Introduction to Database Management System

Database Systems

- 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.
- A modern database system is a complex software system whose task is to manage a large, complex collection of data.
- Databases touch all aspects of our lives

Database Applications Examples

- Enterprise Information
  - Sales: customers, products, purchases
  - Accounting: payments, receipts, assets
  - Human Resources: Information about employees, salaries, payroll taxes.
- Manufacturing: management of production, inventory, orders, supply chain.
- Banking and finance
  - customer information, accounts, loans, and banking transactions.
  - Credit card transactions
  - Finance: sales and purchases of financial instruments (e.g., stocks and bonds; storing real-time market data
- Universities: registration, grades

Database Applications Examples

- Airlines: reservations, schedules
- Telecommunication: records of calls, texts, and data usage, generating monthly bills, maintaining balances on prepaid calling cards
- Web-based services
  - Online retailers: order tracking, customized recommendations
  - Online advertisements
- Document databases
- Navigation systems: For maintaining the locations of varies places of interest along with the exact routes of roads, train systems, buses, etc.

Purpose of DBMS

- **File based System:-**
- File based systems were an early attempt to computerize the manual filing system that we all are familiar with.

- A collection of application programs that perform services for the end-users such as the production of reports. Each program defines and manages its own data.

2.Difficulty in accessing data

- In file processing, to allow users to manipulate the information, the system has a number of application programs.
- When we want a particular set of information with file system, the system programmer has to write the necessary application program.
- Eg: customers name in a particular city.
- i.e., File processing environments do not allow needed data to be retrieved in a convenient and efficient manner.

- The various **disadvantages** of keeping organizational information in a file processing system are:

1. Data redundancy and inconsistency

- The same information may be duplicated in several files.
- For eg:, the address and telephone no: of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost.
- In addition, it may lead to data inconsistency, i.e., the various copies of the same data may no longer agree.
- For eg; a changed address may be reflected in savings-account records but not elsewhere in the system.

3. Separation and isolation of data

Since data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

Data Isolation deals with consistency and completeness of **data** retrieved by queries unaffecting a user **data** by other user actions.

4. Integrity problems

The data values stored in the database must satisfy certain types of consistency constraints.

For eg:, the balance of a bank account may never fall below a prescribed amount. Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the program to enforce them. The problem is compounded when constraints involve several data items from different files.

5. Atomicity problems

- A computer is subject to failure. In many applications , it is crucial that, once a failure occurs and has been detected, the data are restored to the consistent state that existed prior to the failure.
- For e.g., consider transferring of money from account A to B. If a system failure occurs during the execution of a program, it is possible that the money was removed from account A but was not credited to account B, resulting in an inconsistent database state. So the fund transfer must be atomic that means it must happen in its entirety or not at all.

7.Security problems.

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

6. Concurrent access anomalies.

- For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously.
- In such an environment, interaction of concurrent updates may result in inconsistent data.
- Consider bank account A, containing Rs.5000/-. If two customers withdraw funds (say Rs.500 & Rs.1000 respectively) from account A at about the same time, the result of the concurrent executions may leave the account in an incorrect state.
- If the two programs run concurrently, they may both red the value Rs.5000, and write back Rs.4500 and Rs.4000 respectively.
- Depending on which one writes the vale last, the account may contain Rs.4500 or Rs.4000, rather than the correct value of Rs.3500. To guard against this possibility, system must maintain some form of supervision

Benefits of Database Approach

- The data can be shared.
- Redundancy can be reduced.
- Inconsistency can be avoided.
- Transaction support can be provided.
- Improved security.
- Improved data integrity.
- Improved maintenance through data independence.
- Increased concurrency.
- Improved backup and recovery services.

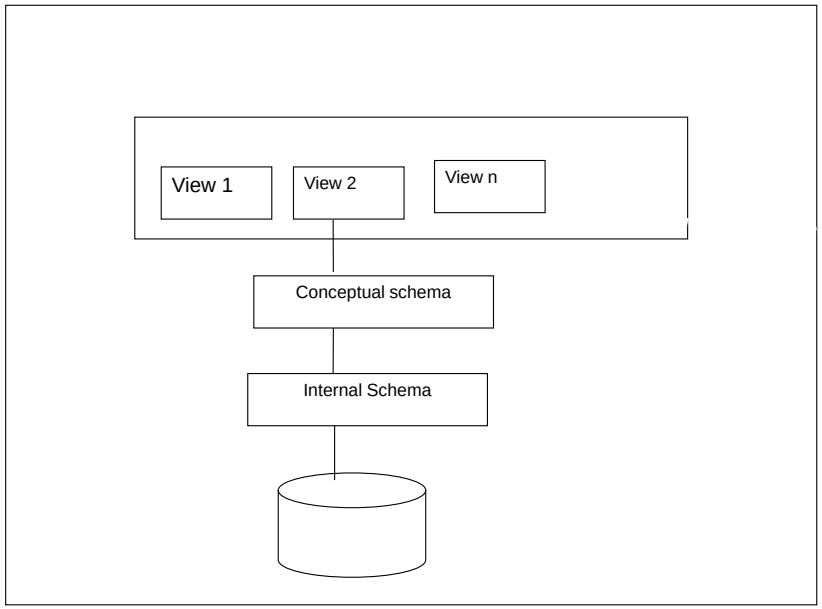
Views of Data – The Three- Level ANSI-SPARC Architecture

- The major purpose of a database system is to provide users with an *abstract view of the data. That is, the system* hides certain details of how the data are stored and maintained.

Data Abstraction

- The system hides certain details of how the data are stored and maintained.
- To retrieve data efficiently, complex data structures are used to represent data in the database. Since many database systems users are not computer trained, developers hide the complexity from users through several *levels of abstraction* to simplify user's interactions with the system.

The Three- Level ANSI-SPARC Architecture



1. Internal or Physical level :

- The lowest level of abstraction describes *how the data are actually stored*.
- At physical level, complexity of data structures is described in detail.
- The internal level covers the physical implementation of the database to achieve optimal runtime performance and storage space utilization.
- It covers the data structures and file organizations used to store data on storage devices.

2. Conceptual or Logical Level :

- The next higher level of abstraction describes *what data are stored in the database and what relationships exist among those data*.
- It thus describes the entire database in terms of a small number of relatively simple structures. The conceptual level represents
  - all entities, their attributes, and their relationships;
  - the constraints on the data;
  - semantic information about the data;
  - security and integrity information.

3. View Level :

- The highest level of abstraction describes only part of the entire database.
- Many users of the database system do not need all the information stored in the database, instead they need to access only a part of the database.
- The view level of abstraction exists to simplify their interaction with the system.
- The system may provide many views for the same database.
- Views provide a level of security. Views can be set up to exclude data that some users should not see.

- Views provide a mechanism to customize the appearance of the database.
- A view can present a consistent, unchanging picture of the structure of the database, even if the underlying database is changed.

Instances and Schemas:-

- The collection of information stored in the database at a particular moment is called an **instance** of the database.
- A **schema** describes the organization of data and relationships within the database. i.e., the overall design of the database is called the **database schema.**
- A database schema corresponds to the variable declarations in a program. Each variable has a particular value at a given instant.
- The values of the variables in a program at a point in time correspond to an instance of a database schema.

- According to the levels of abstraction, database systems have several schemas.
  - The **physical schema (internal schema)** describes the database design at the physical level.
  - The **logical schema (conceptual schema)** describes the database design at the logical level.
  - The schemas at the view level can be called as **sub schemas (external schema)** and they describe views of the database for particular users.

Data Models

A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

A data model provides a way to describe the design of a database at the physical, logical, and view levels.

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.

- The columns of the table correspond to the attributes of the record type.
- Relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.

- Relational Model.

  The 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.

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.

Object-Based Data Model.

Object-oriented programming (especially in Java,C++, or C#) has become the dominant software-development methodology.

This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods(functions), and object identity.

The object-relational data model combines features of the object-oriented data model and relational data model.

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.

The Extensible Markup Language (XML) is widely used to represent semi structured data.

Database Languages:-

- A database system provides a **data definition language** to specify the database schema and a **data manipulation language** to express database queries and updates.

DDL

- Specify a database schema by a set of definitions expressed by a special language called a **data-definition language (DDL).**
- For instance, the following SQL DDL statement defines the department table:

create table department (dept name char (20), building char (15), budget numeric (12,2));

- The DDL is also used to specify additional properties of the data.

- We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a **data storage and definition language(DSDL).**

- The SQL DDL also supports a number of types of integrity constraints.
- For example, one can specify that the dept name attribute value is a primary key, ensuring that no two departments can have the same department name.
- As another example, one can specify that the dept name attribute value appearing in any instructor record must also appear in the dept name attribute of some record of the department table

Referential Integrity.

Ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity).

For example, the department listed for each course must be one that actually exists.

.

Integrity constraints

Domain Constraints.

- A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types).

- Declaring an attribute to be of a particular domain acts as a constraint on the values that it can take.

Database modifications can cause violations of referential integrity.

When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.

Assertions:-

An assertion is any condition that the database must always satisfy. Domain constraints and referential-integrity constraints are special

forms of assertions. However, there are many constraints that we cannot express by using only these special forms.

Authorization.

We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database.

- **read authorization**

which allows reading, but not modification of data.

- **insert authorization**

which allows insertion of new data, but not modification of existing data.

- **update authorization**

which allows modification,but not deletion, of data.

- **delete authorization**

which allows deletion of data. We may assign the user all, none, or a combination of these types of

authorization.

Data Manipulation Language (DML)

- DML is a language that provides a set of operations to support the basic data manipulation operations on the data held in the database.
- Data manipulation operation usually includes
  - The retrieval of information stored in the database
  - The insertion of new information into the database.
  - The deletion of information from the database.
  - The modification of information stored in the database.
- A DML is a language that enables users to access or manipulate data as organized by the appropriate data model.

- The part of a DML that involves data retrieval is called **query language**.
- A query language can be defined as a high level special purpose language used to satisfy diverse requests for the retrieval of data held in the database

- **Two types of DML:**
  - **Procedural DML** - require a user to specify what data are needed and how to get those data.
  - **Non Procedural DML (Declarative DML)**- require a user to specify what data are needed without specifying how to get those data.

- A query takes as input several tables (possibly only one) and always returns a single table. Here is an example of an SQL query that finds the names of all instructors in the History department:

select instructor.name from instructor where instructor.dept name = 'History';

Database Users and User Interfaces

- There are four different types of database-system users, differentiated by the way they expect to interact with the system.

Database 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 clerk in the university who needs to add a new instructor to the database

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

Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.

Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

- **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.

Database Administrator

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

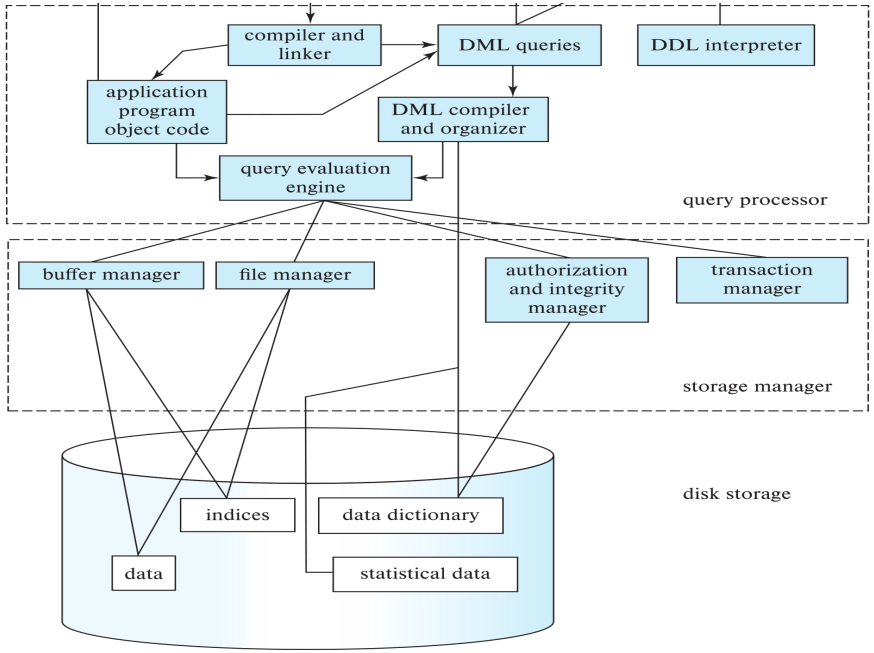
The functions of a DBA include:

- **Schema definition**-TheDBA creates the original database schema by executing a set of data definition statements in the DDL.
- **Granting of authorization for data access.**
  - By granting different types of authorization, the database administrator can regulate which parts of the database various users can access.
  - The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

Routine maintenance.

Examples of the database administrator's routine maintenance activities are:

- ◦Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
- ◦Ensuring that enough free disk space is available for normal operations,and upgrading disk space as required.
- ◦Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.



Database and Application Architecture

- The figure summarizes how different types of users interact with a database, and how the different components of a database engine are connected to each other.
- shows the architecture of a database system that runs on a centralized server machine.
- The centralized architecture shown is applicable to shared-memory server architectures, which have multiple CPUs and exploit parallel processing, but all the CPUs access a common shared memory.

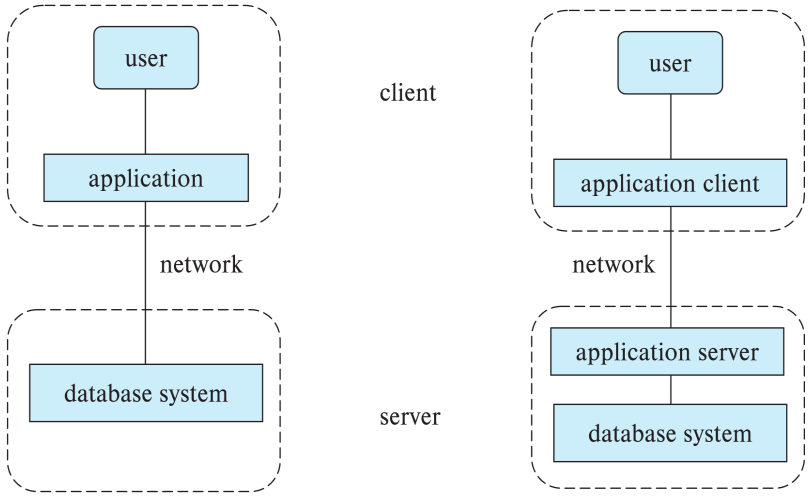
- To scale up to even larger data volumes and even higher processing speeds, parallel databases are designed to run on a cluster consisting of multiple machines.
- Further, distributed databases allow data storage and query processing across multiple geographically separated machines.

- Consider the architecture of applications that use databases as their backend.
- Database applications can be partitioned into two or three parts, as shown in *Figure*
- Earlier-generation database applications used a two-tier architecture, where the application resides at the client machine, and invokes database system functionality at the server machine through query language statements.

- In contrast, modern database applications use a three-tier architecture, where the client machine acts as merely a front end and does not contain any direct database calls; web browsers and mobile applications are the most commonly used application clients today.
- The front end communicates with an application server.

- The application server, in turn, communicates with a database system to access data.
- The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients.
- Three tier applications provide better security as well as better performance than two-tier applications.

Two-tier and three-tier architectures



(a) Two-tier architecture (b) Three-tier architecture

Modeling

- A *database* can be modeled as:
  - a collection of entities,
  - relationship among entities.
- An **entity** is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- Entities have **attributes**
  - Example: people have *names* and *addresses*
- An **entity set** is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees

Entity Sets *instructor* and *student*

- instructor_ID instructor_name

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

instructor

student-ID student_name

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

student

Relationship Sets

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

  Example:

  44553 (Peltier) *advisor* 22222 (Einstein)

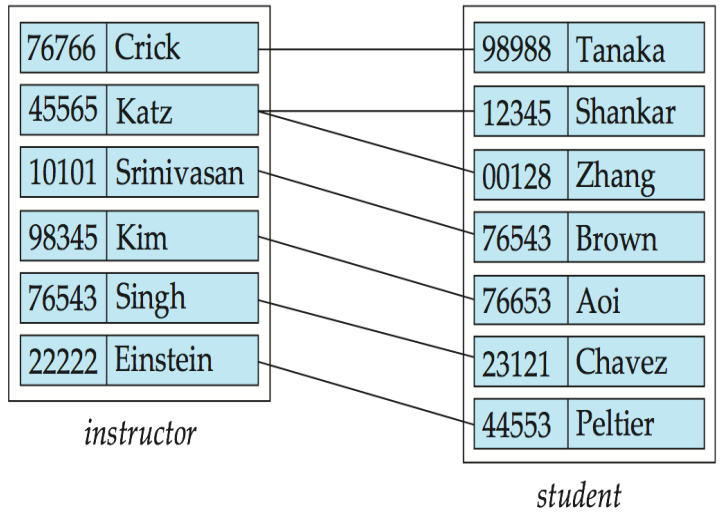
  student entity relationship set *instructor* entity
- A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

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

  where $(e_1, e_2, \ldots, e_n)$ is a relationship
  - Example:

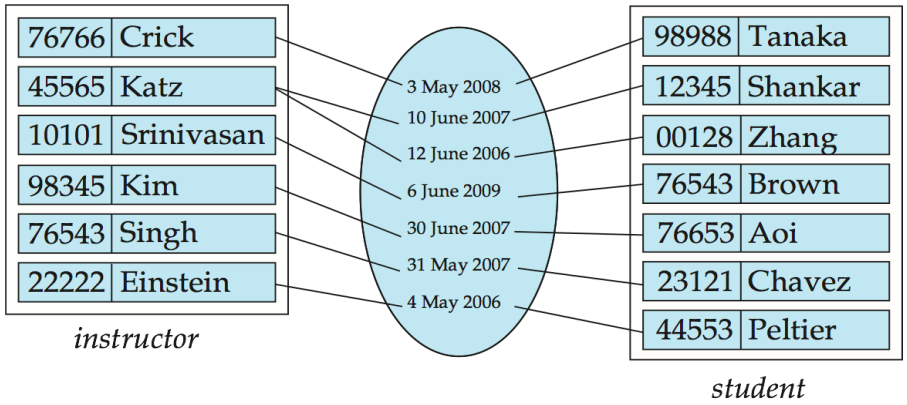
    $(44553,22222) \in advisor$

Relationship Set *advisor*



Relationship Sets (Cont.)

- An **attribute** can also be property of 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.



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*

Attributes

- An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set.
  - Example:

    instructor = (*ID, name, street, city, salary*)
    course= (*course_id, title, credits*)
- **Domain** – the set of permitted values for each attribute
- Attribute types:
  - **Simple** and **composite** attributes.

    ***Simple*:attributes that cannot be divided into subparts.**

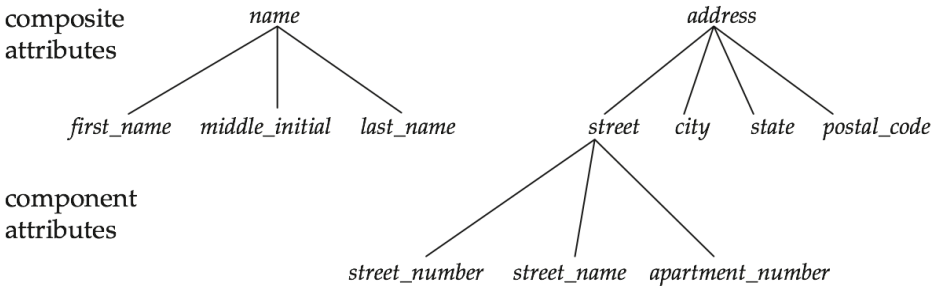
    Eg:ID

    ***Composite*:attributes that can be divided into subparts.**

    Eg:Name could be structured as first_name,middle_initial and last name.
  - **Single-valued** and **multivalued** attributes
  - Single valued:Attributes that have only single value for a particular entity.Eg:ID.
  - Multivalued:Attributes that have set of values for a specific entity.
    - Example: multivalued attribute: *phone_numbers*
  - **Derived attributes**
    - Can be computed from other attributes.
    - Example: age, given date_of_birth
    - Here **age** is the **derived attribute** and **date_of_birth** is **stored attribute.**

    ❖**Null attributes**
    - Attributes takes a null value when an entity does not have a value for it.

Composite Attributes



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

- **One to one**

An entity in A is associated with atmost one entity in B and an entity in B is associated with atmost one entity in A.

- **One to many**

An entity in A is associated with any number(zero or more) entities in B. An entity in B is associated with atmost one entity in A.

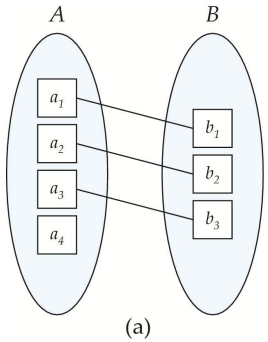
- **Many to one**

An entity in A is associated with atmost one entity in B. An entity in B is associated with any number(zero or more) entities in A.

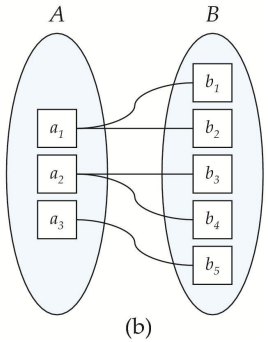
- **Many to many**

An entity in A is associated with any number(zero or more) entities in B. An entity in B is associated with any number(zero or more) entities in A.

Mapping Cardinalities

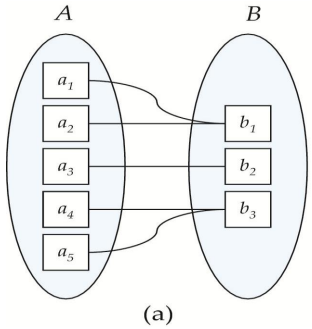


One to one

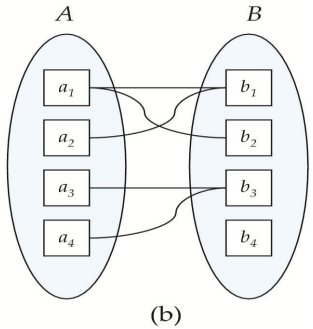


One to many

Mapping Cardinalities



Many to one



Many to many

Participation Contraints

- The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in atleast one relationship in R.
- The participation of an entity set E in a relationship set R is said to be partial if some entities in E participates in relationships in R.

Keys

- A **super key** of an entity set is a set of one or more attributes whose values uniquely determine each entity.
- A **candidate key** of an entity set is a minimal super key
  - *ID* is candidate key of *instructor*
  - *course_id* is candidate key of *course*
- Although several candidate keys may exist, one of the candidate keys is selected to be the **primary key**.

Keys for Relationship Sets

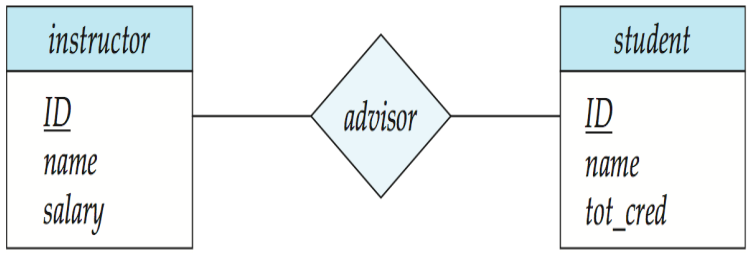
- The combination of primary keys of the participating entity sets forms a super key of a relationship set.
  - (*s_id, i_id*) is the super key of *advisor*
  - *NOTE: this means* ***a pair of entity sets can have at most one relationship in a particular relationship set.***
    - Example: if we wish to track multiple meeting dates between a student and her advisor, we cannot assume a relationship for each meeting. We can use a multivalued attribute though
- Must consider the mapping cardinality of the relationship set when deciding what are the candidate keys
- Need to consider semantics of relationship set in selecting the *primary key* in case of more than one candidate key

Redundant Attributes

- Suppose we have entity sets
  - *instructor*, with attributes including *dept_name*
  - *department*

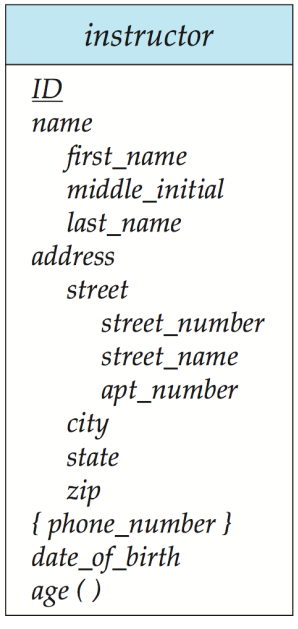
  and a relationship
  - *inst_dept* relating *instructor* and *department*
- Attribute *dept_name* in entity *instructor* is redundant since there is an explicit relationship *inst_dept* which relates instructors to departments
  - The attribute replicates information present in the relationship, and should be removed from *instructor*
  - BUT: when converting back to tables, in some cases the attribute gets reintroduced, as we will see.

E-R Diagrams

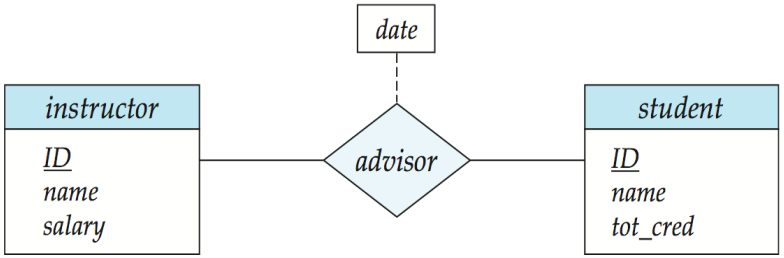


- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Attributes listed inside entity rectangle
- Underline indicates primary key attributes

Entity With Composite, Multivalued, and Derived Attributes



Relationship Sets with Attributes

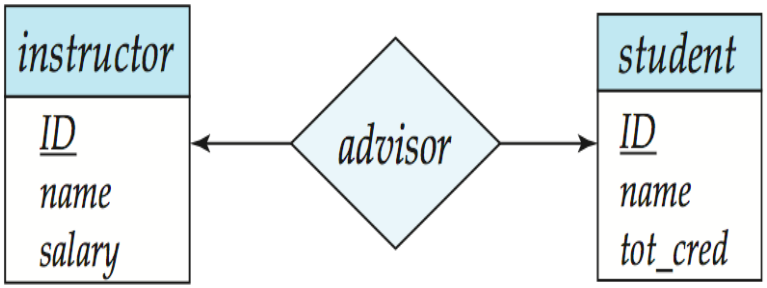


Cardinality Constraints

- We express cardinality constraints by drawing either a directed line (→), signifying "one," or an undirected line (—), signifying "many," between the relationship set and the entity set.

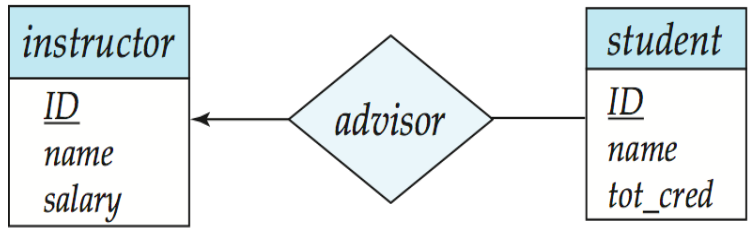
One-to-One Relationship

- one-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 at most one instructor via *advisor*



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,

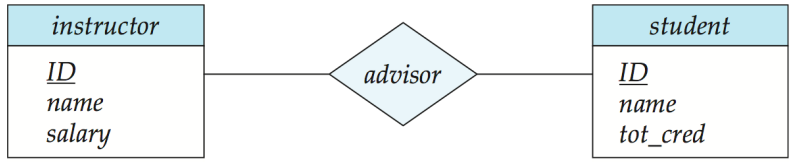


Many-to-One Relationships

- 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*

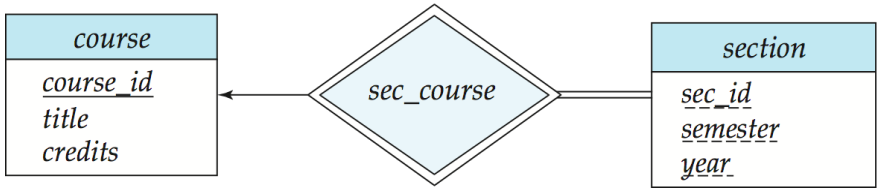


Participation of an Entity Set in a Relationship Set

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

  E.g., participation of *section* in *sec_course* is total
  - every *section* must have an associated course
- Partial participation: some entities may not participate in any relationship in the relationship set

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

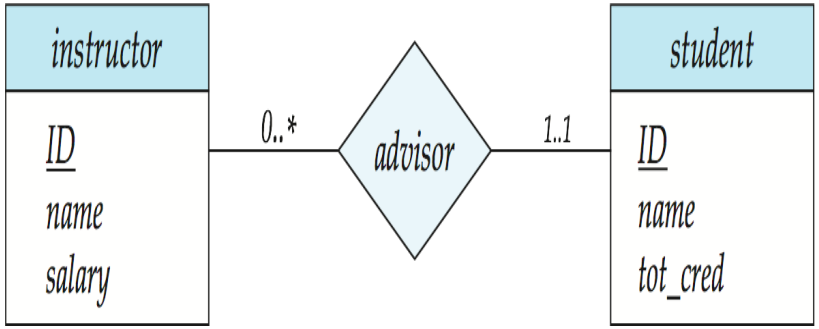


Notation for Expressing More Complex Constraints

- 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.

Roles

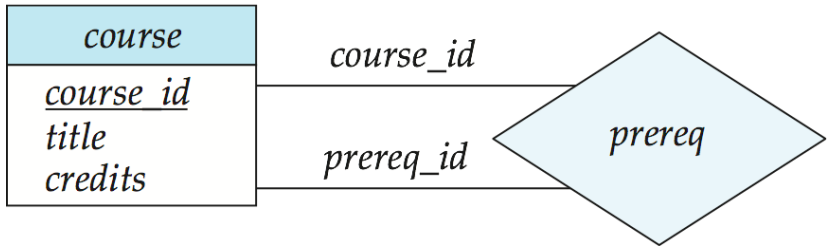
Instructor can advise 0 or more students. A student must have 1 advisor; cannot have multiple advisors



- Each entity type that participates in a relationship type plays a particular role in the relationship.
- The role name signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.
- For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.

- Role names are not necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name.
- In some cases the same entity type participates more than once in a relationship type in different roles.
- In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships.**

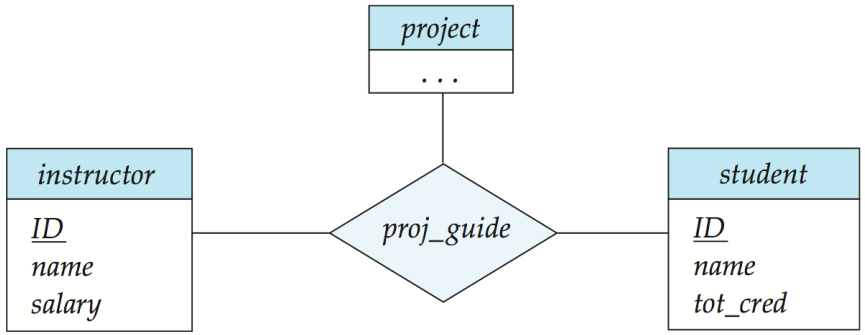
- The labels "*course_id*" and "*prereq_id*" are called **roles**.



Nonbinary Relationship Sets

- Nonbinary relationship sets can be specified easily in an E-R diagram.
- It consists of the three entity sets instructor, student, and project, related through the relationship set proj guide.
- We can specify some types of many-to-one relationships in the case of nonbinary relationship sets.
- Suppose a student can have at most one instructor as a guide on a project.
- This constraint can be specified by an arrow pointing to instructor on the edge from proj guide.

E-R Diagram with a Ternary Relationship



Weak Entity Sets

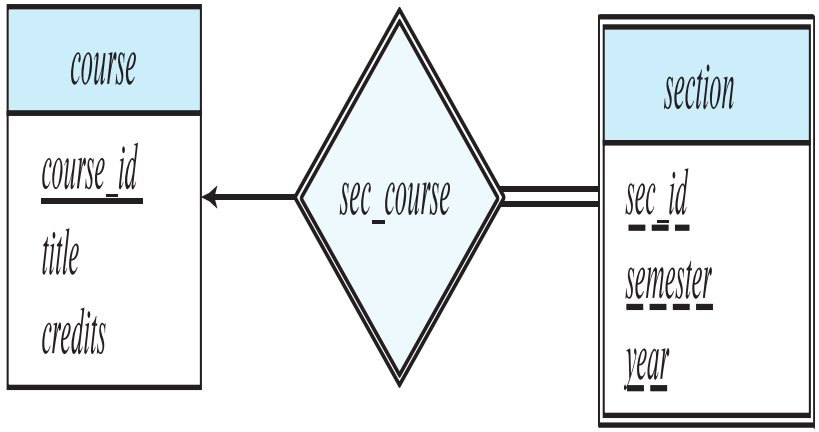
- An entity set that does not have a primary key is referred to as a **weak entity set**.
- The existence of a weak entity set depends on the existence of a **identifying entity set**
  - It must relate to the identifying entity set via a total, one-to-many relationship set from the identifying to the weak entity set
  - **Identifying relationship** depicted using a double diamond
- The **discriminator** (*or partial key)* of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.
- The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set's discriminator.

- 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**.

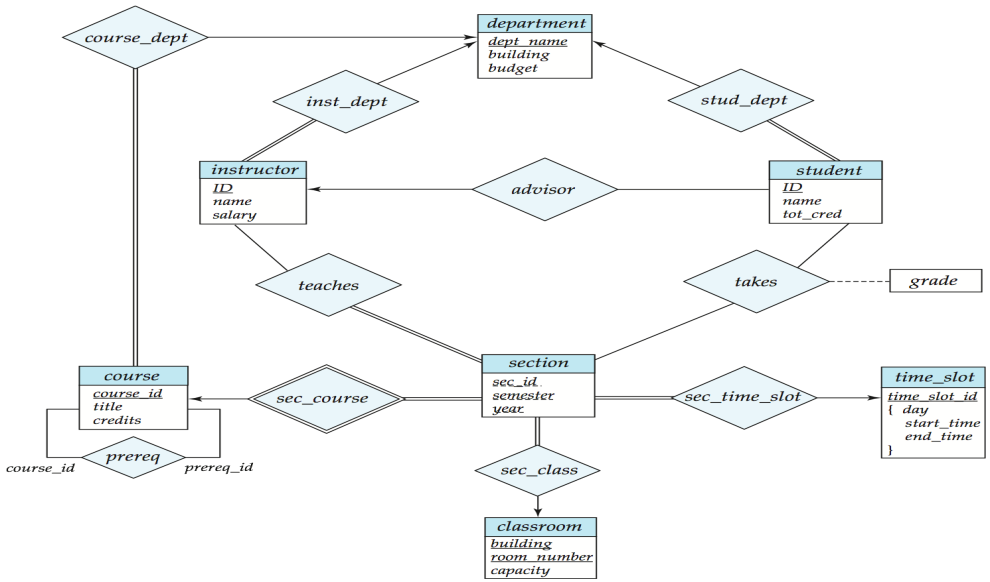
Weak Entity Sets (Cont.)

- 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*)

Weak Entity Sets (Cont.)



E-R Diagram for a University Enterprise



Extended E-R Features

- Although the basic E-R concepts can model most database features, some aspects of a database may be more aptly expressed by certain extensions to the basic E-R model.
- We discuss the extended E-R features of specialization, generalization, higher- and lower-level entity sets, attribute inheritance, and aggregation.

Specialization

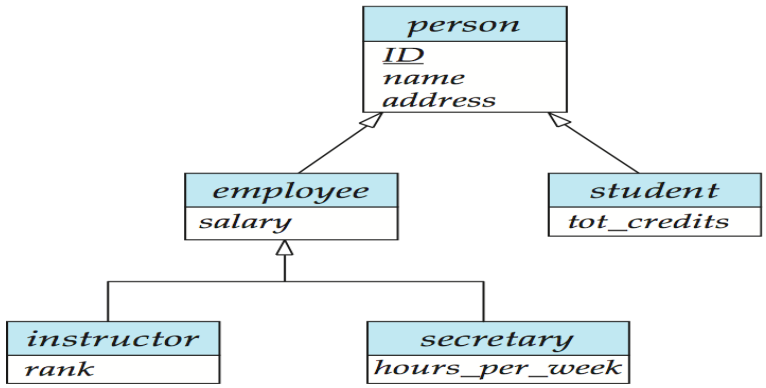
- An entity set may include subgroupings of entities that are distinct in some way from other entities in the set.
- For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set.
- The E-R model provides a means for representing these distinctive entity groupings.
- As an example, the entity set person may be further classified as one of the following:
- employee.
- student.

- Each of these person types is described by a set of attributes that includes all the attributes of entity set person plus possibly additional attributes.
- For example, employee entities may be described further by the attribute salary, whereas student entities may be described further by the attribute tot cred.
- The process of designating subgroupings within an entity set is called **specialization.**
- The specialization of person allows us to distinguish among person entities according to whether they correspond to employees or students: in general, a person could be an employee, a student, both, or neither.

- Suppose the university divides students into two categories: graduate and undergraduate.
- Graduate students have an office assigned to them. Undergraduate students are assigned to a residential college.
- Each of these student types is described by a set of attributes that includes all the attributes of the entity set student plus additional attributes.
- The university could create two specializations of student, namely graduate and undergraduate.
- Student entities are described by the attributes ID, name, address, and tot cred.
- The entity set graduate would have all the attributes of student and an additional attribute office number.
- The entity set undergraduate would have all the attributes of student, and an additional attribute residential college

- We can apply specialization repeatedly to refine a design. For instance, university employees may be further classified as one of the following:
- instructor.
- secretary
- Each of these employee types is described by a set of attributes that includes all the attributes of entity set employee plus additional attributes.
- For example, instructor entities may be described further by the attribute rank while secretary entities are described by the attribute hours per week.
- Further,secretary entities may participate in a relationship secretary for between the secretary and employee entity sets, which identifies the employees who are assisted by a secretary.

Specialization Example



Extended ER Features: 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.

Specialization and Generalization (Cont.)

- Can have multiple specializations of an entity set based on different features.
- E.g., *permanent_employee* vs. *temporary_employee*, in addition to *instructor* vs. *secretary*
- Each particular employee would be
  - a member of one of *permanent_employee* or *temporary_employee*,
  - and also a member of one of *instructor*, *secretary*
- The ISA relationship also referred to as **superclass - subclass** relationship

Attribute Inheritance

- A crucial property of the higher- and lower-level entities created by specialization and generalization is **attribute inheritance.**
- The attributes of the higher-level entity sets are said to be inherited by the lower-level entity sets.
- For example, student and employee inherit the attributes of person.
- Thus, student is described by its ID, name, and address attributes, and additionally a tot cred attribute; employee is described by its ID, name, and address attributes, and additionally a salary attribute.
- Attribute inheritance applies through all tiers of lower-level entity sets; thus, instructor and secretary, which are subclasses of employee, inherit the attributes ID, name, and address from person, in addition to inheriting salary from employee.
- A lower-level entity set (or subclass) also inherits participation in the relationship sets in which its higher-level entity (or superclass) participates.
- Like attribute inheritance, participation inheritance applies through all tiers of lower-level entity sets.

Design Constraints on a Specialization/Generalization

- One type of constraint involves determining which entities can be members of a given lower-level entity set. Such membership may be one of the following:

- **Condition-defined**
- In condition-defined lower-level entity sets, membership is evaluated on the basis of whether or not an entity satisfies an explicit condition or predicate.
- For example, assume that the higher-level entity set student has the attribute student type.
- All student entities are evaluated on the defining student type attribute. Only those entities that satisfy the condition student type = "graduate" are allowed to belong to the lower-level entity set graduate student.
- All entities that satisfy the condition student type = "undergraduate" are included in undergraduate student.
- Since all the lower-level entities are evaluated on the basis of the same attribute (in this case, on student type), this type of generalization is said to be **attribute-defined.**

Design Constraints on a Specialization/Generalization

- **User-defined**
- User-defined lower-level entity sets are not constrained by a membership condition; rather, the database user assigns entities to a given entity set.
- For instance, let us assume that, after 3 months of employment, university employees are assigned to one of four work teams.
- We therefore represent the teams as four lower-level entity sets of the higher-level employee entity set.
- A given employee is not assigned to a specific team entity automatically on the basis of an explicit defining condition.
- Instead, the user in charge of this decision makes the team assignment on an individual basis. The assignment is implemented by an operation that adds an entity to an entity sets.

- A second type of constraint relates to whether or not entities may belong to more than one lower-level entity set within a single generalization. The lowerlevel entity sets may be one of the following:

- **Disjoint**
- A disjointness constraint requires that an entity belong to no more than one lower-level entity set.
- In our example, student entity can satisfy only one condition for the student type attribute; an entity can be either a graduate student or an undergraduate student, but cannot be both.

- **Overlapping**

In overlapping generalizations, the same entity may belong to more than one lower-level entity set within a single generalization.

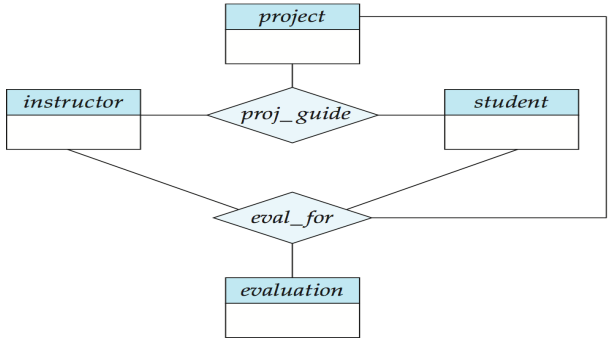
- consider the employee work-team example, and assume that certain employees participate in more than one work team.
- A given employee may therefore appear in more than one of the team entity sets that are lowerlevel entity sets of employee.
- Thus, the generalization is **overlapping.**

Design Constraints on a Specialization/Generalization (Cont.)

- **Completeness constraint** -- specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization.
  - **total**: an entity must belong to one of the lower-level entity sets
  - **partial**: an entity need not belong to one of the lower-level entity sets

Aggregation

- One limitation of the E-R model is that it cannot express relationships among relationships.
- 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

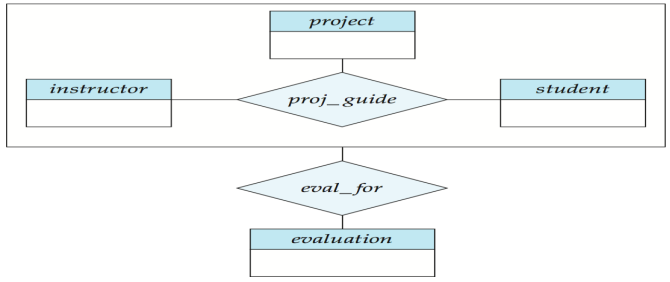


Aggregation (Cont.)

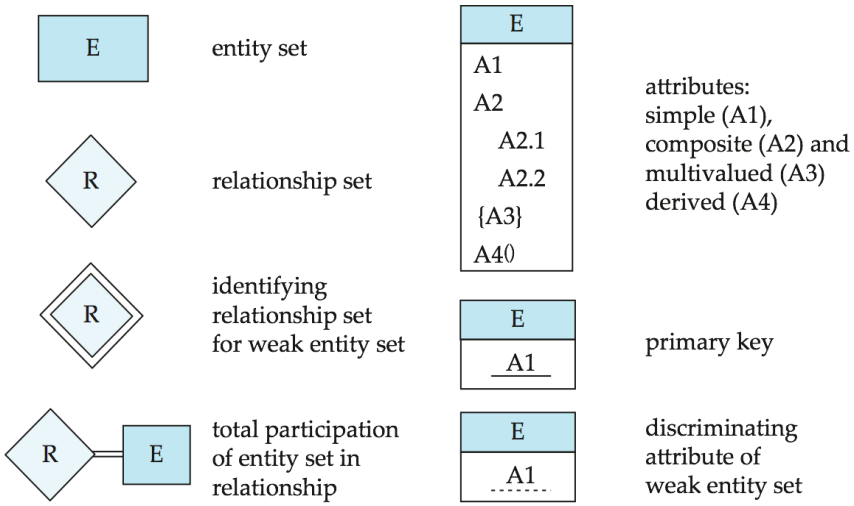
- Aggregation is an abstraction through which relationships are treated as higher-level entities.
- Relationship sets *eval_for* and *proj_guide* represent overlapping information
  - Every *eval_for* relationship corresponds to a *proj_guide* relationship
  - However, some *proj_guide* relationships 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.)

- 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



Summary of Symbols Used in E-R Notation



Symbols Used in E-R Notation (Cont.)