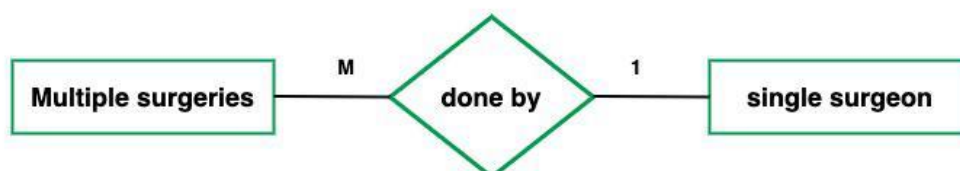


*Using sets, one-to-many cardinality can be represented as:*

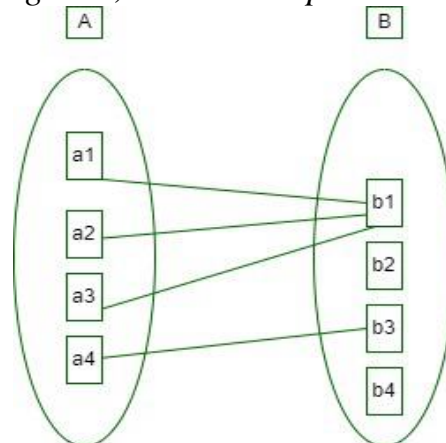


3. **Many-to-One:** When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be  $n$  to 1. It means that for one course there can be  $n$  students but for one student, there will be only one course.

The total number of tables that can be used in this is 3.



*Using Sets, it can be represented as:*



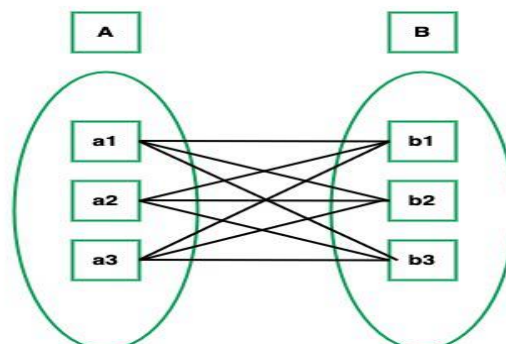
In this case, each student is taking only 1 course but 1 course has been taken by many students.

4. **Many-to-Many:** When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

The total number of tables that can be used in this is 3.



*Using Sets, it can be represented as:*



## Participation Constraint

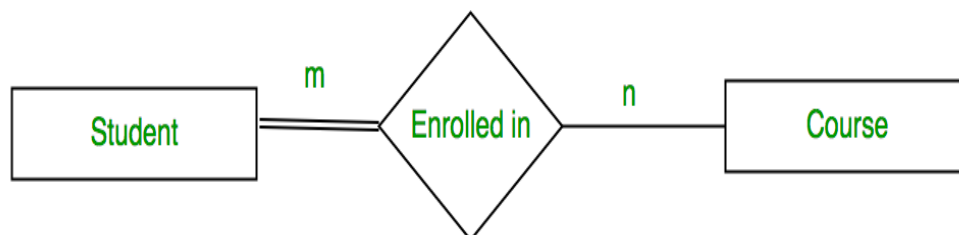
Participation Constraint is applied to the entity participating in the relationship set.

**1. Total Participation** – The participation of an entity set  $E$  in a relationship set  $R$  is said to be **total** if every entity in  $E$  participates in at least one relationship in  $R$ . If each student must enroll in a course, the participation of students will be total. Total participation is shown by a double line in the ER diagram.

**2. Partial Participation** – If only some entities in  $E$  participate in relationships in  $R$ , the participation of entity set  $E$  in relationship  $R$  is said to be **partial**.

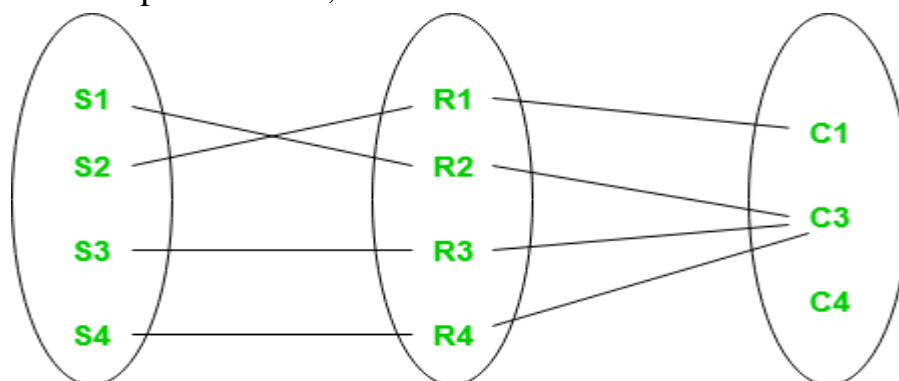
If some courses are not enrolled by any of the students, the participation in the course will be partial.

The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



### *Total Participation and Partial Participation*

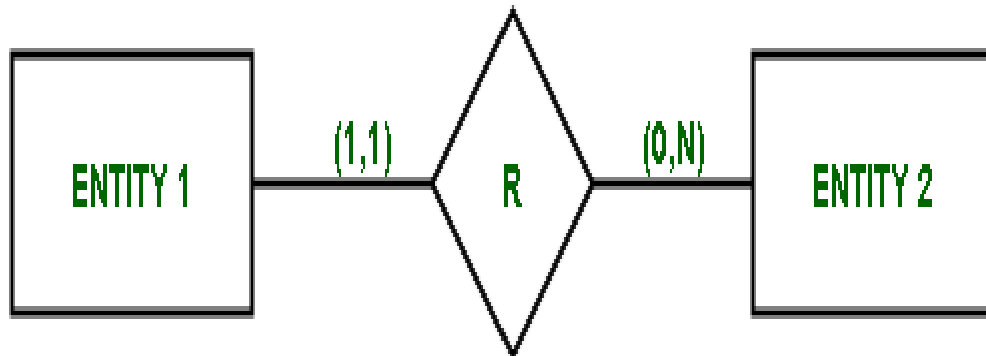
Using Set, it can be represented as,



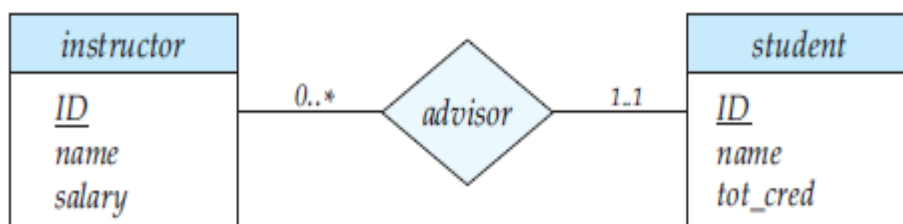
### *Set representation of Total Participation and Partial Participation*

Every student in the Student Entity set participates in a relationship but there exists a course C4 that is not taking part in the relationship.

**Structural Constraints:** Cardinality Ratios and Participation Constraints taken together are called Structural Constraints. The name constraints refer to the fact that such limitations must be imposed on the data, for the DBMS system to be consistent with the requirements.



The Structural constraints are represented by **Min-Max notation**. This is a pair of numbers  $(m, n)$  that appear on the connecting line between the entities and their relationships. The minimum number of times an entity can appear in a relation is represented by  $m$  whereas, the maximum time it is available is denoted by  $n$ . If  $m$  is 0 it signifies that the entity is participating in the relation partially, whereas, if  $m$  is either greater than or equal to 1, it denotes total participation of the entity.

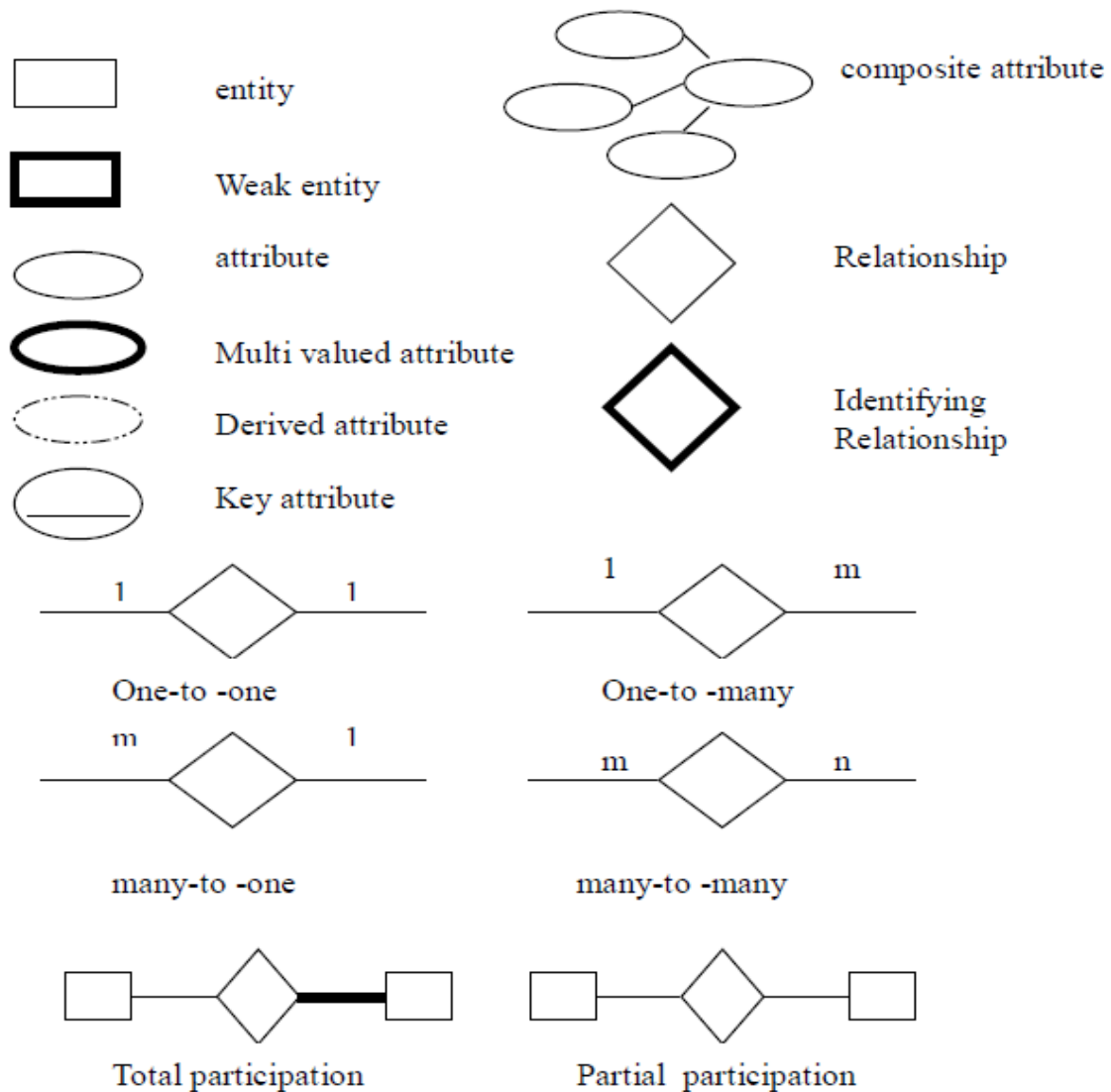


**Figure 7.10** Cardinality limits on relationship sets.

## Entity-Relationship Diagrams

The overall logical structure of a database represents graphically with the help of an ER diagrams.

**Symbols use ER- diagram:**



## Steps to Create an ERD (E-R Diagram)

Following are the steps to create an ERD



### E-R diagram for the University

In our university database, we have a constraint that each instructor must have exactly one associated department. As a result, there is a double line in Figure 7.15 between *instructor* and *inst\_dept*, indicating total participation of *instructor* in *inst\_dept*; that is, each instructor must be associated with a department. Further, there is an arrow from *inst\_dept* to *department*, indicating that each instructor can have at most one associated department.

Similarly, entity sets *course* and *student* have double lines to relationship sets *course\_dept* and *stud\_dept* respectively, as also entity set *section* to relationship set *sec\_time\_slot*. The first two relationships, in turn, have an arrow pointing to the other relationship, *department*, while the third relationship has an arrow pointing to *time\_slot*.

Further, Figure 7.15 shows that the relationship set *takes* has a descriptive attribute *grade*, and that each student has at most one advisor. The figure also shows that *section* is now a weak entity set, with attributes *sec\_id*, *semester*, and *year* forming the discriminator; *sec\_course* is the identifying relationship set relating weak entity set *section* to the strong entity set *course*.

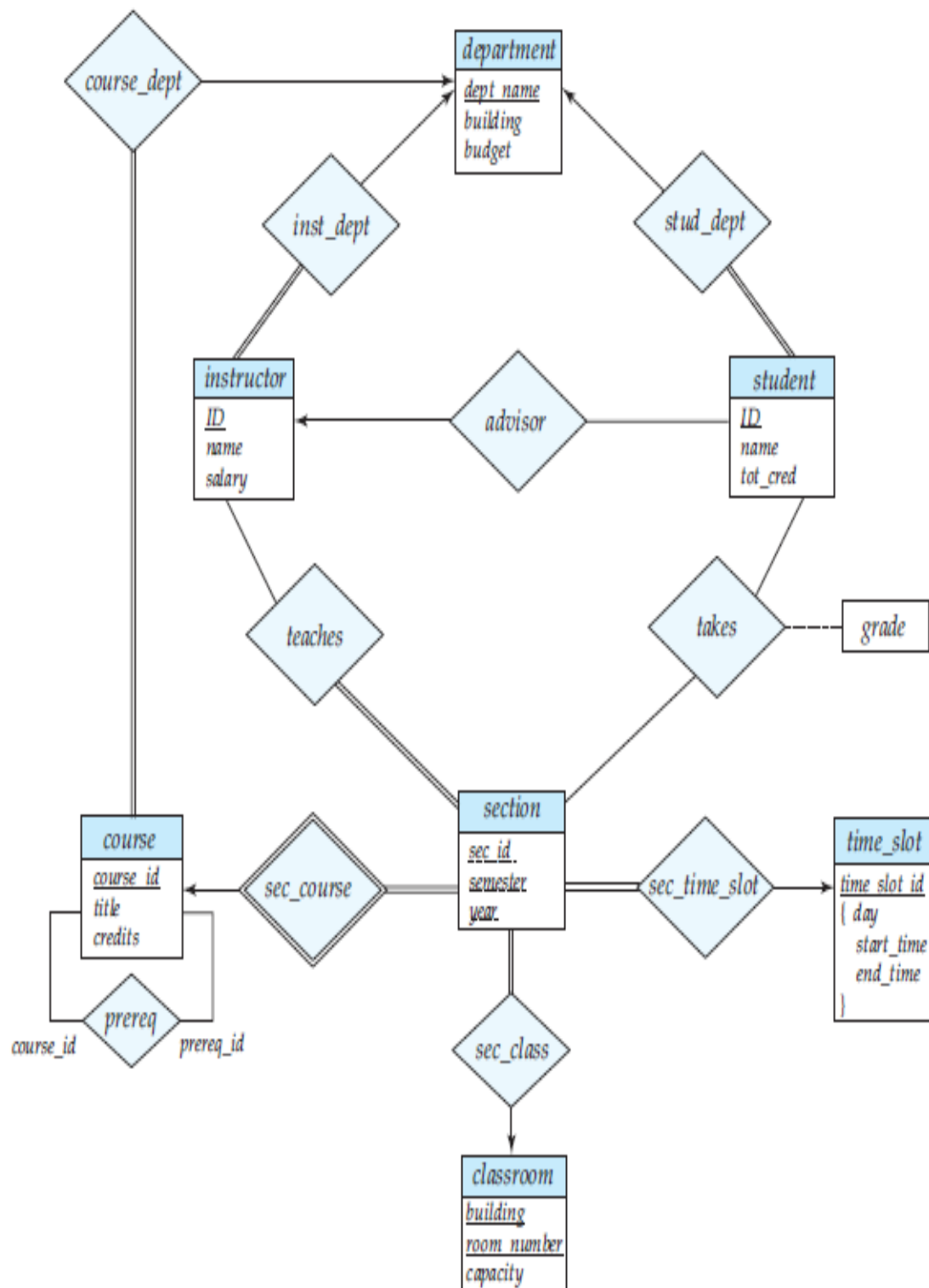
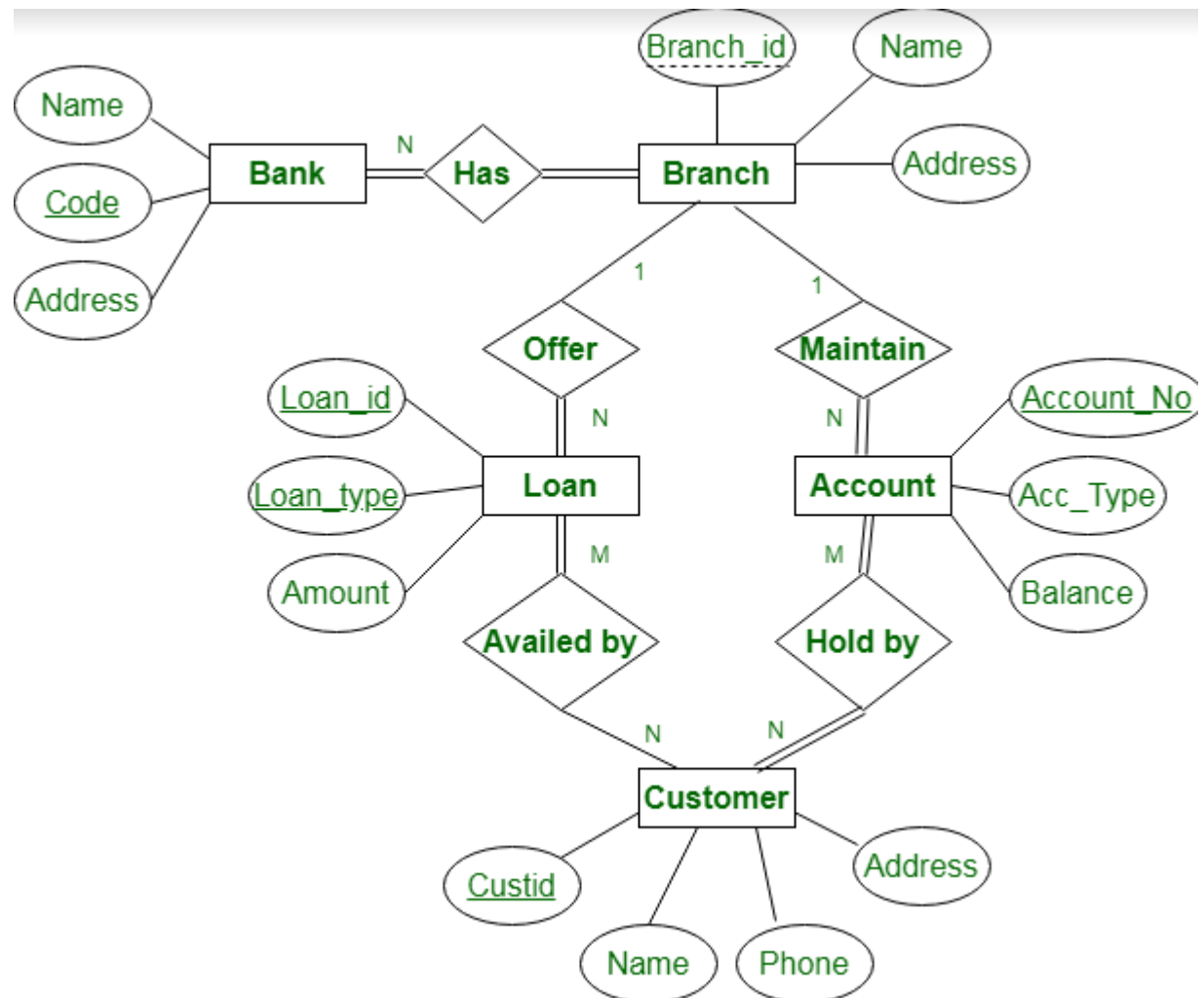


Figure 7.15 E-R diagram for a university enterprise.



**ER Diagram of Bank Management System :**

- Bank have Customer.
- Banks are identified by a name, code, address of main office.
- Banks have branches.
- Branches are identified by a branch\_no., branch\_name, address.
- Customers are identified by name, cust-id, phone number, address.
- Customer can have one or more accounts.
- Accounts are identified by account\_no., acc\_type, balance.
- Customer can avail loans.
- Loans are identified by loan\_id, loan\_type and amount.
- Account and loans are related to bank's branch.
- 



**Entities** and their **Attributes** are :

- **Bank Entity:** Attributes of Bank Entity are Bank Name, Code and Address. Code is Primary Key for Bank Entity.
- **Customer Entity:** Attributes of Customer Entity are Customer\_id, Name, Phone Number and Address. Customer\_id is Primary Key for Customer Entity.
- **Branch Entity:** Attributes of Branch Entity are Branch\_id, Name and Address. Branch\_id is Primary Key for Branch Entity.
- **Account Entity:** Attributes of Account Entity are Account\_number, Account\_Type and Balance. Account\_number is Primary Key for Account Entity.
- **Loan Entity:** Attributes of Loan Entity are Loan\_id, Loan\_Type and Amount. Loan\_id is Primary Key for Loan Entity.

## Reduction to Relational Schemas

We can represent a database that conforms to an E-R database schema by a collection of relation schemas. For each entity set and for each relationship set in the database design, there is a unique relation schema to which we assign the name of the corresponding entity set or relationship set.

Both the E-R model and the relational database model are abstract, logical representations of real-world enterprises. Because the two models employ similar design principles, we can convert an E-R design into a relational design.

### 1. Representation of Strong Entity Sets with Simple Attributes

Let  $E$  be a strong entity set with only simple descriptive attributes  $a_1, a_2, \dots, a_n$ .

We represent this entity by a schema called  $E$  with  $n$  distinct attributes. Each tuple in a relation on this schema corresponds to one entity of the entity set  $E$ .

*student (ID, name, tot cred)*

*classroom (building, room number, capacity)*

*department (dept name, building, budget)*

*course (course id, title, credits)*

*instructor (ID, name, salary)*

All the strong entity sets, except *time slot*, have only simple attributes

## 2. Representation of Strong Entity Sets with Complex Attributes

The relational schema derived from the version of entity set *instructor* with complex attributes, without including the multi valued attribute, is thus:

*instructor* (ID, *first name*, *middle name*, *last name*,  
*street number*, *street name*, *apt number*,  
*city*, *state*, *zip code*, *date of birth*)

## 3. Representation of Weak Entity Sets

Let  $A$  be a weak entity set with attributes  $a_1, a_2, \dots, a_m$ . Let  $B$  be the strong entity set on which  $A$  depends. Let the primary key of  $B$  consist of attributes  $b_1, b_2, \dots, b_n$ . We represent the entity set  $A$  by a relation schema called  $A$  with one attribute for each member of the set:

$$\{a_1, a_2, \dots, a_m\} \cup \{b_1, b_2, \dots, b_n\}$$

Consider the weak entity set *section* in the University E-R diagram. This entity set has the attributes: *sec id*, *semester*, and *year*. The primary key of the *course* entity set, on which *section* depends, is *course id*. Thus, we represent *section* by a schema with the following attributes:

*section* (*course id*, *sec id*, *semester*, *year*)

## 4. Representation of Relationship Sets

Let  $R$  be a relationship set, let  $a_1, a_2, \dots, a_m$  be the set of attributes formed by the union of the primary keys of each of the entity sets participating in *Relation R*.

*teaches* (ID, *course id*, *sec id*, *semester*, *year*)

*takes* (ID, *course id*, *sec id*, *semester*, *year*, *grade*)

*advisor* (*s ID*, *i ID*)

*sec\_course* (*course id*, *sec id*, *semester*, *year*)

*sec\_time\_slot* (*course id*, *sec id*, *semester*, *year*, *time slot id*)

*sec\_class* (*course id*, *sec id*, *semester*, *year*, *building*, *room number*)

*inst\_dept* (ID, *dept name*)

*stud\_dept* (ID, *dept name*)

*course\_dept* (*course\_id*, *dept name*)

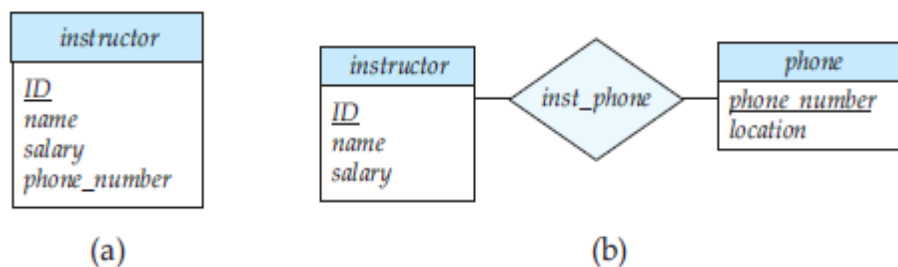
## Entity-Relationship Design Issues

The notions of an entity set and a relationship set are not precise, and it is possible to define a set of entities and the relationships among them in a number of different ways.

### 1. Use of Entity Sets versus Attributes

Consider the entity set *instructor* with the additional attribute *phone number*. It can easily be argued that a phone is an entity in its own right with attributes *phone number* and *location*; the location may be the office or home where the phone is located, with mobile (cell) phones perhaps represented by the value “mobile.” If we take this point of view, we do not add the attribute *phone number* to the *instructor*. Rather, we create:

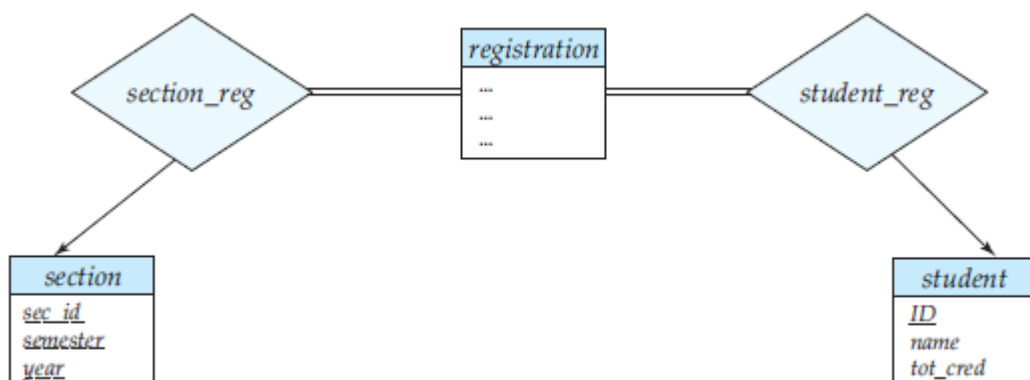
- A *phone* entity set with attributes *phone number* and *location*.
- A relationship set *inst phone*, denoting the association between instructors and the phones that they have.



*Alternatives for adding phone to the instructor entity set.*

### 2. Use of Entity Sets versus Relationship Sets

We have an entity set to represent the course-registration record. Let us call that entity set *registration*. Each *registration* entity is related to exactly one student and to exactly one section, so we have two relationship sets, one to relate course registration records to students and one to relate course-registration records to sections.

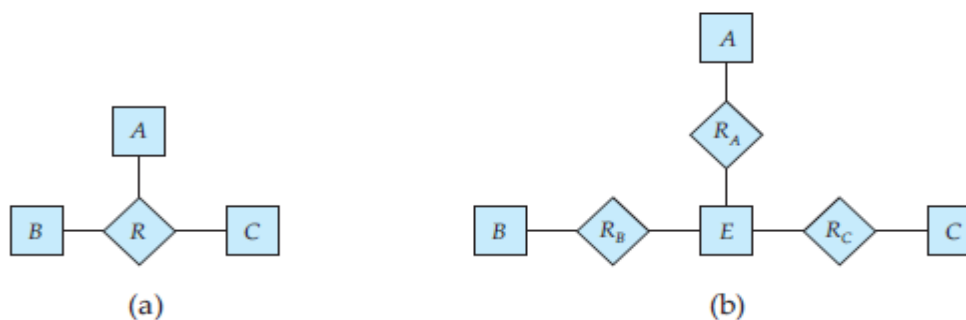


*Replacement of takes by registration and two relationship sets*

### 3. Binary versus $n$ -ary Relationship Sets

Relationships in databases are often binary. Some relationships that appear to be non binary could actually be better represented by several binary relationships.

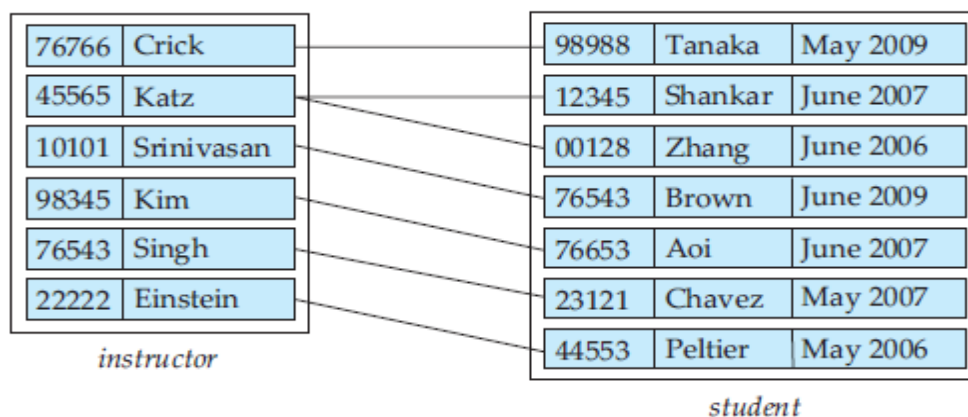
For instance, one could create a ternary relationship *parent*, relating a child to his/her mother and father. However, such a relationship could also be represented by two binary relationships, *mother* and *father*, relating a child to his/her mother and father separately. Using the two relationships *mother* and *father* provides us a record of a child's mother, even if we are not aware of the father's identity; a null value would be required if the ternary relationship *parent* is used.



*Ternary relationship versus three binary relationships.*

### 4. Placement of Relationship Attributes

The cardinality ratio of a relationship can affect the placement of relationship attributes. Thus, attributes of one-to-one or one-to-many relationship sets can be associated with one of the participating entity sets, rather than with the relationship set. For instance, let us specify that *advisor* is a one-to-many relationship set such that one instructor may advise several students, but each student can be advised by only a single instructor. In this case, the attribute *date*, which specifies when the instructor became the advisor of a student, could be associated with the *student* entity set,



*date as an attribute of the student entity set.*

## Extended E-R Feature

As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modelling. Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better.

Hence, as part of the **Extended ER Model**, along with other improvements, three new concepts were added to the existing ER Model:

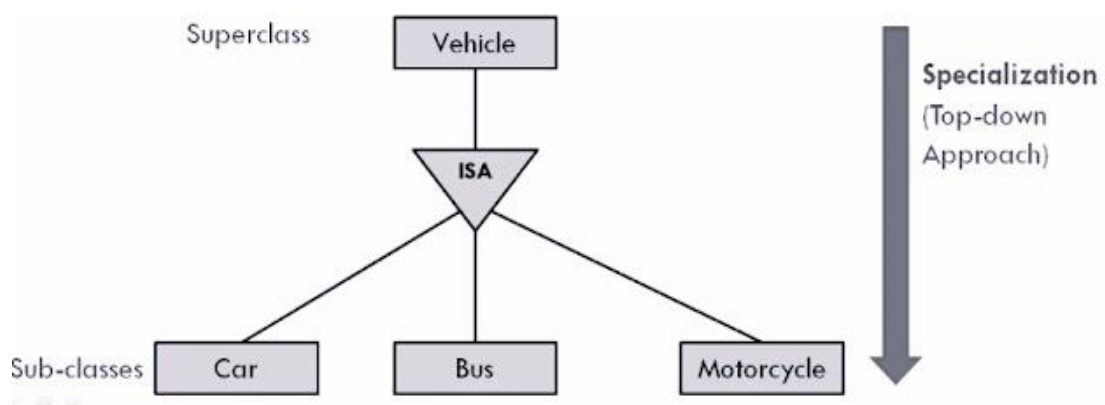
1. **Specialization**
2. **Generalization**
3. **Aggregation**

### 1. Specialization

- In Specialization, an entity is broken down into sub-entities based on their characteristics.
- Specialization is a "**Top-down approach**" where higher level entity is specialized into two or more lower level entities.
- Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics
- Specialization can be repeatedly applied to refine the design of schema.
- Normally, the super class is defined first, the subclass and its related attributes are defined next, and relationship set are then added.
- Depicted by triangle component labeled **ISA**
- Specialization is opposite of Generalization.

#### Example:

*Vehicle* entity can be a Car, Bus or Motorcycle.

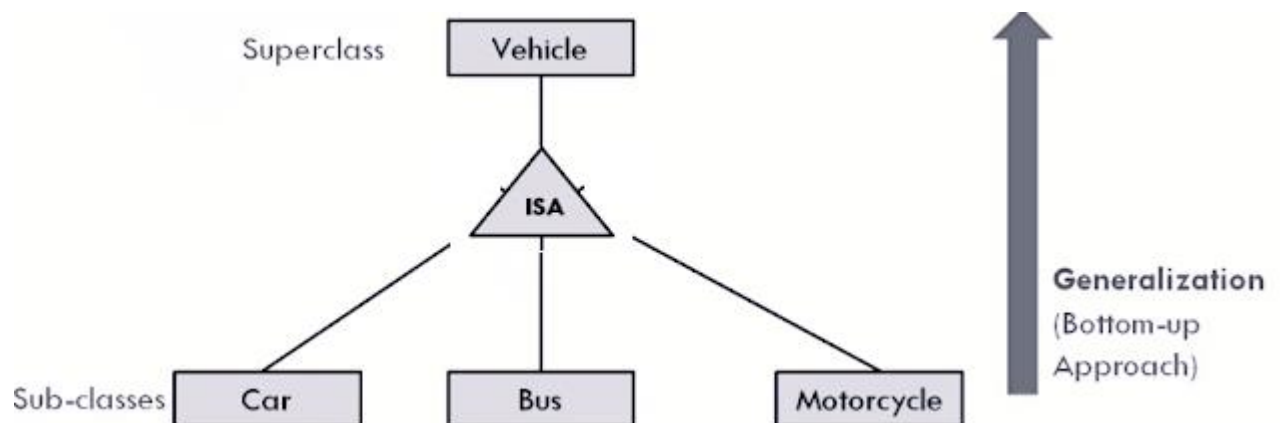


## 2. Generalization

- Generalization is the process of extracting common properties from a set of entities and create a generalized entity from it.
- Generalization is a "**bottle-up approach**" in which two or more entities can be combined to form a higher level entity if they have some attributes in common.
- Subclasses are combined to make a superclass.
- Generalization is used to emphasize the similarities among lower-level entity set and to hide differences in the schema.

### Example:

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



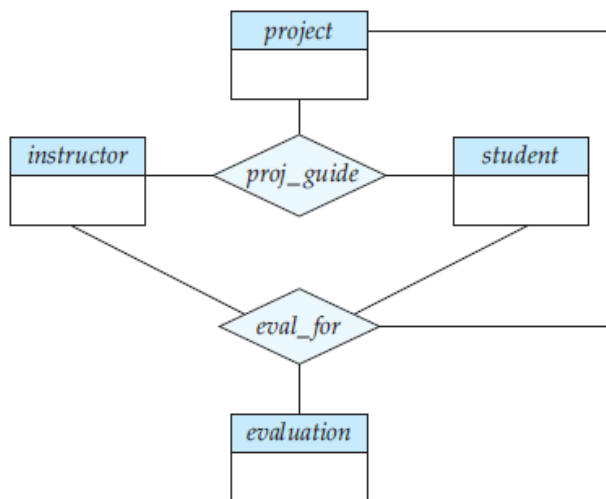
## 3. Aggregation

- Aggregation is used when we need to express a relationship among relationships.
- Aggregation is an abstraction through which relationships are treated as higher level entities.
- Aggregation is a process when a relationship between two entities is considered as a single entity and again this single entity has a relationship with another entity.

**Note:** Basic E-R model can't represent relationships involving other relationships.

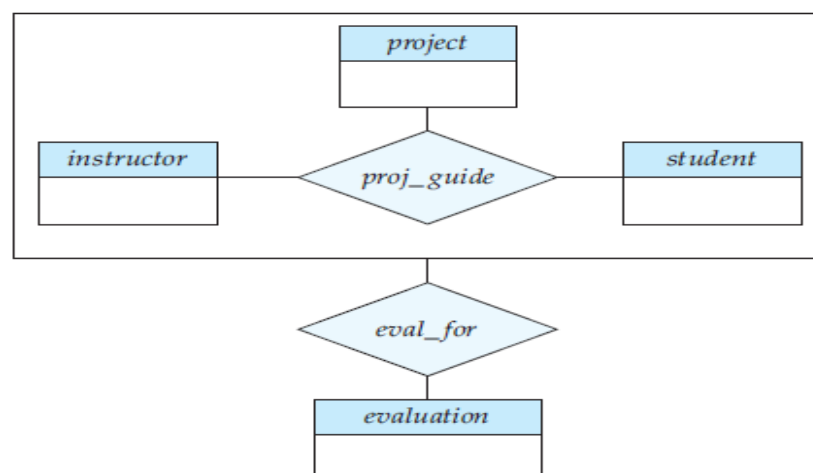
**Example:**

Now suppose that each instructor guiding a student on a project is required to file a monthly evaluation report. We model the evaluation report as an entity *evaluation*, with a primary key *evaluation id*. One alternative for recording the (*student*, *project*, *instructor*) combination to which an *evaluation* corresponds is to create a quaternary (4-way) relationship set *eval\_for* between *instructor*, *student*, *project*, and *evaluation*.



*E-R diagram with redundant relationships*

The relationship set *proj guide* (relating the entity sets *instructor*, *student*, and *project*) as a higher-level entity set called *proj guide*. Such an entity set is treated in the same manner as is any other entity set. We can then create a binary relationship *eval\_for* between *proj\_guide* and *evaluation* to represent which (*student*, *project*, *instructor*) combination an *evaluation* is for.



*E-R diagram with aggregation*



## Structure of Relational Databases

A relational database consists of a collection of **tables**, each of which is assigned a unique name. For example, consider the *instructor* table, which stores information about instructors. The table has four column headers: *ID*, *name*, *dept\_name*, and *salary*. Each row of this table records information about an instructor,

Consisting of the instructor's *ID*, *name*, *dept name*, and *salary*.

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

The *instructor* relation

Thus, in the relational model the term **relation** is used to refer to a table, while the term **tuple** is used to refer to a row. Similarly, the term **attribute** refers to a column of a table.

Examining the above table, we can see that the relation *instructor* has four attributes:

*ID*, *name*, *dept name*, and *salary*.

We use the term **relation instance** to refer to a specific instance of a relation, i.e., containing a specific set of rows. The instance of *instructor table* has 12 tuples, corresponding to 12 instructors.

## Database Schema

When we talk about a database, we must differentiate between the **database schema**, which is the logical design of the database, and the **database instance**, which is a snapshot of the data in the database at a given instant in time.

The concept of a relation corresponds to the programming-language notion of a variable, while the concept of a **relation schema** corresponds to the programming-language notion of type definition.

**Schema:** Schema refers to the basic structure of how one needs to store data in any database. There are basically three types of Schema: Physical Schema, Logical Schema and View Schema.

**Physical Schema** – This schema describes the Database designed at a physical level.

**Logical Schema** – This schema describes the Database designed at a logical level.

**View schema-** In view schema, the database is designed at the view level. This schema describes the user interaction with the database system.

**Instance:** It refers to a collection of all the information and data stored at any given moment.

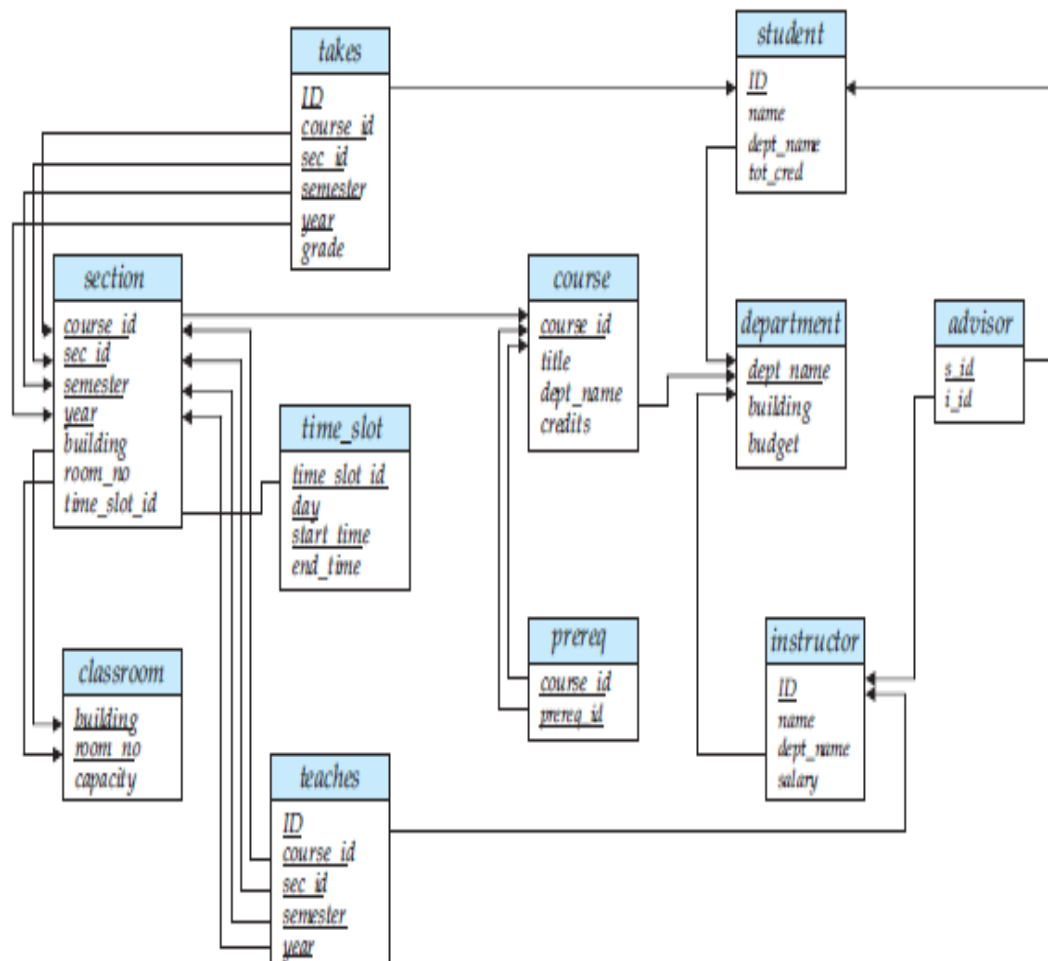
#### Difference between Schema and Instance

Schema	Instance
It is the overall description of the database.	It is the collection of information stored in a database at a particular moment.
The schema is same for the whole database.	Data in instances can be changed using addition, deletion, and updation.
Does not change Frequently.	Changes Frequently.
Defines the basic structure of the database i.e. how the data will be stored in the database.	It is the set of Information stored at a particular time.
Affects the entire database structure.	Affects only the current state of data.
Requires significant effort and planning to change.	Easily altered by performing <u>CRUD</u> (Create, Read, Update, Delete) operations.

Schema	Instance
Table structures, relationships, constraints.	Data entries, records in tables.

## Schema Diagrams

A database schema, along with primary key and foreign key dependencies, can be depicted by **schema diagrams**. The schema diagram for our university organization. Each relation appears as a box, with the relation name at the top in blue, and the attributes listed inside the box. Primary key attributes are shown underlined. Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation.



Schema diagram for the university database.