Unit 1: Introduction to Database

Chapter 1: Introduction to Database

- Purpose of Database Systems,
- View of Data, Data Models,
- Database Architecture,
- Roles in Database Environment,
- The Entity-Relationship Model,
- Entity-Relationship Diagrams,
- Reduction to Relational Schemas,
- Introduction to Relational Model,
- Relational Query Languages- The Relational Algebra,

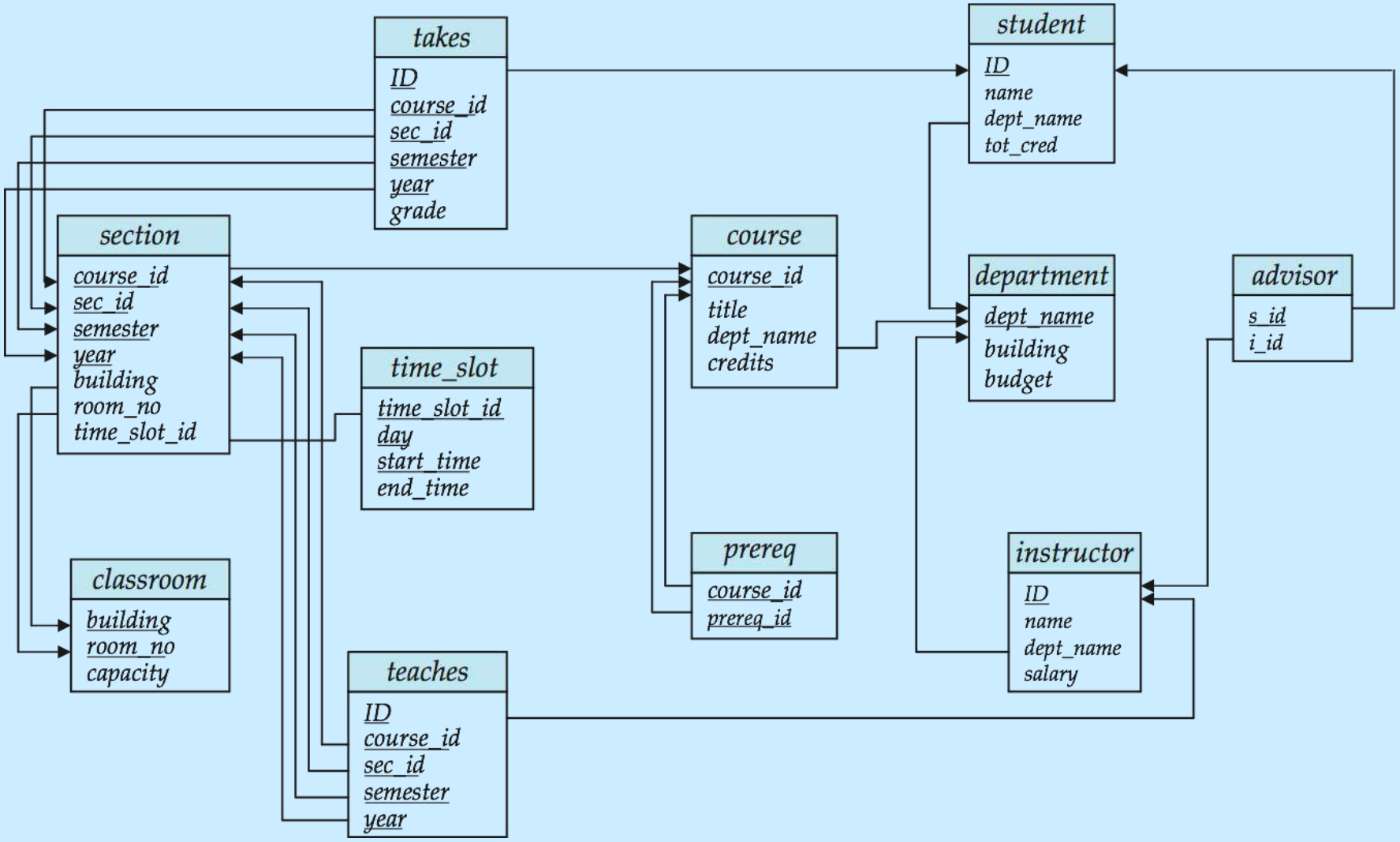
Introduction

- A **database** to be a collection of related data.
- The ***Database Management System (DBMS)*** is a Collection of interrelated data and Set of programs to access the data.
- The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient and efficient.*
- A ***database application*** is simply a program that interacts with the database at some point in its execution.
- ***Database system*** to be a collection of application programs that interact with the database along with the DBMS and database itself (access the data).

Database Management System (DBMS)

- 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 Applications:
  - Banking: 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
- Databases can be very large.
- Databases touch all aspects of our lives

Schema Diagram for University Database



Traditional File-Based Systems

- As an example of such methods consider part of a university organization that, among other data, keeps information about all instructors, students, departments, and course offerings.
- One way to keep the information on a computer is to store it in operating system files.
- Application program examples
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts
- This typical **file-processing system is supported by a conventional operating** system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files.

Drawbacks of using file systems to store data

- Data redundancy and inconsistency
  - Multiple file formats, duplication of information in different files
- Difficulty in accessing data
  - Need to write a new program to carry out each new task
- Data isolation — multiple files and formats
- Integrity problems
  - Integrity constraints (e.g., account balance > 0) become "buried" in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones

Drawbacks of using file systems to store data (Cont.)

- Atomicity of updates
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not happen at all
- Concurrent access by multiple users
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
- Security problems
  - Hard to provide user access to some, but not all, data

Database systems offer solutions to all the above problems

Levels of Abstraction

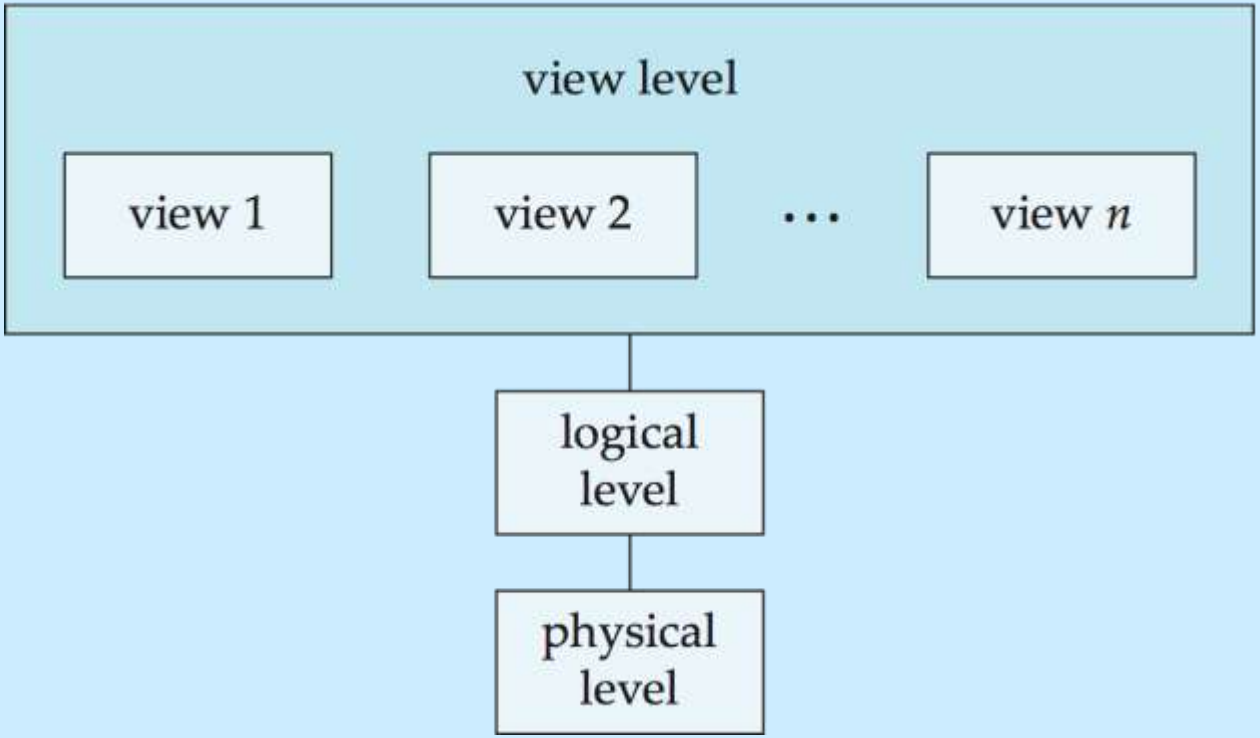
- The ***Database Management System (DBMS)*** is a Collection of interrelated data and Set of programs to access and modify these data.
- A major purpose of a database system is to provide users with an *abstract view of the data.*
- **Physical level:** describes **how** a record (e.g., customer) is stored.

```
type instructor = record
        ID : string;
        name : string;
        dept_name : string;
        salary : integer;
            end;
```

- **Logical level:** describes **what** data stored in database, and the relationships among the data. Handled by database administrator.
- **View level:** Highest level of abstraction. Application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

View of Data

An architecture for a database system



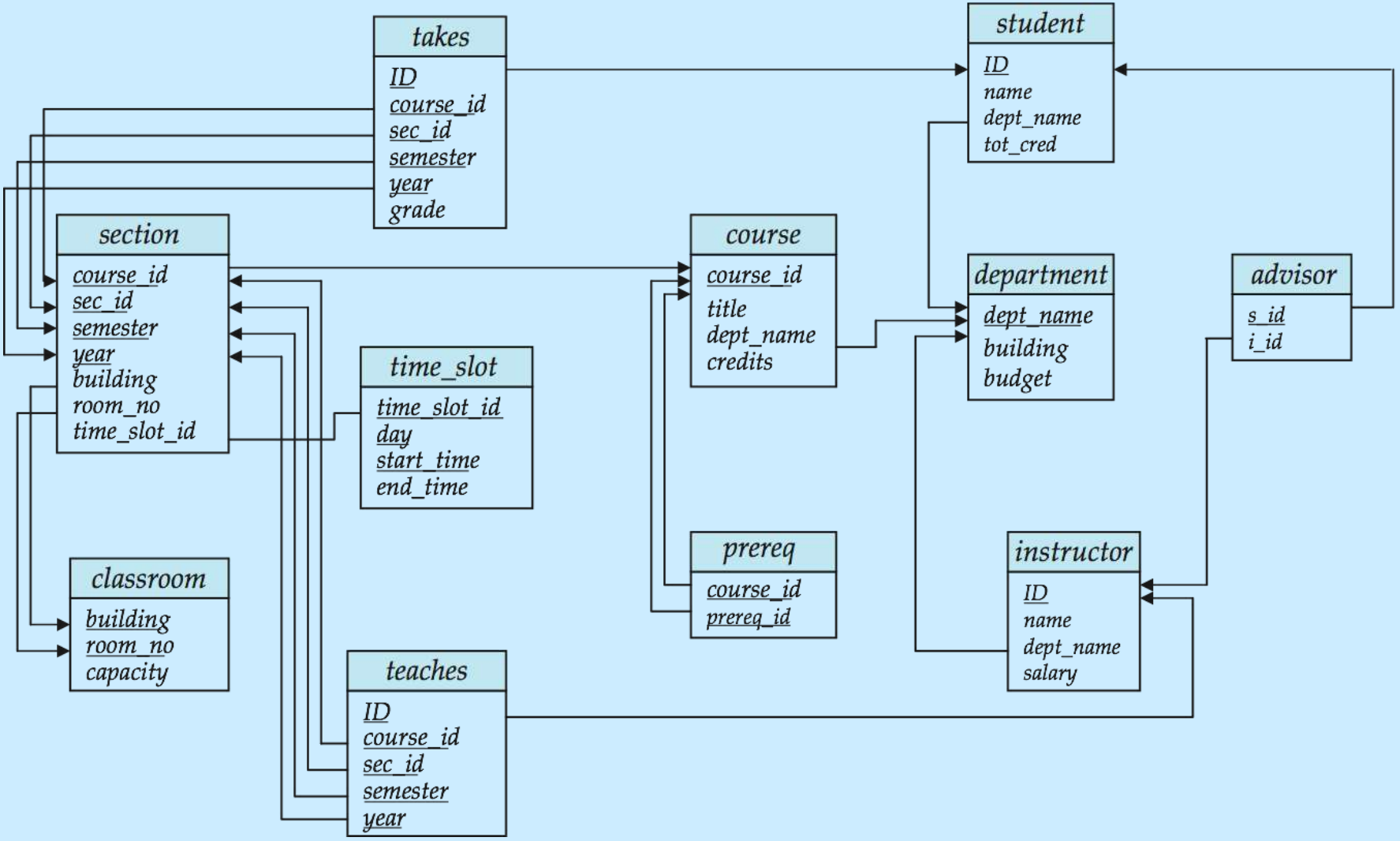
Views have several other benefits:

- *Views provide a level of security.*
- *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*

Instances and Schemas

- **Schema** – the logical structure of the database (Overall design)
  - Example: The database consists of information about a set of customers and accounts and the relationship between them
  - **Physical schema**: database design at the physical level
  - **Logical schema**: database design at the logical level
  - **Subschemas**: A database may have several schemas at view level.
- **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.

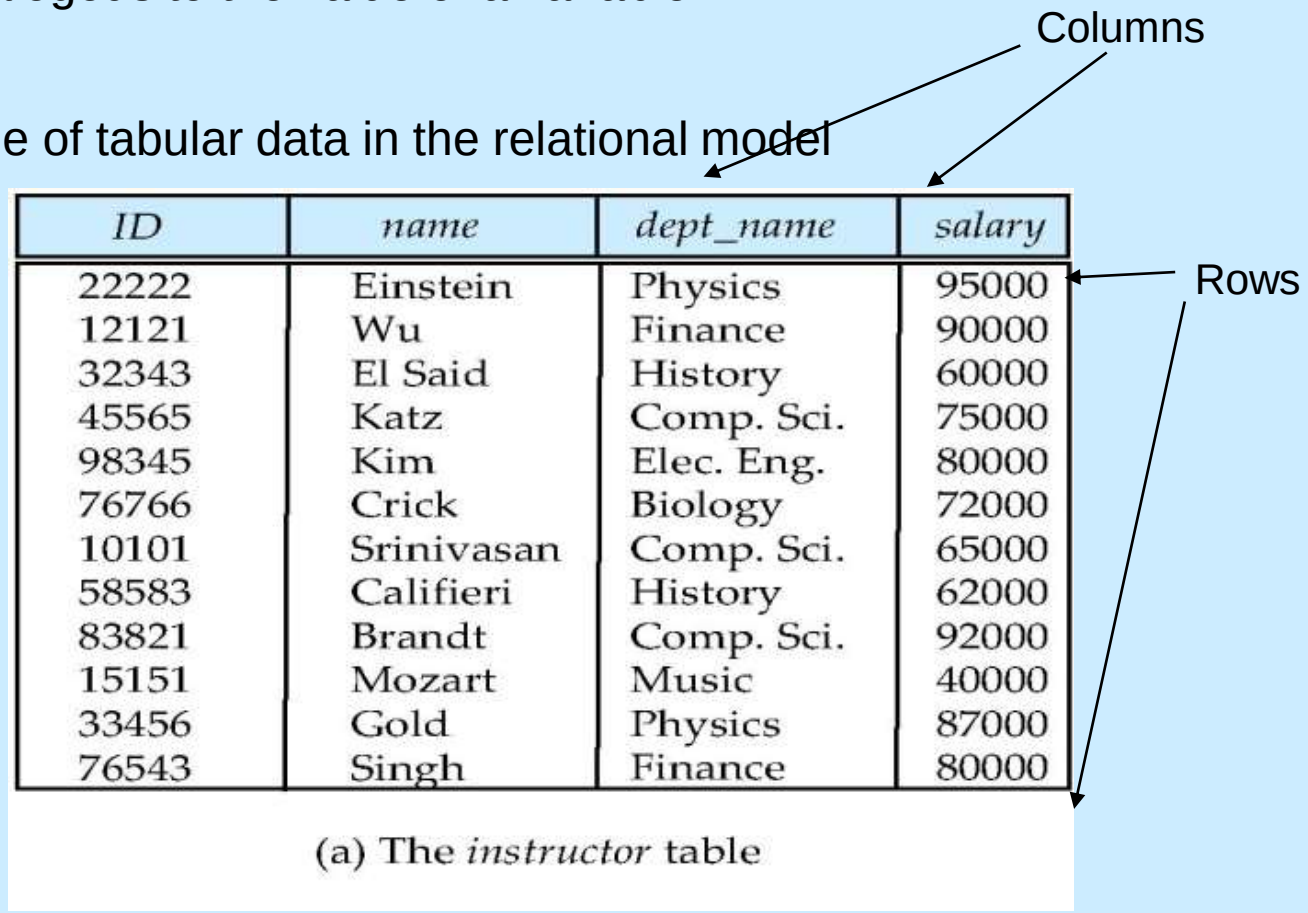
Schema Diagram for University Database



Instances and Schema

- **Instance** – Database changes over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database.
  - Analogous to the value of a variable

- Example of tabular data in the relational model



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

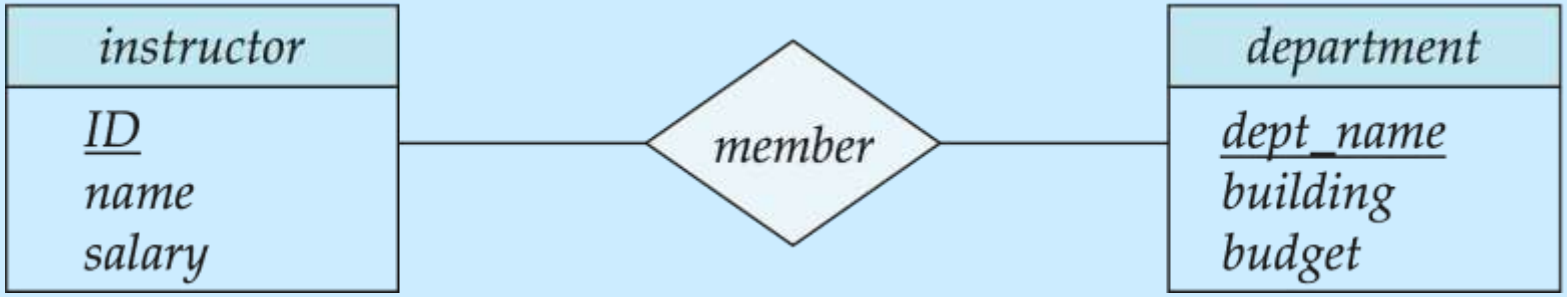
(a) The *instructor* table

Data Models

- A collection of conceptual tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Entity-Relationship data model (mainly for database design)
- Relational model
- Object-based data models (Object-oriented and Object-relational)
- Semistructured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

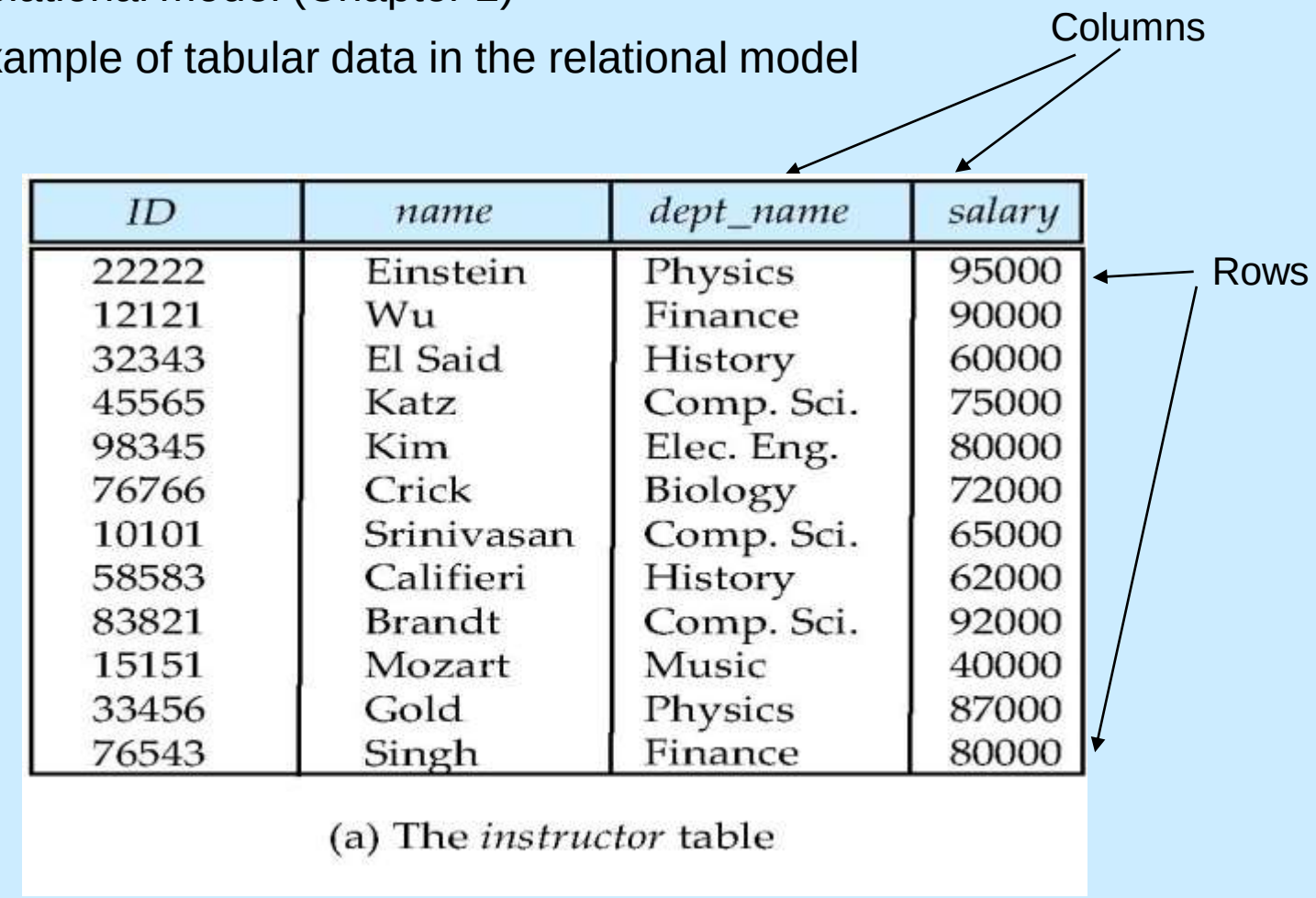
The Entity-Relationship Model

- Models an enterprise as a collection of *entities* and *relationships*
  - Entity: a “thing” or “object” in the real world 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:*



Relational Model

- Relational model (Chapter 2)
- Example of tabular data in the relational model



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

(a) The *instructor* table

Object-Relational Data Models

- Relational model: flat, “atomic” values
- Object Relational Data Models
  - Extend the relational data model by including object orientation and constructs to deal with added data types.
  - Allow attributes of tuples to have complex types, including non-atomic values such as nested relations.
  - Preserve relational foundations, in particular the declarative access to data, while extending modeling power.
  - Provide upward compatibility with existing relational languages.

XML: Extensible Markup Language

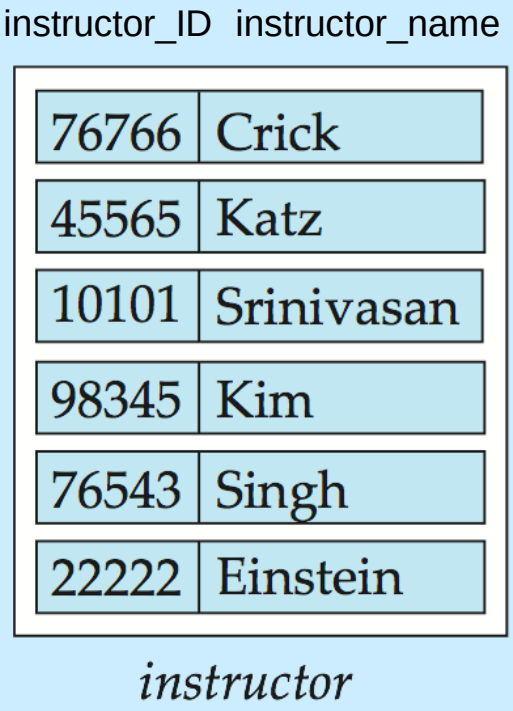
- Defined by the WWW Consortium (W3C)
- Originally intended as a document markup language not a database language
- The ability to specify new tags, and to create nested tag structures made XML a great way to exchange **data**, not just documents
- XML has become the basis for all new generation data interchange formats.
- A wide variety of tools is available for parsing, browsing and querying XML documents/data

Entity-Relationship Model

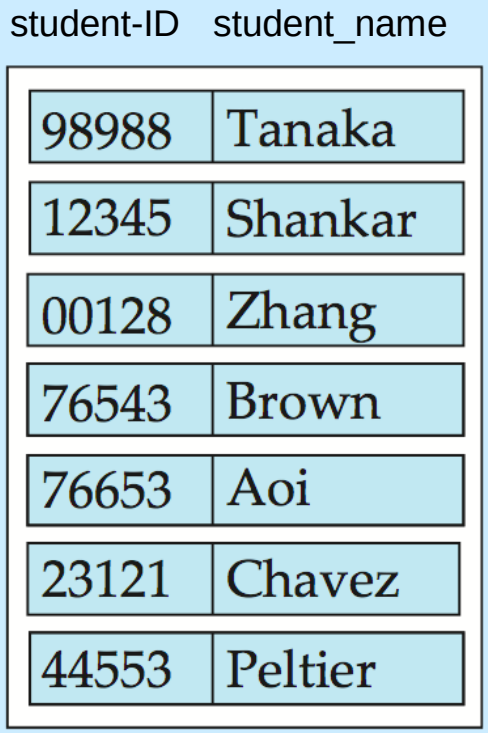
Entity-Relationship Model

- A *database* can be modeled as:
  - a collection of entities,
  - relationship among entities.
- An **entity** is a thing" or "object" in the real world that 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 or attributes.
  - Example: set of all persons, companies, trees, students

Entity Sets *instructor* and *student*



instructor



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 set of relationships of the same type.
  Mathematical relation among $n \geq 2$ entities, each taken from entity sets

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

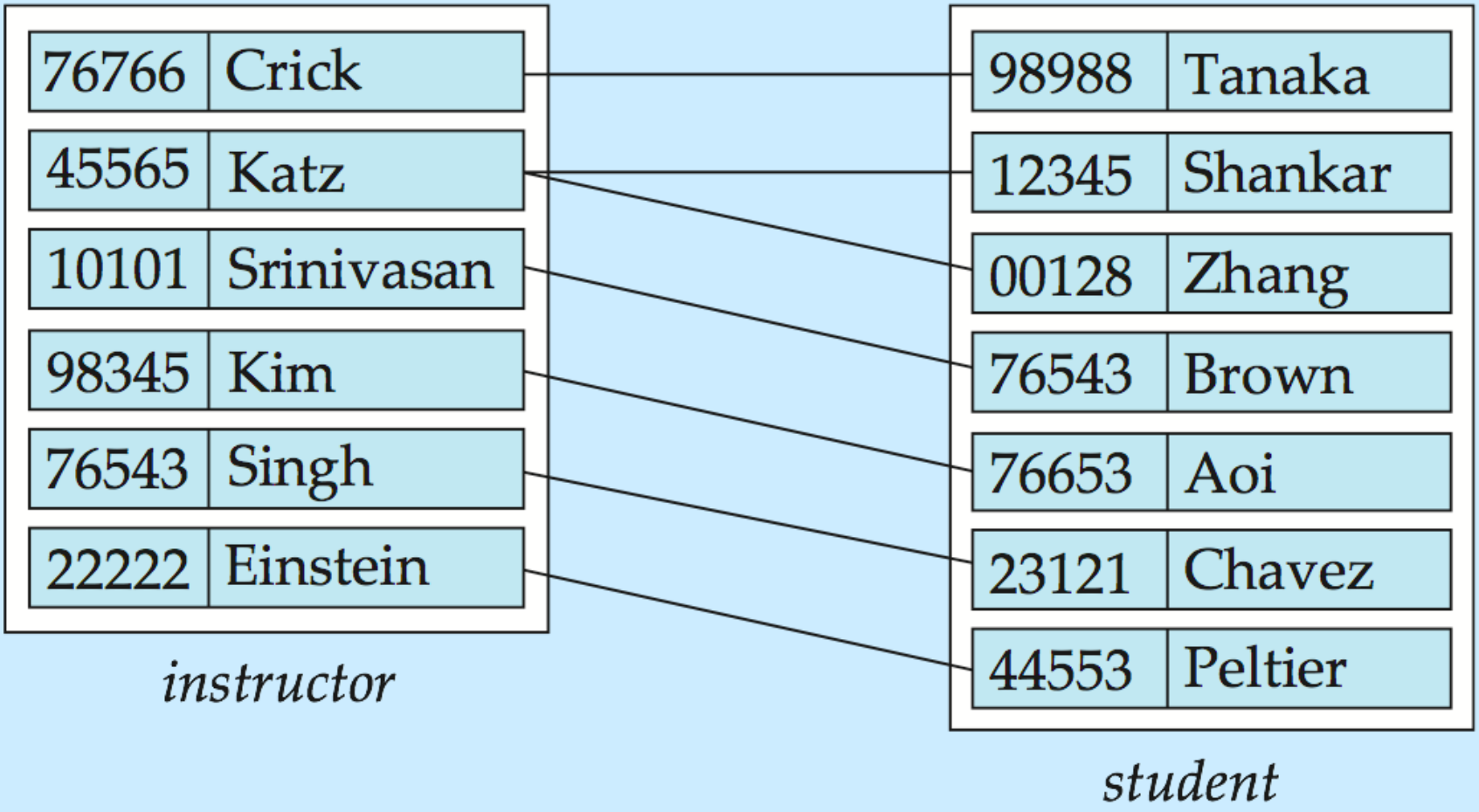
  where $(e_1, e_2, \ldots, e_n)$ is a relationship R.

  - Example:

    $(44553,22222) \in advisor$

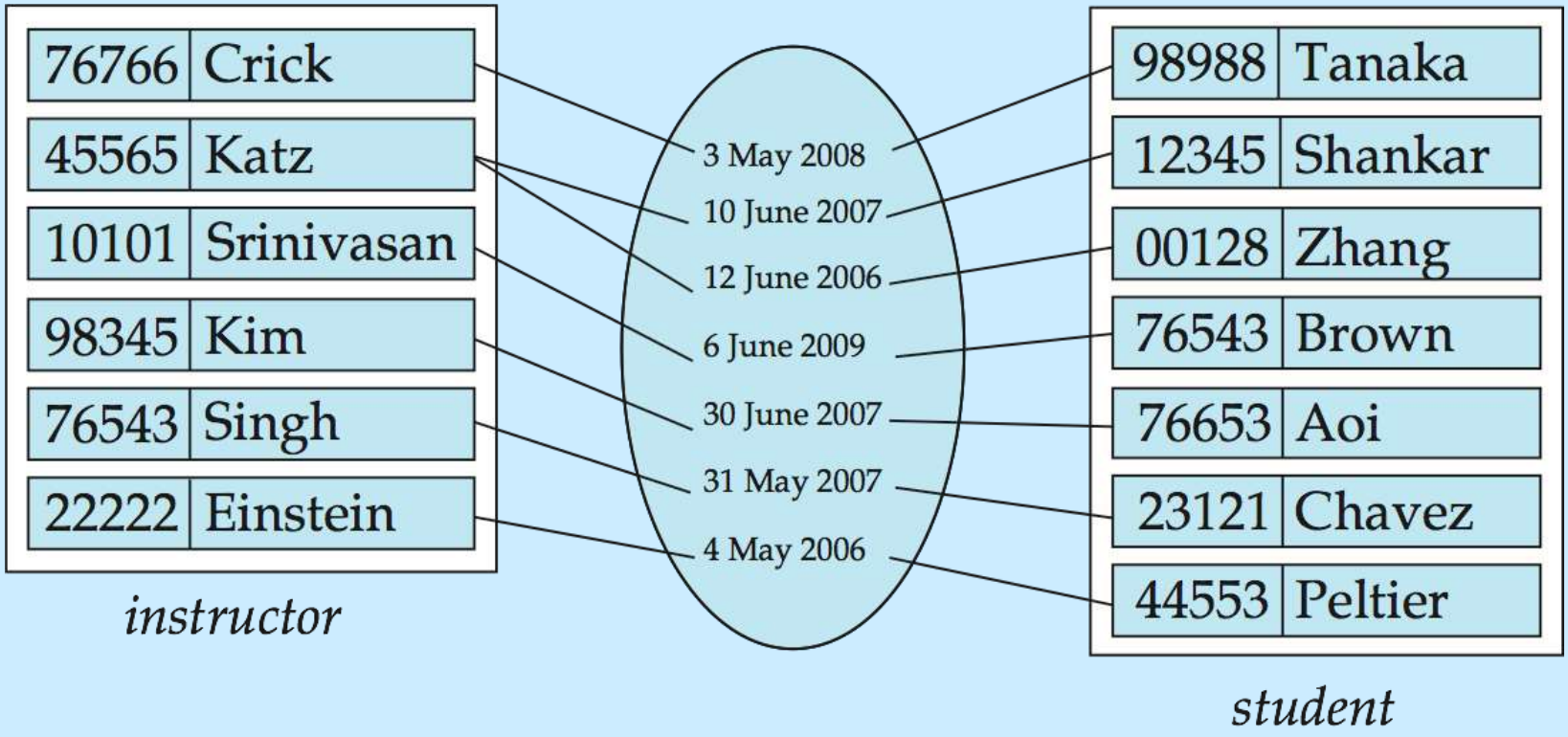
  The association between entity sets is referred to as **participation**;
  that is, the entity sets E1, E2, …, En participate in relationship set R.

Relationship Set *advisor*



Relationship Sets (Cont.)

- A relationship may also have its own attributes call **descriptive** attributes.
- 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

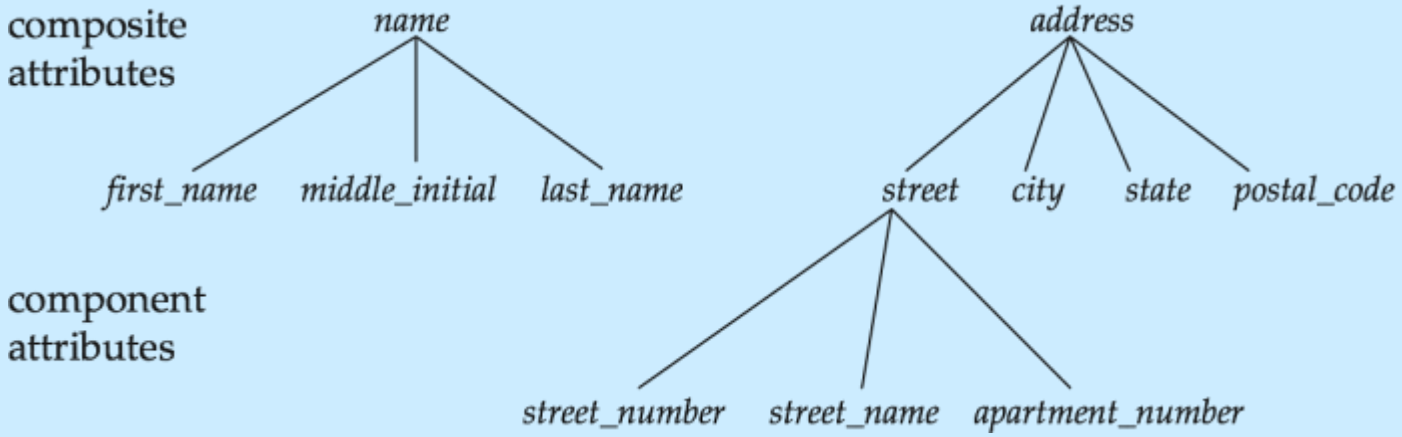
- **Binary relationship:**
  - involve two entity sets (or degree two).
  - most relationship sets in a database system are binary.
- Relationships between more than two entity sets are rare. Most relationships are binary. (More on this later.)
- **Ternary Relationship:**
  - Example: *students* work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

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.



Attributes

- Attribute types (Continue…):
  - **Single-valued** and **multivalued** attributes
    - Example: multivalued attribute: *phone_numbers*
  - **Derived** attributes
    - Can be computed from other attributes
    - Example: age, given date_of_birth

  Null attributes: Either missing or not known

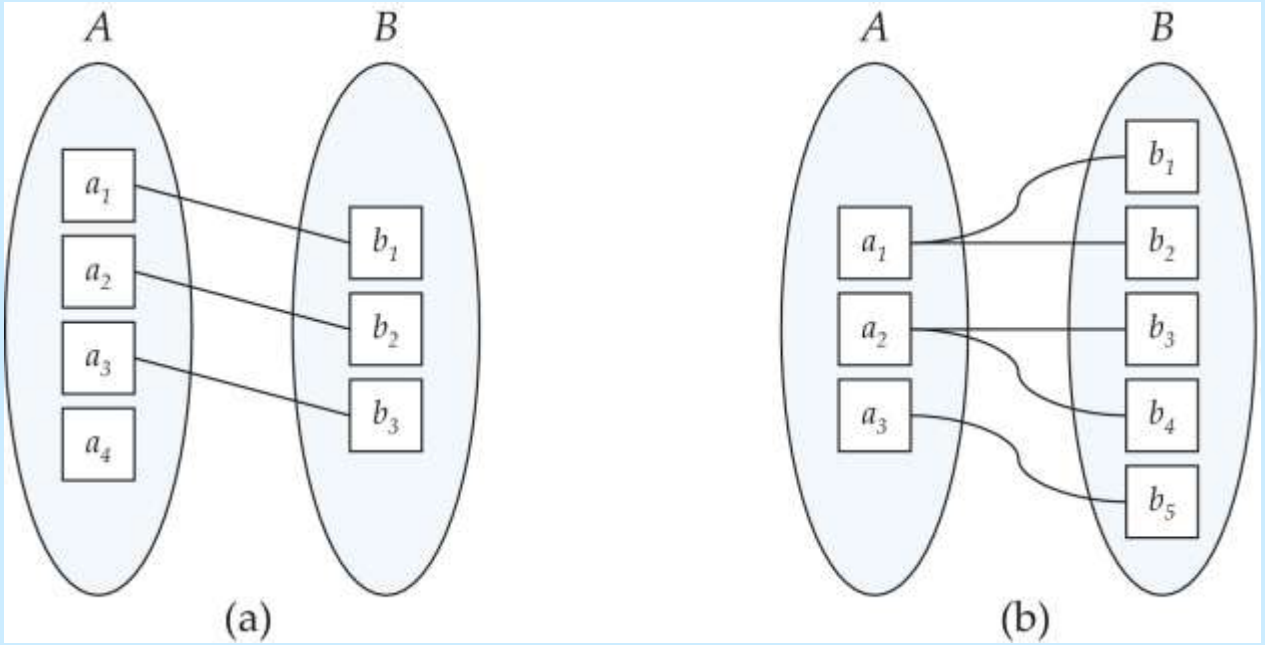
Constraints

- An E-R enterprise schema may define certain constraints to which the contents of database must conform.
- There are two types of constraints:
  - Mapping Cardinality
  - Participation Constraint.

Mapping Cardinality Constraints

- **Mapping cardinality or cardinality ratio,** 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

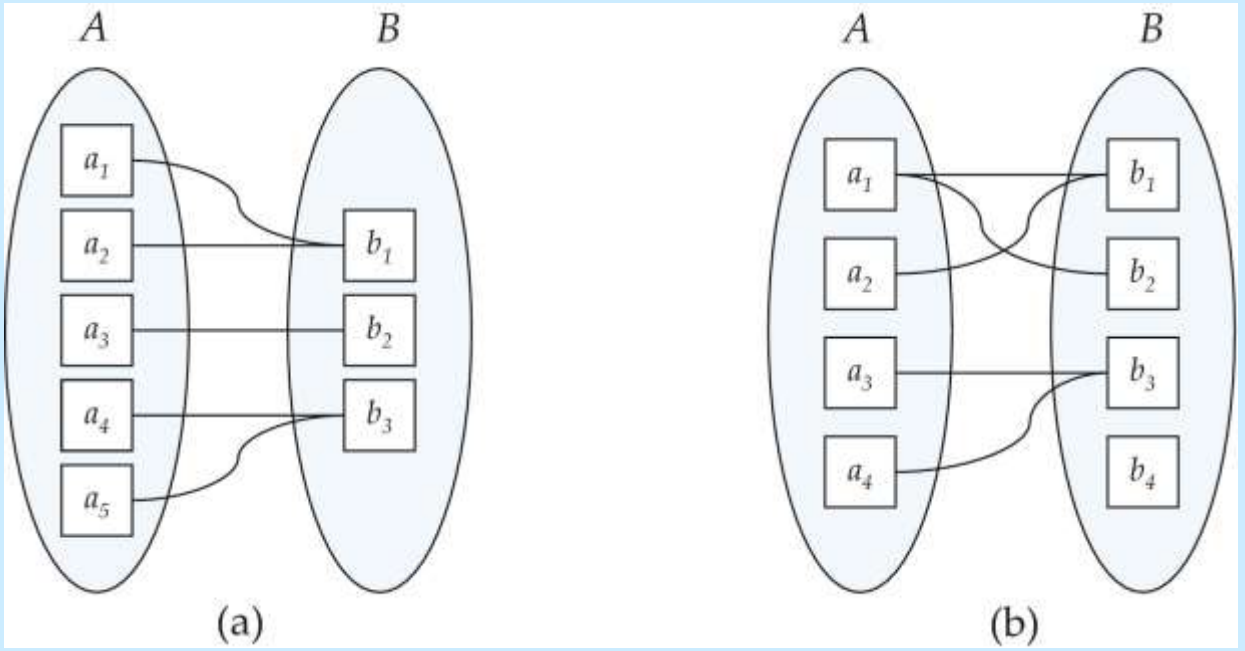


One to one

One to many

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

Mapping Cardinalities



Many to one

Many to many

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

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 *student* in *advisor* is total
    - every student must have a instructor associated to it via advisor.
- Partial participation: If only some of entities in E participate in any relationship R in the relationship set
  - E.g. participation of *instructor* in *advisor* is partial

Keys

- A key is a set of attributes that sufficient to distinguish entities from each other.
- A *super key* of an entity set is a set of one or more attributes whose values uniquely identify each entity.
  - (*Id, name*) is the super key of *instructor.*

  A super key may contain extraneous attributes.
- A *candidate key* of an entity set is a minimal super key
  - *Id* is candidate key of *instructor*

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

- Although several candidate keys may exist, one of the candidate keys is selected to be the *primary key*.
- Primary key is a candidate key, chosen by the database designer to identify entities within an entity set.
- The primary key should be chosen such that its attributes are never, or very rarely changed and its value should be unique.

Keys

- A relation, say r1, may include among its attributes the primary key of another relation, say r2. This attribute is called a **foreign key from r1, referencing r2.**
- The relation r1 is also called the **referencing** relation of the foreign key dependency, and r2 is called the **referenced relation** of the foreign key.
- For example, the attribute dept_name in instructor is a foreign key from instructor, referencing department, since dept_name is the primary key of department.
  - instructor: with attributes (i_*ID*, *name,* ***dept_name,*** *salary).*
  - department: with attributes (***dept_name****, building, budget).*

Keys for Relationship Sets

- Let R be a relationship set involving entity sets E1, E2, . . . , En. Let primary key(Ei) denote the set of attributes that forms the primary key for entity set Ei .
- If the relationship set *R has no attributes associated with it, then the set of* attributes

$$primary\text{-}key(E_1) \cup primary\text{-}key(E_2) \cup \cdots \cup primary\text{-}key(E_n)$$

describes an individual relationship in set *R.*

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

$$primary\text{-}key(E_1) \cup primary\text{-}key(E_2) \cup \cdots \cup primary\text{-}key(E_n) \cup \{a_1, a_2, \ldots, a_m\}$$

describes an individual relationship in set *R.*

- In both of the above cases, the set of attributes

$$primary\text{-}key(E_1) \cup primary\text{-}key(E_2) \cup \cdots \cup primary\text{-}key(E_n)$$

forms a superkey for the relationship set.

Keys for Relationship Sets

- The structure of the primary key for the relationship set depends on the mapping cardinality of the relationship .
- ***Ex:*** consider the entity sets instructor and student, and the relationship set advisor, with attribute date.

1. Many-to-many from instructor to student : the primary key of *advisor consists of the union of the primary keys of instructor and student.*
2. One-to-many from instructor to student : the primary key of *advisor is* simply the primary key of *student.*
3. Many-to-one from instructor to student : the primary key of *advisor is* simply the primary key of instructor.
4. One-to-one from instructor to student : the primary key of *advisor is either* primary key of *student or* primary key of instructor.

University Database

- The entity sets and their attributes below, with primary keys underlined:
  - classroom: with attributes (*building, room number, capacity).*
  - department: with attributes (*dept_name, building, budget).*
  - course: with attributes (*course_id, title, credits).*
  - instructor: with attributes (i_*ID, name, salary).*
  - section: with attributes (*course id, sec_id, semester, year).*
  - student: with attributes (*s_ID, name, tot_cred).*
  - Time_slot: with attributes (*time_slot_id, day, start_time, end_time).*

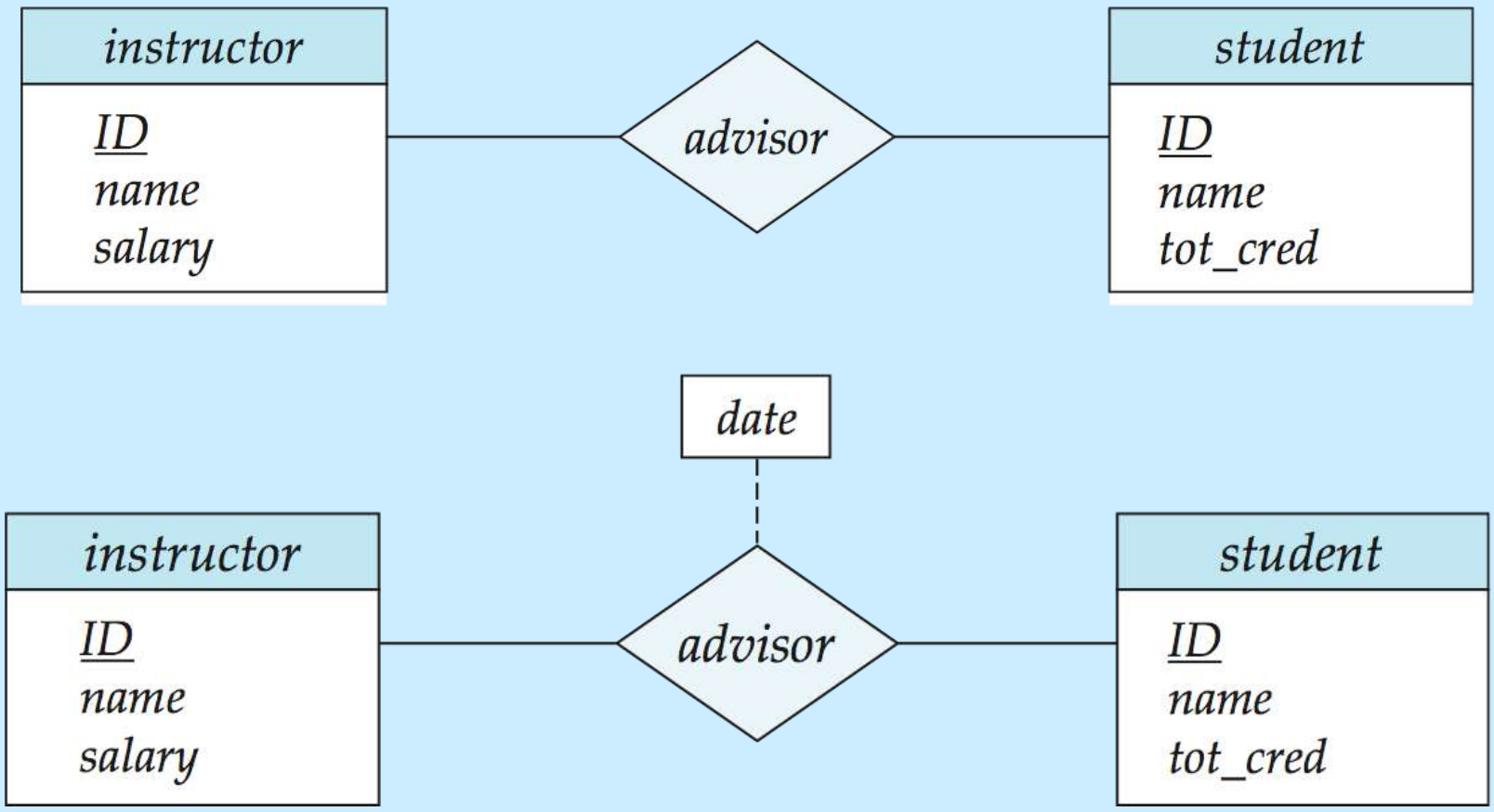
University Database

- The relationship sets in our design are listed below:
  - *teaches (i_ID, course id, sec id, semester, year)*
  - *takes (s_ID, course id, sec id, semester, year, grade)*
  - *prereq (course id, prereq id)*
  - *advisor (s ID, i ID)*
  - *sec_course (course id, sec id, semester, year)*
  - *sec_time_slot (course id, sec id, semester, year, time slot id)*
  - *sec_class (course id, sec id, semester, year, building, room number)*
  - *inst_dept (i_ID, dept_name)*
  - *stud_dept (s_ID, dept_name)*
  - *course_dept (course id, dept_name)*.

E-R Diagrams

- An **E-R diagram can express the overall logical** structure of a database graphically
- An E-R diagram consists of the following major components:
  - **Rectangles divided into two parts represent entity sets.** The first part, contains the name of the entity set. The second part contains the names of all the attributes of the entity set.
  - **Diamonds** represent relationship sets**.**
  - **Undivided rectangles** represent the attributes of a relationship set.
  - **Attributes** that are part of the primary key are underlined.
  - **Lines** link entity sets to relationship sets**.**
  - **Dashed** lines link attributes of a relationship set to the relationship set.
  - **Double lines** indicate total participation of an entity in a relationship set.
  - **Double diamonds** represent identifying relationship sets linked to weak entity sets

E-R Diagrams


instructor
ID
name
salary
advisor
student
ID
name
tot_cred
date
instructor
ID
name
salary
advisor
student
ID
name
tot_cred

Cardinality Constraints

- The relationship set *advisor, between the instructor and student entity sets may be* one-to-one, one-to-many, many-to-one, or many-to-many.
- 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

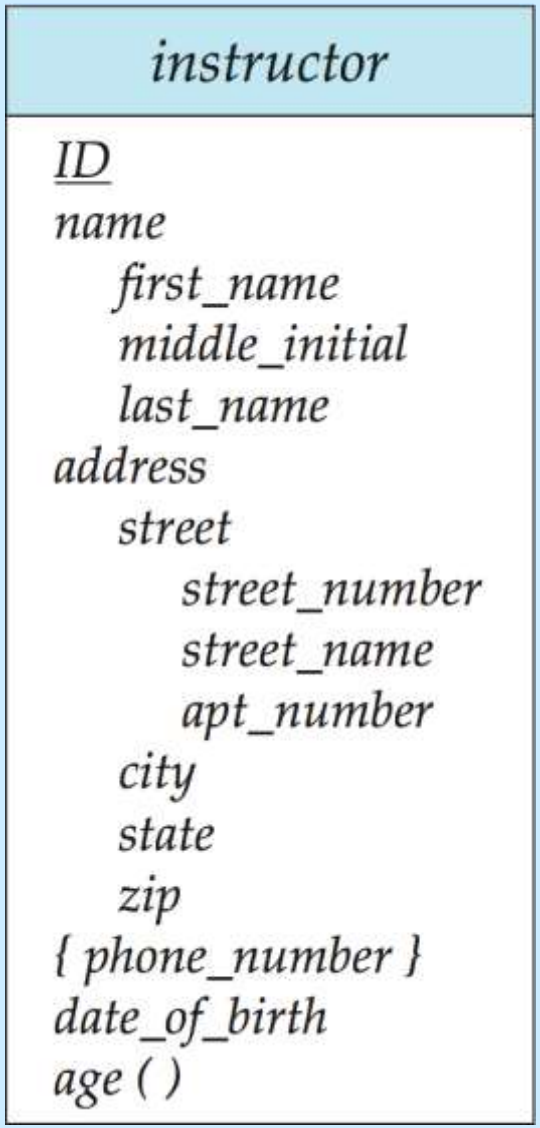
- An instructor is associated with several (possibly 0) students via *advisor*
- A student is associated with several (possibly 0) instructors via *advisor*

Alternative Notation for Cardinality Limits

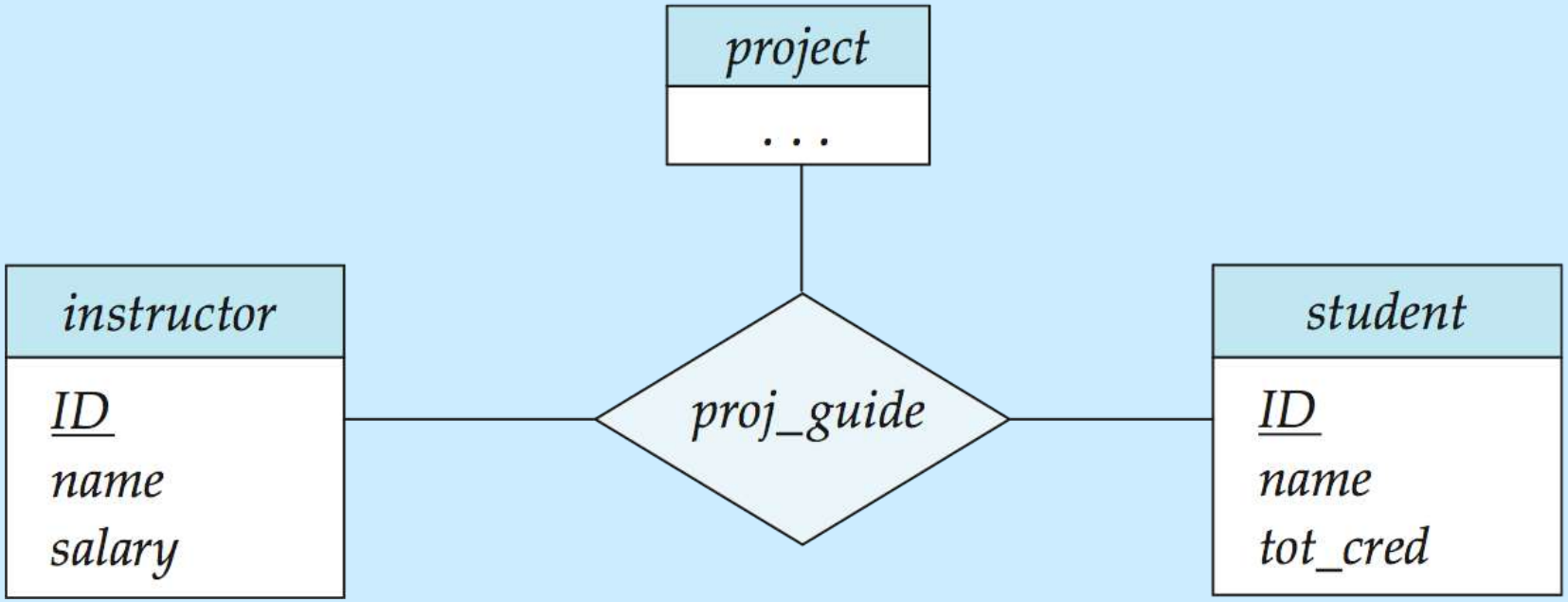
Cardinality limits can also express participation constraints. A line may have an associated minimum and maximum cardinality, shown in the form *l..h, where l is the minimum and h the maximum cardinality.*

- A minimum value of 1 indicates total participation of the entity set in the relationship set; that is, each entity in the entity set occurs in at least one relationship in that relationship set.
- A maximum value of 1 indicates that the entity participates in at most one relationship, while a maximum value ∗ indicates no limit.

Entity With Composite, Multivalued, and Derived Attributes



E-R Diagram with a Ternary Relationship



Weak Entity Sets

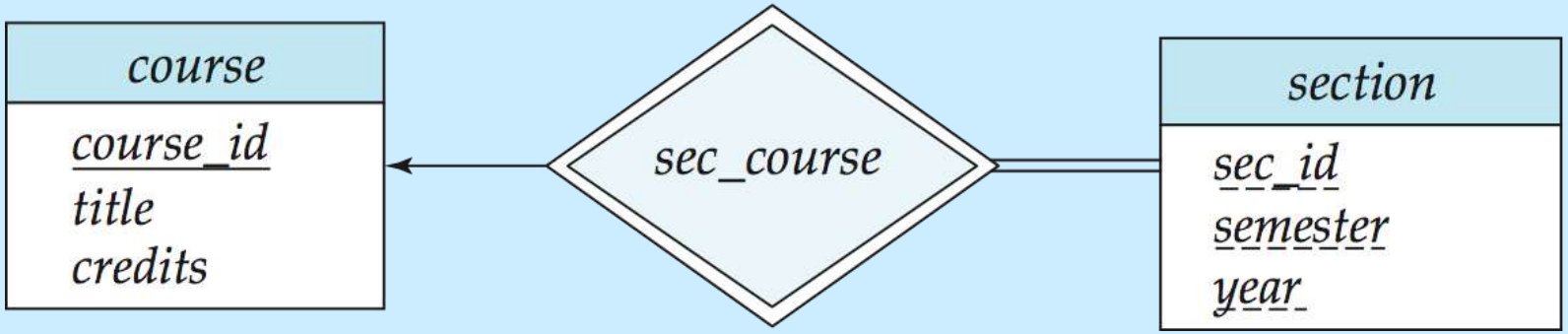
- An entity set that does not have a primary key is referred to as a **weak entity set**.
- For a weak entity set to be meaningful, it must be associated with another entity set, called the **identifying** or **owner** entity set.
- Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be existence dependent on the identifying entity set.
- The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**.
  - The identifying relationship is many-to-one from the weak entity set to the identifying entity set, and
  - The participation of the weak entity set in the relationship is total.
  - The identifying relationship set should not have any descriptive attributes

Weak Entity Sets

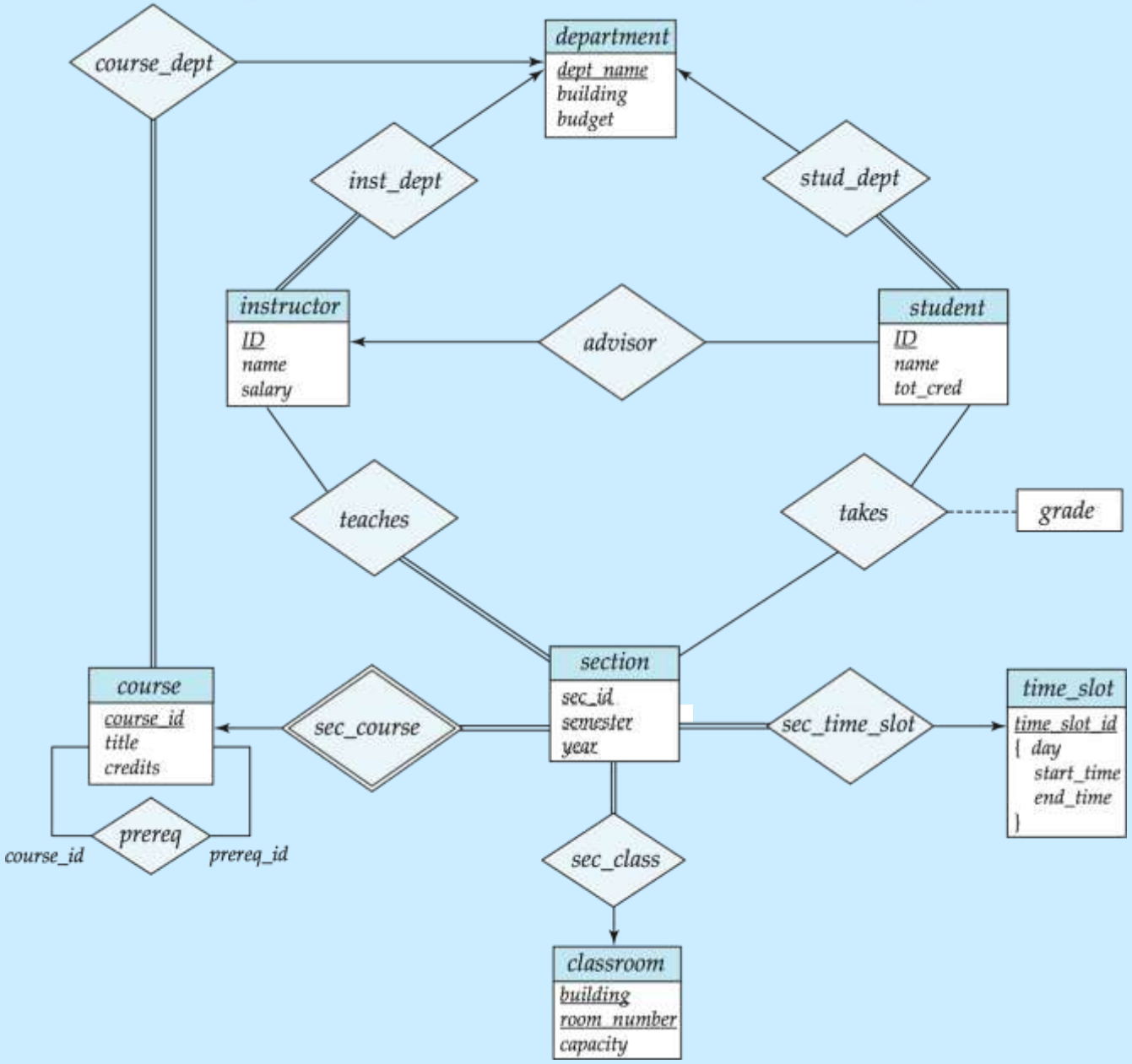
- The **discriminator** (*or partial key)* of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.
- The **primary key** of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set's discriminator.
- We underline the discriminator of a weak entity set with a dashed line.
- We put the identifying relationship of a weak entity in a double diamond.
- Primary key for *section* – (*course_id, sec_id, semester, year*)

Participation of an Entity Set in a Relationship Set

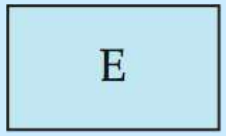
- Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set
  - E.g., participation of *section* in *sec_course* is total
    - every *section* must have an associated course
- Partial participation: some entities may not participate in any relationship in the relationship set
  - Example: participation of *instructor* in *advisor* is partial



E-R Diagram for a University Enterprise



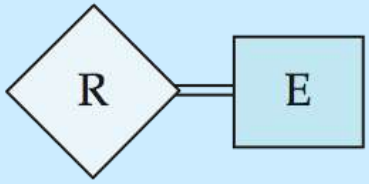
Summary of Symbols Used in E-R Notation



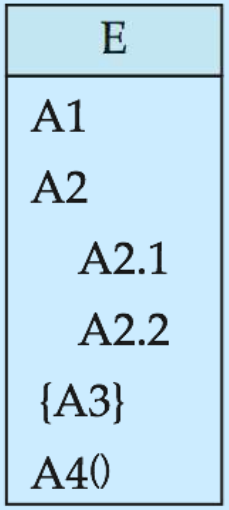
entity set

relationship set

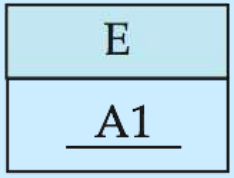
identifying relationship set for weak entity set



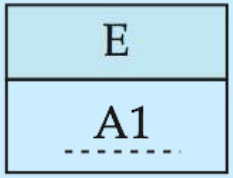
total participation of entity set in relationship



attributes:
simple (A1),
composite (A2) and
multivalued (A3)
derived (A4)

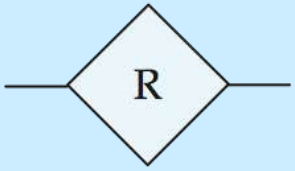


primary key

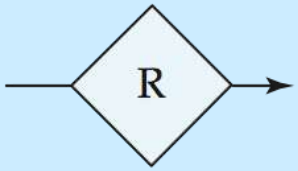


discriminating attribute of weak entity set

Symbols Used in E-R Notation (Cont.)

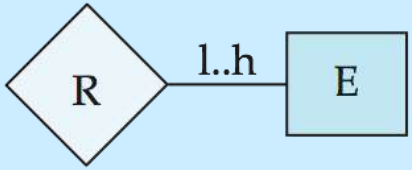


many-to-many relationship

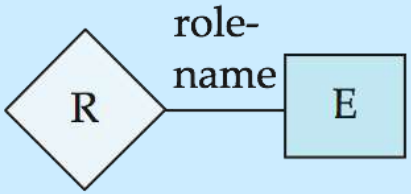


many-to-one relationship

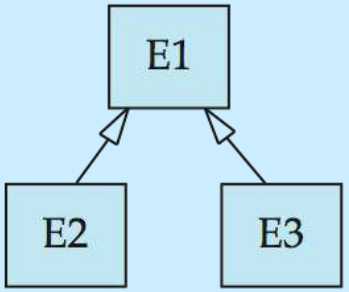
one-to-one relationship



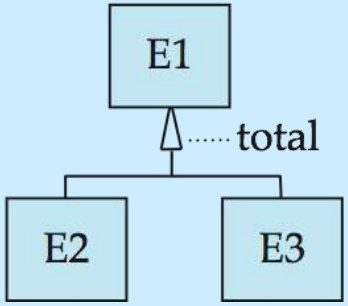
cardinality limits



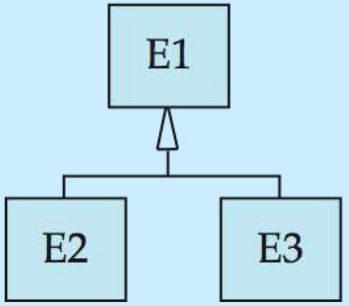
role indicator



ISA: generalization or specialization



total (disjoint) generalization

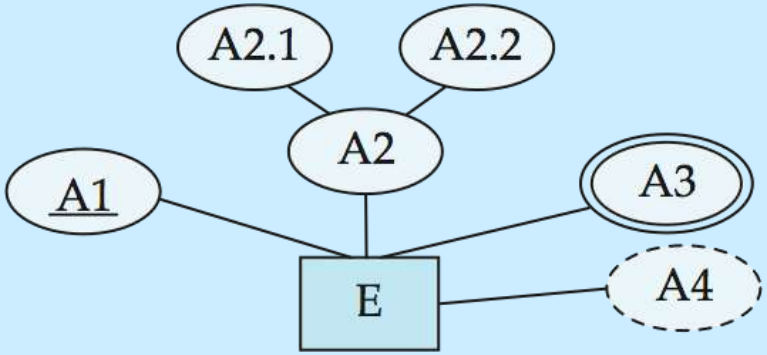


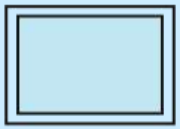
disjoint generalization

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

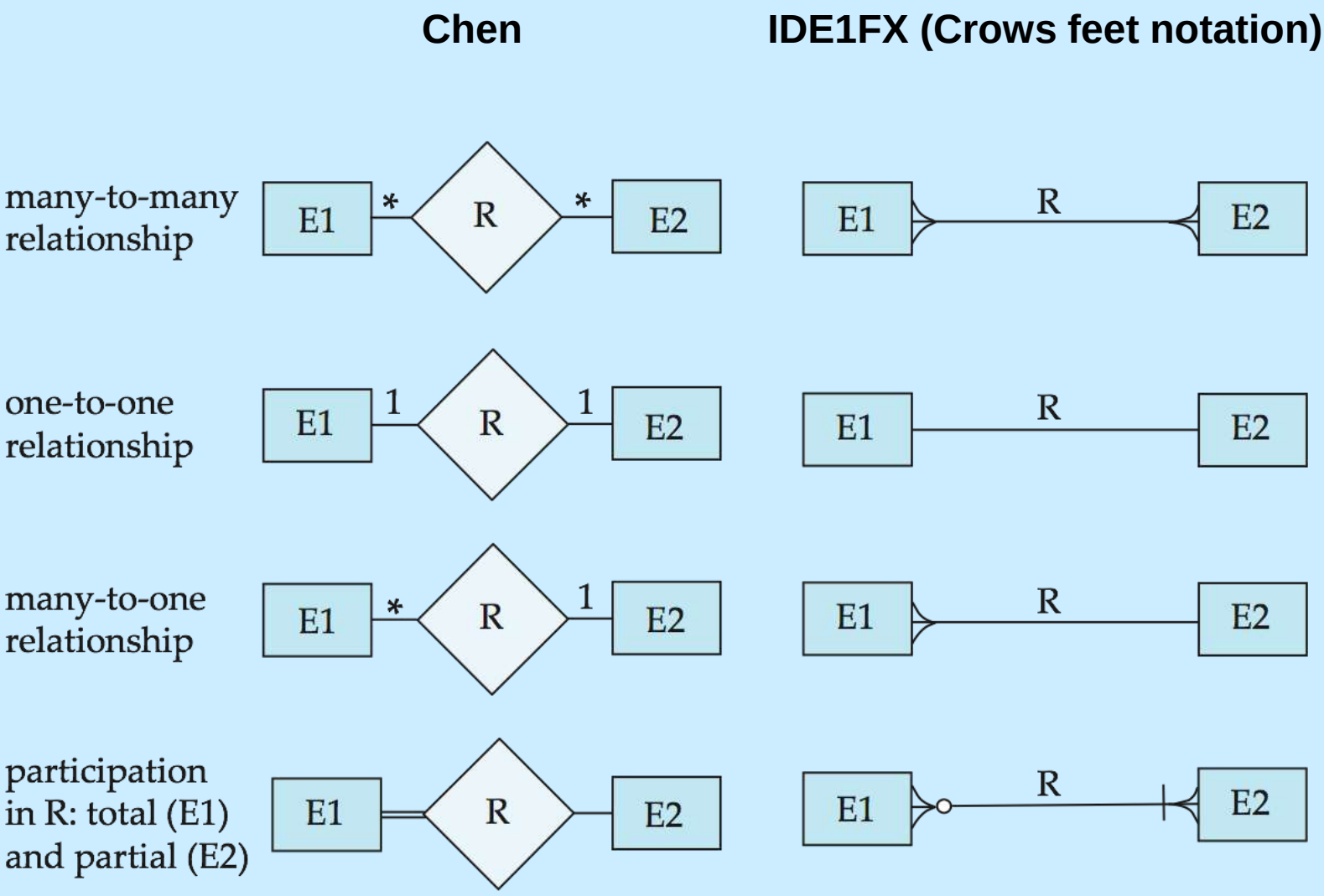


weak entity set 

generalization 

total generalization 

Alternative ER Notations



Reduction to Relational Schemas

Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as *relation schemas* that represent the contents of the database.
- A database which conforms to an E-R diagram can be represented by a collection of schemas (tables).
- For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.
- Each schema has a number of columns (generally corresponding to attributes), which have unique names.

Representing Strong Entity Sets With Simple Attributes

- Let E be a strong entity set with only simple descriptive attributes a1, a2,..., an.
- We represent this entity by a schema called E with n distinct attributes.
- Each tuple in a relation on this schema corresponds to one entity of the entity set E

classroom (*building, room number, capacity).*

department (*dept_name, building, budget).*

course (*course_id, title, credits).*

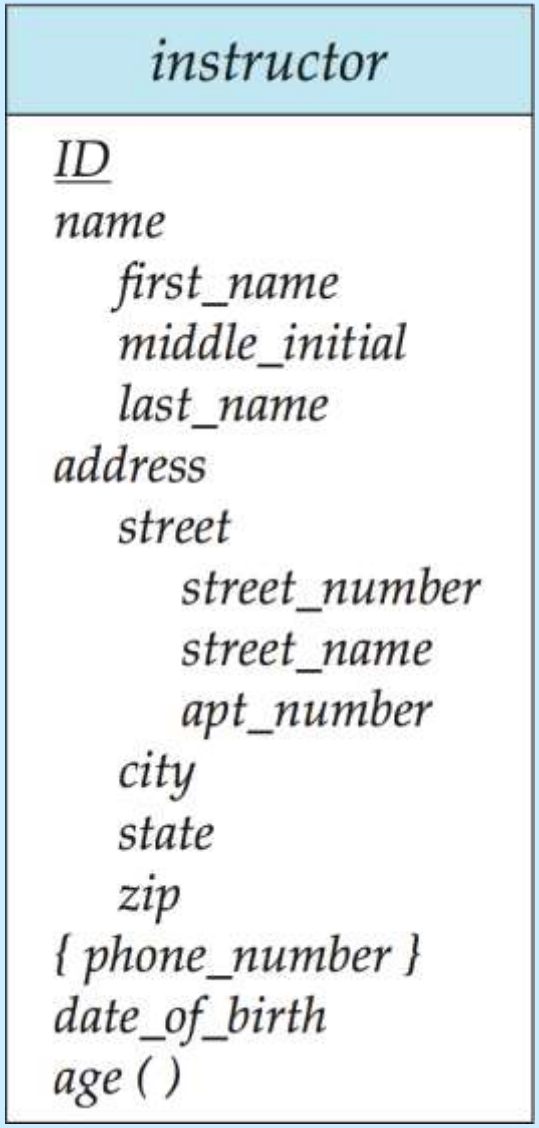
instructor (i_*ID, name, salary).*

student (s_*ID, name, tot_cred).*

department relation

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

Composite and Multivalued Attributes



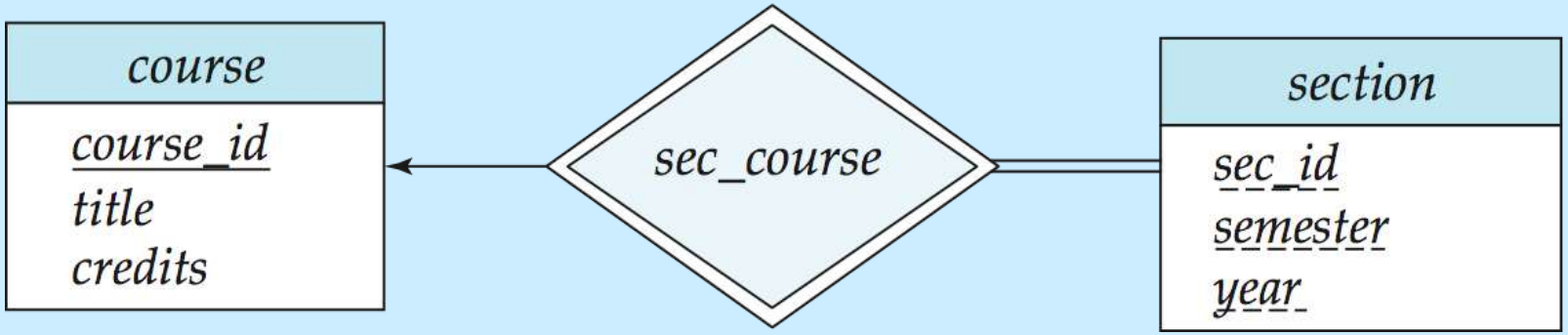
- Composite attributes are flattened out by creating a separate attribute for each component attribute
  - Example: given entity set *instructor* with composite attribute *name* with component attributes *first_name* and *last_name* the schema corresponding to the entity set has two attributes *name_first_name* and *name_last_name*
    - *Prefix omitted if there is no ambiguity*
- Ignoring multivalued attributes, extended instructor schema is
  - *instructor(ID, first_name, middle_initial, last_name, street_number, street_name, apt_number, city, state, zip_code, date_of_birth)*

Composite and Multivalued Attributes

- A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM*
  - Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*
  - Example: Multivalued attribute *phone_number* of *instructor* is represented by a schema:
    inst_phone= (*ID*, *phone_number*)
  - Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
    - For example, an *instructor* entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:
      (22222, 456-7890) and (22222, 123-4567)

Representing Weak Entity Sets With Simple Attributes

- A strong entity set reduces to a schema with the same attributes.
  e.g. *student(ID, name, tot_cred)*
- A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set
  e.g. *section (course_id, sec_id, sem, year)*



section relation

course_id	*sec_id*	*semester*	*year*
BIO-101	1	Summer	2009
BIO-301	1	Summer	2010
CS-101	1	Fall	2009
CS-101	1	Spring	2010
CS-190	1	Spring	2009
CS-190	2	Spring	2009
CS-315	1	Spring	2010
CS-319	1	Spring	2010
CS-319	2	Spring	2010
CS-347	1	Fall	2009
EE-181	1	Spring	2009
FIN-201	1	Spring	2010
HIS-351	1	Spring	2010
MU-199	1	Spring	2010
PHY-101	1	Fall	2009

Representing Relationship Sets

- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
- Example: schema for relationship set *advisor*

  advisor = (*s_id, i_id*)

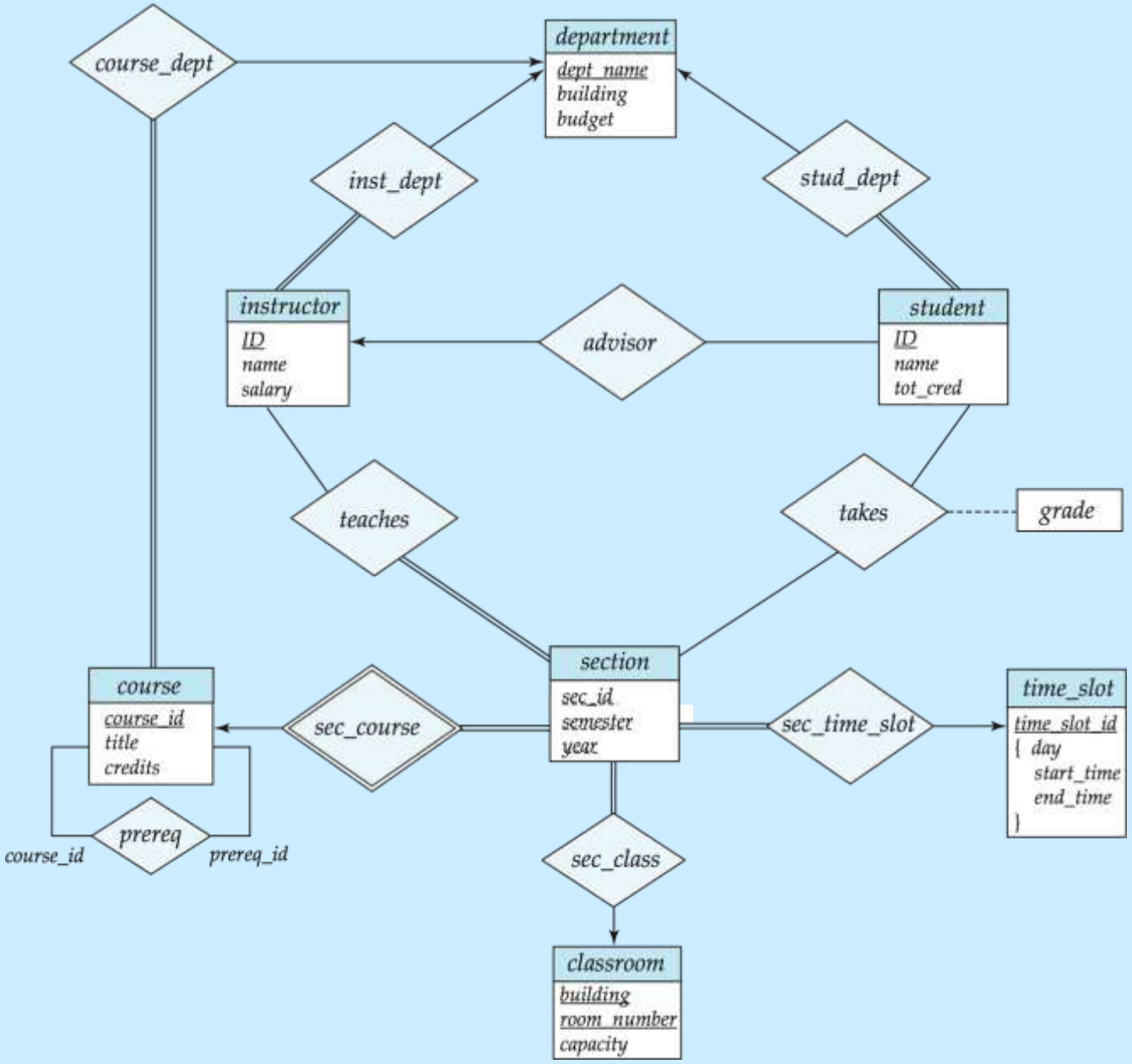
teaches relation between section and instructor

instructor (i_*ID*, *name, salary)*

section: with attributes (*course id*, *sec_id*, *semester*, *year)*

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

E-R Diagram for a University Enterprise



Schemas for Relationship sets

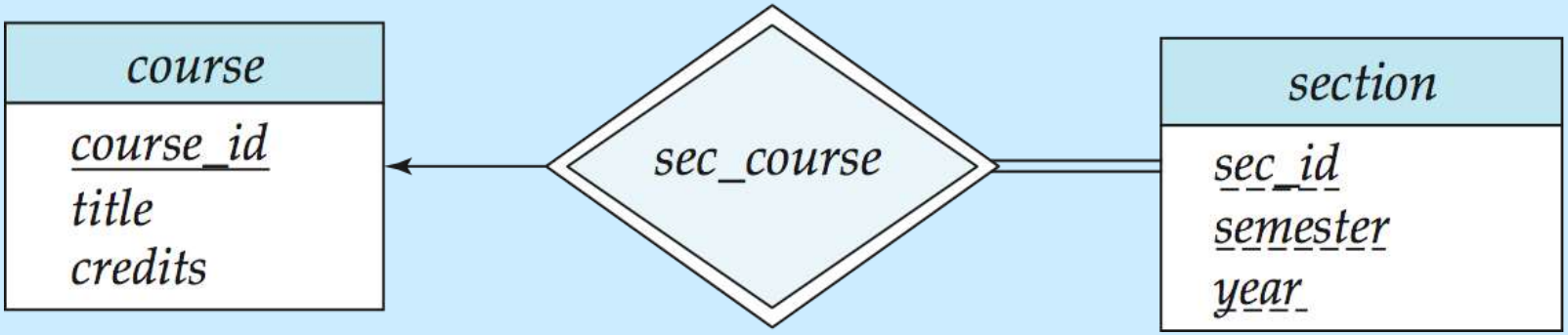
- The relationship sets in our design are listed below:
  - *teaches (i_ID, course id, sec id, semester, year)*
  - *takes (s_ID, course id, sec id, semester, year, grade)*
  - *prereq (course id, prereq id)*
  - *advisor (s_ID, i_ID)*
  - *sec_course (course id, sec id, semester, year)*
  - *sec_time_slot (course id, sec id, semester, year, time slot id)*
  - *sec_class (course id, sec id, semester, year, building, room number)*
  - *inst_dept (i_ID, dept_name)*
  - *stud_dept (s_ID, dept_name)*
  - *course_dept (course id, dept_name)*.

Redundancy of Schemas

- A relationship set linking a weak entity set to the corresponding strong entity set is treated specially.
- The primary key of a weak entity set includes the primary key of the strong entity set and its own discriminator.
- Ex: the weak entity set section is dependent on the strong entity set course via the relationship set sec_course.
- The primary key of section is {course_id, sec_id, semester, year} and the primary key of course is course_ id.
- Since sec_course has no descriptive attributes, the sec_course schema has attributes {course_id, sec_id, semester, year}.
- The schema for the entity set *section and* sec_course becomes same. So the *sec_course schema is redundant.*

Redundancy of Schemas

- In general, the schema for the relationship set linking a weak entity set to its corresponding strong entity set is redundant and does not need to be present in a relational database design based upon an E-R diagram.

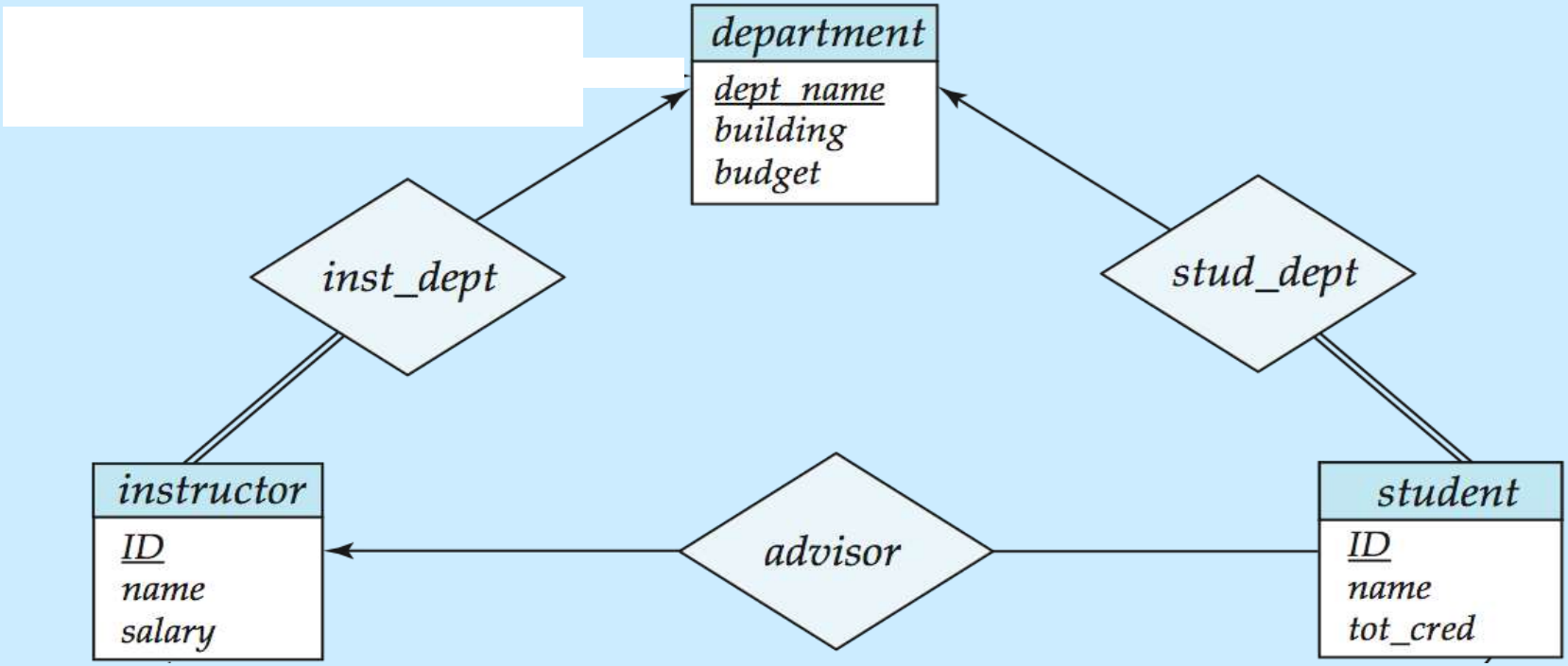


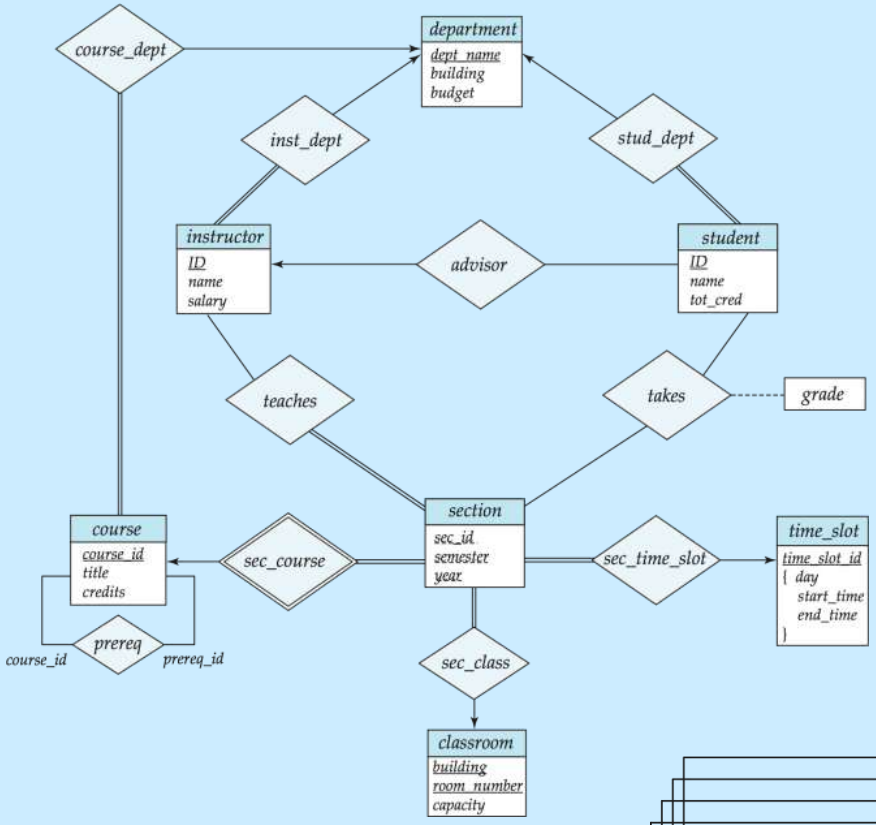
Combination of Schemas

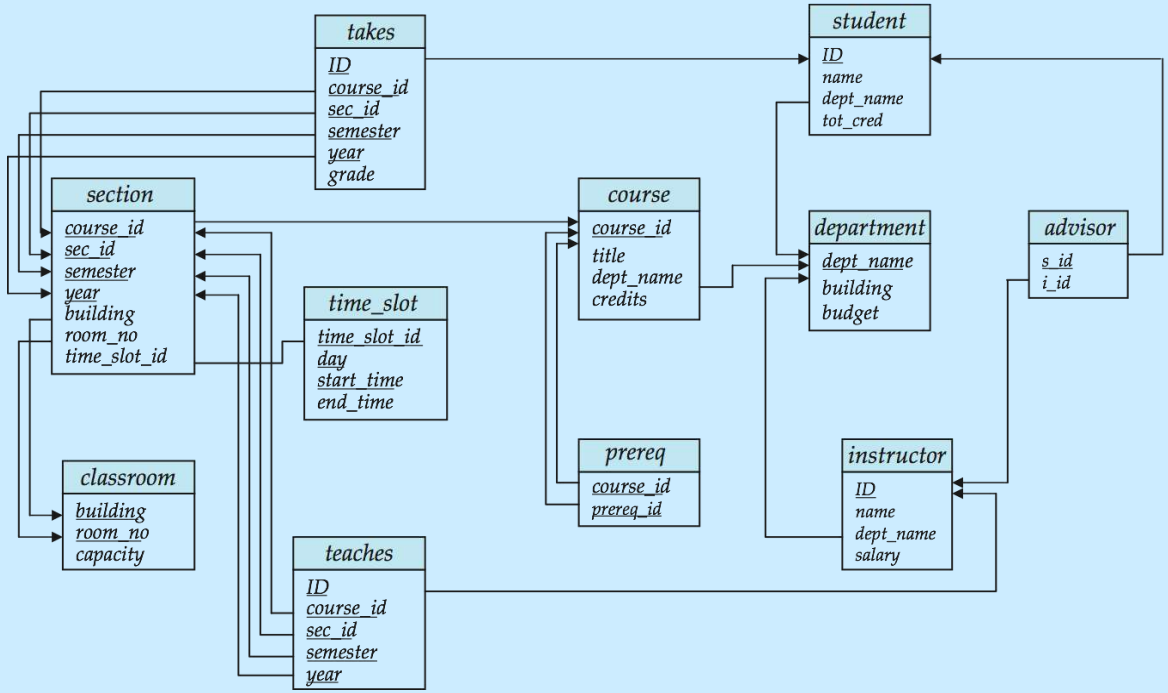
- Consider a many-to-one relationship set AB from entity set A to entity set B.
- Suppose further that the participation of A in the relationship is total; that is, every entity in the entity set A must participate in the relationship AB.
- Then we can combine the schemas A and AB to form a single schema consisting of the union of attributes of both schemas.
- The primary key of the combined schema is the primary key of the entity set into whose schema the relationship set schema was merged.

Combination of Schemas

Example: Instead of creating a schema for relationship set *inst_dept*, add an attribute *dept_name* to the schema arising from entity set *instructor*




course_dept
department
dept_name
building
budget
inst_dept
stud_dept
instructor
ID
name
salary
advisor
student
ID
name
tot_cred
teaches
takes
grade
course
course_id
title
credits
sec_course
section
sec_id
semester
year
sec_time_slot
time_slot
time_slot_id
day
start_time
end_time
prereq
course_id
prereq_id
sec_class
classroom
building
room_number
capacity


takes
ID
course_id
sec_id
semester
year
grade
student
ID
name
dept_name
tot_cred
section
course_id
sec_id
semester
year
building
room_no
time_slot_id
course
course_id
title
dept_name
credits
department
dept_name
building
budget
advisor
s_id
i_id
time_slot
time_slot_id
day
start_time
end_time
classroom
building
room_no
capacity
prereq
course_id
prereq_id
instructor
ID
name
dept_name
salary
teaches
ID
course_id
sec_id
semester
year

University Database

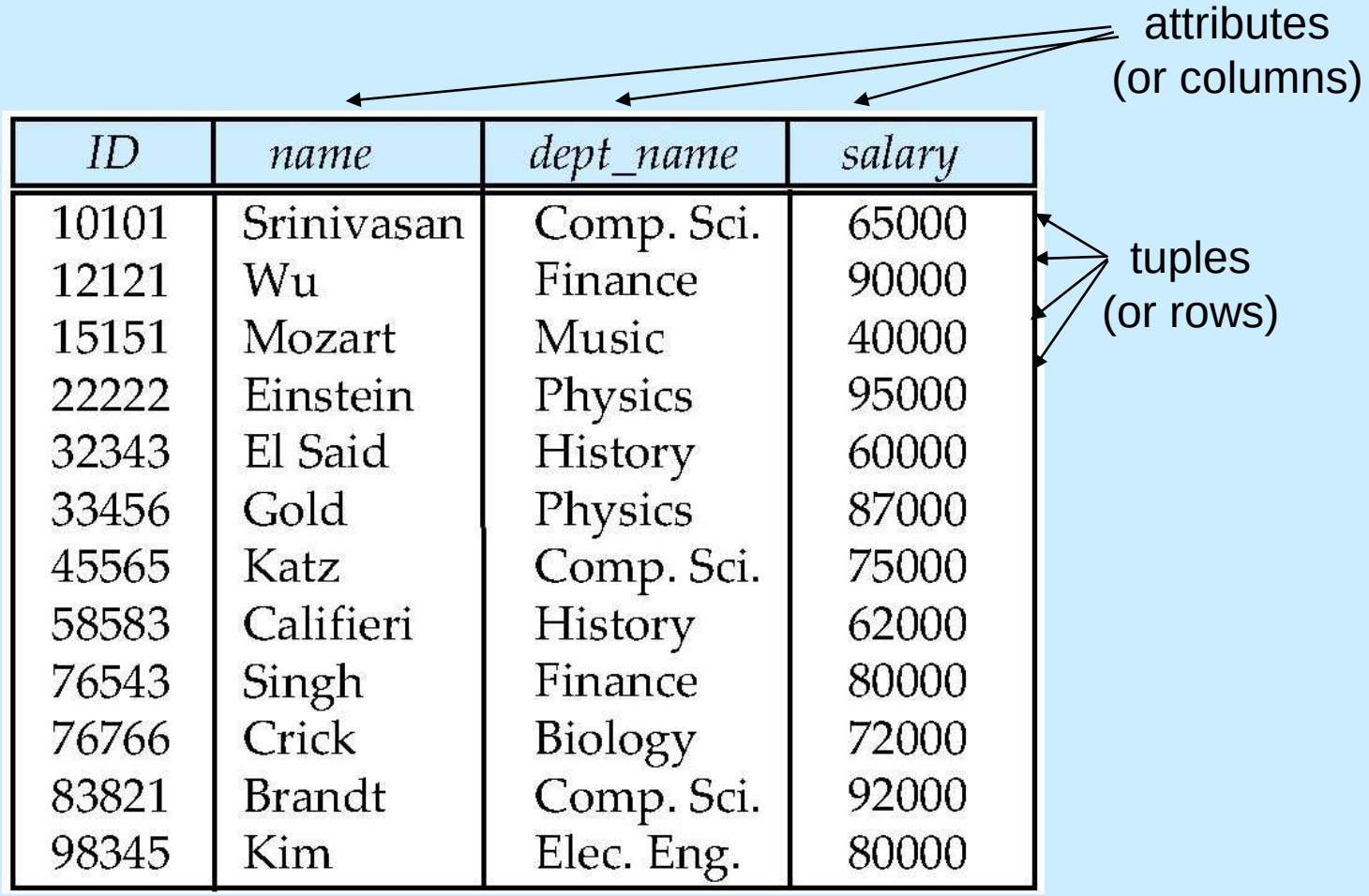
- The relation schema generated from entity sets, with primary keys underlined:
  - classroom: with attributes (*building, room number, capacity).*
  - department: with attributes (*dept_name, building, budget).*
  - **course: with attributes (*course_id, title, dept_name, credits).***
  - **instructor: with attributes (i*_ID, name, dept_name, salary).***
  - **student: with attributes (s_ID, name, dept_name, tot_cred).***
  - section: with attributes (*course id, sec_id, semester, year, building, room_number, time_slot_id).*
  - Time_slot: with attributes (*time_slot_id, day, start_time, end_time).*
  - *teaches (i_ID, course id, sec id, semester, year)*
  - *takes (s_ID, course id, sec id, semester, year, grade)*
  - *prereq (course id, prereq id)*
  - *advisor (s_ID, i_ID)*

Introduction to Relational Model

Relational Model

- A relational database consists of a collection of tables, each of which is assigned a unique name.
- Each table has some number of column with unique name.
- For example, consider the instructor table, which stores information about instructors.
- The table has four column headers: *ID, name, dept_name*, and *salary*.
- Each row of this table records information about an instructor, consisting of the instructor's ID, name, dept_name, and salary.
- In general, a row in a table represents a relationship among a set of values.
- In the relational model the term *relation* is used to refer to a table, while the term *tuple* is used to refer to a row. Similarly, the term *attribute* refers to a column of a table.

Example of a Relation: Instructor



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

Course relation

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

Prereq relation

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

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples

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

Attribute

- For each attribute of a relation, there is a set of permitted values, called the **domain** of that attribute.
- Thus, the domain of the salary attribute of the instructor relation is the set of all possible salary values, while the domain of the *name* attribute is the set of all possible instructor names.
- We require that, for all relations *r,* the domains of all attributes of *r* be atomic*.*
- A domain is **atomic** if elements of the domain are considered to be indivisible units.
- The special value ***null*** is a member of every domain
- The null value causes complications in the definition of many operations

Database Schema

- **Database schema** is the logical design of the database, and the **database instance** is a snapshot of the data in the database at a given instant in time.

Department relation

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

Relation Schema and Instance

- $A_1, A_2, \ldots, A_n$ are *attributes*
- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

  Example:

  instructor = (*ID*, *name*, *dept_name*, *salary*)

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

Section relation

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Teaches relation

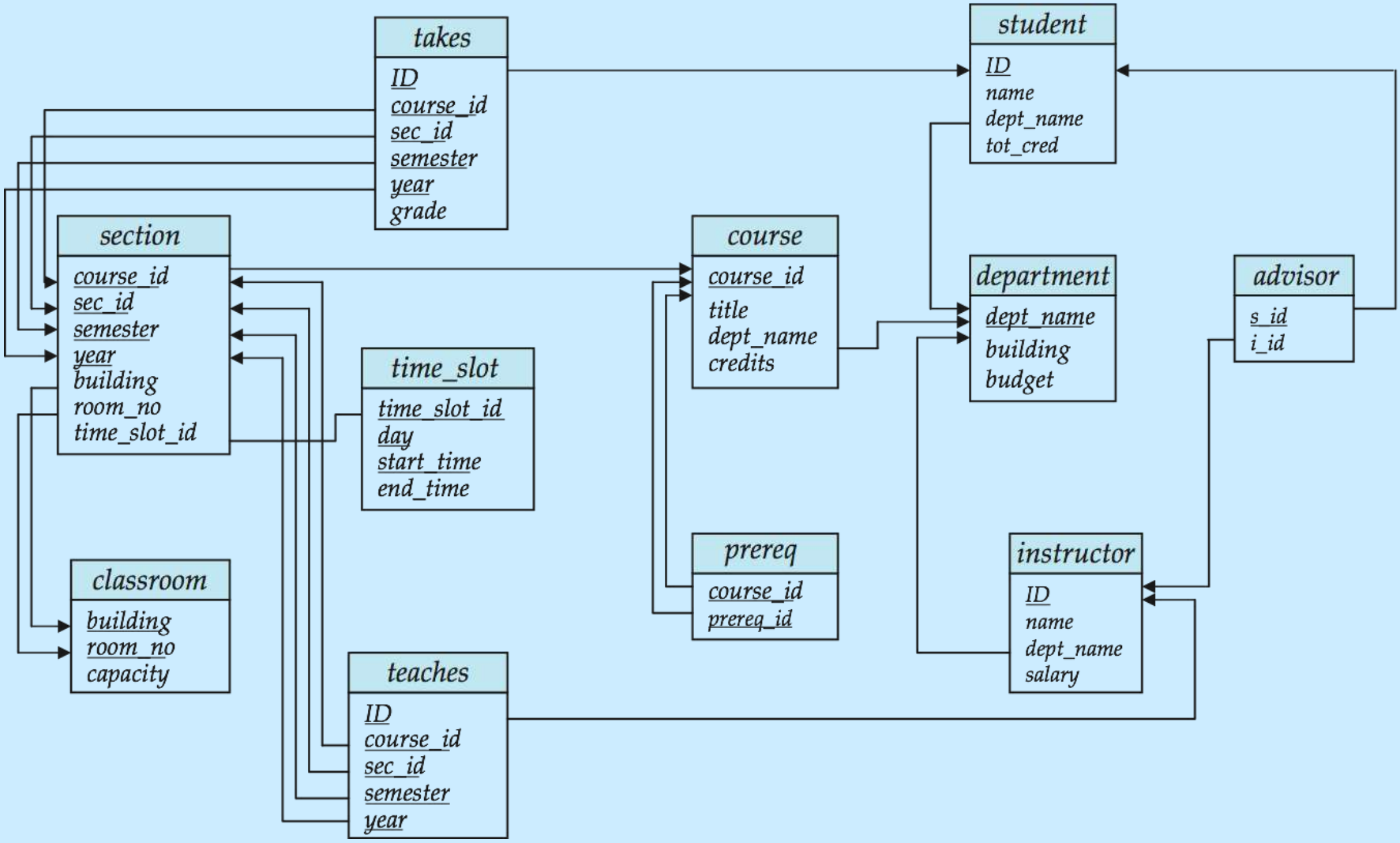
ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Keys

- Let K $\subseteq$ R
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey *K* is a **candidate key** if *K* is minimal

  Example: {*ID*} is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**.
  - which one?
- A relation, say r1, may include among its attributes the primary key of another relation, say r2. This attribute is called a **foreign key from r1, referencing r2.**
- The relation r1 is also called the **referencing** relation of the foreign key dependency, and r2 is called the **referenced relation** of the foreign key.
- For example, the attribute dept_name in instructor is a foreign key from instructor, referencing department, since dept_name is the primary key of department.

Schema Diagram for University Database



- classroom(*building, room number, capacity).*
- department (*dept_name, building, budget).*
- course (*course_id, title, dept_name, credits).*
- instructor (i*_ID, name, dept_name, salary).*
- section (*course id, sec_id, semester, year, building, room_number, time_slot_id).*
- student (s*_ID, name, dept_name, tot_cred).*
- Time_slot (*time_slot_id, day, start_time, end_time)*
- *teaches (i_ID, course id, sec id, semester, year)*
- *takes (s_ID, course id, sec id, semester, year, grade)*
- *prereq (course id, prereq id)*
- *advisor (s ID, i ID)*

Relational Query Languages

- A **query language** is a language in which a user requests information from the database.
- Query languages can be categorized as either procedural or nonprocedural.
- In a **procedural language** the user describes the desired information with giving a specific procedure for obtaining that information.
- In a **nonprocedural language,** the user describes the desired information without giving a specific procedure for obtaining that information.
- “Pure” languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- Relational operators

Relational Algebra

- Procedural language . The operators take one or two relations as inputs and produce a new relation as output.
- Six basic operators
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ
- Additional operations:
  - Set intersection
  - Natural join
  - Assignment
  - Outer join.

Select Operation – Example

- Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Select Operation

- The select operation (unary) selects those tuples that satisfy a given predicate. We use the lowercase Greek letter sigma (σ) to denote selection. The predicate appears as a subscript to σ. The argument relation is in parentheses after the σ
- Notation: $\sigma_p(r)$
- *p* is called the **selection predicate**
- Defined as:

$$\sigma_p(\boldsymbol{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where *p* is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: =, $\neq$, >, $\geq$. <. $\leq$

Select Operation - Example

Example: Select those tuples of the instructor relation where the instructor is in the “Physics” department

$$\sigma_{dept\_name=\text{“Physics”}}(instructor)$$

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

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Instructor relation

Select Operation - Example

Find all instructors with salary greater than $90,000 by writing

$$\sigma_{salary>90000}(instructor)$$

- In general, we allow comparisons using =, ≠, <, >, ≤ and ≥ in the selection predicate.
- Furthermore, we can combine several predicates into a larger predicate by using the connectives and (∧), or (∨), and not (¬).
- Thus, to find the instructors in Physics with a salary greater than $90,000, we write:

$$\sigma_{dept\_name=\text{“Physics”} \wedge salary > 90000}(instructor)$$

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000

Project Operation – Example

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\Pi_{A,C}\,(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Project Operation

- The project operation is an unary operation, which projects those attributes which are listed in query.
- Projection is denoted by the uppercase Greek letter pi (Π).
- We list those attributes that we wish to appear in the result as a subscript to Π. The argument relation follows in parentheses.
- Notation:

$$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result

Project Operation Example

Example: List all instructors' ID, name, and salary, but do not care about the dept name.

$$\Pi_{ID,\ name,\ salary}\ (instructor)$$

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

Composition of Relational Operations

- Find the name of instructors in the physics department.

$$\sigma_{dept\_name=\text{“Physics”}}(instructor)$$

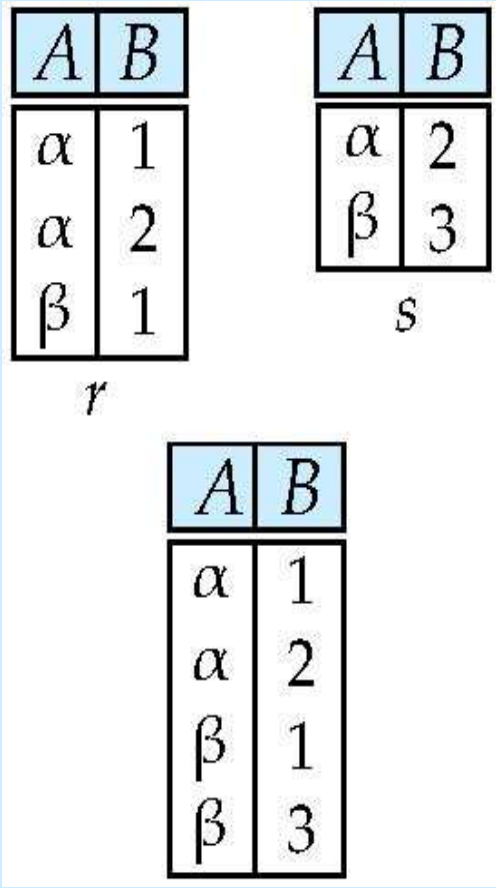
ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

$$\Pi_{name}\left(\sigma_{dept\_name=\text{“Physics”}}(instructor)\right)$$

name
Einstein
Gold

Union Operation – Example

- Relations *r, s:*



A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∪ s:

A	B
α	1
α	2
β	1
β	3

Union Operation

- Output the union of tuples from the two input relations.
- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.
  1. *r, s* must have the *same* **arity** (same number of attributes)
  2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)
- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) \cup \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

Section relation

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

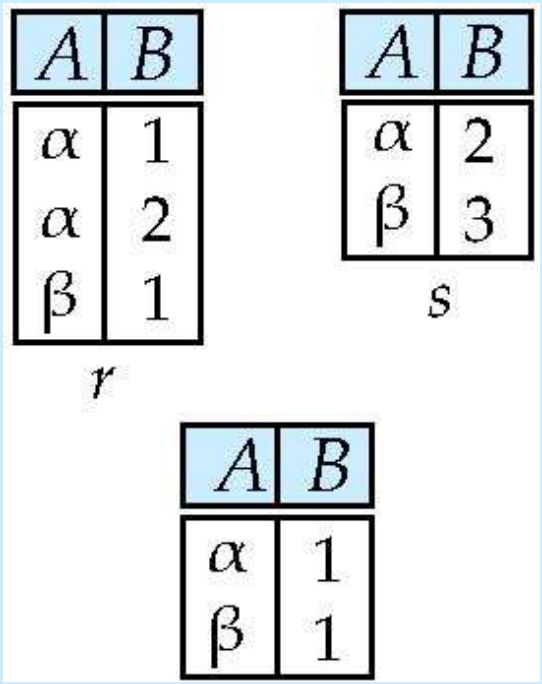
$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) \cup \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Union operation automatically remove duplicates

Set difference of two relations

- Relations *r*, *s*:



A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- *r – s:*

A	B
α	1
β	1

Set Difference Operation

- The set difference operation, denoted by -, allows us to find tuples that are in one relation but are not in another.
- The expression r − s produces a relation containing those tuples in r but not in s.
- Notation $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must be taken between **compatible** relations.
  - *r* and *s* must have the same arity
  - attribute domains of *r* and *s* must be compatible
- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) - \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) - \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

course_id
CS-347
PHY-101

Cartesian-Product Operation – Example

- Relations *r, s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Cartesian-Product Operation

- The Cartesian product operation, denoted a cross (×), allows us to combine information from any two relations.
- Notation r x s
- Defined as:

$$r \times s = \{t\ q \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \varnothing$).
- If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.

Instructor relation

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

teaches relation

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Instructor X teaches

Inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2009
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2010
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Find the names of all instructors in the Physics department together with the course id of all courses they taught.

$\sigma_{dept\_name=\text{“Physics”}}(instructor \times teaches)$

inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
22222	Einstein	Physics	95000	10101	CS-437	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009
22222	Einstein	Physics	95000	32343	HIS-351	1	Spring	2010
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
33456	Gold	Physics	87000	10101	CS-437	1	Fall	2009
33456	Gold	Physics	87000	10101	CS-315	1	Spring	2010
33456	Gold	Physics	87000	12121	FIN-201	1	Spring	2010
33456	Gold	Physics	87000	15151	MU-199	1	Spring	2010
33456	Gold	Physics	87000	22222	PHY-101	1	Fall	2009
33456	Gold	Physics	87000	32343	HIS-351	1	Spring	2010
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…

$\sigma_{instructor.Id = teaches.ID}(\sigma_{dept\_name="Physics"}(instructor\ X\ teaches))$

inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009

$$\Pi_{name, course\_id}(\sigma_{instructor.Id = teaches.ID}(\sigma_{dept\_name=\text{“Physics”}}(instructor \times teaches)))$$

name	*course_id*
Einstein	PHY-101

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Notation: lower case Greek letter rho (ρ)

$$\rho_x(E)$$

returns the expression *E* under the name *X*

- If a relational-algebra expression *E* has arity *n*, then

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

returns the result of expression *E* under the name *X*, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

$$\rho_{physics\_instructor}(\sigma_{dept\_name=\text{“Physics”}}(instructor))$$

Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\Pi_S(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x (E_1)$, x is the new name for the result of E_1

Additional Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- Set intersection
- Natural join
- Assignment
- Outer join

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- Assume:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$
- Example: to find all courses taught in the Fall 2009 semester, and in the Spring 2010 semester.

$$\Pi_{course\_id} (\sigma_{semester=\text{“Fall”} \wedge year=2009} (section)) \cap \Pi_{course\_id} (\sigma_{semester=\text{“Spring”} \wedge year=2010} (section))$$

Set-Intersection Operation – Example

- Relation *r, s*:

A	*B*
α	1
α	2
β	1

r

A	*B*
α	2
β	3

s

- $r \cap s$

A	*B*
α	2

Natural-Join Operation

- Disadvantage of Cartesian product is that we need to apply selection on those attributes which are common in both relations.
- The natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.
- It is denoted by the join symbol ⋈.
- The natural-join operation forms a Cartesian product of its two argument relations, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes.

Natural-Join Operation

Find the names of all instructors in the Physics department together with the course id of all courses they taught.

Instructor* ⋈ *teaches

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Natural-Join Operation

$\sigma_{dept\_name="Physics"}(instructor \bowtie teaches)$

inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009

$\Pi_{name,\ course\_id}(\sigma_{dept\_name="Physics"}(instructor \bowtie teaches))$

name	course_id
Einstein	PHY-101

Natural-Join Operation

relations $r(R)$ and $s(S)$. The natural join of r and s, denoted by $r \bowtie s$, is a relation on schema $R \cup S$ formally defined as follows:

$$r \bowtie s = \Pi_{R \cup S} (\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} (r \times s))$$

where $R \cap S = \{A_1, A_2, \ldots, A_n\}$.

If r(R) and s(S) are relations without any attributes in common, that is, $R \cap S = \emptyset$, then $r \bowtie s = r \times s$.

Natural Join: Example

- Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach
  - $\Pi_{name,\ title}(\sigma_{dept\_name=\text{“Comp. Sci.”}}(instructor \bowtie teaches \bowtie course))$

name	*title*
Brandt	Game Design
Brandt	Image Processing
Katz	Image Processing
Katz	Intro. to Computer Science
Srinivasan	Intro. to Computer Science
Srinivasan	Robotics
Srinivasan	Database System Concepts

- Natural join is associative
  - $(instructor \bowtie teaches) \bowtie course$ is equivalent to $instructor \bowtie (teaches \bowtie course)$
- Natural join is commutative
  - $instructor \bowtie teaches$ is equivalent to $teaches \bowtie instructor$

Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) **false** by definition.
    - We shall study precise meaning of comparisons with nulls later

Outer Join – Example

- Relation *instructor1*

ID	*name*	*dept_name*
10101 12121 15151	Srinivasan Wu Mozart	Comp. Sci. Finance Music

- Relation *teaches1*

ID	*course_id*
10101 12121 76766	CS-101 FIN-201 BIO-101

- Natural Join

  instructor ⋈ *teaches*

ID	*name*	*dept_name*	*course_id*
10101 12121	Srinivasan Wu	Comp. Sci. Finance	CS-101 FIN-201

Outer Join – Example

- The **left outer join** ⟕ takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join.

- Left Outer Join

  instructor ⟕ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*

Outer Join – Example

- The **right outer join** ⟖ takes all tuples in the right relation that did not match with any tuple in the left relation, pads the tuples with null values for all other attributes from the left relation, and adds them to the result of the natural join.

- Right Outer Join

  instructor ⟖ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

Outer Join – Example

- The **full outer join** ⟗ **d*oes both the left and right outer join operations,*** padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join.

- Full Outer Join

  instructor ⟗ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*
76766	null	null	BIO-101

Summery of Relational Algebra

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary}>=85000}(instructor)$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID,\ salary}(instructor)$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
$\bowtie$ (Natural Join)	$instructor \bowtie department$
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
$\times$ (Cartesian Product)	$instructor \times department$
	Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
$\cup$ (Union)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	Output the union of tuples from the two input relations.

Roles in the Database Environment

- There are four distinct types of people that participate in the DBMS environment:
  - Data and database administrators,
  - Database designers,
  - Application developers, and
  - End-users.

Data and Database Administrators

- **Data Administrator (DA)** : is responsible for the management of the data resource including
  - Database planning,
  - Development and maintenance of standards,
  - Policies and procedures, and
  - Conceptual/logical database design.
- **Database Administrator (DBA)** requiring detailed knowledge of the target DBMS and the system environment and responsible for the physical realization of the database, including
  - physical database design and implementation,
  - security and integrity control,
  - maintenance of the operational system, and
  - ensuring satisfactory performance of the applications for users.

Database Designers

- **Logical database designer:** is concerned with
  - identifying the data (that is, the entities and attributes),
  - the relationships between the data, and
  - the constraints on the data that is to be stored in the database.
- The logical database designer must have a thorough and complete understanding of the organization's data and any constraints on this data.
- The work of the logical database designer into two stages:
  - conceptual database design, which is independent of implementation details such as the target DBMS, application programs, programming languages, or any other physical considerations;
  - logical database design, which targets a specific data model, such as relational, network, hierarchical, or object-oriented.

Database Designers

- **Physical database designer :** decides how the logical database design is to be physically realized. This involves:
  - mapping the logical database design into a set of tables and integrity constraints;
  - selecting specific storage structures and access methods for the data to achieve good performance;
  - designing any security measures required on the data.
- conceptual and logical database design are concerned with the *what,* physical database design is concerned with the *how.*

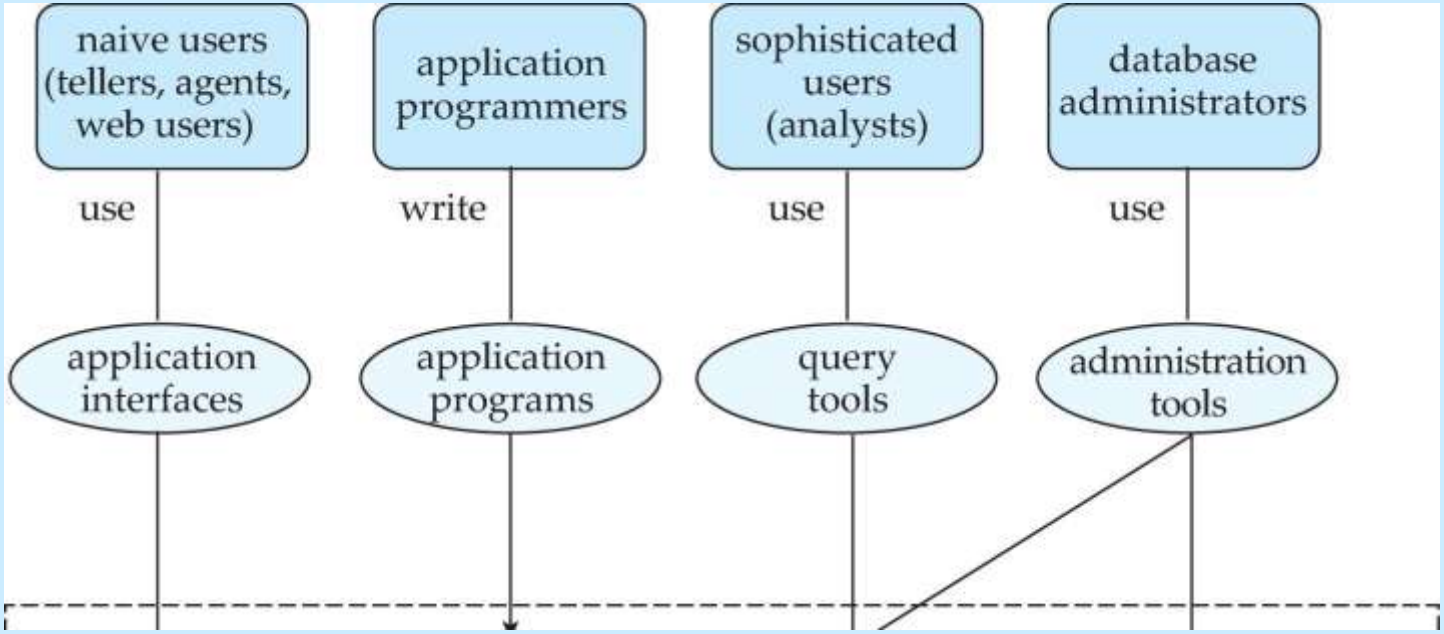
Application Developers

- Once the database has been implemented, the application programs are implemented by application developer for end-users.
- This includes retrieving data, inserting, updating, and deleting data.

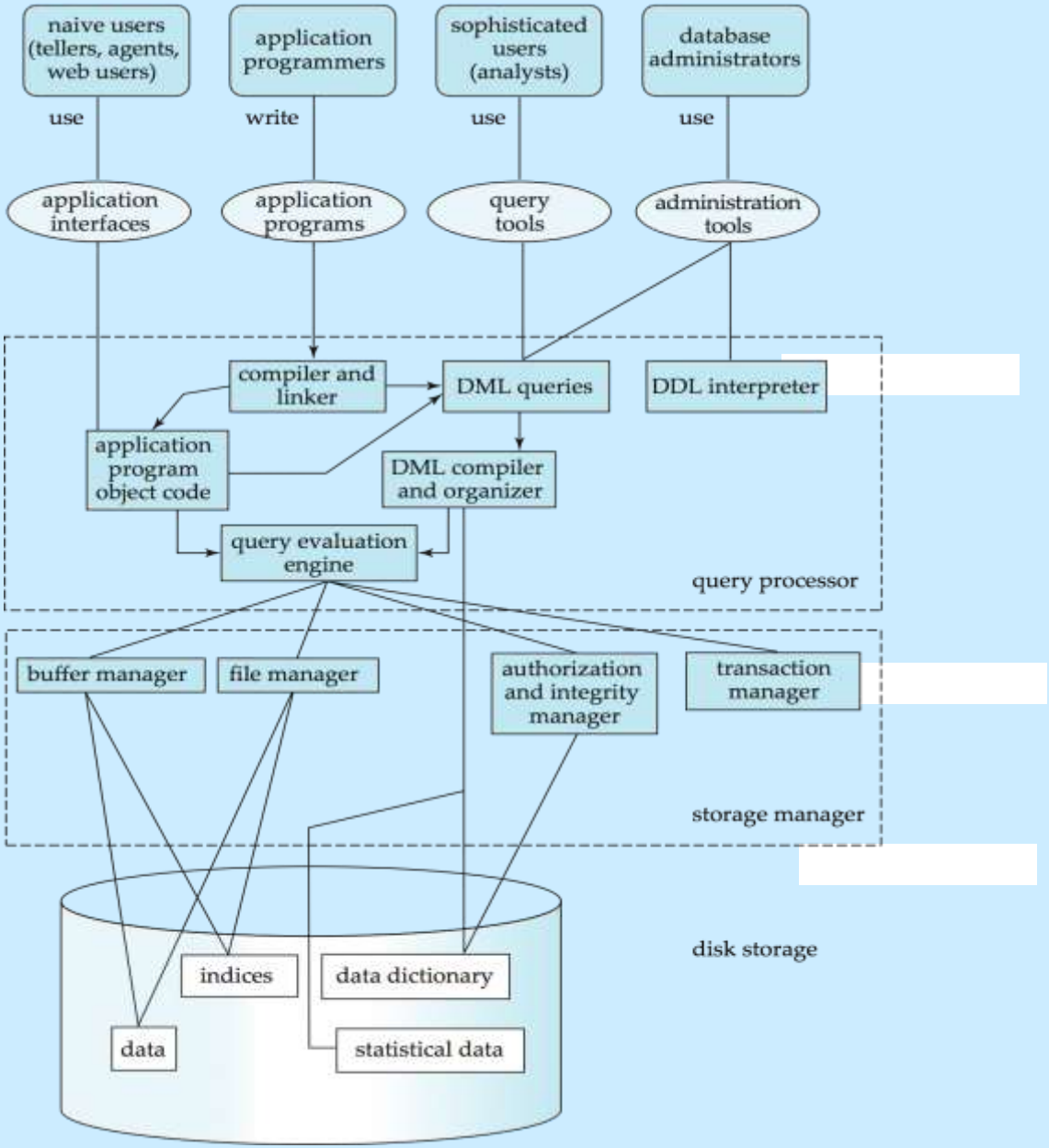
End-Users

- The end-users are the ‘clients’ for the database. End-users can be classified according to the way they use the system:
  - **Naïve users:** are typically unaware of the DBMS.
  - They invoke database operations by entering simple commands or choosing options from a menu.
  - For example, the checkout assistant at the local supermarket uses a bar code reader to find out the price of the item.
  - **Sophisticated users**: is familiar with the structure of the database and the facilities offered by the DBMS.
  - Sophisticated end-users may use a high-level query language such as SQL to perform the required operations.

Database Users and Administrators



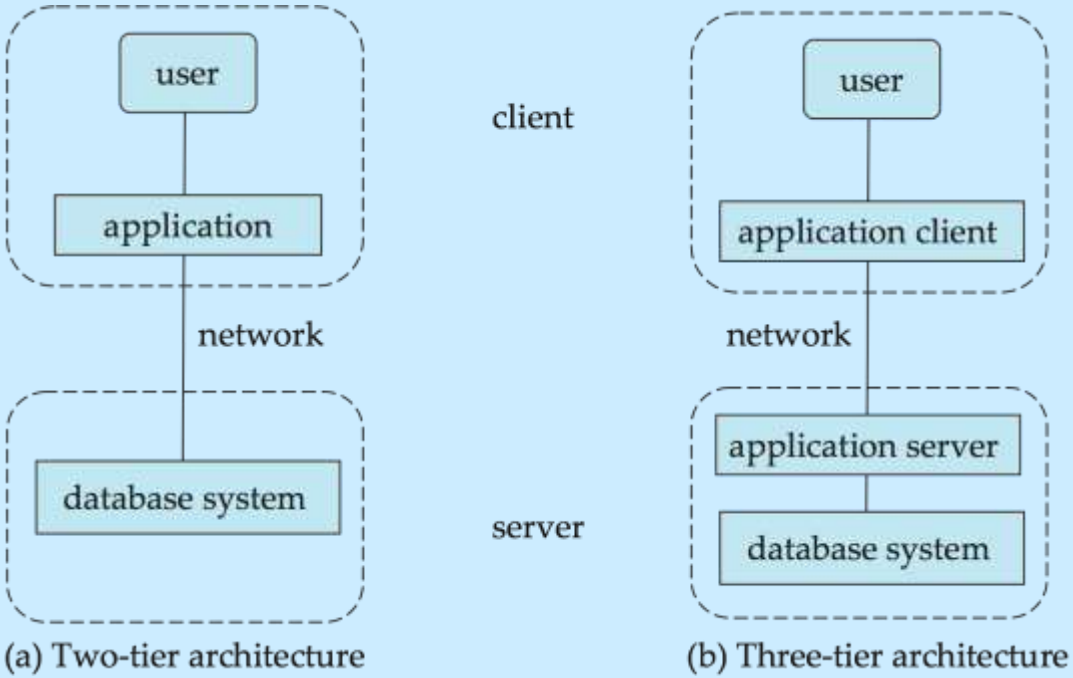
Database System Internals



Database Architecture

The architecture of a database systems is greatly influenced by the underlying computer system on which the database is running:

- Centralized
- Client-server
- Parallel (multi-processor)
- Distributed



Database applications are usually partitioned into two or three parts.

- In a two-tier architecture, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.
- In contrast, in a three-tier architecture, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, usually through a forms interface. The application server in turn communicates with a database system to access data.
- Three-tier applications are more appropriate for large applications, and for applications that run on the World Wide Web.

End of Unit 1