Week 4 Lecture 1

Class	BSCCS2001
Created	@September 29, 2021 11:42 AM
Materials	
Module #	16
Type	Lecture
Week #	4

Formal Relational Query Languages

- Relational Algebra
  - Procedural and Algebra based
- Tuple Relational Calculus
  - Non-procedural and Predicate Calculus based
- Domain Relational Calculus
  - Non-procedural and Predicate Calculus based

Relational Algebra

- Created by Edgar F. Codd at IBM in 1970
- Procedural Language
- Six basic operators
  - Select: σ
  - Project: Π
  - Union: $\cup$
  - Set difference: $-$
  - Cartesian product: $\times$
  - Rename: ρ
- The operators take one or two relations as inputs and produce a new relation as the result

SELECT operation

- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

  $\sigma_p(r) = \{t | t \in r \ and \ p(t)\}$

  where p is a formula in propositional calculus consisting of terms connected by

  $\wedge$ (**and**)

  $\vee$ (**or**)

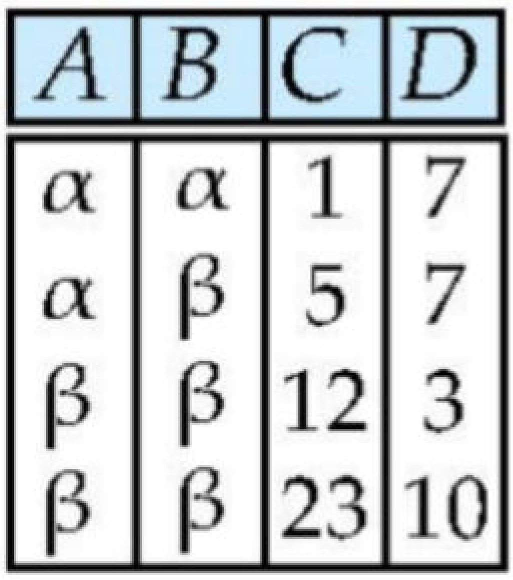
  $\neg$ (**not**)

  Each term is one of:

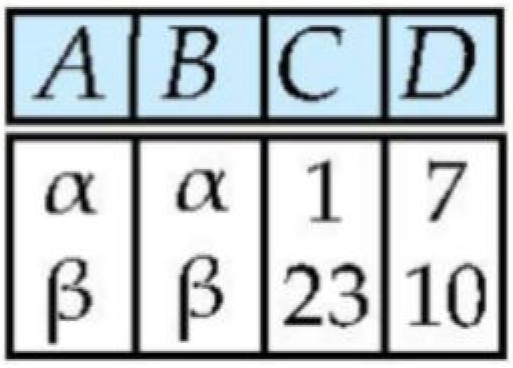
  $< attribute > op < attribute > or < constant >$

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

  $\sigma_{dept\_name \ ='Physics'} \ (instructor)$



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



A	B	C	D
α	α	1	7
β	β	23	10

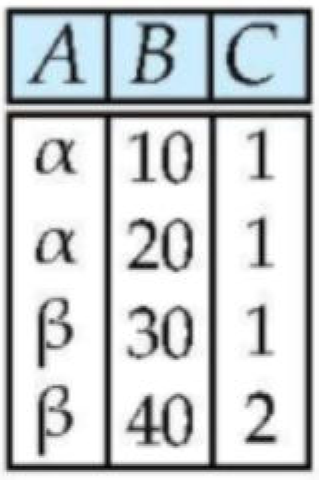
$$\sigma_{A=B \ \wedge \ D > 5} (r)$$

PROJECT operation

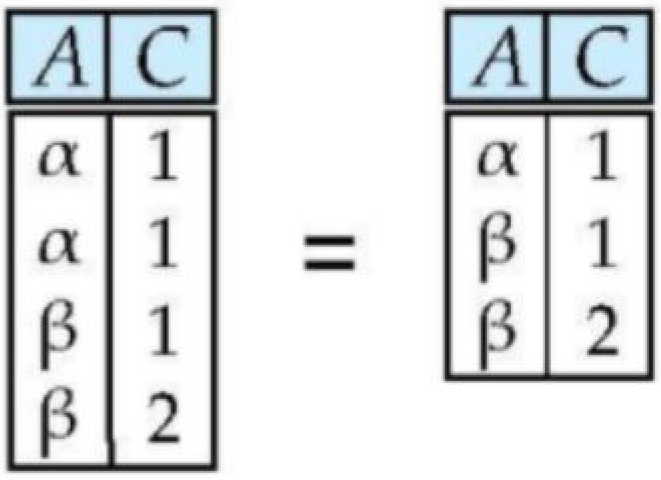
- Notation: $\Pi_{A_1, A_2, ... A_k}(r)$

  where A_1, A_2 are attribute names and r is a relation
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed.
- Duplicate rows removed from result, since relations are sets
- **Example:** To eliminate the `dept_name` attribute of `instructor`

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



A	B	C
α	10	1
α	20	1
β	30	1
β	40	2



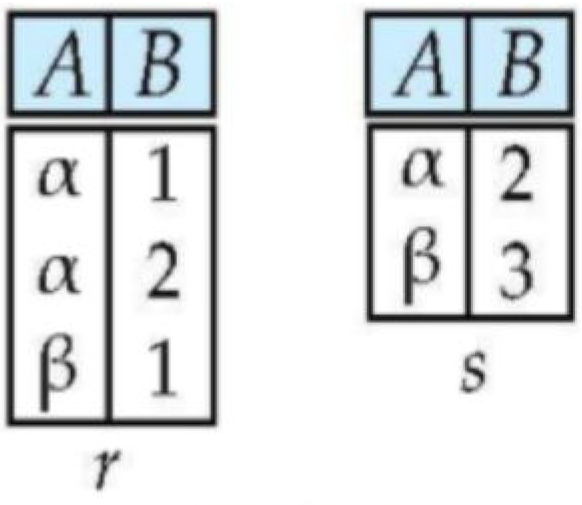
A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

UNION operation

- Notation: $r \cup s$
- Defined as: $r \cup s = \{t | t \in r \text{ or } t \in s\}$
- For $r \cup s$ to be valid:
    - $r,\ s$ must have the same **arity** (same number of attributes)
    - The attribute domains must be compatible (ie: same data type)
    - **Example:** To find all the courses taught in the Fall 2009 semester or in the Spring 2010 semester or in both
      $\Pi_{course\_id}(\sigma_{semester="Fall" \wedge year=2009}(section)) \cup \Pi_{course\_id}(\sigma_{semester="Spring" \wedge year=2010}(section))$

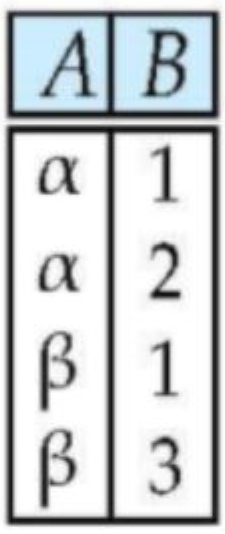


A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

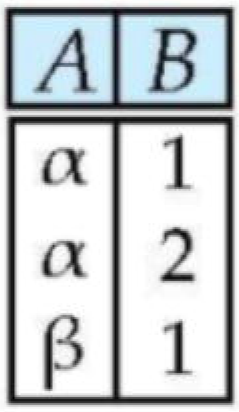


A	B
α	1
α	2
β	1
β	3

r ∪ s

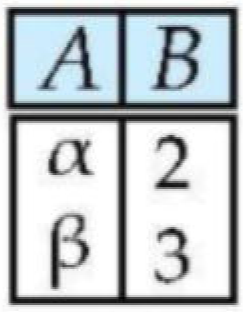
DIFFERENCE operation

- Notation: $r - s$
- Defined as: $r - s = \{t | t \in r \ and \ t \notin s\}$
- 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 the courses taught in the Fall 2009 semester, but not in the Spring 2010 semester
  $\Pi_{course\_id}(\sigma_{semester="Fall" \land \ year=2009}(section)) - \Pi_{course\_id}(\sigma_{semester="Spring" \land \ year=2010}(section))$



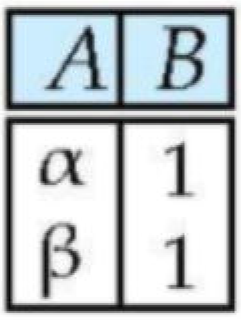
A	B
α	1
α	2
β	1

r



A	B
α	2
β	3

s

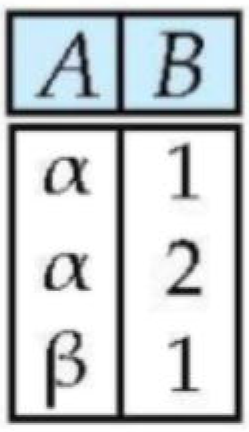


A	B
α	1
β	1

$r - s$

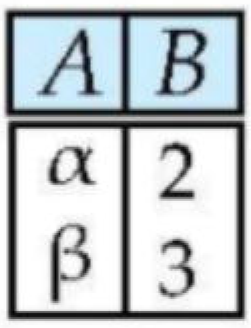
INTERSECTION operation

- Notation: $r \cap s$
- Defined as:
  $r \cap s = \{t | t \in r \ and \ t \in s\}$
- Assume:
  - $r, \ s$ have the same ability
  - Attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$



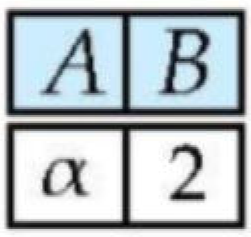
A	B
α	1
α	2
β	1

r



A	B
α	2
β	3

s



A	B
α	2

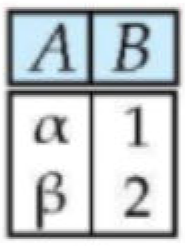
$r \cap s$

CARTESIAN-PRODUCT operation

- Notation: $r \times s$
- Defined as:

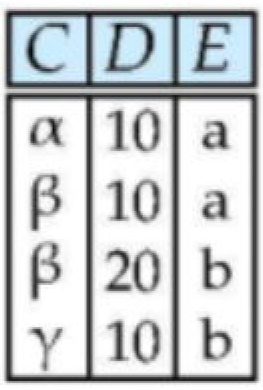
  $r \times s = \{t\ q | t \in r\ and\ q \in s\}$
- Assume that attributes of $r(R)$ and $s(S)$ are disjoint

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



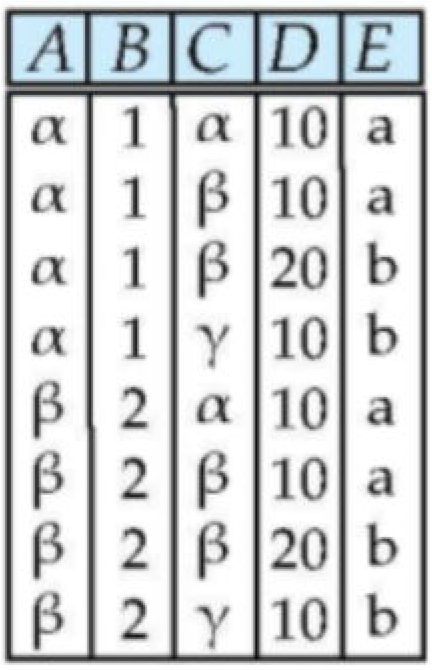
A	B
α	1
β	2

r



C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

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

r x *s*

RENAME operation

- Allows us to name and, therefore, refer to the results of relational-algebra expressions
- Allows us to refer to a relation by more than one name
- **Example:**

  $\rho_x(E)$

  returns the expression E under the name X
- If a relational algebra expression E has arity n, then

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

  returns the result of the expression E under the name X and with the attributes renamed to

  $A_1, A_2, ..., A_n$

DIVISION operation

- The division operation is applied to two relations
- $R(Z) \div S(X)$, where X subset Z
- Let $Y = Z - X$ (and hence $Z = X \cup Y$)

  that is, let Y be the set of attributes of R that are not attributes of S
- The result of **DIVISION** is a relation $T(Y)$ that includes a tuple t if tuples t_R appear in R with $t_R[Y] = t$, and with
  - $t_R[X] = t_s$ for every tuple t_S in S
- For a tuple t to appear in the result T of the **DIVISION**, the value in t must appear in R in combination with every tuple in S
- Division is a derived operation and can be expressed in terms of other operations
- $r \div s \equiv \Pi_{R-S}(r) - \Pi_{R-S}(r)((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$

DIVISION Example #1

R

Lecturer	Module
Brown	Compilers
Brown	Databases
Green	Prolog
Green	Databases
Lewis	Prolog
Smith	Databases

S

Subject
Prolog

R | S

Lecturer
Green
Lewis

DIVISION Example #2

R

Lecturer	Module
Brown	Compilers
Brown	Databases
Green	Prolog
Green	Databases
Lewis	Prolog
Smith	Databases

S

Subject
Databases
Prolog

R | S

Lecturer
Green

DIVISION Example #3

A

sno	pno

B1

pno

A / B1

sno

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

sno
s1
s2
s3
s4

A / B2

sno
s1
s4

A / B3

sno
s1

DIVISION Example #4

- Relation r, s

r

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
∈	6
∈	1
β	2

s

B
1
2

r ÷ s

A
α
β

DIVISION Example #5

- Relation r, s:

r

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

s

D	E
a	1
b	1

r ÷ s

A	B	C
α	a	γ
γ	a	γ

eg: Students who have taken both "a" and "b" courses, with instructor "1"

(Find all the students who have taken all courses given by the instructor 1)

Week 4 Lecture 2

Class	BSCCS2001
Created	@September 30, 2021 10:23 AM
Materials	
Module #	17
Type	Lecture
Week #	4

Formal Relational Query Languages (part 2)

Predicate Logic

Predicate Logic or **Predicate Calculus** is an extension of **Propositional Logic** or **Boolean Algebra**

It adds the concept of predicates and quantifiers to better capture the meaning of statements that cannot be adequately expressed by propositional logic

Tuple Relational Calculus and **Domain Relational Calculus** are based on **Predicate Calculus**

Predicate

- Consider the statement: "x is greater than 3"
  - It has 2 parts
  - The first part is the variable **x**
    - It is the subject of the statement
  - The second part **"is greater than 3"**
    - It is the predicate of the statement
    - This refers to the property that the subject of the statement can have
- The statement "x is greater than 3" can be denoted by $P(x)$ where P denotes the predicate "is greater than 3" and x is the variable
- The predicate P can be considered as a function. It tells the truth value of the statement $P(x)$ at x
  - Once a value has been assigned to the variable x, the statement $P(x)$ becomes a proposition and has a $truth$ or $false$ value

- In general, a statement involving n variables $x_1, x_2, x_3, ..., x_n$ can be denoted by $P(x_1, x_2, x_3, ..., x_n)$
  - Here, P is also referred to as the n-place predicate or an n-ary predicate

Quantifiers

In predicate logic, predicates are used alongside quantifiers to express the extent to which a predicate is true over a range of elements

Using ***quantifiers*** to create such propositions is called ***quantification***

There are 2 types of quantifiers:

- **Universal Quantifier**
- **Existential Quantifier**

Universal Quantifier

Universal Quantification: Mathematical statements sometimes assert that a property is true for all the values of a variable in a particular domain, called the **Domain of Discourse**

- Such a statement is expressed using universal quantification
- The universal quantification of $P(x)$ for a particular domain is the proposition that assert that $P(x)$ is $true$ for all values of x in this domain
- The domain is very important here since it decides the possible values of x
- Formally, the universal quantification of $P(x)$ is the statement "$P(x)$ for all values of x in the domain"
- The notation $\forall P(x)$ denotes the universal quantification of $P(x)$
  - Here, $\forall$ is called the **universal quantifier**
  - $\forall P(x)$ is read as ***"for all x P(x)"***
- **Example:** Let $P(x)$ be the statement $"x + 2 > x"$
  - What is the truth value of the statement $\forall x P(x)?$
- **Solution:** As $x + 2$ is greater than x for any real number, so $P(x) \equiv T$ for all x or $\forall x P(x) \equiv T$

Existential Quantifier

Existential Quantification: Some mathematical statements assert that there is an element with a certain property

Such statements are expressed by existential quantification

Existential quantification can be used to form a proposition that is true if and only if $P(x)$ is $true$ for at least one value of x in the domain

- Formally, the existential quantification of $P(x)$ is the statement "There exists an element x in the domain such that $P(x)$"
- The notation $\exists P(x)$ denotes the existential quantification of $P(x)$
  - Here $\exists$ is called the existential quantifier
  - $\exists P(x)$ is read as "There is at least one such x such that $P(x)$"
- **Example:** Let $P(x)$ be the statement $"x > 5"$
  - What is the truth value of the statement $\exists x P(x)?$
- **Solution:** $P(x)$ is $true$ for all real numbers greater than 5 and $false$ for all real numbers less than 5
  - So, $\exists x P(x) \equiv T$

Tuple Relational Calculus

TRC is a non-procedural query language, where each query is of the form

$\{t | P(t)\}$

where t = resulting tuples

$P(t)$ = known as predicate and these are the conditions that are used to fetch t

$P(t)$ may have various conditions logically combined with **OR($\vee$), AND($\wedge$), NOT($\neg$)**

It also uses quantifiers:

$\exists t \in r(Q(t))$ = "there exists" a tuple in t in relation r such that predicate $Q(t)$ is true

$\forall t \in r(Q(t))$ = $Q(t)$ is true "for all" tuples in relation r

- $\{P | \exists S \in Students\ and\ (S.CGPA > 8 \wedge P.name = S.name \wedge P.age = S.age)\}$:
  returns the name and age of students with a CGPA above 8

Predicate Calculus Formula

- Set of attributes and constants
- Set of comparison operators: (eg: $<, \leq, =, \neq, >, \geq$)
- Set of connectives: and($\wedge$), or($\vee$), not($\neg$)
- Implication ($\Rightarrow$)**:** $x \Rightarrow y$, if x is true, then y is true
  $x \Rightarrow y \equiv \neg x \vee y$
- Set of quantifiers:
  - $\exists t \in r(Q(t)) \equiv$ "there exists" a tuple in t in relation r such that predicate $Q(t)$ is true
  - $\forall t \in r(Q(t)) \equiv Q$ is true "for all" tuples t in relation r

TRC Example #1

Student

Fname	Lname	Age	Course
David	Sharma	27	DBMS
Aaron	Lilly	17	JAVA
Sahil	Khan	19	Python
Sachin	Rao	20	DBMS
Varun	George	23	JAVA
Simi	Verma	22	JAVA

Q. 1: Obtain the first name of students whose age is greater than 21

Solution:

$\{t.Fname \mid Student \wedge t.age > 21\}$

$\{t.Fname \mid t \in Student \wedge t.age > 21\}$

$\{t \mid \exists s \in Student(s.age > 21 \wedge t.Fname = s.Fname)\}$

Fname
David
Varun
Simi

TRC Example #2

Consider the relational schema

```
student(rollNo, name, year, courseId)
course(courseId, cname, teacher)
```

Q. 2: Find out the names of all the students who have taken the course named 'DBMS'

- $\{t \mid \exists s \in student\ \exists c \in course(s.courseId = c.courseId \wedge c.cname =' DBMS' \wedge t.name = s.name)\}$
- $\{s.name \mid s \in student \wedge \exists c \in course(s.courseId = c.courseId \wedge c.cname =' DBMS')\}$

Q. 3: Find out the names of all students and their rollNo who have taken the course named 'DBMS'

- $\{s.name, s.rollNo \mid s \in student \land \exists c \in course(s.courseId = c.courseId \land c.cname =' DBMS')\}$
- $\{t \mid \exists s \in student\ \exists c \in course(s.courseId = c.courseId \land c.cname =' DBMS' \land t.name = s.name \land t.rollNo = s.rollNo)\}$

TRC Example #3

Consider the following relations:

```
Flights(flno, from, to, distance, departs, arrive)
Aircraft(aid, aname, cruisingrange)
Certified(eid, aid)
Employees(eid, ename, salary)
```

Q. 4: Find the eids of pilots certified for Boeing aircraft

RA

$\Pi_{eid}(\sigma_{aname='Boeing'}(Aircraft \bowtie Certified))$

TRC

- $\{C.eid \mid C \in Certified \land \exists A \in Aircraft(A.aid = C.aid \land A.aname =' Boeing')\}$
- $\{T \mid \exists C \in Certified\ \exists A \in Aircraft(A.aid = C.aid \land A.aname =' Boeing' \land T.eid = C.eid)\}$

TRC Example #4

Consider the following relations:

```
Flights (flno, from, to, distance, departs, arrives)
Aircraft (aid, aname, cruisingrange)
Certified (eid, aid)
Employees (eid, ename, salary)
```

Q. 5: Find the names and salaries of certified pilots working on Boeing aircrafts

RA

$\Pi_{ename,salary}(\sigma_{aname='Boeing'}(Aircraft \bowtie Certified \bowtie Employees))$

TRC

$\{P \mid \exists E \in Employees\ \exists C \in Certified\ \exists A \in Aircraft(A.aid = C.aid \land A.aname =' Boeing' \land E.eid = C.eid \land P.ename = E.ename \land P.salary = E.salary)\}$

TRC Example #5

Consider the following relations:

```
Flights (flno, from, to, distance, departs, arrive)
Aircraft (aid, aname, cruisingrange)
Certified (eid, aid)
Employees (eid, ename, salary)
```

Q. 6: Identify the flights that can be piloted by every pilot whose salary is more than $\$100,000$

- $\{Fl.flno \mid F \in Flights \land \exists C \in Certified\ \exists E \in Employees(E.salary > 100,000 \land E.eid = C.eid)\}$

Safety of Expressions

- It is possible to write tuple calculus expressions that generate infinite relations
- For example, $\{t \mid \neg t \in r\}$ results in an infinite relation if the domain of any attribute of the relation r is infinite
- To guard against the problem, we restrict the set of allowable expressions to safe expressions
- An expression $\{t \mid P(t)\}$ in the tuple relational calculus is safe if every component of t appears in one of the relations, tuples or constants that appear in P
  - **NOTE:** This is more than just a syntax condition
  - Eg: $\{t \mid t[A] = 5 \lor true\}$ is not safe → it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in P

Domain Relational Calculus

- A non-procedural query language equivalent in power to the tuple relational calculus
- Each query is an expression of the form:

  $\{< x_1, x_2, ..., x_n > | P(x_1, x_2, ..., x_n)\}$

  - $x_1, x_2, ..., x_n$ represents domain variables
  - P represents a formula similar to that of the predicate calculus

Equivalence of Relational Algebra, Tuple Relational Calculus & Domain Relational Calculus

SELECT operation

$R = (A,\ B)$

Relational Algebra: $\sigma_{B=17}(r)$

Tuple Calculus: $\{t \mid t \in r \land B = 17\}$

Domain Calculus: $\{< a, b > \mid < a, b > \in r \land b = 17\}$

PROJECT operation

$R = (A,\ B)$

Relational Algebra: $\Pi_A(r)$

Tuple Calculus: $\{t \mid \exists p \in r(t[A] = p[A])\}$

Domain Calculus: $\{< a > \mid \exists\ b\ (< a, b > \in r)\}$

COMBINING operation

$R = (A,\ B)$

Relational Algebra: $\Pi_A(\sigma_{B=17}(r))$

Tuple Calculus: $\{t \mid \exists p \in r(t[A] = p[A] \land p[B] = 17)\}$

Domain Calculus: $\{< a > \mid \exists\ b\ (< a, b > \in r \land b = 17)\}$

UNION

$R = (A, B, C) \quad S = (A, B, C)$

Relational Algebra: $r \cup s$

Tuple Calculus: $\{t \mid t \in r \lor t \in s\}$

Domain Calculus: $\{< a, b, c > \mid < a, b, c > \in r \lor < a, b, c > \in s\}$

SET DIFFERENCE

$R = (A, B, C) \quad S = (A, B, C)$

Relational Algebra: $r - s$

Tuple Calculus: $\{t \mid t \in r \land t \notin s\}$

Domain Calculus: $\{< a, b, c > \mid < a, b, c > \in r \land < a, b, c > \notin s\}$

INTERSECTION

$R = (A, B, C) \quad S = (A, B, C)$

Relational Algebra: $r \cap s$

Tuple Calculus: $\{t \mid t \in r \land t \in s\}$

Domain Calculus: $\{< a, b, c > \mid < a, b, c > \in r \land < a, b, c > \in s\}$

CARTESIAN / CROSS PRODUCT

$R = (A, B) \quad S = (C, D)$

Relational Algebra: $r \times s$

Tuple Calculus: $\{t \mid \exists p \in r \exists q \in s(t[A] = p[A] \wedge t[B] = p[B] \wedge t[C] = q[C] \wedge t[D] = q[D])\}$

Domain Calculus: $\{< a, b, c, d > \mid < a, b > \in r \wedge < c, d > \in s\}$

NATURAL JOIN

$R = (A, B, C, D) \quad S = (B, D, E)$

Relational Algebra:

$r \bowtie s$

$\Pi_{r.A, r.B, r.C, r.D, s.E}(\sigma_{r.B = s.B \wedge r.D = s.D}(r \times s))$

Tuple Calculus:

$\{t \mid \exists\ p \in r\ \exists\ q \in s(t[A] = p[A] \wedge t[B] = p[B] \wedge t[C] = p[C] \wedge t[D] = p[D] \wedge t[E] = q[E] \wedge p[B] = q[B] \wedge p[D] = q[D])\}$

Domain Calculus:

$\{< a, b, c, d, e > \mid < a, b, c, d > \in r \wedge < b, d, e > \in s\}$

DIVISION

$R = (A, B) \quad S = (B)$

Relational Algebra: $r \div s$

Tuple Calculus: $\{t \mid \exists\ p \in r\ \forall\ q \in s(p[B] = q[B] \Rightarrow t[A] = p[A])\}$

Domain Calculus: $\{< a > \mid < a > \in r \wedge \forall < b > (< b > \in s \Rightarrow < a, b > \in r)\}$

Source: *https://www2.cs.sfu.ca/CourseCentral/354/louie/Equiv_Notations.pdf*

Week 4 Lecture 3

Class	BSCCS2001
Created	@September 30, 2021 4:40 PM
Materials	
Module #	18
Type	Lecture
Week #	4

Entity-Relationship Model

Design Process

What is a Design?

A Design:

- Satisfies a given (perhaps informal) functional specification
- Conforms to the limitations of the target medium
- Meets implicit or explicit requirements on performance and resource usage
- Satisfies implicit or explicit design criteria on the form of the artifact
- Satisfies restrictions on the design itself, such as its length or cost, or the tools available for doing the design

Role of Abstraction

- Disorganized Complexity results from
    - Storage (STM) limitations of the human brain - an individual can simultaneously comprehend of the order of seven, plus or minus two chunks of information
    - Speed limitations of human brain - it takes the mind about five seconds to accept a new chunk of information
- **Abstraction** provides the major tool to handle Disorganized Complexity by chunking information
- Ignore in-essential details, deal only with the generalized, idealized model of the world

    Consider: A binary number `110010101001`

    Hard to remember

Try the octal form: `(110)(010)(101)(001)` $\Longrightarrow$ **6251**

Or the hex form: `(1100)(1010)(1001)` $\Longrightarrow$ **CA9**

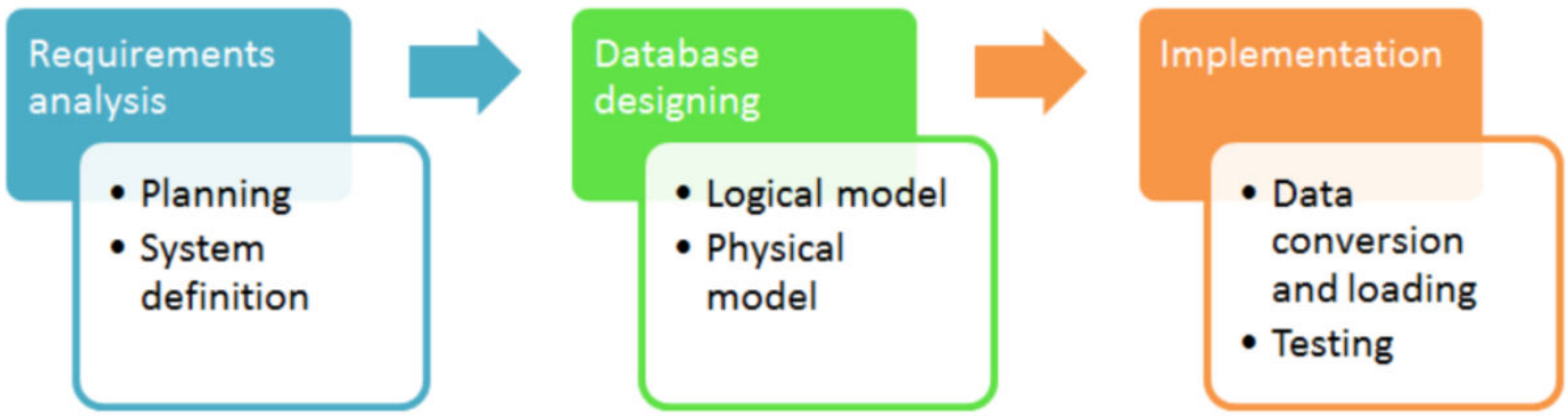
Model Building

- Physics
  - Time-Distance Equation
  - Quantum Mechanics
- Chemistry
  - Valency-bond Structures
- Geography
  - Maps
  - Projections
- Electrical Circuits
  - Kirchoff's Loop Equations
  - Time Series Signals and FFT
  - Transistor Models
  - Schematic Diagrams
  - Interconnect Routing
- Building & Bridges
  - Drawings - Plan, Elevation, Side view
  - Finite Element Models

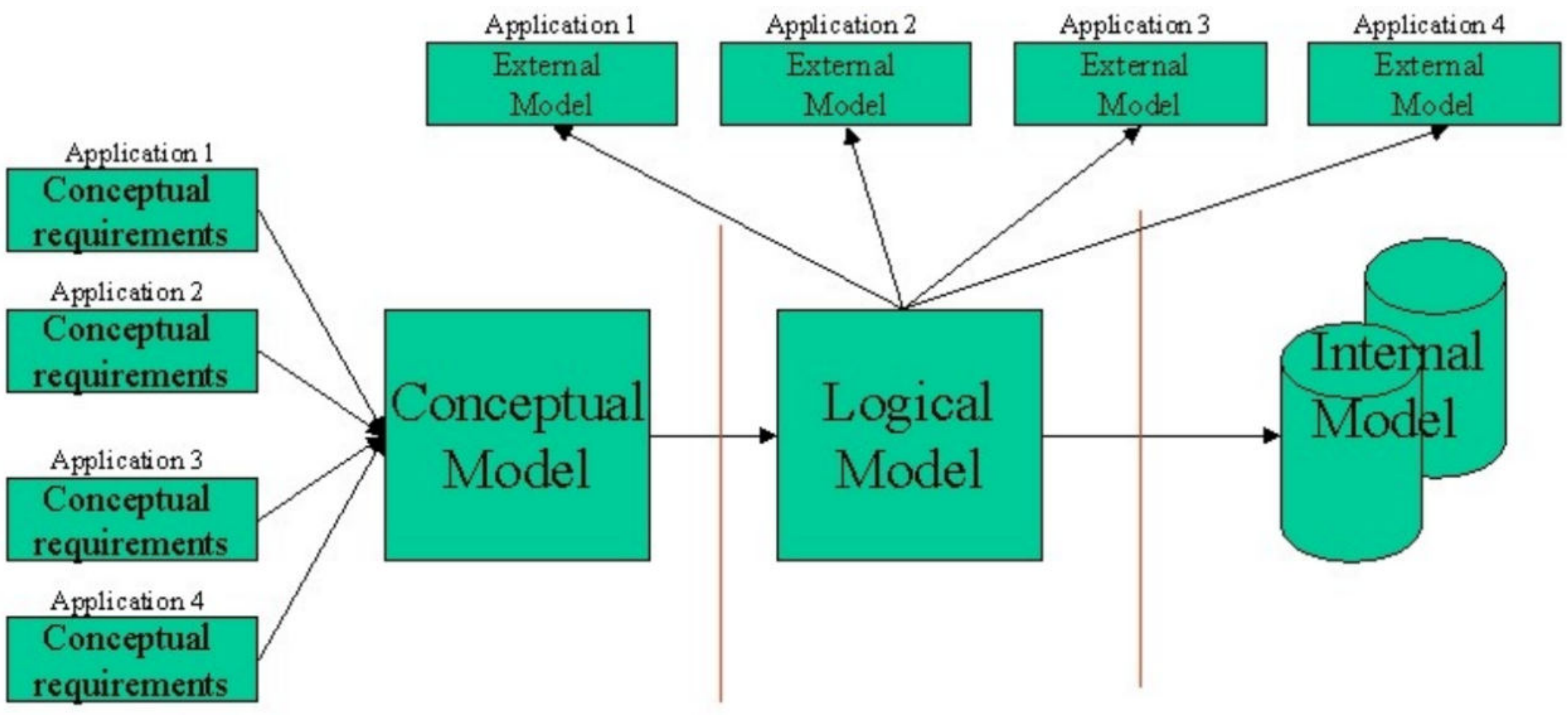
- Models are common in all engineering disciplines
- Model building follows principles of decomposition, abstraction and hierarchy
- Each model describes a specific aspect of the system
- Build new models upon old proven models

Design Approach

- **Requirement Analysis:** Analyse the data needs of the prospective DB users
  - Planning
  - System Defining
- **DB Designing:** Use a modeling framework to create abstraction of the real world
  - Logical Model
  - Physical Model
- **Implementation**
  - Data Conversion and Loading
  - Testing



- **Logical Model:** Deciding on a good DB schema
  - Business Decision: What attributes should we record in the DB?
  - Computer Science Decision: What relation schema should we have and how should the attributes be distributed among the various relation schema?
- **Physical Model:** Deciding on the physical layout of the DB



- **Entity Relationship Model**
    - Models an enterprise as a collection of entities and relationships
        - Entity → A distinguishable "thing" or "object" in the enterprise
            - Described by a set of attributes
        - Relationship → An association among multiple entities
    - Represented by an **Entity-Relationship** *or* **ER diagram**
- **Database Normalization**
    - Formalize what designs are bad and test for them

Entity Relationship (ER) Model

ER Model: Database Modeling

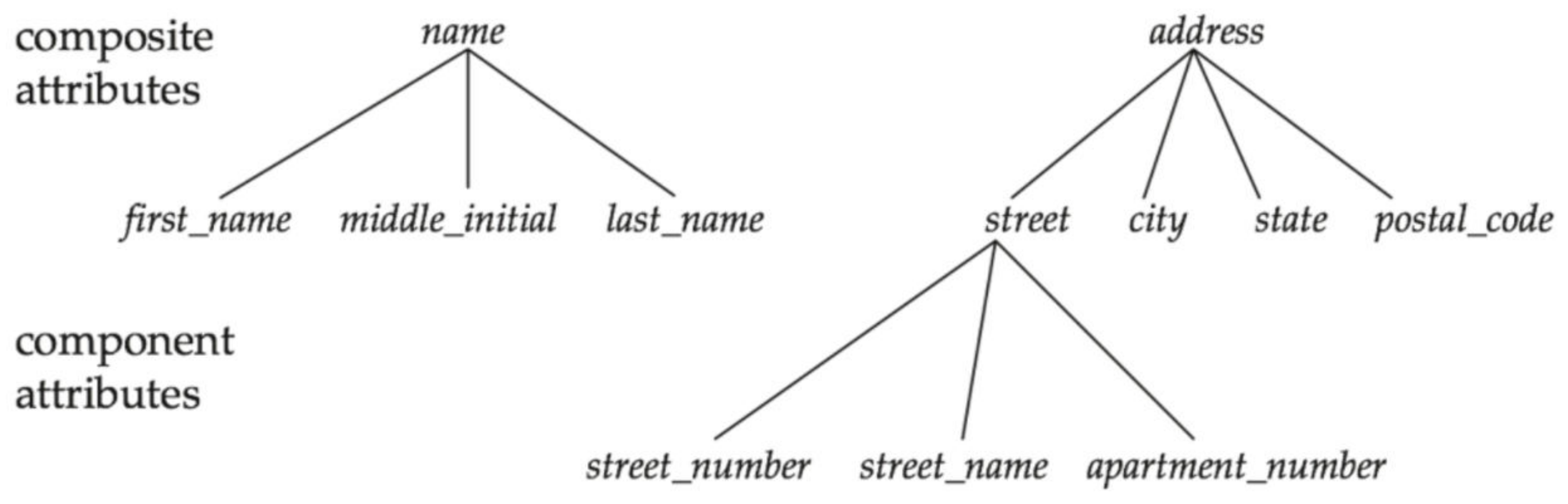
- The ER data model was developed to facilitate DB design by allowing specification of an enterprise schema that represents the overall logical structure of a DB
- The ER model is useful in mapping the meanings and interactions of the real world enterprises onto a conceptual schema
- The ER data model employs three basic concepts:
    - Attributes
    - Entity sets
    - Relationship sets
- The ER model also has an associated diagrammatic representation, the ER diagram, which can express the overall logical structure of a DB graphically

Attributes

- An attribute is a property associated with an entity / entity set
- Based on the values of certain attributes, an entity can be identified uniquely
- Attribute types:
    - Simple and Composite attributes
    - Single-valued and Multi-valued attributes
        - **Example:** Multi-valued attribute: *phone_numbers*
    - Derived attributes
        - Can be computed from other attributes
        - Example: age, given date_of_birth

- Domain: The set of permitted values for each attribute

Attributes: Composite



Entity sets

- An entity is an object that exists and is distinguishable from other objects
  - **Example:** specific person, company, event, plant
- An entity set is a set of entities of the same type that share the same properties
  - **Example:** set of all persons, companies, trees, holidays
- An entity is represented by a set of attributes: ie, descriptive properties possessed by all members of an entity set
  - **Example:**

```
instructor = (ID, name, street, city, salary)
course = (course_id, title, credits)

-- Here ID and course_id are the primary keys, but
-- the tool I am using to make PDFs is not marking them underline
```

- A subset of the attributes form a primary key of the entity set; that is, uniquely identifying each member of the set
  - Primary key of an entity set is represented by underlining it

Entity sets - instructor and student

instructor

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

student

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

Relationship sets

- A relationship is an association among several entities

  Example:

44553 (Peltier)	*advisor*	22222 (Einstein)
student entity	relationship set	instructor entity

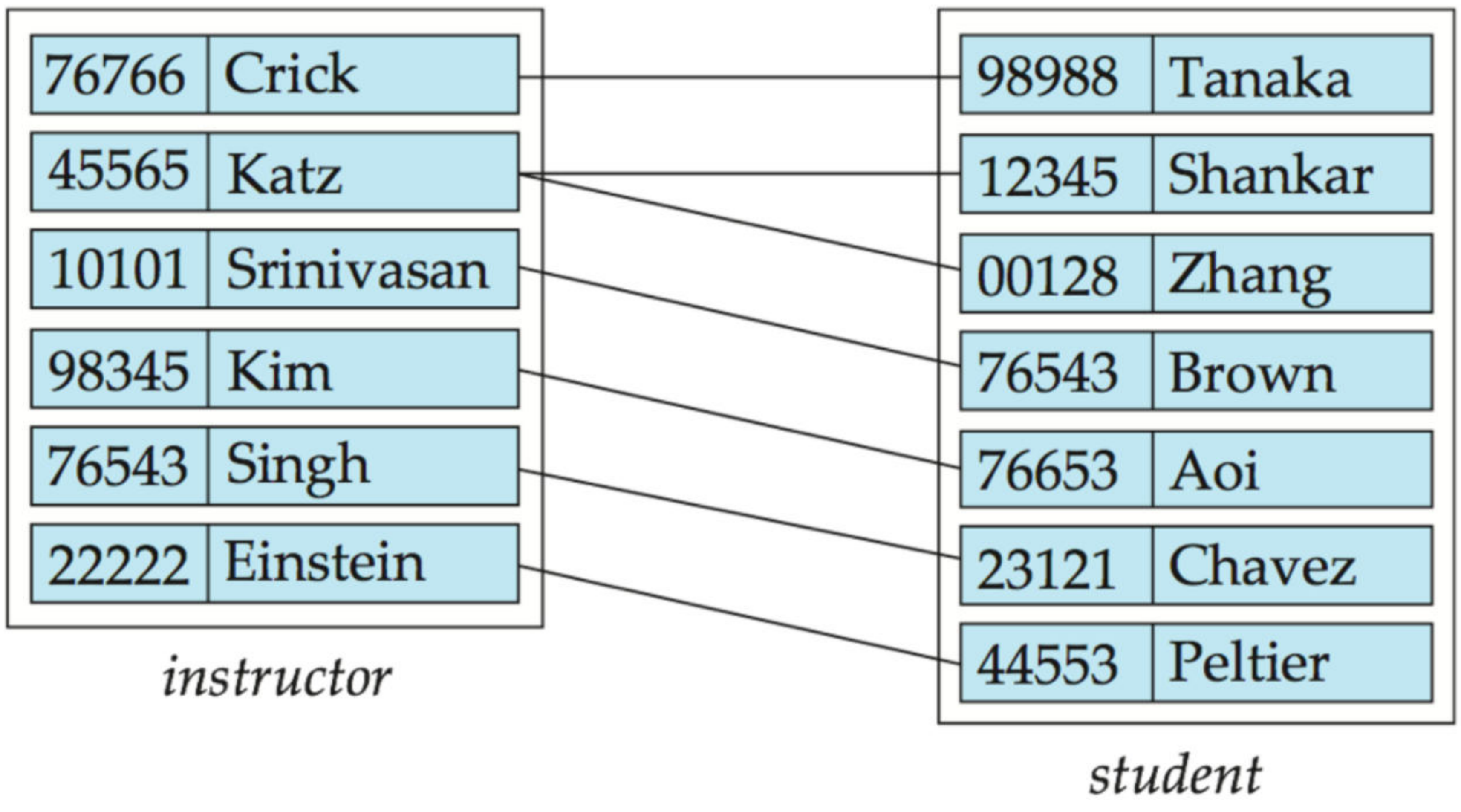
- A relationship set is a mathematical relation among $n \geq 2$ entities, each taken form entity sets

  $\{(e_1, e_2, ..., e_n) | e_1 \in E_1, e_2 \in E_2, ..., e_n \in E_n\}$

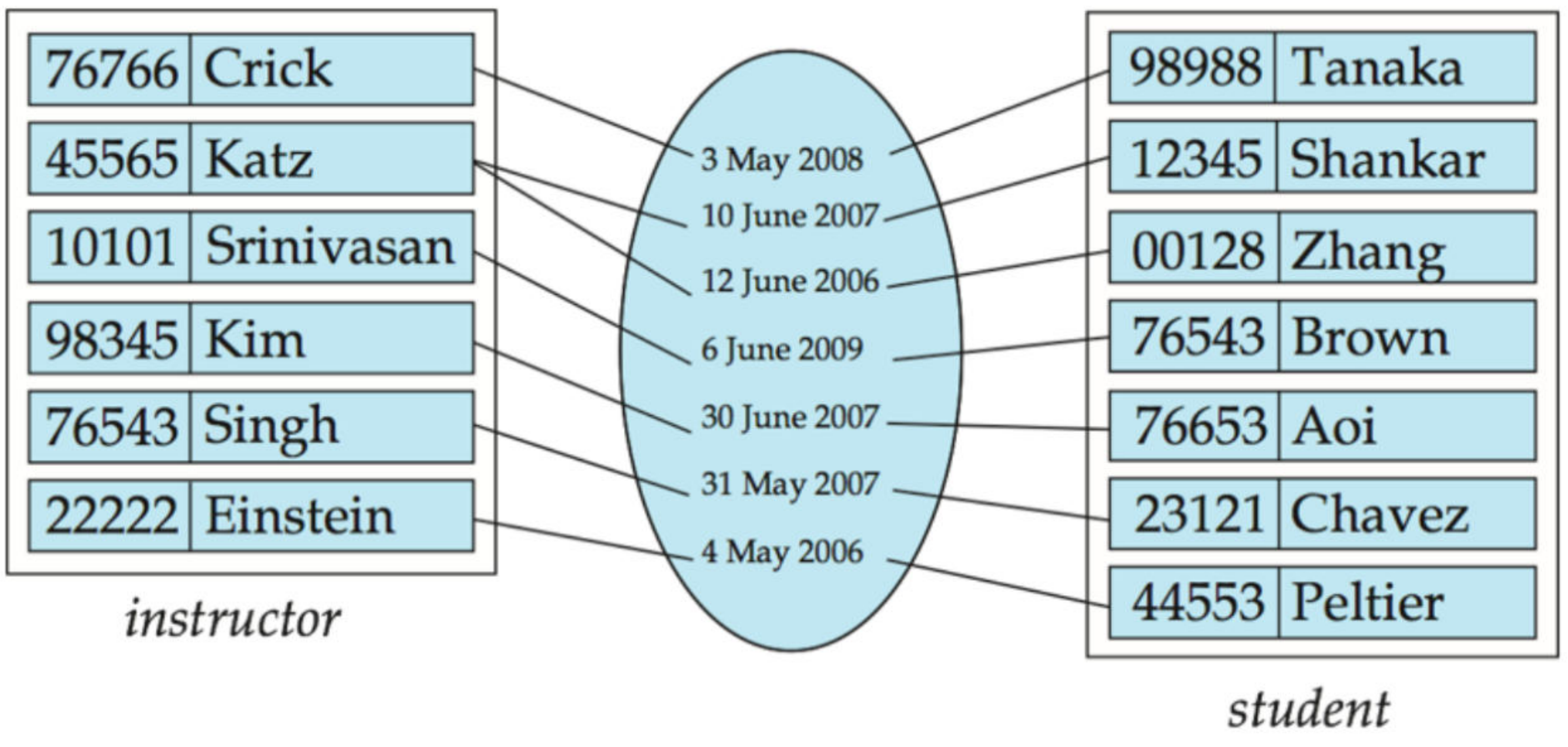
  where $(e_1, e_2, ..., e_n)$ is a relationship

  - **Example:** $(44553, 22222) \in advisor$

Relationship set: advisor



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



- **Binary relationship**
  - involves two entity sets (or degree two)
  - most relationship sets in a database systems are binary
- Relationships between more than two entity sets are rare
  - Most relationships are binary
  - **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: Redundant

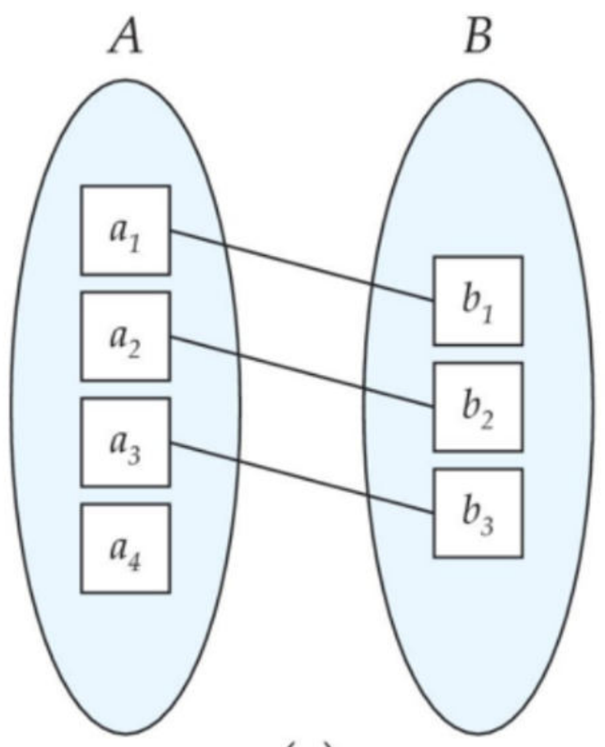
- Suppose we have entity sets:

  - *instructors*, with attributes: *ID, name, dept_name, salary*
  - *department*, with attributes: *dept_name, building, budget*
- We model the fact that each instructor has an associated department using a relationship set *inst_dept*
- The attribute *dept_name* appears in both entity sets
  - Since it is the primary key for the entity set department, it replicates information present in the relationship and is therefore redundant in the entity set instructor and needs to be removed
- **BUT:** When converting back to tables, in some cases the attributes gets re-introduced, as we will see later

Mapping Cardinality: Constraints

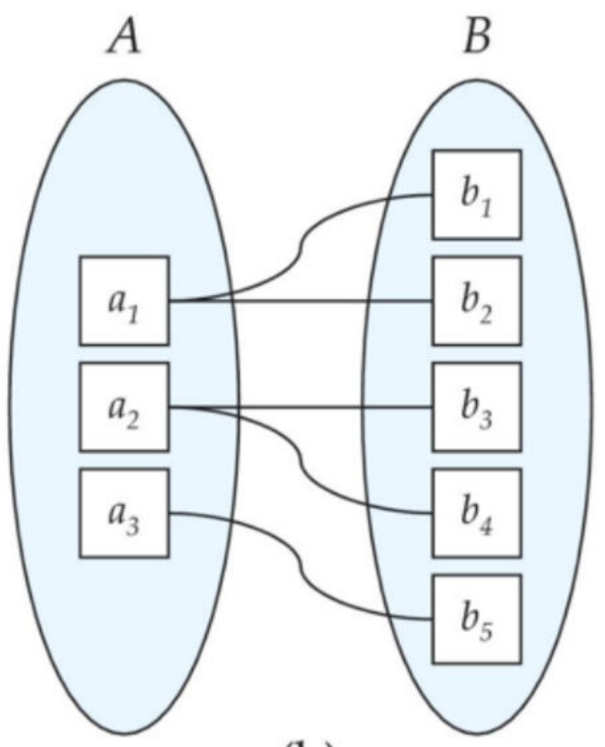
- Express the number of entities to which another entity can be associated via a relationship set
- Most useful in describing binary relationship sets
- For a binary relationship set the mapping cardinality must be one of the following types:
  - One to One
  - One to Many
  - Many to One
  - Many to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

Mapping Cardinalities



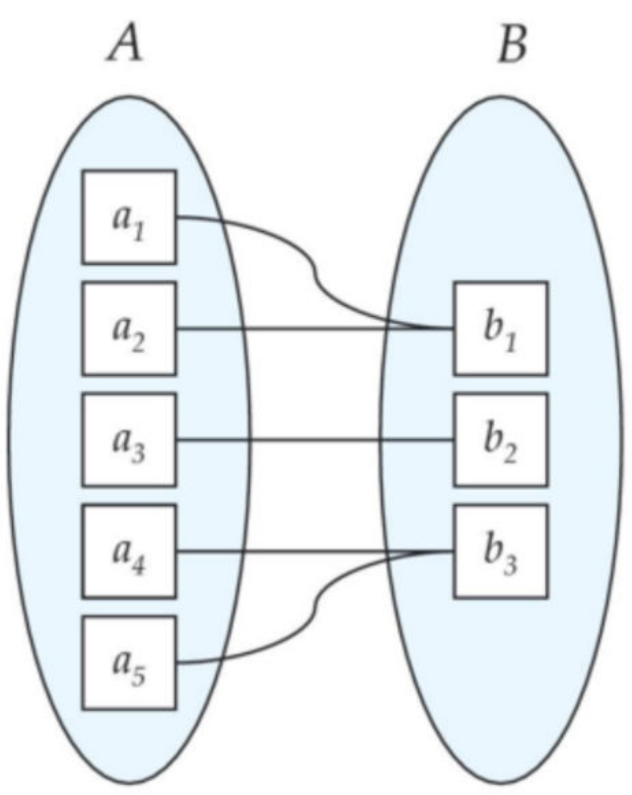
(a)

One to one



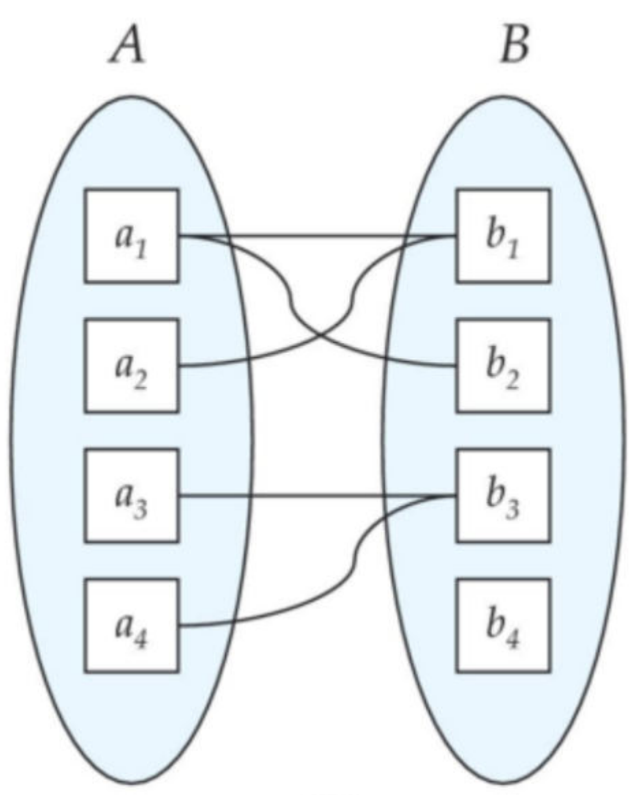
(b)

One to many



(a)

Many to one



(b)

Many to many

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

Weak Entity sets

An entity set may be one of the two types:

- Strong entity set
  - A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities
  - In other words, ***a primary key exists for a strong entity set***
  - Primary key of a strong entity set is represented by underlining it
- Weak entity set
  - A weak entity set is an entity set that does not contain sufficient attributes to uniquely identify its entities
  - In other words, ***a primary key does not exist for a weak entity set***
  - However, it contains a partial key called as the **discriminator**
  - Discriminator can identify a group of entities from the entity set
  - Discriminator is represented by underlining with a dashed line
- Since a weak entity set does not have a primary key, it cannot independently exist in the ER model
- It features in the model in relationship with a strong entity set
  - This is called as **the identifying relationship**
- Primary Key of a Weak entity set
  - The combination of discriminator and primary key of the strong entity set makes it possible to uniquely identify all entities of the weak entity set
  - Thus, this combination serves as a primary key for the weak entity set
  - Clearly, this primary key is not formed by the weak entity set completely
  - **Primary Key of a Weak Entity Set = Its own discriminator + Primary Key of Strong Entity Set**
- Weak entity set must have **total participation** in the identifying relationship
  - That is, all the entities must feature in the relationship

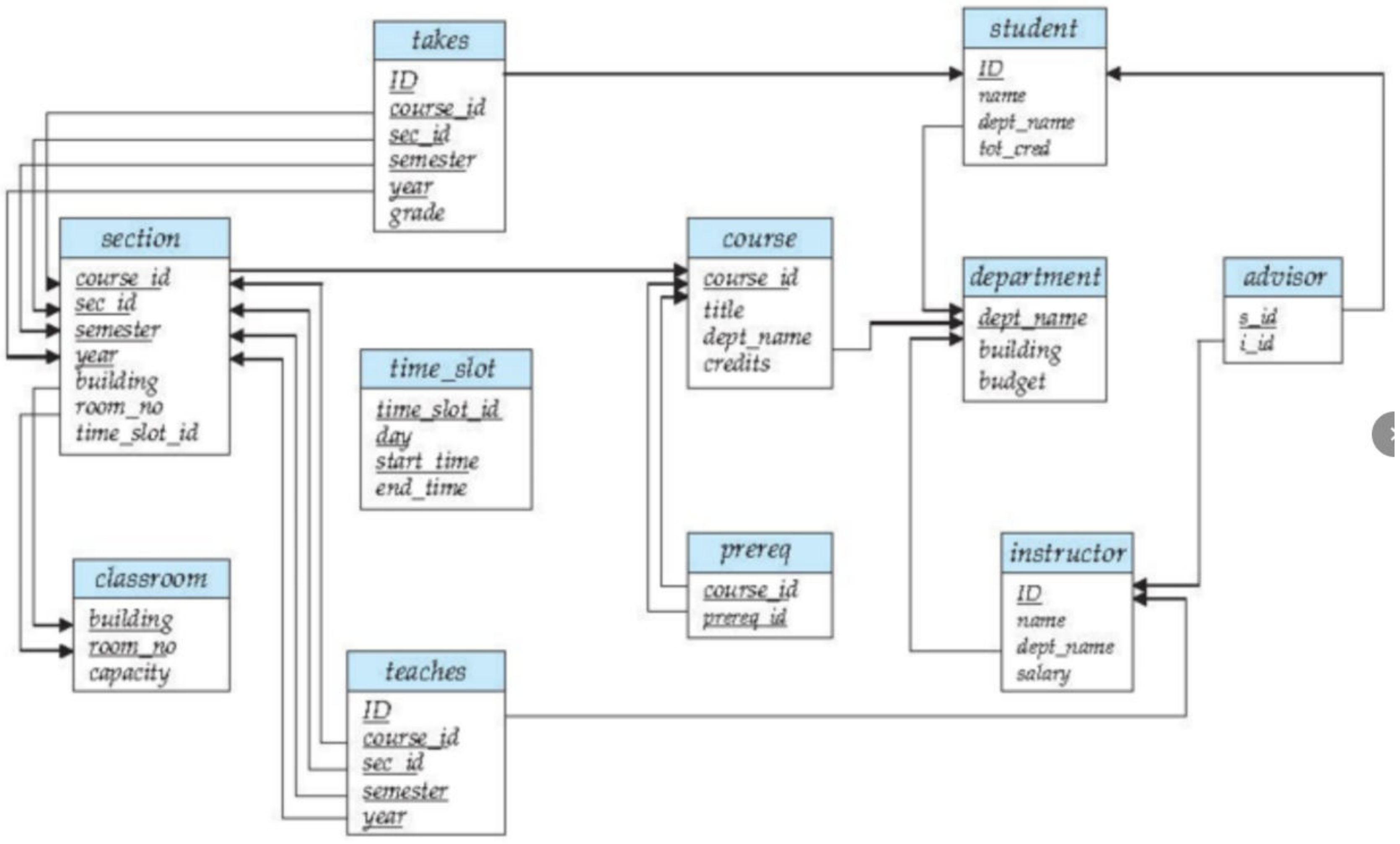
Weak Entity set: Example

- **Strong Entity Set:** *Building*(building_no, buildname, address)

   - *building_no* is the primary key here
- **Weak Entity Set:** *Apartment*(door_no, floor)
   - *door_no* is its discriminator as *door_no* alone can not identify an apartment uniquely
   - There may be several other buildings having the same door number
- **Relationship:** *BA* between *Building* and *Apartment*
- By **total participation** in *BA*, each apartment must be present in at least one building
- In contrast, *Building* has **partial participation** in *BA* only as there might exist some buildings which has not apartment
- **Primary Key:** To uniquely identify an apartment
   - First, *building_no* is required to identify the particular building
   - Second, *door_no* of the apartment is required to uniquely identify the apartment
- Primary Key of Apartment = Primary Key of the Building + Its own discriminator = *building_no* + *door_no*

Weak Entity set: Example #2

- Consider a section entity, which is uniquely identified by a *course_id*, *semester*, *year* and *sec_id*
- Clearly, section entities are related to course entities
   - Suppose we create a relationship set *sec_course* between entity sets *section* and *course*
- Note that the information in *sec_course* is redundant, since section already has an attribute *course_id,* which identifies the course with which the section is related



Week 4 Lecture 4

Class	BSCCS2001
Created	@September 30, 2021 6:29 PM
Materials	
Module #	19
Type	Lecture
Week #	4

Entity-Relationship Model (part 2)

ER Diagram

Entity Sets

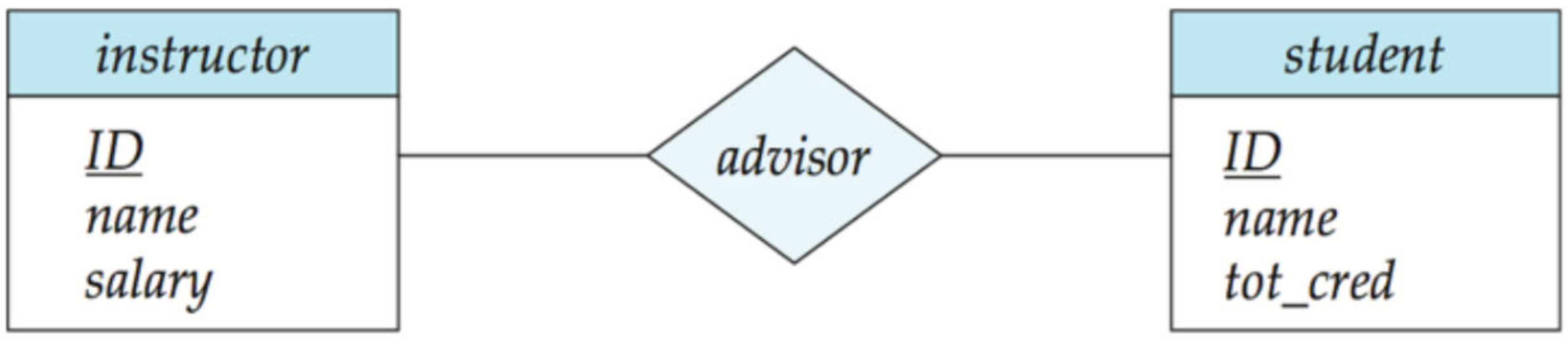
- Entities can be represented graphically as follows:
    - Rectangles represent entity set
    - Attributes are listed inside entity rectangle
    - Underline indicates primary key attributes

instructor
ID
name
salary

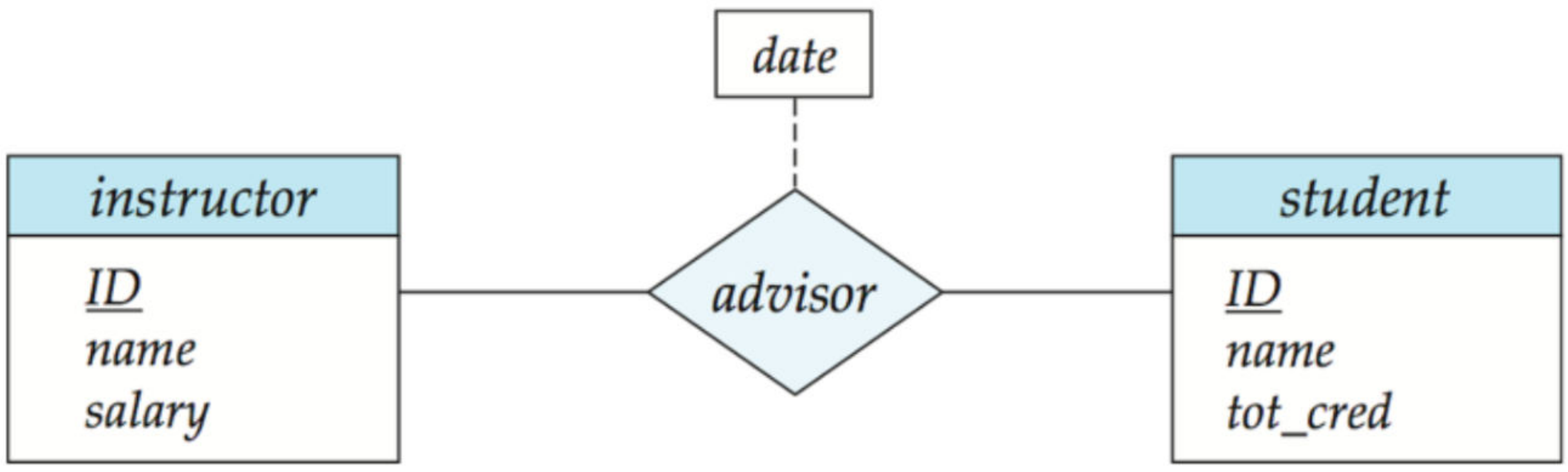
student
ID
name
tot_cred

Relationship sets

- Diamonds represent relationship sets

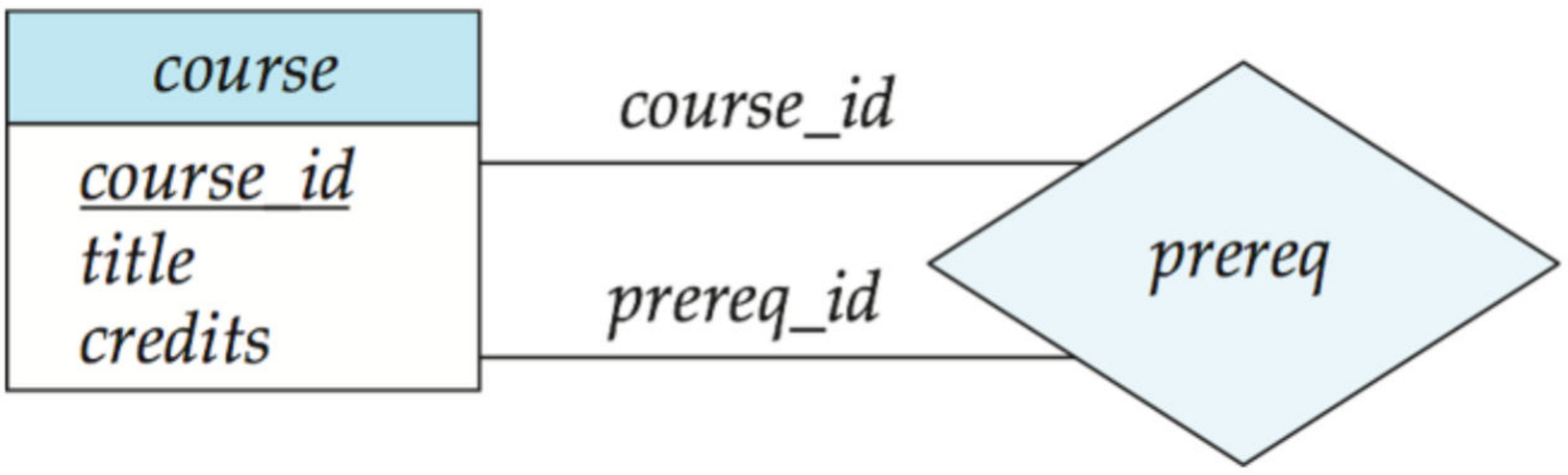


Relationship sets with attributes



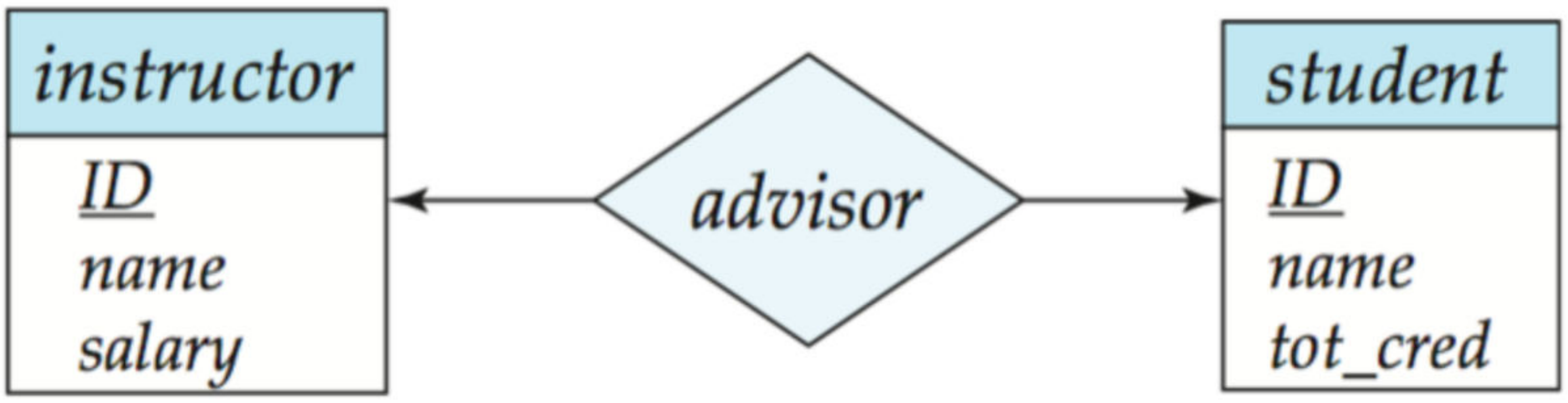
Roles

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



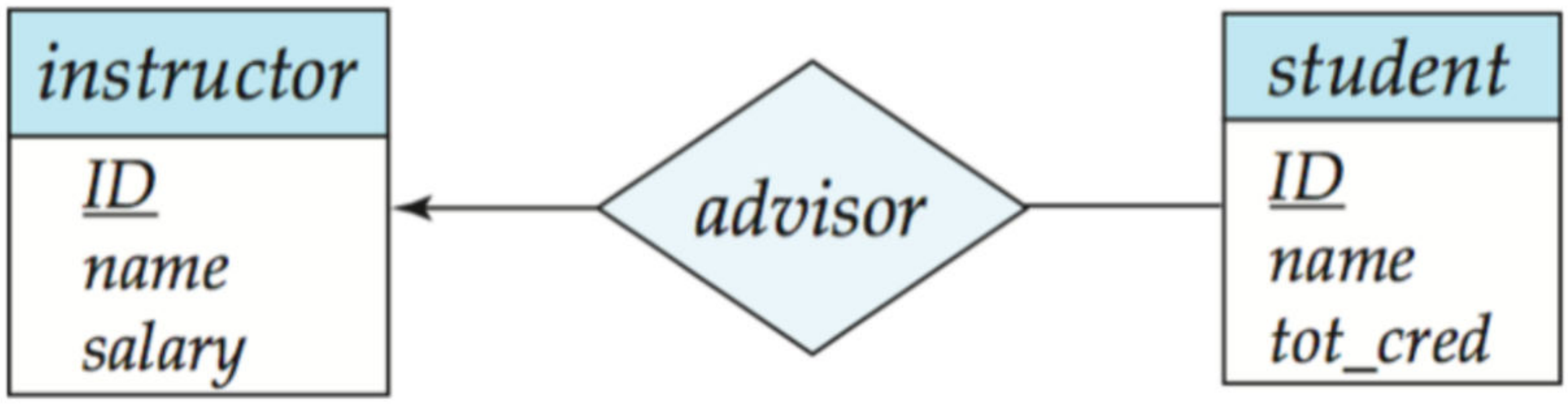
Cardinality Constraints

- 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 between an *instructor* and a *student*:
  - A student is associated with at most one instructor via the relationship *advisor*
  - An instructor is associated with at most one student via the relationship *advisor*



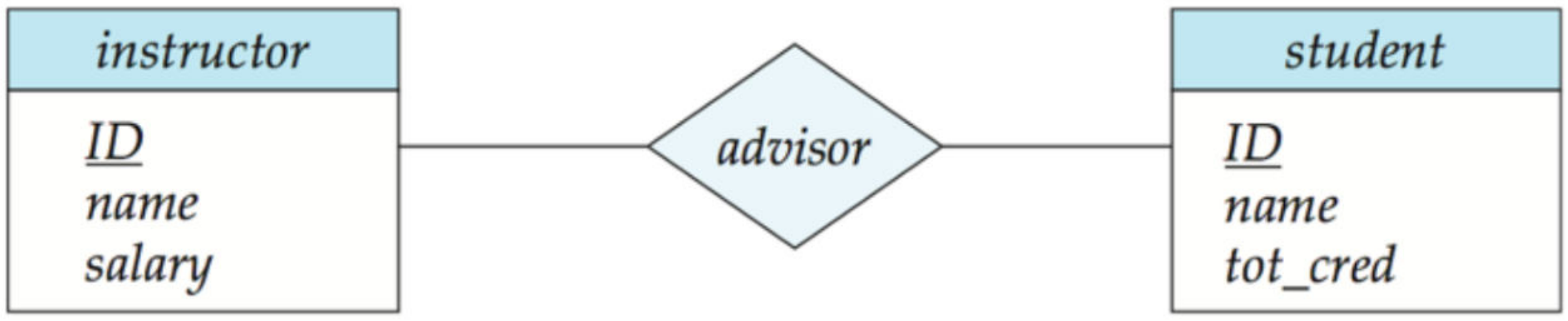
One-to-Many relationship

- 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*



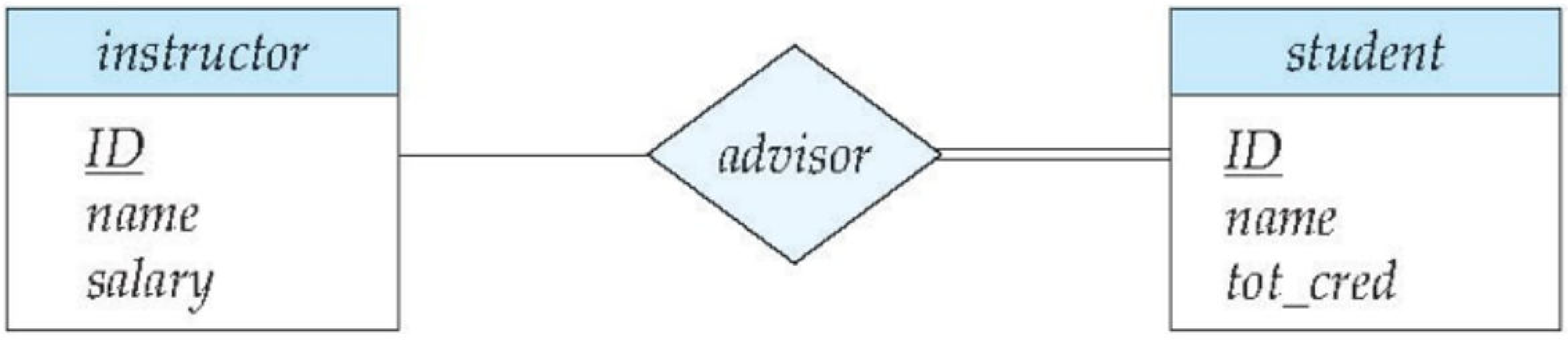
Many-to-Many relationship

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



Total and Partial participation

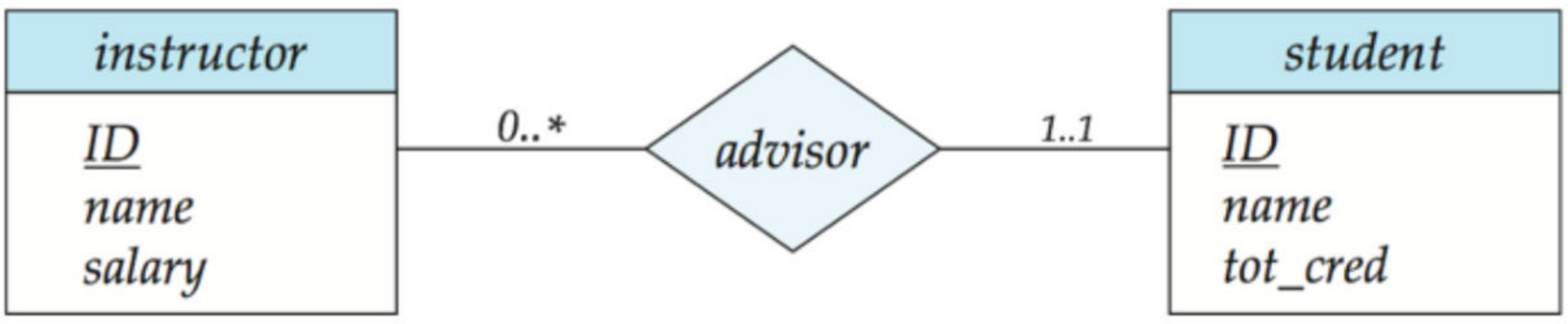
- Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set



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

Notation for expressing more complex constraints

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



Instructor can advise 0 or more students

A student must have 1 advisor; cannot have multiple advisors

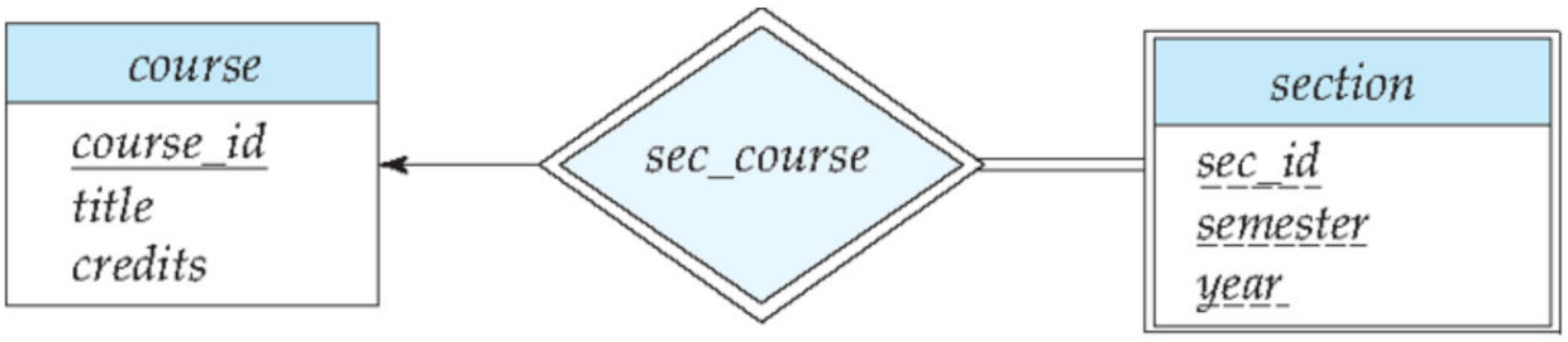
Notation to express entity with complex attributes

$instructor$

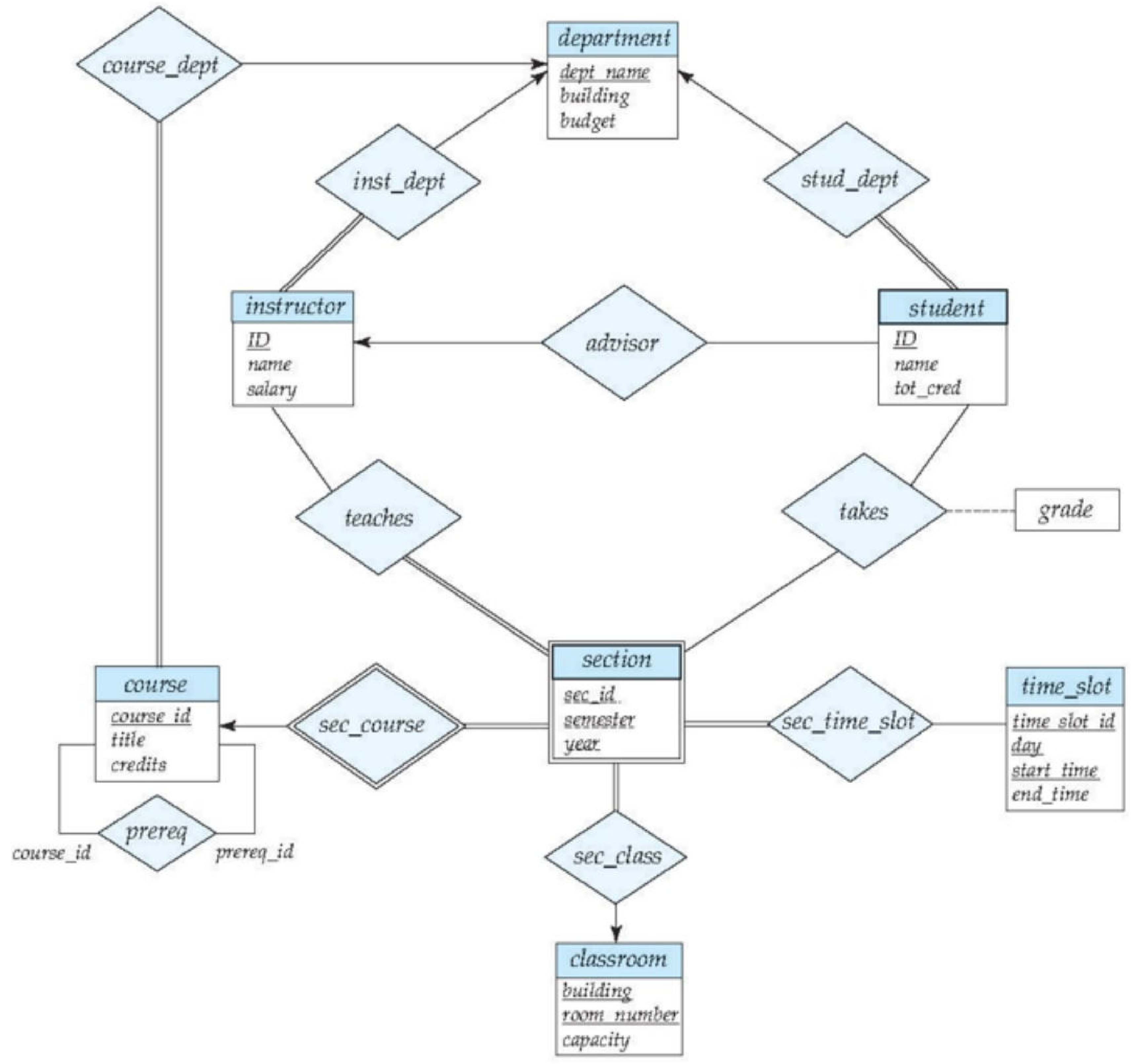
```
ID
name
  first_name
  middle_initial
  last_name
address
  street
    street_number
    street_name
    apt_number
  city
  state
  zip
{ phone_number }
date_of_birth
age()
```

Expressing Weak entity sets

- In ER 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*)



ER diagram for a University enterprise



ER Model to Relational Schema

Reduction to Relation Schema

- Entity sets and relationship sets can be expressed uniformly as *relation schemas* that represent the contents of the DB
- A DB which conforms to an ER diagram can be represented by a collection of schemas
- 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

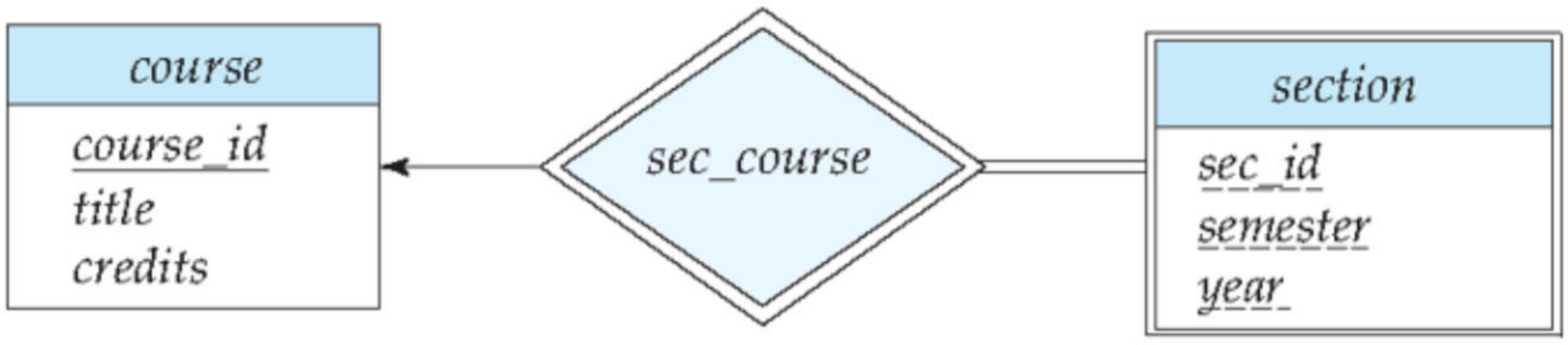
Representing entity sets

- A strong entity set reduces to a schema with the same attributes

  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

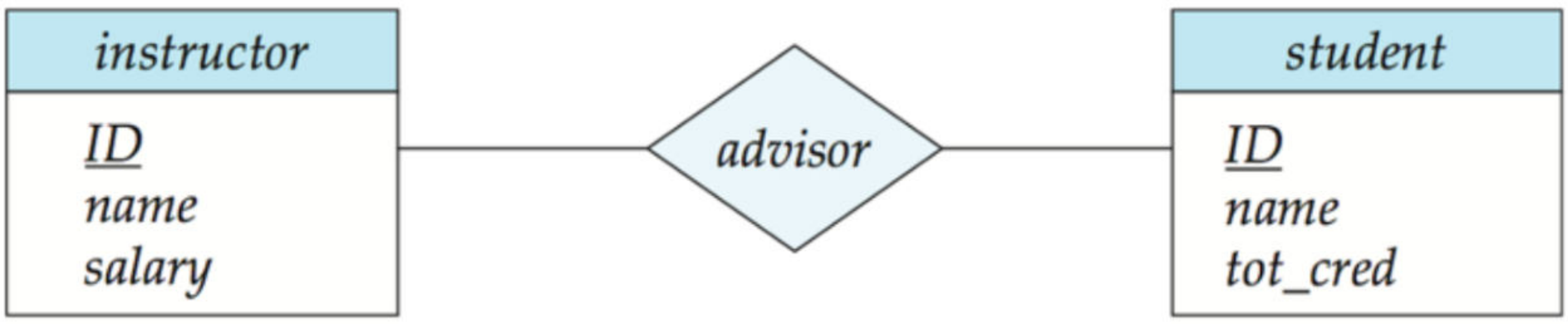
  section (course_id, sec_id, sem, year)



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 = *(<u>s_id, i_id</u>)*



Representation of entity sets with composite attributes

- Composite attributes are flattened out by creating a separate attribute for each component attribute
  - **Example:** Given entity set **instructor** with composite attribute **<u>name</u>** with component attributes *<u>first_name</u>* and *<u>last_name</u>* the schema corresponding to the entity set has two attributes **name_first_name** and **name_last_name**
    - Prefix omitted if there is no ambiguity (**name_first_name** could simply be **first_name**)
- Ignoring multi-valued 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)
```

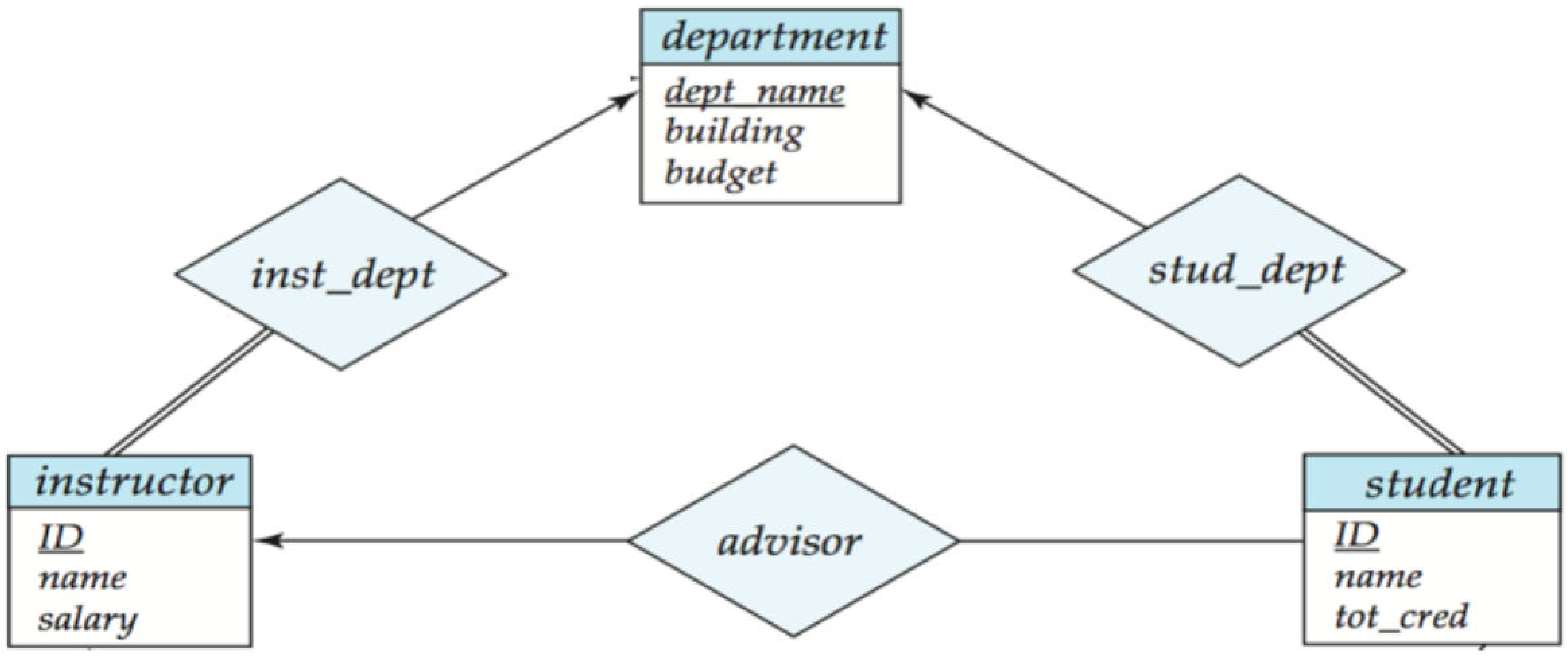
Representation of Entity sets with multi-valued attributes

- A multi-valued 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 multi-valued attribute M
- **Example:** Multi-valued attribute phone_number of instructor is represented by a schema:

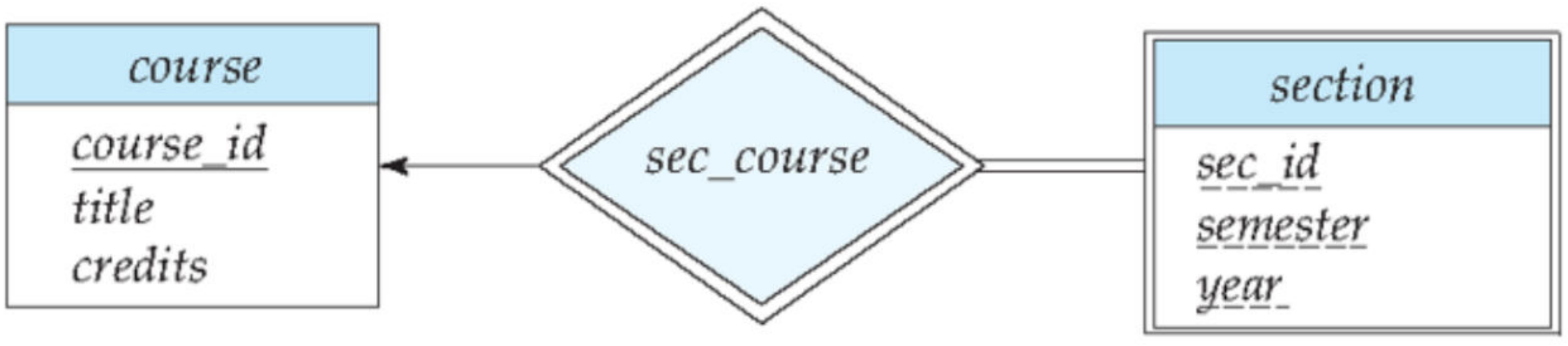
  inst_phone = (*<u>ID</u>*, *<u>phone_number</u>*)
- Each value of the multi-valued attribute maps to a separate tuple of the relation on schema EM
  - **For example:** an instructor entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples: (22222, 456-7890) and (22222, 123-4567)

Redundancy of the Schema

- Many-to-One and One-to-Many relationship sets that are total on the many-side can be represented by adding an extra attribute to the "many" side, containing the primary key of the "one" side
- **Example:** Instead of creating a schema for relationship set *inst_dept*, add an attribute *dept_name* to the schema arising from entity set *instructor*



- For One-to-One relationship sets, either side can be chosen to act as the "many" side
  - That is, an extra attribute can be added to either of the tables corresponding to the two entity sets
- If participation is partial on the "many" side, replacing a schema by an extra attribute in the schema corresponding to the "many" side could result in null values
- The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant
- **Example:** The section schema already contains the attributes that would appear in the *sec_course* schema



Week 4 Lecture 5

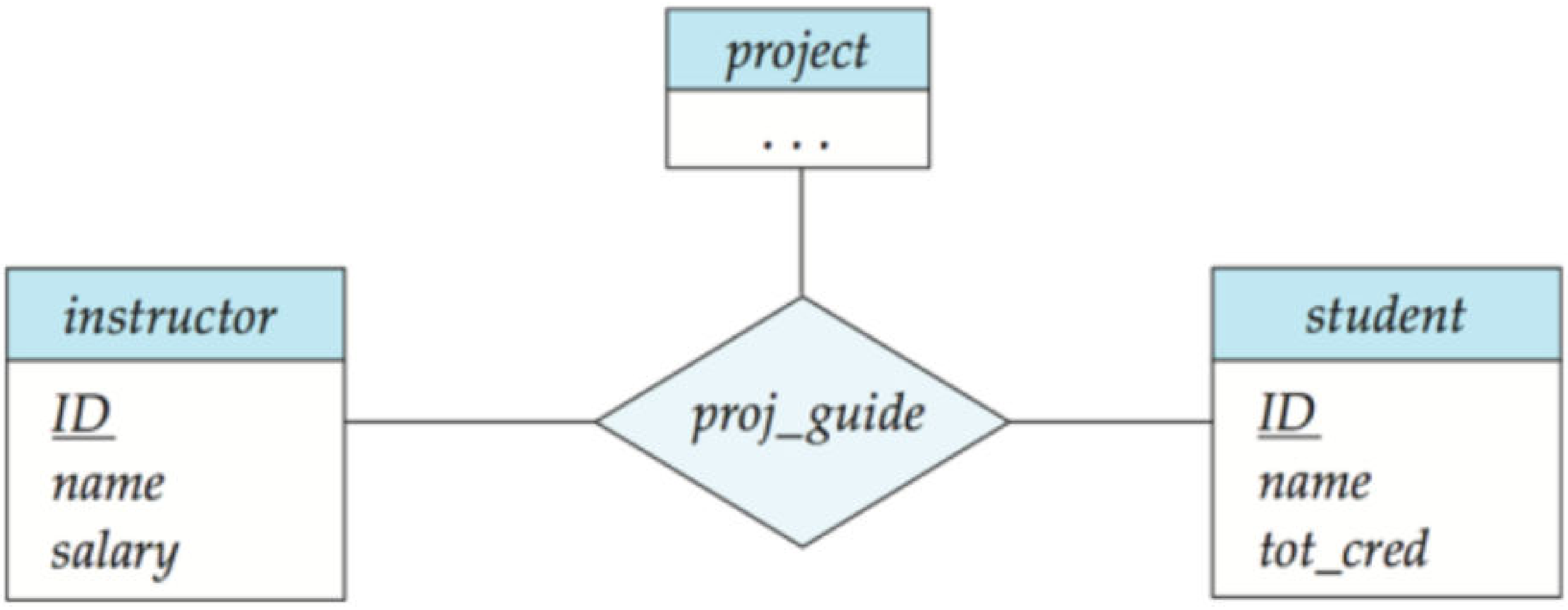
Class	BSCCS2001
Created	@September 30, 2021 8:37 PM
Materials	
Module #	20
Type	Lecture
Week #	4

Entity-Relationship Model (part 3)

Extended ER features

Non-binary Relationship sets

- Most relationship sets are binary
- There are occasions when it is more convenient to represent relationships as non-binary
- ER diagram with a Ternary Relationship



Cardinality constraints on Ternary Relationship

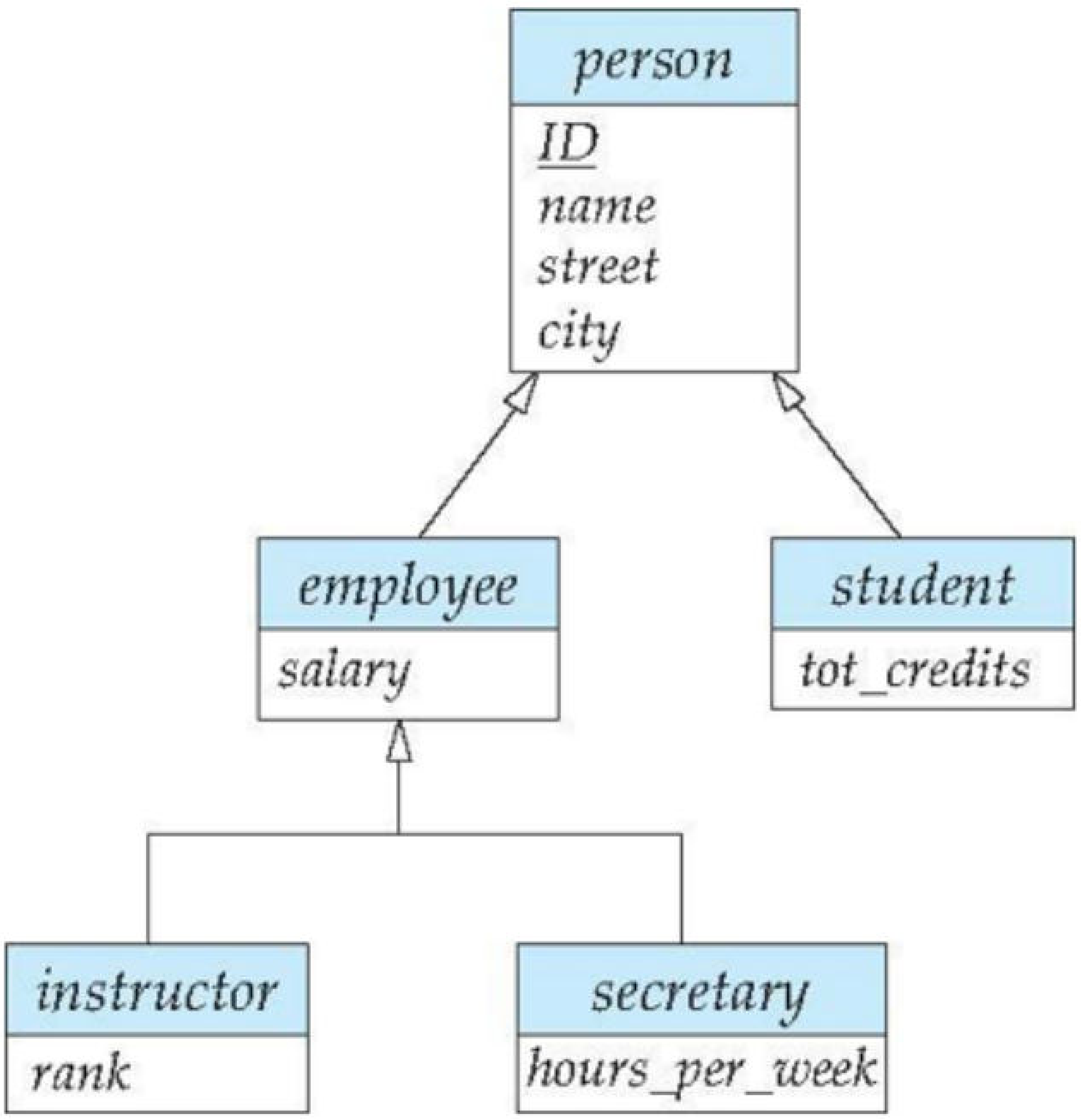
- We allow at most one arrow out of a ternary (or greater degree) relationship to indicate a cardinality constraint
- For example, an arrow from *proj_guide* to *instructor* indicates each student has at most one guide for a project
- If there is more than one arrow, there are two ways of defining the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

Specialization: ISA

- **Top-down design process:** We designate sub-groupings within an entity set that are distinctive from other entities in the set
- These sub-groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set
- Depicted by a triangle component leveled ISA (eg: instructor "is a" person)
- **Attribute inheritance:** A lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked

---

- **Overlapping:** *employee* and *student*
- **Disjoint:** *instructor* and *secretary*
- Total and Partial



Representing Specialization via Schema

- Method 1:
    - Form a schema for the higher-level entity

    - Form a schema for each lower-level entity set, include primary key of higher-level entity set and local attributes

schema	attributes
person	ID, name, street, city
student	ID, tot_cred
employee	ID, salary

    - **Drawback:** Getting information about an employee requires accessing two relations, the one corresponding to the low-level schema and the one corresponding to the high-level schema
- Method 2:
    - Form a schema for each entity set with all local and inherited attributes

Name	Tags
person	ID, name, street, city
student	ID, name, street, city, tot_cred
employee	ID, name, street, city, salary

    - **Drawback:** name, street and city may be stored redundantly for people who are both students and employees

Generalization

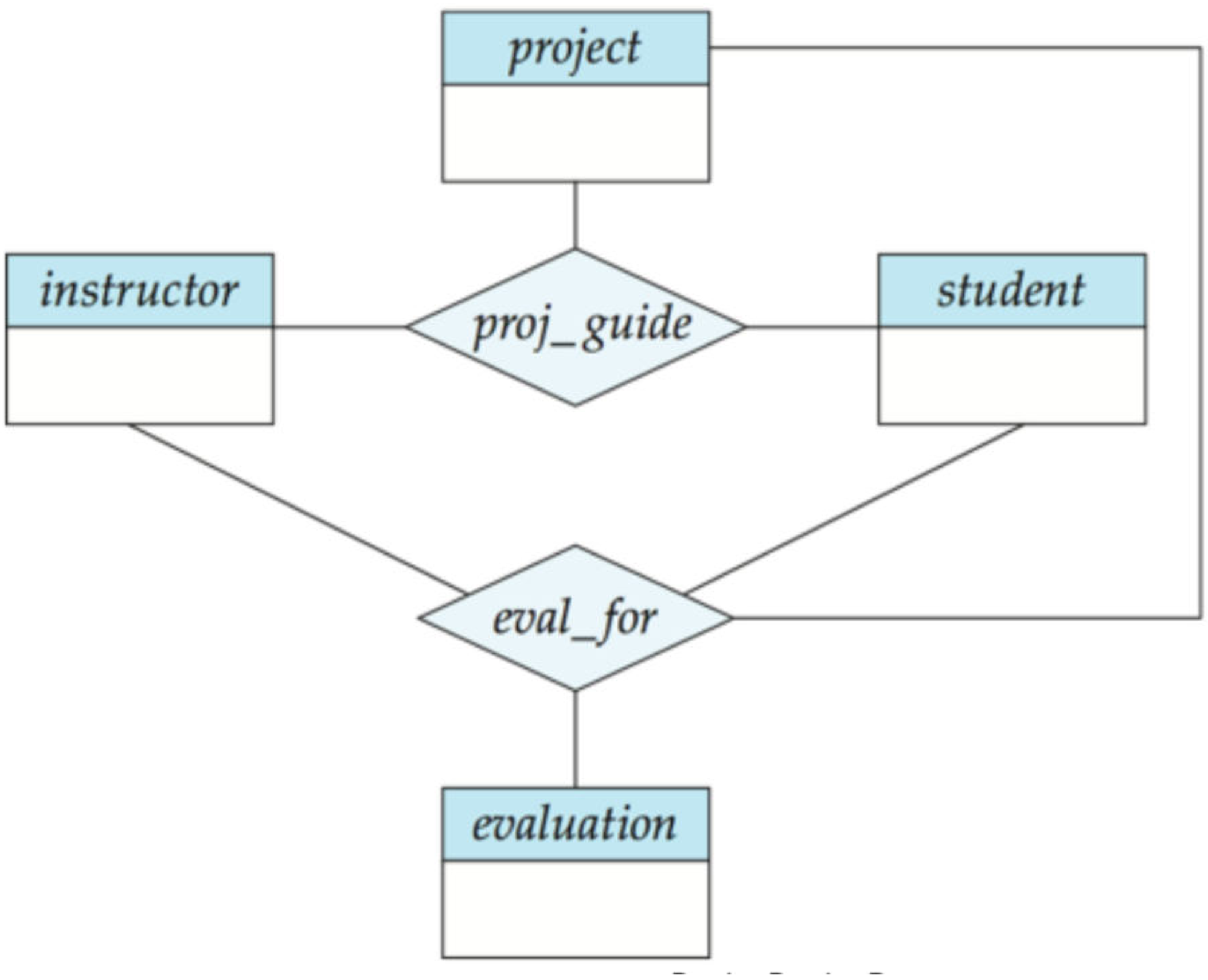
- **Bottom-up design process:** Combine a number of entity sets that share the same features into a higher-level entity set
- Specialization and generalization are simple inversions of each other; they are represented in an ER diagram in the same way
- The terms specialization and generalization are used interchangeably

Design constraints on a specialization / generalization

- **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
- Partial generalization is the default
    - We can specify total generalization in an ER diagram by adding the keyword **total** in the diagram
    - Drawing a dashed line from the keyword to the corresponding hollow arrow-head to which it applies (for a total generalization) or to the set of hollow arrow-heads to which it applies (for an overlapping generalization)
- The student generalization is total
    - All student entities must be either graduate or undergraduate
    - Because the higher-level entity set arrived at through generalization is generally composed of only those entities in the lower-level entity sets, the completeness constraint for a generalized higher-level entity set is usually total

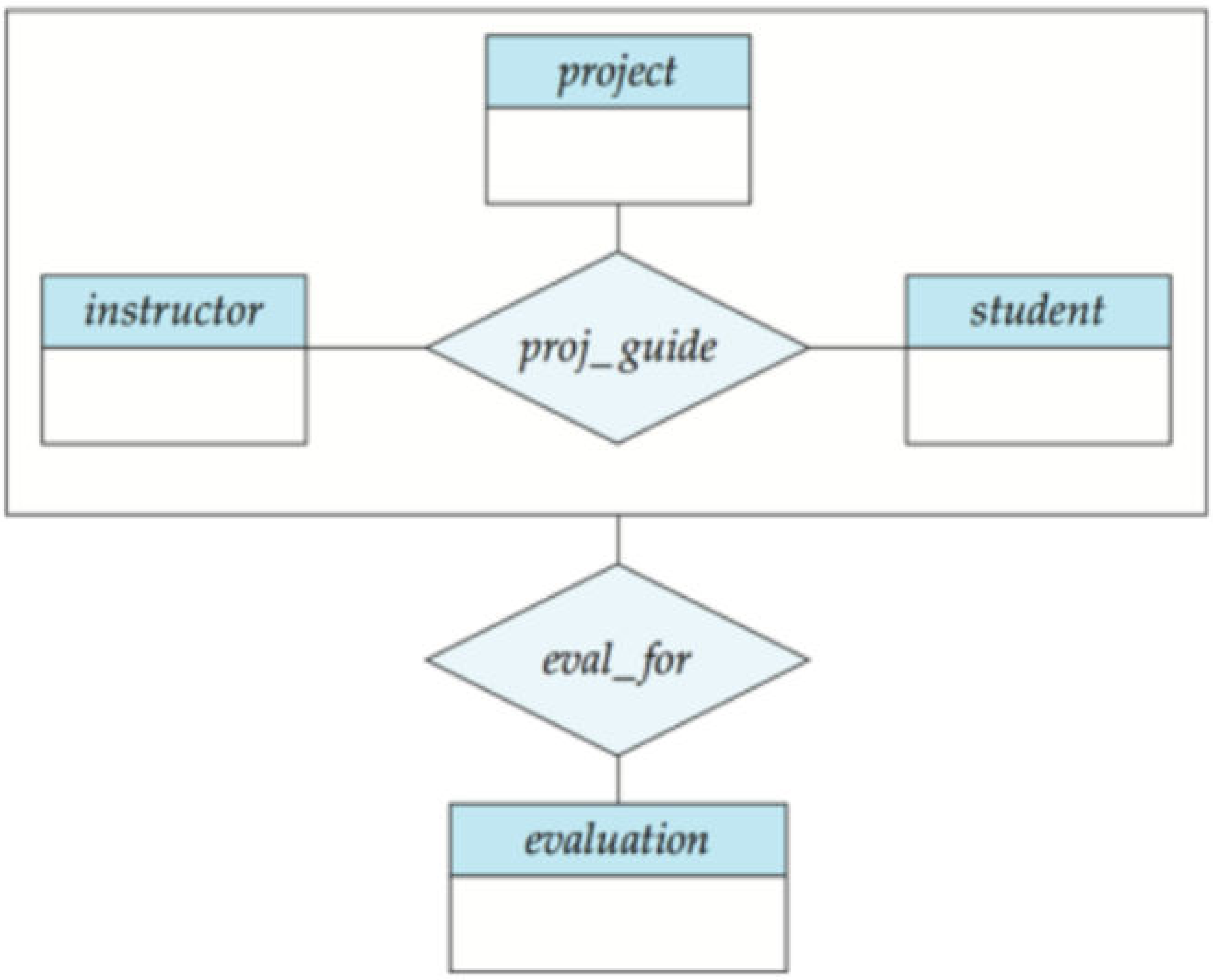
Aggregation

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



- 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 cannot 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

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



Representing aggregation via Schema

- To represent aggregation, create a schema containing
    - Primary key of the aggregated relationship
    - The primary key of the associated entity set
    - Any descriptive attributes
- In our example
    - The schema

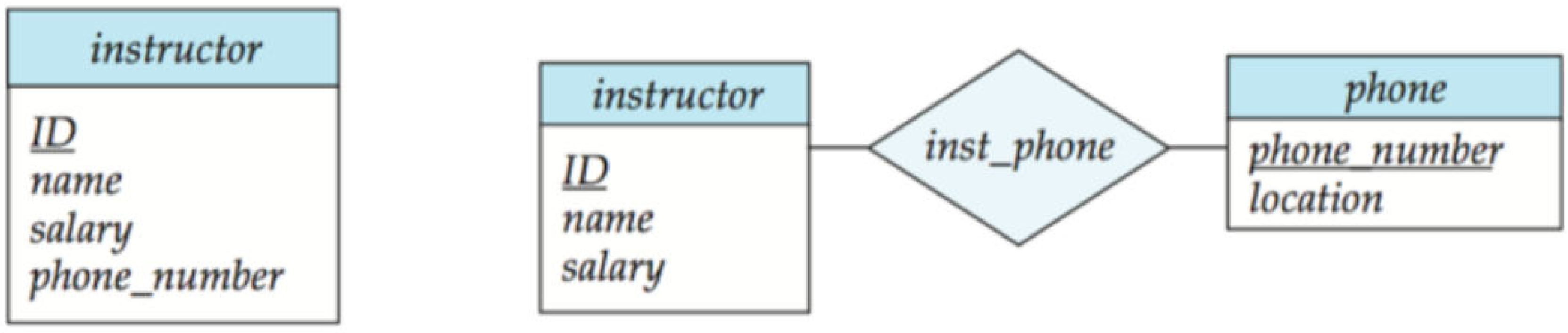
      textiteval_for is:

      eval_for (s_ID, project_id, i_ID, evaluation_id)
    - The schema *proj_guide* is redundant

Design Issues

Entities v/s Attributes

- Use of entity sets v/s attributes

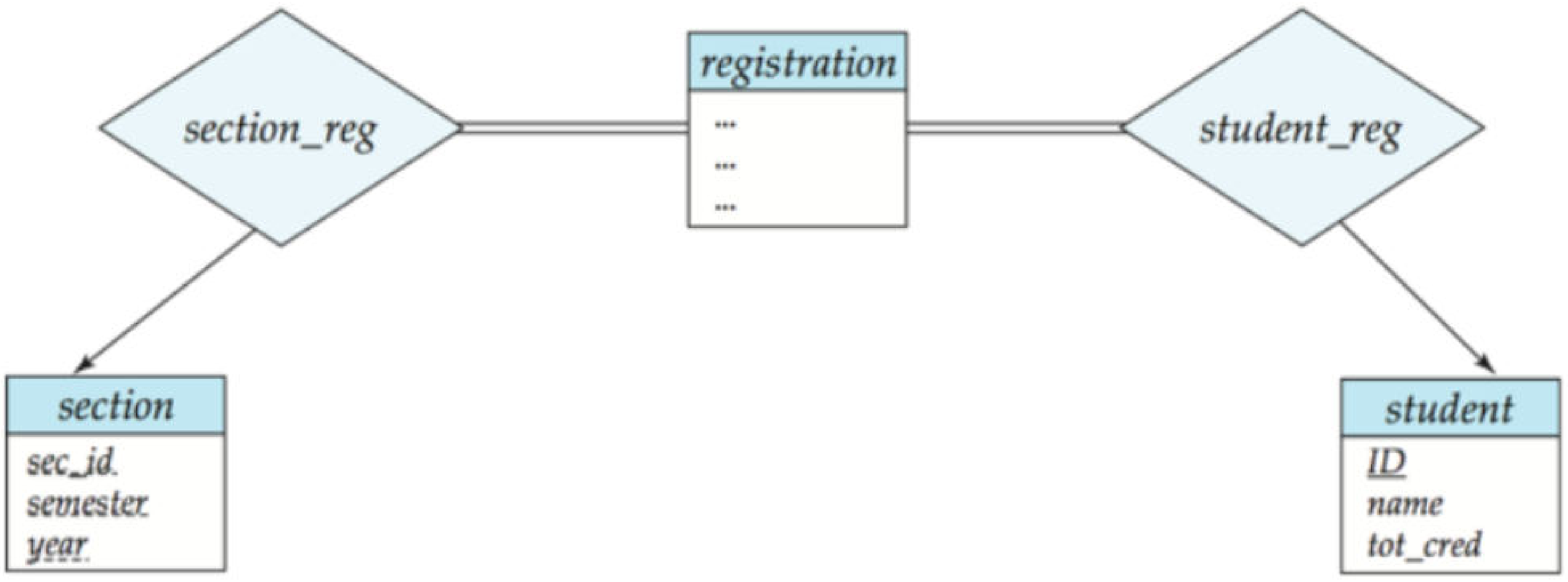


- Use of phone as an entity allows extra information about phone numbers (plus multiple phone numbers)

Entities v/s Relationship sets

- **Use of entity sets v/s relationship sets**

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



- **Placement of relationship attributes**

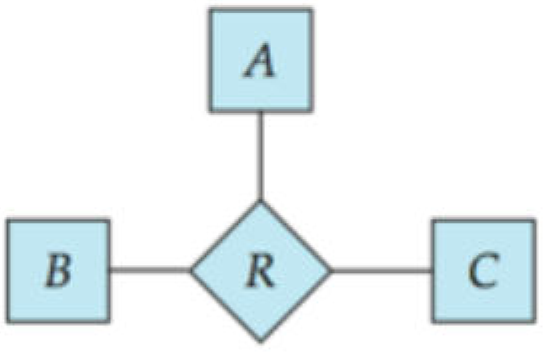
  For example, attribute date as attribute of advisor or as attribute of student

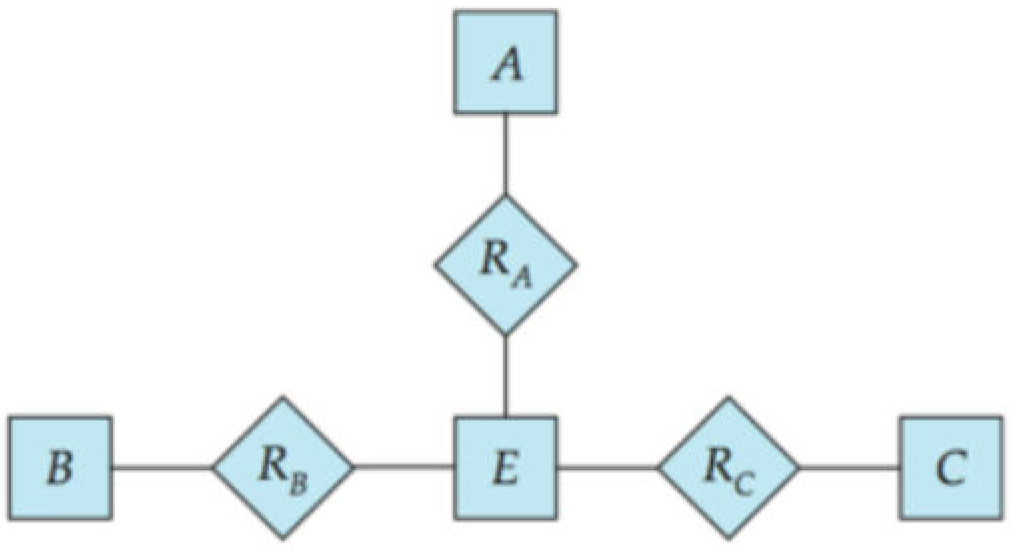
Binary v/s Non-binary Relationships

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

Binary v/s Non-binary Relationships: Conversion

- In general, any non-binary relationship can be represented using binary relationships by creating an artificial entity set
  - Replace R between entity sets A, B and C by an entity set E, and three relationship sets:
    - R_A, relating E and A
    - R_B, relating E and B
    - R_C, relating E and C
  - Create an identifying attribute for E and add any attributes of R to E
  - For each relationship (a_i, b_i, c_i) in R, create
    - A new entity e_i in the entity set E
    - add (e_i, a_i) to R_A
    - add (e_i, b_i) to R_B
    - add (e_i, c_i) to R_C



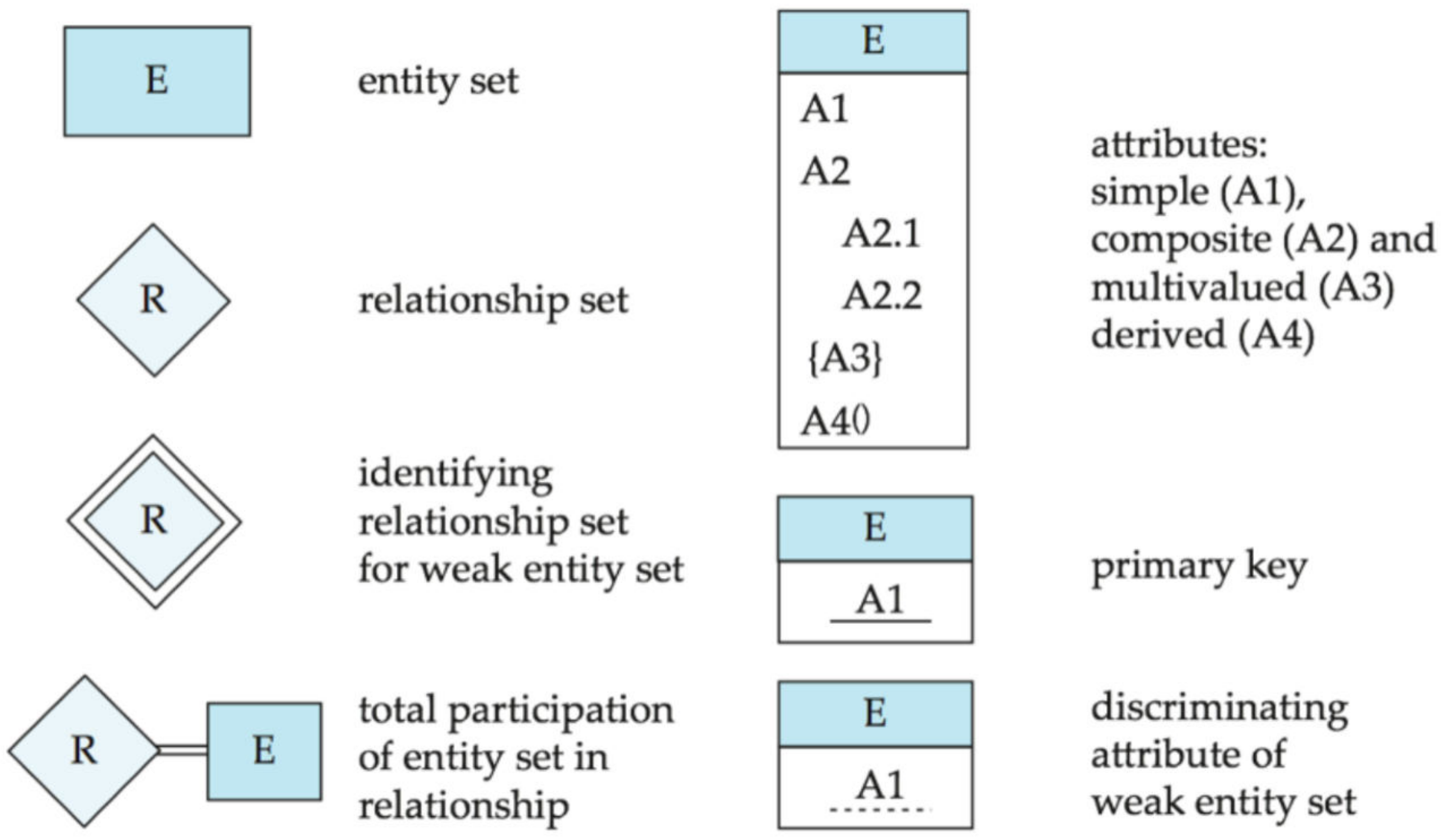


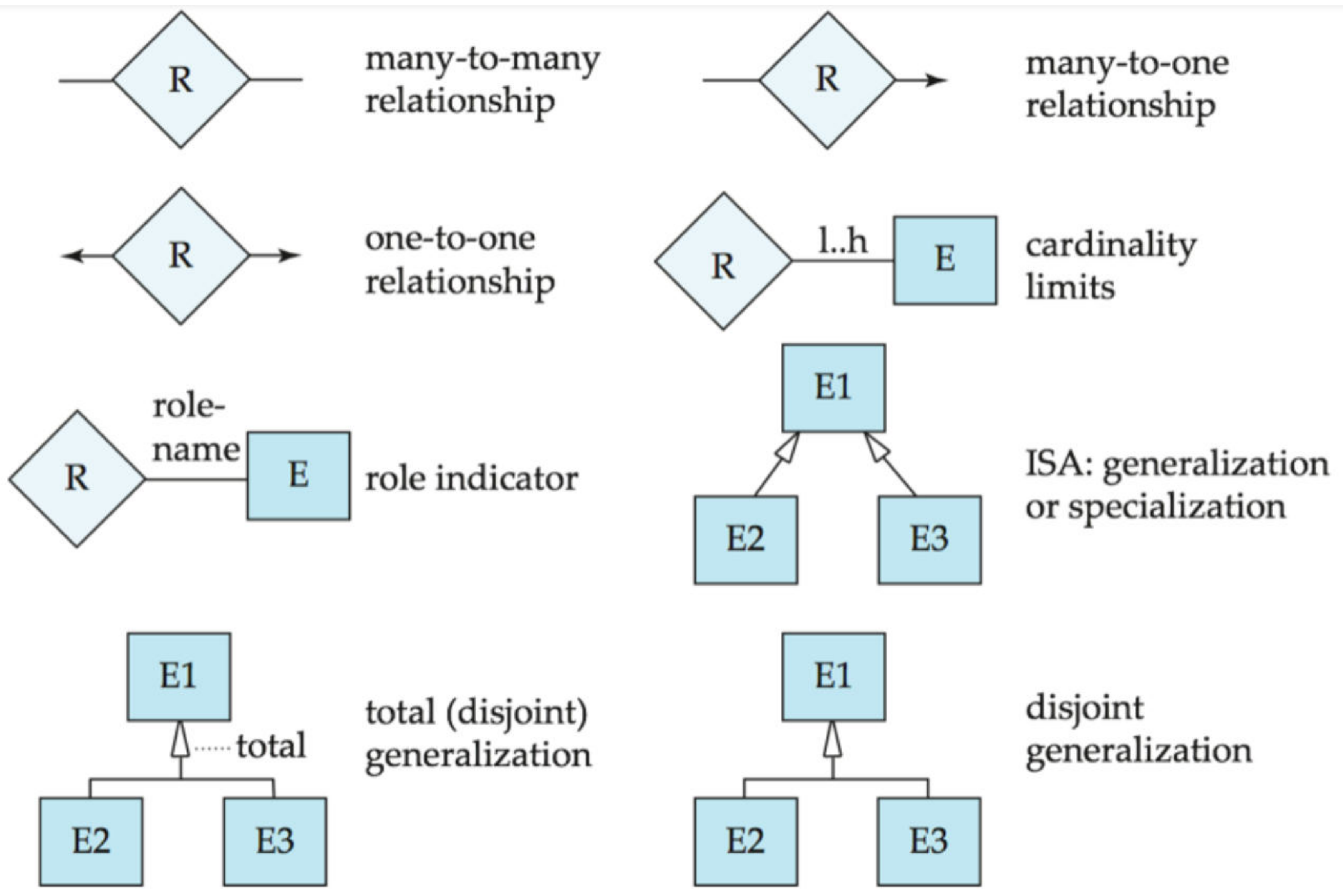
- Also need to translate constraints
    - Translating all constraints may not be possible
    - There may be instance in the translated schema that cannot correspond to any instance of R
        - **Exercise:** add constraints to the relationships $R_A, R_B\ and\ R_C$ to ensure that a newly created entity corresponds to exactly one entity in each of entity sets — A, B and C
    - We can avoid creating an identifying attribute by making E, a weak entity set identified by the three relationship sets

ER Design Decisions

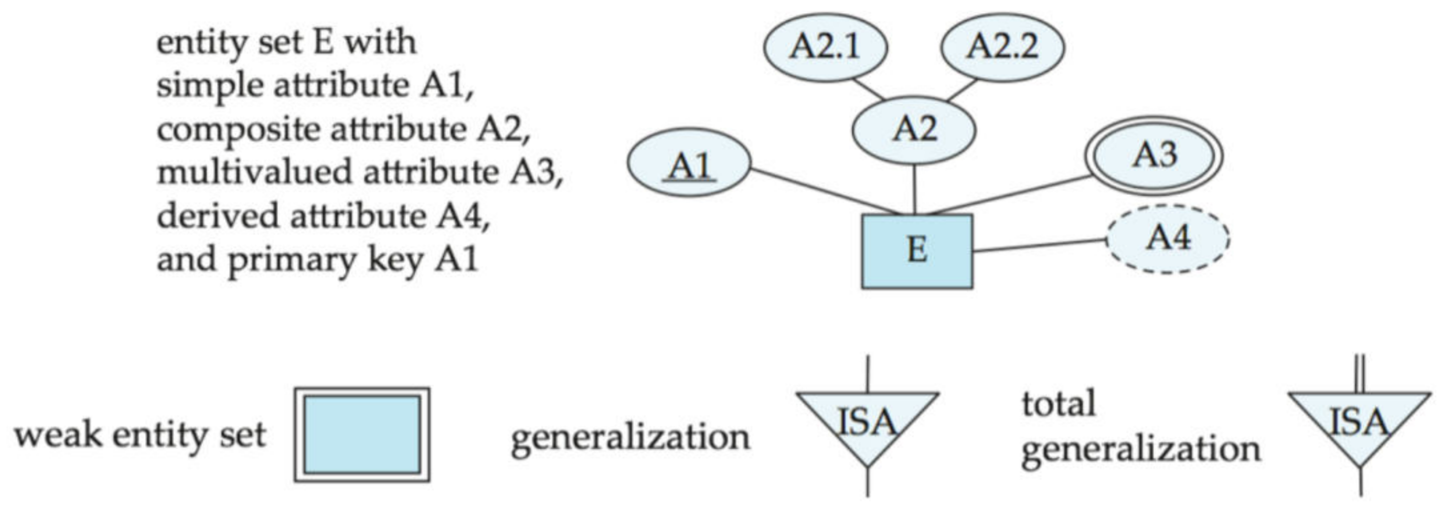
- The use of an attribute or entity set to represent an object
- Whether a real-world concept is best expressed by an entity or a relationship set
- The use of a ternary relationship versus a pair of binary relationships
- The use of strong or weak entity set
- The use of specialization/generalization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

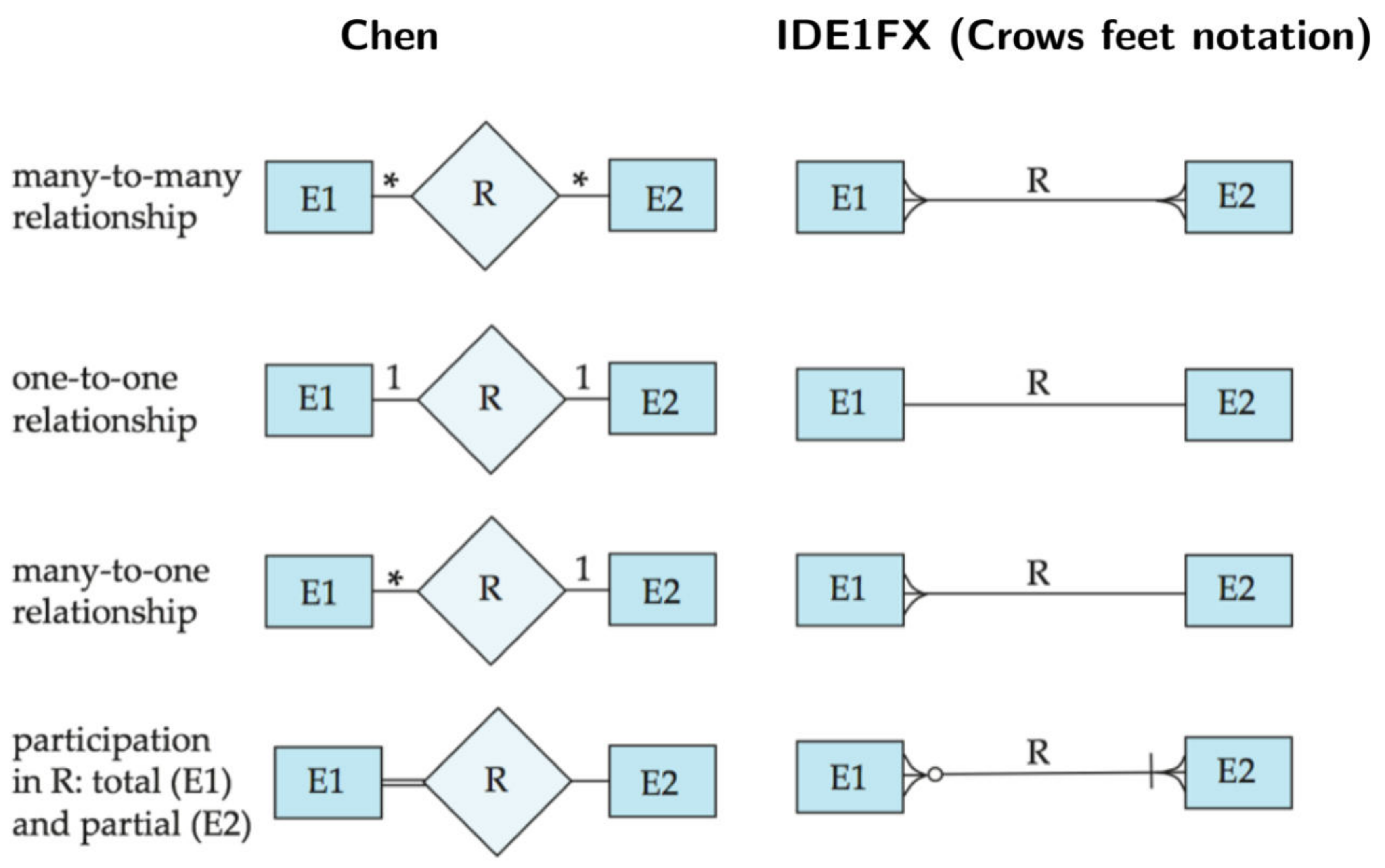
Symbols used in the ER Notation


E
entity set
R
relationship set
R
identifying relationship set for weak entity set
R
E
total participation of entity set in relationship
E
A1
A2
A2.1
A2.2
{A3}
A4()
attributes: simple (A1), composite (A2) and multivalued (A3) derived (A4)
E
A1
primary key
E
A1
discriminating attribute of weak entity set


R
many-to-many relationship
R
many-to-one relationship
R
one-to-one relationship
R
1..h
E
cardinality limits
R
role-name
E
role indicator
E1
E2
E3
ISA: generalization or specialization
E1
total
E2
E3
total (disjoint) generalization
E1
E2
E3
disjoint generalization

- Chen, IDE1FX,. . .


entity set E with simple attribute A1, composite attribute A2, multivalued attribute A3, derived attribute A4, and primary key A1
A2.1
A2.2
A2
A1
A3
E
A4
weak entity set
generalization
ISA
total generalization
ISA


Chen
IDE1FX (Crows feet notation)
many-to-many relationship
one-to-one relationship
many-to-one relationship
participation in R: total (E1) and partial (E2)
E1
E2
R
*
1