

1902AS303 -Database Management Systems

Module I - Introduction to Database system

Introduction to Database Systems

A Database Management System (DBMS) is a software system that is designed to manage and organize data in a structured manner. It allows users to create, modify, and query a database, as well as manage the security and access controls for that database.

- Collection of interrelated data
- Set of programs to access the data
- DBMS contains information about a particular enterprise
- DBMS provides an environment that is both convenient and efficient to use

Key features of a DBMS include:

1. Data modeling: A DBMS provides tools for creating and modifying data models, which define the structure and relationships of the data in a database.
2. Data storage and retrieval: A DBMS is responsible for storing and retrieving data from the database, and can provide various methods for searching and querying the data.
3. Concurrency control: A DBMS provides mechanisms for controlling concurrent access to the database, to ensure that multiple users can access the data without conflicting with each other.
4. Data integrity and security: A DBMS provides tools for enforcing data integrity and security constraints, such as constraints on the values of data and access controls that restrict who can access the data.
5. Backup and recovery: A DBMS provides mechanisms for backing up and recovering the data in the event of a system failure.

Paradigm Shift from File System to DBMS

File System manages data using files on a hard disk. Users are allowed to create, delete, and update the files according to their requirements. Let us consider the example of file-based University Management System. Data of students is available to their respective Departments, Academics Section, Result Section, Accounts Section, Hostel Office, etc. Some of the data is common for all sections like Roll No, Name, Father Name, Address, and Phone number of students but some data is available to a particular section only like Hostel allotment number which is a part of the hostel office. Let us discuss the issues with this system:

- **Redundancy of data:** Data is said to be redundant if the same data is copied at many places. If a student wants to change their Phone number, he or she has to get it updated in various sections. Similarly, old records must be deleted from all sections representing that student.
- **Inconsistency of Data:** Data is said to be inconsistent if multiple copies of the same data do not match each other. If the Phone number is different in Accounts Section and Academics Section, it will be inconsistent. Inconsistency may be because of typing errors or not updating all copies of the same data.
- **Difficult Data Access:** A user should know the exact location of the file to access data, so the process is very cumbersome and tedious. If the user wants to search the student hostel allotment number of a student from 10000 unsorted students' records, how difficult it can be.
- **Unauthorized Access:** File Systems may lead to unauthorized access to data. If a student gets access to a file having his marks, he can change it in an unauthorized way.
- **No Concurrent Access:** The access of the same data by multiple users at the same time is known as concurrency. The file system does not allow concurrency as data can be accessed by only one user at a time.
- **No Backup and Recovery:** The file system does not incorporate any backup and recovery of data if a file is lost or corrupted.

Disadvantages of DBMS

- **Cost of Hardware and Software:** It requires a high speed of data processor and large memory size to run DBMS software.
- **Size:** It occupies a large space of disks and large memory to run them efficiently.
- **Complexity:** Database system creates additional complexity and requirements.
- **Higher impact of failure:** Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

DATABASE APPLICATIONS

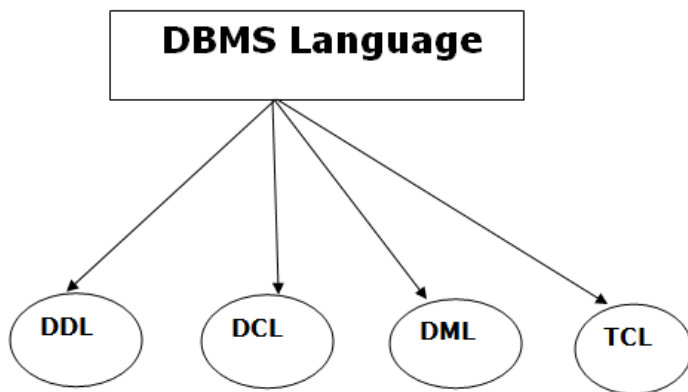
- Banking: all transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Online retailers: order tracking, customized recommendations
- Manufacturing: production, inventory, orders, supply chain
- Human resources: employee records, salaries, tax deductions

Instances and Schemas

- Similar to types and variables in programming languages
- **Schema** – the logical structure of the database
 - Example: The database consists of information about a set of customers and accounts and the relationship between them)
 - Analogous to type information of a variable in a program
 - **Physical schema:** database design at the physical level

- **Logical schema:** database design at the logical level
- **Instance** – the actual content of the database at a particular point in time
 - Analogous to the value of a variable
- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
 - Applications depend on the logical schema
 - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

Types of Database Language



1. Data Definition Language

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

Here are some tasks that come under DDL:

Create: It is used to create objects in the database.

Alter: It is used to alter the structure of the database.

Drop: It is used to delete objects from the database.

Truncate: It is used to remove all records from a table.

Rename: It is used to rename an object.

Comment: It is used to comment on the data dictionary.

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

2. Data Manipulation Language

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

Here are some tasks that come under DML:

Select: It is used to retrieve data from a database.

Insert: It is used to insert data into a table.

Update: It is used to update existing data within a table.

Delete: It is used to delete all records from a table.

Merge: It performs UPSERT operation, i.e., insert or update operations.

Call: It is used to call a structured query language or a Java subprogram.

Lock Table: It controls concurrency.

3. Data Control Language

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

(But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

Grant: It is used to give user access privileges to a database.

Revoke: It is used to take back permissions from the user.

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

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

4. Transaction Control Language

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

Here are some tasks that come under TCL:

Commit: It is used to save the transaction on the database.

Rollback: It is used to restore the database to original since the last Commit.

History of Database Systems

1950s and early 1960s:

- Data processing using magnetic tapes for storage
 - Tapes provided only sequential access
- Punched cards for input

Late 1960s and 1970s:

- Hard disks allowed direct access to data
- Network and hierarchical data models in widespread use
- Ted Codd defines the relational data model
 - Would win the ACM Turing Award for this work
 - IBM Research begins System R prototype
 - UC Berkeley begins Ingres prototype
- High-performance (for the era) transaction processing

1980s:

- Research relational prototypes evolve into commercial systems
 - SQL becomes industrial standard
- Parallel and distributed database systems
- Object-oriented database systems

1990s:

- Large decision support and data-mining applications
- Large multi-terabyte data warehouses
- Emergence of Web commerce

Early 2000s:

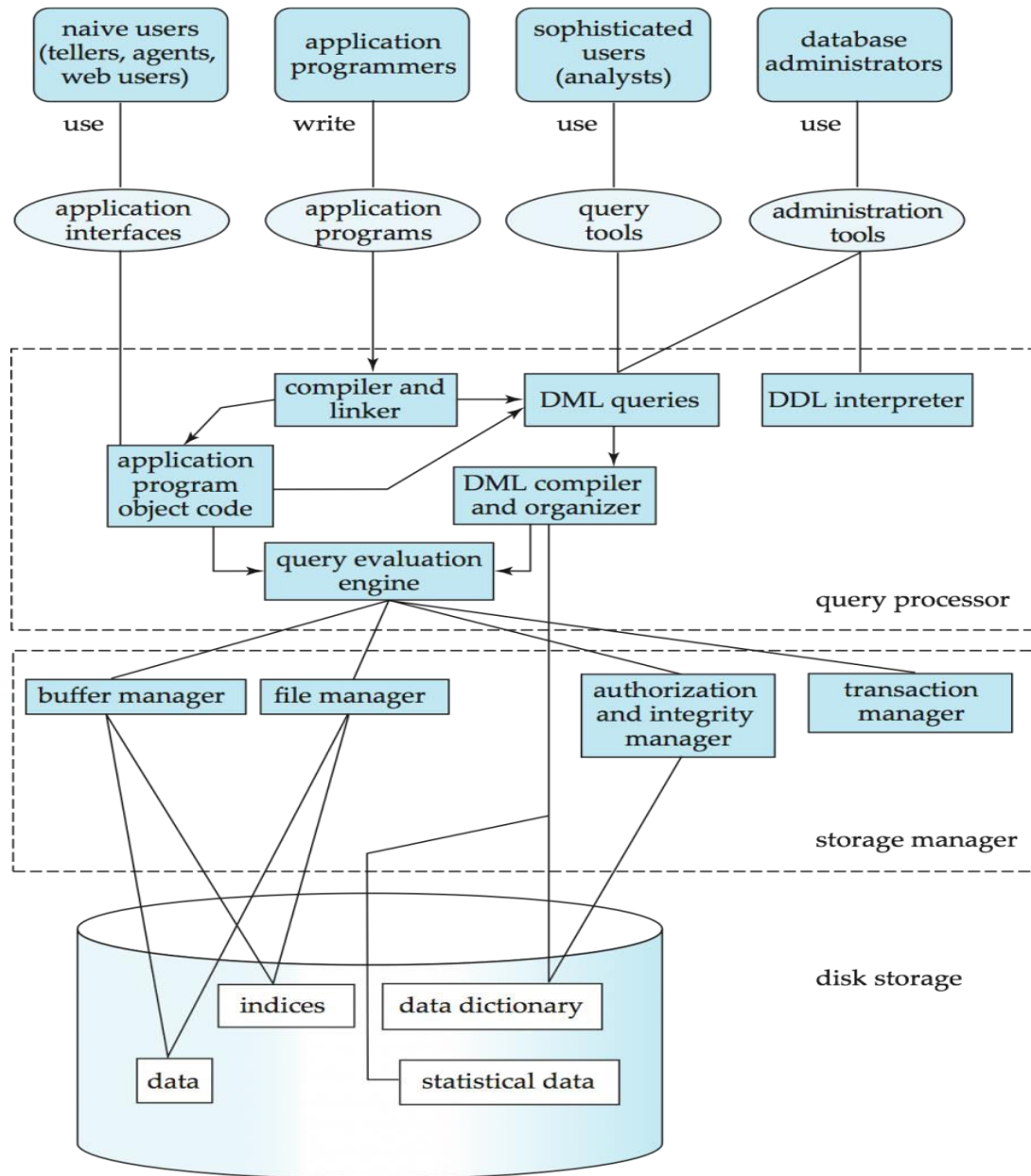
- XML and XQuery standards

Automated database administration

Later 2000s:

- Giant data storage systems
 - Google BigTable, Yahoo PNuts, Amazon, ..

Database System Architecture



Database Users:

Users are differentiated by the way they expect to interact with the system:

- **Application programmers:**
 - Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces.

- Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports without writing a program.
- **Sophisticated users:**
 - Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language.
 - They submit each such query to a query processor, whose function is to break down DML statements into instructions that the storage manager understands.
- **Specialized users :**
 - Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.
 - Among these applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.
- **Naïve users :**
 - Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.
 - For example, a bank teller who needs to transfer \$50 from account A to account B invokes a program called transfer. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred

Database Administrator:

- Coordinates all the activities of the database system. The database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties include:
 - **Schema definition:** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
 - **Storage structure and access method definition.**
 - **Schema and physical organization modification:** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.
 - **Granting user authority to access the database:** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access.
 - **Specifying integrity constraints.**
 - **Monitoring performance and responding to changes in requirements.**

Query Processor:

The query processor will accept query from user and solves it by accessing the database.

Parts of Query processor:

- **DDL interpreter**

This will interprets DDL statements and fetch the definitions in the data dictionary.

- **DML compiler**

a. This will translates DML statements in a query language into low level instructions that the query evaluation engine understands.

b. A query can usually be translated into any of a number of alternative evaluation plans for same query result DML compiler will select best plan for query optimization.

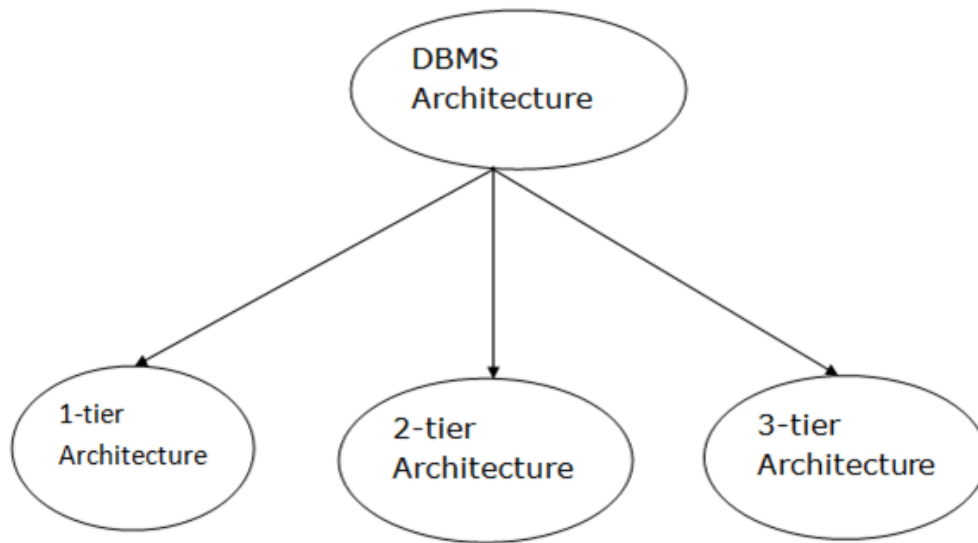
- **Query evaluation engine**

This engine will execute low-level instructions generated by the DML compiler on DBMS.

Storage Manager/Storage Management:

- A storage manager is a program module which acts like interface between the data stored in a database and the application programs and queries submitted to the system.
- Thus, the storage manager is responsible for storing, retrieving and updating data in the database.
- **The storage manager components include:**
 - **Authorization and integrity manager:** Checks for integrity constraints and authority of users to access data.
 - **Transaction manager:** Ensures that the database remains in a consistent state although there are system failures.
 - **File manager:** Manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
 - **Buffer manager:** It is responsible for retrieving data from disk storage into main memory. It enables the database to handle data sizes that are much larger than the size of main memory.
 - **Data structures implemented by storage manager.**
 - **Data files:** Stored in the database itself.
 - **Data dictionary:** Stores metadata about the structure of the database.
 - **Indices:** Provide fast access to data items.

Types of DBMS Architecture



Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

1-Tier Architecture

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

2-Tier Architecture

- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
- The user interfaces and application programs are run on the client-side.

- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.

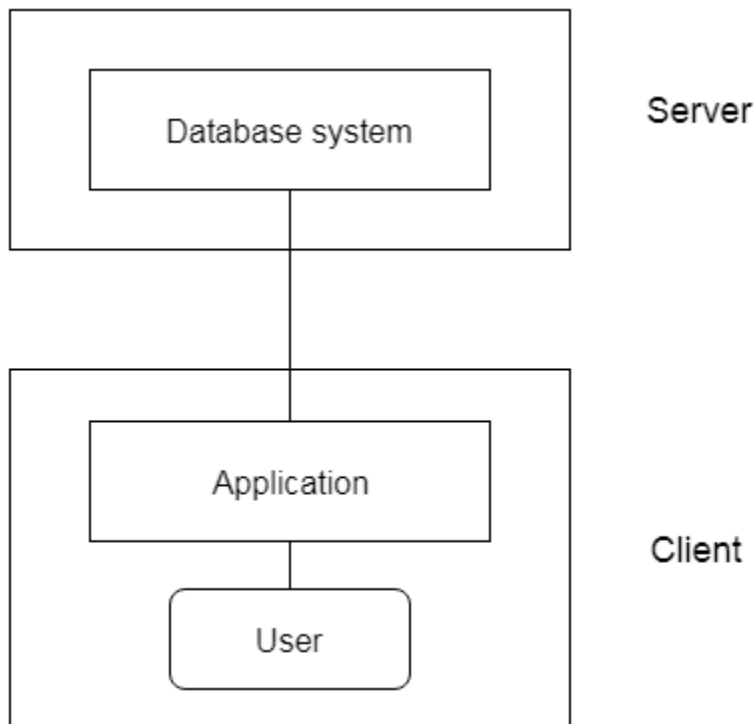


Fig: 2-tier Architecture

3-Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.

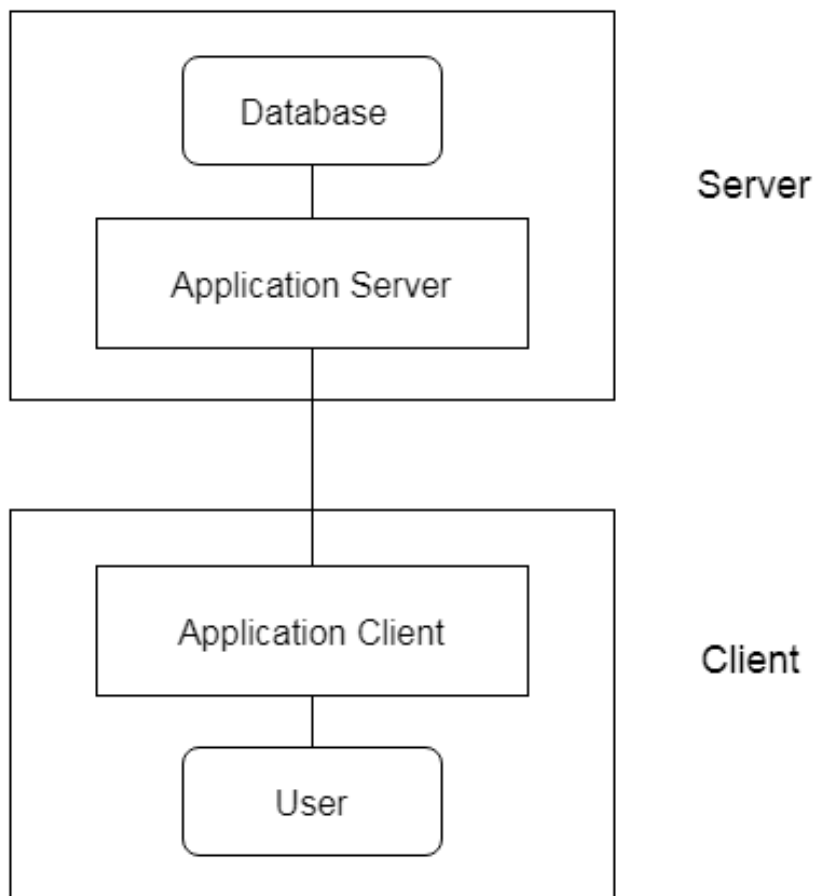


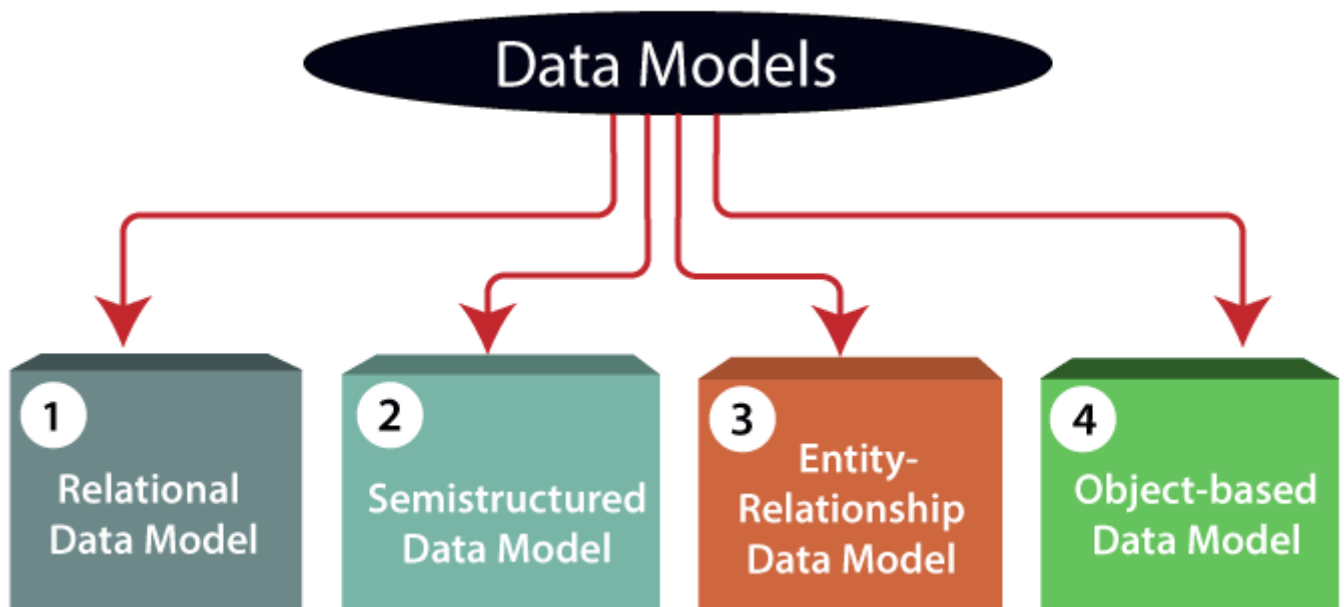
Fig: 3-tier Architecture

Data Models

- Data Model is a collection of concept that can be used to describe structure of the database.
- A collection of tools for describing
 - Data
 - Data relationships
 - Data semantics
 - Data constraints
- Relational model

- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Other older models:
 - Network model
 - Hierarchical model

Data Model is the modeling of the data description, data semantics, and consistency constraints of the data. It provides the conceptual tools for describing the design of a database at each level of data abstraction. Therefore, there are following four data models used for understanding the structure of the database:



1) Relational Data Model: This type of model designs the data in the form of rows and columns within a table. Thus, a relational model uses tables for representing data and in-between relationships. Tables are also called relations. This model was initially described by Edgar F. Codd, in 1969. The relational data model is the widely used model which is primarily used by commercial data processing applications.

2) Entity-Relationship Data Model: An ER model is the logical representation of data as objects and relationships among them. These objects are known as entities, and relationship is an association among these entities. This model was designed by Peter Chen and published in 1976 papers. It was widely used in database designing. A set of attributes describe the entities. For example, student_name, student_id describes the 'student' entity. A set of the same type of entities is known as an 'Entity set', and the set of the same type of relationships is known as 'relationship set'.

3) Object-based Data Model: An extension of the ER model with notions of functions, encapsulation, and object identity, as well. This model supports a rich type system that includes structured and collection types. Thus, in 1980s, various database systems following the object-oriented approach were developed. Here, the objects are nothing but the data carrying its properties.

4) Semistructured Data Model: This type of data model is different from the other three data models (explained above). The semistructured data model allows the data specifications at places where the individual data items of the same type may have different attributes sets. The Extensible Markup Language, also known as XML, is widely used for representing the semistructured data. Although XML was initially designed for including the markup information to the text document, it gains importance because of its application in the exchange of data.

Aspect	Traditional File System	Database Management System (DBMS)
Data Redundancy	High – same data stored in multiple files	Low – data is centralized and normalized
Data Inconsistency	Common due to duplicate data	Reduced via constraints and controlled access
Data Access	Manual, program-specific	Query-based (SQL), flexible and efficient
Security / Access Control	Minimal – OS-level only	Role-based access, fine-grained control
Data Integrity	Hard to enforce	Enforced through constraints (PK, FK, CHECK)
Concurrent Access	Risky – may lead to corruption	Supports ACID transactions for consistency
Backup and Recovery	Manual and error-prone	Automated and robust mechanisms
Scalability	Limited	Highly scalable with indexing and partitioning
Data Relationships	Handled through custom code	Built-in relational model supports joins, keys
Example	File folders, spreadsheets	MySQL, PostgreSQL, Oracle, SQLite

EXAMPLE: University Student Records

Traditional File System:

- Each department (e.g., Academics, Accounts, Library) stores its own student records in separate Excel files.
- The same student's name, roll number, and contact details are duplicated across files.
- Updates in one file (e.g., name correction) are **not reflected** in others, leading to **data inconsistency**.
- Anyone with access to the file can edit all data—**no fine-grained access control**.
- Writing reports (e.g., total fees paid by CS students) requires **manual filtering and coding**.

DBMS Solution:

- All student data stored **once** in a centralized relational database.
- Each department accesses only the data it needs through **views and permissions**.
- If a student's phone number is updated, the change is **automatically reflected** across all modules.
- DBMS **enforces rules** like: roll numbers must be unique (Primary Key), only authorized staff can update marks (Access Control).
- Complex queries (e.g., list all students who paid fees and cleared library dues) can be done in **seconds using SQL**.
- Supports **backups** and can restore data after system failure.



Chapter 6: Database Design Using the E-R Model



Outline

- Overview of the Design Process
- The Entity-Relationship Model
- Complex Attributes
- Mapping Cardinalities
- Primary Key
- Extended E-R Features
- Entity-Relationship Design Issues
- Alternative Notations for Modeling Data (UML)
- Other Aspects of Database Design



Design Phases

- Initial phase -- characterize fully the data needs of the prospective database users.
- Second phase -- choosing a data model
 - Applying the concepts of the chosen data model
 - Translating these requirements into a conceptual schema of the database.
 - A fully developed conceptual schema indicates the functional requirements of the enterprise.
 - Describe the kinds of operations (or transactions) that will be performed on the data.



Design Phases (Cont.)

- Final Phase -- Moving from an abstract data model to the implementation of the database
 - Logical Design – Deciding on the database schema. Database design requires that we find a “good” collection of relation schemas.
 - Business decision – What attributes should we record in the database?
 - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
 - Physical Design – Deciding on the physical layout of the database



Design Alternatives

- In designing a database schema, we must ensure that we avoid two major pitfalls:
 - Redundancy: a bad design may result in repeat information.
 - Redundant representation of information may lead to data inconsistency among the various copies of information
 - Incompleteness: a bad design may make certain aspects of the enterprise difficult or impossible to model.
- Avoiding bad designs is not enough. There may be a large number of good designs from which we must choose.



Design Approaches

- Entity Relationship Model (covered in this chapter)
 - Models an enterprise as a collection of *entities* and *relationships*
 - Entity: a “thing” or “object” in the enterprise that is distinguishable from other objects
 - Described by a set of *attributes*
 - Relationship: an association among several entities
 - Represented diagrammatically by an *entity-relationship diagram*:
- Normalization Theory (Chapter 7)
 - Formalize what designs are bad, and test for them



Outline of the ER Model



ER model -- Database Modeling

- The ER data model was developed to facilitate database design by allowing specification of an **enterprise schema** that represents the overall logical structure of a database.
- The ER data model employs three basic concepts:
 - entity sets,
 - relationship sets,
 - attributes.
- The ER model also has an associated diagrammatic representation, the **ER diagram**, which can express the overall logical structure of a database graphically.



Entity Sets

- An **entity** is an object that exists and is distinguishable from other objects.
 - Example: specific person, company, event, plant
- An **entity set** is a set of entities of the same type that share the same properties.
 - Example: set of all persons, companies, trees, holidays
- An entity is represented by a set of attributes; i.e., descriptive properties possessed by all members of an entity set.
 - Example:
instructor = (ID, name, salary)
course = (course_id, title, credits)
- A subset of the attributes form a **primary key** of the entity set; i.e., uniquely identifying each member of the set.



Entity Sets -- *instructor* and *student*

76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

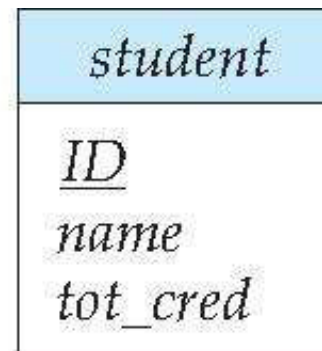
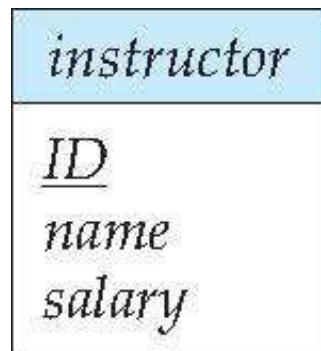
98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student



Representing Entity sets in ER Diagram

- Entity sets can be represented graphically as follows:
 - Rectangles represent entity sets.
 - Attributes listed inside entity rectangle
 - Underline indicates primary key attributes





Relationship Sets

- A **relationship** is an association among several entities

Example:

44553 (Peltier)	<u>advisor</u>	22222 (<u>Einstein</u>)
<i>student</i> entity	relationship set	<i>instructor</i> entity

- A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

$$\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$

where (e_1, e_2, \dots, e_n) is a relationship

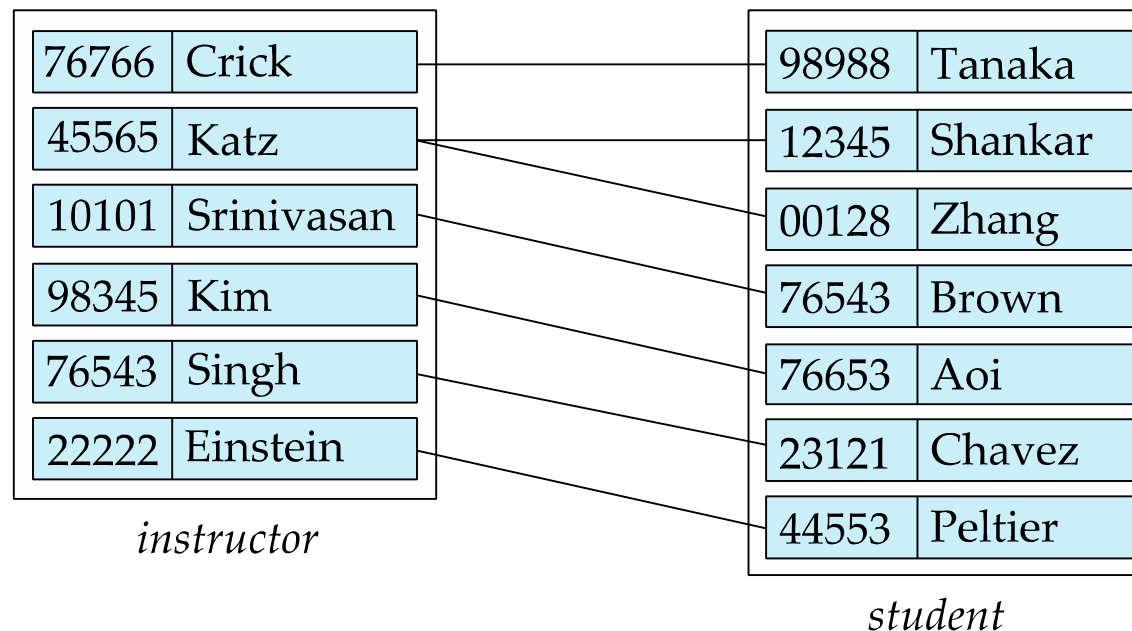
- Example:

$$(44553, 22222) \in \text{advisor}$$



Relationship Sets (Cont.)

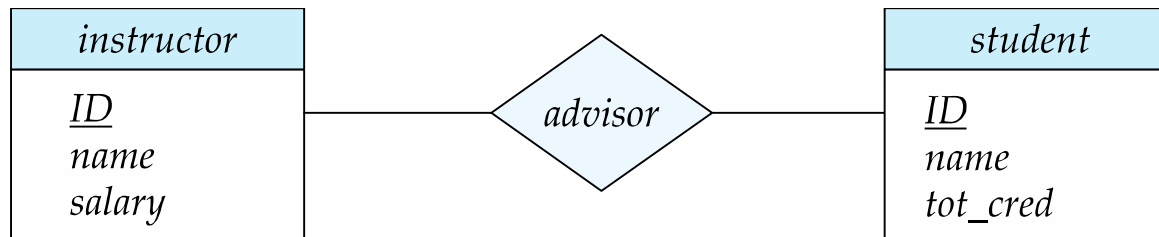
- Example: we define the relationship set *advisor* to denote the associations between students and the instructors who act as their advisors.
- Pictorially, we draw a line between related entities.





Representing Relationship Sets via ER Diagrams

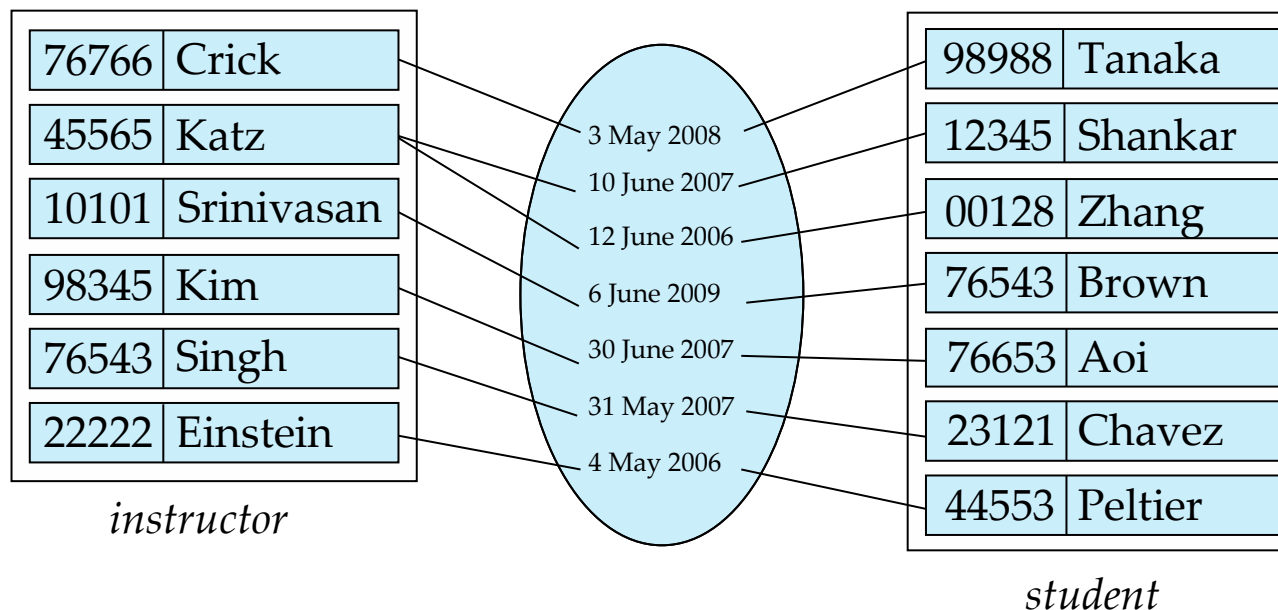
- Diamonds represent relationship sets.





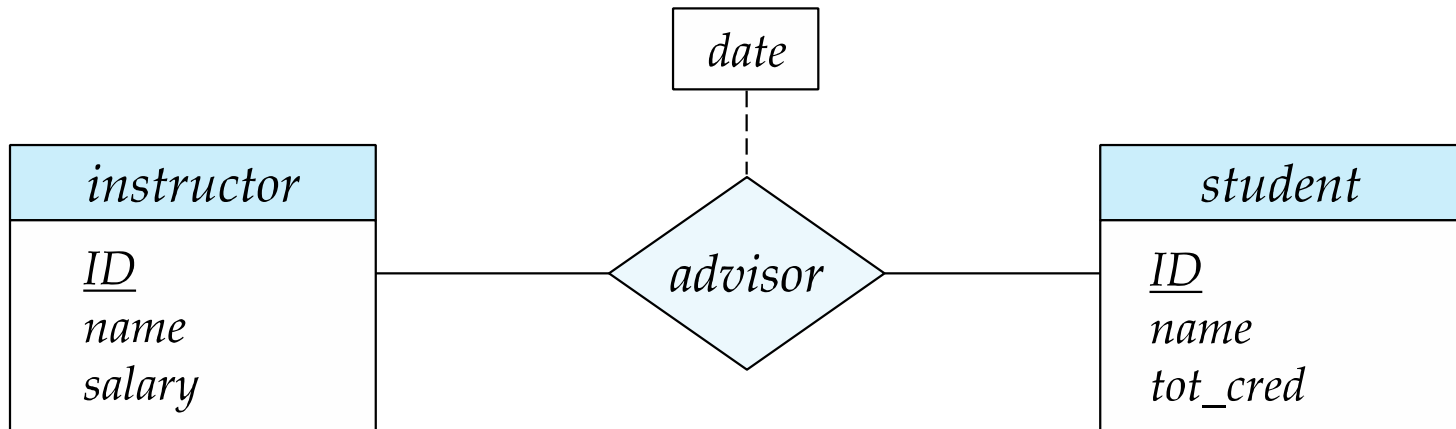
Relationship Sets (Cont.)

- An attribute can also be associated with a relationship set.
- For instance, the *advisor* relationship set between entity sets *instructor* and *student* may have the attribute *date* which tracks when the student started being associated with the advisor





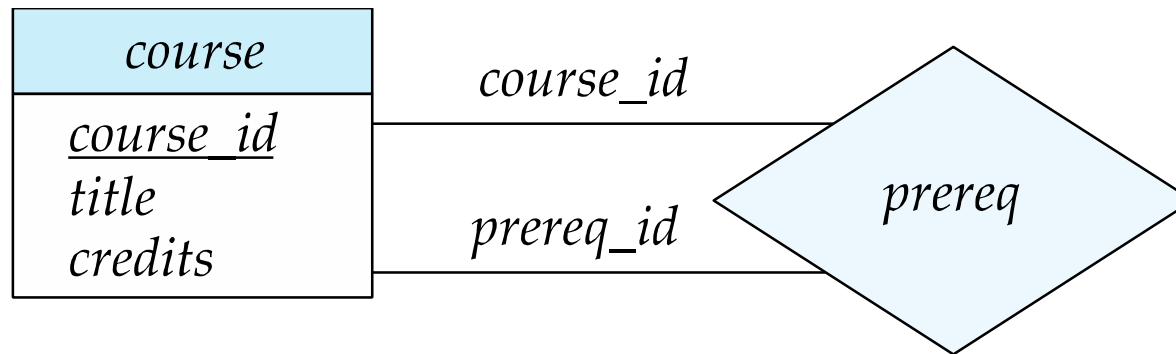
Relationship Sets with Attributes





Roles

- Entity sets of a relationship need not be distinct
 - Each occurrence of an entity set plays a “role” in the relationship
- The labels “*course_id*” and “*prereq_id*” are called **roles**.





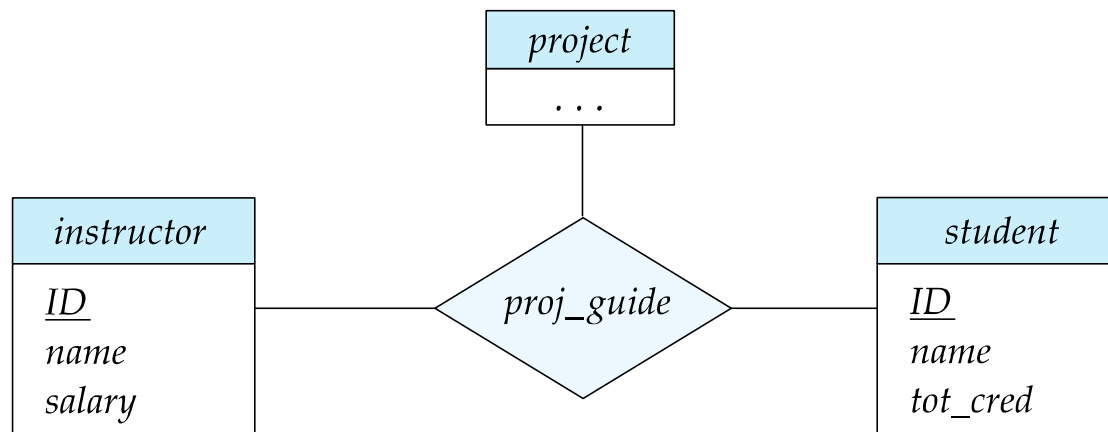
Degree of a Relationship Set

- Binary relationship
 - involve two entity sets (or degree two).
 - most relationship sets in a database system are binary.
- Relationships between more than two entity sets are rare. Most relationships are binary. (More on this later.)
 - Example: *students* work on research *projects* under the guidance of an *instructor*.
 - relationship *proj_guide* is a ternary relationship between *instructor*, *student*, and *project*



Non-binary Relationship Sets

- Most relationship sets are binary
- There are occasions when it is more convenient to represent relationships as non-binary.
- E-R Diagram with a Ternary Relationship





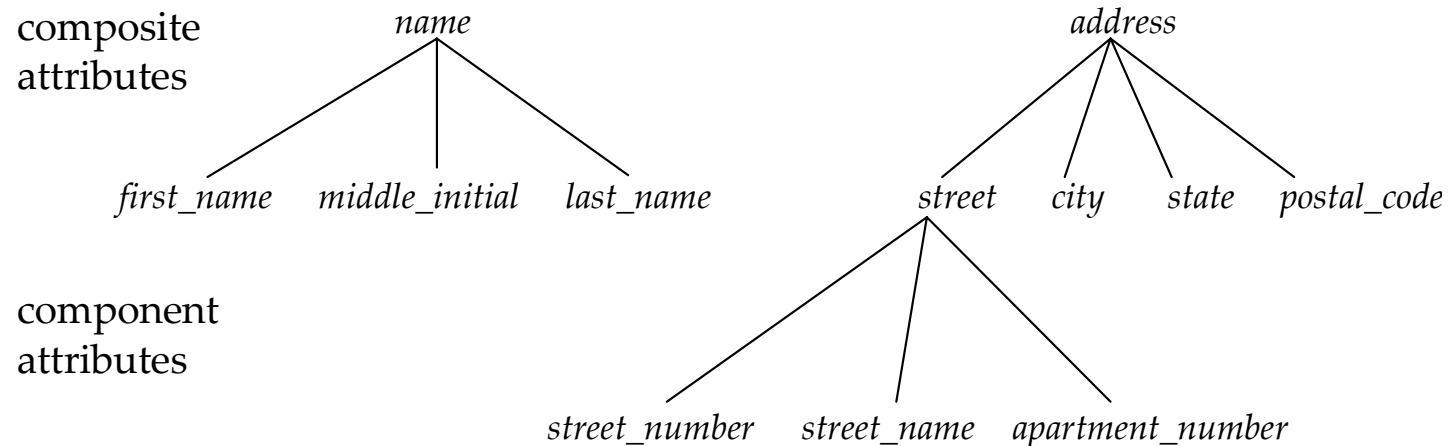
Complex Attributes

- Attribute types:
 - **Simple** and **composite** attributes.
 - **Single-valued** and **multivalued** attributes
 - Example: multivalued attribute: *phone_numbers*
 - **Derived** attributes
 - Can be computed from other attributes
 - Example: age, given date_of_birth
- **Domain** – the set of permitted values for each attribute



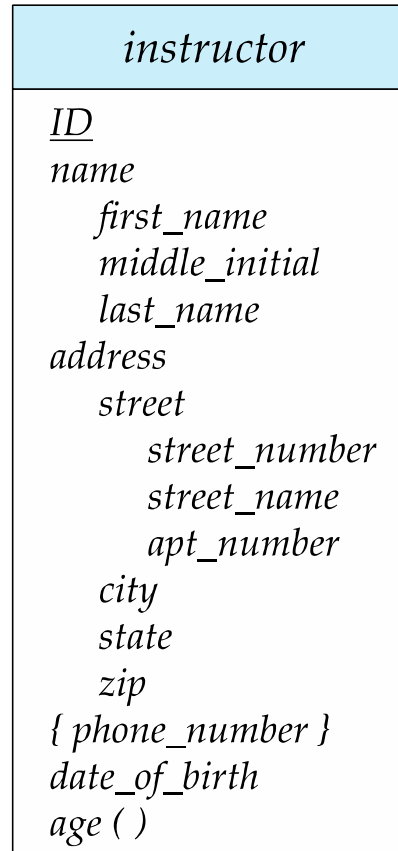
Composite Attributes

- Composite attributes allow us to divided attributes into subparts (other attributes).





Representing Complex Attributes in ER Diagram



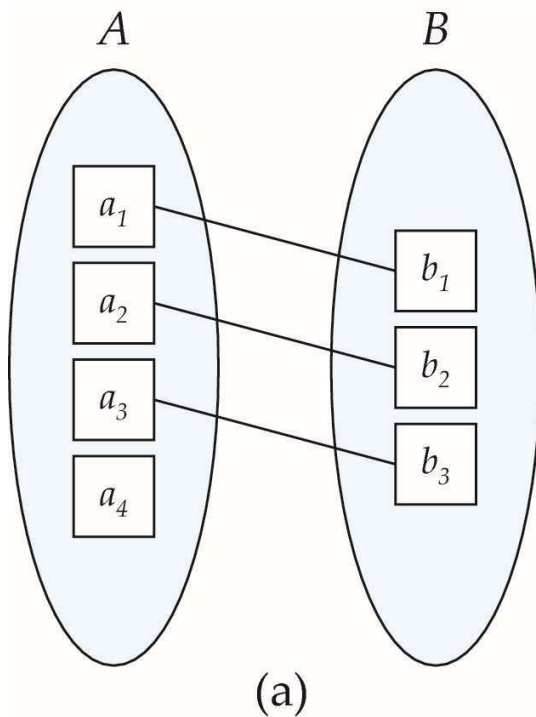


Mapping Cardinality Constraints

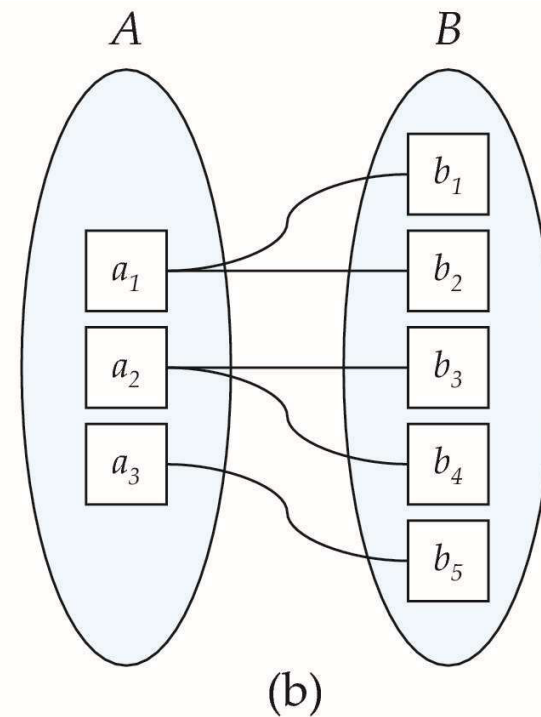
- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
 - One to one
 - One to many
 - Many to one
 - Many to many



Mapping Cardinalities



One to one

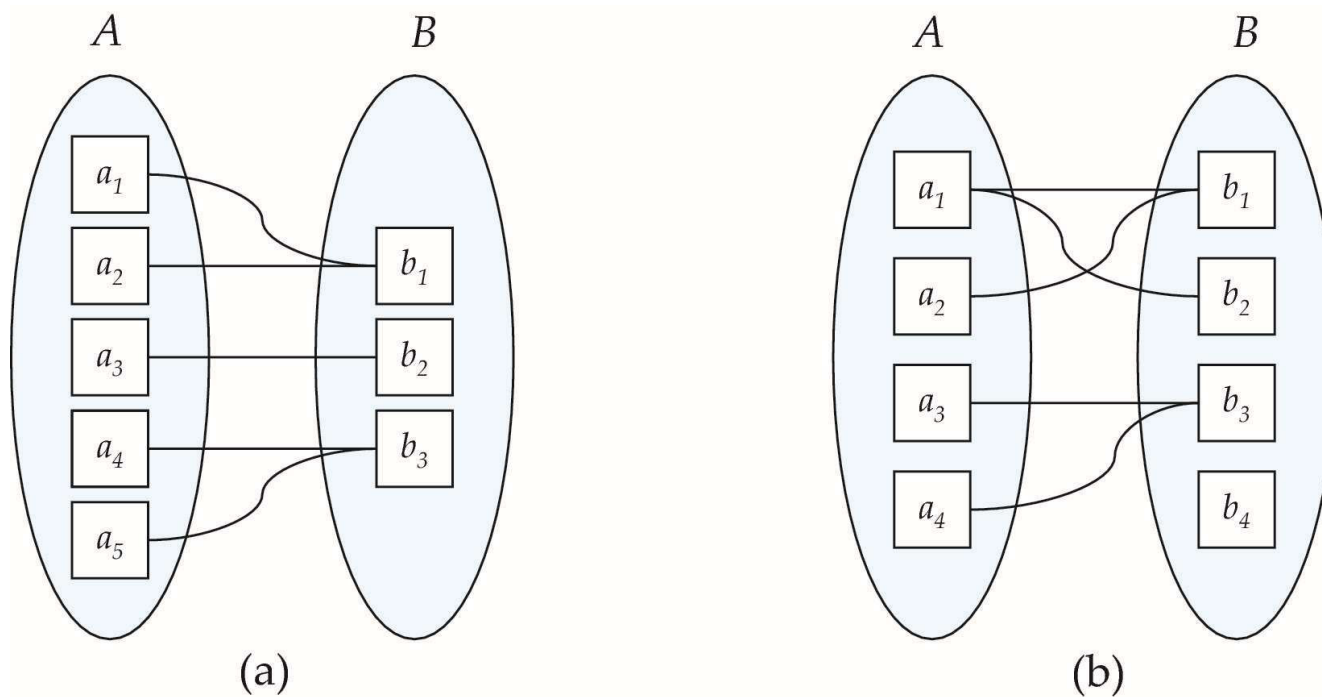


One to many

Note: Some elements in A and B may not be mapped to any elements in the other set



Mapping Cardinalities



Many to
one

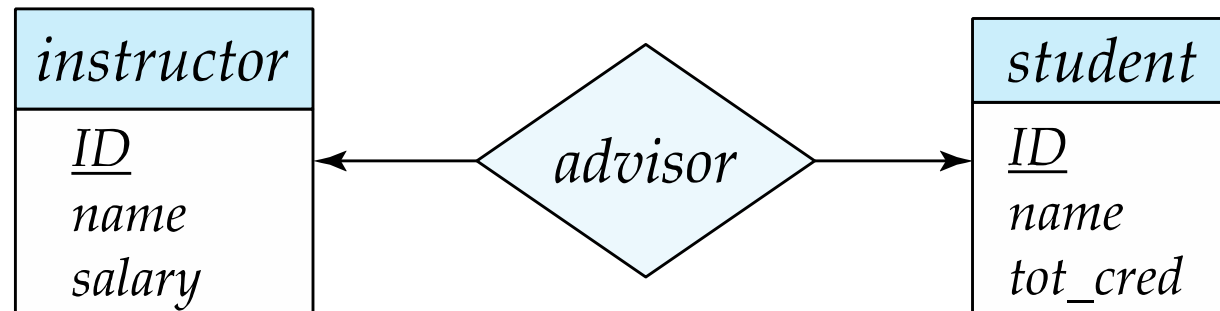
Many to many

Note: Some elements in A and B may not be mapped to any elements in the other set



Representing Cardinality Constraints in ER Diagram

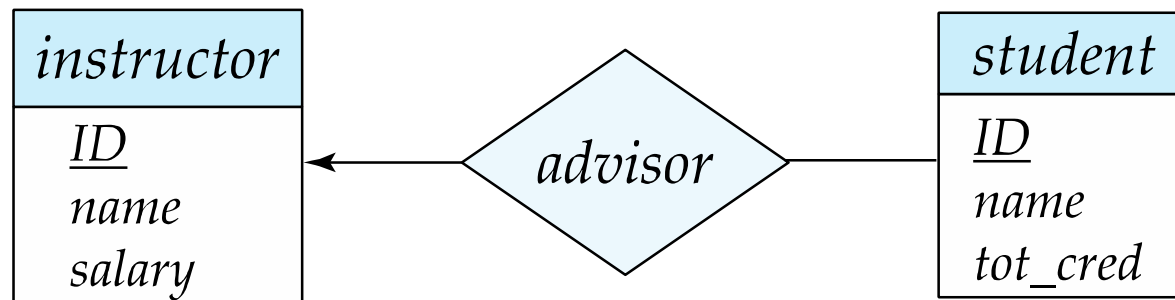
- We express cardinality constraints by drawing either a directed line (\rightarrow), signifying “one,” or an undirected line (—), signifying “many,” between the relationship set and the entity set.
- One-to-one relationship between an *instructor* and a *student* :
 - A student is associated with at most one *instructor* via the relationship *advisor*
 - A *student* is associated with at most one *department* via *stud_dept*





One-to-Many Relationship

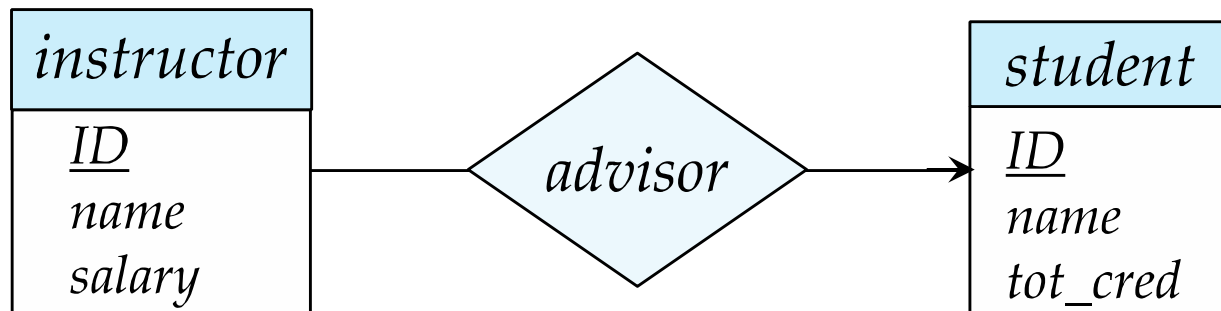
- one-to-many relationship between an *instructor* and a *student*
 - an instructor is associated with several (including 0) students via *advisor*
 - a student is associated with at most one instructor via *advisor*,





Many-to-One Relationships

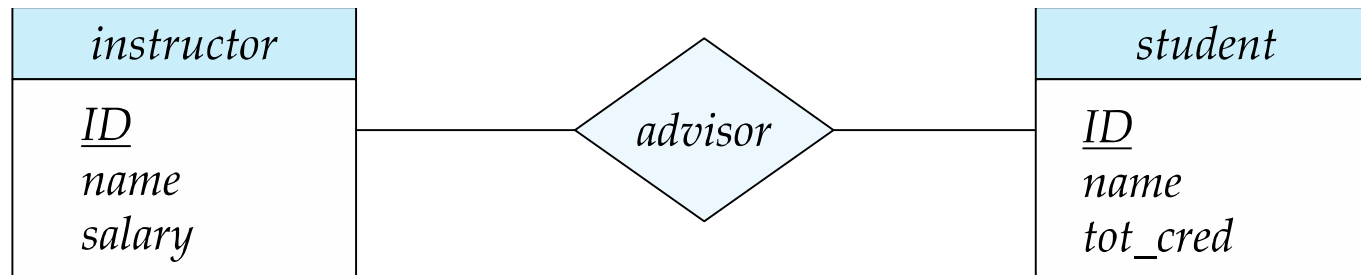
- In a many-to-one relationship between an *instructor* and a *student*,
 - an instructor is associated with at most one student via *advisor*,
 - and a student is associated with several (including 0) instructors via *advisor*





Many-to-Many Relationship

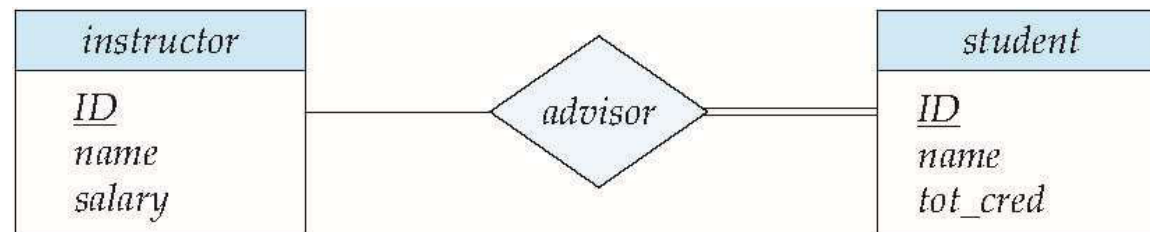
- An instructor is associated with several (possibly 0) students via *advisor*
- A student is associated with several (possibly 0) instructors via *advisor*





Total and Partial Participation

- **Total participation** (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set



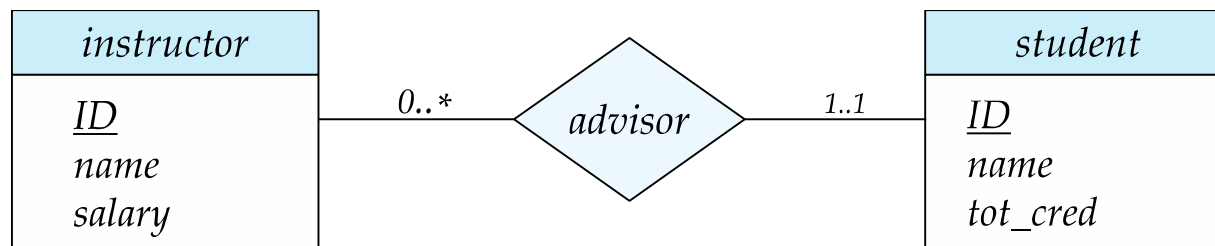
participation of *student* in *advisor* relation is total

- every *student* must have an associated instructor
- **Partial participation:** some entities may not participate in any relationship in the relationship set
 - Example: participation of *instructor* in *advisor* is partial



Notation for Expressing More Complex Constraints

- A line may have an associated minimum and maximum cardinality, shown in the form $l..h$, where l is the minimum and h the maximum cardinality
 - A minimum value of 1 indicates total participation.
 - A maximum value of 1 indicates that the entity participates in at most one relationship
 - A maximum value of * indicates no limit.



Instructor can advise 0 or more students. A student must have 1 advisor; cannot have multiple advisors



Cardinality Constraints on Ternary Relationship

- We allow at most one arrow out of a ternary (or greater degree) relationship to indicate a cardinality constraint
- For example, an arrow from *proj_guide* to *instructor* indicates each student has at most one guide for a project
- If there is more than one arrow, there are two ways of defining the meaning.
 - For example, a ternary relationship R between A , B and C with arrows to B and C could mean
 1. Each A entity is associated with a unique entity from B and C or
 2. Each pair of entities from (A, B) is associated with a unique C entity, and each pair (A, C) is associated with a unique B
 - Each alternative has been used in different formalisms
 - To avoid confusion we outlaw more than one arrow



Primary Key

- Primary keys provide a way to specify how entities and relations are distinguished. We will consider:
 - Entity sets
 - Relationship sets.
 - Weak entity sets



Primary key for Entity Sets

- By definition, individual entities are distinct.
- From database perspective, the differences among them must be expressed in terms of their attributes.
- The values of the attribute values of an entity must be such that they can uniquely identify the entity.
 - No two entities in an entity set are allowed to have exactly the same value for all attributes.
- A key for an entity is a set of attributes that suffice to distinguish entities from each other



Primary Key for Relationship Sets

- To distinguish among the various relationships of a relationship set we use the individual primary keys of the entities in the relationship set.
 - Let R be a relationship set involving entity sets E_1, E_2, \dots, E_n
 - The primary key for R consists of the union of the primary keys of entity sets E_1, E_2, \dots, E_n
 - If the relationship set R has attributes a_1, a_2, \dots, a_m associated with it, then the primary key of R also includes the attributes a_1, a_2, \dots, a_m
- Example: relationship set “advisor”.
 - The primary key consists of *instructor.ID* and *student.ID*
- The choice of the primary key for a relationship set depends on the mapping cardinality of the relationship set.



Choice of Primary key for Binary Relationship

- Many-to-Many relationships. The preceding union of the primary keys is a minimal superkey and is chosen as the primary key.
- One-to-Many relationships . The primary key of the “Many” side is a minimal superkey and is used as the primary key.
- Many-to-one relationships. The primary key of the “Many” side is a minimal superkey and is used as the primary key.
- One-to-one relationships. 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.



Choice of Primary key for Nonbinary Relationship

- If no cardinality constraints are present, the superkey is formed as described earlier. and it is chosen as the primary key.
- If there are cardinality constraints are present:
 - Recall that we permit at most one arrow out of a relationship set.
 - AVI



Weak Entity Sets

- Consider a *section* entity, which is uniquely identified by a *course_id*, *semester*, *year*, and *sec_id*.
- Clearly, section entities are related to course entities. Suppose we create a relationship set *sec_course* between entity sets *section* and *course*.
- Note that the information in *sec_course* is redundant, since *section* already has an attribute *course_id*, which identifies the course with which the section is related.
- One option to deal with this redundancy is to get rid of the relationship *sec_course*; however, by doing so the relationship between *section* and *course* becomes implicit in an attribute, which is not desirable.



Weak Entity Sets (Cont.)

- An alternative way to deal with this redundancy is to not store the attribute *course_id* in the *section* entity and to only store the remaining attributes *section_id*, *year*, and *semester*.
 - However, the entity set *section* then does not have enough attributes to identify a particular *section* entity uniquely
- 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.
- A **weak entity set** is one whose existence is dependent on another entity, called its **identifying entity**
- Instead of associating a primary key with a weak entity, we use the identifying entity, along with extra attributes called **discriminator** to uniquely identify a weak entity.



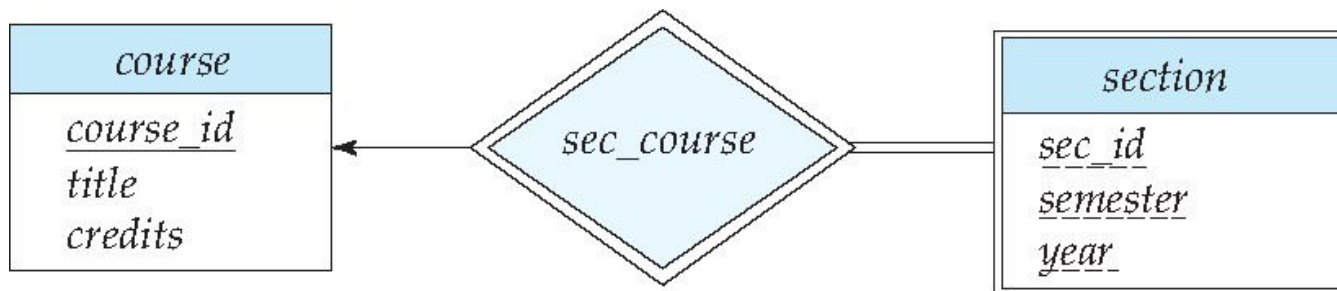
Weak Entity Sets (Cont.)

- 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**.
- Note that the relational schema we eventually create from the entity set *section* does have the attribute *course_id*, for reasons that will become clear later, even though we have dropped the attribute *course_id* from the entity set *section*.



Expressing Weak Entity Sets

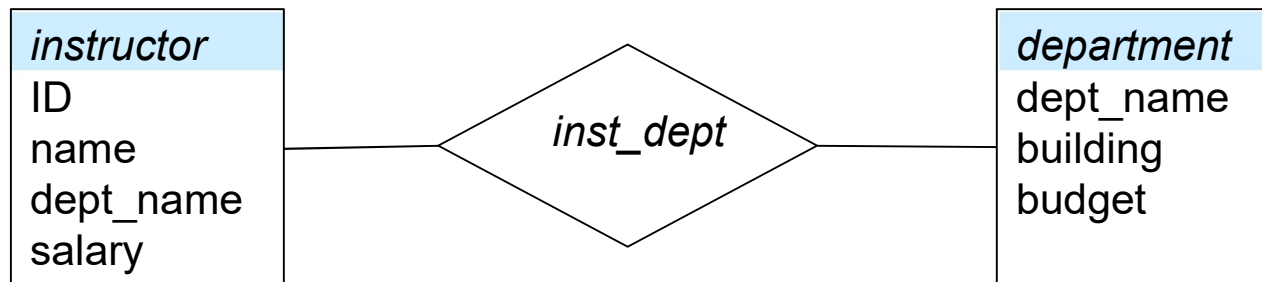
- In E-R diagrams, a weak entity set is depicted via a double rectangle.
- We underline the discriminator of a weak entity set with a dashed line.
- The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a double diamond.
- Primary key for *section* – (*course_id*, *sec_id*, *semester*, *year*)





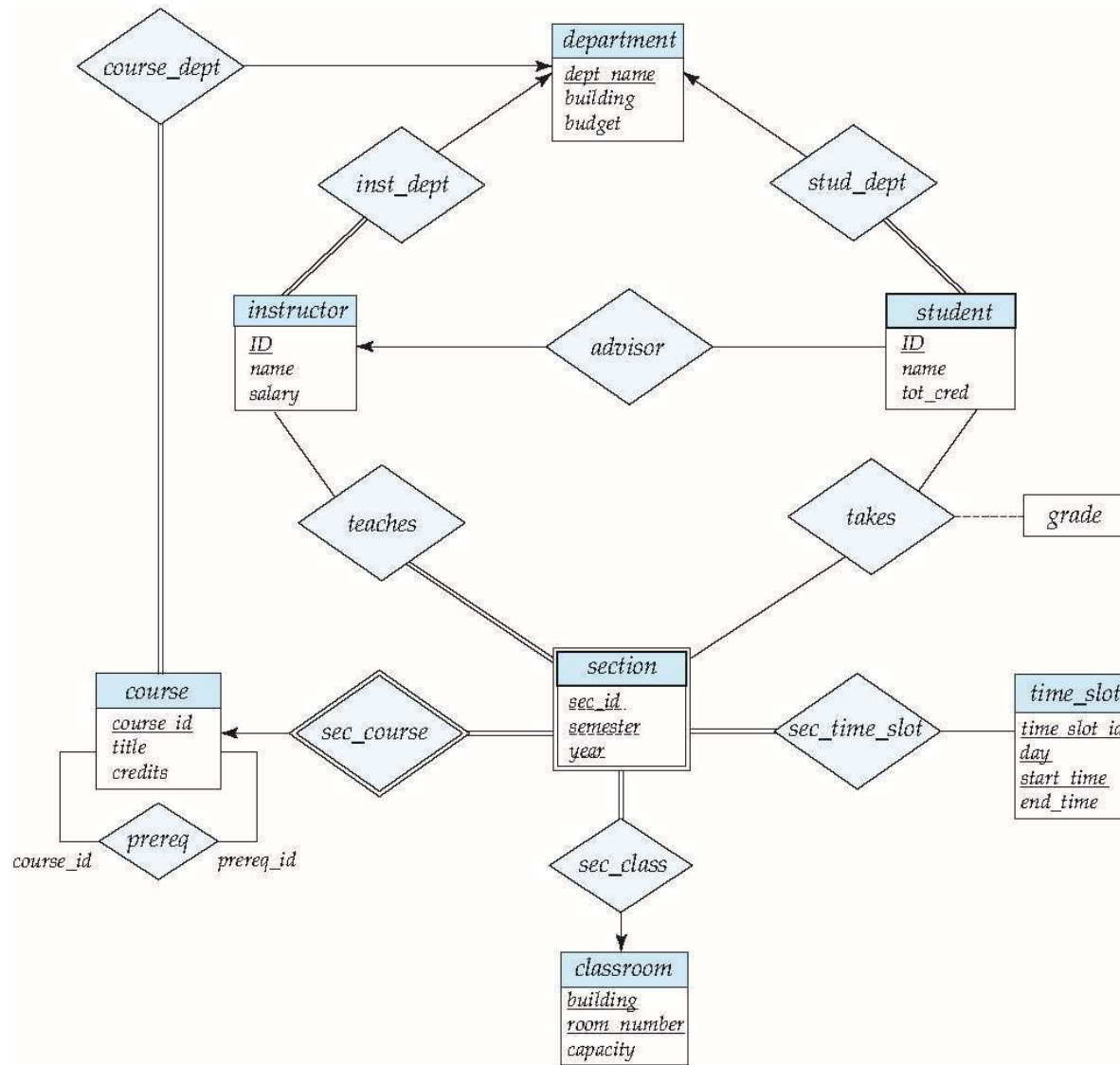
Redundant Attributes

- Suppose we have entity sets:
 - *instructor*, with attributes: *ID*, *name*, *dept_name*, *salary*
 - *department*, with attributes: *dept_name*, *building*, *budget*
- We model the fact that each instructor has an associated department using a relationship set *inst_dept*
- The attribute *dept_name* in *instructor* replicates information present in the relationship and is therefore redundant
 - and needs to be removed.
- BUT: when converting back to tables, in some cases the attribute gets reintroduced, as we will see later.





E-R Diagram for a University Enterprise





Extended E-R Features



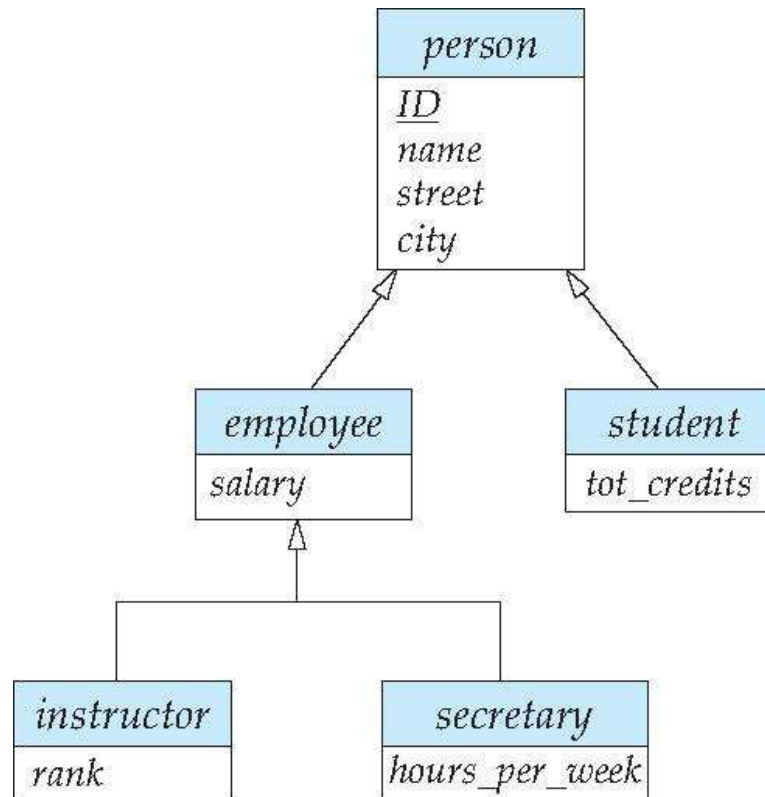
Specialization

- Top-down design process; we designate sub-groupings within an entity set that are distinctive from other entities in the set.
- These sub-groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (e.g., *instructor* “is a” *person*).
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.



Specialization Example

- **Overlapping** – *employee* and *student*
- **Disjoint** – *instructor* and *secretary*
- Total and partial





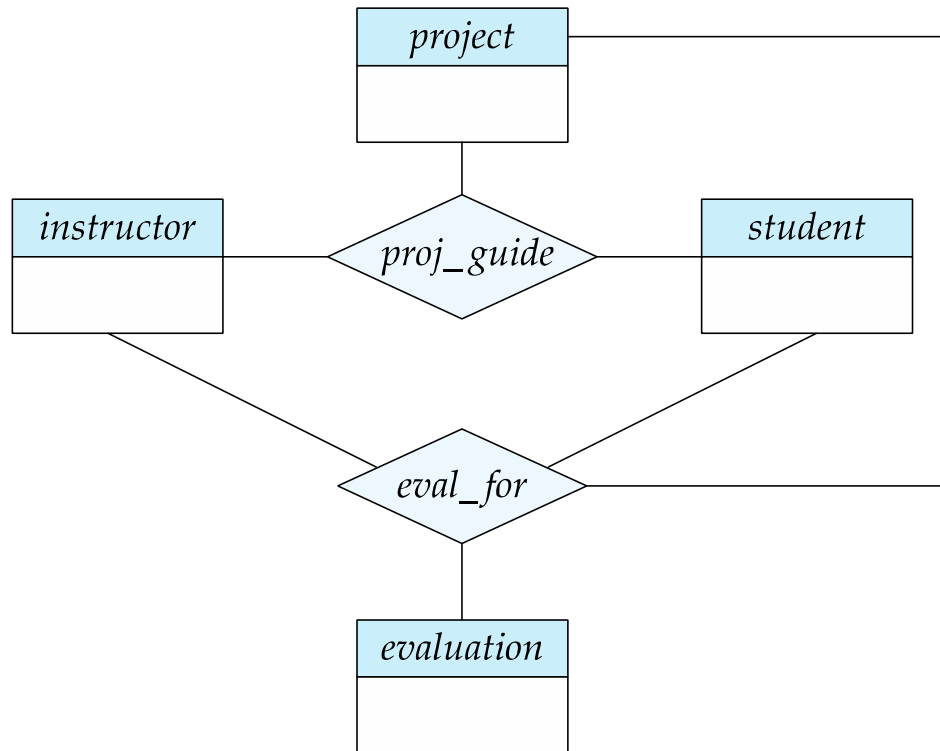
Generalization

- **A bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.



Aggregation

- Consider the ternary relationship *proj_guide*, which we saw earlier
- Suppose we want to record evaluations of a student by a guide on a project





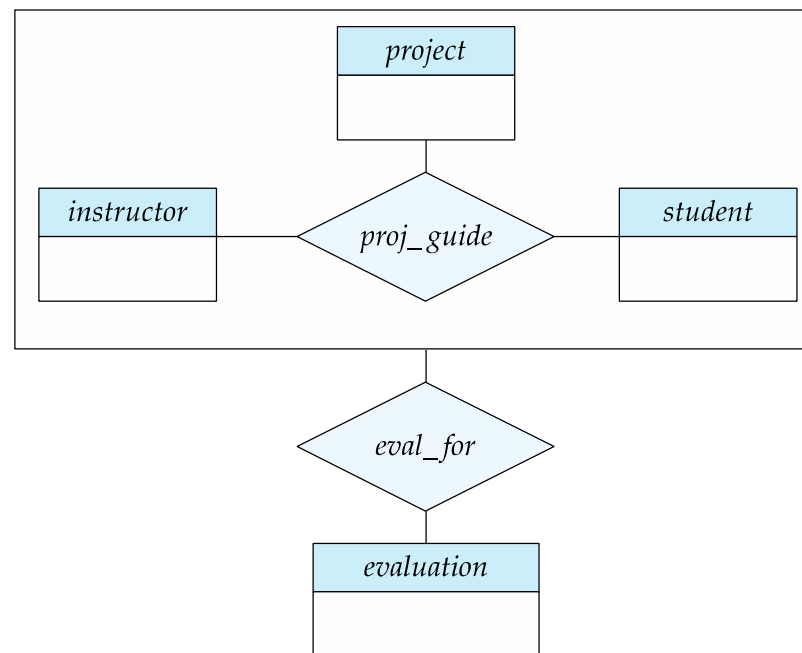
Aggregation (Cont.)

- Relationship sets *eval_for* and *proj_guide* represent overlapping information
 - Every *eval_for* relationship corresponds to a *proj_guide* relationship
 - However, some *proj_guide* relationships may not correspond to any *eval_for* relationships
 - So we can't discard the *proj_guide* relationship
- Eliminate this redundancy via *aggregation*
 - Treat relationship as an abstract entity
 - Allows relationships between relationships
 - Abstraction of relationship into new entity



Aggregation (Cont.)

- Eliminate this redundancy via *aggregation* without introducing redundancy, the following diagram represents:
 - A student is guided by a particular instructor on a particular project
 - A student, instructor, project combination may have an associated evaluation



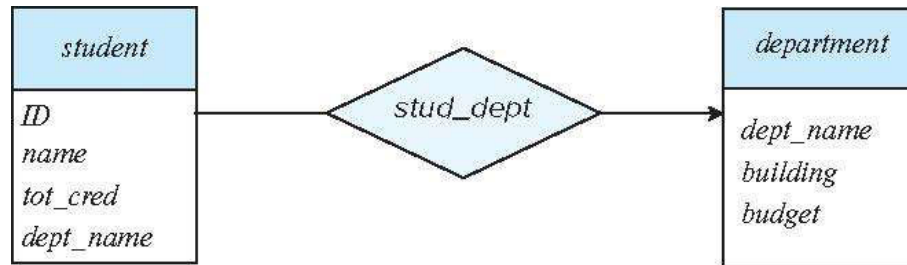


Design Issues

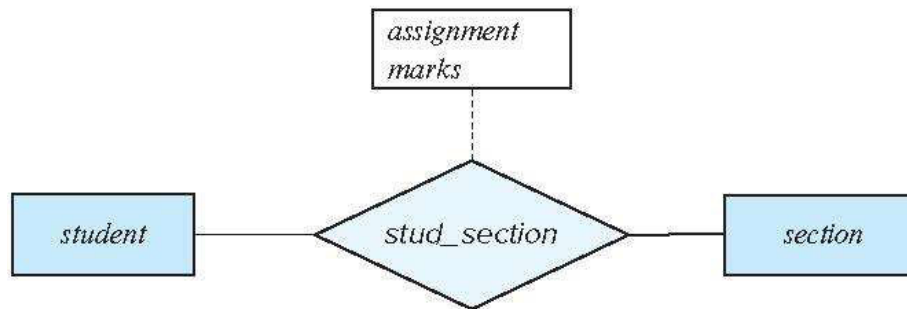


Common Mistakes in E-R Diagrams

- Example of erroneous E-R diagrams



(a) Incorrect use of attribute

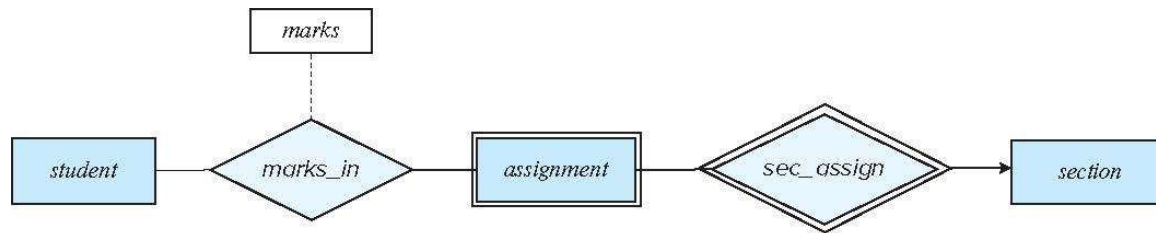


(b) Erroneous use of relationship attributes

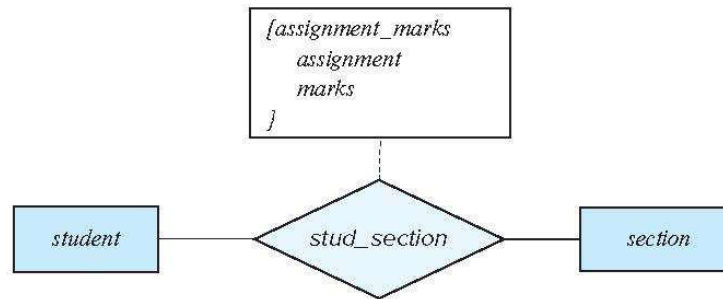


Common Mistakes in E-R Diagrams (Cont.)

- Correct versions of the E-R diagram of previous slide



(c) Correct alternative to erroneous E-R diagram (b)

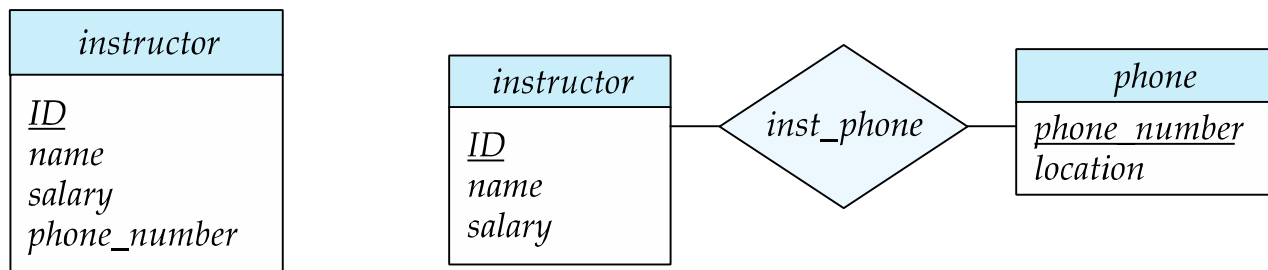


(d) Correct alternative to erroneous E-R diagram (b)



Entities vs. Attributes

- Use of entity sets vs. attributes



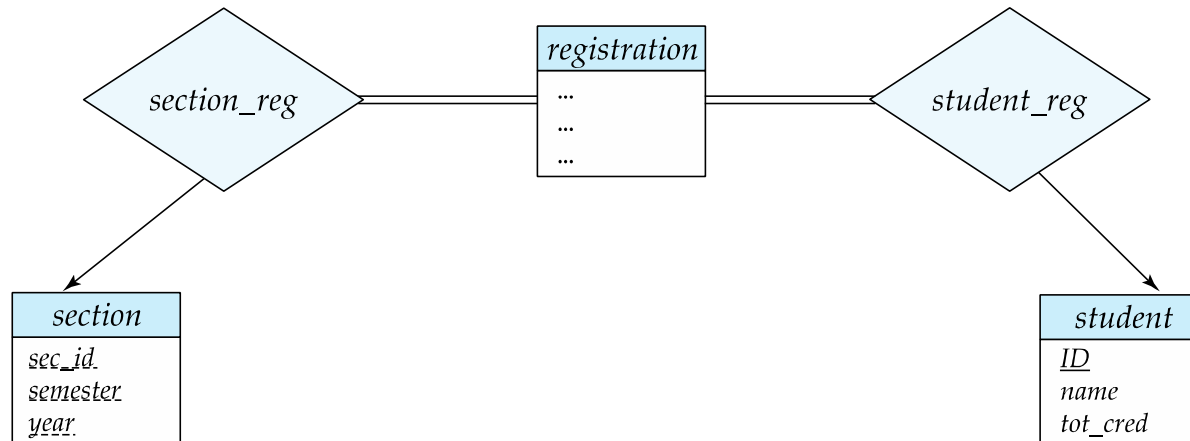
- Use of phone as an entity allows extra information about phone numbers (plus multiple phone numbers)



Entities vs. Relationship sets

- **Use of entity sets vs. relationship sets**

Possible guideline is to designate a relationship set to describe an action that occurs between entities



- **Placement of relationship attributes**

For example, attribute date as attribute of advisor or as attribute of student



Binary Vs. Non-Binary Relationships

- Although it is possible to replace any non-binary (n -ary, for $n > 2$) relationship set by a number of distinct binary relationship sets, a n -ary relationship set shows more clearly that several entities participate in a single relationship.
- Some relationships that appear to be non-binary may be better represented using binary relationships
 - For example, a ternary relationship *parents*, relating a child to his/her father and mother, is best replaced by two binary relationships, *father* and *mother*
 - Using two binary relationships allows partial information (e.g., only mother being known)
 - But there are some relationships that are naturally non-binary
 - Example: *proj_guide*

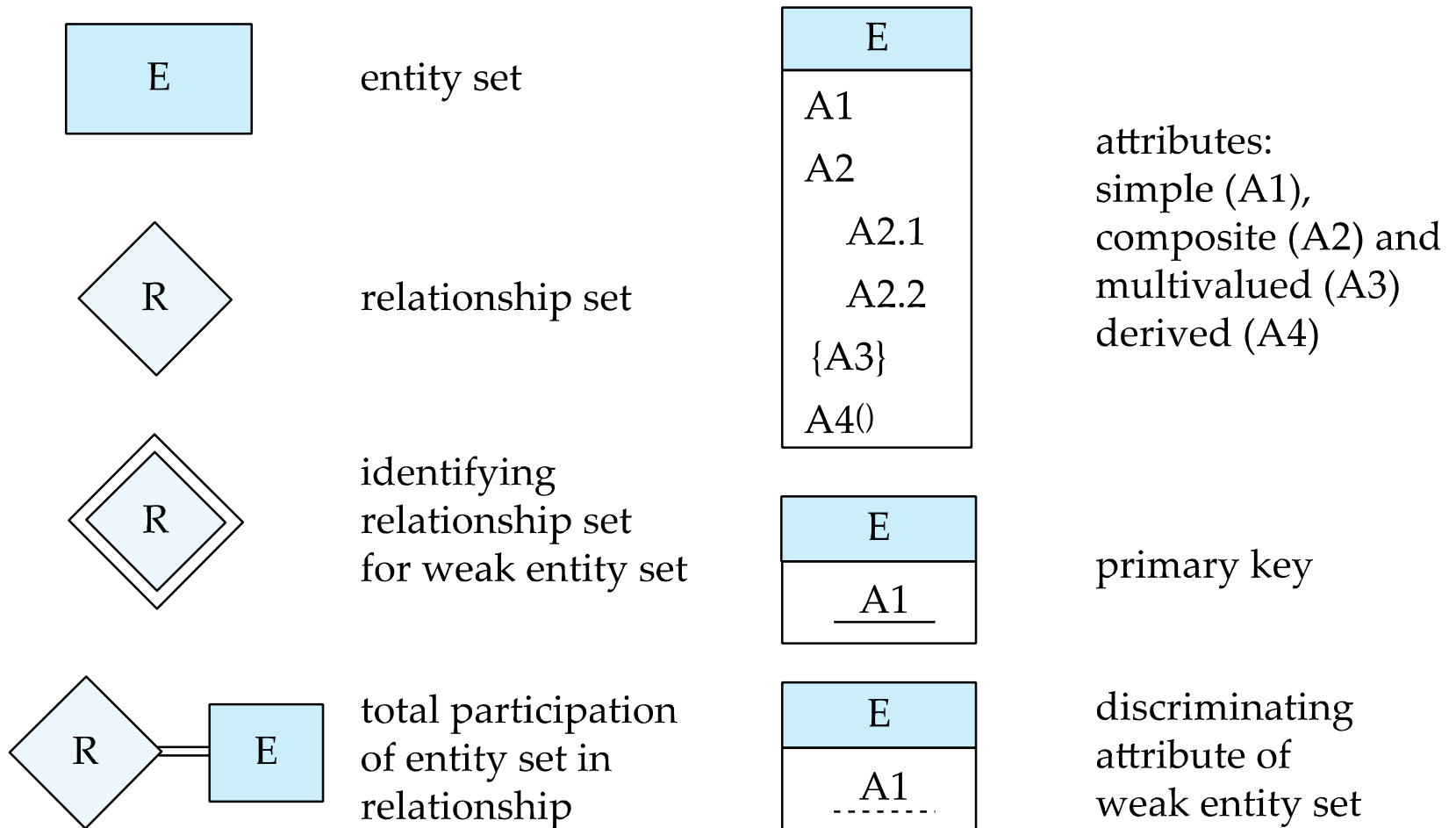


E-R Design Decisions

- The use of an attribute or entity set to represent an object.
- Whether a real-world concept is best expressed by an entity set or a relationship set.
- The use of a ternary relationship versus a pair of binary relationships.
- The use of a strong or weak entity set.
- The use of specialization/generalization – contributes to modularity in the design.
- The use of aggregation – can treat the aggregate entity set as a single unit without concern for the details of its internal structure.

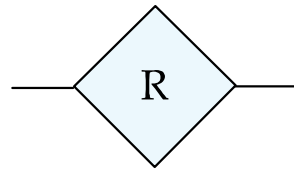


Summary of Symbols Used in E-R Notation

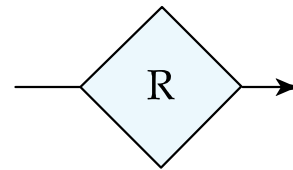




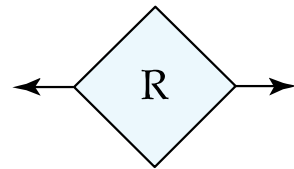
Symbols Used in E-R Notation (Cont.)



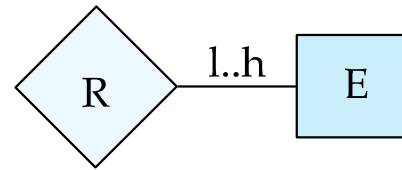
many-to-many
relationship



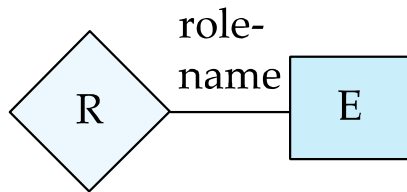
many-to-one
relationship



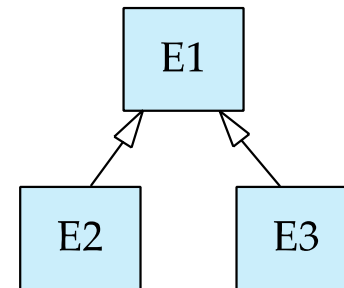
one-to-one
relationship



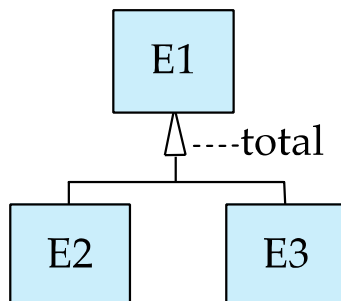
cardinality
limits



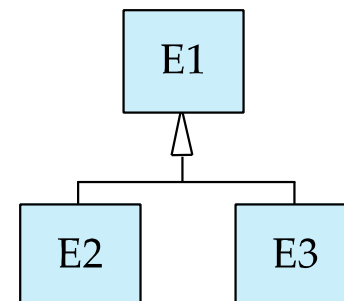
role indicator



ISA: generalization
or specialization



total (disjoint)
generalization



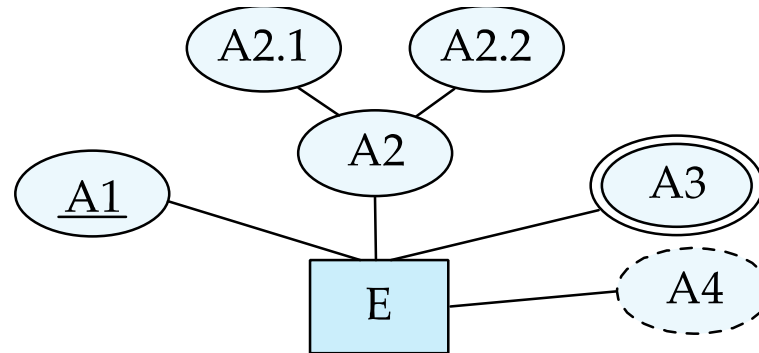
disjoint
generalization



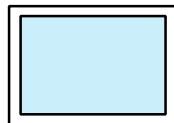
Alternative ER Notations

- Chen, IDE1FX, ...

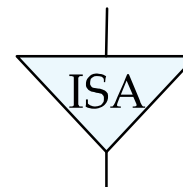
entity set E with
simple attribute A1,
composite attribute A2,
multivalued attribute A3,
derived attribute A4,
and primary key A1



weak entity set



generalization



total
generalization



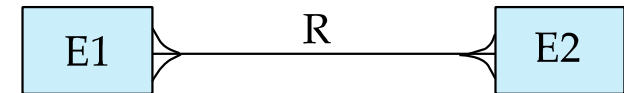
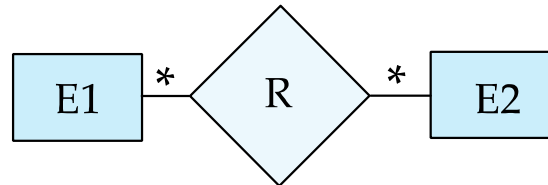


Alternative ER Notations

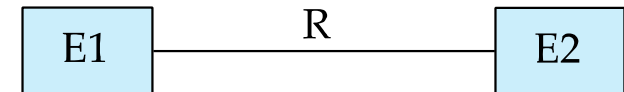
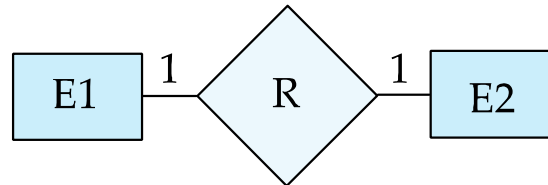
Chen

IDE1FX (Crows feet notation)

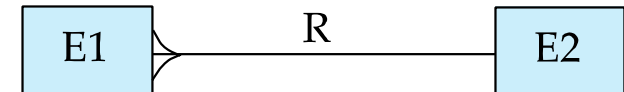
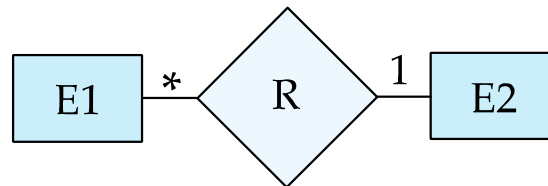
many-to-many
relationship



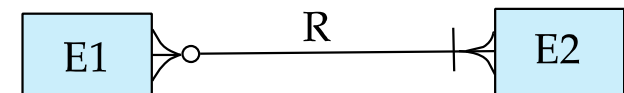
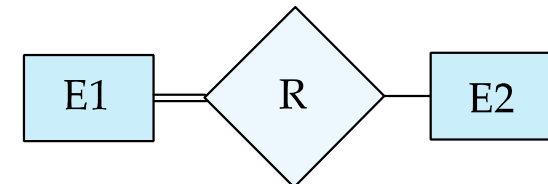
one-to-one
relationship



many-to-one
relationship



participation
in R: total (E1)
and partial (E2)





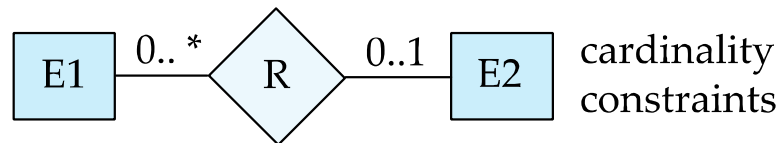
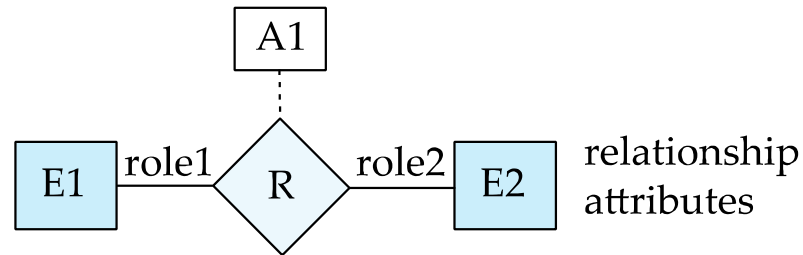
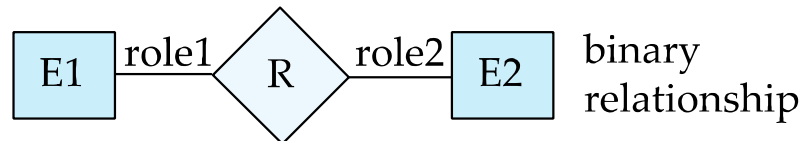
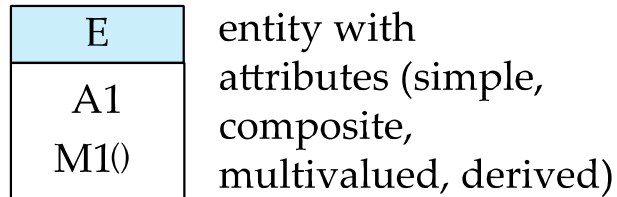
UML

- **UML**: Unified Modeling Language
- UML has many components to graphically model different aspects of an entire software system
- UML Class Diagrams correspond to E-R Diagram, but several differences.

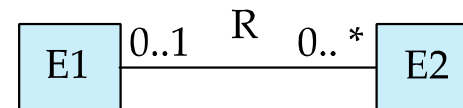
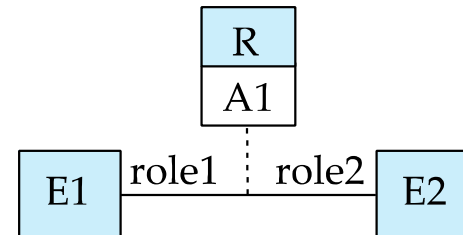
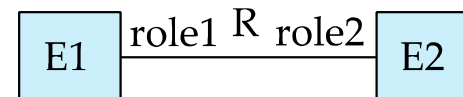
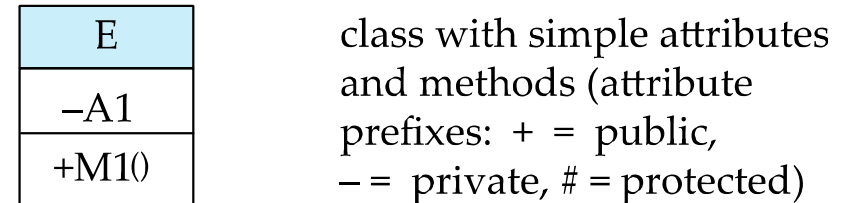


ER vs. UML Class Diagrams

ER Diagram Notation



Equivalent in UML

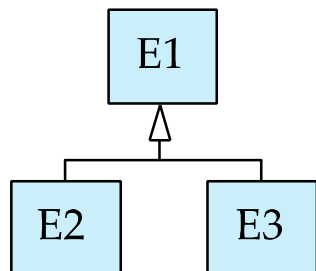
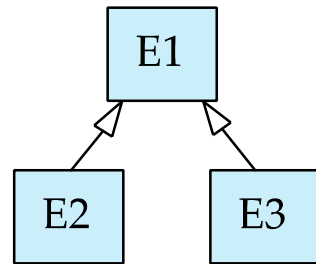
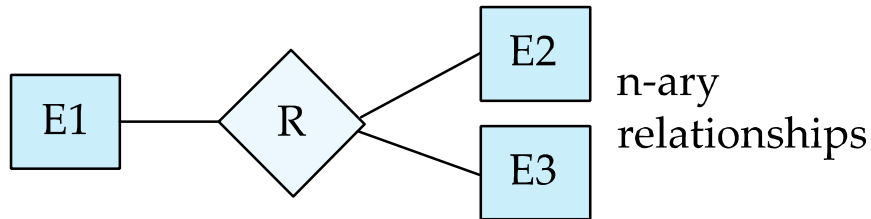


*Note reversal of position in cardinality constraint depiction

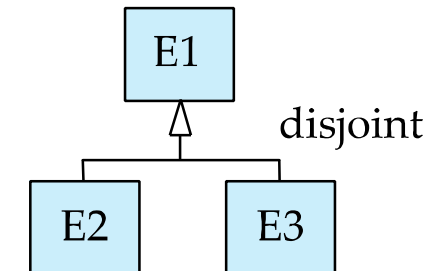
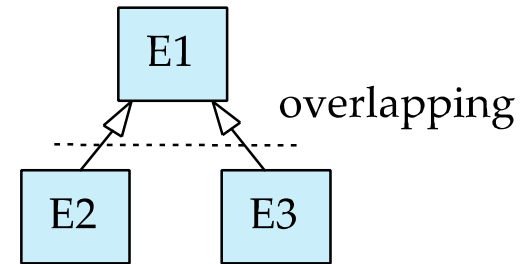
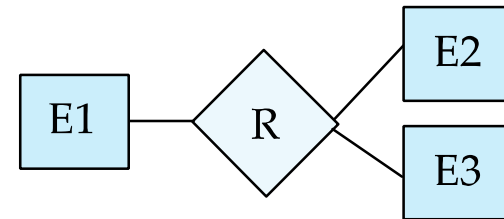


ER vs. UML Class Diagrams

ER Diagram Notation



Equivalent in UML



*Generalization can use merged or separate arrows independent of disjoint/overlapping



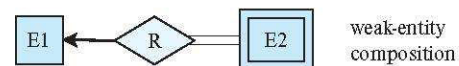
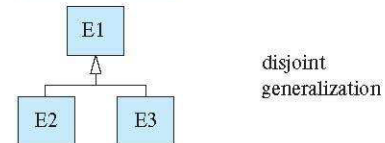
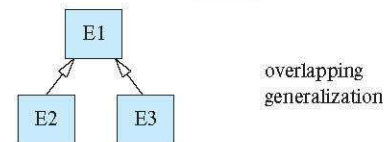
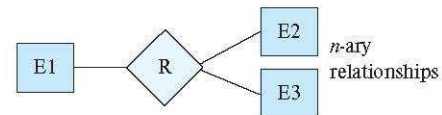
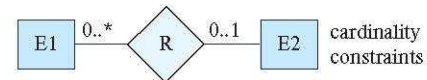
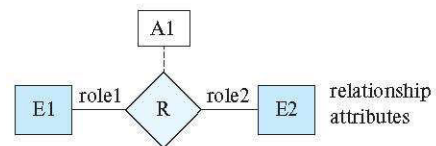
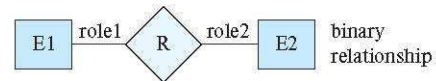
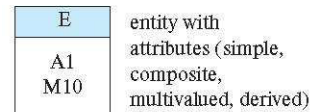
UML Class Diagrams (Cont.)

- Binary relationship sets are represented in UML by just drawing a line connecting the entity sets. The relationship set name is written adjacent to the line.
- The role played by an entity set in a relationship set may also be specified by writing the role name on the line, adjacent to the entity set.
- The relationship set name may alternatively be written in a box, along with attributes of the relationship set, and the box is connected, using a dotted line, to the line depicting the relationship set.

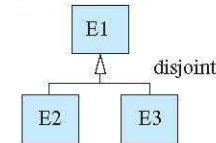
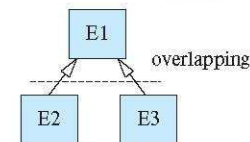
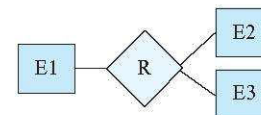
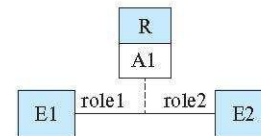
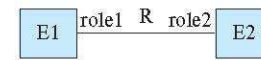
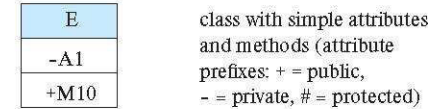


ER vs. UML Class Diagrams

ER Diagram Notation



Equivalent in UML





Other Aspects of Database Design

- Functional Requirements
- Data Flow, Workflow
- Schema Evolution



End of Chapter 6

Keys

For example, ID is used as a key in the Student table because it is unique for each student. In the PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.

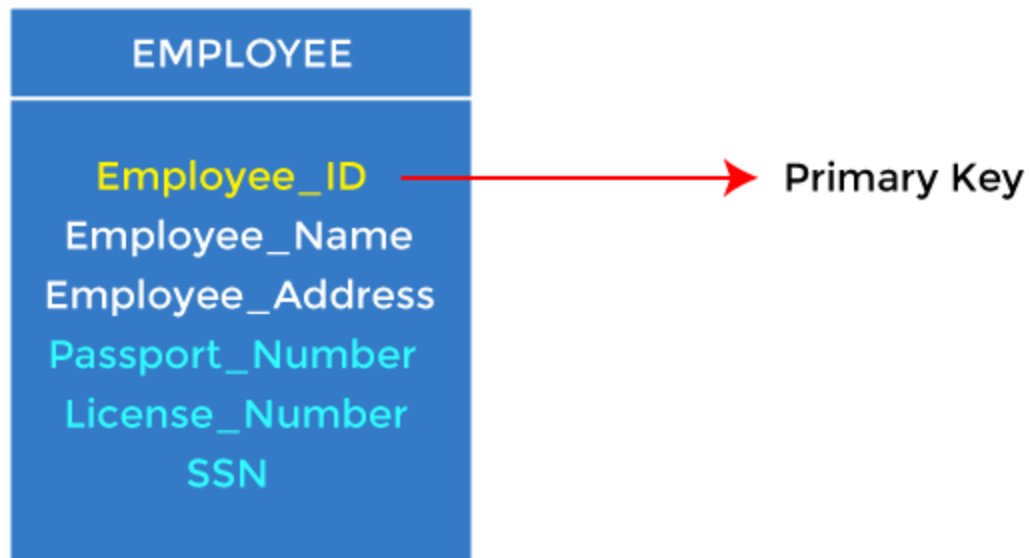
STUDENT
ID
Name
Address
Course

PERSON
Name
DOB
Passport, Number
License_Number
SSN

Types of keys:

1. Primary key

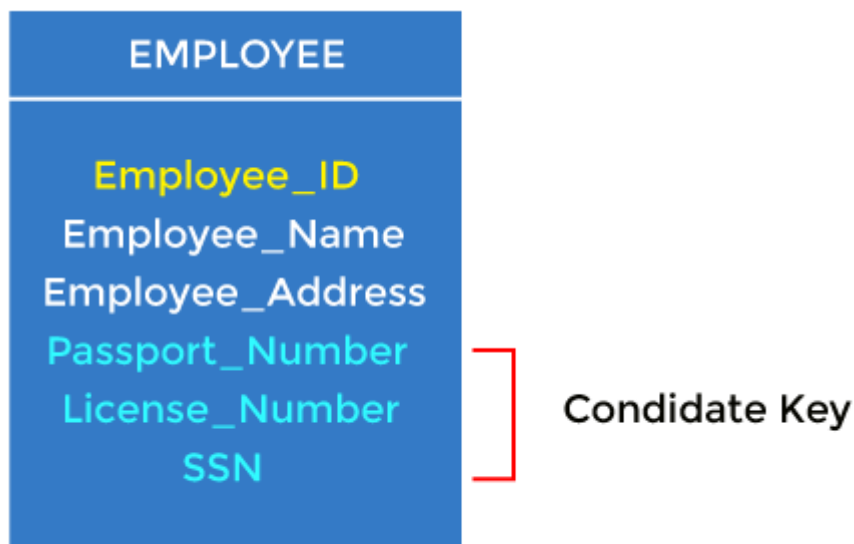
- It is the first key used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.
- In the EMPLOYEE table, ID can be the primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary keys since they are also unique.
- For each entity, the primary key selection is based on requirements and developers.



2. Candidate key

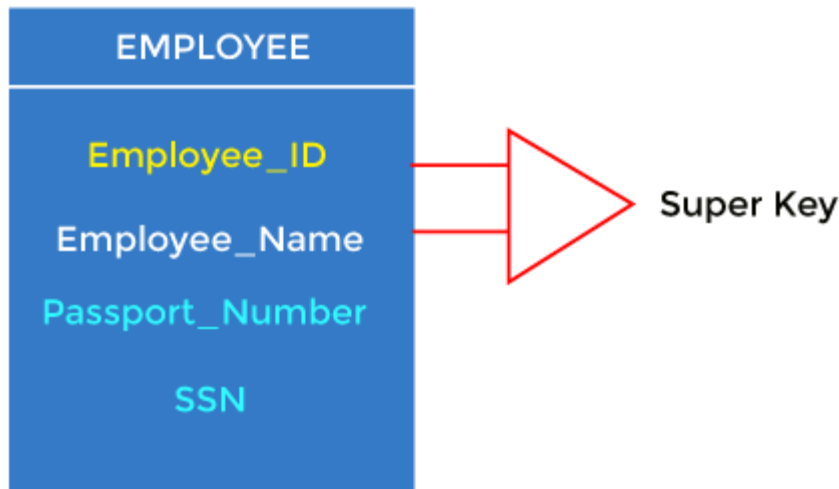
- A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
- Except for the primary key, the remaining attributes are considered a candidate key.
The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. The rest of the attributes, like SSN, Passport_Number, License_Number, etc., are considered a candidate key.



3. Super Key

Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.

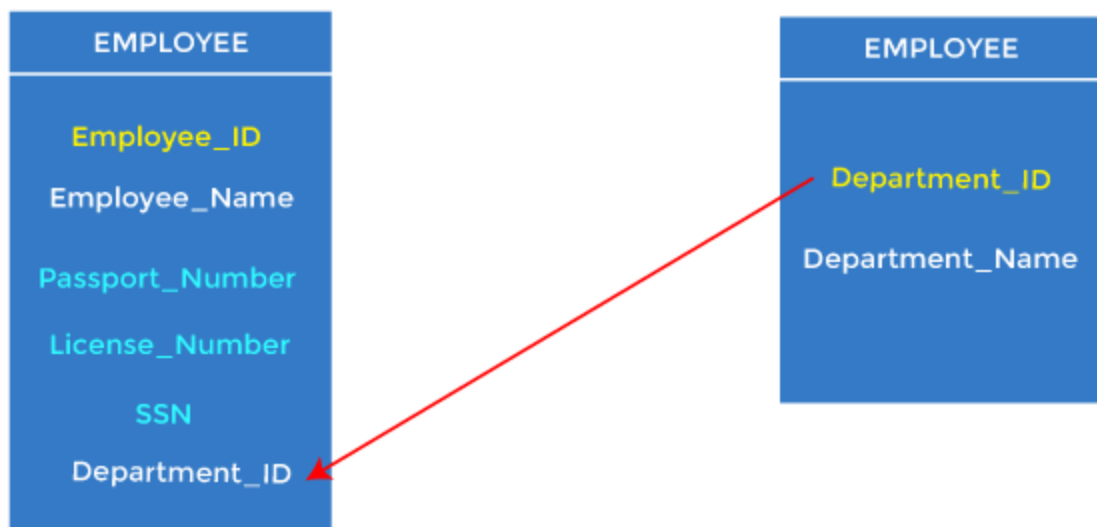


For example: In the above EMPLOYEE table, for(EMPLOYEE_ID, EMPLOYEE_NAME), the name of two employees can be the same, but their EMPLOYEE_ID can't be the same. Hence, this combination can also be a key.

The super key would be EMPLOYEE-ID (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

4. Foreign key

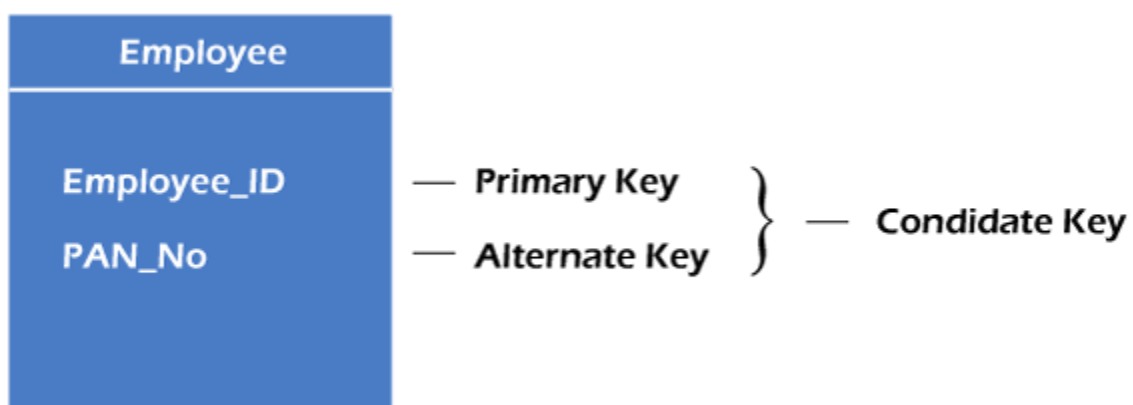
- Foreign keys are the column of the table used to point to the primary key of another table.
- Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id, as a new attribute in the EMPLOYEE table.
- In the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



5. Alternate key

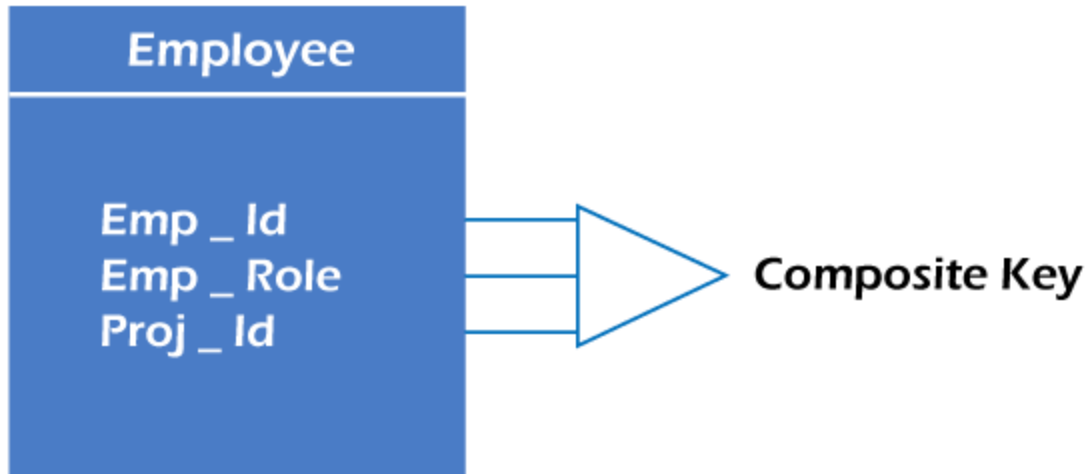
There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternate key. **In other words**, the total number of the alternate keys is the total number of candidate keys minus the primary key. The alternate key may or may not exist. If there is only one candidate key in a relation, it does not have an alternate key.

For example, employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. In this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as the Alternate key.

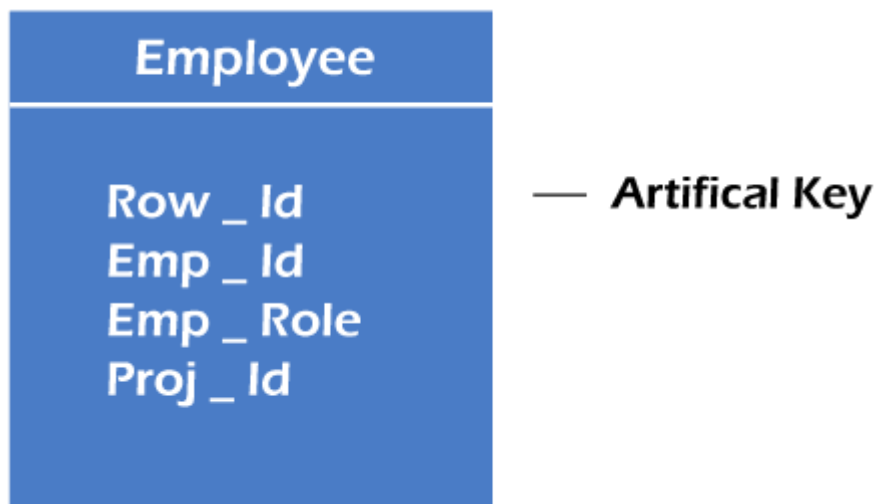


6. Composite key

Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.



For example, in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously. So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute.



7. Artificial key

The key created using arbitrarily assigned data are known as artificial keys. These keys are created when a primary key is large and complex and has no relationship with many other relations. The data values of the artificial keys are usually numbered in a serial order.

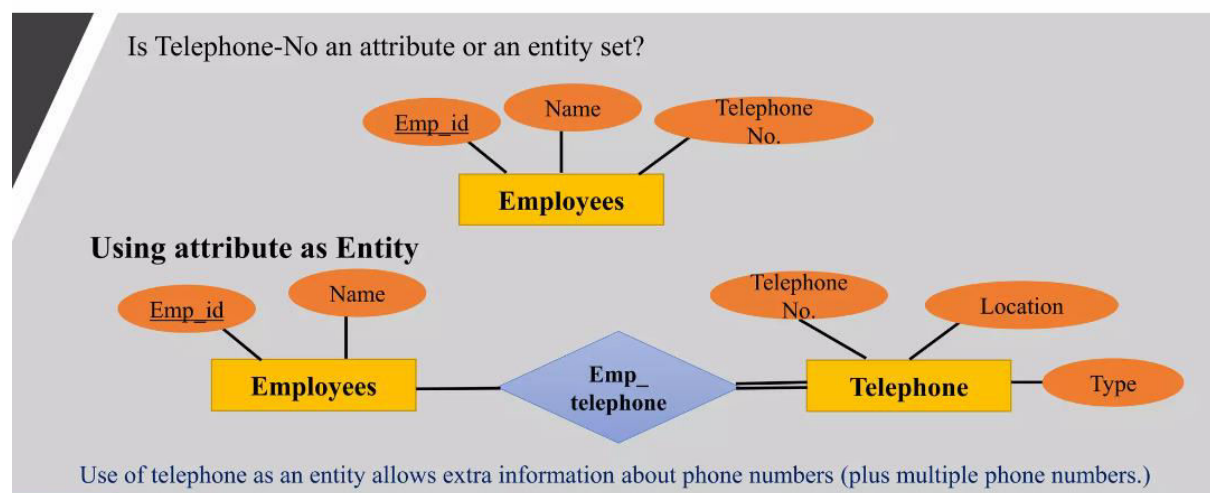
For example, the primary key, which is composed of Emp_ID, Emp_role, and Proj_ID, is large in employee relations. So it would be better to add a new virtual attribute to identify each tuple in the relation uniquely.

ER Design Issues

1) Use of Entity Set vs Attributes

The use of an entity set or attribute depends on the structure of the real-world enterprise that is being modelled and the semantics associated with its attributes. It leads to a mistake when the user use the primary key of an entity set as an attribute of another entity set. Instead, he should use the relationship to do so. Also, the primary key attributes are implicit in the relationship set, but we designate it in the relationship sets.

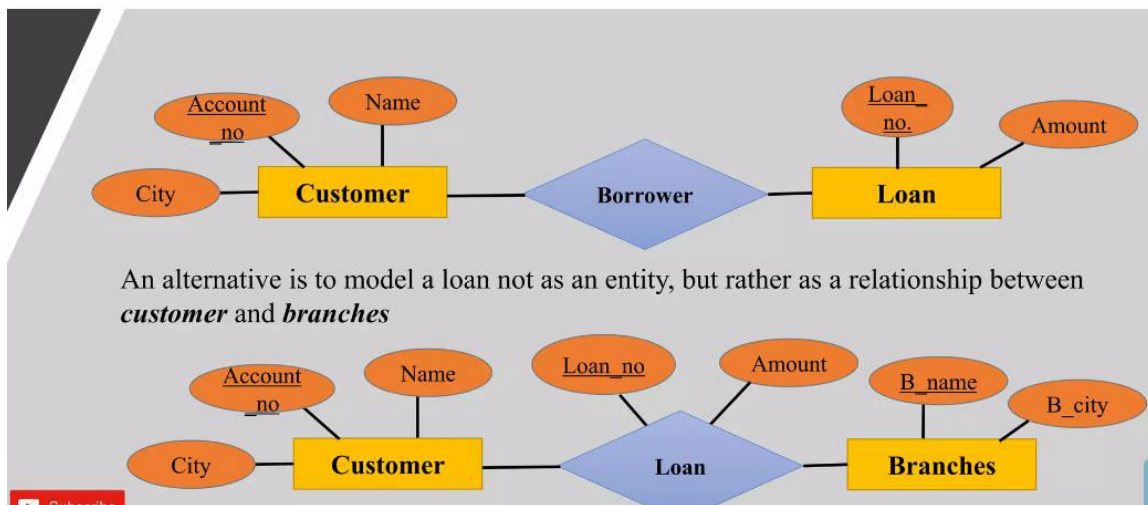
Entity vs Attribute



2) Use of Entity Set vs. Relationship Sets

It is difficult to examine if an object can be best expressed by an entity set or relationship set. To understand and determine the right use, the user need to designate a relationship set for describing an action that occurs in-between the entities. If there is a requirement of representing the object as a relationship set, then its better not to mix it with the entity set.

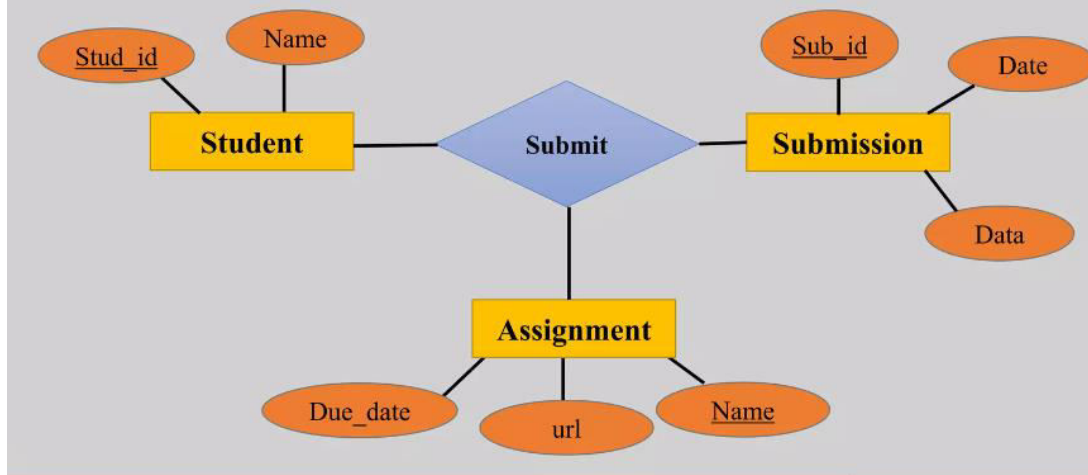
Entity vs Relationship

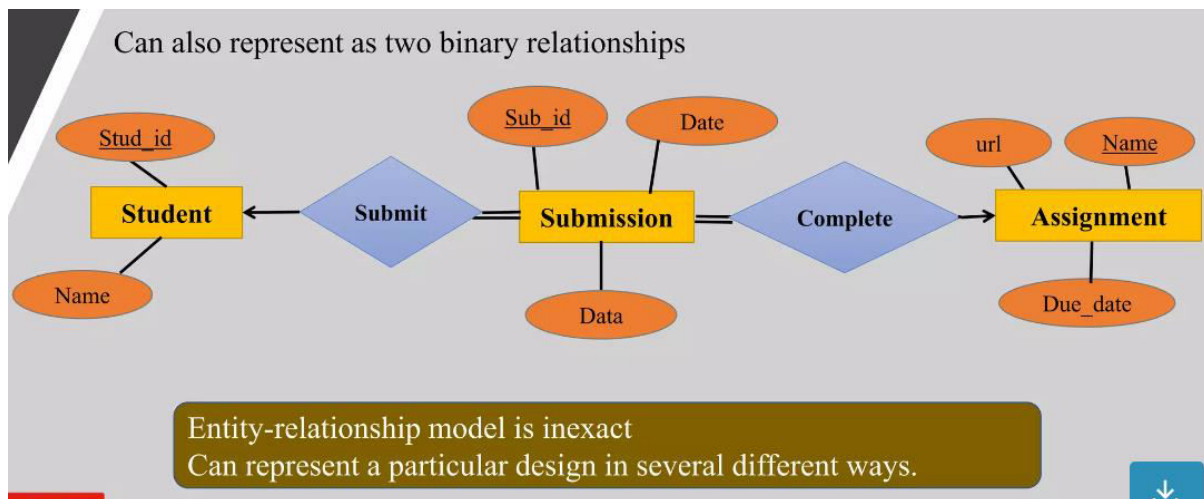


3) Use of Binary vs n-ary Relationship Sets

Generally, the relationships described in the databases are binary relationships. However, non-binary relationships can be represented by several binary relationships. For example, we can create and represent a ternary relationship 'parent' that may relate to a child, his father, as well as his mother. Such relationship can also be represented by two binary relationships i.e, mother and father, that may relate to their child. Thus, it is possible to represent a non-binary relationship by a set of distinct binary relationships.

Ternary relationship between student, assignment, and submission. Allow multiple submission for a particular assignment from a particular student.





Total and Partial Participation

- n Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set

participation of *student* in *advisor* relation is total

- ▶ every *student* must have an associated instructor

- n Partial participation: some entities may not participate in any relationship in the relationship set

- 1 Example: participation of *instructor* in *advisor* is partial

Design a conceptual schema by creating an ER diagram

Step 1. Identify the entity types.

Step 2. Identify the relationship types and their participating entity types.

Step 3. Identify the attributes, keys, and identifier of each entity type and relationship type and obtain an ER diagram

Step 4. Convert the ER diagram to a normal form ER diagram.

Step 5. Translate the NF-ER diagram to a relational database schema (or other data models)



Entity



Relationship



Attribute



Weak Entity



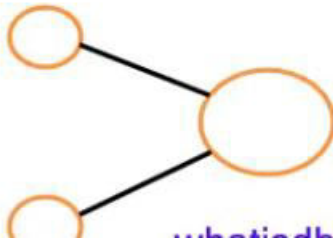
Weak Entity
Relationship



Multivalued
Attribute



Key Attribute



Composite
Attribute

