



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Introduction

Dr. Nagasundari S

Department of Computer Science and Engineering

Unit 1: Introduction to Database Management

Database System Applications, Purpose, View of data, Database Languages, Database design, Introduction to databases, Database application architecture, Users and Administrators, E-R Model, reducing ER to a relational schema. Structure of relational databases, Database schema, and its constraints, Keys

14 Hours

Unit 2: Relational Model and Database Design

Relational operations (Algebra), Unary Operations - Unity, Binary, Aggregate Functions, Grouping, SQL ,overview, Data definition, Structure of SQL queries, Additional Basic Operations, Set Operations, Null Values, Aggregate Functions, Nested Subqueries, Database Modification, Join expressions, Views, Triggers, Functions, and Procedures, Introduction to strategies of Query processing and Query optimization.

14 Hours

Unit 3: Advanced Design Concepts and Implementation

Functional Dependencies, Inference Rules, Closure, Equivalence, Minimal Cover Normal Forms Based on Primary Keys (1NF, 2NF, and 3NF), General Definitions of Second and Third Normal Forms Boyce-Codd Normal Form, Properties of Relational Decompositions, Overview of Higher Normal Forms.

14 Hours

Unit 4: Advanced Databases

Database transactions, Concurrency control, Locking, Recovery, Database Security, Introduction to NoSQL databases, Document database (MongoDB), Key-Value database (DynamoDB), Graph databases (Neo4j), Wide-column store database (HBase)

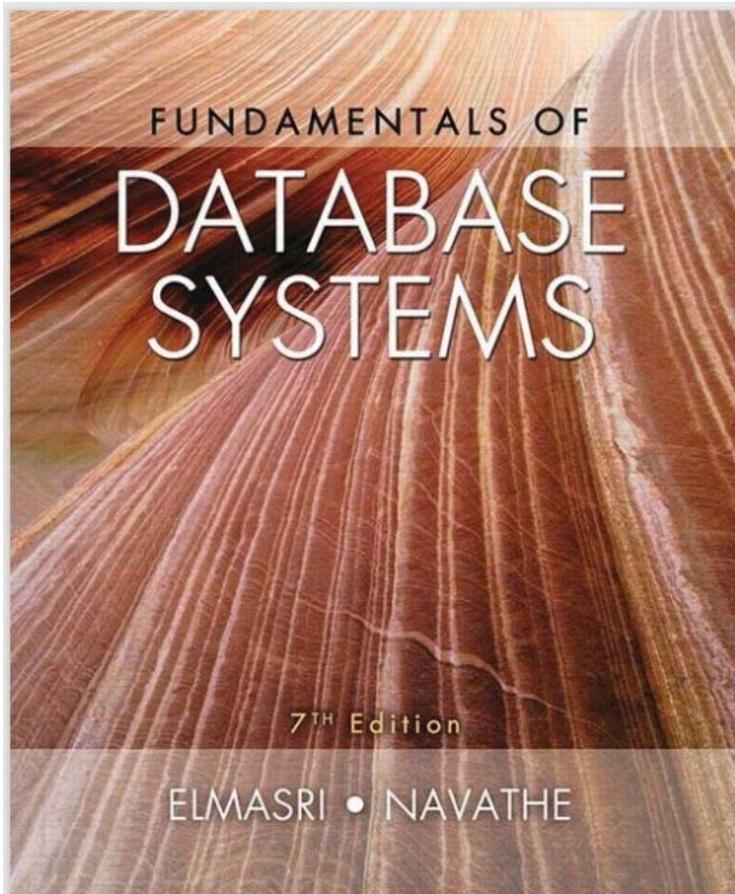
14 Hours

Lab/Hands-on sessions

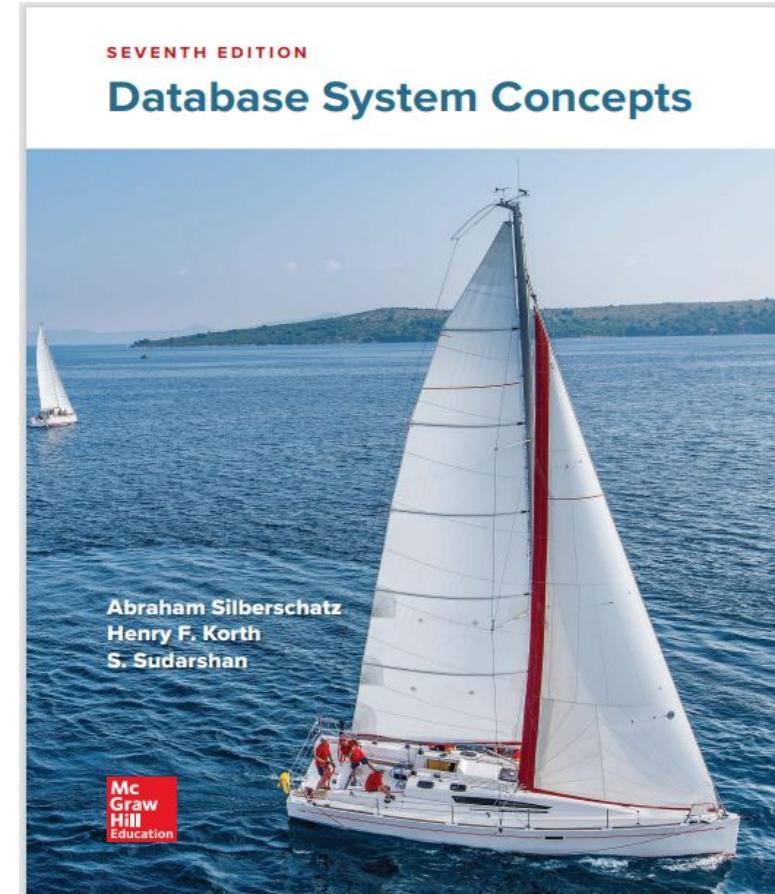
- Draw an ER diagram for a given problem statement
- Conversion of an ER diagram into Relational schema
- DDL – create, constraints, alter, rename, drop, truncate table, Views. DML – Insert, Update, Delete, Transactions - commit, rollback, savepoint
- SQL - Set operators: union, intersect, minus.
- SQL – Aggregate functions.
- SQL – Joins: inner, outer; Sub queries: correlated and uncorrelated. SQL – Creating Functions and Procedures
- SQL – Creating Triggers and Cursors
- XML- Database access. NoSQL database queries
- High-level programming language accessing a database using an API.

Tools/ Languages: MySQL Workbench, Python, ERwin, Any other tool for ER modeling

Textbook 1 (TB1)



Textbook 2 (TB2)



DBMS- 5 credits course

DBMS Theory		
Component	Conducted Marks	Scaled Marks
ISA-01	40	20
ISA-02	40	20
ESA	100	50
Jackfruit - Experiential Learning - Hackathon (Team work)	20	10
DBMS Lab		
Orange – Team Assignments (Continuous Evaluation)	20	10
Banana – Use Case based Exercises (Individual)	$9 * 10 = 90$	10
Total Marks	120	

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.



Dr. Nagasundari S

Department of Computer Science and Engineering

- Have you ever wondered on how a large e-commerce website like Amazon is able to retrieve the products that you would want to purchase?
- Nowadays social media (eg: Instagram) one of the most active platforms where people post their day to day activities etc. how and where is all these information stored? By typing a person's name we can get to see their entire profile, posts that person has posted how is this possible?
- In a bank, multiple transactions take place everyday. Even if 2 transactions are happening simultaneously from account, still the money in the accounts is properly deducted and credited how does this happen?



- Suppose you visit your doctor for a general checkup. Now by typing in your name or your ID, all your medical history details would be visible to the doctor. How does this happen?
- When you search for a specific topic on Google, how does the search engine quickly retrieve relevant web pages from millions of websites and present them to you?

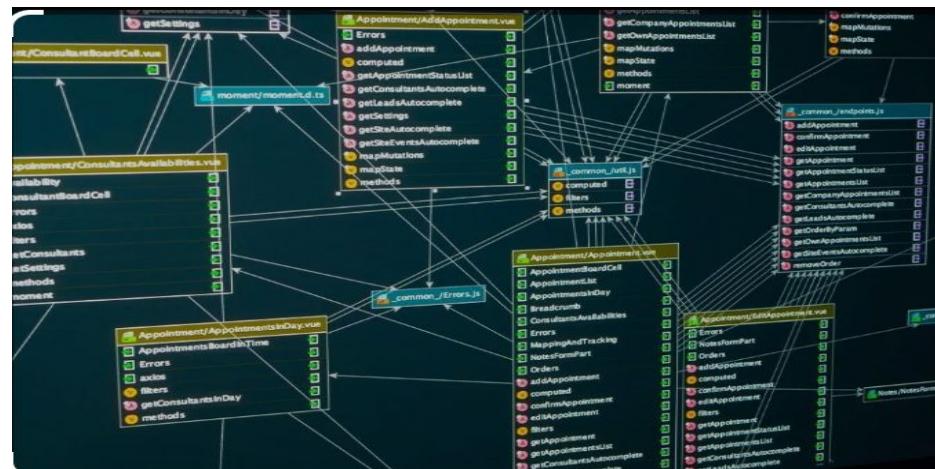


All the above questions have a single answer and that is **DATABASE**

Before going ahead and understanding what database exactly and how is designed and used for all the other application, let us try answering a few more questions:

- What do you mean by **Data**?
- How is it different from **Information**?

In current world, there is a lot of data that is available, how are all these stored or organised so that we can get the required information?



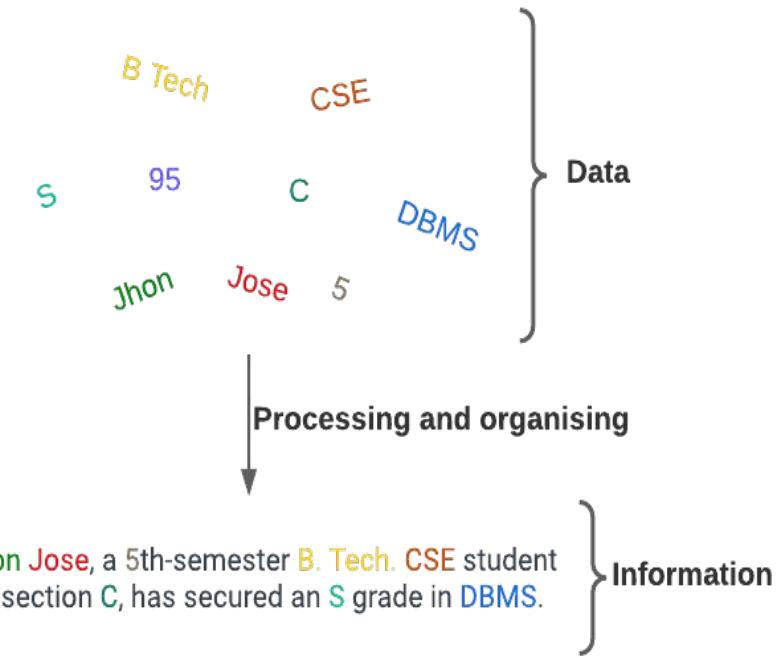
What Do you mean by Data & Information?

Data

- Data represents the raw material, like individual puzzle pieces, that serves as the foundation for knowledge.
- Data is a raw and unorganized fact that is required to be processed to make it meaningful.

Information

- When data is processed, organized, structured, or presented in a given context to make it useful is called information.
- Data is a collection of facts, while information puts those facts into context
- It is organized and is utilized by humans in some significant way to make decisions and draw some conclusions



Data doesn't depend on information, Information depends on data

What is a Database?

Database

- A database is a **collection of related data** representing some aspect of the real world, also called the **mini-world** or the **universe of discourse (UoD)**.
- It is a **logically coherent collection**, meaning it is not just a random assortment of data but is organized with inherent meaning and structure.
- Databases are designed, built, and populated for a **specific purpose**, catering to the needs of applications or systems that interact with them.
- They can **vary in size and complexity**, ranging from small databases used by individual applications to large-scale enterprise databases handling vast amounts of data.

Could you think of what do these individual terms mean w.r.t student information Database?

What is a Database

Student Information Database for a university

Collection of Related Data: The database contains various pieces of information related to students, courses, and grades. It includes tables for students, courses, and enrollments, all connected through relationships.

Miniworld (Universe of Discourse): The miniworld in this context is the university's academic environment. The database captures essential aspects of the real-world entities like students and courses within the university.

Logically Coherent Collection: The database is not a random assortment of data but organized logically. For instance, student information is stored in the "Students" table with fields like student ID, name, and contact information, creating a coherent representation of students' details.

Designed for a Specific Purpose: The database is designed to store and manage student information, course details, and enrollments efficiently. It serves the purpose of providing a centralized system for university administrators to access and update student-related data.

Any Size and Complexity: The database's size and complexity can vary based on the university's size and the number of students and courses it manages. Larger universities might have more extensive databases with additional tables for faculty, academic departments, and research projects.

Database Management System

- A modern database system is a complex software system whose task is to manage a large, complex collection of data.
- DBMS contains information about a particular enterprise
 - Collection of interrelated data
 - Set of programs to access the data
 - An environment that is both convenient and efficient to use
- Database systems are used to manage collections of data that are:
 - Highly valuable
 - Relatively large
 - Accessed by multiple users and applications, often at the same time.

Let us break these statements down further and understand them in the context of the student information database

Student Information Database

Information about a Particular Enterprise:

- This Database is tailored for the university's academic needs, storing vital data on students, courses, faculty, and administration.

Convenient and Efficient Environment:

- Modern DBMS offers a user-friendly interface and query languages for easy data interaction.
- It ensures efficient data operations and optimization for enhanced system performance.

Collection of Interrelated Data:

- Interrelated data in "Students," "Courses," and "Enrollments" tables enables coherent retrieval and presentation, reflecting real-world interactions between students and courses.

Set of Programs to Access the Data:

- Administrators access data via web-based app for student info, while faculty use custom apps for course management and grading.

Managing a Large, Complex Collection of Data:

- The database efficiently manages diverse data elements for students, courses, enrollments, and grades.
- It scales to accommodate university growth and maintains performance and reliability even with an expanding data volume.

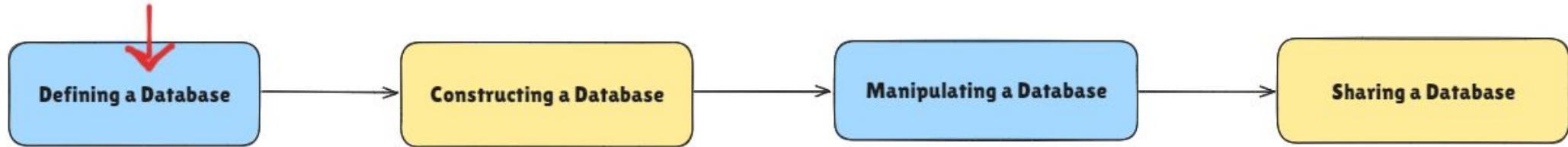
Accessed by Multiple Users and Applications:

- supports concurrent access for administrators, faculty, and students with specific access rights.
- The DBMS effectively manages concurrent access, preventing conflicts and ensuring data integrity.

Highly Valuable and Relatively Large Data:

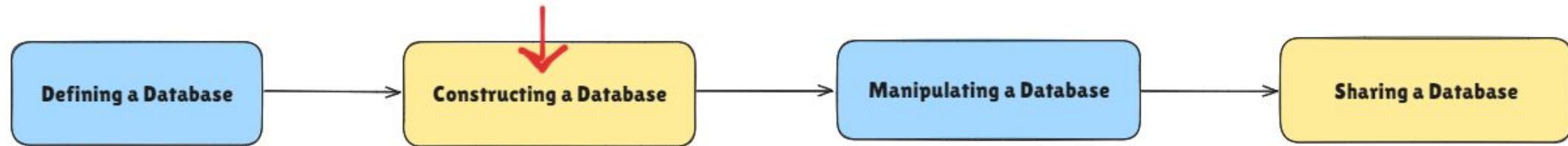
- Database holds invaluable data for academic administration, supporting student progress tracking, course planning, and faculty management.
- Due to the substantial student and course count, it qualifies as a relatively large data collection.

Database Management System



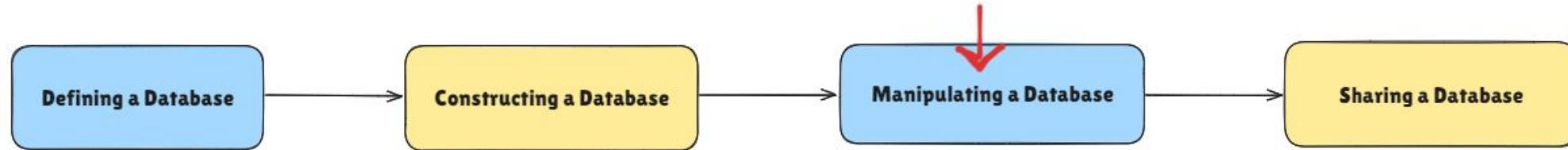
- A general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications
 - **Defining a database** involves specifying the data types, structures, and constraints of the data to be stored in the database. By defining the structure and data types, the DBMS ensures the consistency and integrity of the stored data.
 - The "Students" table may include fields such as "Student ID" (numeric), "Name" (text), "Email" (text), "Date of Birth" (date), and "Major" (text).
 - The "Courses" table may have fields like "Course ID" (numeric), "Course Name" (text), "Credits" (numeric), and "Instructor" (text).

Database Management System



- A general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications
 - **Constructing the database** is the process of storing the data on some storage medium that is controlled by the DBMS. It organizes the data in a way that allows for efficient retrieval and manipulation. The DBMS ensures data is securely stored and can be accessed by authorized users or applications.
 - The DBMS may store the Student Information Database on a server with appropriate data storage capacity and performance capabilities.

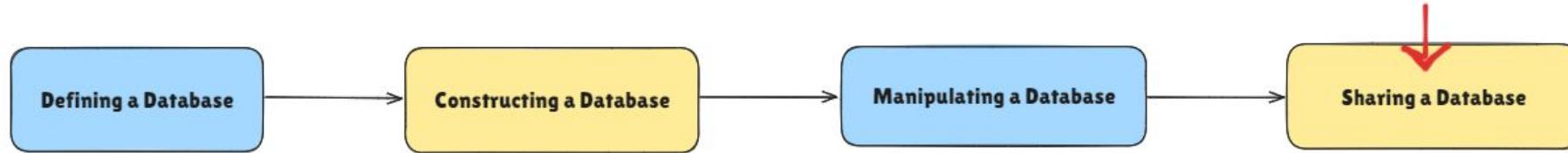
Database Management System



- A general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications
 - **Manipulating a database** includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the mini world, and generating reports from the data. It ensures that any changes made to the database follow predefined constraints and maintain data consistency.
 - A university administrator may use SQL queries to retrieve specific information, such as "Retrieve all students enrolled in Computer Science courses." The DBMS processes the query, retrieves the relevant data from the database, and presents it to the user.
 - Allows authorized users to update the database, such as adding new students or updating grades.

Database Management System

Database Management System



- A general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications
 - **Sharing a database** allows multiple users and programs to access the database simultaneously, provided they have the necessary permissions
 - Faculty members, administrative staff, and students can access relevant information concurrently, as long as they have the necessary permissions.
 - For example, faculty members can access student details for their courses, while administrators can update and maintain the overall database.

Questions

Some questions to revise

- 1) How do you define data, and how does it differ from information in the context of database management systems?
- 2) List out the data types you would take to define a database for an online bookstore.
- 3) Explain the role of a DBMS in managing highly valuable and relatively large collections of data. Why is this important for organizations?
- 4) What are some of the common operations performed on a database? How do these operations interact with the data stored in the database?

How are databases used?

Some online resources -

- <https://dev.mysql.com/doc/>
- <https://www.oracle.com/in/database/what-is-database/>
- https://docs.oracle.com/cd/E17952_01/index.html
- <https://downloads.mysql.com/docs/mysql-tutorial-excerpt-8.0-en.a4.pdf>



PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Introduction

Dr. Nagasundari S

Department of Computer Science and Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.



Dr. Nagasundari S

Department of Computer Science and Engineering

Why study databases

Databases help us to:

- Store large amounts of data (file structure, disk management)
- Understand the data (data models)
- Keeps data secure (security, recovery)
- Find required data and use/manipulate it (query languages, concurrency control, and data analysis tools)
- Get accurate information (as databases have built-in constraints and checks help in this)
- Maintain Data integrity (ensures data is accurate and consistent)

1) Enterprises

- **Customer Relationship Management (CRM):** Databases store customer interactions, preferences, and sales data.
- **Supply Chain Management:** Databases track inventory levels, supplier information, and logistics.
- **Human Resources:** Manage employee data, payroll, and benefits using databases.
- **Financial Systems:** Handle accounting, budgeting, and financial reporting through database solutions.

- **Example:** **Salesforce CRM** is widely used by enterprises to manage customer relationships and sales.



2) Manufacturing

- **Production Tracking:** Databases monitor production schedules, inventory levels, and quality control.
- **Supply Chain Management:** Manage supplier information, procurement, and logistics using databases.
- **Product Lifecycle Management:** Track product design, development, and updates in a database.
- **Example:** **Toyota** uses databases to manage their Just-In-Time (JIT) manufacturing processes, ensuring efficient production and inventory management.



3) Banking and Finance

- **Account Management:** Databases store customer accounts, transactions, and balance information.
- **Loan Processing:** Manage loan applications, approvals, and repayments through a database system.
- **Fraud Detection:** Monitor transactions for unusual activity and potential fraud using database systems.
- **Example:** **JPMorgan Chase** uses advanced database systems to manage customer accounts, process transactions, and detect fraud.



4) University

- **Student Records:** Databases store student information such as personal details, course registrations, grades, and attendance.
- **Course Management:** Databases manage course offerings, schedules, and faculty assignments.
- **Library Systems:** Track book inventory, borrowing history, and reservations using databases.
- **Alumni Relations:** Maintain contact information and other records in a centralized database.
- **Example:** **University of California, Berkeley** uses a comprehensive database system to manage student records, academic schedules, and library resources.

5) Airlines

- **Reservation Systems:** Databases manage flight bookings, seat allocations, and passenger information.
- **Flight Operations:** Track flight schedules, crew assignments, and aircraft maintenance through databases.
- **Customer Service:** Store frequent flyer data, customer preferences, and service history in a database.
- **Example:** Amadeus is a global reservation database system used by many airlines for booking and flight management.



6) Telecommunication

- **Customer Data:** Databases store customer information, billing details, and service preferences.
- **Network Management:** Monitor network usage, service quality, and fault management using databases.
- **Service Provisioning:** Manage the activation and maintenance of customer services through a database system.
- **Example:** AT&T utilizes databases to manage customer data, monitor network performance, and provide reliable telecommunication services.



7) Web-based Services

- **User Data Management:** Databases store user profiles, preferences, and activity logs.
- **Content Management:** Manage website content, including articles, images, and videos, using a database system.
- **Analytics:** Track user interactions, traffic patterns, and conversion rates through databases.
- **Example:** **Netflix** uses a sophisticated database system to manage user preferences, content recommendations, and streaming data.



8) Document Databases

- **Data Storage:** Document databases store semi-structured data in formats like JSON, BSON, or XML, allowing for flexible schema design and easy data retrieval.
- **Content Management:** Manage collections of documents such as articles, patents, research papers, and other unstructured content.
- **Scalability:** Document databases are designed to handle large volumes of data and can scale horizontally across multiple servers.
- **Use Cases:** Often used for content management systems, catalogues, and data lakes where data doesn't fit neatly into relational models.
- **Example:** **MongoDB** is a popular document database used by companies like **The New York Times** to manage and deliver vast amounts of news articles and media content.



9) Navigation Systems

- **Location Data:** Databases store geographic information including locations of places of interest, routes, and geographic features.
- **Real-Time Updates:** Manage real-time traffic data, route optimization, and point-of-interest information.
- **User Interaction:** Provide features like route planning, navigation guidance, and local searches.
- **Integration:** Often integrated with other systems such as weather services, local businesses, and emergency services to enhance user experience.
- **Example:** **Google Maps** uses a complex database system to manage and update location data, provide navigation services, and offer real-time traffic updates to millions of users worldwide.



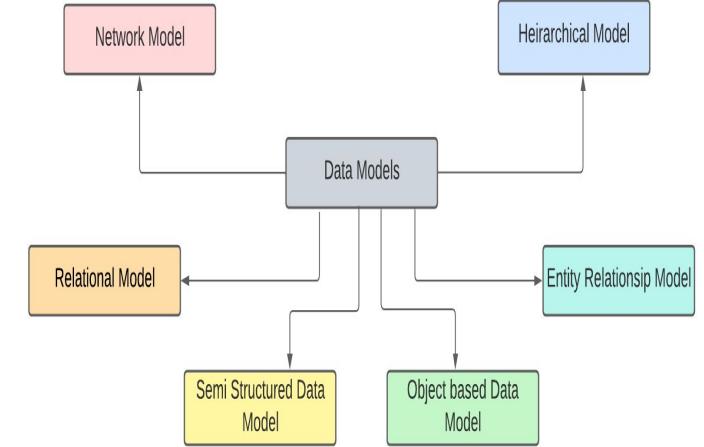
Google Maps

How are databases used?

There are two modes in which databases are used today

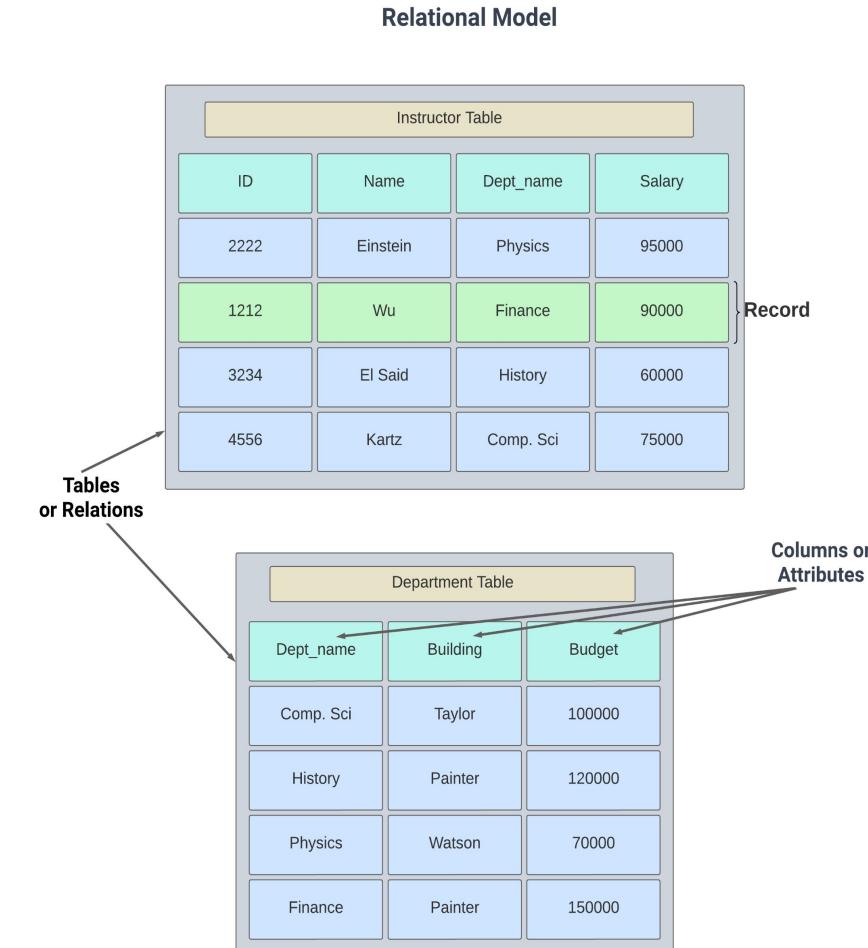
- **Online Transaction Processing (OLTP):**
 - Used by a large number of users for small data retrieval and updates
 - common in most database applications like banking, universities, and airlines.
- **Data Analytics/Online Analytical Processing(OLAP):**
 - Involves processing data to draw conclusions and create predictive models for business decisions.
 - Examples include loan approval, targeted advertisements, and manufacturing decisions.
 - Data mining combines AI and statistical techniques for efficient analysis of large databases.

- 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.
 - Data models
 - A collection of conceptual tools for describing data, data relationships, data semantics and constraints.
 - Data abstraction
 - Hide the complexity of data structures to represent data in the database from users through several levels of data abstraction.
- The structure of a database is defined by its data model, which includes tools to describe data, relationships, semantics, and consistency constraints.



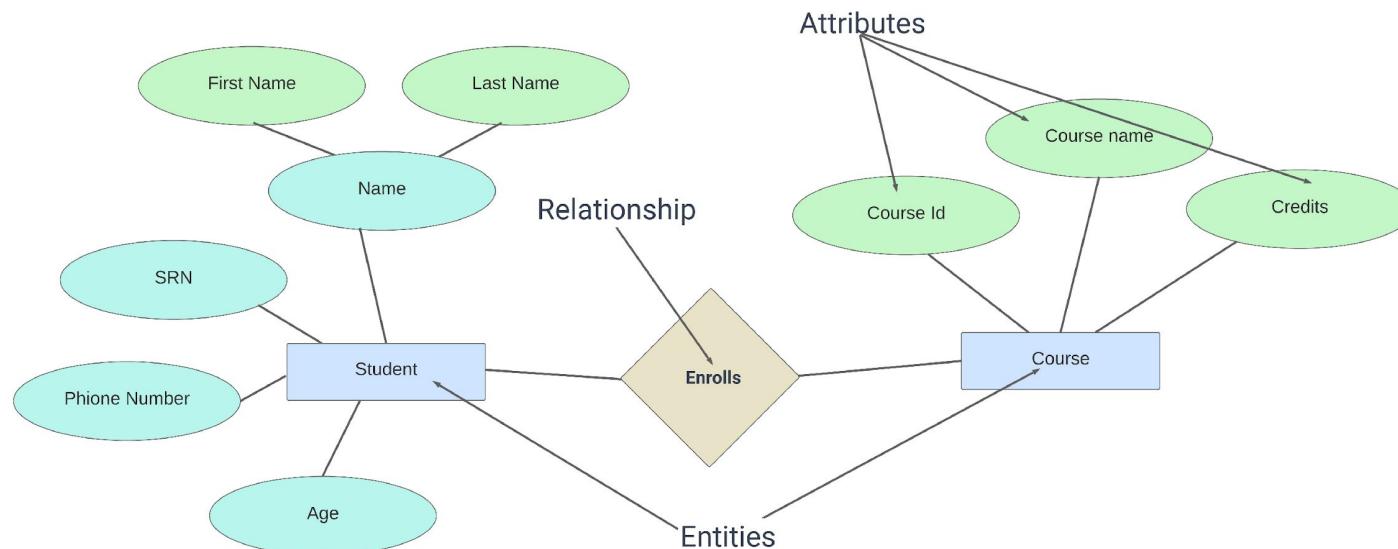
RELATIONAL MODEL

- Uses a collection of tables to represent both data and the relationships among those data.
- Each table has multiple columns, and each column has a unique name.
- Tables are also known as relations.
- The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types.
- Each **table contains records of a particular type**. Each record type defines a fixed number of fields or attributes.
- Each **row of the table represents one piece of information**
- The columns of the table correspond to the attributes of the record type.
- The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.



Entity-Relationship Model

- The entity-relationship (E-R) data model uses a collection of basic objects, called entities, and relationships among these objects.
- An entity is a “thing” or “object” in the real world that is distinguishable from other objects.
- The entity-relationship model is widely used in database design.



Semi Structured Data Model

- The semi-structured data model permits the specification of data where individual **data items of the same type may have different sets of attributes**.
- This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes.
- JSON and Extensible Markup Language (XML) are widely used semi-structured data representations
- A database model where there is **no separation between the data and the schema**, and the amount of structure used depends on the purpose.
- The advantages of this model are the following: It can represent the information of some data sources that cannot be constrained by schema.

```
<purchase_order>
  <identifier> P-101 </identifier>
  <purchaser>
    <name> Cray Z. Coyote </name>
    <address> Route 66, Mesa Flats, Arizona 86047, USA </address>
  </purchaser>
  <supplier>
    <name> Acme Supplies </name>
    <address> 1 Broadway, New York, NY, USA </address>
  </supplier>
  <itemlist>
    <item>
      <identifier> RS1 </identifier>
      <description> Atom powered rocket sled </description>
      <quantity> 2 </quantity>
      <price> 199.95 </price>
    </item>
    <item>
      <identifier> SG2 </identifier>
      <description> Superb glue </description>
      <quantity> 1 </quantity>
      <unit-of-measure> liter </unit-of-measure>
      <price> 29.95 </price>
    </item>
  </itemlist>
  <total_cost> 429.85 </total_cost>
  <payment_terms> Cash-on-delivery </payment_terms>
  <shipping_mode> 1-second-delivery </shipping_mode>
</purchase_order>
```

Example of XML
data

Object-Based Data Model.

- Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology.
- This led initially to the development of a distinct object-oriented data model, but today the concept of objects is well integrated into relational databases.
- Standards exist to store objects in relational tables. Database systems allow procedures to be stored in the database system and executed by the database system.
- This can be seen as extending the relational model with notions of encapsulation, methods, and object identity.

Questions

Some questions to revise

- 1) List and describe three real-world applications of databases in different industries.
How do these applications benefit from using databases?
- 2) What are the two modes in which databases are used today?
- 3) Discuss the various data models. How does the relational model differ from the semi-structured and object-based models?
- 4) Explain the significance of document databases in storing and managing semi-structured data. How does MongoDB exemplify the use of document databases in real-world applications?



PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Purpose of Database System

Dr. Nagasundari S

Department of Computer Science and Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S

Department of Computer Science and Engineering



File-based System

- One way to keep information on a computer is to store it in permanent files.
- A company system has a number of application programs; each of them is designed to manipulate data files.
- These application programs have been written at the request of the users in the organization.
- New applications are added to the system as the need arises.

Let us take an example of a University Organization system. We need to store information about the following entities:

- Instructors, Students, Departments, Courses offered.
- One of the ways to keep this information is to store it in **operating file systems**
- To allow the users to manipulate the information, the system would have a number of application programs that manipulate the files including:
 - Addition of new students, instructors, and courses
 - Register students for different courses
 - Assign grades to students, compute GPA, and generate transcripts.

Scenario: The university decides to add a new major.

What specific steps must be taken to integrate a new major into the university's existing system?

1. Create a New Department:

- Set up a new department for the major.
- Create new permanent files for department information.

2. Develop New Application Programs:

- Write programs to manage rules for the new major.
- Ensure programs handle course registration, grading, and transcripts for the new major.

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.

What is the need for a Database Management System when we can do the university organization using a simple file-processing system?

Consider the following situations:

- Double Major Students :

What if there is a student who has enrolled for a double major (Physics and Computer Science) how many times do we store all his details?

- Updating information :

What happens if a student changes their address or phone number? Is it guaranteed that the update reflects in all relevant files across departments?

- Efficient Data Retrieval:

The university transport department wants to find out all students who live within a particular postal code area. Can he quickly retrieve the data or does he have to build another application for it?

- Data Format Consistency:

What if files that store these data are written in different formats is it easy to access the data in this form?

- Adding New Constraints:

Suppose a new constraint is to be added, can it be done efficiently without having to make extensive changes to existing application programs?

There are a lot of drawbacks to the file-processing system :

- Data Redundancy and Inconsistency
- Difficulty in accessing the data
- Data isolation
- Integrity problems
- Atomicity problems
- Concurrent access anomalies
- Security problems

1. Data Redundancy & Inconsistency

Consider the question of the student enrolled for a double major. What would happen in this situation in case of file system?

Details of the students would be duplicated for each department - **Data Redundancy**

This would lead to the following problems:

- Takes up more storage space
- May also lead to **Data Inconsistency** in case of changes

Solution to this problem:

- Store information in centralized location thus Minimizes redundancy
- Changes can be made at a single spot, thus ensuring consistency
- This can be achieved through a database system

2. Difficulty in accessing the data

Consider that the university wants **to find all the students living in a specific postal code area**. How will we execute this in traditional file system?

- Either we need to get a list of all students and manually sift through it
- Or create another special program for this task

Both the above-mentioned are time-consuming and inconvenient

Thus we can see that Conventional file-processing systems are not designed for quick and efficient data retrieval, especially when new types of queries arise.

The solution to this problem:

- Central information store for fast retrieval
- Use a powerful query language for answering new queries
- Efficient ways to search and filter data for speeding up data retrieval
- This can be achieved through a database system

3. Data Isolation

Consider a concurrency situation, where identifying when and how changes made by one operation become visible to other concurrent users.

This is a challenge because New applications has to retrieve the appropriate data, which might be stored in various files.

A file-based system must manage, or prevent, concurrency by the application programs.

When an application opens a file, it will lock the file. This means that no one else has access to the file at the same time.

The solution to this problem:

- Ability to allow multiple transactions to access same data without interfering with each other
- Consistent view of the data

4. Integrity problems

Scenario: Each department has an **account for research funds, and the account balance must always be above 0.**

How can we implement this?

Each application program is created for maintaining departmental accounts but adding new constraints becomes challenging as it requires editing all the existing application programs.

The solution to this problem:

- Specify integrity constraints at the database level to specify rules directly in the schema
- System should enforce constraints automatically
- To add a new constraint, updating the schema alone will be sufficient, and the system will handle the rest automatically.
- This would be easily solved by using DBMS which provides all these facilities

5. Atomicity problems:

In a computer system failure can happen unexpectedly

Consider a banking system, a sum of 5000 rupees is getting transferred from account A to B.

What happens internally?

- An amount of 5000 Rs. gets deducted from account A
- The amount (5000 Rs.) gets credited to account B

Now **either both the steps should be performed or none should be executed.**

In a file system, The program might successfully deduct 5000 from account A but fail to credit it to account B.

This results in an **inconsistent** database state, with money missing from one account and not added to the other.

The solution to this problem:

- Atomicity should be ensured
- Automatic rollback in case of failure
- DBMS would ensure this and makes it reliable and consistent even if failure occurs

6. Concurrent Access Anomalies

Consider a bank transaction example. There is a bank account (Account A) with a balance of \$10,000.

- Two bank clerks attempt to deduct money from Account A at almost the same time - one deducts \$500 and the other \$100.
- Since they work concurrently, they both read the initial balance of \$10,000, subtract their respective amounts, and write back the results.
- Depending on which one writes the new balance last, the account could end up with either \$9,500 or \$9,900 instead of the correct value of \$9,400.

The solution to this problem:

- Transaction and Locking to be implemented
- Transactions ensure atomicity
- Locking prevents conflicting changes.
- The DBMS would provide these and maintains data consistency and integrity

7. Security problems

Imagine a university using a file-based system. Over time, they've added various application programs without a central security system.

As a result, payroll personnel might accidentally see and even change academic records, compromising data privacy and confidentiality because everyone has the same permissions.

The solution to this problem:

- Create a provision for administrator to define access controls and permissions for each user or group.
- These permissions should be enforced at the database level, regardless of the application programs used.
- DBMS supports various methods for introducing security like User Authentication, Data Encryption, Auditing and Logging, Centralized security management

- **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 (Metadata).
- **Insulation between programs and data, and data abstraction**
 - The structure of data files is stored in the DBMS catalog separately from the access programs (program-data independence)
- **Support multiple views of the data**
 - DBMSs support different types of users, each of whom may require a different perspective or view of the database. 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.
- **Sharing of data and multi user transaction processing**
 - A multi user DBMS, allows multiple users to access the database at the same time. It ensures that data changes are effected in a controlled manner so that the result of the changes are correct

- Controlling redundancy in data storage.
- Sharing of data among multiple users.
- Restricting unauthorized access to data
- Providing persistent storage for program Objects
 - E.g., Object-oriented DBMSs make program objects persistent
- Providing Storage Structures and search techniques for efficient Query Processing
- Providing backup and recovery
- Providing multiple interfaces to different classes of users.

Advantages of Database Systems

- Representing complex relationships among data.
- Enforcing integrity constraints on the database.
- Drawing inferences and actions from the stored data using deductive and active rules and triggers
- Potential for enforcing standards
- Reduced application development time
- Flexibility to change data structures
- Availability of current information
- Economies of scale

Identify the Problem: Case Study Applications

1. Scenario 1: Banking Transactions:

At a bank, two customers attempt to withdraw money from the same account simultaneously. Due to a lack of concurrency control, the system fails to manage the simultaneous transactions correctly, resulting in the account being overdrawn.

Problem Type: Concurrent Access Anomalies



Scenario 2: Customer Records:

A company stores customer information in multiple spreadsheets, with each department maintaining its own copy. Occasionally, customer addresses and contact details differ between spreadsheets, leading to confusion and errors in communication.

Problem Type: Data Redundancy and Inconsistency

1

Name	Address	Phone
John Doe	123 Elm Street	555-9067
Grace	12 Oak Street	123-8907
Judy	45 York Lane	237-1123

2

Name	Address	Phone
Grace	12 Oak Street	123-8907
John Doe	129 Blueberry Lane	555-9067
Judy	45 York Lane	237-1123

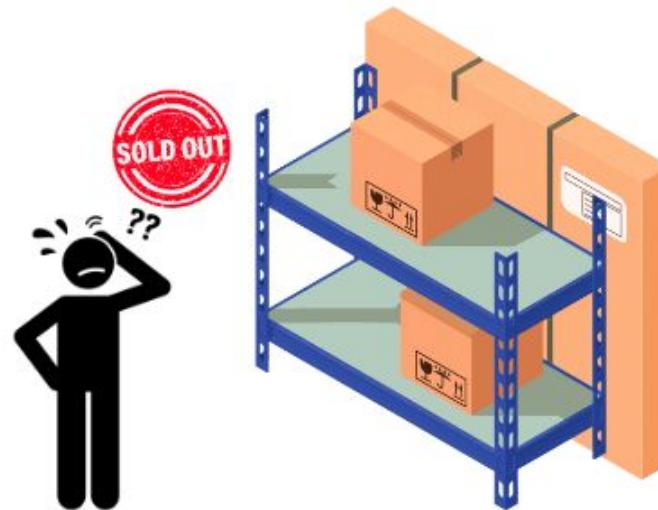
3

Name	Address	Phone
John Doe	123 Elm Street	555-9067
Judy	45 York Lane	237-1123
Grace	12 Oak Street	123-8907

Scenario 3: Inventory and Orders:

A small business uses separate systems to manage inventory and orders. The inventory system is not updated in real-time with sales data, leading to situations where the business oversells products, unaware that the stock has been depleted.

Problem Type: Data Isolation

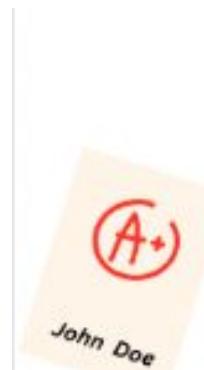


Scenario 4: Student Grades:

A university records student grades manually, and sometimes errors are made in entering the data, such as typing grades incorrectly. There is no system in place to check for or correct these errors automatically.

But, if grade is entered as 'x' which is not a valid grade then constraints can be added to handle.

Problem Type: Integrity Problems



Name	Grade
John Doe	X
Judy	A
Grace	C



PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Data Abstraction & Instances, Schema

Dr. Nagasundari S

Department of Computer Science and Engineering

Database Management Systems

Unit 1: Introduction to Database Management

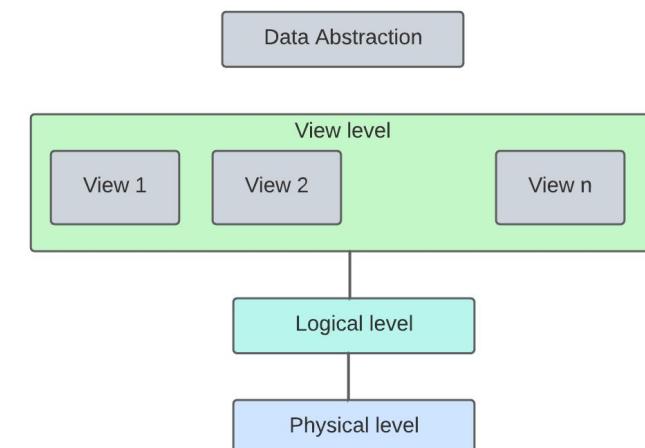
Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering



Data Abstraction

- DBMS plays an important role in many areas. One of the main requirements is that it must retrieve the data efficiently.
- This need for efficiency has led the database developers to use complex data structures to represent data in a database
- Many database-system users are not computer trained and do not know how to write complex queries for getting information they are looking for.
- Hence developer hide all these complexity from the users through several layers of data abstraction, in order to simplify the users' interaction with the system
- The different levels of data abstraction:
 - Physical Level
 - Logical level
 - View level



Let us now have a look at what happens at each level:

Physical Level :

- It is the lowest level of data abstraction
- It describes how the data is actually stored in the computer

Logical Level :

- This describes what kind of data is stored in the database and how different pieces of information are related to each other
- The logical level thus describes the entire database in terms of a small number of relatively simple structures.
- This level provides a more organized and simpler view of the data

Let us now have a look at what happens at each level:

View Level :

- This is the highest level of abstraction that individual users interact with and that only displays a 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.
- At this level, different users might see different parts of the database, depending on their needs and permissions

Data Abstraction

Now we can notice that:

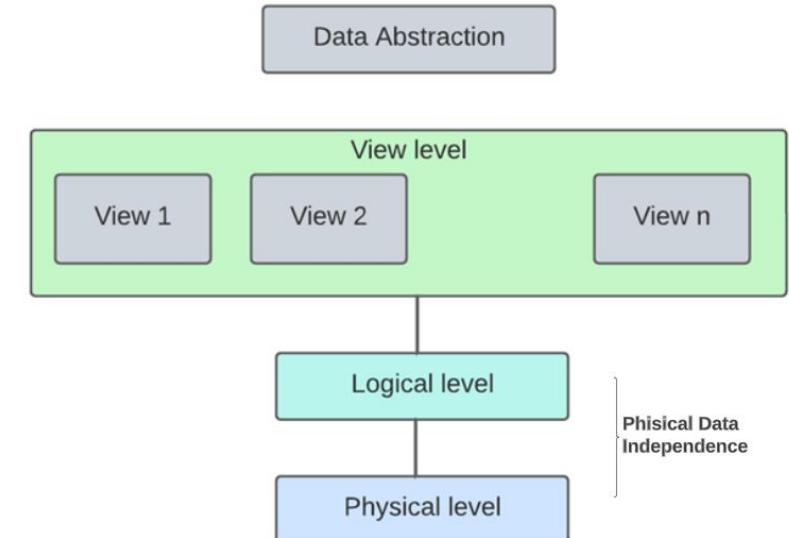
- Although the implementation of simple structures at the logical level may involve complex physical-level structures, the user at the logical level does not need to be aware of this.
- Similarly the user who views the database for various purposes need not be aware of the Logical level implementation, for them it would be like an interface and need not worry about the logical view

PHYSICAL DATA INDEPENDENCE:

- It refers to the characteristic of being able to modify the physical schema without any alterations to the conceptual or logical schema, done for optimization purposes.

Example :

- the Conceptual structure of the database would not be affected by any change in the storage size of the database system server. Changing from sequential to random access files is one such example.
- These alterations or modifications to the physical structure may include:
 - Utilizing new storage devices.
 - Modifying data structures used for storage.
 - Altering indexes or using alternative file organization techniques etc.

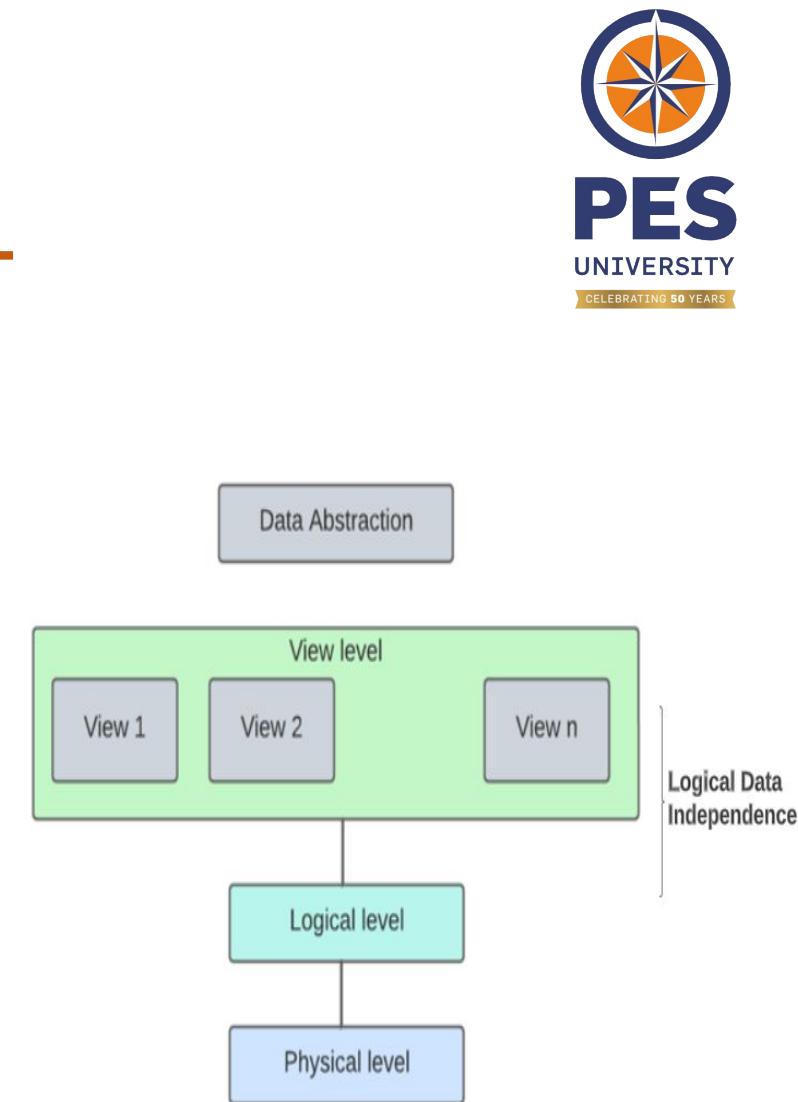


LOGICAL DATA INDEPENDENCE

- It refers characteristic of being able to modify the logical schema without affecting the external schema or application program. The user view of the data would not be affected by any changes to the conceptual view of the data.

Example:

- These changes may include the insertion or deletion of attributes,
- altering table structures entities or relationships to the logical schema, etc.



Data Abstraction

Now let us understand this with the help of an example of a University database

- Let the university have the given Record types/ tables as shown

Now for the given database,

- Physical level:**
 - Specify how student records, course information, and department data are stored on the hard drive or in memory.
- Logical Level:**
 - It would define entities like "Department," "Student," and "Course" and their attributes (e.g., department name, student ID, course code).
 - It also outlines the relationships between these entities, such as "Students belong to Departments" and "Courses are taken by Students."

Department

Dept_ID	Dept_Name	Building

Course

Course_ID	Course_Name	Credits	Dept_ID

Student

SRN	Name	Dept_ID	Total_Credits

Data Abstraction

Now let us understand this with the help of an example of a University database

- Let the university have the given Record types/ tables as shown

Now for the given database,

- View Level:**

- A faculty member may have access to a view showing only the courses they are teaching and the students enrolled in those courses.
- A student, on the other hand, may have a view displaying only the courses they are registered for and their respective grades.
- These views provide a simplified and customized perspective of the database for different users.

Similarly , how could physical and logical data independence be of an advantage in the case of the university database?

Department

Dept_ID	Dept_Name	Building
101	Computer Science	Building A

Course

Course_ID	Course_Name	Credits	Dept_ID
CS101	Computer Fundamentals	3	101

Student

SRN	Name	Dept_ID	Total_Credits
101	John Doe	101	3

Data Abstraction

Let us consider the situation where the University decides to migrate its database from one storage system to another or optimize the storage layout to improve performance. What would be the consequence of this?

Without Physical data independence:

- Minor changes to the storage system or hardware upgrades could potentially break existing queries, applications, and interfaces.
- As a result, any update to the storage system would require extensive modifications to the application layer, leading to higher maintenance costs and longer development cycles.

Department

Dept_ID	Dept_Name	Building

Course

Course_ID	Course_Name	Credits	Dept_ID

Student

SRN	Name	Dept_ID	Total_Credits

Data Abstraction

Let us consider the situation where the University decides to migrate its database from one storage system to another or optimize the storage layout to improve performance. What would be the consequence of this?

With physical data independence:

- the database administrators can make these changes without altering the logical schema, which means users' queries and applications will continue to work seamlessly.
- It allows for easier maintenance, upgrades, and performance enhancements without disrupting the application layer.

Department

Dept_ID	Dept_Name	Building

Course

Course_ID	Course_Name	Credits	Dept_ID

Student

SRN	Name	Dept_ID	Total_Credits

Data Abstraction

Let's say the University decides to reorganize its department structure and add new attributes to the "Department" entity. Then what would be the effect?

Without Logical data independence:

- Any alteration to the logical schema, such as adding or removing attributes, or changing relationships between entities, would result in changes propagating throughout the applications that rely on the database.
- This would force developers to modify all affected application code to accommodate the changes in the schema.
- Consequently, even minor changes to the data model could lead to widespread disruption and downtime for the applications, increasing the risk of introducing errors and inconsistencies.

Department

Dept_ID	Dept_Name	Building

Course

Course_ID	Course_Name	Credits	Dept_ID

Student

SRN	Name	Dept_ID	Total_Credits

Data Abstraction

Let's say the University decides to reorganize its department structure and add new attributes to the "Department" entity. Then what would be the effect?

With Logical data independence:

- The database designers can make these modifications without impacting the existing application programs that access the database.
- This flexibility allows for easier adaptability to changes in the University's requirements and business rules without causing application disruptions.

Department

Dept_ID	Dept_Name	Building

Course

Course_ID	Course_Name	Credits	Dept_ID

Student

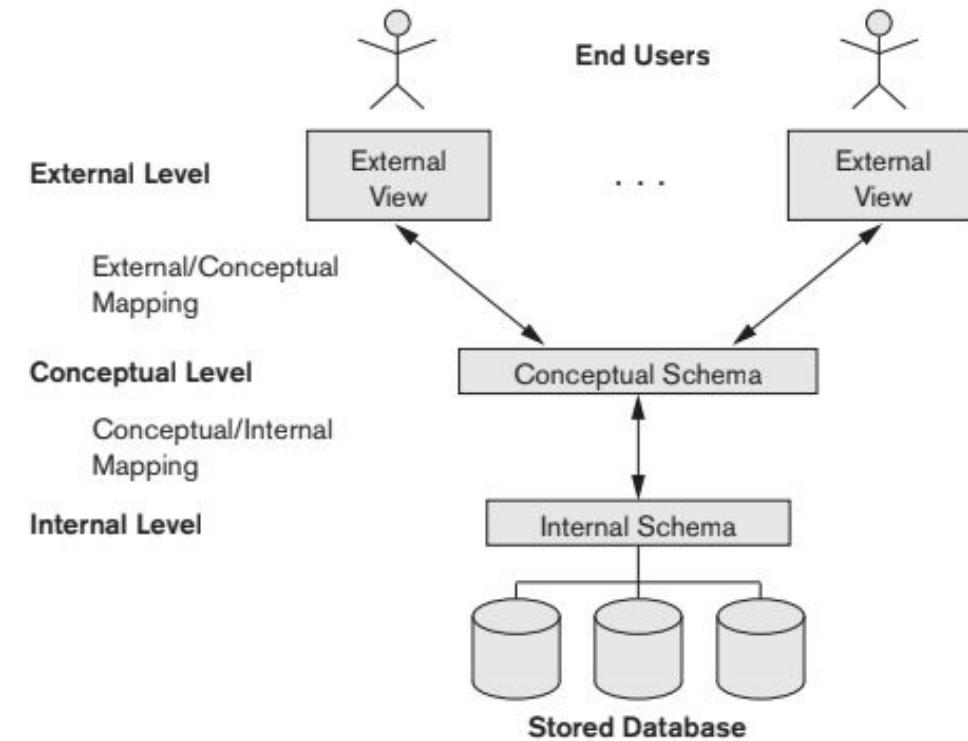
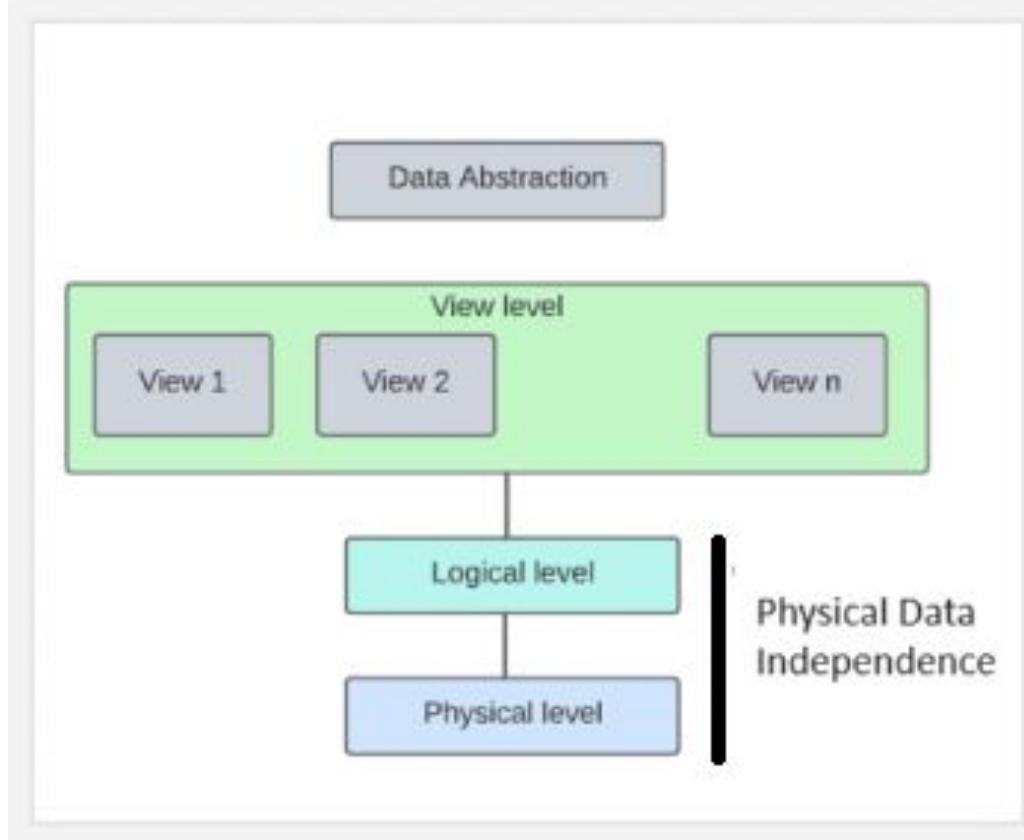
SRN	Name	Dept_ID	Total_Credits

Instances and Schema

- Databases change over time as information is inserted and deleted or modified.
- **The collection of information stored in the database at a particular moment is called an instance of the database.**
- **The overall design of the database is called the database schema.**
- Let us try to understand what that exactly means with an analogy to a program written in any programming language
 - **Database Schema** => variable declarations along with the associated type definitions
 - **Instances** => Value of the variable at a particular point in time in the program

Database Management Systems

Three Schema Architecture



Instances and Schema

The databases would have different type of schemas, partitioned according to the different levels of abstraction

- **Physical Schema/Internal Schema** : describes the database design at the physical level
- **Logical Schema/Conceptual Schema** : describes database design at the logical level
- **Subschemas/External View Schema** : there could be several schemas at the view level called subschemas, that describes different views

Considering the university Database, what would these schemas represent?

Instances and Schema

Let's consider three levels of schema and understand them w.r.t University database:

- **Physical Schema:**
 - This describes how the data is stored in the database physically. It includes details like data storage format, file organization, and indexing methods. Changes in the physical schema should not affect the applications using the database.
 - Example: How data is stored on the hard disk, **like using B-trees or hash indexes for efficient retrieval.**
- **Logical Schema:**
 - This defines the database design at a higher level, focusing on the structure and relationships between the tables. It is essential for application developers as they build software based on logical schema.
 - Example: Defining the relationships between tables, such as **the one-to-many relationship between students and courses, where one student can be enrolled in many courses, but each course can have multiple students.**

Let's consider three levels of schema and understand them w.r.t University database:

- **View Schema (Subschema):**
 - This describes different views of the database that are tailored to specific user needs or applications. Views allow users to see only relevant information and protect sensitive data.
 - Example: Creating a view that shows only the names of enrolled students and their corresponding course names, hiding other details like grades and personal information.

- Out of all the schema types, the **logical schema** holds the most significance when it comes to impacting application programs. **This is because programmers utilize the logical schema to develop applications.**
- The physical schema is hidden beneath the logical schema and can usually be changed easily without affecting application programs.
- Application programs are said to exhibit physical data independence if they do not depend on the physical schema and thus need not be rewritten if the physical schema changes.
- It is possible to create schemas that have problems, such as unnecessarily duplicated information.
 - If we store the details of the department chairperson in the faculty table, i.e. every faculty would have the attribute of “chairperson”, then there is a lot of space that gets wasted as the chairperson would be the same for the entire department and we are storing it for every faculty. Rather the better idea would be to store it in the department’s database.

Database State

- The actual data in a database may change quite frequently.
- The data in the database at a particular moment in time is called a **database state** or **snapshot**.
- It is also called the *current set of occurrences* or **instances** in the database.
- In a given database state, each schema construct has its own current set of instances.
- Many database states can be constructed to correspond to a particular database schema.

Database State V/S Schema

- The distinction between database schema and database state is very important.
- When we **define** a new database, we specify its database schema only to the DBMS.
- At this point, the corresponding database state is the empty state with no data.
- We get the initial state of the database when the database is first **populated** or **loaded** with the initial data.
- From then on, every time an update operation is applied to the database, we get another database state. At any point in time, the database has a *current state*.

Database State V/S Schema

A simplified COMPANY relational database schema

EMPLOYEE

Ename	<u>Ssn</u>	Bdate	Address	Dnumber
-------	------------	-------	---------	---------

F.K.

P.K.

DEPARTMENT

F.K.

Dname	<u>Dnumber</u>	Dmgr_ssn
-------	----------------	----------

P.K.

Sample database state for the relational database schema

EMPLOYEE

Ename	<u>Ssn</u>	Bdate	Address	Dnumber
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1

DEPARTMENT

Dname	<u>Dnumber</u>	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

Case Study: Marvel Cinematic Universe (MCU) Character Management Database:

Scenario: The Marvel Studios team manages a database for tracking MCU characters, their movies, and key events. The database includes tables for characters, movies, and appearances. The team has decided to make several updates to optimize the database and enhance data access for the creative team working on new projects and storylines.

Details:

- Implemented a B-Tree index on the 'character_id' field in the 'characters' table to speed up lookups for characters across different movies.
- Created a view named 'CharacterAppearances' that shows each character's name, the movies they appeared in, and key events related to their appearances.
- Updated the 'movies' table to include a new column for 'release_date' to better track the release schedule of MCU films.



Question: Identify the type of Schema (View, Logical, Physical) for each of the scenarios mentioned above.

Case Study: Marvel Cinematic Universe (MCU) Character Management Database:

Details:

- Implemented a B-Tree index on the ‘character_id’ field in the ‘characters’ table to speed up lookups for characters across different movies.
- Created a view named ‘CharacterAppearances’ that shows each character’s name, the movies they appeared in, and key events related to their appearances.
- Updated the ‘movies’ table to include a new column for ‘release_date’ to better track the release schedule of MCU films.

Solution:

- **Physical Schema:** The B-Tree index on the ‘character_id’ field affects the physical schema. B-Trees improve query performance by allowing efficient retrieval of character information.
- **View Schema:** Creating the ‘CharacterAppearances’ views involve the view schema.
- **Logical Schema:** Adding the ‘release_date’ column in the ‘movies’ table modifies the logical schema. This affects how the movie data is structured.





PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Database Languages

Dr. Nagasundari S

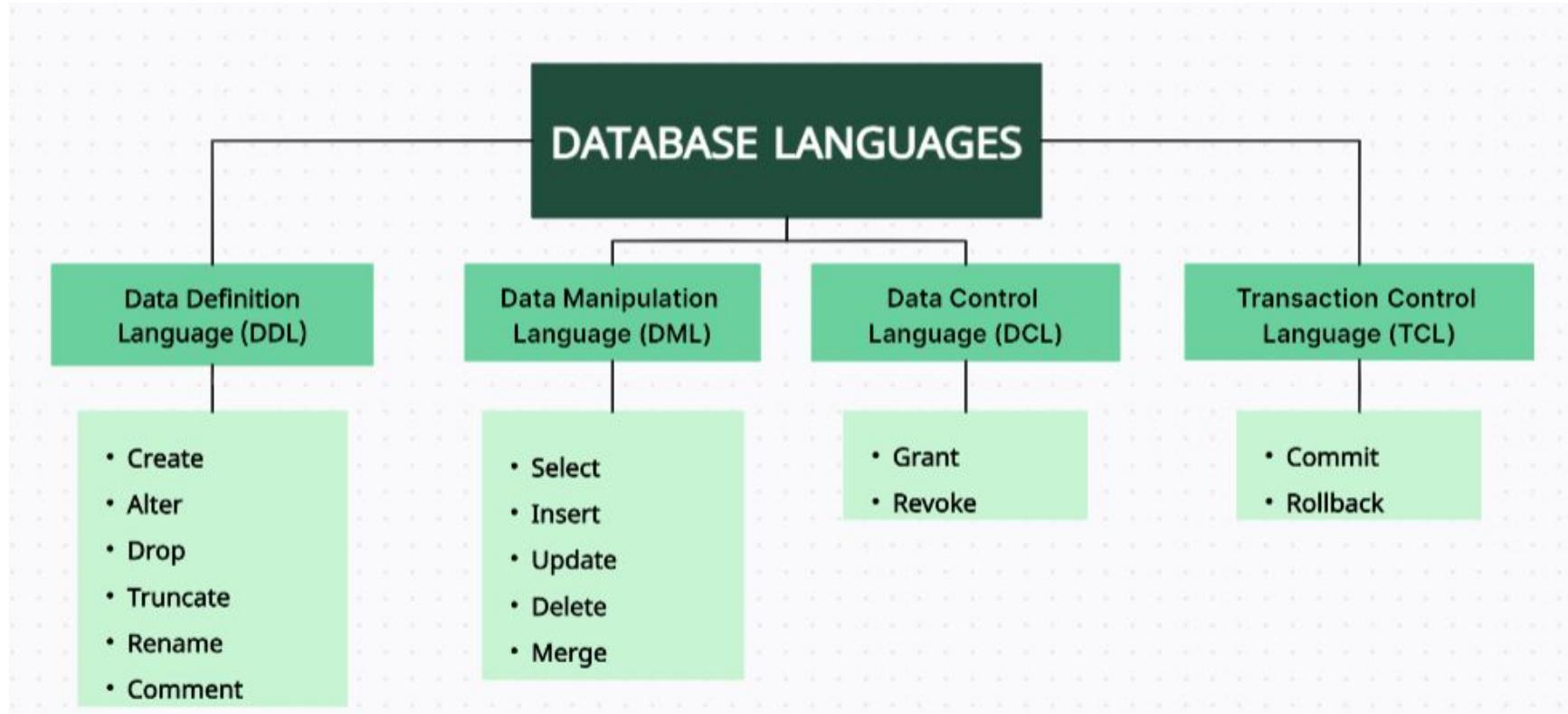
Department of Computer Science and Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering



DBMS Languages:

- Database Management Systems (DBMS) use specialized languages to define, manage, and manipulate databases.
- Key types of DBMS languages include Data Definition Language (DDL), Storage Definition Language (SDL), View Definition Language (VDL), and Data Manipulation Language (DML).
- Modern DBMS often integrate these languages into a comprehensive database language like SQL.

Data Definition Language (DDL)

- DDL is used to define the database schema, including conceptual and internal schemas.
- In systems with no strict separation between schema levels, DDL defines both.
- DDL compiler processes DDL statements, storing schema descriptions in the DBMS catalog.

SDL & VDL

- Storage Definition Language (SDL):
 - Used for specifying internal schema and storage-related details like indexing and data mapping.
 - Often combined with other functions in modern relational DBMS.
- View Definition Language (VDL):
 - Specifies user views and their mappings to the conceptual schema.
 - SQL often fulfills the role of VDL by defining application views as results of queries.

Data Manipulation Language (DML)

- DML is used to manipulate data within the database, including retrieval, insertion, deletion, and modification.
- Modern DBMS integrate DML with other languages, such as SQL, for seamless database management.
- Types of DML:
 - High-Level/Declarative(Nonprocedural): Set-oriented, declarative, used for specifying complex operations.
 - Focuses on declaring what needs to be achieved without specifying the exact steps to get there
 - SQL queries are a prime example of non-procedural DML

Example :

```
SELECT first_name, last_name, email FROM employees WHERE employee_id = 100;
```

You're specifying what data you want but not how the database should find and return that data. The database's query optimizer decides the best execution plan.

- Low-Level (Procedural):
 - Record-at-a-time, requires programming constructs like loops.
 - The developer explicitly writes out the steps needed to perform a task. This means specifying how to retrieve, insert, update, or delete data in a database.

Example :

```
DECLARE
    new_employee_id NUMBER;
BEGIN
    INSERT INTO employees (first_name, last_name, email)
    VALUES ('John', 'Doe', 'john.doe@example.com')
    RETURNING employee_id INTO new_employee_id;

    COMMIT;
END;
```

The steps are explicitly defined: insert the data, capture the ID, and then commit the transaction.

Integration in Modern DBMS

- Modern DBMS often use a comprehensive language like SQL, integrating DDL, DML, and VDL.
- Storage definitions (SDL) are managed separately to optimize database performance.
- SQL includes constructs for schema definition, data manipulation, and constraint specification.

User Interaction with DML

- High-Level DMLs can be used interactively or embedded in general-purpose programming languages.
- **Host Language:** When DML is embedded, the primary language is called the host language, and the DML is a data sublanguage.
- **Query Language:** High-level DML used standalone is called a query language, commonly used by casual end users.
- User-friendly interfaces are available for naive users to interact with databases without in-depth knowledge of DML.

Questions:

1. The _____ is used by database designers to define both conceptual and internal schemas in DBMSs where no strict separation of levels is maintained.

1. In most modern relational DBMSs, _____ performs the role of View Definition Language (VDL) to define user or application views.

1. In modern DBMSs, the Storage Definition Language (SDL) is typically used as a standalone language to define the internal schema and storage structures.

1. A high-level DML is often referred to as _____ because it specifies which data to retrieve rather than how to retrieve it.

Questions:

1. The **Data Definition Language (DDL)** is used by database designers to define both conceptual and internal schemas in DBMSs where no strict separation of levels is maintained.

1. In most modern relational DBMSs, **SQL** performs the role of View Definition Language (VDL) to define user or application views.

1. In modern DBMSs, the Storage Definition Language (SDL) is typically used as a standalone language to define the internal schema and storage structures. **False**

1. A high-level DML is often referred to as **declarative/Non-Procedural** because it specifies which data to retrieve rather than how to retrieve it.



PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Database Design

Dr. Nagasundari S

Department of Computer Science and Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering

Why Database Design ?

Scenario:

Imagine you are tasked with designing a database for a new online bookstore. The bookstore needs to manage a large catalog of books, customer information, orders, and reviews. Customers should be able to search for books, place orders, leave reviews, and track their order history.

Question

What factors would you need to consider while designing the database system to ensure it meets all the needs of the online bookstore?

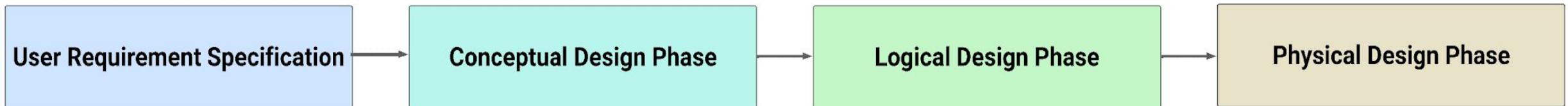
(Hint :

- Types of data Eg: books, customers, orders, reviews?
- How will you ensure data integrity and security?)



- Database design mainly involves the design of the database schema.
- The design of a complete database application environment that meets the needs of the enterprise being modeled requires attention to a broader set of issues.
- A high-level data model gives designers a conceptual framework to specify user data requirements and structure the database accordingly.

The various steps involved in the design process are :



Let us now try and understand what happens at each step:

1. USER REQUIREMENT SPECIFICATION

- Interact with domain experts and users to fully understand the data needs and functional requirements of the database.
- Identify the operations or transactions that users will perform on the data.
- Document and review the user requirements to ensure clarity and accuracy.

2. CONCEPTUAL DESIGN PHASE

- Choose a high-level data model (e.g., entity-relationship model) to conceptualize the data requirements and relationships.
- Develop a conceptual schema that provides a detailed overview of the enterprise's data structure.
- Review the schema to ensure all data requirements are met and remove any redundant features.

3. LOGICAL DESIGN PHASE

- Map the high-level conceptual schema to the implementation data model of the database system that will be used (e.g., relational model).
- Use normalization algorithms to generate a set of tables that properly group attributes and reduce data redundancy.
- Confirm that the logical design meets the functional requirements specified in the user requirements phase.

4. PHYSICAL DESIGN PHASE

- Specify the physical features of the database, including file organization and internal storage structures.
- Consider performance optimization and **choose appropriate indexing** techniques to enhance data retrieval speed.
- Review and refine the physical design to ensure it aligns with the requirements and constraints of the database system and hardware.



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Database Engine

Dr. Nagasundari S

Department of Computer Science and Engineering

Database Engine

A database system is partitioned into modules that deal with each of the responsibilities of the overall system.

The functional components of a database system can be divided into

- The storage manager,
- The query processor component,
- The transaction management component.

Storage Manager

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

The storage manager is responsible for the following tasks:

- Interaction with the OS file manager
- Efficient storing, retrieving and updating of data

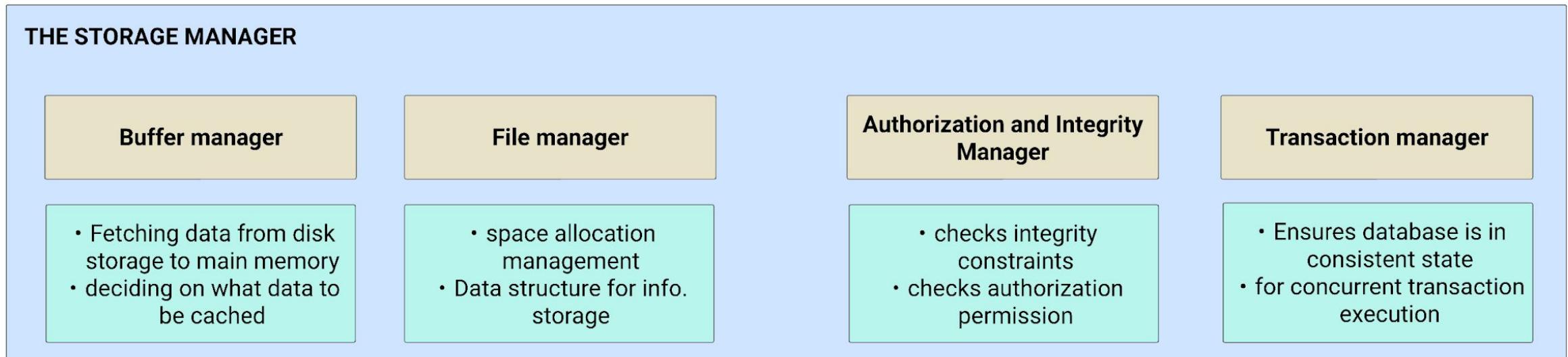
The storage manager implements several data structures as part of the physical system implementation:

- Data files -- store the database itself
- Data dictionary -- stores metadata about the structure of the database, in particular the schema of the database.
- Indices -- can provide fast access to data items. A database index provides pointers to those data items that hold a particular value.

Storage Manager

The storage manager components include:

- Authorization and integrity manager
- Transaction manager
- File manager
- Buffer manager



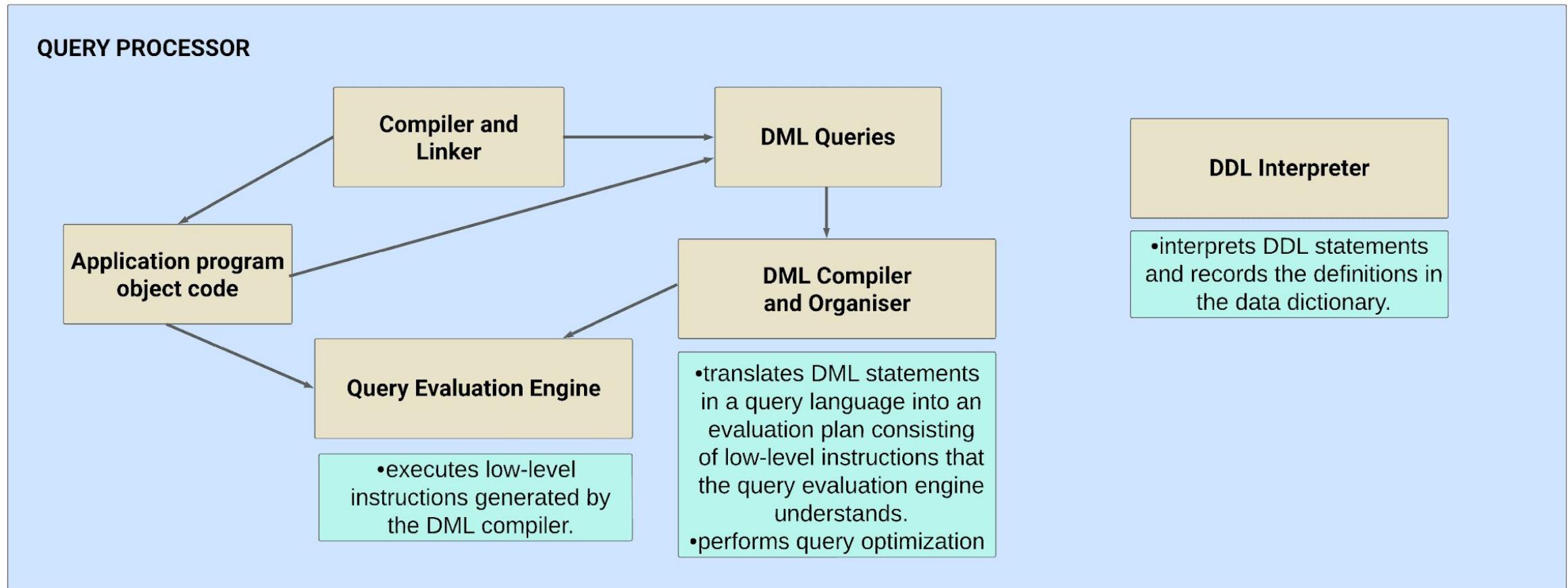
Query Processor

- The query processor is important because it helps the database system to simplify and facilitate access to data.
- The query processor allows database users to obtain good performance while being able to work at the view level and not be burdened with understanding the physical-level details of the implementation of the system.

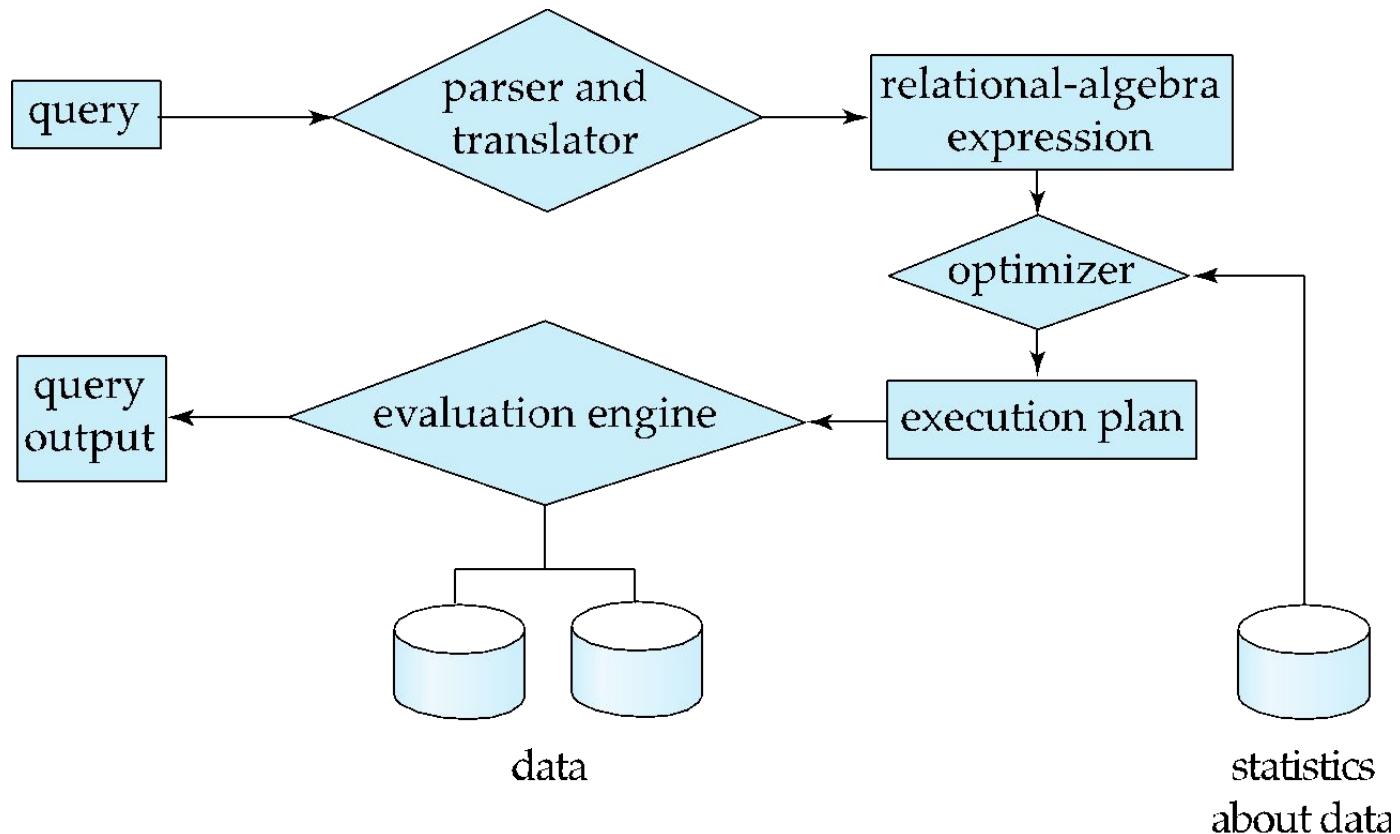
The query processor components include:

- DDL interpreter
- DML compiler
 - performs **query optimization**, i.e. it picks the lowest cost evaluation plan from among the various alternatives.
- Query evaluation engine

- The internal structure of query optimizer:



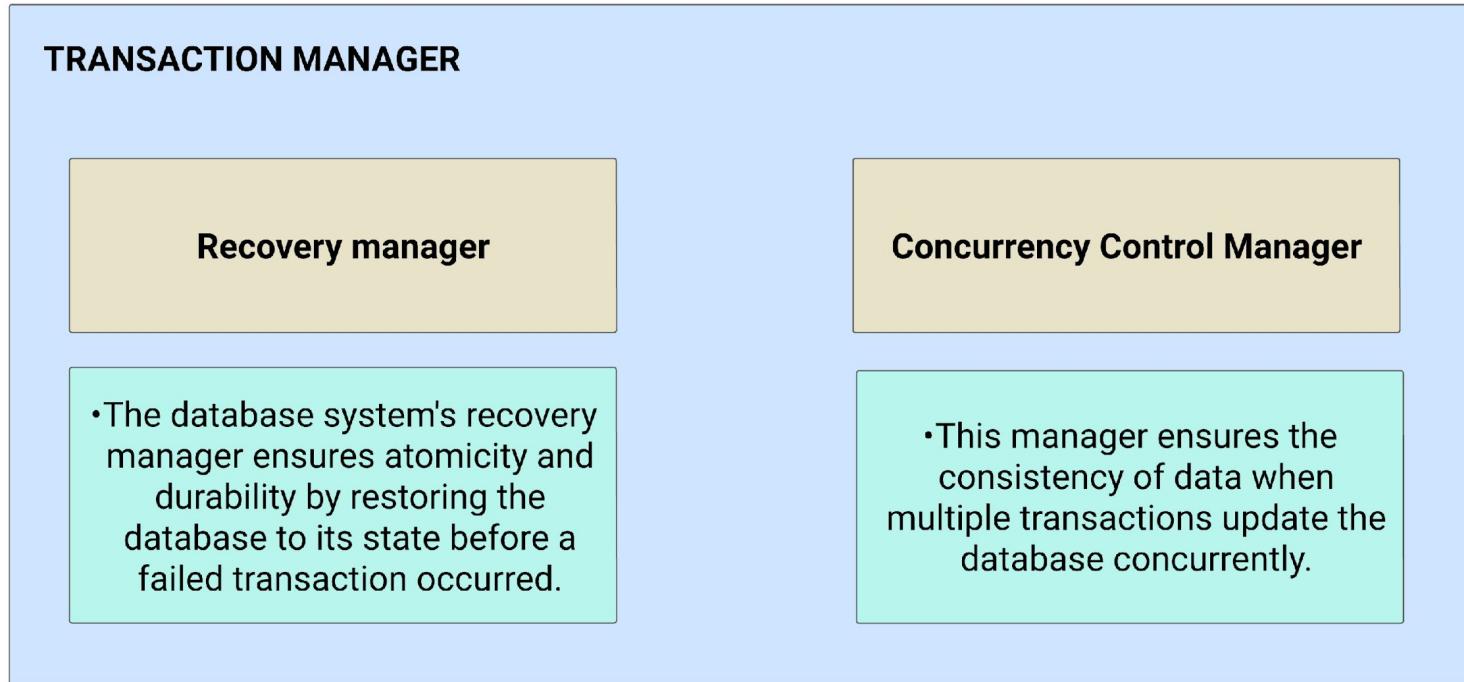
- 1. Parsing and translation
- 2. Optimization
- 3. Evaluation



Transaction Management

- A transaction is a collection of operations that performs a single logical function in a database application
- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- Transaction properties: A transaction is a logical unit of work that must maintain atomicity (all-or-none execution), consistency (database remains valid throughout), and durability (changes are permanent even after system failure).
- Consistency and atomicity: During the execution of a transaction, temporary inconsistency may be allowed, but the database must return to a consistent state after the transaction completes successfully.

- The transaction Manager



Application of transactions: Transactions are widely used in various applications, including financial, telecommunication, and long-duration activities like product design or administrative workflows.

Database Management Systems

Database System Structure

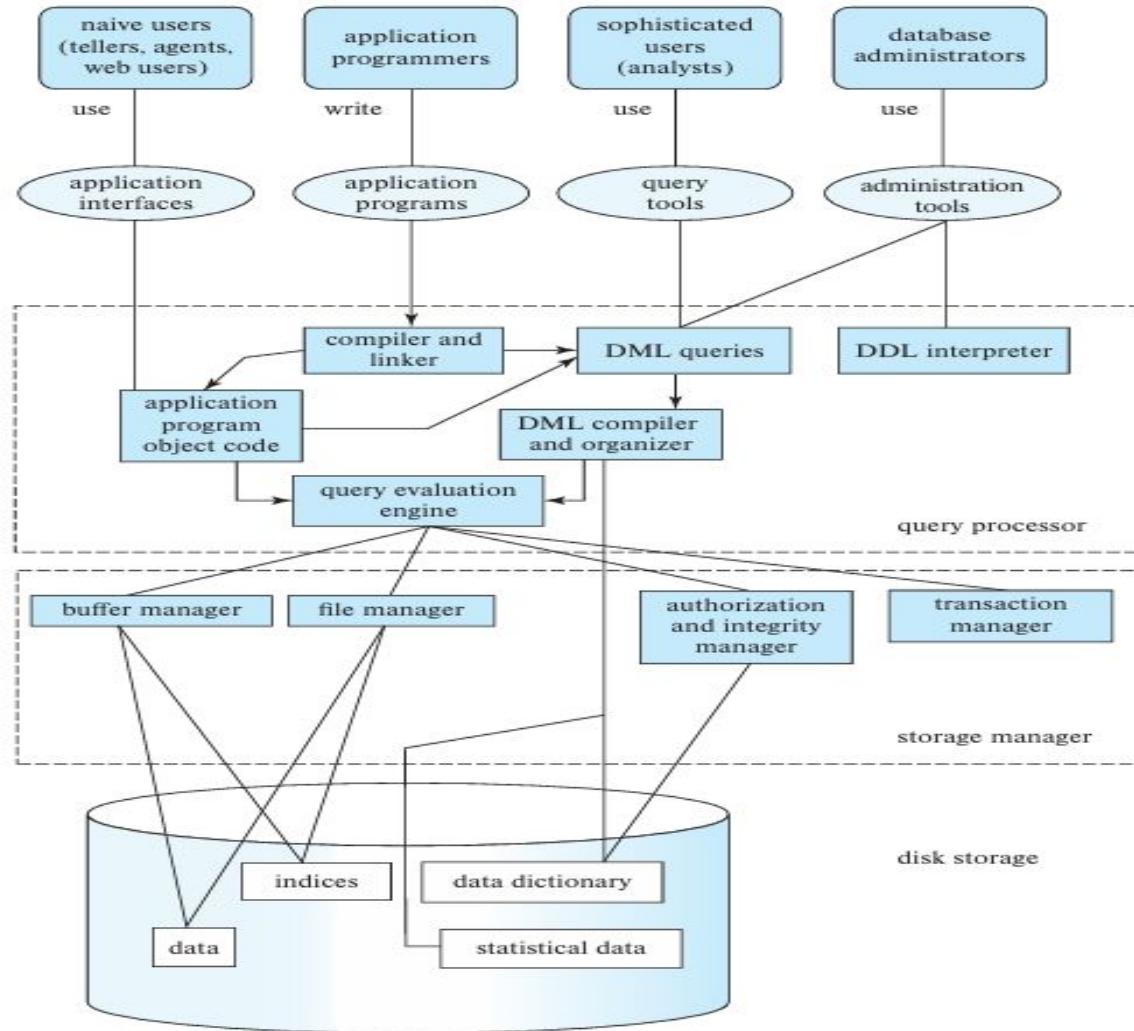


Figure 1.3 System structure.



PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



Database Management Systems

Database and Application Architecture

Dr. Nagasundari S

Department of Computer Science and Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering

Scenario:

Imagine you are responsible for the database system of a popular online multiplayer game. The game has thousands of players online simultaneously, from different parts of the world. Each player needs real-time updates for their game stats, friends list, and in-game purchases.

Question:

How would you design the database system to ensure efficient, secure, and real-time data access for all players?



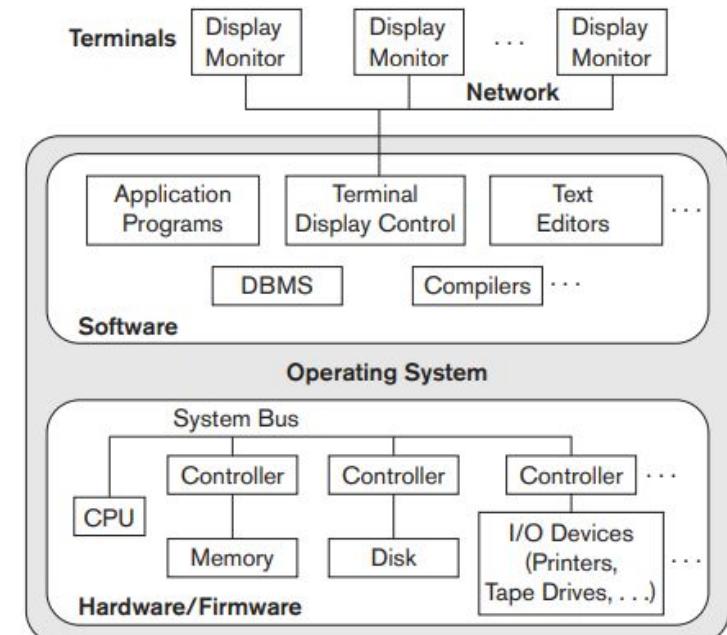
- Centralized databases (Centralized architecture)

Characteristics:

- **Single Location:** All data is stored and managed on a central server or database.
- **Shared Memory:** Suitable for systems with multiple CPUs that access a common shared memory space.
- **Resource Allocation:** Resources like CPU, memory, and storage are centralized, making it easier to manage and maintain.

Use Cases:

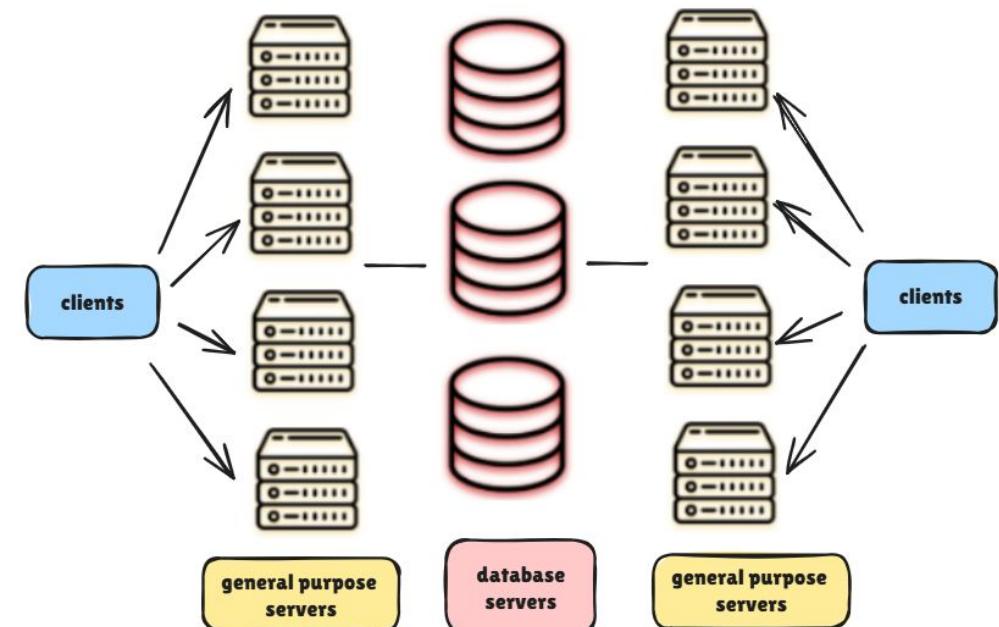
- **Small to Medium-Sized Applications:** Suitable for applications with moderate data needs and user loads.
- **Legacy Systems:** Often used in older systems where scaling is not as critical or feasible.



Centralized/Shared Memory DB

Client Server Architecture:

- Client-server architecture is a model where a client (the user or application) interacts with a server (the database system) to access and manage data.
- The client is the user-facing component that interacts with the server to request data or perform operations. (web browser, mobile apps, desktop applications)
- The server is the backend component that manages the database, processes requests from clients, and handles data storage and retrieval. (MySQL, Oracle, Postgres)



Parallel databases:

- Designed to run on a cluster of multiple machines
- Enables better scalability and higher processing capabilities.
- Many cores shared memory and Shared Disk

Parallel databases are ideal for **high-performance applications** requiring **extensive data processing**, such as large-scale data warehousing, real-time analytics, and **high-throughput** transactional systems.

Example: How do you think NASA manages and processes the vast amounts of satellite data it collects?

Answer: NASA uses parallel databases and high-performance computing (**HPC**) systems. NASA can efficiently handle and analyze massive datasets from satellite missions, enabling quick and insightful analysis for complex tasks like climate modeling and environmental monitoring.



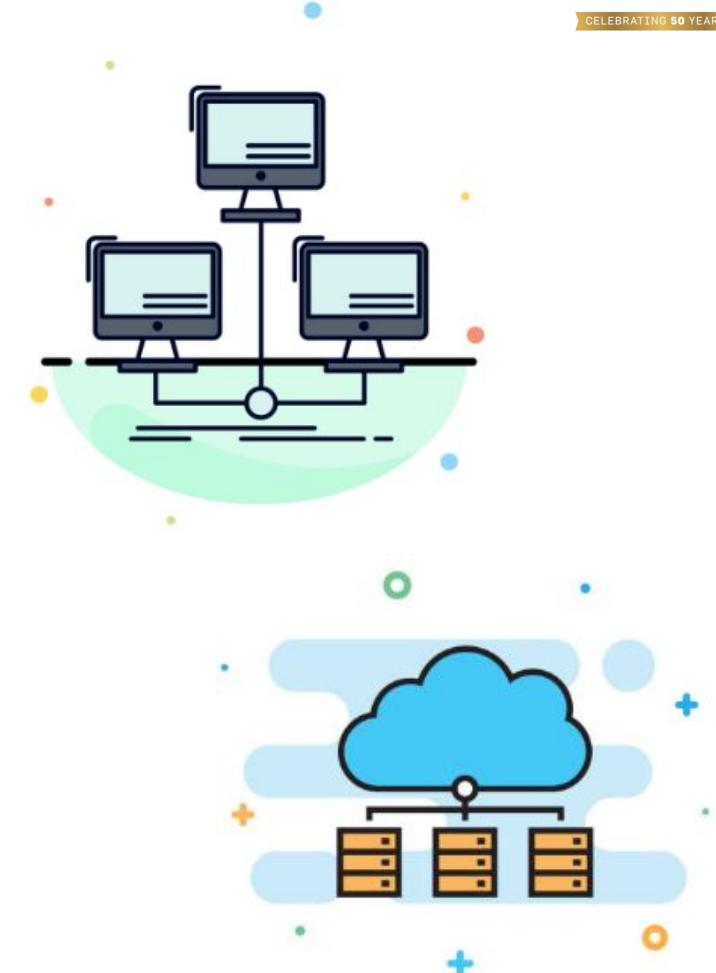
NASA's HPC Cluster

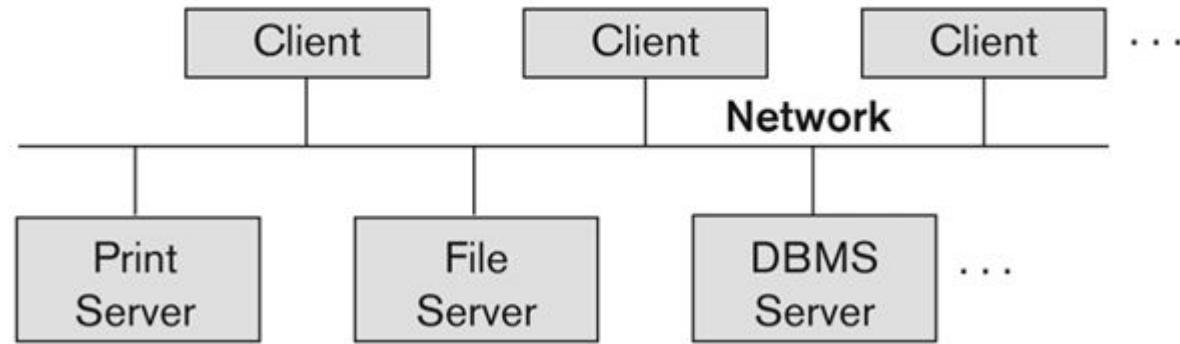
Distributed Database:

A distributed database is a database system in which data is stored across multiple physical locations. These locations can be spread across various machines within a single data center or across different geographic locations.

Advantages:

- **Scalability:** Can handle large amounts of data and high traffic by distributing the load across multiple nodes.
- **Fault Tolerance:** Provides high availability and fault tolerance through data replication and distribution.
- **Performance:** Can improve performance by distributing queries and data access across multiple servers.





In a two-tier database architecture, there are two main components: the client (**user interface**) and the server (**database**).

- **Client:** This is the user interface where users interact with the application. It can be a desktop application or a web-based interface.
- **Server:** This is where the database is stored and managed. The server handles data storage, retrieval, and management tasks.
- Client programs send query and transaction requests via ODBC or JDBC API, the server processes these requests and sends the results back to the client, which then processes and displays the results.

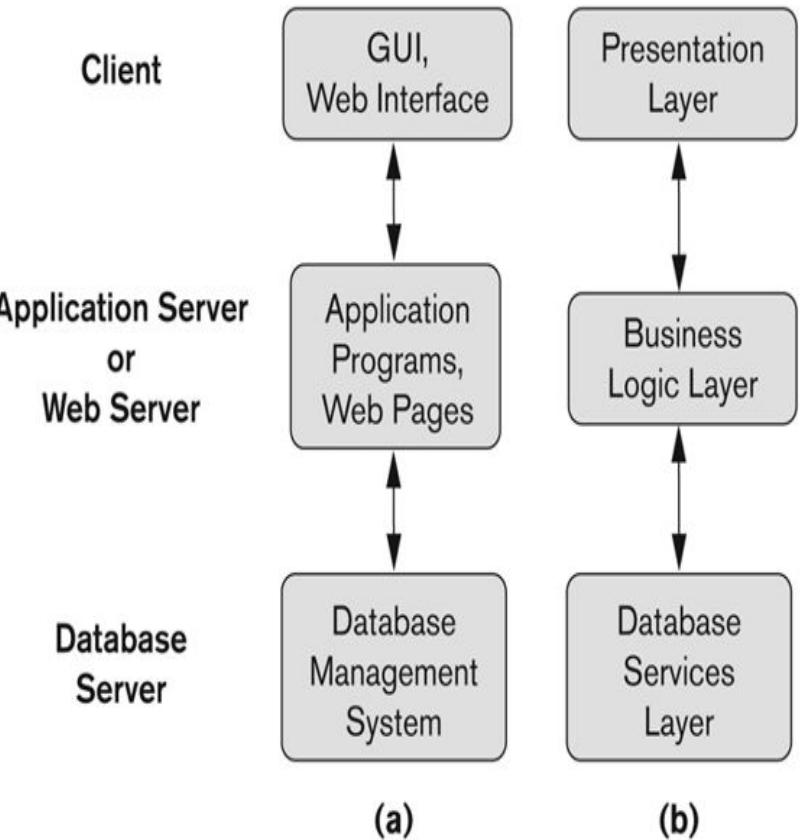
Clients

- Provide appropriate interfaces through a client software module to access and utilize the various server resources.
- Clients may be diskless machines or PCs or Workstations with disks with only the client software installed.
- Connected to the servers via some form of a network.
- (LAN: local area network, wireless network, etc.)

Servers

- Provides database query and transaction services to the clients
- Relational DBMS servers are often called SQL servers, query servers, or transaction servers
- Applications running on clients utilize an Application Program Interface (**API**) to access server databases via standard interface such as:
- ODBC: Open Database Connectivity standard
- JDBC: for Java programming access
- Client and server must install appropriate client module and server module software for ODBC or JDBC

- Common for Web applications
- Intermediate Layer called Application Server or Web Server:
 - Stores the web connectivity software and the business logic part of the application used to access the corresponding data from the database server
 - Acts like a conduit for sending partially processed data between the database server and the client.
- Three-tier Architecture Can Enhance Security:
 - Database server only accessible via middle tier
 - Clients cannot directly access database server



Scenarios for Architecture Selection:

1. **Weather Forecasting System:** A weather forecasting system collects data from various sensors and satellites, processes the data to generate forecasts, and distributes the information to users through a web interface and mobile application.
2. **E-Learning Platform:** An e-learning platform offers online courses to students around the world. The platform needs a database to manage course content, track student progress, and support interactive features such as quizzes and discussion forums.
3. **Social Media Platform:** A social media platform with millions of users worldwide needs a database system that can handle high read and write loads, store large amounts of user-generated content, and provide fast access to data.



Scenarios for Architecture Selection: Solutions -

- **Weather Forecasting System:** Parallel database architecture.

This architecture can handle large volumes of data from sensors and satellites, process it efficiently, and distribute forecasts quickly through web and mobile interfaces.

- **E-Learning Platform:** 3-tier architecture.

This architecture supports the management of course content, tracking of student progress, and interactive features like quizzes and discussion forums, ensuring scalability and efficient data access.

- **Social Media Platform:** Distributed database architecture.

This architecture supports high read and write loads and large amounts of user-generated content, providing fast access to data for users worldwide.



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Database Users & Administrators

Dr. Nagasundari S
Department of Computer Science and Engineering

Database Users

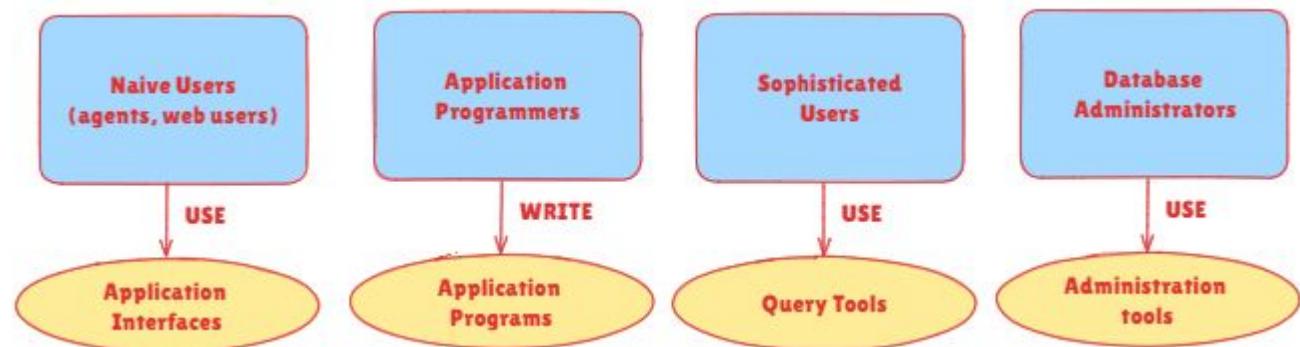
People who work with a database can be categorized as:

1. Database Users
2. Database Administrators.

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 different types of users.

1. Naïve Users
2. Application Programmers
3. Sophisticated Users
4. Database Administrators



Naïve Users:

- Naive users also known as **Parametric End users**, don't have any knowledge of databases but still frequently use the database applications to get the desired results. With the help of the interface provided by the DBMS applications.
- **For Example:** Have you ever tried to book a ticket for your favorite movie? It's straightforward and easy.
- But, do you have any idea about the complex database operations happening behind the scenes to make this process smooth and efficient?
- In database terminology, you are a **naive user** or a **parametric end user**. This means you interact with the database through a predefined interface without needing to know anything about how the database works.



Application programmers :

- Application Programmers also known as Back-End Developers, are computer professional users who are responsible **for developing the application programs.**
- **Example:** For an e-commerce site, application programmers connect the app to the database, write queries to display products, manage user registrations and orders, handle database errors, and secure user data.

Sophisticated Users:

- Sophisticated users are individuals who possess a deep understanding of database management systems (DBMS) and are proficient in using Data Definition Language (DDL) and Data Manipulation Language (DML) commands.
- **Example:** Data engineers and developers often access databases directly using SQL queries to fetch, delete, update, or insert data, making them sophisticated users.

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data.

Database Administrator is an individual or a team of users who defines the database schema and takes charge of controlling various levels of the database within the organization.

Functions of a Database Administrator (DBA):

- 1. Schema definition:** DBA creates the original database schema using data definition statements (DDL).
- 2. Storage structure and access-method definition:** DBA specifies parameters for data physical organization and index creation.
- 3. Schema and physical-organization modification:** DBA makes changes to reflect organization's needs or enhance performance.
- 4. Granting authorization for data access:** DBA regulates user access by granting different types of authorization.
- 5. Routine maintenance:** DBA performs periodic backups, ensures disk space availability, and monitors database performance.

Questions:

1. "Application programmers are responsible for defining storage structures and access methods in the database." – True/False
1. "A _____ user interacts with a system using predefined interfaces and does not need to write code."
1. "The role of a _____ includes creating the original database schema and managing user access."
1. "What type of user would most likely use a query language to retrieve specific data from a database?"

Questions:

1. "Application programmers are responsible for defining storage structures and access methods in the database." – False (Database Administrators handle this)
1. "A Naïve user interacts with a system using predefined interfaces and does not need to write code."
1. "The role of Database Administrator includes creating the original database schema and managing user access."
1. "What type of user would most likely use a query language to retrieve specific data from a database?" - Sophisticated User



PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Database Design - Overview

S Nagasundari

Department of Computer Science Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering

Recap – Design Phases

- As we have seen before the design phases would involve the following steps:

- Characterizing User Data Needs
- Choosing a Data Model and Translating Requirements
- Review and Refinement of Schema
- Functional Requirements Specification
- Logical-Design Phase
- Physical-Design Phase

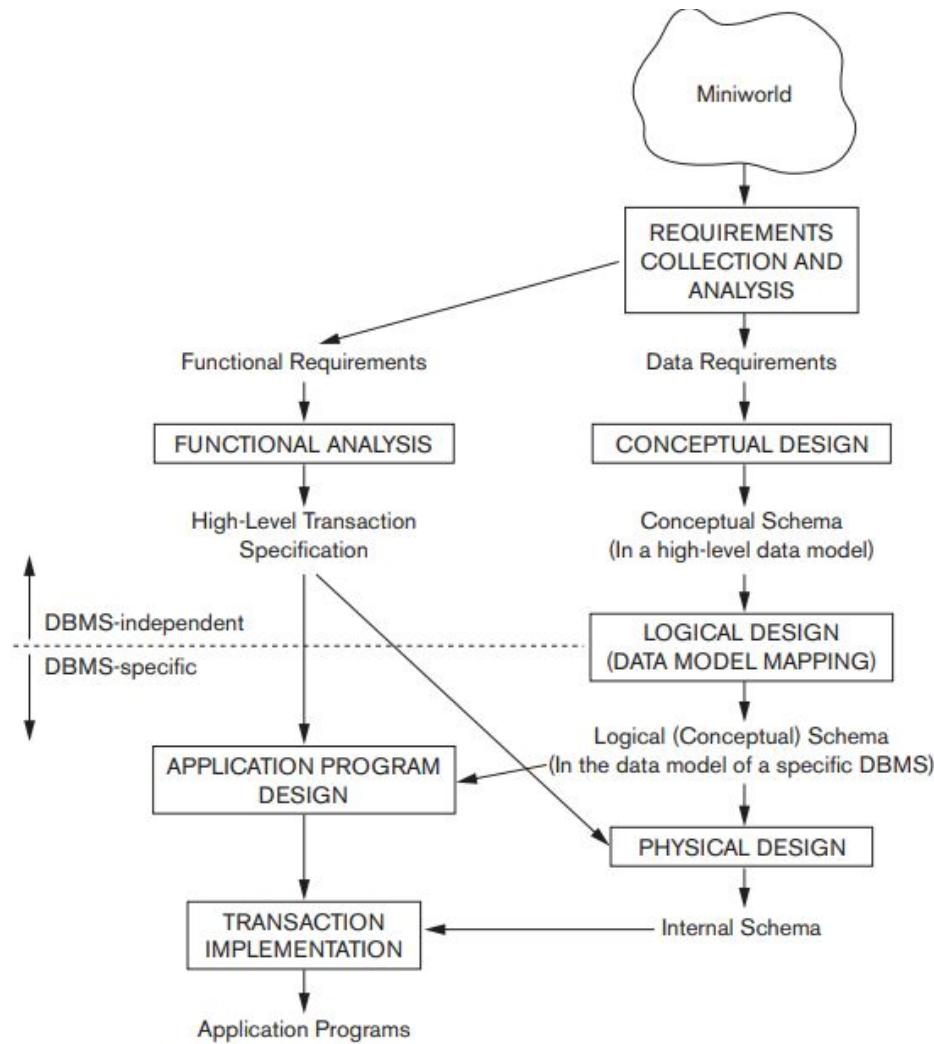


Fig : A simplified diagram to illustrate the main phases of database design

Let us just have a look at the design phases again:

Characterizing User Data Needs:

- Interaction with domain experts and users.
- Specification of user requirements.

Choosing a Data Model and Translating Requirements:

- Selection of a data model.
- Translation of requirements into the conceptual schema.
- Using entity-relationship model for conceptual design.

Review and Refinement of Schema:

- Confirming data requirement satisfaction.
- Eliminating redundant features.

Let us just have a look at the design phases again:

Functional Requirements Specification:

- Users describe operations (transactions) on data.
- Ensure schema meets functional requirements.

Logical-Design Phase:

- Mapping conceptual schema to implementation data model.
- Using entity-relationship model for relational design

Physical-Design Phase:

- Using system-specific database schema.
- Specifying physical features (file organization, indexes).

Why do we need a design phase?

You are designing a database for an e-commerce platform. The platform is running smoothly, but after launch, you realize the database schema didn't account for tracking customer reviews.

Problem: *Adding the reviews feature now requires significant changes to the database schema, which impacts numerous queries and updates scattered across the application code. This leads to extended downtime and increased costs.*

Why is it crucial to thoroughly plan and carry out the database design phase before building the rest of the application?



Why do we need a design phase?

- The physical schema of a database can be changed relatively easily after an application has been built.
- However, changes to the logical schema are usually harder to carry out, since they may affect a number of queries and updates scattered across application code.
- It is therefore important to carry out the database design phase with care, before building the rest of the database application.

- A major part of the database design process is deciding how to represent in the design the various types of “**things**” such as people, places, products, and the like.
- We use the term **entity** to refer to any such distinctly identifiable item.
 - Considering a university database, the entities would be:
 - instructors
 - students
 - departments
 - courses, etc.
- The various entities are **related** to each other in a variety of ways, all of which need to be captured in the database design.
 - For example:
 - A student takes a course offering
 - An instructor teaches a course offering

- While designing a database we must try to avoid two major pitfalls:
 - **Redundancy**
 - **Incompleteness**
- Let us try and understand what these are with respect to the university database system, and what are their drawbacks if not handled

Redundancy

- A bad design leads to repeating the same information unnecessarily which would lead to **redundancy**.
- For example in the university database, let us that we are storing the department_ID along with the department name for every course in the course entity
 - Course(Department_ID, department_name, course_ID, course_name, credits)

Course_ID	Course_Name	Credits	Department_ID	Department_Name
101	Database Systems	4	1	Computer Science
102	Algorithms	3	1	Computer Science
201	Calculus	4	2	Mathematics

In this situation, a department (Computer Science) offers many courses, leading to unnecessary repetition of storing the department name along with the ID for each course (**redundancy**).

- It would suffice to store only the department_ID with each course and to associate the department name with the Department_ID only once, in a department entity

Department entity:

Department_ID	Department_Name
1	Computer Science
2	Mathematics

Course entity:

Course_ID	Course_Name	Credits	Department_ID
101	Database Systems	4	1
102	Algorithms	3	1
201	Calculus	4	2

- Redundancy can also occur in a relational schema.
- Imagine a single table with all course information repeated for each section.
- This creates unnecessary redundancy, as each course details are duplicated.

- **Problem with Redundancy:**
 - Redundant copies of data can become inconsistent if updated without care.
 - Different sections of a course could end up having different titles if not properly managed.
- **Example: Inconsistent Course Titles:**
 - Multiple offerings of a course might have the same identifier but different titles.
 - This inconsistency makes it unclear which title is correct.
- **Optimal Data Management:**
 - Ideally, information should be stored in one place to avoid redundancy.
 - A course's title should be stored only in a single entity (like the course entity) and linked to offerings as needed.

Incompleteness

- For instance, if we only have **entities** for course offerings(Sections) and not for the actual courses, it creates difficulties in modeling.
 - Imagine if we only track when courses are offered but don't have a separate entity for courses.
 - This design limitation would make it impossible to represent new courses that aren't currently offered.
- Similarly in **Relations**:
 - In terms of relational databases, having a single table with repeated course information for each section causes problems.
 - This approach prevents us from representing new courses that don't yet have offerings.

- **Challenge in Representing New Information:**

- With this flawed design, adding details about a new course becomes tricky, as it must have an associated section to fit the structure.
- Attempting to use null values for section information is a less desirable workaround.

- **Drawbacks of Workaround:**

- Using null values to represent missing information is not an elegant solution.
- It might run into issues due to constraints like primary-key requirements.

- **Avoiding Poor Designs is Insufficient:**
 - Simply avoiding bad designs is not enough in database design.
 - Multiple good design options might be available, creating the need to select wisely.
- **The Complexity of Design Choices:**
 - For instance, consider a simple scenario where a customer buys a product.
 - A key decision arises: Is the sale a relationship between customer and product, or is the sale itself a distinct entity connected to both customer and product?
- **Impact of Design Choices:**
 - This choice can significantly influence the effective modeling of various enterprise aspects.
 - Different design approaches might lead to different insights and representations of business operations.

- **Scale of Decision-Making:**
 - In real-world scenarios, numerous entities and relationships require similar decisions.
 - This makes database design a challenging task, demanding careful consideration.
- **Database Design Challenge:**
 - The complexity of design choices highlights the intricate nature of database design.
 - It involves determining suitable structures for various entities and relationships.
- **Combination of Science and Aesthetic Judgment:**
 - Database design necessitates a balanced approach of scientific principles and intuitive "good taste."
 - Successful design involves a mix of methodical analysis and informed decision-making.



PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Entity-Relationship Model

S Nagasundari

Department of Computer Science Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering

- **What is an Entity?**

- An entity is a distinct "thing" or "object" in the real world.
- It's different from other objects and can be a person, course, or more.

- **Properties of an Entity:**

- An entity has specific attributes or properties.
- Some properties must hold unique values to tell entities apart.

- **Example - Student Entity:**

- In a university, each student is an entity.
- A unique "Student_ID" property distinguishes individuals.

The Entity-Relationship (E-R) model is a data modeling technique designed to assist in database design.

It enables the creation of an enterprise schema that captures the logical structure of a database.

Purpose and Function: The E-R model aids in representing real-world entities, their meanings, and interactions within a conceptual schema for database design.

- E-R diagrams provide a graphical representation of the database's logical structure. They use symbols to depict entities, relationships, and attributes, making complex relationships easier to understand.
- E-R diagrams are valued for their simplicity and clarity, making them a widely used tool in database design.
- By mapping out the entities, attributes, and relationships before implementation, E-R Diagrams help in identifying potential issues, redundancies, or inconsistencies in the database design. This leads to more efficient and optimized database structures, reducing the need for costly modifications later.
- E-R Diagrams serve as a comprehensive documentation tool that records the structure of the database.

- **Unique Identification of entity:**
 - Unique property values identify each entity.
 - For instance, a student's "student_id" value like PES001 makes them distinct.
 - Courses can also be entities in a university setting.
 - A "course_id" attribute uniquely identifies each course.
- **Concrete and Abstract Entities:**
 - Entities can be concrete (like people or books) or abstract (like courses or reservations).
 - Both types serve as distinct objects in the system.

- **Entity Set (or Entity Collection):**

- An entity set is a group of entities of the same kind sharing common attributes or properties.
- For instance, "instructor" represents all university instructors, and "student" represents all university students.

E1 is an entity having Entity Type Student and the set of all students is called Entity Set.

- **Extension vs. Entity Set:**

- The real collection of entities is the "extension" of an entity set.
- Similar to the difference between a relation and a relation instance.

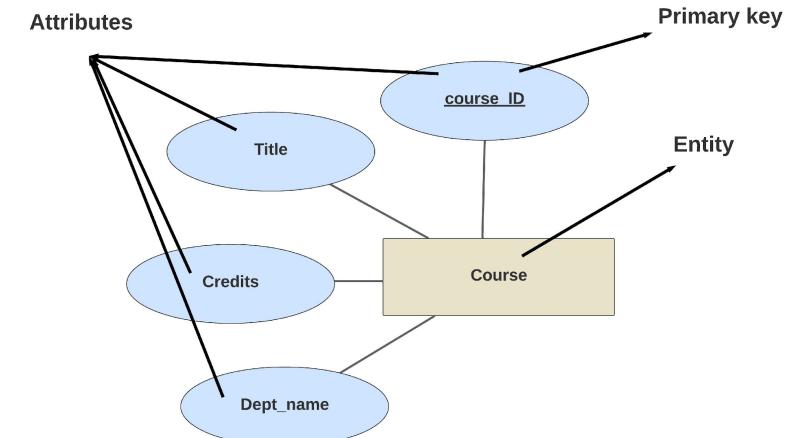
- **Non-Disjoint Entity Sets:**

- Entity sets can overlap these sorts of entity sets are called non-disjoint sets.
- Example: "person" entity set includes instructors, students, both, or neither.

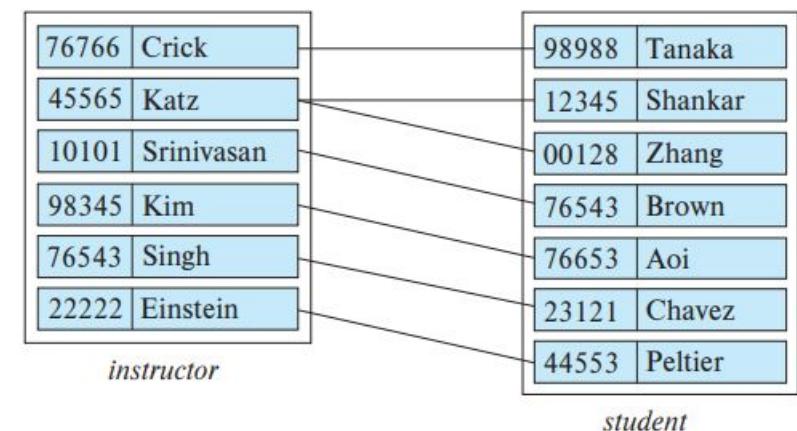
Entity Type Name:	EMPLOYEE	COMPANY
Entity Set: (Extension)	Name, Age, Salary	Name, Headquarters, President
	$e_1 \bullet$ (John Smith, 55, 80k) $e_2 \bullet$ (Fred Brown, 40, 30K) $e_3 \bullet$ (Judy Clark, 25, 20K) ⋮	$c_1 \bullet$ (Sunco Oil, Houston, John Smith) $c_2 \bullet$ (Fast Computer, Dallas, Bob King) ⋮

Figure 3.6
Two entity types,
EMPLOYEE and
COMPANY, and some
member entities of
each.

- Entity and Attribute representation:
 - An E-R diagram uses rectangles to depict entity sets, and ovals to represent the attributes.
- Primary Key Indication:
 - Primary key attributes are underlined.
- For instance, let us consider a university database
 - Universities track various aspects, leading to multiple entity sets.
 - Courses is an entity set with attributes like course id, title, dept name, and credits
 - For courses the ER representation is as shown

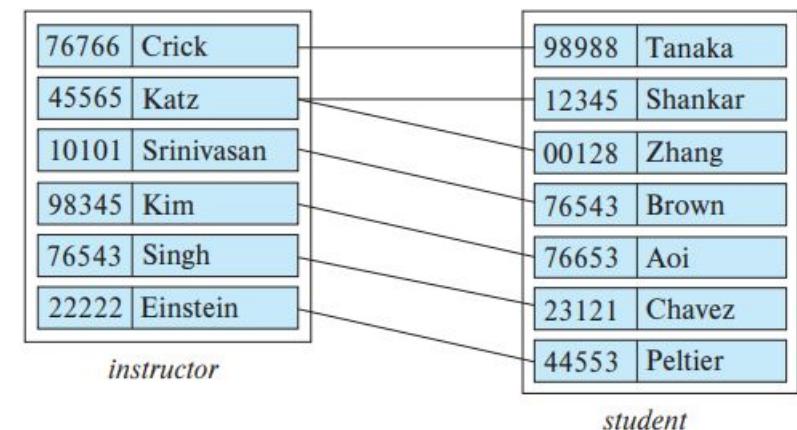


- **What is a Relationship:**
 - A relationship signifies connections between entities.
 - Example: "advisor" links instructor Katz to student Shankar.
- **Relationship Set:**
 - A relationship set contains relationships of the same type.
 - Example: "advisor" relationship set links students and their advisors.
- **Entity Sets and Relationships:**
 - Consider "instructor" and "student" entity sets.
 - "advisor" relationship set signifies student-advisor connections.

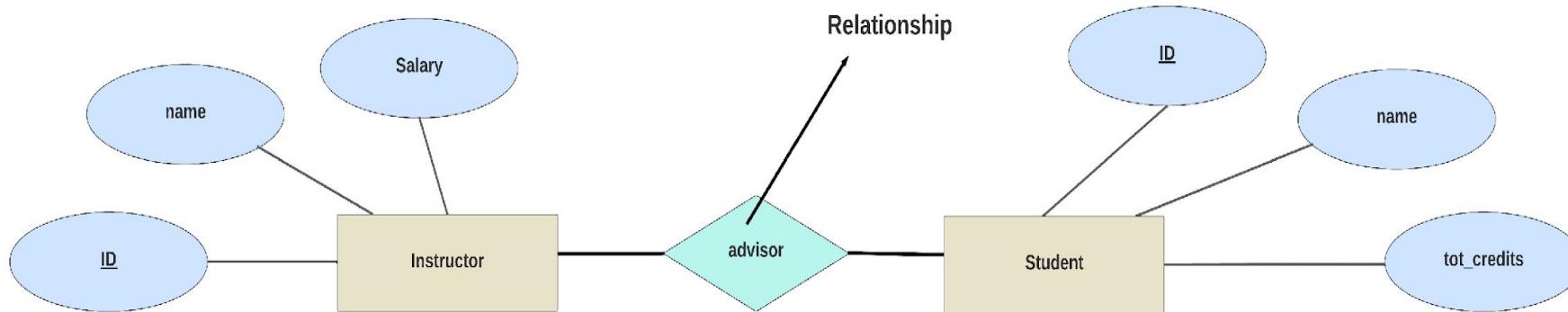


- **Relationship Instance:**

- A relationship instance in an E-R schema represents an association between the named entities in the real-world enterprise that is being modeled.
- As an illustration, the individual instructor entity Katz, who has instructor ID 45565, and the student entity Shankar, who has student ID 12345, participate in a relationship instance of advisor.
- This relationship instance represents that in the university, the instructor Katz is advising student Shankar



- **Representation in E-R Diagram:**
 - In an E-R diagram, relationships are depicted by diamonds.
 - Lines connect the diamond to relevant entity sets (rectangles).
- For example let us try and represent the advisor relation



Relationship sets

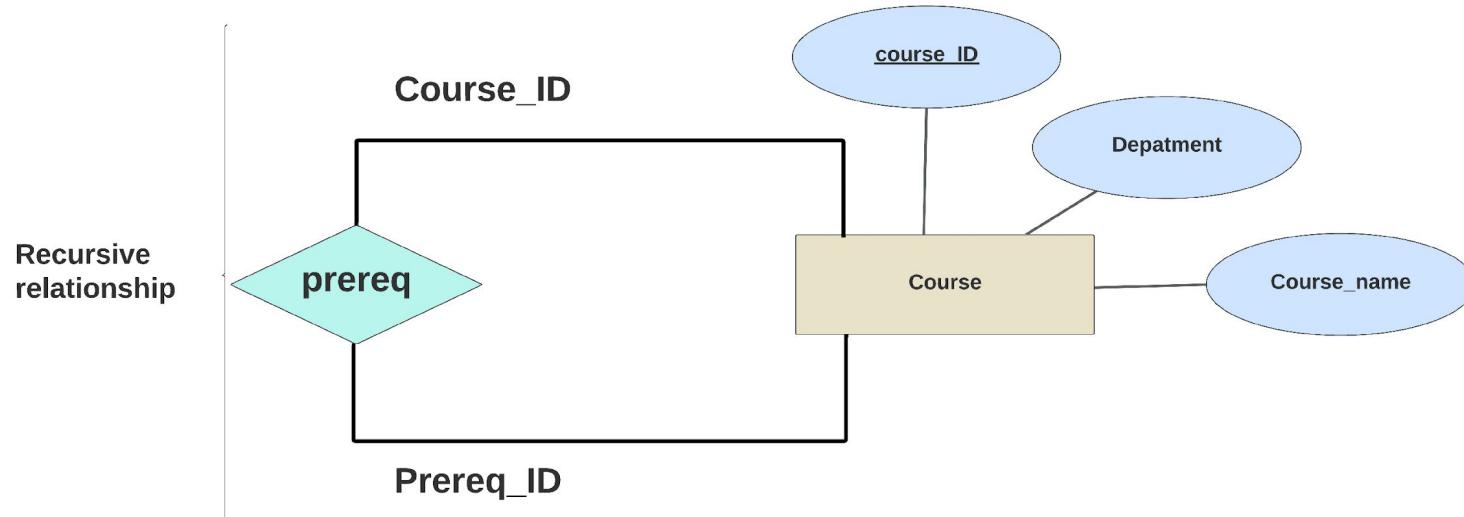
- A relationship type R among n entity types E₁, E₂, . . . , E_n defines a set of associations—or a relationship set—among entities from these entity types.
- Similar to the case of entity types and entity sets, a relationship type and its corresponding relationship set are customarily referred to by the same name, R.
- Mathematically, the relationship set R is a set of relationship instances r_i, where each r_i associates n individual entities (e₁, e₂, . . . , e_n), and each entity e_j in r_i is a member of entity set E_j, $1 \leq j \leq n$.
- Hence, a relationship set is a mathematical relation on E₁, E₂, . . . , E_n; alternatively, it can be defined as a subset of the Cartesian product of the entity sets $E_1 \times E_2 \times \dots \times E_n$.
- Each of the entity types E₁, E₂, . . . , E_n is said to participate in the relationship type R; similarly, each of the individual entities e₁, e₂, . . . , e_n is said to participate in the relationship instance r_i = (e₁, e₂, . . . , e_n).

Recursive Relationships

- The function that an entity plays in a relationship is called that entity's **role**.
- Since entity sets participating in a relationship set are generally distinct, roles are implicit and are not usually specified.
- However, they are useful when the meaning of a relationship needs clarification. Such is the case when the entity sets of a relationship set are not distinct; that is, the same entity set participates in a relationship set more than once, in different roles.
- In this type of relationship set, sometimes called a **recursive relationship set**, explicit role names are necessary to specify how an entity participates in a relationship instance.

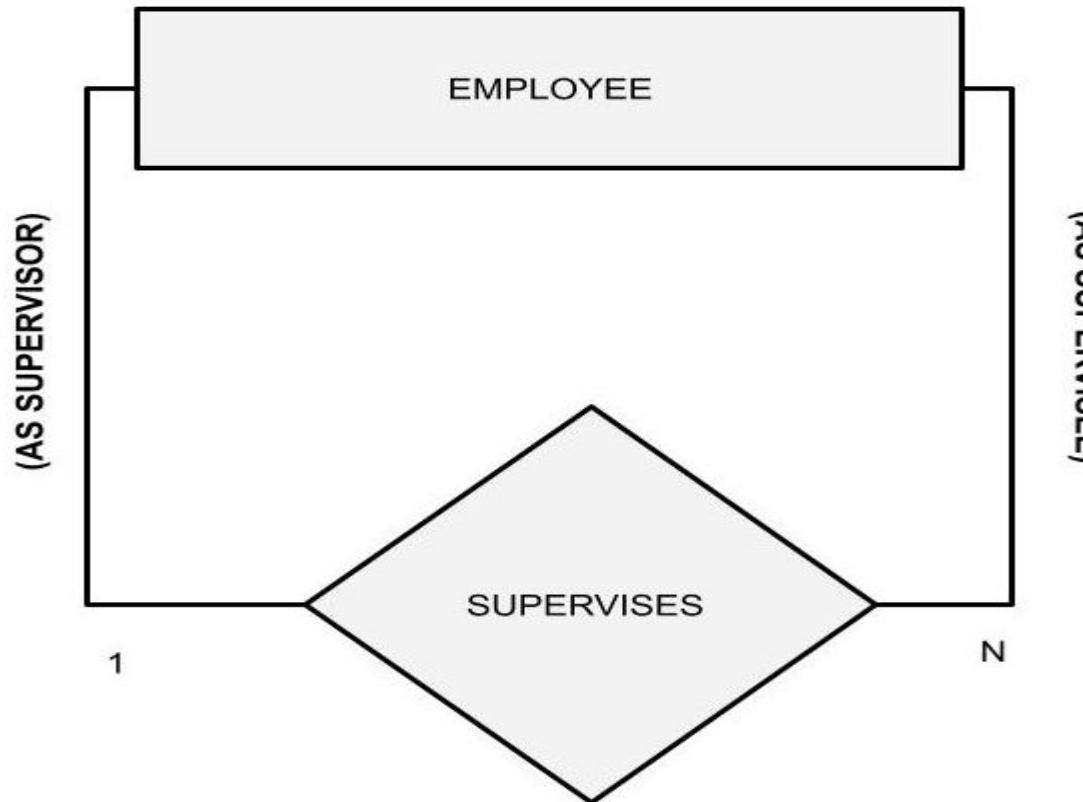
- Let us try and understand it with the help of an example
- consider the entity-set course that records information about all the courses offered in the university. Now some courses that are offered by the university have some prerequisite courses which are needed to be completed
- To depict the situation where one course (C2) is a prerequisite for another course (C1) we have a relationship set **prereq** that is modeled by ordered pairs of course entities.
- So how do we indicate this in the ER diagram?

- We indicate roles in E-R diagrams by labeling the lines that connect diamonds to rectangles.
- The diagram below shows the role indicators course id and prereq id between the course entity set and the prereq relationship set.



Recursive Relationship - Employee - Supervisor Example

Can you think of another example of a recursive-relationship?



Relationship Set and it's degree

- The relationship sets **advisor** is an example of a **binary relationship set**—that is, one that involves two entity sets.
- Most of the relationship sets in a database system are binary.
- Occasionally, however, relationship sets could even involve more than two entity sets. The number of entity sets that participate in a relationship set is the **degree of the relationship set**.
 - A binary relationship set is of degree 2;
 - A ternary relationship set is of degree 3.

Relationship Set Example

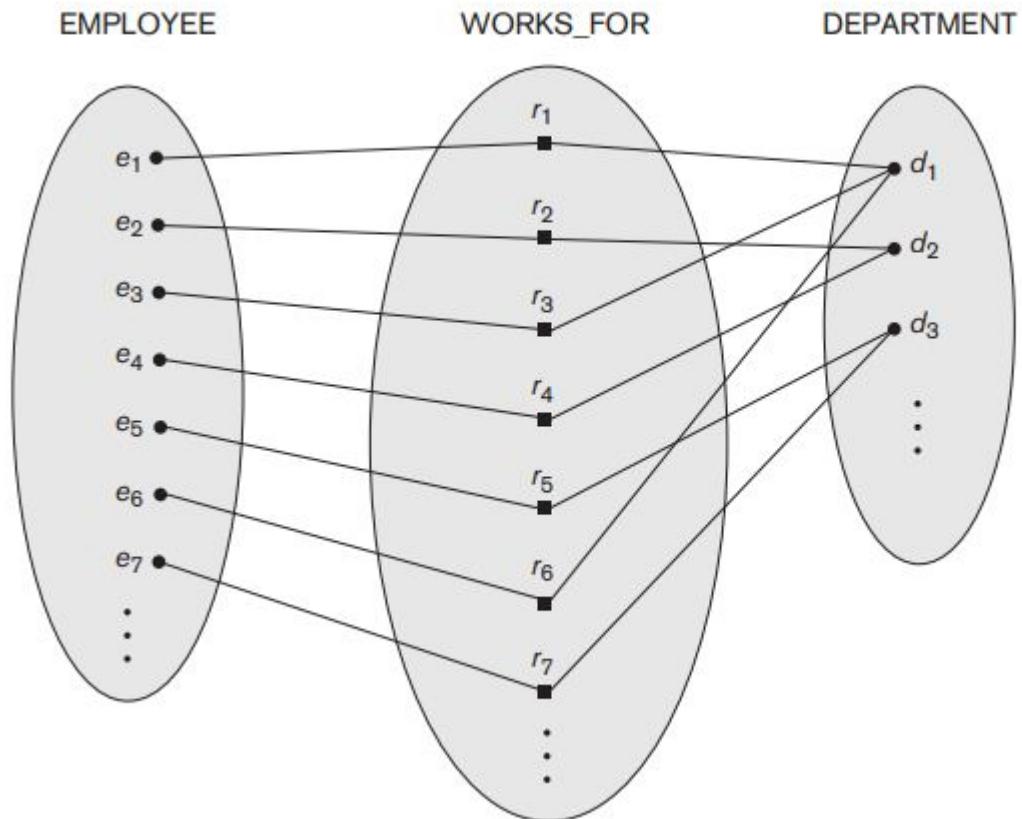
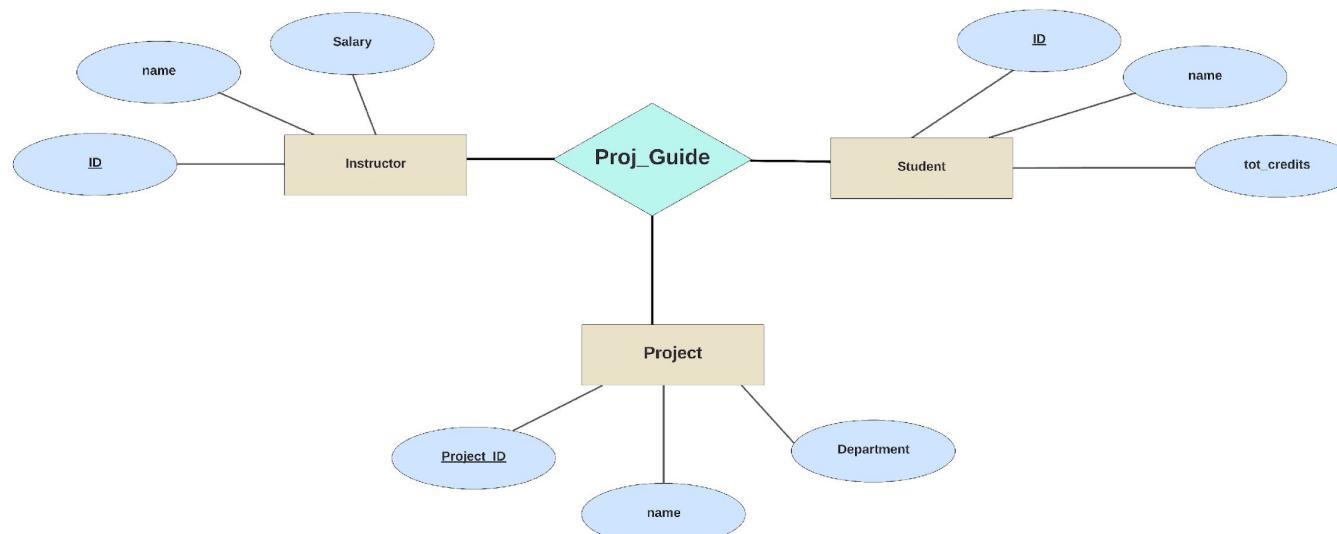


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

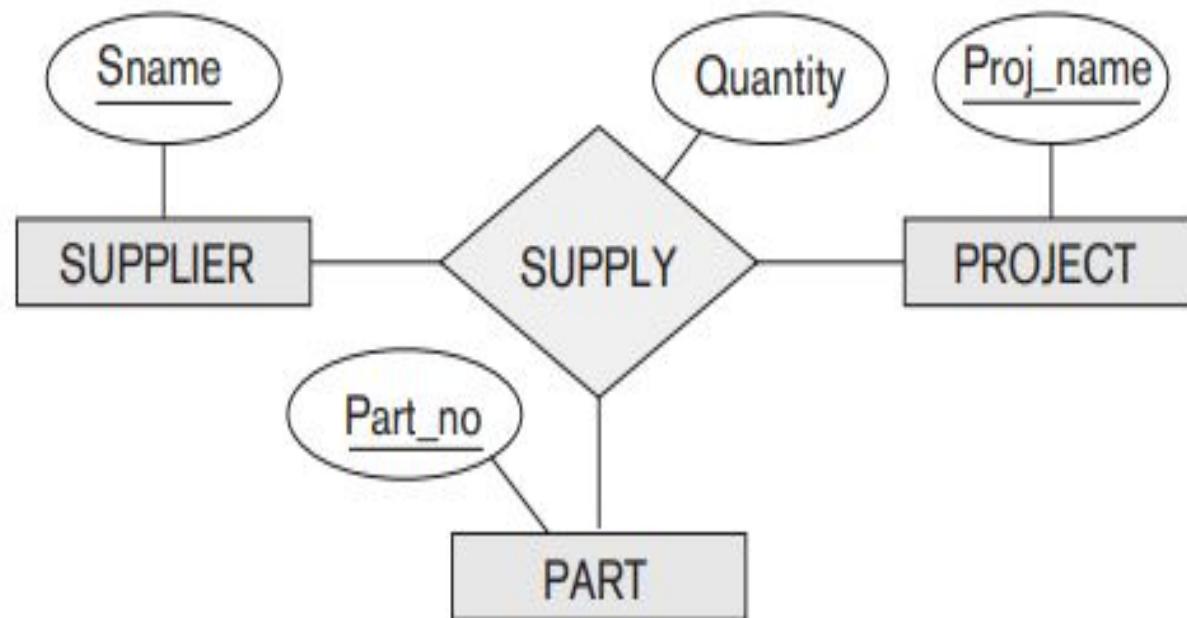
- Let us consider an example, suppose that we have an entity set project that represents all the research projects carried out in the university.
- Consider the entity sets instructor, student, and project. Each project can have multiple associated students and multiple associated instructors. Furthermore, each student working on a project must have an associated instructor who guides the student on the project.
- Suppose we were to represent the information of which instructor is guiding which student for which project
- How do we represent this or model this in an ER diagram?

Ternary Relationship Example

- To represent the above situation, we would have to relate the three entity sets through a **ternary relationship** set **proj_guide**, This ternary relation relates entity sets instructor, student, and project.
- An instance of projguide indicates that a particular student is guided by a particular instructor on a particular project.
- Note that a student could have different instructors as guides for different projects, which cannot be captured by a binary relationship between students and instructors.



Ternary Relationship Example





PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Attributes

S Nagasundari

Department of Computer Science Engineering

Database Management Systems

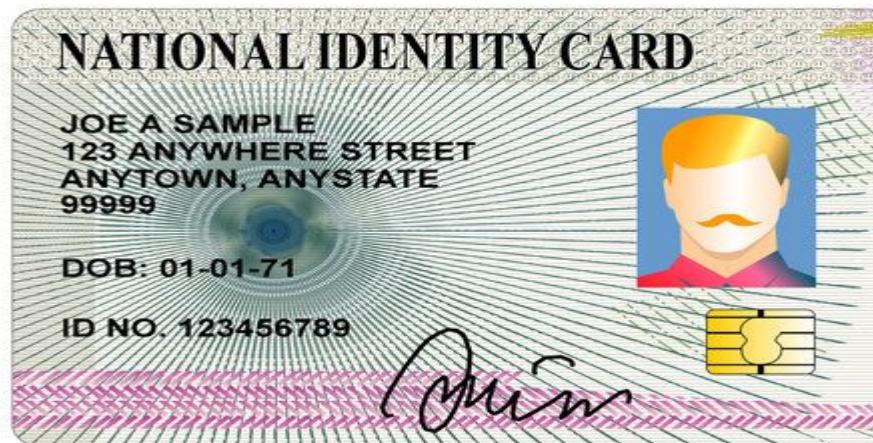
Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering

Attributes

- An entity's attributes provide distinct values for identification.
 - For instructors, the ID attribute stands out as a unique identifier.
- It prevents confusion when multiple instructors share the same name.
- How do we decide on the unique identifier for the case of Instructors?
 - Government-issued IDs were historically used, but they raise security and privacy concerns.
 - Enterprises opt to establish their own identifiers to ensure security and privacy.



- Entities are represented by specific attributes.
- Attributes are qualities or characteristics of each entity.
- They signify the type of information stored for each entity.
- Every entity possesses its own set of values for attributes.
- Example: Instructors have unique values for attributes like ID, name, dept name, and salary.

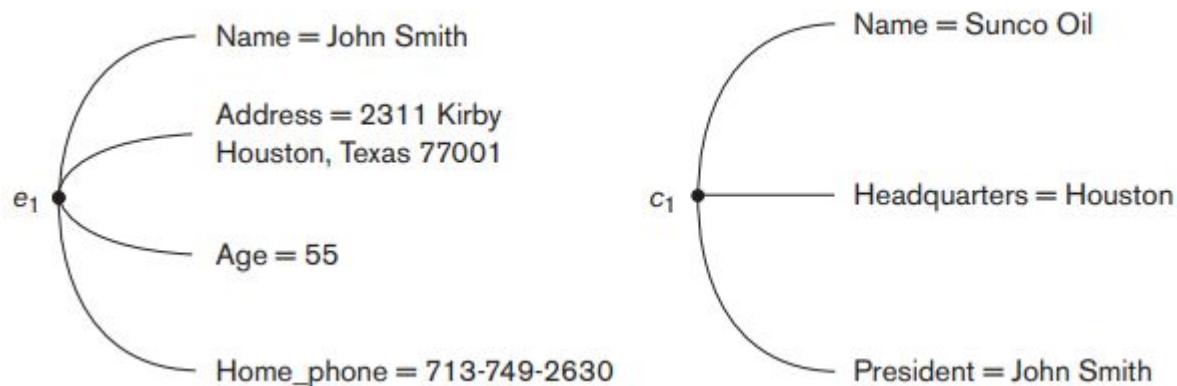
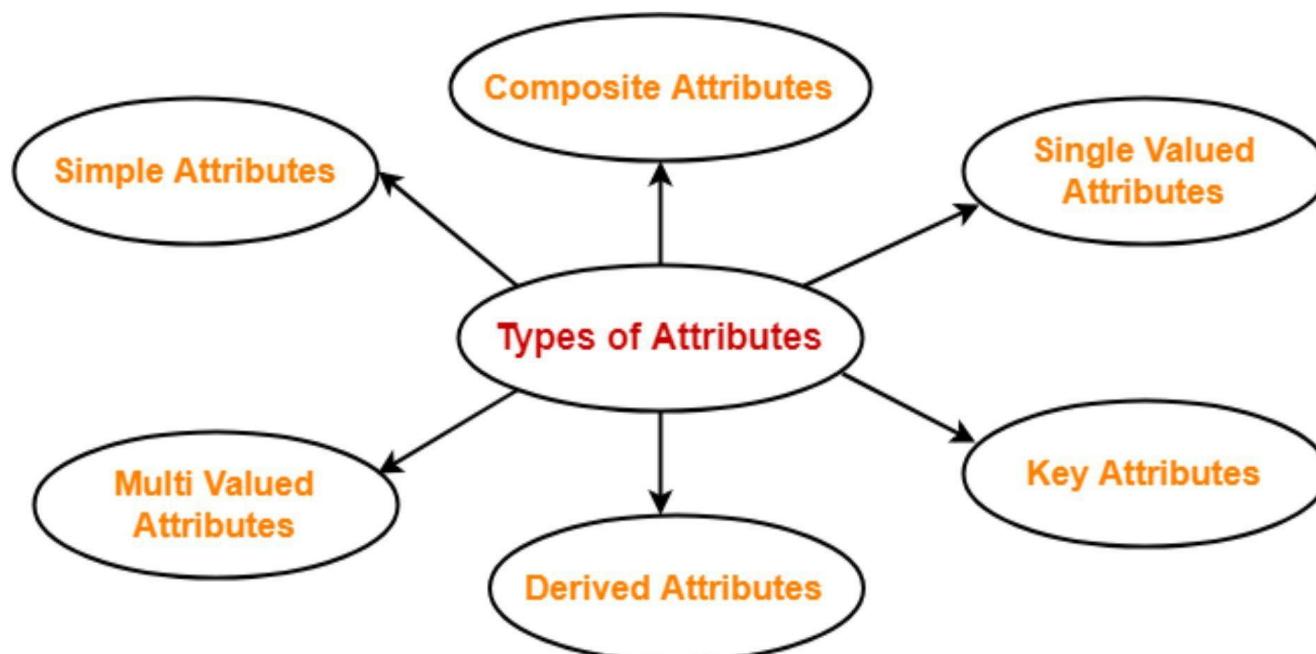


Figure 3.3
Two entities,
EMPLOYEE e_1 , and
COMPANY c_1 , and
their attributes.

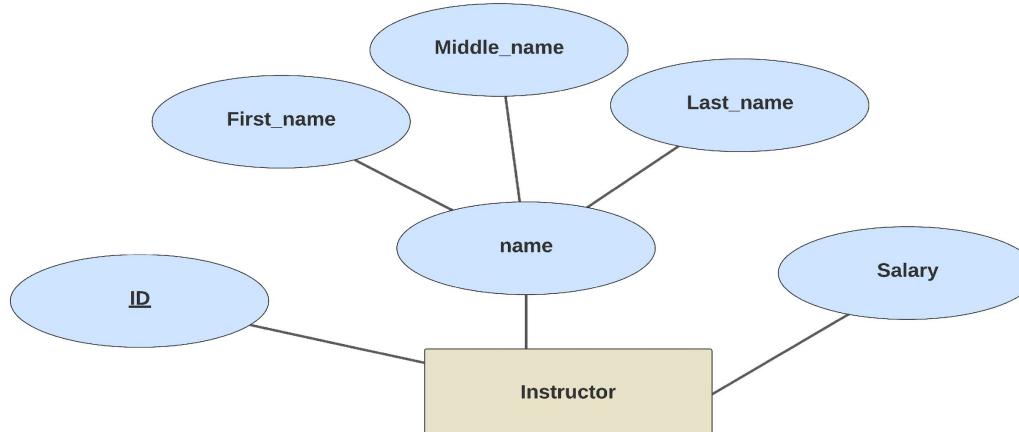
- For each attribute there are a set of permitted values that can be taken which is called DOMAIN or VALUE SET of that attribute.
 - If the range of ages allowed for employees is between 16 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70.
 - We can specify the value set for the Name attribute to be the set of strings of alphabetic characters separated by blank characters, and so on.
- Mathematically, an attribute A of entity set E whose value set is V can be defined as a function from E to the power set⁶ P(V) of V:
$$A : E \rightarrow P(V)$$

- An attribute used in E-R model, can be characterized as:
 - Simple / Composite attribute
 - Single-valued / Multivalued attributes
 - Derived attributes



Simple and Composite Attribute

- Formally saying the attributes that cannot be further subdivided into components is a **simple attribute**
 - Ex: The ID of the student or Instructor, Roll number of the student, phone number, email id, etc.
- On the contrary those attributes that can be split into components are called a **composite attribute**
 - Ex: The address can be further split into house number, street number, city, state, country, and pin code,
 - The name can also be split into first name middle name, and last name.



Composite Attributes

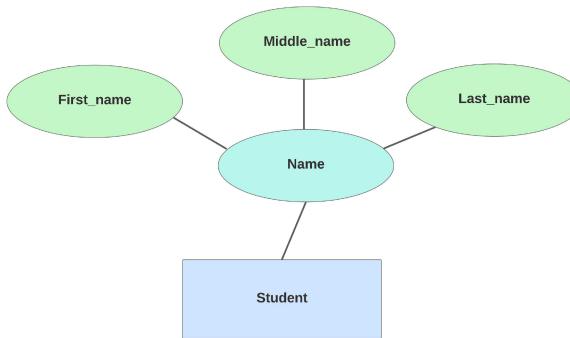
- Let us consider the university database. Like any other organization, the management would keep a record of all the addresses of the student, instructors and the staff
- Now address would contain the entire street, city, state, and postal_code information.
- Let us say the address of the student is :
 - "123 Main Street, Yeshwanthpur, Bangalore, 560063"
- Now this is how all the information is stored in one attribute.
 - Let us say we would want to group students through the postal code for allotting the busses. How do we do it?
 - Currently the postal code is not an attribute alone or stored specifically as an attribute so we cannot get the postal code alone as it is a part of the address
- Thus just having the address stored as one attribute is not always appropriate, especially when we would want to retrieve students on a part of the address.

Composite Attributes

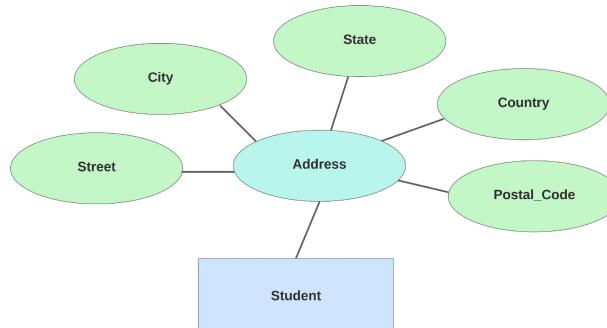
- For solving this issue, we try to store these types of attributes by dividing the attribute into subparts.
- These attributes which can be divided into subparts are called '**Composite attributes**'.
- Using composite attributes in a design schema is a good choice if a user will wish to refer to an entire attribute on some occasions, and to only a component of the attribute on other occasions.

Representation of the Composite attribute

- Let us consider the examples discussed above, and how are they represented it in the E-R diagram
 - Name could be subdivided into First Name, Middle Name, Last Name

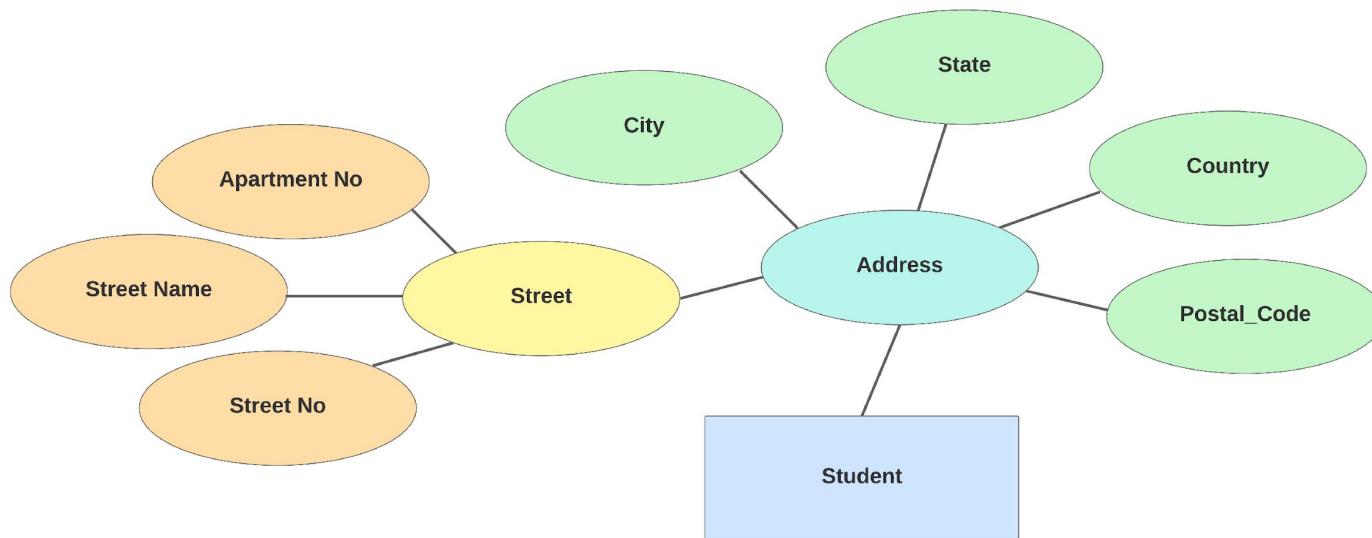


- The address can be further split into street, city, state, country, and pin code



Representation of the Composite attribute

- Composite attributes may appear as a hierarchy.
- In the composite attribute address, its component attribute street can be further divided into street number, street name, and apartment number.

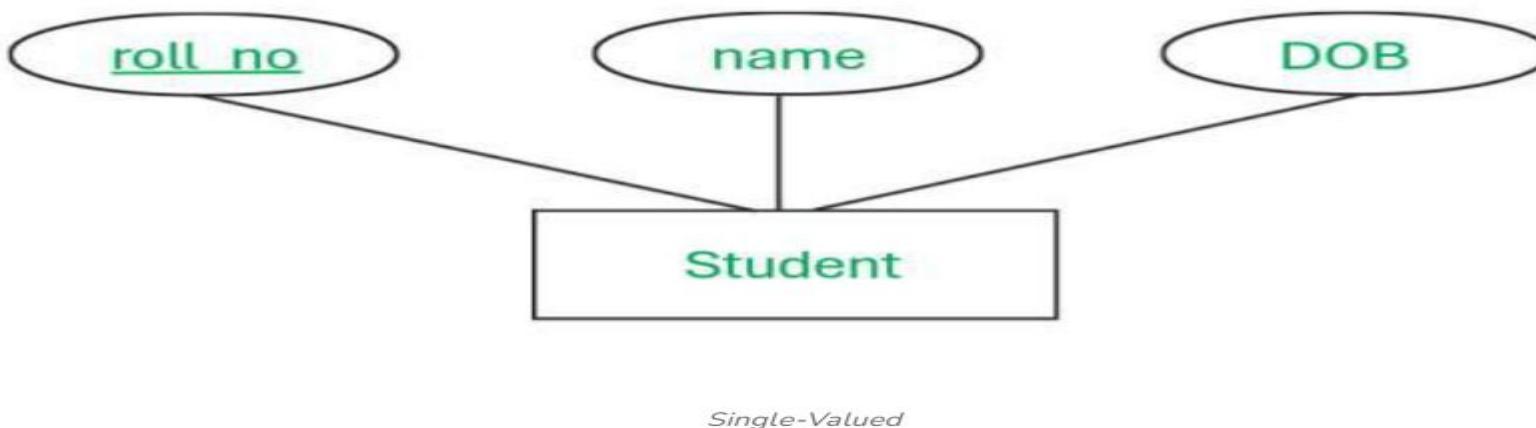


Multivalued Attributes

- Suppose the university has decided to add the phone number to the instructor relation.
- An instructor may have Zero, One, or Several phone numbers, and different instructors may have different number of phone numbers.
- Same is the case with the instructor's mail_id.
- Till now all the attributes we have studied were having only a single value and not multiple values. So how do we deal with these type of situation ?
- These types of attributes which can have multiple values are called as **multivalued Attributes**
- Examples for the same are :
 - Instructor Phone numbers
 - Instructors mail_id

Single-valued and Multivalued Attributes

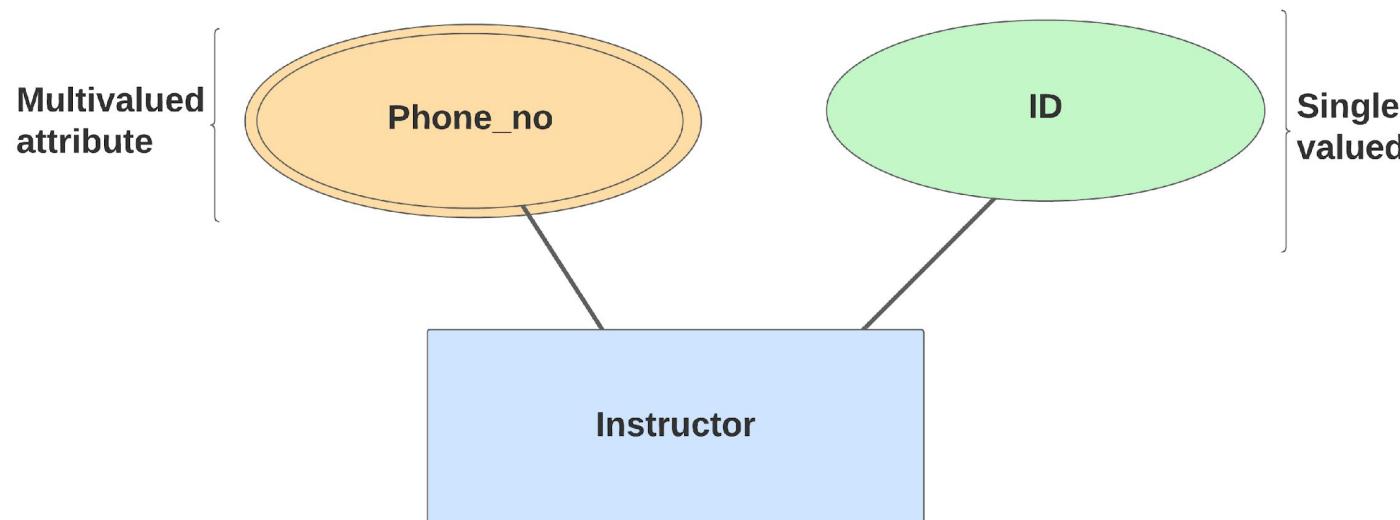
- The attribute which takes up only a single value for each entity instance is a **single-valued attribute**.
 - Example: The name of the instructor, age of the student, etc.



- The attribute which takes up more than a single value for each entity instance is a **multi-valued attribute**.
 - Example: Phone number of the instructor, Mail id, etc.

Representation of Multivalued Attributes

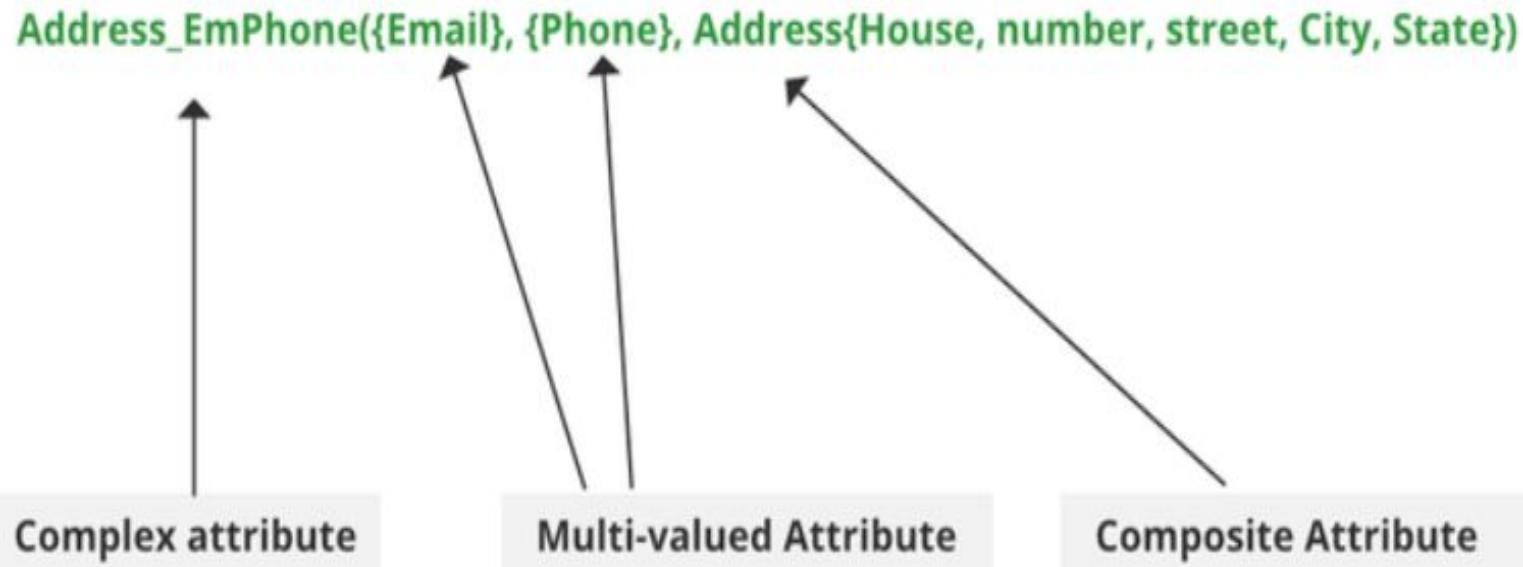
- The multivalued attribute is represented in the E-R diagram as shown below:
- The multivalued attribute is represented by a double ellipse.
- Considering the instructor's Phone_no as a multivalued attribute and ID as an single-valued attribute:



Complex Attributes

- Those attributes, which can be formed by the nesting of composite and multi-valued attributes, are called “**Complex Attributes**”.
- These attributes are rarely used in DBMS.
- Let us take an example of the same.
 - Let us consider that the student relation contains an attribute called PreviousDegrees that stores all the previous education qualifications of the students
 - PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}
 - Multiple PreviousDegrees values can exist. Each has four subcomponent attributes: College, Year, Degree, Field
 - Therefore it is a complex attribute

Given ,an attribute Address_EmPhone consisting of email, phone and Address. Can you identify which are the different attribute types that make up this attribute.



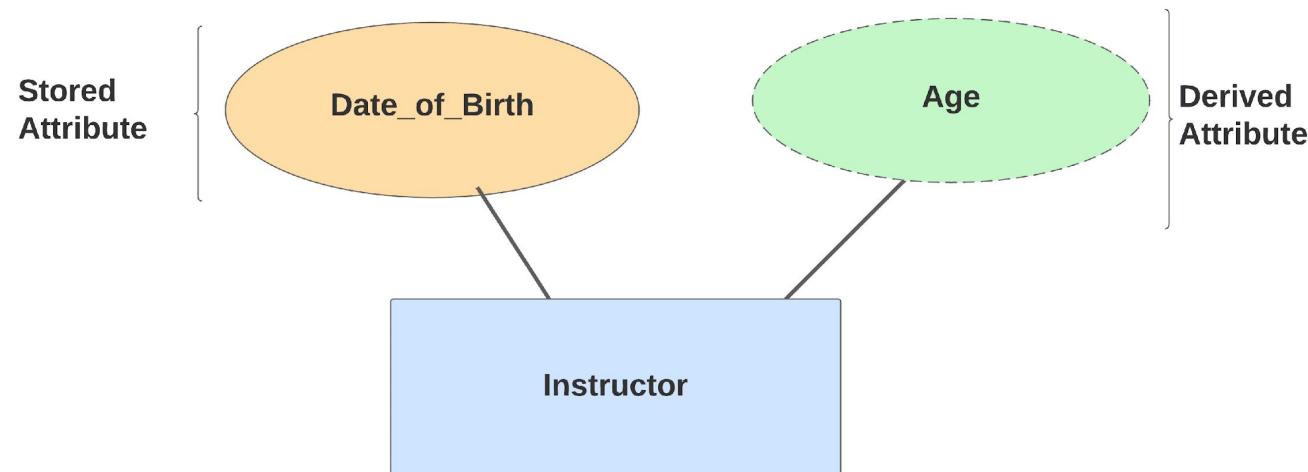
Derived Attributes

- Consider the following schema for the Instructor relation
 - **Instructor** (ID, Name, Department, Date_of_Birth, Age)
- Can you identify any problem in the above schema?
- If you have noticed, we know that the age can be calculated from the Date of birth, i.e. we can take the system's current date and the Date of birth of the instructor and we can find the age.
- So there is no need to store this information in the actual database and can be calculated.

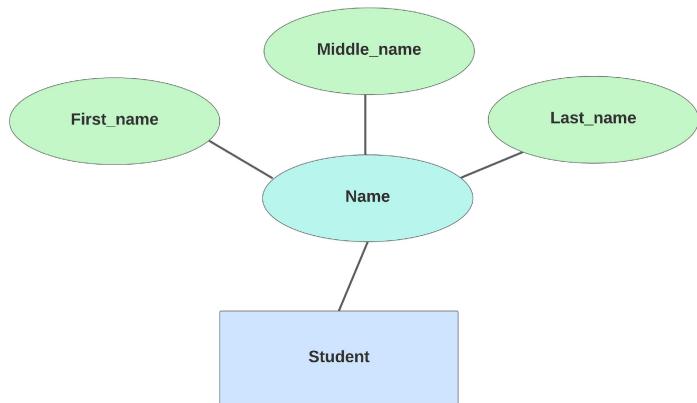
Stored and Derived Attributes

- So as shown above, the value of the age attribute can be derived from the date of Birth.
- Therefore the Age attribute is called a Derived Attribute
- And the Date_of_Birth attribute is called as a stored attribute.
- The **stored attribute** are those attributes that are actually stored in the database.
 - Example: Date_of_Birth is a stored attribute
- An attribute that can be derived from other attributes is **derived attributes**.
 - Example: the Age attribute of the instructor is a derived attribute

- Considering the above example of the instructor relation, Date_of_Birth is a stored attribute and Age is an derived attribute
- In the below E-R Diagram, Age (derived attribute) is represented by a dashed oval.
- The E-R representation would be:



- Let us consider the diagram representation shown:
- What if the student doesn't have a middle name?



- In such situations the value of the middle name would be set to NULL

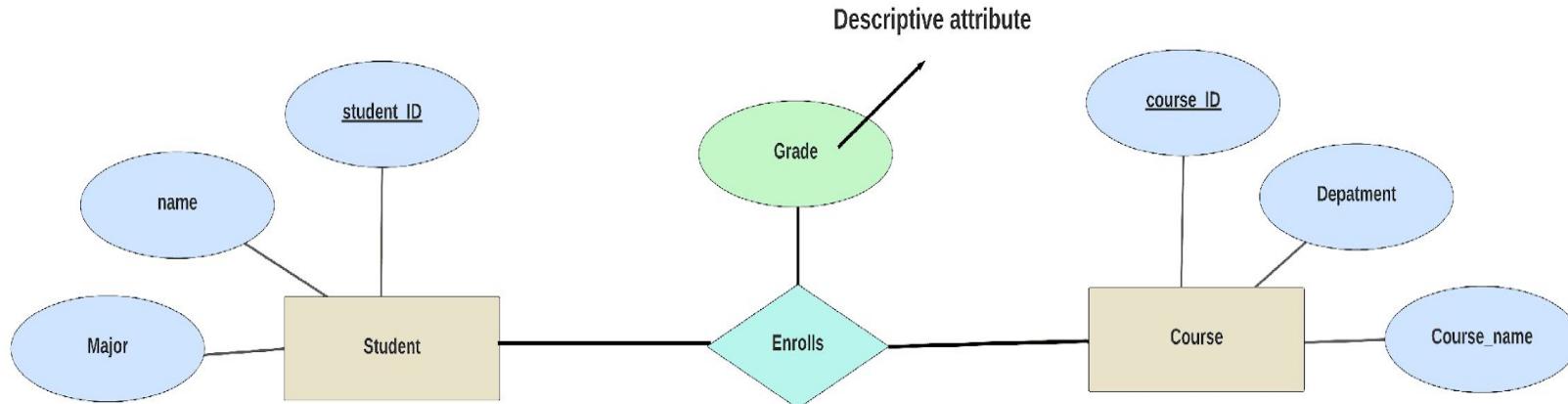
- An attribute takes a null value when an entity does not have a value for it.
- So what does it mean if an attribute is NULL?
 - The Null value may indicate “**not applicable**”—that is, the value does not exist for the entity.
 - Null can also designate that an attribute value is **unknown**. An unknown value may be either:
 - **missing** (the value does exist, but we do not have that information)
 - **not known** (we do not know whether or not the value actually exists).
- A null value for the apartment number attribute could mean:
 - Address does not include an apartment number (**not applicable**),
 - That an apartment number exists but we do not know what it is (**missing**),
 - We do not know whether or not an apartment number is part of the instructor’s address (**unknown**).

Descriptive Attributes

- Let us now consider the scenario that every student enrolls for certain courses.
- Now as we know along with what each student enrolls for which course, the university must also keep track of the grades that the student would score for that particular course.
- Now where do we place this grade attribute? In the Student entity or the Courses entity?
- Actually if we observe this attribute is associated with the relation 'enroll' i.e. only if the student enrolls for the particular course, he can get a grade. Such type of attributes are called **Descriptive attributes**
- So it wouldn't be correct to place it in any of the entities
- Then where do we place them?

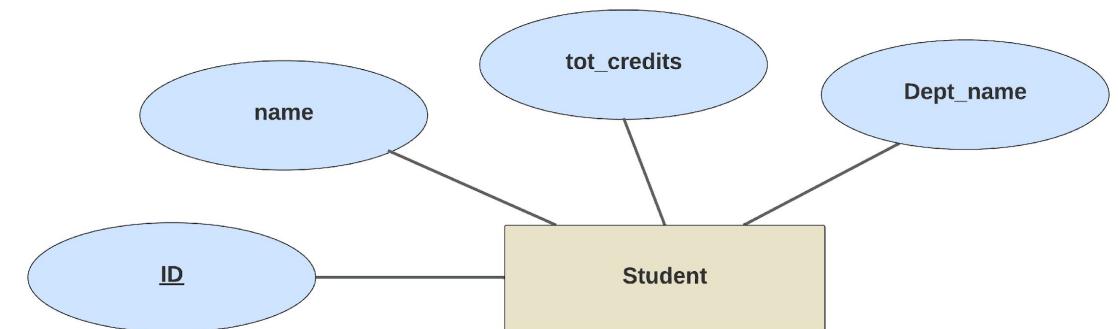
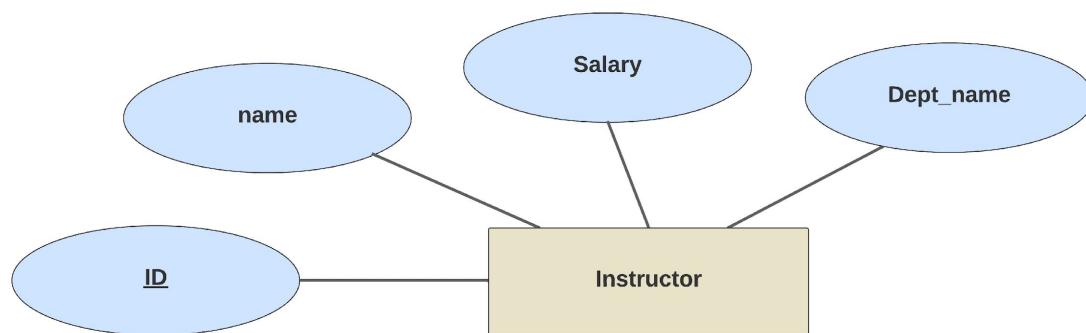
Descriptive Attributes

- A relationship may also have attributes called descriptive attributes.
- An attribute of a relationship set is represented in an E-R diagram by an oval connected to the relationship's diamond
- The below diagram shows the discussed example in the form of an ER diagram



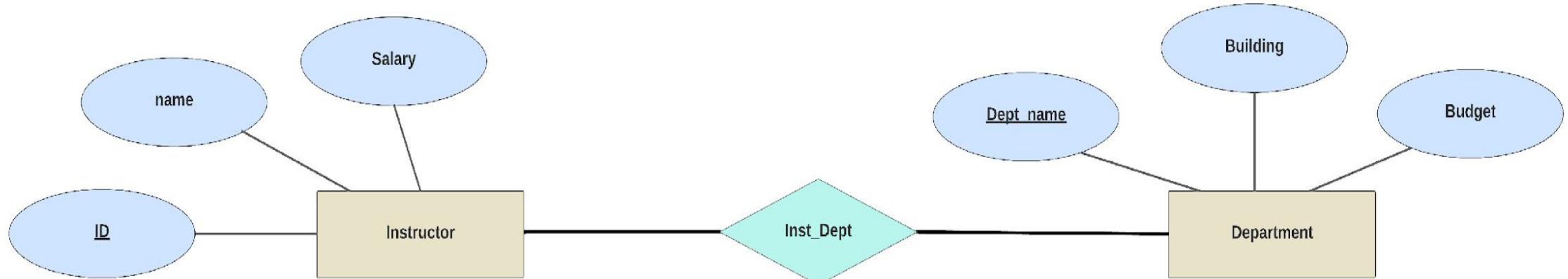
Redundant attributes

- In the university organization, consider the following entity sets:
 - Student
 - Instructor
- Attributes are selected to capture values in the database;
 - Instructors: (ID, name, dept_name, salary)
 - Students: (ID, name, dept_name, tot_cred)
- Additional attributes like phone number, office number are optional based on the designer's understanding.



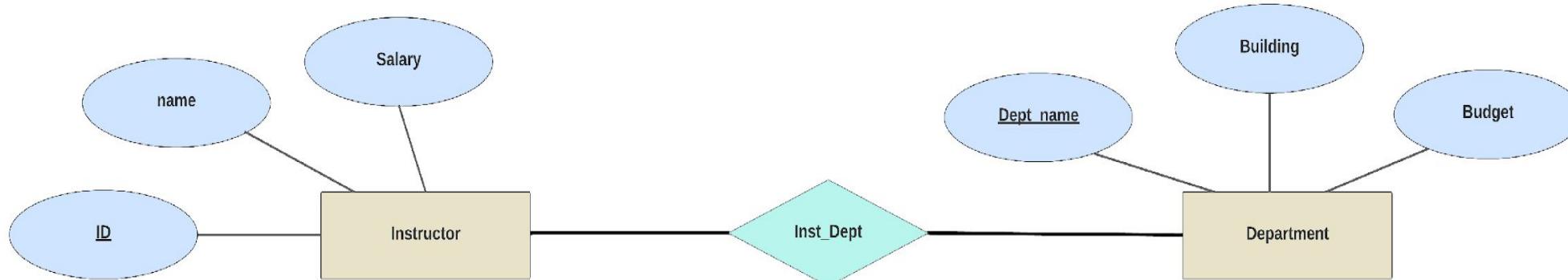
Redundant attributes

- Consider the entity department in the university database:
 - Department (dept_name, building, budget)
- Relationship sets connect entities together, like instructor and department.
- Redundancy can occur due to relationship sets, leading to attribute removal for clarity.
- The dept_name attribute, redundant in both instructor and department, is removed.
- Instead we represent this using a relationship set (inst_dept) between instructor and department.



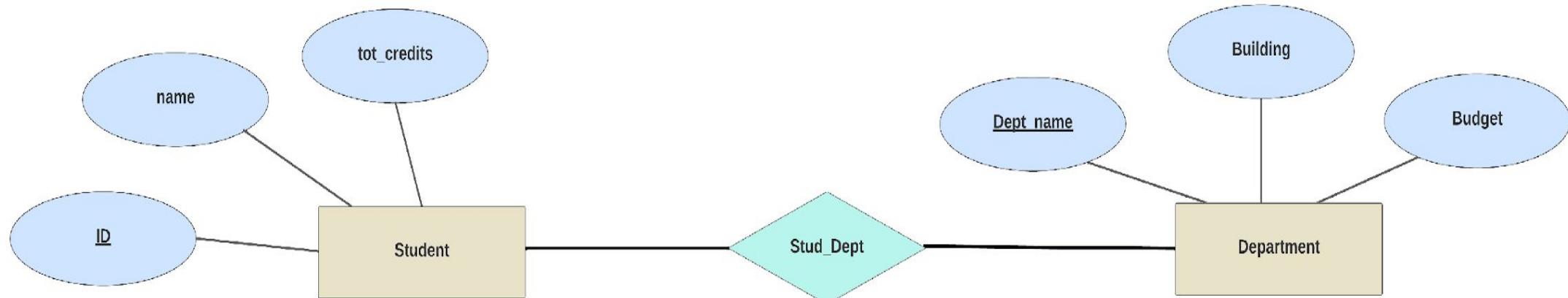
Redundant attributes

- Redundant dept name attribute in instructor is removed since it's a primary key in the department table.
- In a relational schema, dept_name may be reintroduced in the instructor's relation if each instructor has one department.
- If instructors have multiple departments, a separate relation (inst_dept) records these relationships.
- Treating instructor-department connection as a relationship prevents assumptions of single department association.



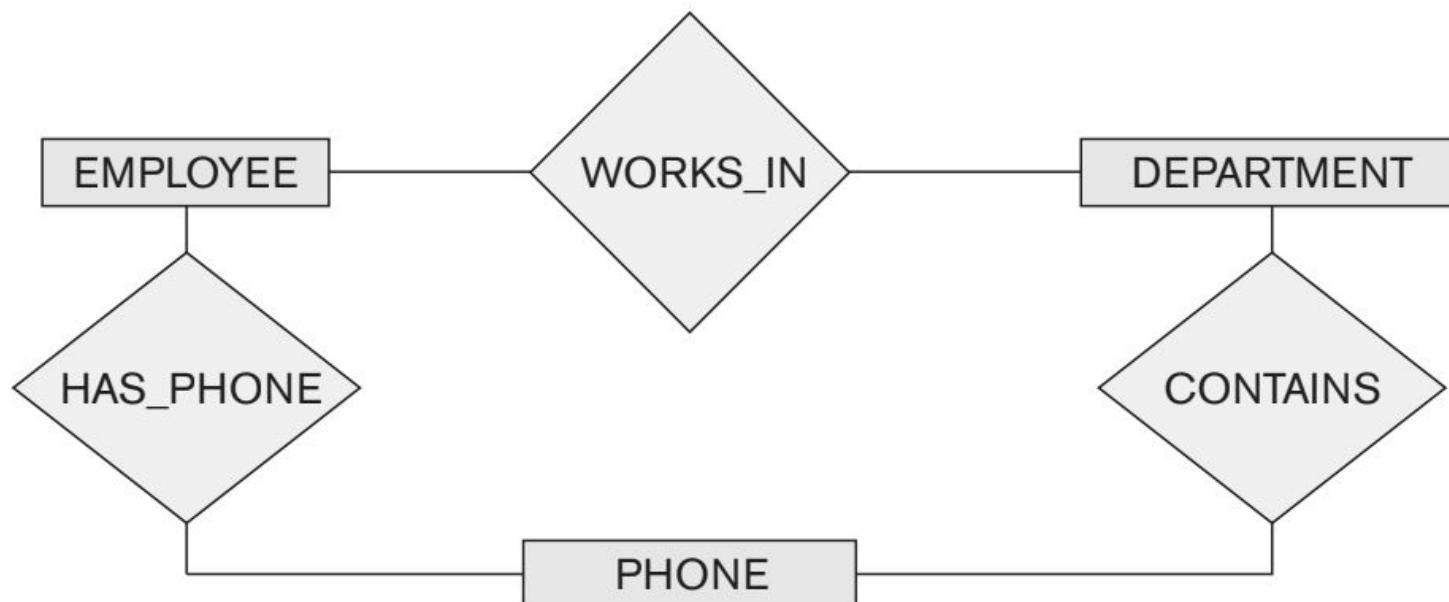
Redundant attributes

- Similarly, student entity set is related to the department entity set through a relationship set (student dept).
- No need for dept name attribute in student entity set.



Redundant relationship - Question

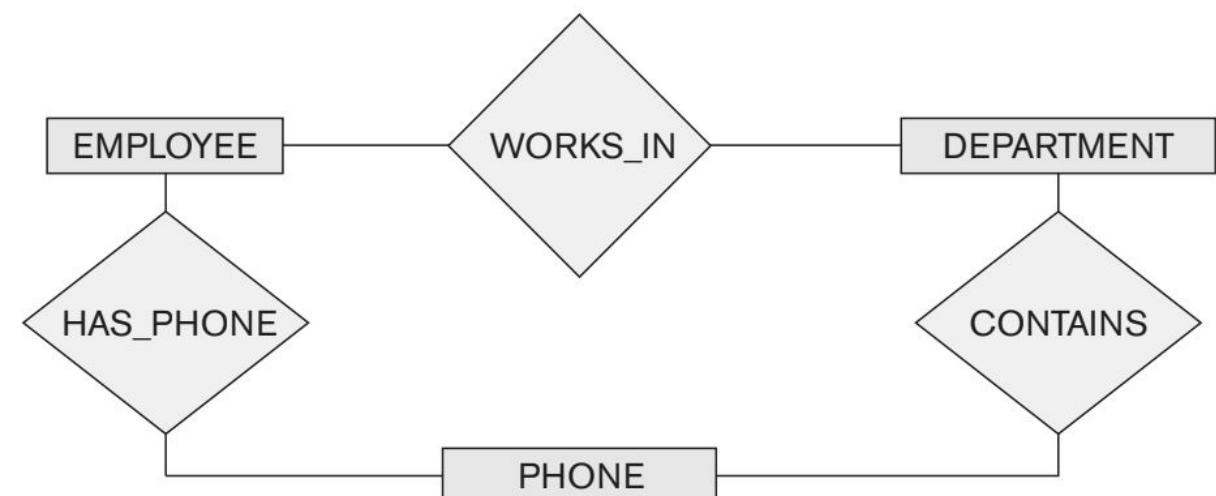
Assume that an employee may work in up to two departments or may not be assigned to any department. Assume that each department must have one and may have up to three phone numbers. Under what conditions would the relationship HAS_PHONE be redundant in this example?



Redundant relationship - Question

Assume that an employee may work in up to two departments or may not be assigned to any department. Assume that each department must have one and may have up to three phone numbers. Under what conditions would the relationship HAS_PHONE be redundant in this example?

- The HAS_PHONE relationship is redundant if phone numbers are exclusively associated with departments, not individual employees.
- If employees only use department phone numbers, their association with these numbers can be inferred through the WORKS_IN relationship, eliminating the need for a separate HAS_PHONE relationship.



Redundant attributes

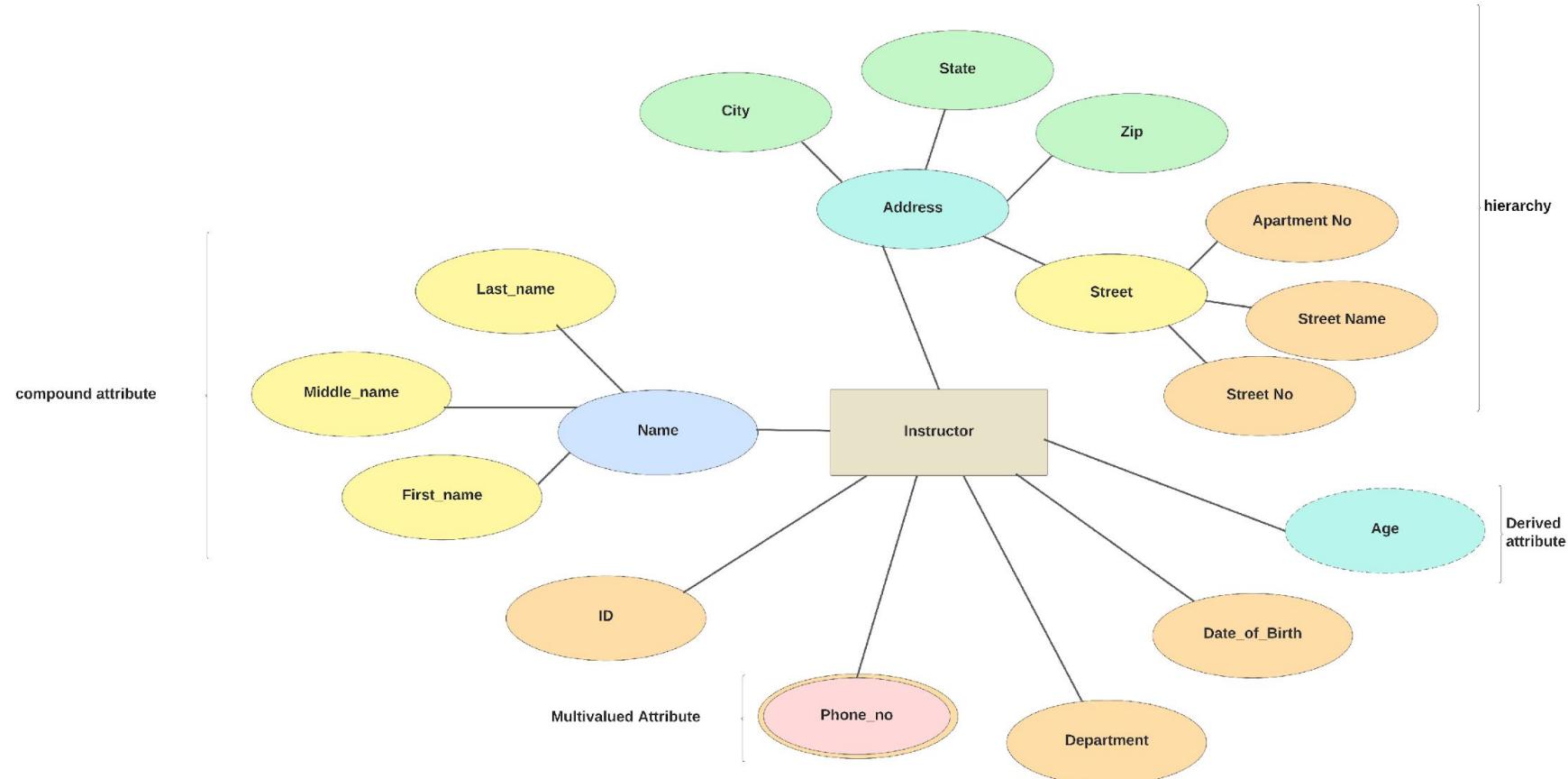
- Consider Course offerings (sections) and associated time slots.
- Each time slot has a unique time slot id and weekly meetings (day, start time, end time). Weekly meeting times are modeled as a multivalued composite attribute.
- Entity sets: section and time slot.
 - Section: (course id, sec id, semester, year, building, room number, time slot id)
 - Time slot: (time slot id, {(day, start time, end time)}).
- Entities related through relationship set sec time slot. Multivalued composite attribute
- Issue of redundancy with time slot id in both entity sets.
- Time slot id is the primary key for the time slot entity.
- Therefore, Remove the time_slot_id attribute from the section entity.
- A good entity relationship model would not contain any redundant attributes
- Redundancy removal enhances data clarity and avoids duplication

Example

- Let us consider the Instructor relation of the university database.
- Every instructor would have a unique ID, along with this the instructor would also have the name, address, phone number, Date of Birth, age, and department details to be there in the database
- Given instructor could be a part of only one department. However, each instructor could have multiple phone numbers.
- The address is subdivided into street, city, state, and zip. The street would be again divided into street_no, street_name, and apt_number.
- The name attribute would be subdivided into first name, middle name, and last name.
- Could you draw an E-R diagram for containing the entity Instructor along with all the attributes?

Database Management Systems

E-R Diagram





PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Mapping Cardinalities

S Nagasundari

Department of Computer Science Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering

- Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set
 - Example: Each instructor can be a part of only one department
- Two main types of **binary relationship constraints**:
 - **Cardinality ratio**
 - **Participation Constraints and Existence Dependencies.**
- The cardinality ratio and participation constraints together, are known as the **structural constraints** of a relationship type

Cardinality Ratio

- Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.
- Mapping cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets.
- A binary relationship set R between entity sets A and B, the possible mapping cardinality are:
 - One-to-one
 - One-to-many
 - Many-to-one
 - Many-to-many

- **One-to-one**

- An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

- **One-to-many**

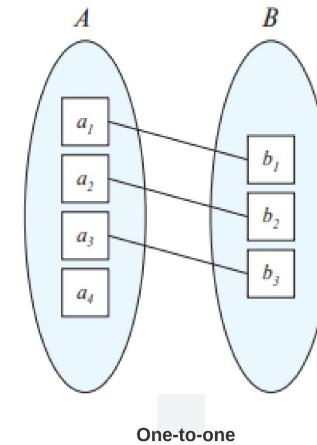
- An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.

- **Many-to-one**

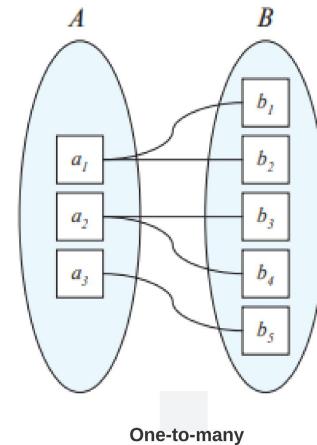
- An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.

- **Many-to-many**

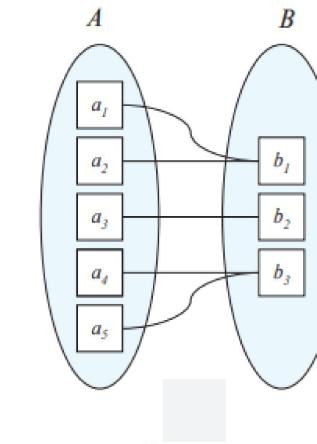
- An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.



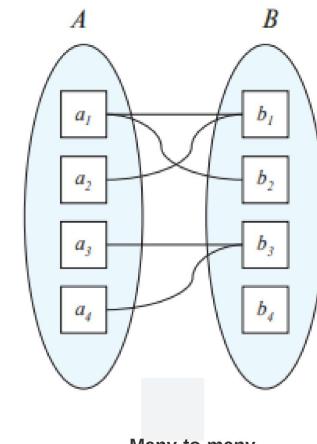
One-to-one



One-to-many



Many-to-one



Many-to-many

- Let us now consider the advisor relationship set.
- A student can be advised by several instructors - this relationship set can be classified as many-to-many.
- In contrast, if a particular university imposes a constraint that a student can be advised by only one instructor, an instructor can advise several students, then the relationship set from instructor to student must be one-to-many.



one-to-many



many-to-many

Cardinality Ratio

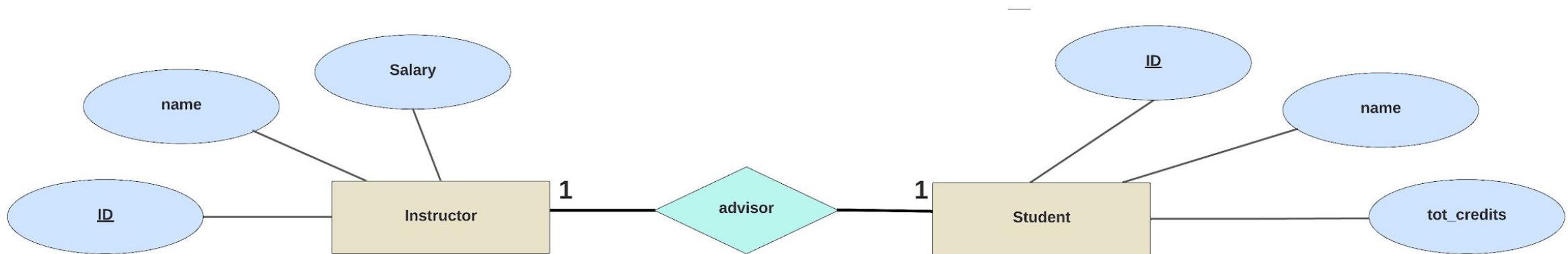
In the E-R diagram notation, we indicate cardinality constraints on a relationship by drawing:

- An **undirected line (—)** between the relationship set and the entity set in question.

Let us try and understand how to use the above mapping technique with the help of university database.

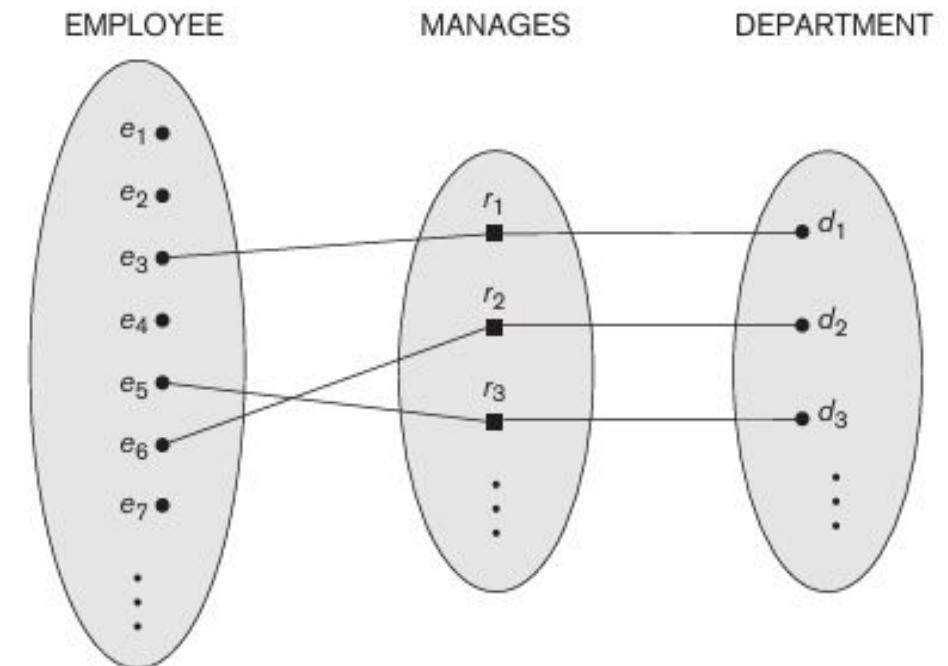
One-to-one (1:1)

- We draw a line from the relationship set to both entity sets.
- For example, the lines to instructor and student indicate that an instructor may advise at most one student, and a student may have at most one advisor.



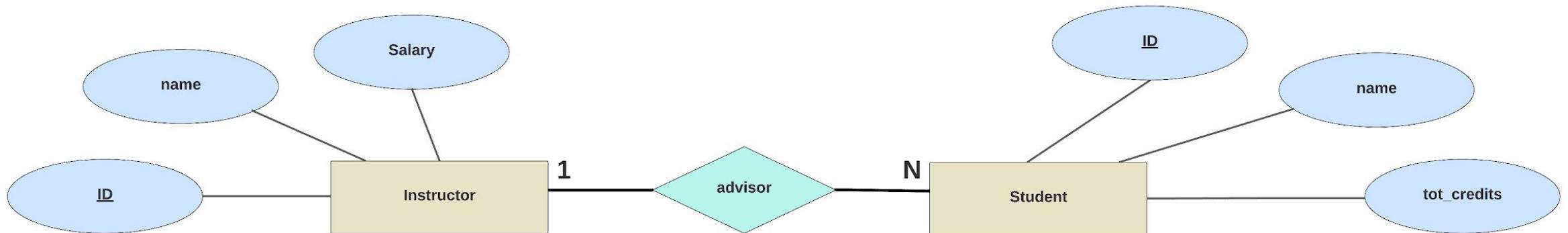
One-to-one (1:1)

- An example of a 1:1 binary relationship is MANAGES as shown in the figure, which relates a department entity to the employee who manages that department.
- This represents that at any point in time an employee can manage at most one department and a department can have at most one manager.



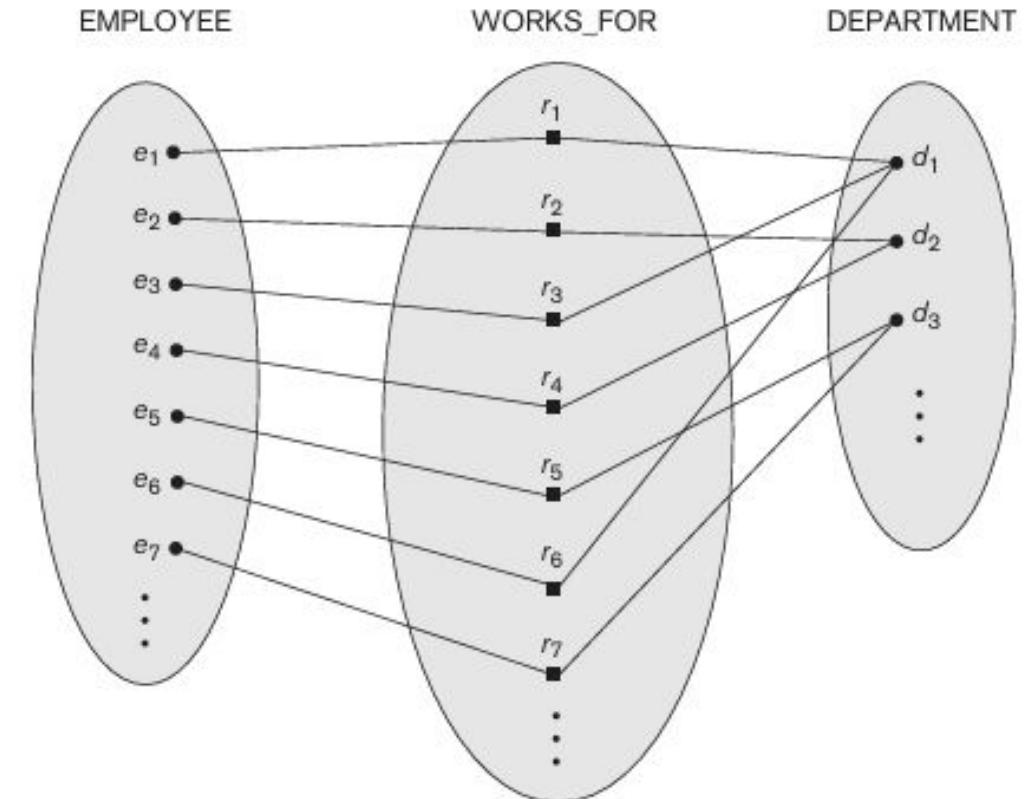
One-to-many (1:N)

- This indicates that an instructor may advise many students, but a student may have at most one advisor.



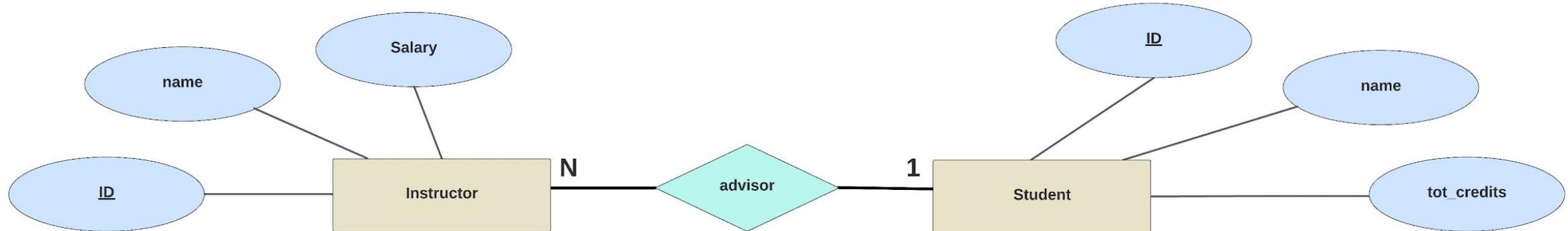
One-to-many (1:N)

- For example, In the WORKS_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N.
- Meaning that each department can be related to any number of employees (N) but an employee can be related to (work for) at most one department.
- This means that for this particular relationship type WORKS_FOR, a particular department entity can be related to any number of employees.



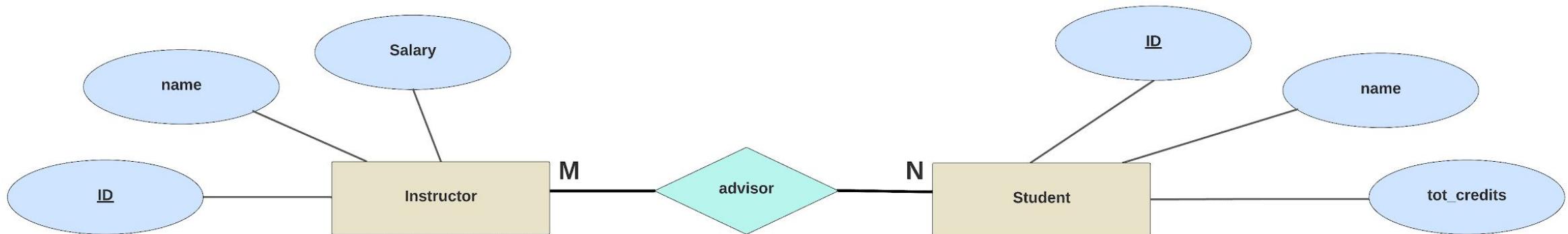
Many-to-one (N:1)

- The converse of the before type
- This indicates that an instructor may advise at most one student, but a student may have many advisors.



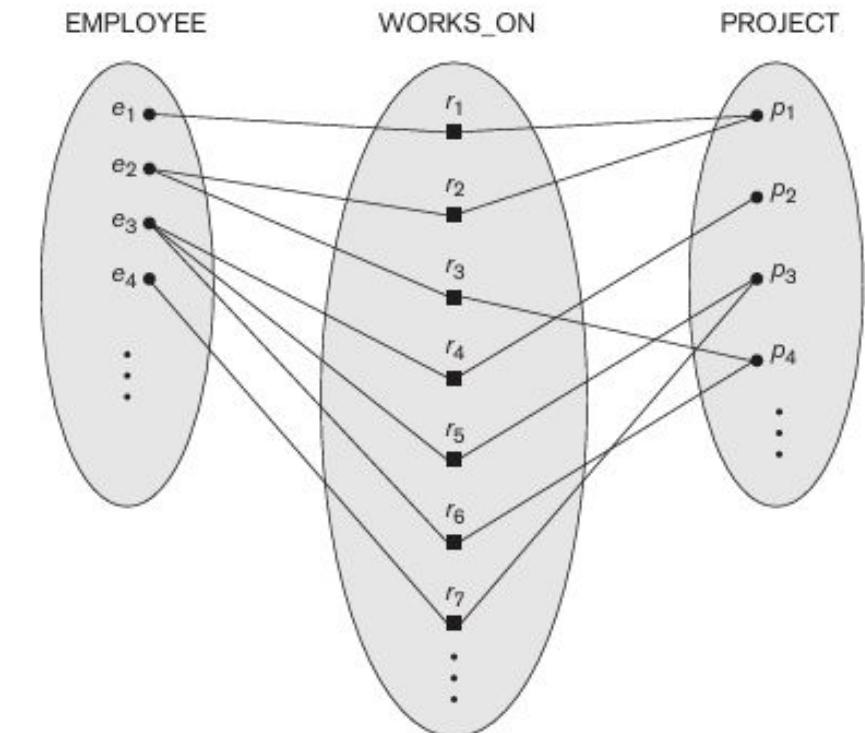
Many-to-many (M:N)

- We draw an undirected line from the relationship set to both entity sets.
- Thus, there are undirected lines from the relationship set advisor to both entity sets instructor and student.
- This indicates that an instructor may advise many students and a student may have many advisors.



Many-to-many (M:N)

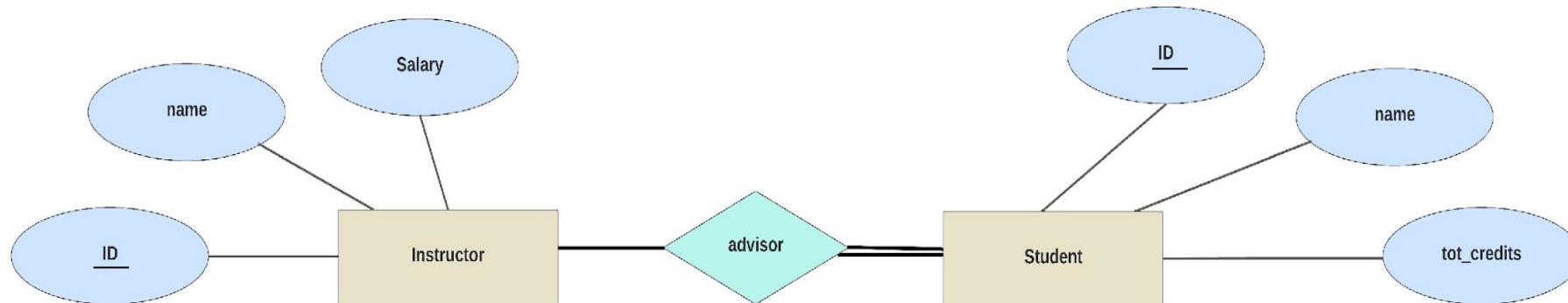
- The relationship type WORKS_ON is of cardinality ratio M:N
- The rule is that an employee can work on several projects and a project can have several employees which is represented by the diagram on the right.



Participation Constraints and Existence Dependencies

- There are two types of participation :
 - **Total participation**
 - **Partial participation**
- The participation of an entity set E in a relationship set R is said to be **total** if every entity in E must participate in at least one relationship in R.
 - Considering the university database, a university may require every student to have at least one advisor Therefore, the participation of student in the relationship set advisor is **total**.
- If it is possible that some entities in E do not participate in relationships in R, the participation of entity set E in relationship R is said to be **partial**.
 - An instructor need not advise any students therefore, The participation of instructor in the advisor relationship set is therefore **partial**

- We indicate the total participation of an entity in a relationship set using double lines.
- The example of the advisor relationship set described in the previous slide would look as shown
- where the double line indicates that a student must have an advisor.



Cardinality ratio

- Let us consider the entities:
 - Student
 - Attributes: Student_ID, Name, Major
 - Course
 - Attributes: Course_ID, Course_Name, Department
- The institution has made a rule that given every student has to be enrolled for a minimum of 2 courses and a maximum of 5 courses
- How would we represent this in the ER diagram?

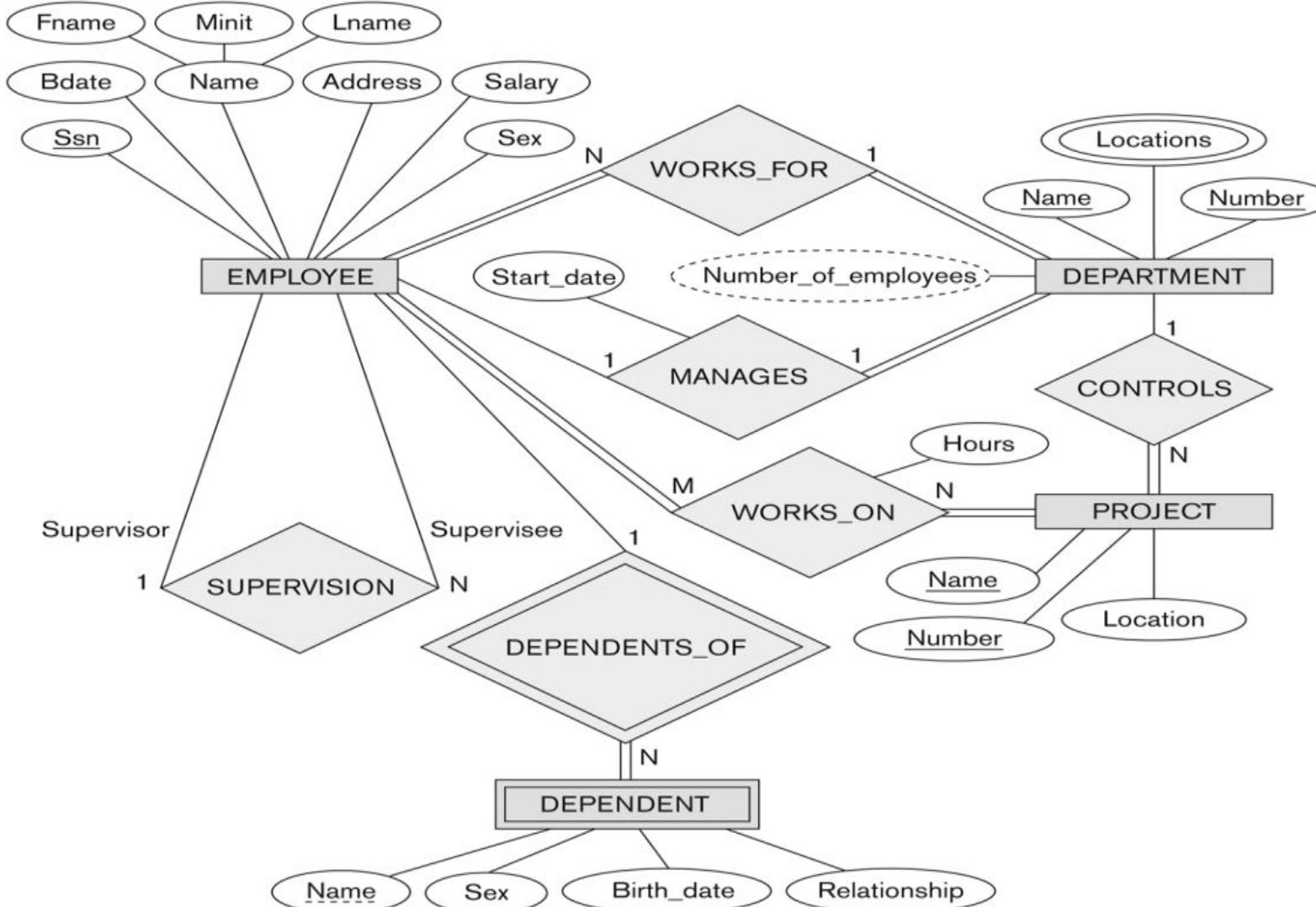
Min-Max notation

- E-R diagrams also provide a way to indicate more complex constraints on the number of times each entity participates in relationships in a relationship set.
- A line may have an associated minimum and maximum cardinality, shown in the form **(l,h)**
 - **l** is the **minimum cardinality**
 - **h** is the **maximum cardinality**.
- A minimum value = 1 \square total participation of the entity set in the relationship set;
- A maximum value = 1 \square that the entity participates in at most one relationship,
- A maximum value = **N** indicates **no limit**.

Min-Max [Entity 1]	Chen-Notation	Min-Max [Entity 2]
(0,1)	1:1	(0,1)
(0,N)	1:n	(0,1)
(0,N)	1:n + total part.	(1,1)
(0,N)	n:m	(0,N)
(1,1)	total part. + 1:1	(0,1)
(1,N)	total part. + 1:n	(0,1)
(1,1)	total part. + 1:1 + total part.	(1,1)
(1,N)	total part. + 1:n + total part.	(1,1)
(1,N)	total part. + n:m	(0,N)
(1,N)	total part. + n:m + total part.	(1,N)

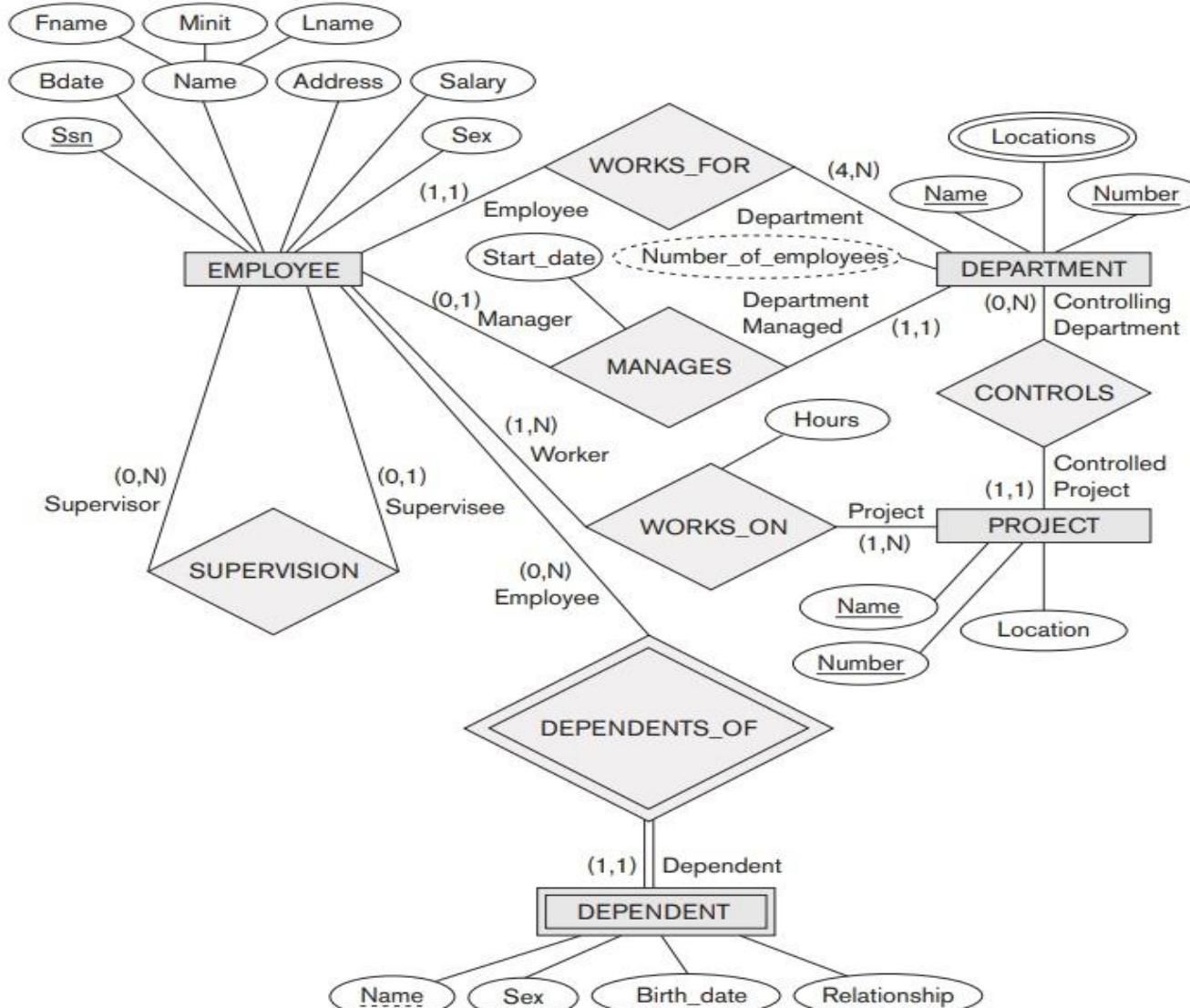
Database Management Systems

Example of Chen Notation



Database Management Systems

(Min,Max) Notation





PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Primary Keys In Relationship and

Entity Sets

S Nagasundari

Department of Computer Science Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering

Primary Key In Entity Sets

- Individual entities are **conceptually distinct**.
- From a database perspective, differences among entities are expressed through attributes.
- Attribute values must uniquely identify each entity.
- **No two entities in an entity set can share the same values for all attributes.**
- Key for an entity is a set of attributes that distinguish entities.
- Concepts of superkey, candidate key, and primary key apply to entity sets.

Primary Key In Relationship Sets

Keys also help uniquely identify relationships. Thus differentiates relationships from each other.

- So we will now look at how can we identify/distinguish relationships
- Let R be a relationship set involving entity sets E1, E2, ..., En.
- Let **primary-key(Ei)** denote the set of attributes that forms the primary key for entity set Ei .
- Assume for now that the attribute names of all primary keys are unique.
- The composition of the primary key for a relationship set depends on the set of attributes associated with the relationship set R.
 - Case 1: Relationship R has **no attributes**
 - Case 2: If relation R has **{a₁, a₂, ..., a_m}** attributes associated with it

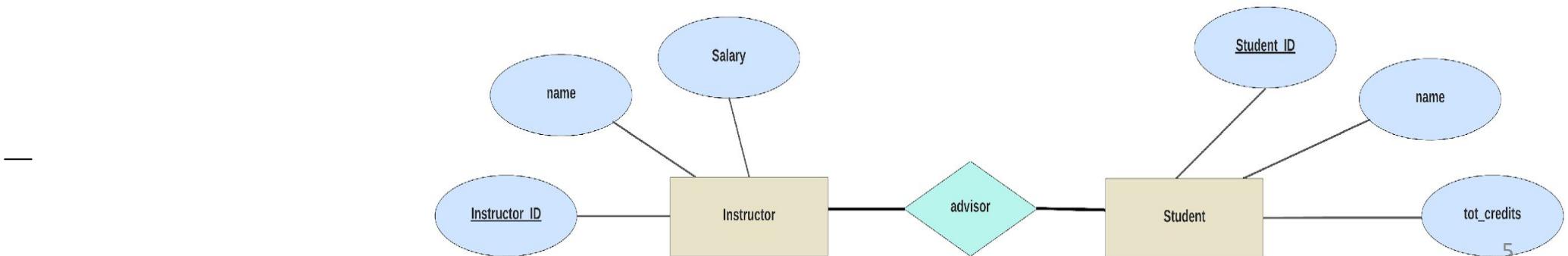
Case 1:

- If the relationship set R has **no attributes** associated with it,
then the set of attributes:

primary-key(E1) \cup primary-key(E2) $\cup \dots \cup$ primary-key(En)

describes an individual relationship in set R.

- For example consider the relationship of advisor between Instructor and Student as shown
- The set of attributes that would describe the relationship would be
 - {primary-key(Instructor) \cup primary-key(Student)}
 - {Instructor_ID, Student_ID}



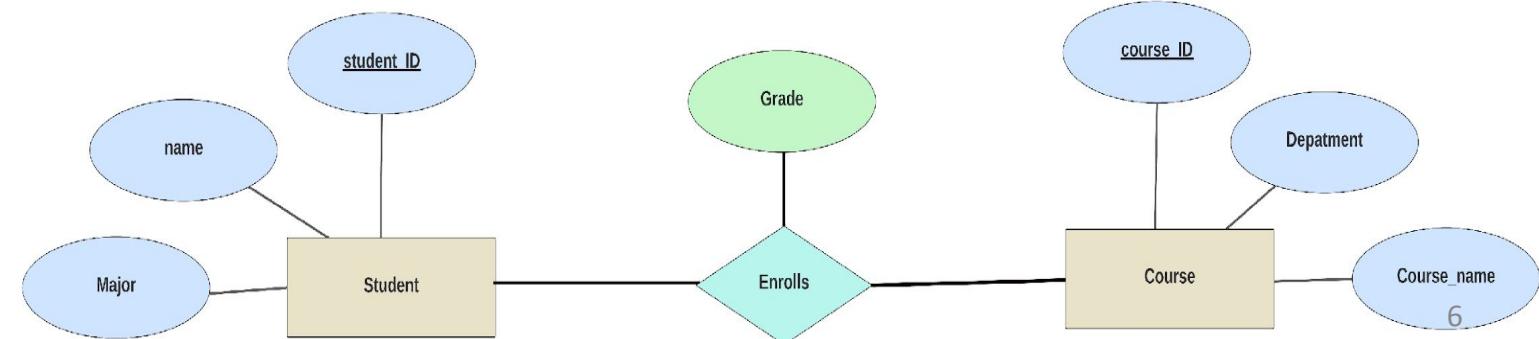
Case 2:

- If the relationship set R has **attributes {a₁, a₂, ..., a_m}** associated with it, then the set of attributes

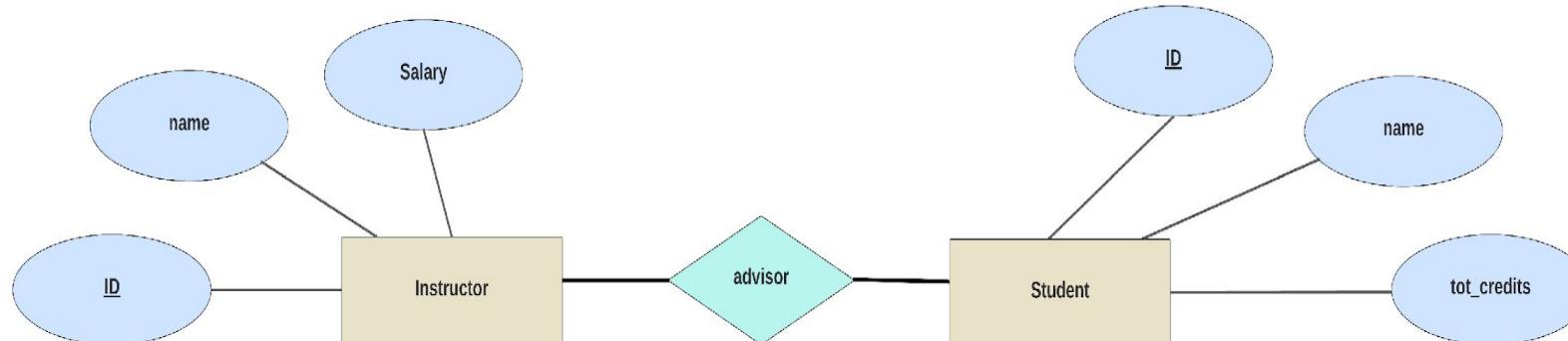
primary-key(E₁) \cup primary-key(E₂) \cup ... \cup primary-key(E_n) \cup {a₁, a₂, ..., a_m}

describes an individual relationship in set R.

- For example let us consider the student enrolls for a course and for each course the student would have an associated grade
- The set of attributes that would describe the relationship would be
 - {primary-key(Student) \cup primary-key(Course) \cup {Grade}}
 - Student_ID, Course_ID, Grade

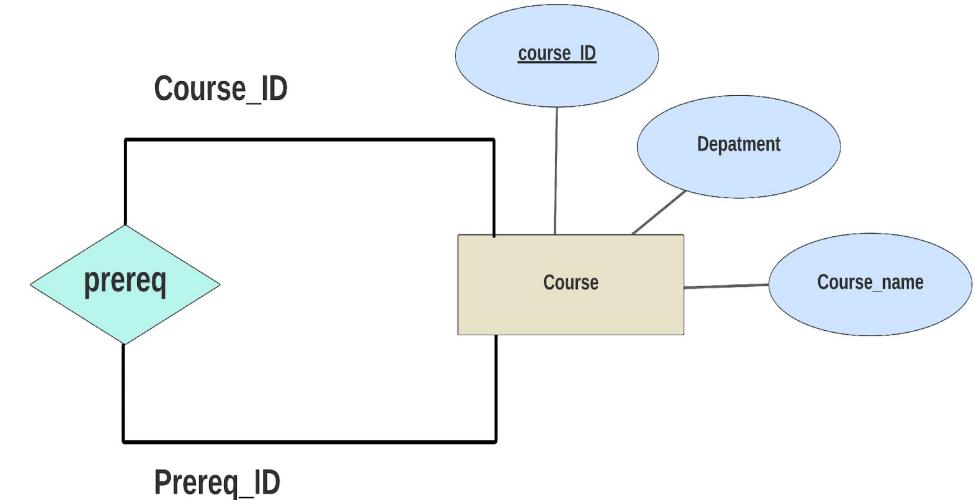


- If the attribute names of primary keys are not unique across entity sets, the attributes **are renamed** to distinguish them;
 - Then the name of the entity set combined with the name of the attribute would form a unique name.
- Considering the below diagram, the attribute name of primary key of both entities are the same (ID)
- So in this case the set of attributes that would describe the relationship would be
 - $\{(Entity\ name + primary_key(Instructor)) \cup (Entity\ name + primary_key(Student))\}$
 - $\{Instructor_ID, Student_ID\}$



Primary Key In Relationship Sets

- If an entity set participates more than once in a relationship set (as in the prereq relationship in the university database), the role name is used instead of the name of the entity set, to form a unique attribute name.
- For example, in the prereq relationship, the Course entity participates in the relation more than once.
- Therefore we use the role name as shown below
- {Course_ID, Prereq_ID}



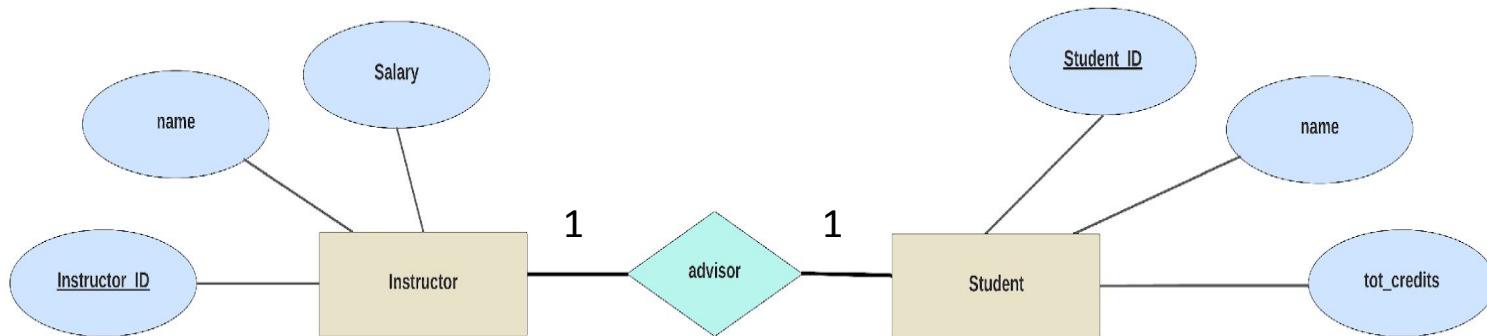
Primary Key In Relationship Sets

- Recall that a relationship set is a set of relationship instances, and each instance is uniquely identified by the entities that participate in it.
- Thus, in both of the cases, {Case 1 or Case 2}
the set of attributes
primary-key(E1) \cup primary-key(E2) \cup ... \cup primary-key(En)
forms a superkey for the relationship set.

- The **choice of the primary key** for a binary relationship set depends on the **mapping cardinality** of the relationship set.
- As we have seen there are 4 different types:
 - One-to-one
 - One-to-many
 - Many-to-one
 - Many-to-many
- Let us find out how do we choose the primary key for these binary relationships

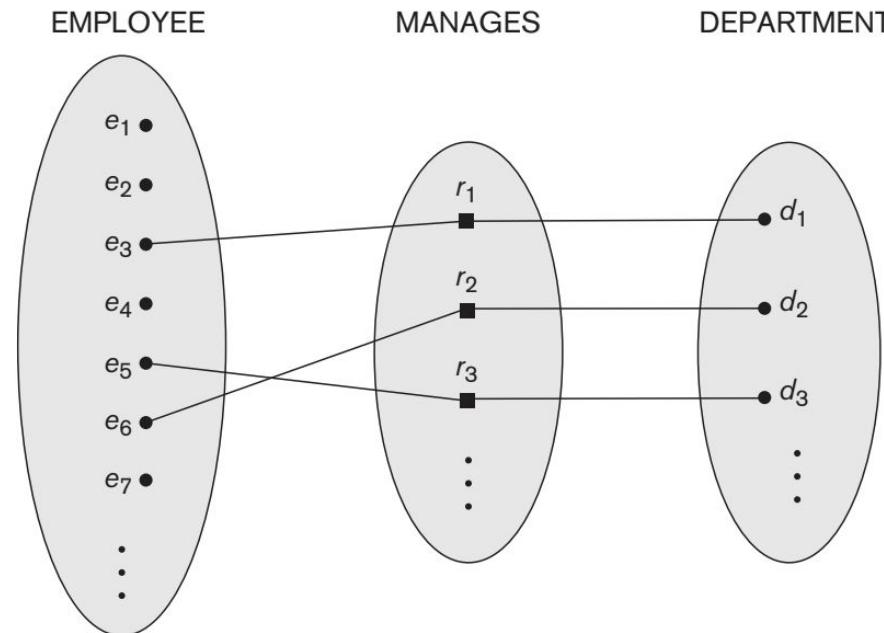
One-to-One

- For this type of relation, the primary key of either one of the participating entity sets forms a minimal superkey, and either one can be chosen as the primary key of the relationship set.
- Let us consider the situation:
- If an instructor can advise only one student, and each student can be advised by only one instructor, that is, if the advisor relationship is one-to-one
 - then the primary key of either student or instructor can be chosen as the primary key for the advisor.



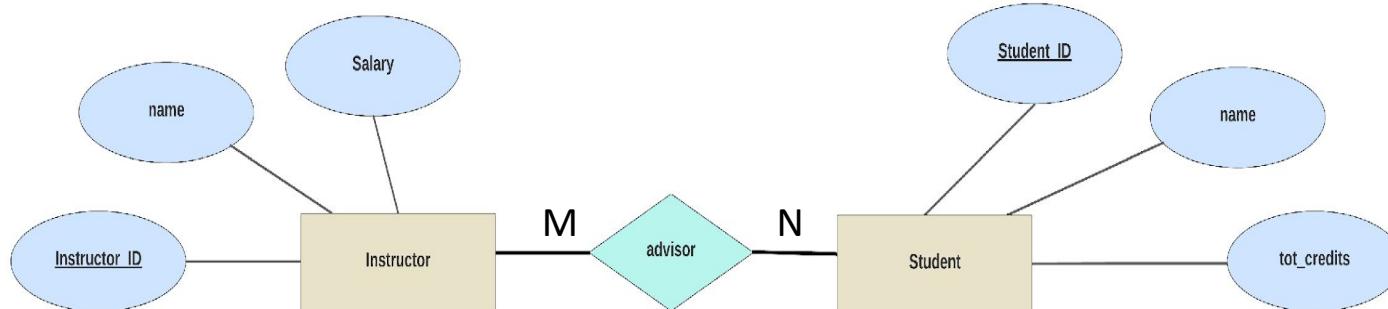
One-to-One

- Let us consider that at any point in time , an employee can manage at most one department and a department can have at most one manager.
- Hence the primary key of either employee or department can be chosen as the primary key for the manages relationship.



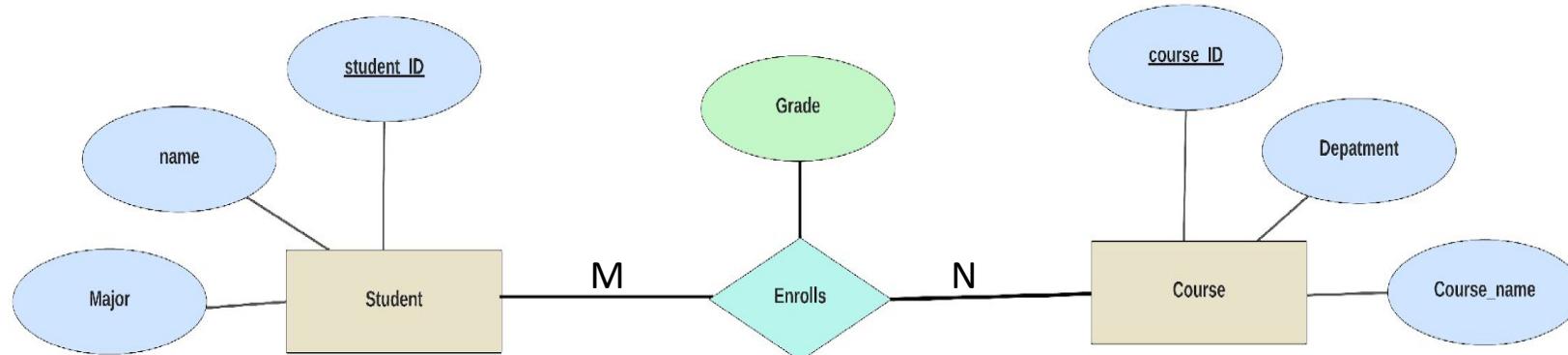
Many-to-Many

- For these relationships, the preceding union of the primary keys is a minimal superkey and is chosen as the primary key.
- As an example we can consider the entity sets instructor and student, and the relationship set advisor. the relationship set is many-to-many.
- The primary key for the advisor relation :
 - $\{\text{primary-key}(\text{Instructor}) \cup \text{primary-key}(\text{Student})\}$
 - $\{\text{Instructor ID, Student ID}\}$



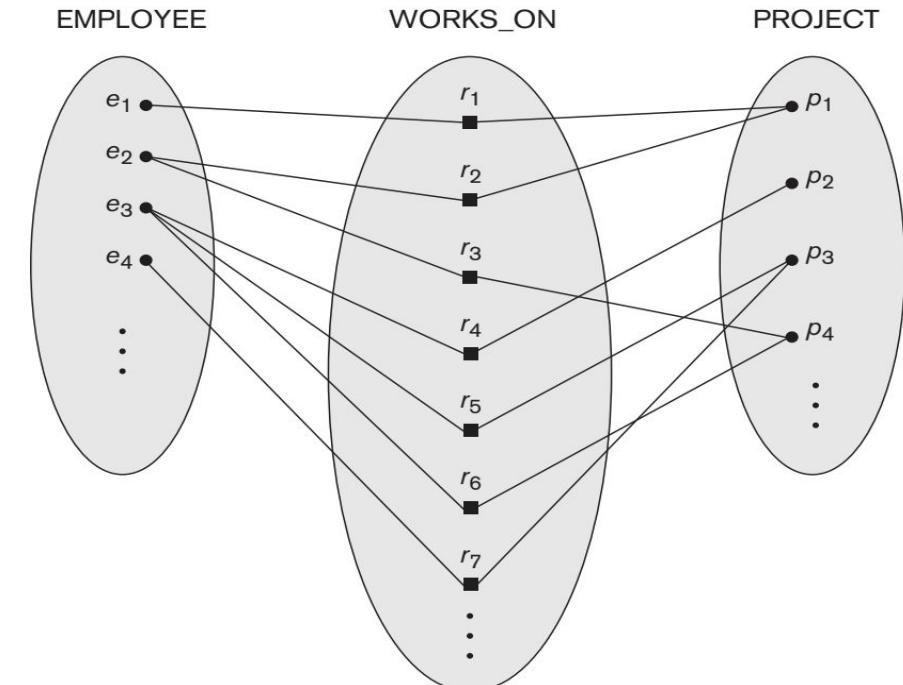
Many-to-Many

- For these relationships, the preceding union of the primary keys is a minimal superkey and is chosen as the primary key.
- As another example we can consider the entity sets student and courses, and the relationship set enrolls. the relationship set is many-to-many.
- The primary key for the advisor relation :
 - {primary-key(Student) \cup primary-key(Course)}
 - {Student_ID, Course_ID}
 - The advisory relation : {Student_ID, Course_ID, Grade}
 - Note that grade is not a part of the primary key



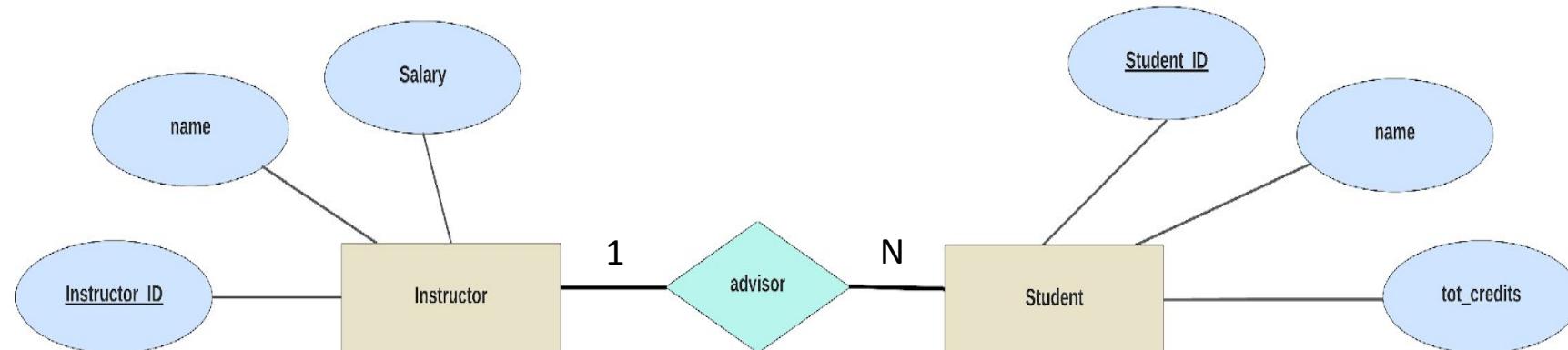
Many-to-Many

- For these relationships, the preceding union of the primary keys is a minimal superkey and is chosen as the primary key.
- As an example we can consider the entity sets Employee and Project, and the relationship works_on.
- The primary key for the works_on relation :
 - $\{\text{primary-key}(\text{Employee}) \cup \text{primary-key}(\text{Project})\}$



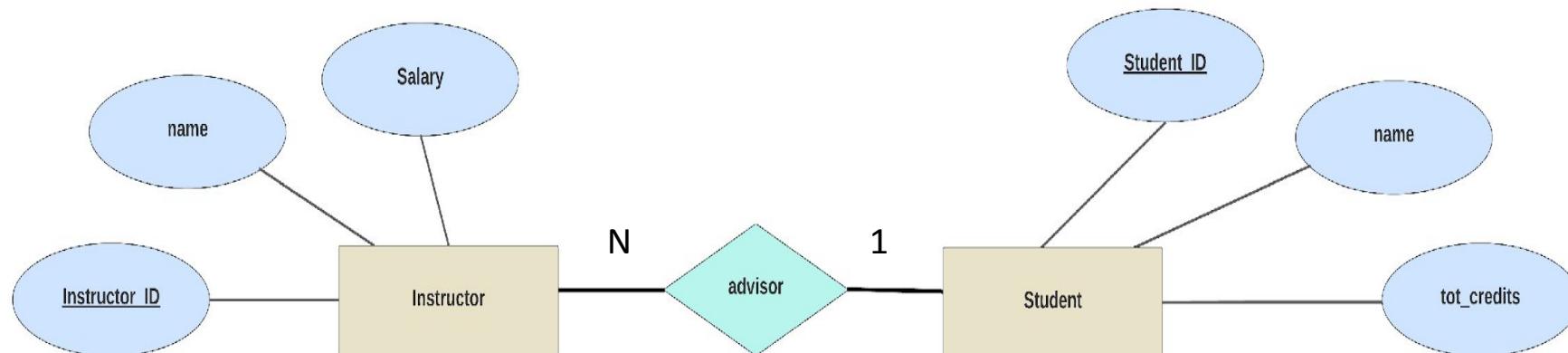
One-to-Many or Many-to-One

- the primary key of the “**many**” side is a **minimal superkey** and is used as the primary key.
- For example, if the advisor relationship is **many**-to-one from **student** to instructor, that is, each student can have at most one advisor
 - then the primary key of advisor is simply the **primary key of student**.



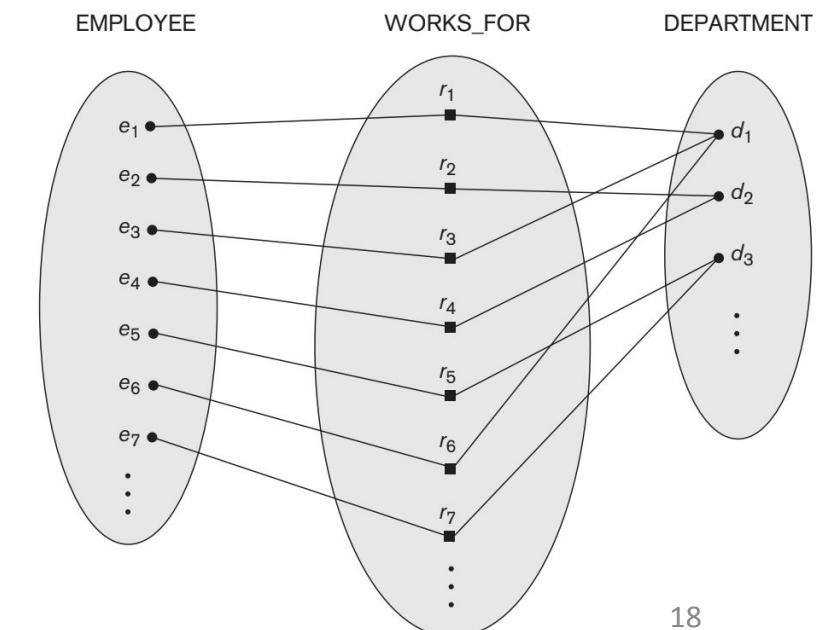
One-to-Many or Many-to-One

- the primary key of the “**many**” side is a **minimal superkey** and is used as the primary key.
- On the contrary, if an instructor can advise only one student, that is, if the advisor relationship is **many-to-one** from **instructor** to student
 - then the primary key of advisor is simply the **primary key of instructor**.



One-to-Many or Many-to-One

- the primary key of the “**many**” side is a **minimal superkey** and is used as the primary key.
- Let us consider the entity sets Employee and Department and the relationship `works_for`.
- In the figure, the employees e_1 , e_3 , and e_6 work for department d_1 ; the employees e_2 and e_4 work for department d_2 ; and the employees e_5 and e_7 work for department d_3 .
 - then the primary key of `works_for` is simply the **primary key of employee**.





PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Weak Entity Sets

S Nagasundari

Department of Computer Science Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering

Weak Entity Sets

- A weak entity set is one whose existence is dependent on another entity set, called its **identifying entity** set;
- Instead of associating a primary key with a weak entity, we use the **primary key** of the **identifying entity**, along with extra attributes, called **discriminator attributes** to uniquely identify a weak entity.
- An entity set that is not a weak entity set is termed a **strong entity** set.
- Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be existence dependent on the identifying entity set.
- The identifying entity set is said to own the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**.

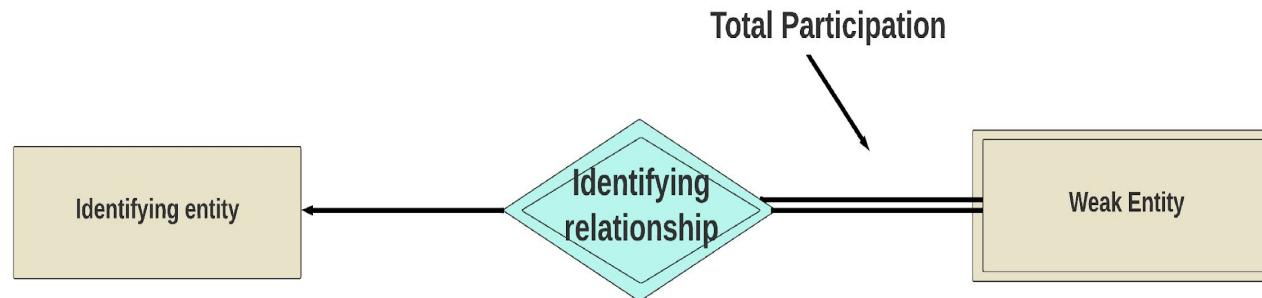
Weak Entity Sets

- In the university database, let us consider the following schemas for the entities course and section:
- Course (Course_ID, title, dept_name, credits)
 - Primary key {Course_ID}
- Section (Course_ID, sec_ID, semester, year, building, room_number, time_slot_id)
 - Primary key {Course_ID, sec_ID, semester, year}
- Now let us consider that every course should have a section where the course would be taking place.
- How do we relate both these entities for showing the above relationship?

Weak Entity Sets

- For the above question, we would connect sections to courses using a relationship called "sec_course".
- But we should note that the "sec course" information is kind of **repetitive** because each section already has a "course id" telling us which course it belongs to.
- To solve this problem there are 2 possibilities:
 - **Possibility 1: Remove the "sec_course" relationship**
 - Then the connection between sections and courses isn't as clear. That is it becomes implicit in an attribute, which is not desirable
 - **Possibility 2: Remove the "course_id" from the section** and keep only the other details like section number, year, and semester.
 - If we do this, sections might share the same details (section number, year, and semester) for different courses, making them hard to tell apart.
- To deal with this problem, we treat the relationship sec_course as a special relationship that provides extra information, in this case, the course id, required to identify section entities uniquely

- The identifying relationship is **many-to-one** from the weak entity set to the identifying entity set, and the participation of the weak entity set in the relationship is total.



many-to-one/ one-to-many:
many : weak entity
one : identifying entity

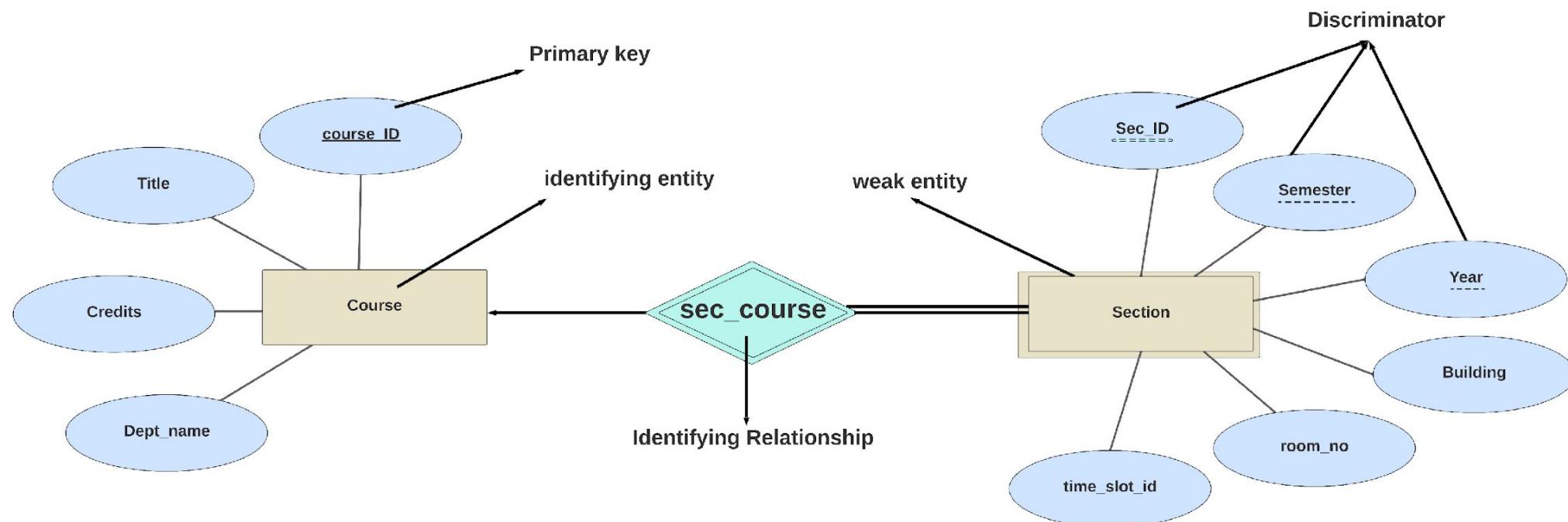
- The identifying relationship set should **not** have any descriptive attributes, since any such attributes can instead be associated with the weak entity set.
- The ER representation of the weak entity is as shown

- For the situation considered for sec_course
 - Weak entity: Section
 - Identifying entity: Course,
 - Identifying relationship: Sec_course.
 - The primary key of the section is formed by the primary key of the identifying entity set (that is, course), plus the discriminator of the weak entity set (that is, section).
 - Primary key is {course id, sec id, year, semester}.

Primary-key(Identifying entity: Course)

Discriminator(weak entity: Section)

- Note that the ER diagram for the same would be as follows:
 - a weak entity set is depicted via a double rectangle
 - discriminator is underlined with a dashed line.
 - The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a double diamond.

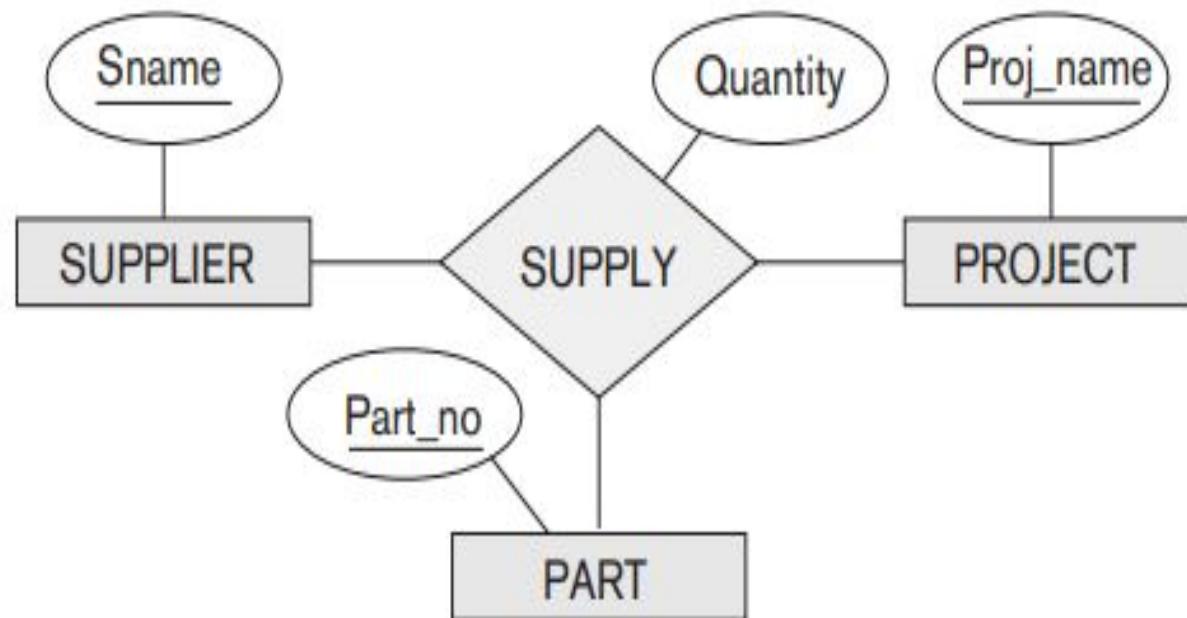


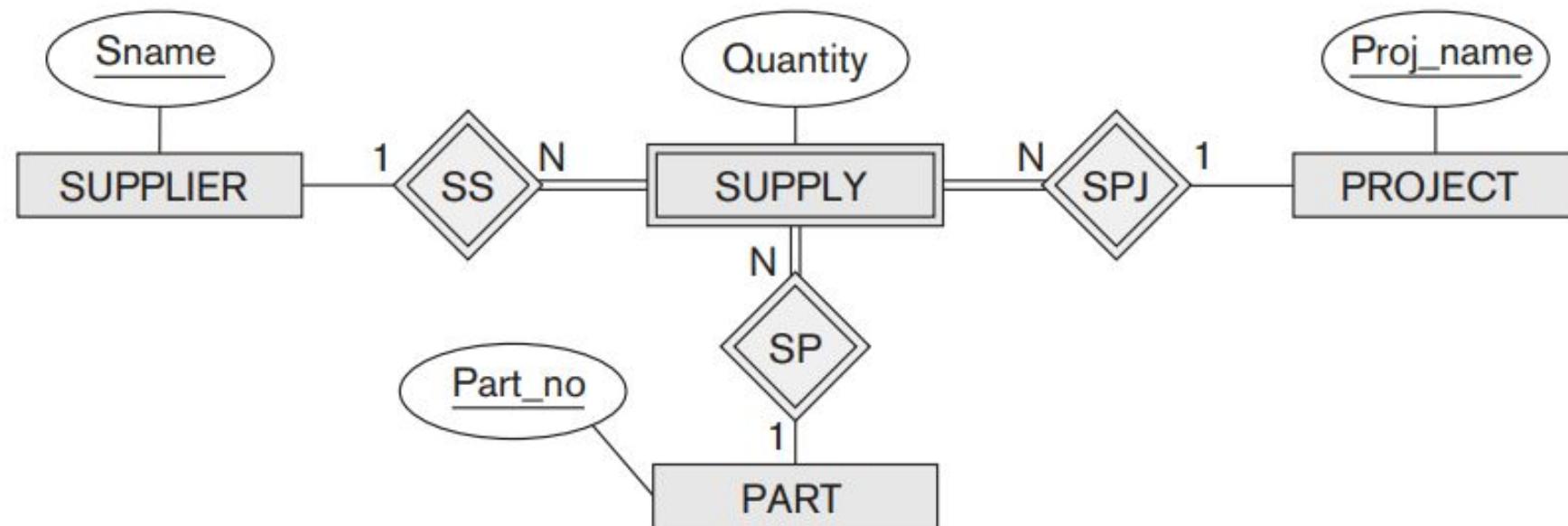
Weak Entity Sets

- Note that we could have chosen to make sec_id globally unique across all courses offered in the university, in which case the section entity set would have had a primary key.
- However, conceptually, a section is still dependent on a course for its existence, which is made explicit by making it a weak entity set.
- In general, a weak entity set must have total participation in its identifying relationship set, and the relationship is many-to-one toward the identifying entity set.

Weak Entity Sets

- A weak entity set can participate in relationships other than the identifying relationship.
 - For instance, the section entity could participate in a relationship with the time slot entity set, identifying the time when a particular class section meets.
- It is also possible to have a weak entity set with more than one identifying entity set. A particular weak entity would then be identified by a combination of entities, one from each identifying entity set.
- The primary key of the weak entity set would consist of the union of the primary keys of the identifying entity sets, plus the discriminator of the weak entity set.





SUPPLY represented as a weak entity type



PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Structure of Relational Model

S Nagasundari

Department of Computer Science Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering

- **The relational model represents how data is stored in Relational Databases.**
- The relational data model uses tables (relations) to store data and represent relationships between them. Each table is assigned a unique name.
- **Each table has multiple columns with unique names, representing attributes of the data.**
- Tables in the relational model are also known as relations and store records of specific types with fixed attributes.
- Information is represented by rows in the tables, where each row represents one piece of data.
- The relational data model is widely used and forms the basis of the majority of modern database systems.
- It belongs to the category of record-based models, organizing data in fixed-format records of various types.

You are tasked with designing a database for a University Management System to manage **instructors** and **courses**. Each instructor has a unique ID, name, department, and salary. Similarly, each course has a unique course ID, title, department, and credits, along with possible **prerequisite courses**.

Question : If you were to design a relational database for this system, what would its structure look like?

(Consider the following:

- How would you define the tables for instructors and courses?
- What primary and foreign keys would you use to ensure unique identification and relationships?
- How would you handle prerequisite courses in the database?)



- Let us consider the information we are supposed to store for an instructor, in this, each instructor would have:
 - a unique identifier called **ID**,
 - his/her **name**,
 - the **department** he is associated with
 - **salary**.
- These attributes are stored in a relational database in the Instructor table, with each attribute as a column header.
- Course details are stored in the Course table with columns for course_ID, topic, department, and credits.
- The Prereq table stores prerequisite details, with columns for course_id and prereq_id, where each row represents a course and its prerequisite.

To summarize the structure of the relational model would be something like this:

- There would be three tables as shown
- Each table would have a unique column name also called attributes

Instructor			
ID	Name	Dept_name	Salary
Course			
Course_ID	Topic	Dept_name	Credits
Prereq			
Course_ID	Prereq_ID		

On populating these tables with values it would look something like this:

ID	name	dept_name	salary
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

Instructor table

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Course table

course_id	prereq_id
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Prereq table

Let us consider the highlighted row in the table Prereq table:

course_id	prereq_id
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

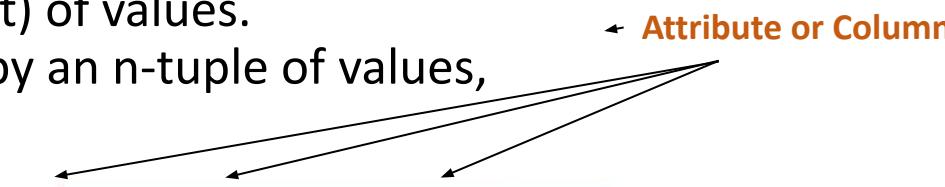
What does this row indicate?

A row in the Prereq table indicates that "BIO-301" requires "BIO-101" as a prerequisite. Similarly, a row in the Instructor table represents the relationship between an instructor's ID and their name, department, and salary.

- So, in general, a row in a table represents a relationship among a set of values.
- A table is a collection of such relationships.
- There is a close correspondence between the concept of the table and the mathematical concept of relation, from which the relational data model takes its name.
- In mathematical terminology, a **tuple** is simply a sequence (or list) of values.
- A relationship between n values is represented mathematically by an n -tuple of values, that is, a tuple with n values corresponds to a row in a table.

Thus, in the relational model:

- the term **relation** is used to refer to a **table**
- The term **tuple** is used to refer to a **row**.
- The term **attribute** refers to a **column** of a table.



The diagram shows a table representing an instructor relation. The columns are labeled **ID**, **name**, **dept_name**, and **salary**. The rows contain data for various instructors. One specific row, for Mozart, is highlighted with a blue background. Three arrows point from the text definitions to this highlighted row: one arrow points from "Attribute or Column" to the **dept_name** column, another from "Record or Tuple" to the highlighted row, and a third from "Instructor table or Relation" to the entire table.

ID	name	dept_name	salary
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

Instructor **table** or **Relation**

- Consider the situation in which the university has employed another instructor in a particular department, then we must modify the instructors' table with the new instructor's table
- So the Instructor's table entries would keep changing as and when there are new instructors or if the current instructor leaves
- A specific instance of a relation, which includes a particular set of rows, is referred to as a Relational Instance.**
- The instance of instructor shown in Figure has 12 tuples, corresponding to 12 instructors.

ID	name	dept_name	salary
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

Instructor table

- Consider the two tables given below. Are both of them the same or different?
- The relation is a set of tuples. **So like in any other set, the Order of arrangement of tuples doesn't matter.**
- Therefore, whether the tuples of a relation are listed in sorted order, as in Instructor table 1, or are unsorted, as in Instructor table 2, does not matter; the relations in the two figures are the same, since both contain the same set of tuples.

ID	name	dept_name	salary
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

Instructor table 1

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

Instructor table 2

Structure of Relational Model – Domain

- In the instructors table the column “Salary” could contain only the value of possible salary values, similarly the column name would contain only a set of all possible instructor names
- Therefore, **For each attribute of a relation, there is a set of permitted values, called the Domain of that attribute.**
- Thus the domain of “Salary” column in Instructors table is the set of all possible salary values
- Similarly, the set of all possible instructors’ names would be the domain of “Name” column in Instructor table

- To maintain simplicity and consistency in the database, we require that all domains for attributes in any relation be **Atomic**.
- An Atomic domain means that each element within the domain is considered one indivisible unit.
- For instance, let's consider if there was a "phone number" attribute in the "instructor" table.
 - If the attribute can store multiple phone numbers for an instructor, it becomes non-atomic because it contains subparts (individual phone numbers within the set).
 - However, even if we store only one phone number, if we further break it down into parts like country code, area code, and local number, it remains non-atomic.
 - To ensure atomicity, we should treat each phone number as a single, indivisible unit. This way, the "phone number" attribute would have an atomic domain, making it simpler to manage and maintain data integrity in the database.

Structure of Relational Model – Null values

- Let's consider the "phone number" attribute in the "instructor" relation. Some instructors may not have a phone number at all, or their phone numbers might be unlisted.
- In such cases, we use the **Null** value to represent that there is no available phone number for the instructor.
- Null values can cause challenges when accessing or updating the database. They may lead to issues in calculations, comparisons, and querying operations.
- It's thus best to eliminate null values whenever possible to maintain data consistency and integrity. Proper data validation and default values can be used to avoid nulls when entering data.



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Database Schema

S Nagasundari

Department of Computer Science Engineering

What do we mean by a database schema and how is it different from a database instance?

- The schema is a logical design of the database.
- In general, a relation schema consists of a list of attributes and their corresponding domains.
- Whereas the database instance is the snapshot of the data in the database at a given instance of time.
- Basically if we consider these w.r.t. programming languages,
 - **Relation** corresponds to a **variable**,
 - **relation schema** corresponds to **type definition**
 - **relation instance** corresponds to **value of a variable**
- The value of a given variable may change with time; similarly, the contents of a relation instance may change with time as the relation is updated. In contrast, the schema of a relation does not generally change.

Let us consider the instructor table and department tables in the university database

Their respective schemas are:

Instructor (ID, name, dept_name, salary)

Department (dept_name, building, budget)

- If we notice, the attribute dept_name appears in both the instructor schema and the department schema.
- Do you think this is a coincidence? If not what is the significance of this?

ID	name	dept_name	salary
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

Instructor table

dept_name	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Department table

- The duplication of dept_name is not a coincidence.
- Using **common attributes** in relation schemas is one way of relating tuples of distinct relations.
- For example, suppose we wish to find information about all the instructors who work in the Watson building.
 - We look first at the department relation to find the dept name of all the departments housed in Watson.
 - Then, for each such department, we look in the instructor relation to find the information about the instructor associated with the corresponding dept name.
 - This way we can make use of the common attributes to get the information we need from different relations.

ID	name	dept_name	salary
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

Instructor table

dept_name	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Department table

Question: Like the above could you come up with the schemas for the following relations that are to be stored in a the database for example:

A company needs to design a database to manage its employees, departments, projects, and related information. The database should track:

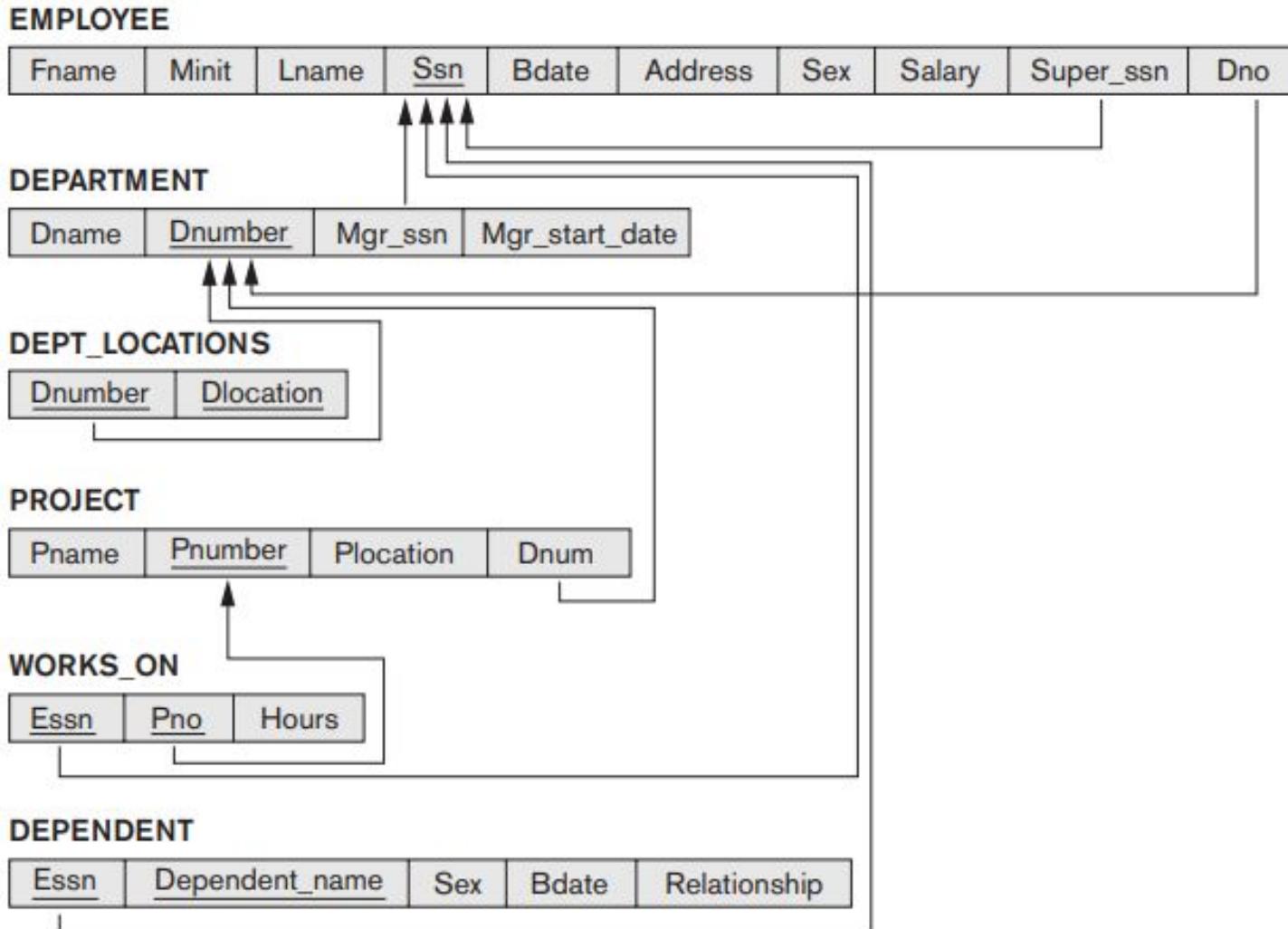
- Employee details including name, ID, salary, their supervisor, etc.
- Department information including name, number, manager, and manager's start date.
- Department locations.
- Project details including name, number, location, and associated department.
- Which employees work on which projects and for how many hours.
- Employee dependents' information including name, relationship to the employee, etc.

- The company database maintains details of employees including their name, ID, birth date, address, gender, salary, and supervisor:
 - **EMPLOYEE** (Fname, Minit, Lname, Ssn, Bdate, Address, Sex, Salary, Super_ssn, Dno)
- Each department in the company has a name, number, manager, and the date when the manager started:
 - **DEPARTMENT** (Dname, Dnumber, Mgr_ssn, Mgr_start_date)
- Departments can have multiple locations:
 - **DEPT_LOCATIONS** (Dnumber, Dlocation)

- The company manages various projects, each with a name, number, location, and associated department
 - **PROJECT** (Pname, Pnumber, Plocation, Dnum)
- Employees work on different projects for a certain number of hours:
 - **WORKS_ON** (Essn, Pno, Hours)
- The database also tracks information about employees' dependents:
 - **DEPENDENT** (Essn, Dependent_name, Sex, Bdate, Relationship)

Database Management Systems

Exercise (Soln.)





PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Keys, Constraints

S Nagasundari

Department of Computer Science Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering

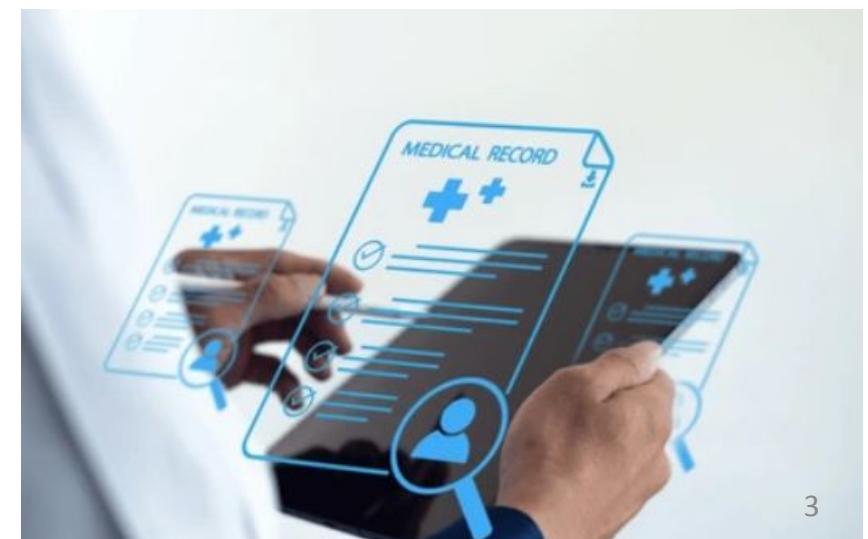
Scenario:

You are designing a database for a hospital management system. A patient's age was recorded in the height column, leading to incorrect medical records. Two patients with the same name have had their medical records mixed up, which is highly dangerous and potentially life-threatening.

Question

How can we design the database to prevent such data entry errors?

(*Hint: enforcing certain “rules”*)



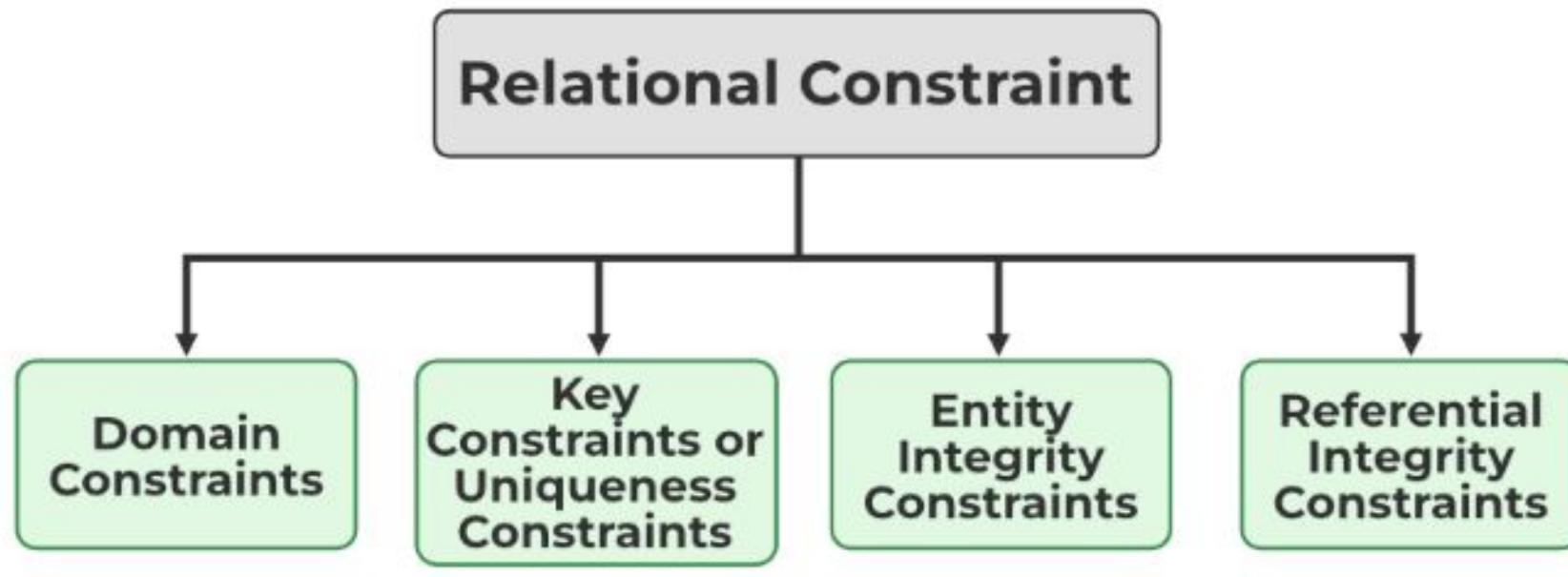
Constraints determine which values are permissible and which are not in the database. They are of three main types:

- **Inherent or Implicit Constraints:**
 - Based on the data model itself. (E.g., relational model does not allow a list as a value for any attribute)
- **Schema-based or Explicit Constraints:**
 - Expressed in the schema by using the facilities provided by the model. (E.g., max. cardinality ratio constraint in the ER model)
- **Application based or semantic constraints:**
 - Beyond the expressive power of the model and must be specified and enforced by the application programs.

Constraints are conditions that must hold on all valid relation states.

The Main types of (**Explicit/Schema-based**) constraints that can be expressed in the relational model:

- **Domain** constraint.
- **Key constraints** and **Constraints on NULL** values
- **Integrity** Constraints
 - **Entity integrity** constraints
 - **Referential integrity** constraints



Types of Relational Constraints

Domain Constraints

Domain Constraint specifies that

- Within each tuple, the value of each attribute A must be an atomic value from the domain $\text{dom}(A)$.
- Every domain must contain atomic values (smallest indivisible units) which means composite and multi-valued attributes are not allowed.
- Every value in a tuple must be from the domain of its attribute (or it could be null, if allowed for that attribute)

Could you think of different datatypes associated with the domain?

Example:

EID	Name	Phone
01	Bikash Dutta	123456789 234456678

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

- The data types associated with domains include
 - Standard **numeric** data types for **integers** (such as short integer, integer, and long integer)
 - **Real numbers** (float and double-precision float).
 - **Characters**,
 - **Booleans**,
 - Fixed-length strings, Variable-length strings, as are date, time, timestamp, and other special data types.
 - Domains can also be described by a **subrange of values** from a data type or as an **enumerated** data type in which all possible values are explicitly listed

- In a database, a relation (or table) consists of a set of tuples (or rows).
- No two tuples can be identical across all attributes.
- Each tuple must have a set of attributes with unique values to ensure distinct identification.



- A superkey is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation.
- Let us consider the example of University DB,
- the **ID** attribute of the relation instructor is sufficient to distinguish one instructor tuple from another. Thus, ID is a superkey.
- The name attribute of instructor, on the other hand, is not a superkey, because several instructors might have the same name.

Formally defining;

- If we say that a subset SK of R is a superkey for r:
 - we are restricting consideration to instances of relations r in which no two distinct tuples have the same values on all attributes in K.
 - That is, if t_1 and t_2 are in r and $t_1 \neq t_2$, then $t_1.SK \neq t_2.SK$.

Superkey of R:

- Is a set of attributes SK of R with the following condition:
 - No two tuples in any valid relation state $r(R)$ will have the same value for SK
 - That is, for any distinct tuples t_1 and t_2 in $r(R)$, $t_1[SK] \neq t_2[SK]$
 - This condition must hold in any valid state $r(R)$

Let us take this situation into consideration:

Instructor (ID, name, dept_name, salary)

- Now as seen before {ID} is a superkey as it uniquely identifies each tuple in the relation
- **What about the set {ID, name} is this also a superkey?**
- The answer is yes
- A superkey may contain extraneous attributes like shown above
- If SK is a superkey, then so is any superset of SK.
- We are often interested in superkeys for which no proper subset is a superkey. Such **minimal superkeys** are called **candidate keys**.

Let us now consider a Car relational schema as shown below

Car (State, Reg#, SerialNo, Make, Model, Year)

- If we notice, the SerialNo is an candidate key as it is unique for the Car
- But we also know that {State, Reg#} together are also unique for a given car
- So in the above case {SerialNo} and {State, Reg#} both are minimal superkeys and also called candidate keys
- Thereby we can conclude that A relation schema may have more than one minimal superkey. In this case, each of them is called a candidate key.

Primary Key

If a relation has several candidate keys, one is chosen arbitrarily to be the **primary key**.

- The primary key attributes are underlined.
 - Example: Consider the CAR relation schema:
 - **CAR**(State, Reg#, SerialNo, Make, Model, Year)
We chose SerialNo as the primary key

State	RegNum	<u>SerialNo</u>	Make	Model	Year
Texas	TX9101	SN1122334455	Ford	Focus	2017
California	CA1234	SN1234567890	Toyota	Corolla	2015

- The primary key value is used to uniquely identify each tuple in a relation
- Provides the tuple identity
- Also used to reference the tuple from another tuple
 - General rule: Choose as primary key the smallest of the candidate keys (in terms of size)
 - Not always applicable – choice is sometimes subjective
 - It is customary to list the primary key attributes of a relation schema before the other attributes;

Let us consider the classroom relation in the university database

- Schema :
 - **classroom (building, room number, capacity)**
- Here the primary key consists of two attributes, building and room number, which are underlined to indicate they are part of the primary key.
- Neither attribute by itself can uniquely identify a classroom, although **together they uniquely identify** a classroom.

Key Constraints

Primary keys must be chosen with care.

- Eg: The name of a person is insufficient, because there may be many people with the same name.
In our country, the aadhar number attribute of a person would be a candidate key.

An alternative is to use some unique combination of other attributes as a key.

- The primary key should be chosen such that its attribute values are never, or are very rarely, changed.
 - **For instance, the address field of a person should not be part of the primary key, since it is likely to change.**
 - Unique identifiers issued by enterprises typically do not change. However, if two enterprises merge and have issued the same identifier, a reallocation may be necessary to ensure uniqueness.

- The schema along with the primary keys for the university database is as shown:

classroom(building, room_number, capacity)

department(dept_name, building, budget)

course(course_id, title, dept_name, credits)

instructor(ID, name, dept_name, salary)

section(course_id, sec_id, semester, year, building, room_number, time_slot_id)

teaches(ID, course_id, sec_id, semester, year)

student(ID, name, dept_name, tot_cred)

takes(ID, course_id, sec_id, semester, year, grade)

advisor(s_ID, i_ID)

time_slot(time_slot_id, day, start_time, end_time)

prereq(course_id, prereq_id)

- A relational database state DB of S is a set of relation states
 - $DB = \{r_1, r_2, \dots, r_m\}$ such that each r_i is a state of R_i and such that the r_i relation states satisfy the integrity constraints specified in IC.
- A relational database state is sometimes called a relational database snapshot or instance.
- A database state that does not obey all the integrity constraints is called **not valid**
- A state that satisfies all the constraints in the defined set of integrity constraints IC is called a **valid state**

Entity integrity constraint states that

- The primary key attributes **PK of each relation schema R in S cannot have null values** in any tuple of $r(R)$.
- This is because primary key values are used to identify the individual tuples.
- $t[PK] \neq \text{null}$ for any tuple t in $r(R)$
- **If PK has several attributes, null is not allowed in any of these attributes**
- Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

Example:

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

Explanation: In the above relation, EID is made the primary key, and the primary key can't take NULL values but in the third tuple, the primary key is null, so it is violating Entity Integrity constraints.

A foreign-key constraint from attribute(s) A of relation r1 to the primary-key B of relation r2 states that

- on any database instance, the value of A for each tuple in r1 must also be the value of B for some tuple in r2.
- Attribute set A is called a foreign key from r1, referencing r2.
- The relation r1 is also called the **referencing relation** of the foreign-key constraint, and r2 is called the **referenced relation**.

Department Table :

Dept_name	Building	Budget
Computer Science	Watson	500000.00
Mathematics	Johnson	300000.00
Physics	Smith	400000.00

Instructor Table :

ID	Name	Dept_name	Salary
1	John	Computer Science	90000.00
2	Jane	Mathematics	85000.00
3	Paul	Physics	80000.00

The `Dept_name` attribute in the `Instructor` table is a foreign key referencing the `Dept_name` attribute in the `Department` table.

For example, let us consider the following schema:

Instructor (ID, name, Dept_name, Salary)

Department (Dept_name, building, budget)

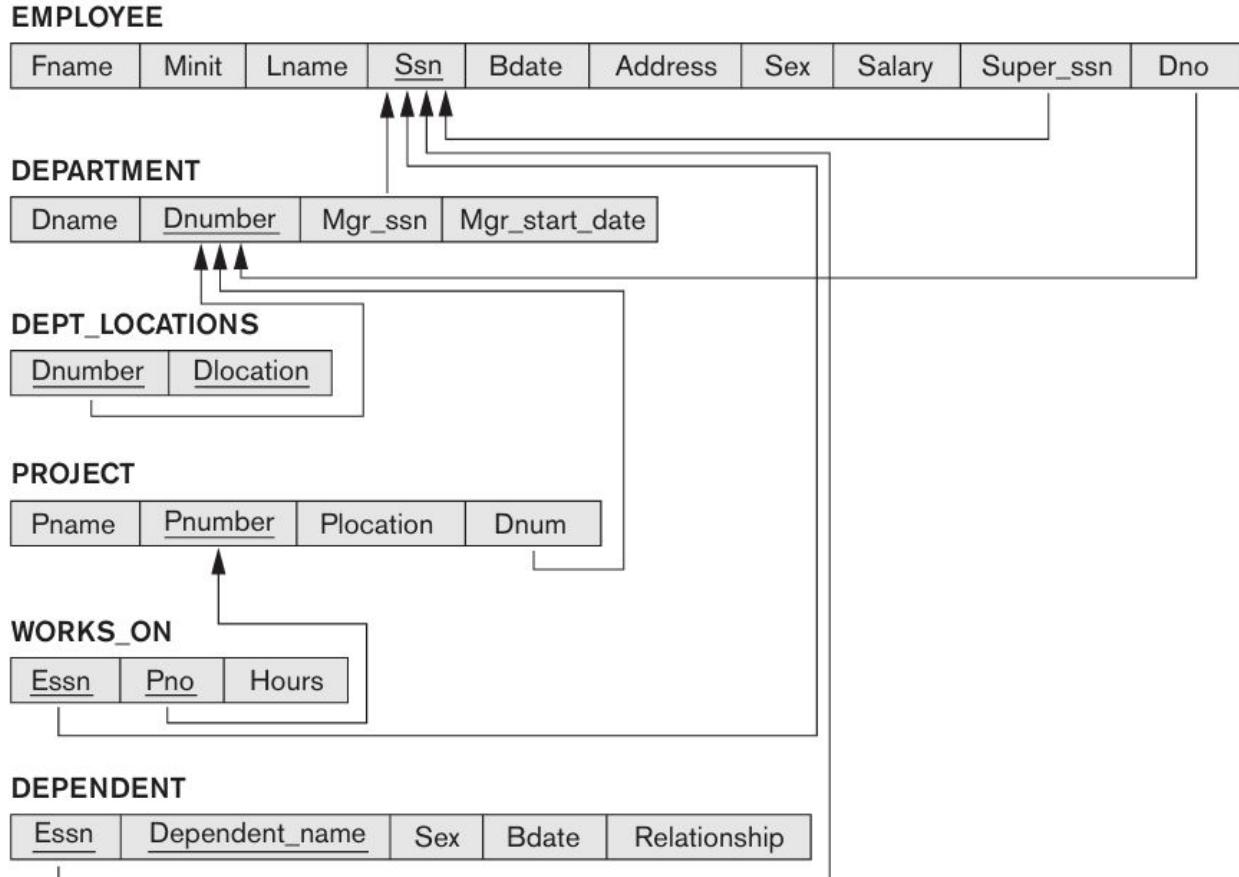
- attribute “dept_name” in instructor is a foreign key from the instructor table, referencing the department table;
- Note that dept_name is the primary key of the department.

So here,

- Instructor referencing relation
- Department referenced relation

**** Note that in a foreign-key constraint, the referenced attribute(s) must be the primary key of the referenced relation ****

- This is a more general case of the foreign key constraint.
- A referential-integrity constraint, relaxes the requirement that the referenced attributes form the primary key of the referenced relation.
- In general, a referential integrity constraint requires that the values appearing in specified attributes of any tuple in the referencing relation also appear in specified attributes of at least one tuple in the referenced relation.



Referential integrity constraints displayed on the COMPANY relational database schema.

- Note that in the above example, the time slot does not form a primary key of the time slot relation, although it is a part of the primary key;
- thus, we cannot use a foreign-key constraint to enforce the above constraint.
- In fact, **foreign-key constraints are a special case of referential integrity constraints**, where the referenced attributes form the primary key of the referenced relation.
- Database systems today typically support foreign-key constraints, but they do not support referential integrity constraints where the referenced attribute is not a primary key

Statement of the constraint:

- Consider two relations R1 and R2, the value in the foreign key column (or columns) FK of the referencing relation R1 can be either:
 - A value of an existing primary key value of a corresponding primary key PK in the referenced relation R2, or
 - A null.
- The FK in R1 should not be a part of its own primary key.

Example:

EID	Name	DNO
01	Divine	12
02	Dino	22
04	Vivian	14

DNO	Place
12	Jaipur
13	Mumbai
14	Delhi

Explanation: In the tables, the DNO of Table 1 is the foreign key, and DNO in Table 2 is the primary key. DNO = 22 in the foreign key of Table 1 is not allowed because DNO = 22 is not defined in the primary key of table 2. Therefore, Referential integrity constraints are violated here.



PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Conversion of E-R Diagram to Relational Schema

S Nagasundari

Department of Computer Science Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S
Department of Computer Science and Engineering

Reducing ER diagrams to Relational Schema

- Both the E-R model and the relational database model are abstract and logical representations of real-world organizations.
- Due to shared design principles, an E-R design can be transformed into a relational design.
- 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.

- Steps to convert ER diagram into a relational schema
 - Mapping of strong entity sets
 - Mapping of weak entity sets
 - Mapping of strong entity sets with complex attributes
 - Compound attributes
 - Derived attributes
 - Multivalued attributes
 - Mapping of relational sets
 - 1:1 or one-to-one
 - 1:N or one-to-many or many-to-one
 - M:N or many-to-many
 - N-ary relations

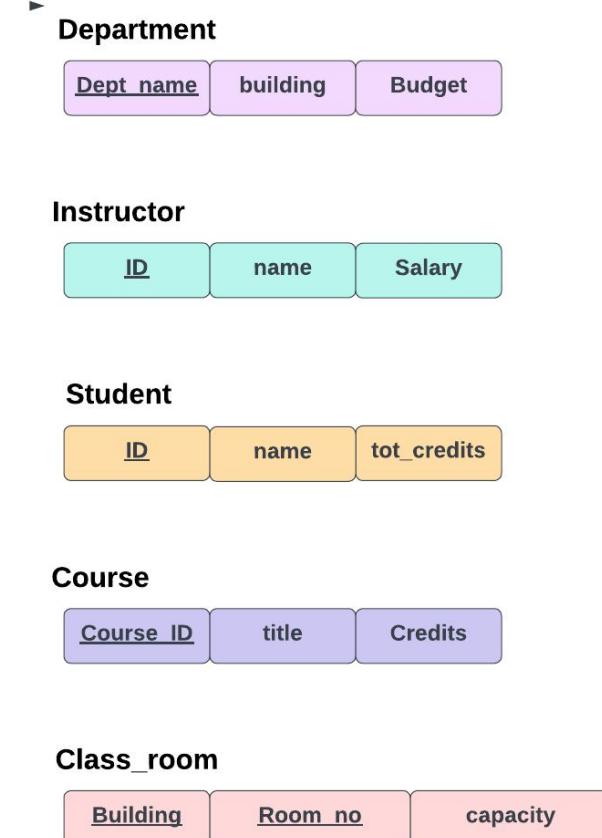
- Let **E** be a strong entity set with only **simple descriptive attributes** a_1, a_2, \dots, a_n .
- We represent this entity with a **schema** called **E** with n distinct attributes.
- Each tuple in a relation on this schema corresponds to one entity of the entity set E.
- For schemas derived from strong entity sets, the primary key of the entity set serves as the primary key of the resulting schema.
- From the university DB:
 - Consider the entity set “student”.
 - Attributes: ID, name, tot_cred.
- We represent this entity set by a schema called student with three attributes:
- **Student** (ID, name, tot_cred)

Student

ID	name	tot_credits
----	------	-------------

- Similarly converting all the strong entities with simple attributes in the university DB, we get the following schemas:

- Classroom**(building, room number, capacity)
- Department**(dept name, building, budget)
- Course**(course id, title, credits)
- Instructor**(ID, name, salary)
- Student**(ID, name, tot cred)

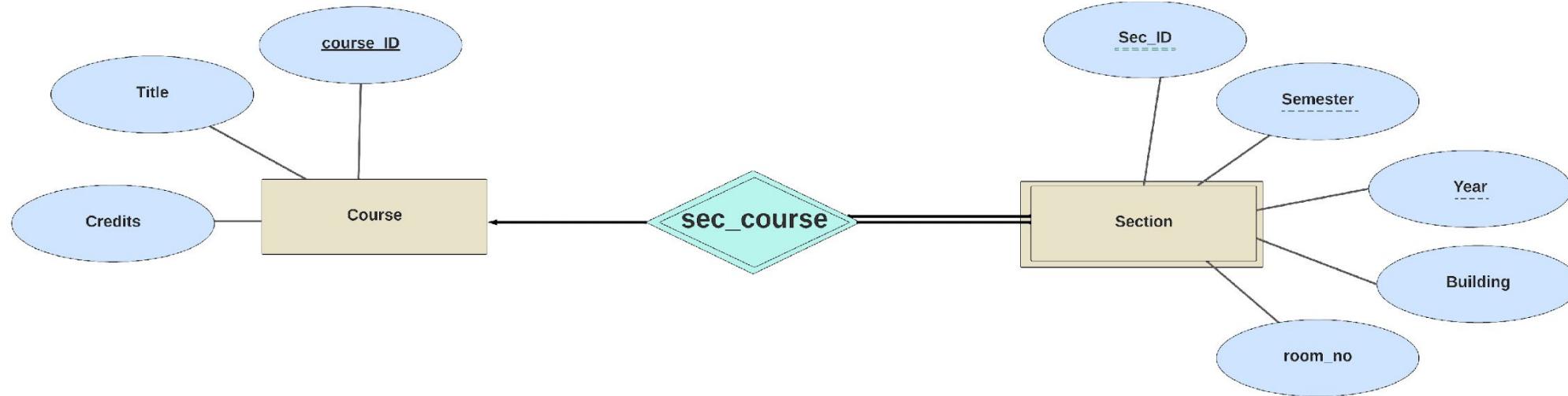


- Let **A** be a **weak entity set** with attributes {a₁, a₂, ..., a_m}.
- Let B be the strong entity set on which A depends. Let the **primary key** of **B** consist of attributes {b₁, b₂, ..., b_n}.
- We represent the entity set A by a relation schema called A with one attribute for each member of the set:

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

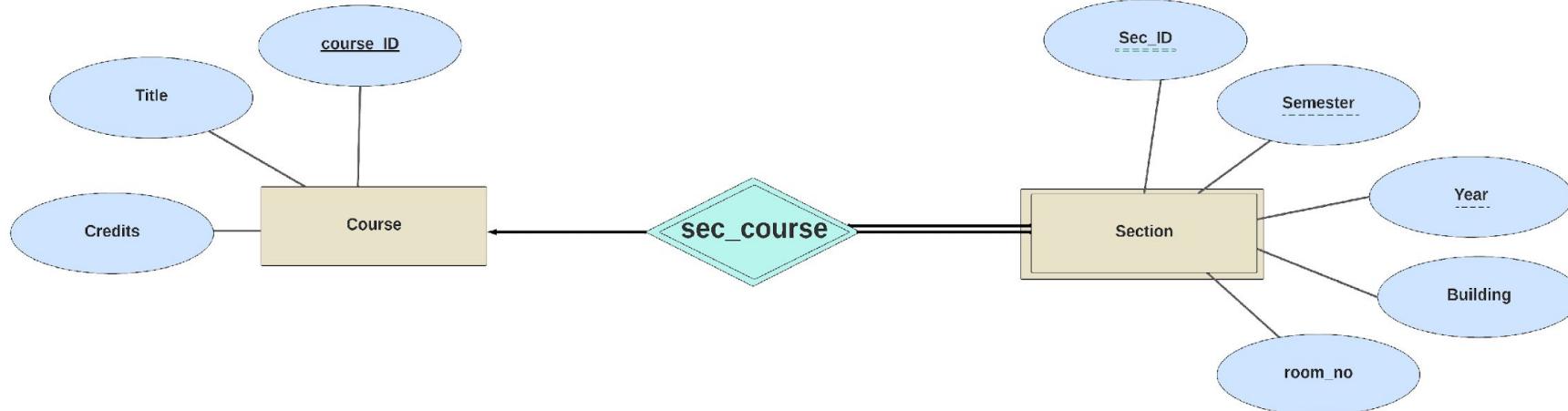
- The primary key consists of the primary key of Identifying/strong entity set union with the discriminator of weak entity
- Foreign key constraint is also added to the relational schema of the weak entity (to the primary-key of the identifying entity set)

- Consider the section entity of the university database as shown

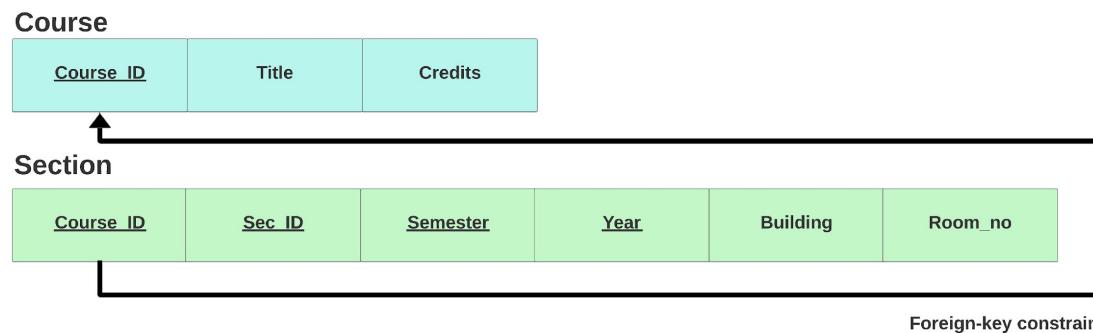


- Relational schema:
 - Section**(course_id, sec_id, semester, year, building, room_no)

- Consider the section entity of the university database as shown

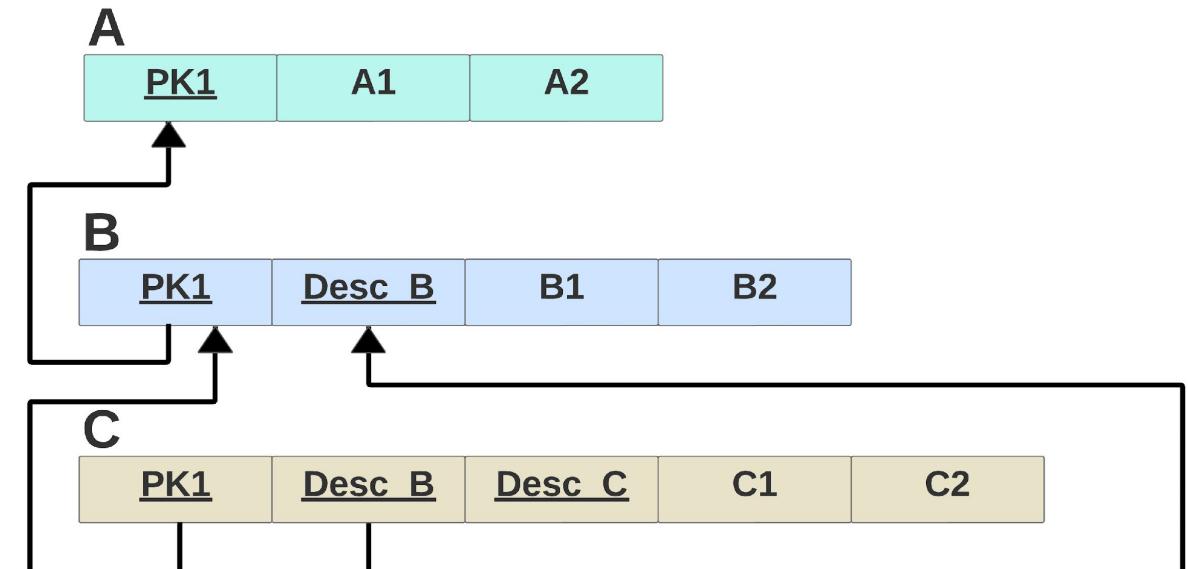
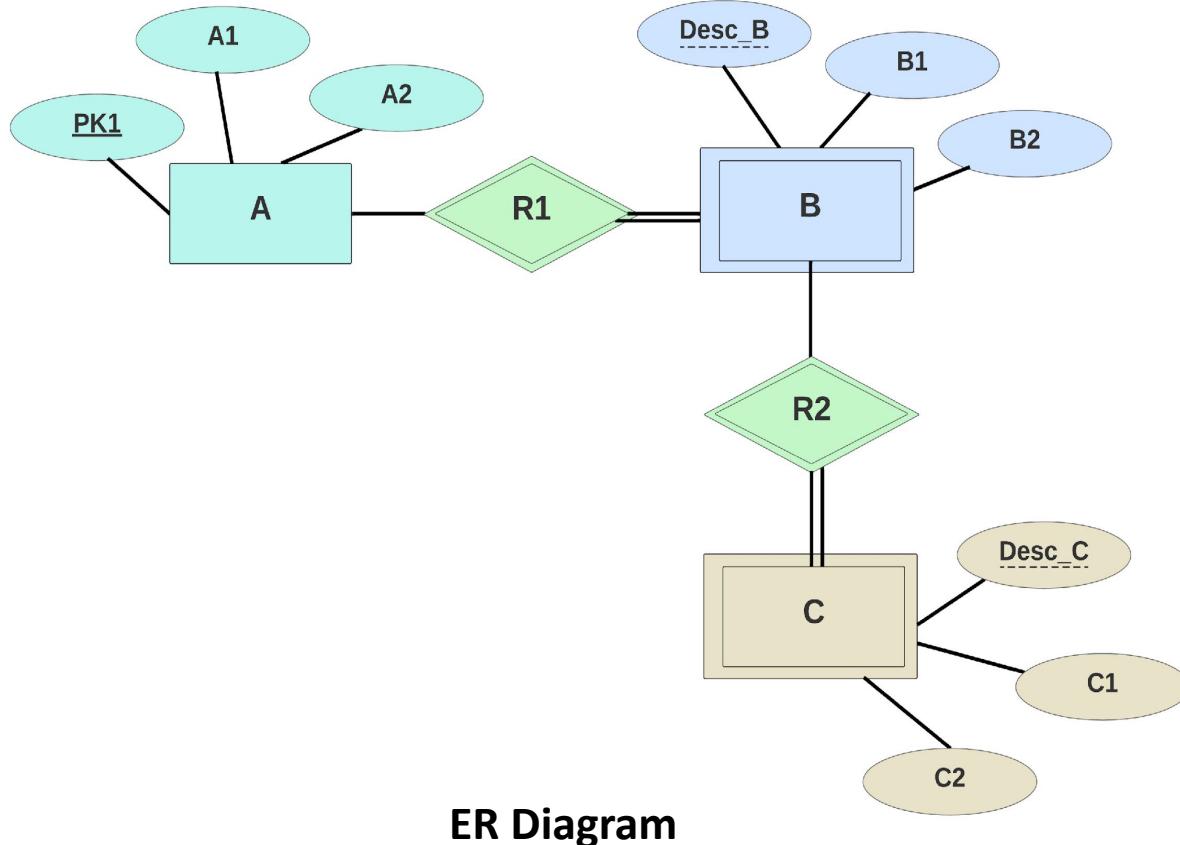


- Relational schema with foreign-key constraint:



- For schemas derived from a weak entity set, the combination of the primary key of the strong entity set and the discriminator of the weak entity set serves as the primary key of the schema.
- In addition to creating a primary key, we also create a foreign-key constraint on the relation A, specifying that the attributes b₁, b₂, ..., b_n reference the primary key of the relation B.
- The foreign-key constraint ensures that for each tuple representing a weak entity, there is a corresponding tuple representing the corresponding strong entity

- In general, for the given ER diagram, the relational schema is as given:



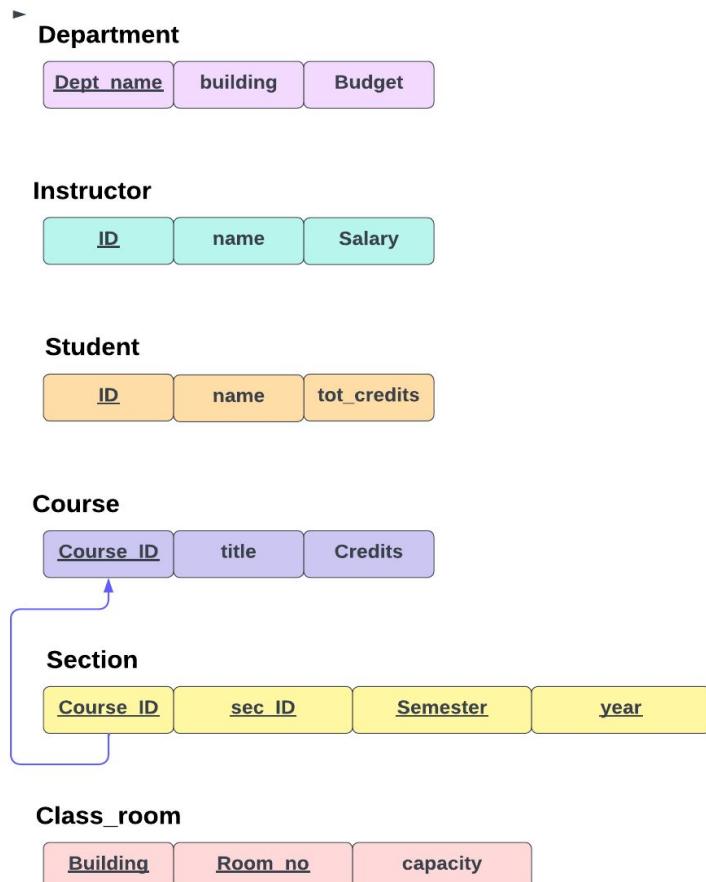
ER Diagram

Relational Schema

Database Management Systems

Representation of Weak Entity Sets

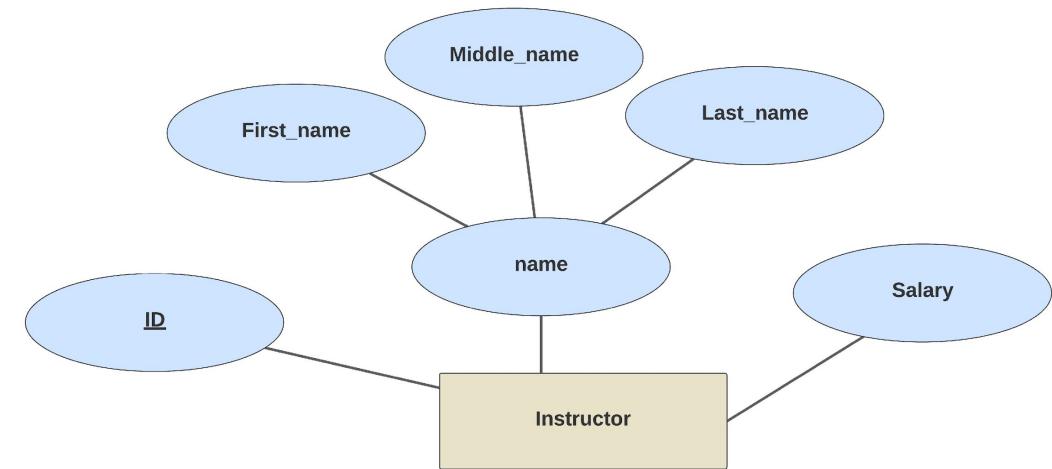
- After incorporating the weak entities the university database would look like:



COMPOSITE ATTRIBUTES

- Composite attributes are handled by creating a separate attribute for each of the component attributes (Do not create a separate attribute for the composite attribute itself).
- Consider the version of the Instructor entity set as shown.
- For the composite attribute “name”, the schema generated for the instructor contains the attribute’s first name, middle initial, and last name;
- The schema:

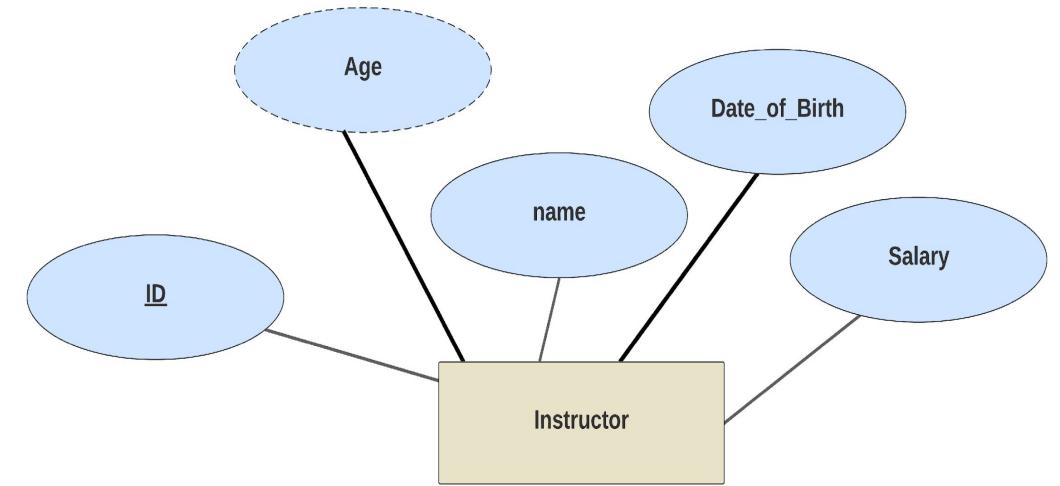
Instructor(ID, First_name, Middle_name, Last_name, Salary)



DERIVED ATTRIBUTES

- Derived attributes are not explicitly represented in the relational data model.
- They can be represented as stored procedures, functions, or methods in other data models
- Consider the version of the Instructor entity set as shown.
- The derived attribute Age would be eliminated in the relational schema
- The schema:

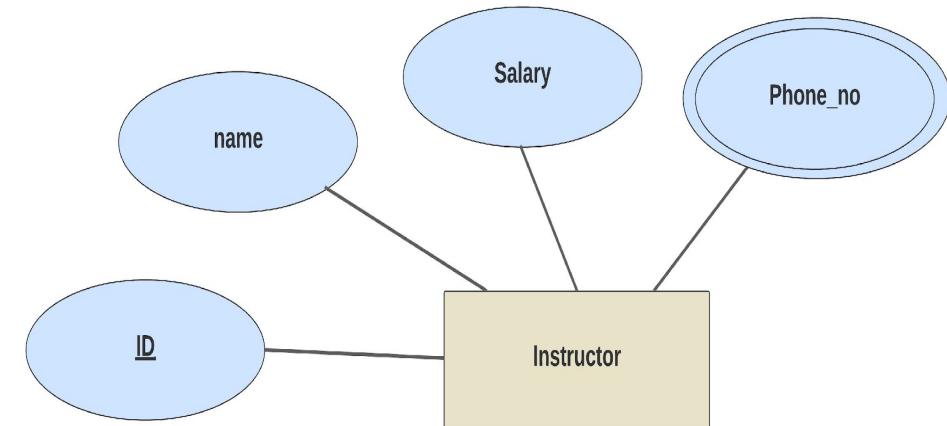
Instructor(ID, name, Date_of_Birth , Salary)



MULTIVALUED ATTRIBUTES

- For a multivalued attribute M, create a relation schema R with an attribute A that corresponds to M and attributes corresponding to the primary key of the entity set or relationship set of which M is an attribute.
- Consider the Instructor entity set as shown where:
 - Primary key: ID
 - Multivalued attribute: Phone_no
- primary key of the new relation schema **R** consists of all attributes of the schema.
- the relational schema:

Instructor(ID, name, Salary)
R : Instructor_phone(ID, Phone_no)

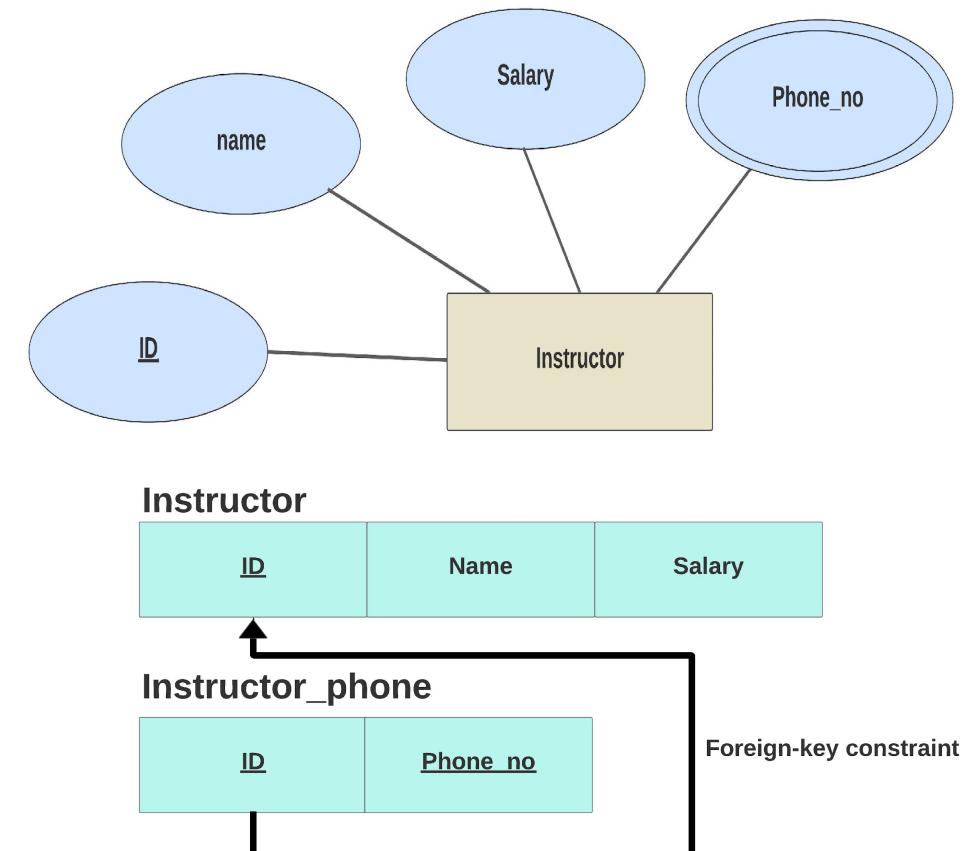


MULTIVALUED ATTRIBUTES

- A **foreign-key constraint** is to be added to the relation schema created from the multivalued attribute.
- In that newly created schema, the attribute generated from the primary key of the entity set must reference the relation generated from the entity set.
- In the above example, the foreign-key constraint on the instructor phone relation would be that attribute ID references the instructor relation
- Schema:

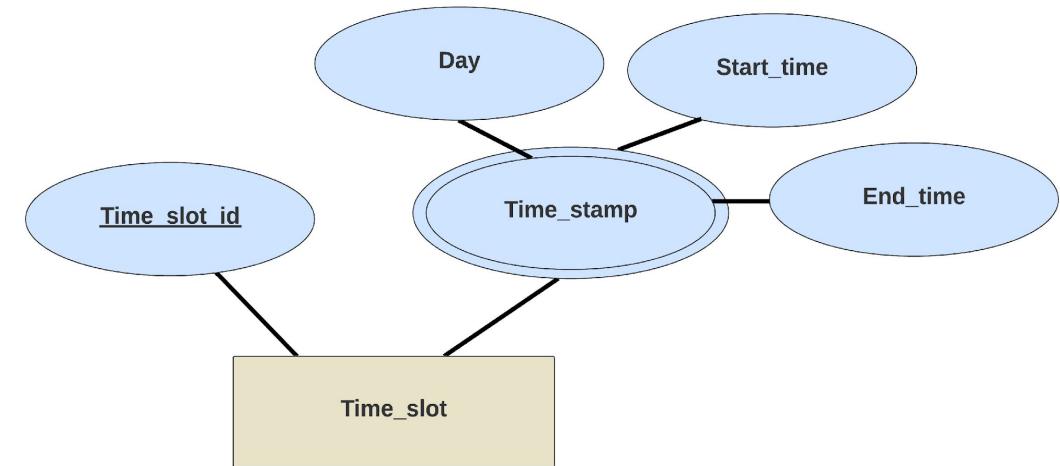
Instructor(ID, name, Salary)

R : Instructor_phone(ID, Phone_no)



MULTIVALUED ATTRIBUTES

- In the case: entity set consists of only two attributes
 - A single primary-key attribute B
 - A single multivalued attribute M
- The relation schema for the entity set would contain only one attribute, namely, the primary-key attribute B.
- In such cases we drop this relation, while retaining the relation schema with the attribute B and attribute A that corresponds to M.
- In the time_slot entity set shown,
 - B: Time_slot_id
 - M: Time_stamp

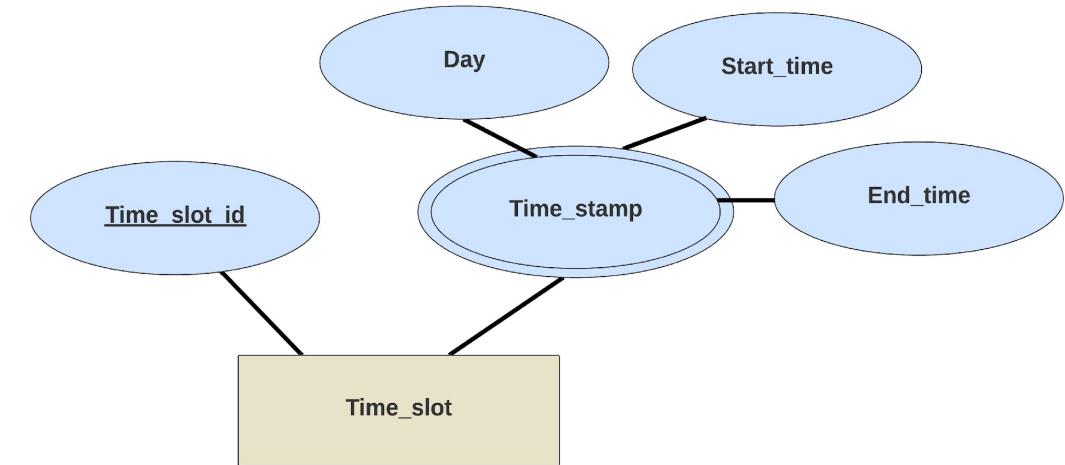


Schema:

Time_slot(Time_slot_id, Day, Start_time, End_time)

MULTIVALUED ATTRIBUTES

- In the schema shown, the End_time is not included as a part of the primary key
- This is because there can not be two meetings of a class that start at the same time on the same day of the week but end at different times;
- Due to this constraint, the end_time has been omitted from the primary key of the time slot schema.

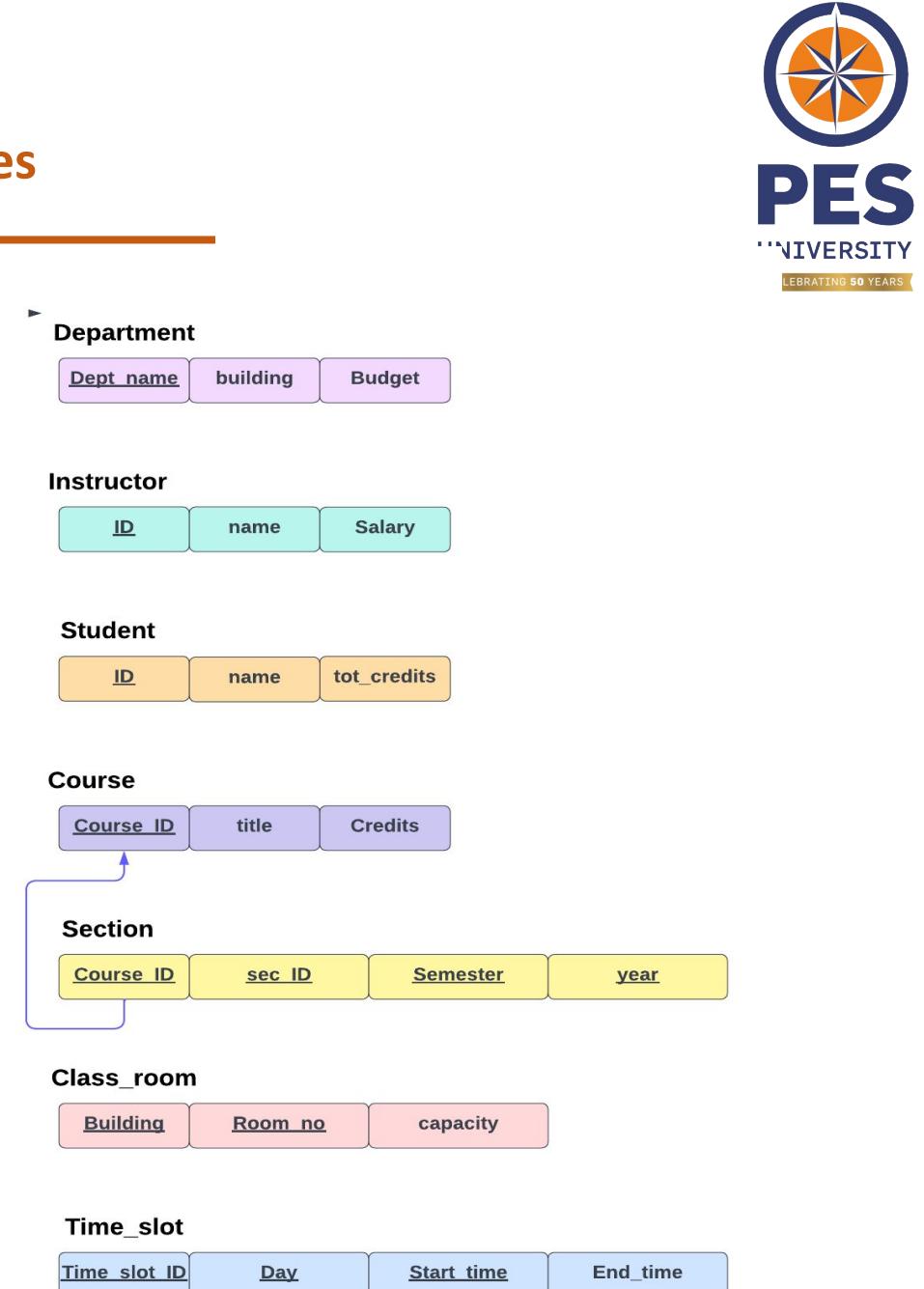


Schema:

Time_slot(Time_slot_id, Day, Start_time, End_time)

Representation of Strong Entity Sets with Complex Attributes

- After incorporating the complex attributes the university database would look like

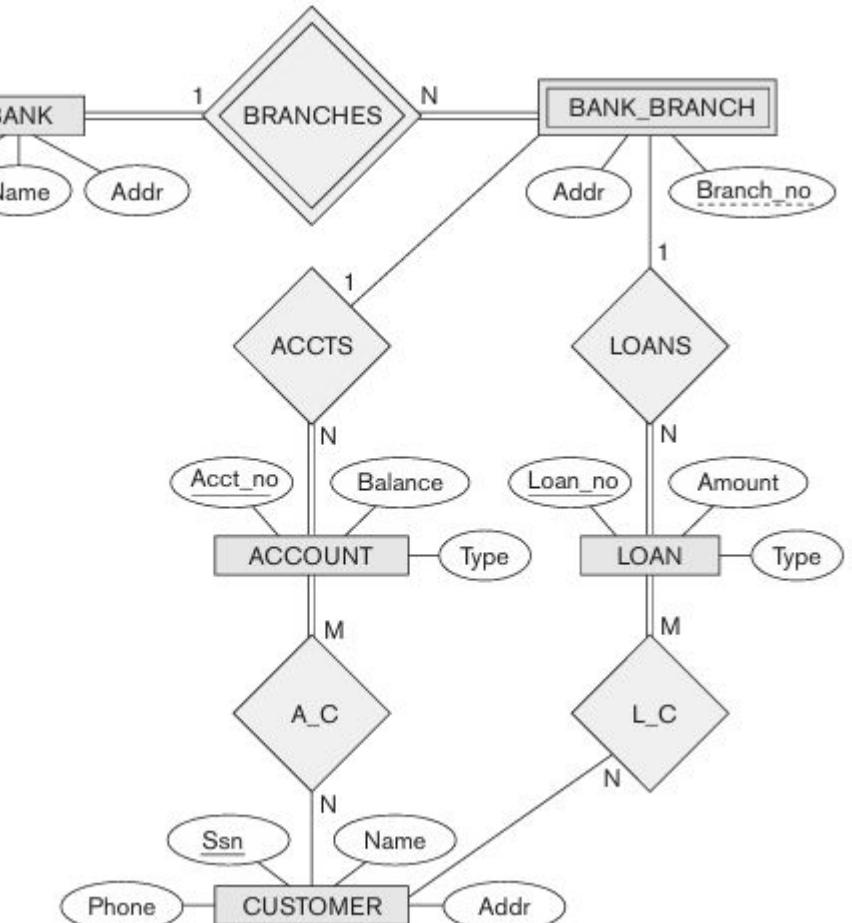


Question

Consider the ER diagram shown in the figure for part of a BANK database.

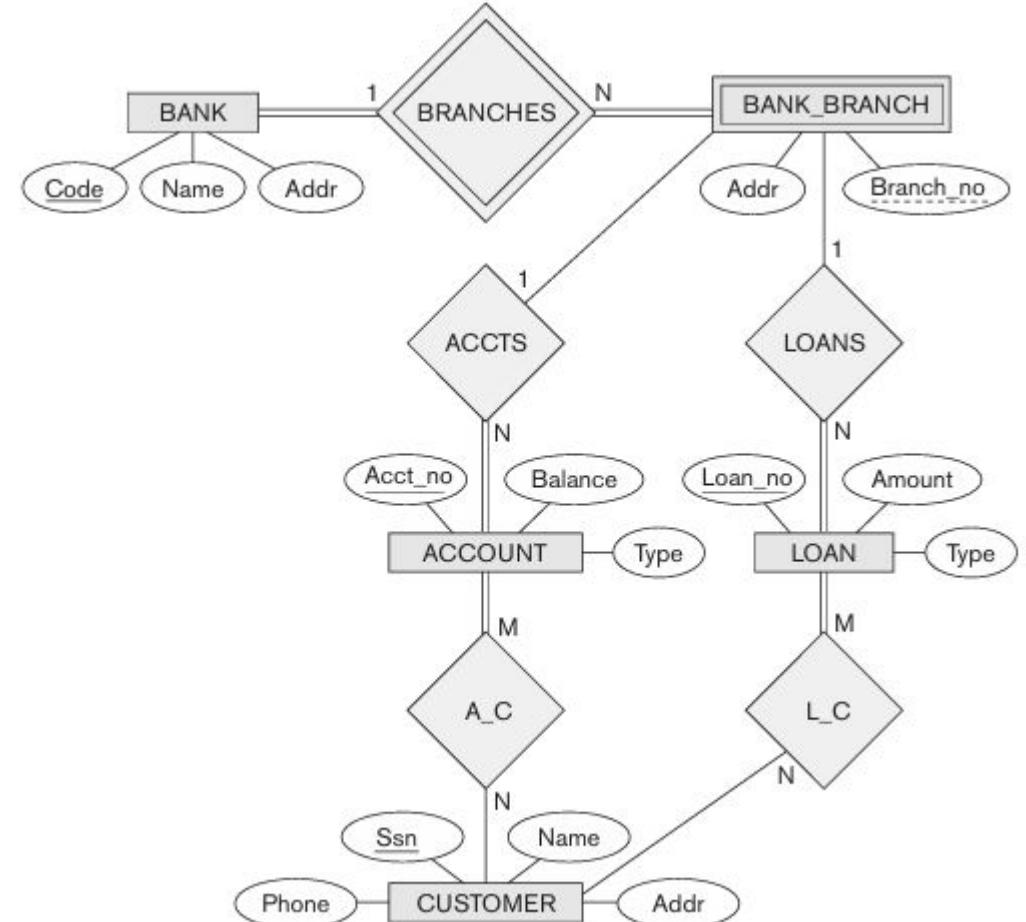
Each bank can have multiple branches, and each branch can have multiple accounts and loans.

- List the strong entity types.
- Is there a weak entity type? If so, give its name, partial key, and identifying relationship.
- What constraints do the partial key and the identifying relationship of the weak entity type specify in this diagram?
•



Question

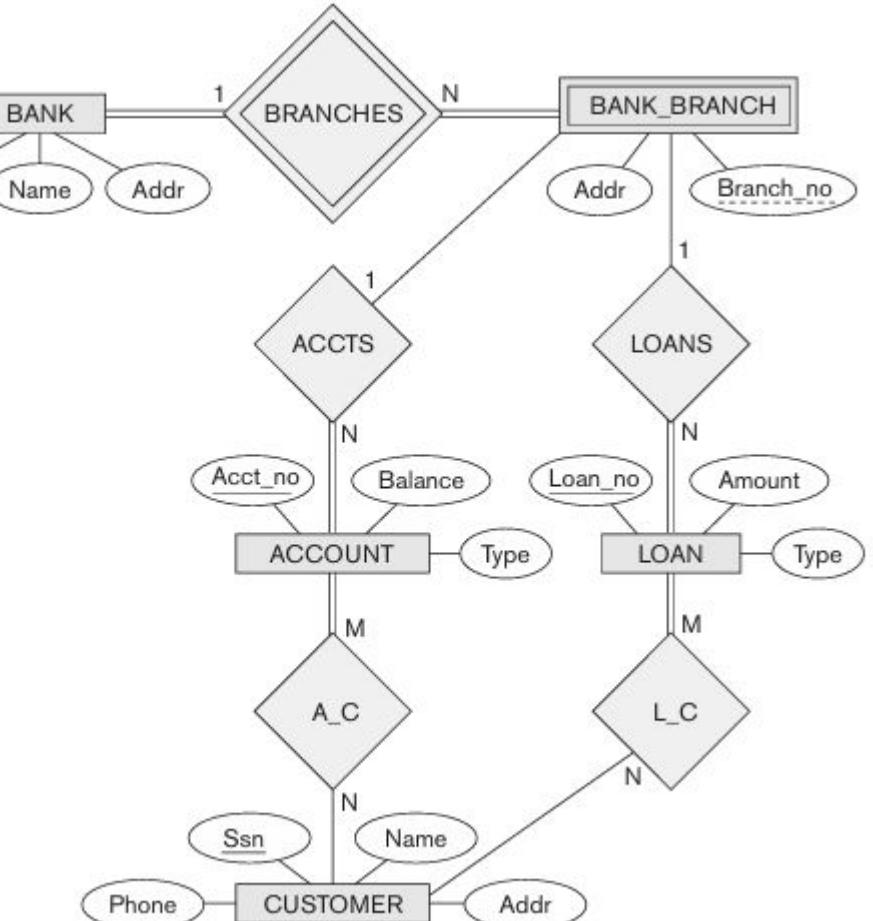
- List the strong entity types.
LOAN, BANK, ACCOUNT and CUSTOMER
- Is there a weak entity type? If so, give its name,partial key, and identifying relationship.
Yes, there is a weak entity BANK_BRANCH and its identifying relationship is BRANCHES.



- What constraints do the partial key and the identifying relationship of the weak entity type specify in this diagram?

The constraints are:

1. No two branches have the same number
2. A bank can have any number of branches but a branch only belongs to one bank.





PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu



PES
UNIVERSITY

CELEBRATING 50 YEARS

Database Management Systems

Conversion of E-R Diagram to Relational Schema

S Nagasundari

Department of Computer Science Engineering

Database Management Systems

Unit 1: Introduction to Database Management

Slides adapted from Author Slides of “Database System Concepts”, Silberschatz, H Korth and S Sudarshan, McGrawHill, 7th Edition, 2019. And Author Slides of Fundamentals of Database Systems”, Ramez Elamsri, Shamkant B Navathe, Pearson, 7th Edition, 2017.

Dr. Nagasundari S

Department of Computer Science and Engineering

Mapping of Binary 1:1 Relation Types

- For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R.
- There are three possible approaches:
 - **Foreign Key (2 relations) approach:**
 - **Merged relation (1 relation) option:**
 - **Cross-reference or relationship relation (3 relations) option:**

- There are three possible approaches:

- **Foreign Key (2 relations) approach:**

- Choose one of the relations-say S and include a foreign key in S the primary key of T.
 - It is better to choose an entity type with total participation in R in the role of S. And add all the descriptive attributes of the relation

- **Merged relation (1 relation) option:**

- This alternative, we merge the two entity types and the relationship into a single relation.
 - This may be appropriate when both participations are total. And add all the descriptive attributes of the relation

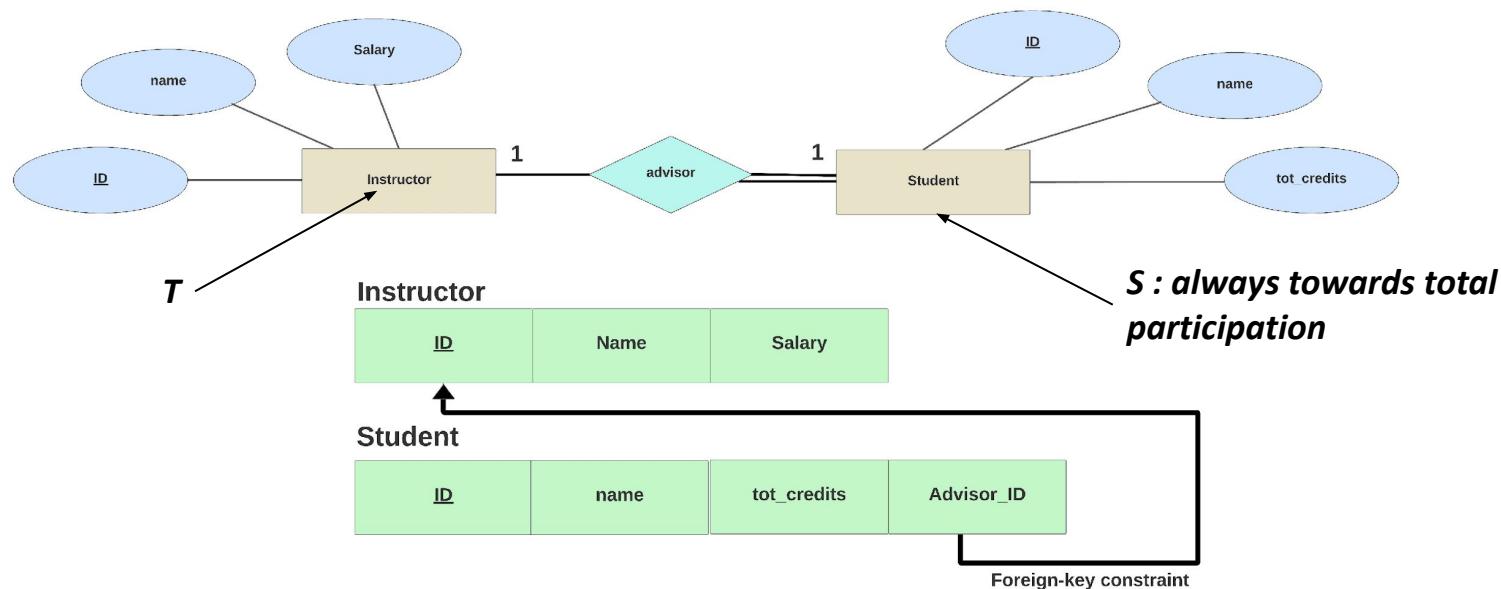
- **Cross-reference or relationship relation (3 relations) option:**

- The third alternative is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types. And add all the descriptive attributes of the relation

Database Management Systems

Representation of Relationship Sets

- Consider for this situation, the advisor relation to be one-to-one or 1:1 (Foreign key approach)



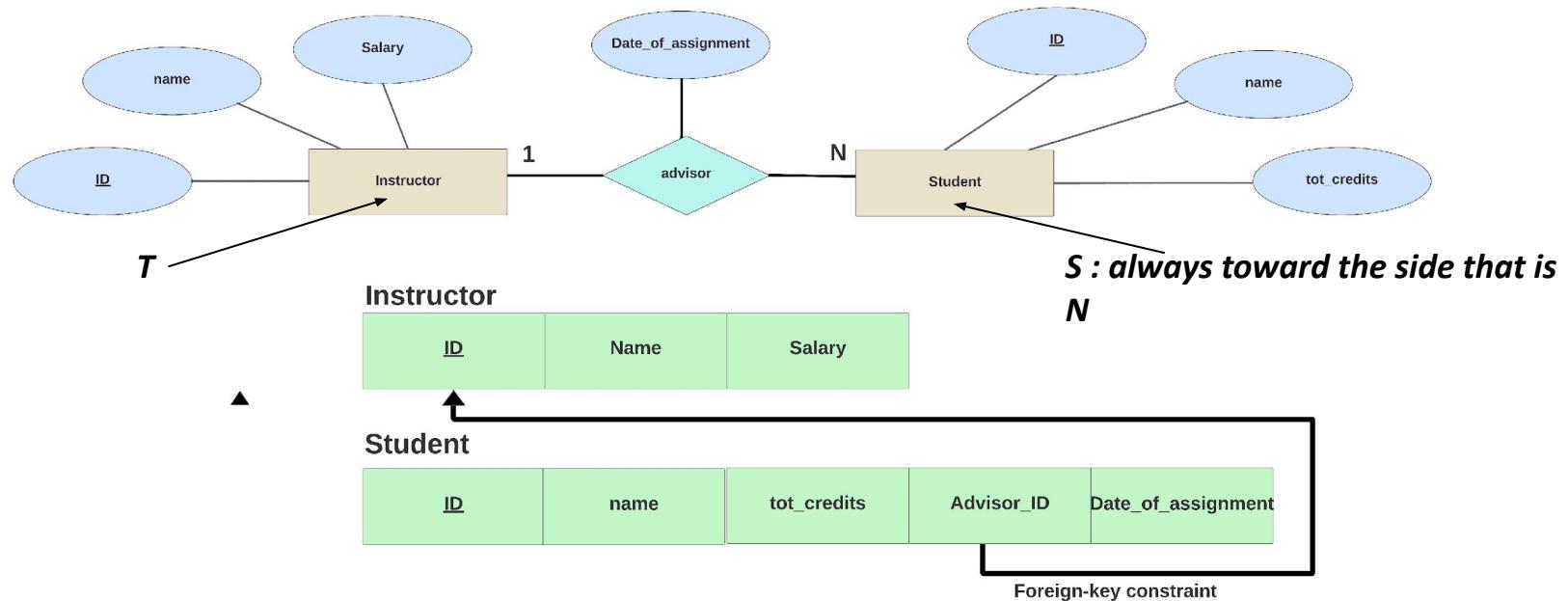
Mapping of Binary 1:N Relation Types

- For each regular binary 1:N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type.
- Include as the foreign key in S the primary key of the relation T that represents the other entity type participating in R.
- Include any simple attributes of the 1:N relation type as attributes of S.

Database Management Systems

Representation of Relationship Sets

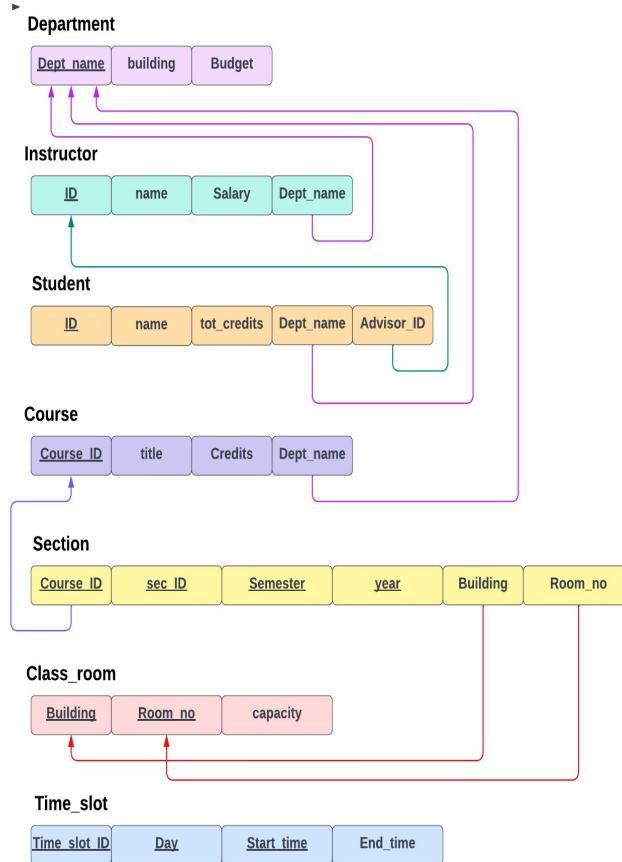
- Consider for this situation, the advisor relation to be one-to-many or 1:N



Database Management Systems

Representation of Relationship Sets

- After incorporating the 1:1 and 1:N relations in the university database would look as shown



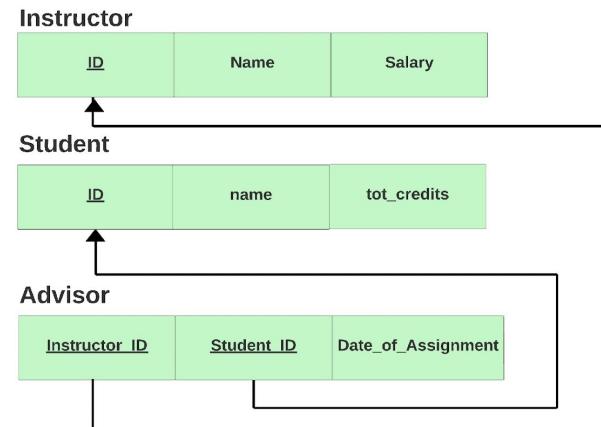
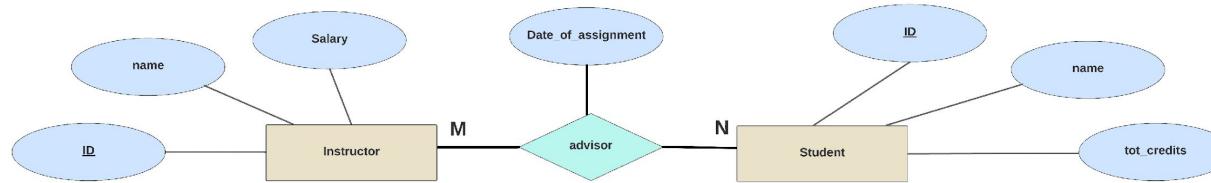
Mapping of Binary M:N Relation Types

- For each regular binary M:N relationship type R, create a new relation S to represent R. This is a relationship relation.
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S.
- Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.

Database Management Systems

Representation of Relationship Sets

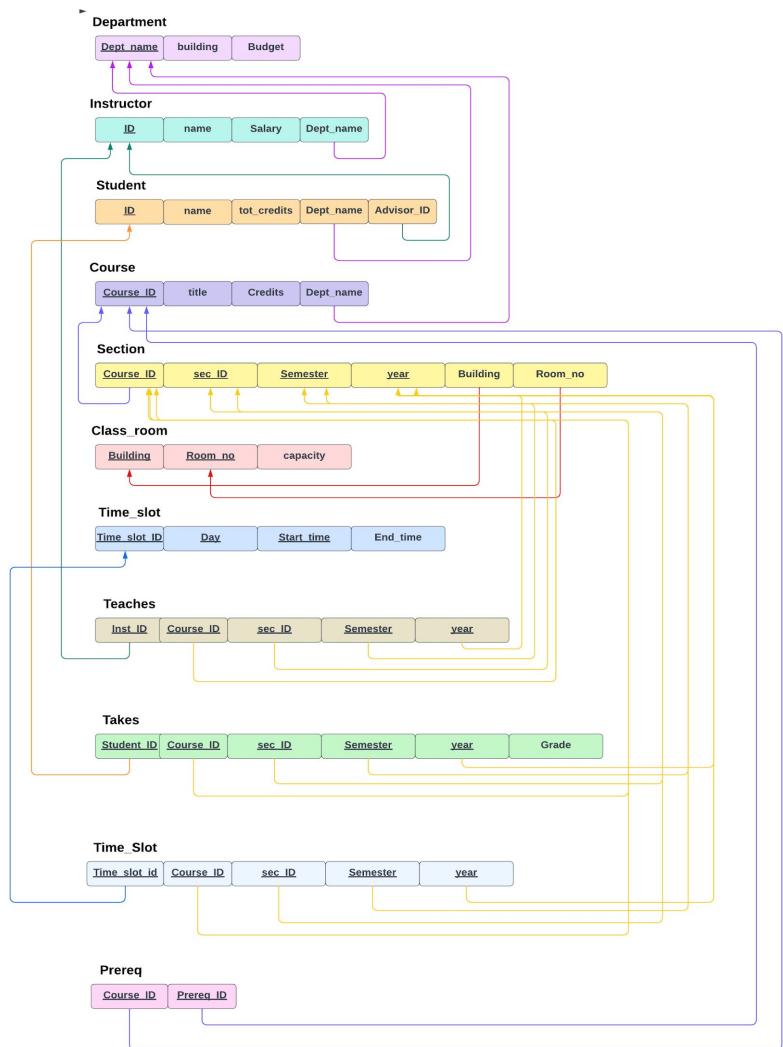
Consider for this situation, the advisor relation to be many-to-many or M:N



Database Management Systems

Representation of Relationship Sets

- After incorporating the M:N relations in the university database would look as shown

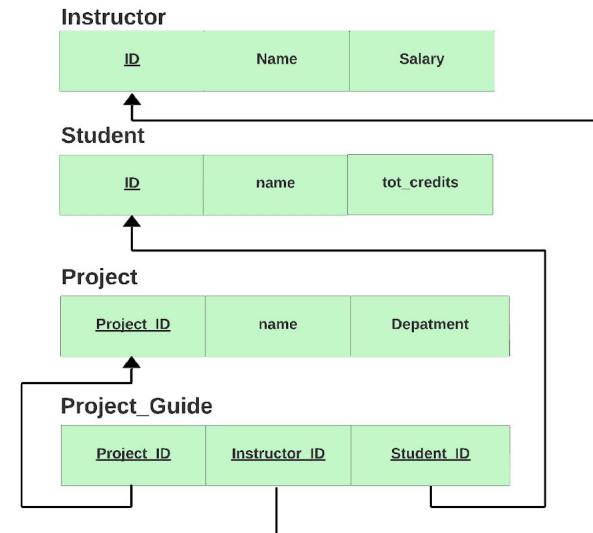
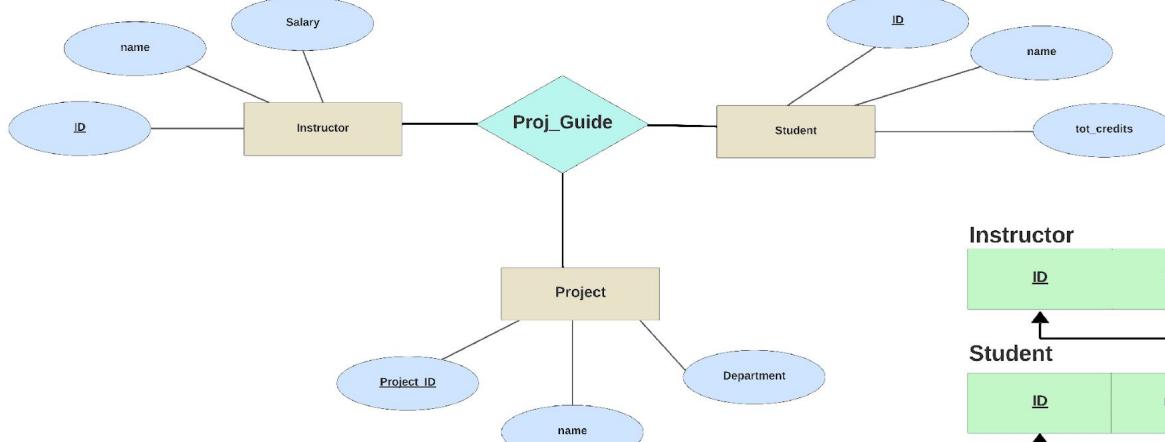


Mapping n-ary relationship sets

- For each n-ary relationship type R, where $n > 2$, create a new relationship S to represent R.
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
- Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S.
- Considering the project_guide ternary relationship discussed earlier, the ER and the corresponding Schema is given in the following slide

Database Management Systems

Representation of Relationship Sets





PES
UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

S Nagasundari

Department of Computer Science and Engineering

nagasundaris@pes.edu