Other Relational Languages

Tuple Relational Calculus

- A nonprocedural query language, where each query is of the form

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

- It is the set of all tuples *t* such that predicate *P* is true for *t*
- *t* is a *tuple variable*, *t* [*A*] denotes the value of tuple *t* on attribute *A*
- $t \in r$ denotes that tuple *t* is in relation *r*
- *P* is a *formula* similar to that of the predicate calculus

Predicate Calculus Formula

1. Set of attributes and constants
2. Set of comparison operators: (e.g., $<$, $\leq$, $=$, $\neq$, $>$, $\geq$)
3. Set of connectives: and ($\wedge$), or (v), not ($\neg$)
4. Implication ($\Rightarrow$): x $\Rightarrow$ y, if x if true, then y is true

$$x \Rightarrow y \equiv \neg x \vee y$$

5. Set of quantifiers:
   - Existential quantifier:
     - $\exists\, t \in r\, (Q\, (t\,)) \equiv$ "there exists" a tuple in t in relation r such that predicate $Q\,(t\,)$ is true
   - Universal quantifier:
     - $\forall t \in r\, (Q\, (t\,)) \equiv Q$ is true "for all" tuples t in relation r

Example Queries

- University Example:
  - instructor(ID, name, dept_name, salary)
  - section(course_id, semester, year, sec_id)
- Find the *ID, name, dept_name, salary* for instructors whose salary is greater than $80,000

$$\{t \mid t \in instructor \land t\,[salary\,] > 80000\}$$

- As in the previous query, but output only the *ID* attribute value

$$\{t \mid \exists\, s \in \text{instructor}\ (t\,[ID\,] = s\,[ID\,] \land s\,[salary\,] > 80000)\}$$

Notice that a relation on schema (*ID*) is *implicitly* defined by the query

Example Queries

- Find the names of all instructors whose department is in the Watson building

$$\{t \mid \exists s \in instructor\ (t\,[name\,] = s\,[name\,] \wedge \exists u \in department\ (u\,[dept\_name\,] = s[dept\_name] \wedge u\,[building] = \text{"Watson"}\,))\}$$

- Find the set of all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both

$$\{t \mid \exists s \in section\ (t\,[course\_id\,] = s\,[course\_id\,] \wedge s\,[semester] = \text{"Fall"} \wedge s\,[\text{year}] = 2009 \vee \exists u \in section\ (t\,[course\_id\,] = u\,[course\_id\,] \wedge u\,[semester] = \text{"Spring"} \wedge u\,[\text{year}] = 2010)\}$$

Example Queries

- Find the set of all courses taught in the Fall 2009 semester, and in the Spring 2010 semester

$$\{t \mid \exists s \in section\ (t\,[course\_id\,] = s\,[course\_id\,] \land s\,[semester] = \text{“Fall”} \land s\,[\text{year}] = 2009$$
$$\land\ \exists u \in section\ (t\ \ [course\_id\,] = u\,[course\_id\,] \land u\,[semester] = \text{“Spring”} \land u\,[\text{year}] = 2010)\}$$

- Find the set of all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\{t \mid \exists s \in section\ (t\,[course\_id\,] = s\,[course\_id\,] \land s\,[semester] = \text{“Fall”} \land s\,[\text{year}] = 2009$$
$$\land \neg\ \exists u \in section\ (t\ \ [course\_id\,] = u\,[course\_id\,] \land u\,[semester] = \text{“Spring”} \land u\,[\text{year}] = 2010)\}$$

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 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 **dom**(*P*) : the relations, tuples, or constants that appear in *P*
  - NOTE: this is more than just a syntax condition.
    - E.g. $\{ t \mid t[A] = 5 \vee \textbf{true} \}$ is not safe --- it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in *P*.

Universal Quantification

- Find all students who have taken all courses offered in the Biology department, and the courses they took:
  - $\{t \mid \forall u \in course\ (\boxed{u\,[dept\_name]=\text{“Biology”} \Rightarrow}$
    $\exists s \in takes\ (t\,[ID] = s\,[ID] \wedge$
    $s\,[course\_id] = u\,[course\_id] \wedge$
    $t[course\_id] = s[course\_id]))\}$
  - Note the schema on *r* is (*ID, course_id*) here.
  - What's the problem with this query?
  - The above query would be unsafe if the Biology department has not offered any courses.

Universal Quantification

- Find all students who have taken all courses offered in the Biology department, and the courses they took:
  - $\{t \mid \boxed{\exists\, r \in student\, (t\,[ID] = r\,[ID]) \wedge}$
    $(\forall\, u \in course\, (u\,[dept\_name] = \text{"Biology"} \Rightarrow$
    $\exists\, s \in takes\, (t\,[ID] = s\,[ID\,] \wedge$
    $s\,[course\_id] = u\,[course\_id] \wedge$
    $t[course\_id] = s[course\_id])))\}$
  - Add the existential quantification on student.

Domain Relational Calculus

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

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

  - $x_1, x_2, \ldots, x_n$ represent domain variables
  - Gives the schema of the output relation explicitly
  - P represents a formula similar to that of the predicate calculus

Example Queries

- Find the *ID, name, dept_name, salary* for instructors whose salary is greater than $80,000
  - $\{<i, n, d, s> \mid <i, n, d, s> \in instructor \wedge s > 80000\}$
- As in the previous query, but output only the *ID* attribute value
  - $\{<i> \mid \exists\, n, d, s\ (<i, n, d, s> \in instructor \wedge s > 80000)\}$
- Find the names of all instructors whose department is in the Watson building

$$\{<n> \mid \exists\, i, d, s\ (<i, n, d, s> \in instructor \wedge \exists\, b, a\ (<d, b, a> \in department \wedge b = \text{“Watson”}))\}$$

Example Queries

- Find the set of all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both

$$\{<c> \mid \exists\, a, s, y, b, t\ (<c, a, s, y, b, t> \in section\ \wedge\ s = \text{“Fall”} \wedge y = 2009)\ \vee \exists\, a, s, y, b, t\ (<c, a, s, y, b, t> \in section\ \wedge\ s = \text{“Spring”} \wedge y = 2010)\}$$

This case can also be written as

$$\{<c> \mid \exists\, a, s, y, b, t\ (<c, a, s, y, b, t> \in section\ \wedge\ ((s = \text{“Fall”} \wedge y = 2009)\ \vee (s = \text{“Spring”} \wedge y = 2010))\}$$

- Find the set of all courses taught in the Fall 2009 semester, and in the Spring 2010 semester

$$\{<c> \mid \exists\, a, s, y, b, t\ (<c, a, s, y, b, t> \in section\ \wedge\ s = \text{“Fall”} \wedge y = 2009)\ \wedge \exists\, a, s, y, b, t\ (<c, a, s, y, b, t> \in section \wedge s = \text{“Spring”} \wedge y = 2010)\}$$

Safety of Expressions

The expression:

$$\{ < x_1, x_2, \ldots, x_n > \mid P\,(x_1, x_2, \ldots, x_n\,)\}$$

is safe if all of the following hold:

1. All values that appear in tuples of the expression are values from *dom* (P) (that is, the values appear either in P or in a tuple of a relation mentioned in P).
2. For every "there exists" subformula of the form $\exists\, x\,(P_1(x\,))$, the subformula is true if and only if there is a value of x in *dom* (P_1) such that $P_1(x\,)$ is true.
3. For every "for all" subformula of the form $\forall x\,(P_1\,(x\,))$, the subformula is true if and only if $P_1(x\,)$ is true for all values x from *dom* (P_1).

Universal Quantification

- Find all students who have taken all courses offered in the Biology department
  - $\{< i > \mid \exists\, n, d, tc\ (< i, n, d, tc > \in student\ \wedge$
    $(\forall\, ci, ti, dn, cr\ (< ci, ti, dn, cr > \in course \wedge dn = \text{"Biology"}$
    $\Rightarrow\ \exists\, si, se, y, g\ (<i, ci, si, se, y, g> \in takes\ ))\}$
  - Note that without the existential quantification on student, the above query would be unsafe if the Biology department has not offered any courses.

Datalog

- Non-procedural query language based on Prolog.
- A Datalog program consists of a set of ***rules***, each defines a *view.*
  - v1(A, B) :- account (A, "Perryridge", B), B > 700.
  - Commas "," read as "AND".

Head

Body

- To retrieve the balance of account A-314:
  - ?- v1("A-314", B).
  - Answer: ("A-314", 780).
- To get the account number and balance of all accounts in v1 with balance more than 1000:
  - ?- v1(A, B), B > 1000.
  - Answer: ("A-205", 1200).

Syntax of Datalog Rules

- Uppercase letters or words starting with uppercase letters as variables
- Lowercase letters and words starting with lowercase letter as relation names and attribute names.
- Positive literal: $p(t_1, t_2, \ldots, t_n)$
  - $t_1, t_2, \ldots, t_n$ are either constants or variables.
  - p is the predicate symbol.
- Negative literal: **not** $p(t_1, t_2, \ldots, t_n)$
- B > 700 can be understand as a literal, too: > (B, 700)
- $p(v_1, v_2, \ldots, v_n)$ is a fact, where $v_1, .., v_n$ are constants.
  - Tuple $(v_1, v_2, \ldots, v_n)$ is in relation p.
- A rule is expressed as:

  $$p(t_1, t_2, \ldots, t_n) \text{ :- } L_1, L_2, .., L_m$$

  ($L_1, L_2, .., L_m$ are literals.)

Semantic of Rules

- The set of facts that can be inferred from a given set of facts *I*, given a rule *R*:
  - *infer(R, I)* = {*p(t1, …, tn)* | there is an instantiation *R'* of *R*,

    where *p(t1, .., tn)* is the head of *R'*,

    and the body of *R'* is satisfied by *I*.}

- Given a set of rules $\mathcal{R}$ = {R1, R2, …, Rn},
  - infer ($\mathcal{R}$, I) = infer (R1, I) U infer (R2, I) U … U infer (Rn, I)
  - This set of rules is basically the Datalog program

- Non-recursive Datalog without arithmetics have equivalent expressive power to basic relational algebra
  - Try this: Find the set of all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester?

Recursion in Datalog

- Deals with recursive data structure, e.g. lists and trees.
- Report_Schema = (employee_name, manager_name)
- Supposed we want to find all employees managed by a person X.

  managed_by (Y, X) :- report (Y, X).

  managed_by (Y, X) :- report(Y, Z), managed_by (Z, X).

- Recursive Datalog contains no negative literals.
- Recursive rules are evaluated by *iterative*ly computing the ***fix point*** or the condition under which no new facts can be inferred → termination.

E-R Model (I)

Modeling

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

Entity Sets *instructor* and *student*

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

instructor

student-ID	student_name
98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

Relationship Sets

- A **relationship** is an association among several (typically two) entities

  Example:

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

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

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

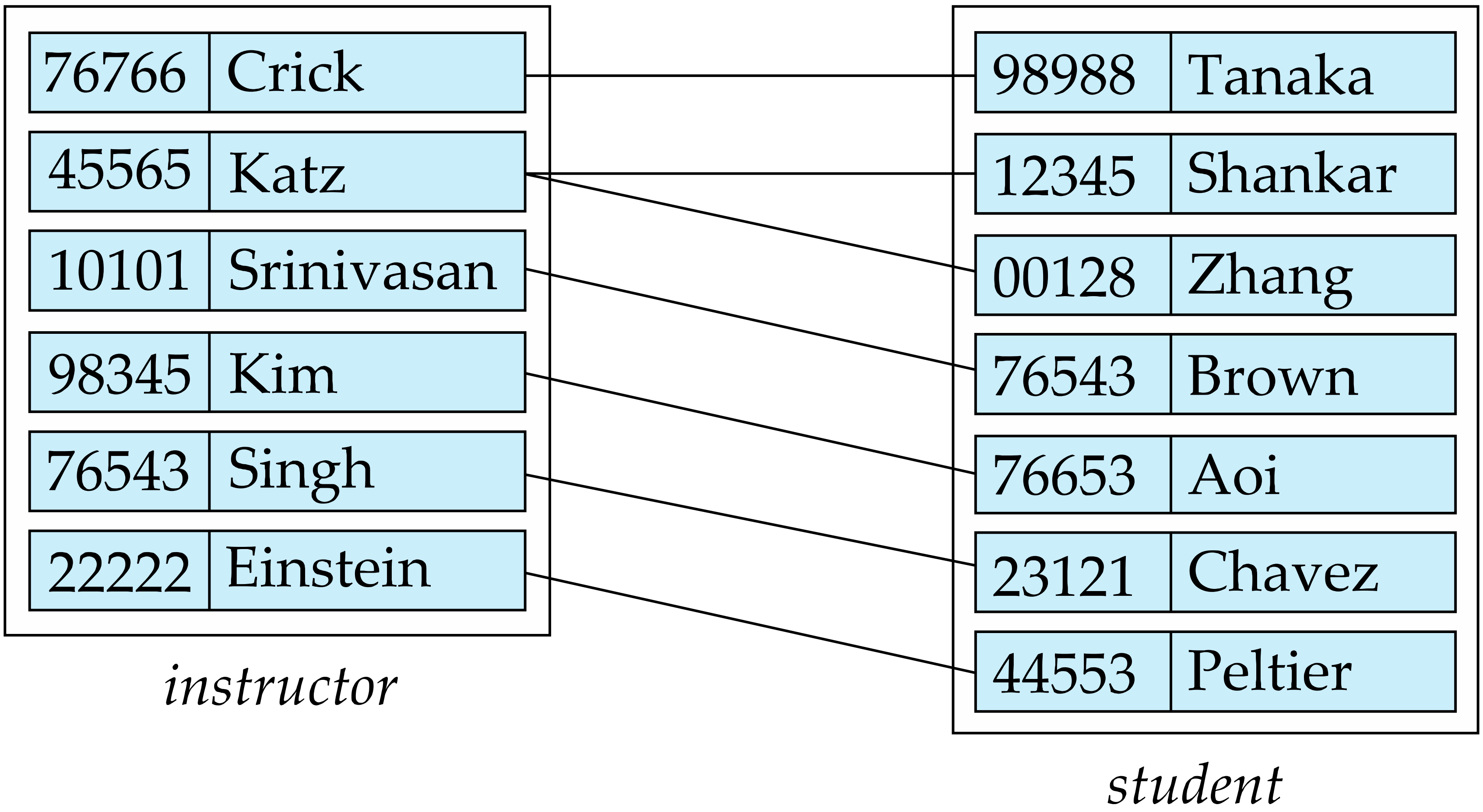
  where $(e_1, e_2, \ldots, e_n)$ is a relationship

  - Example:

    $(44553,22222) \in$ *advisor*

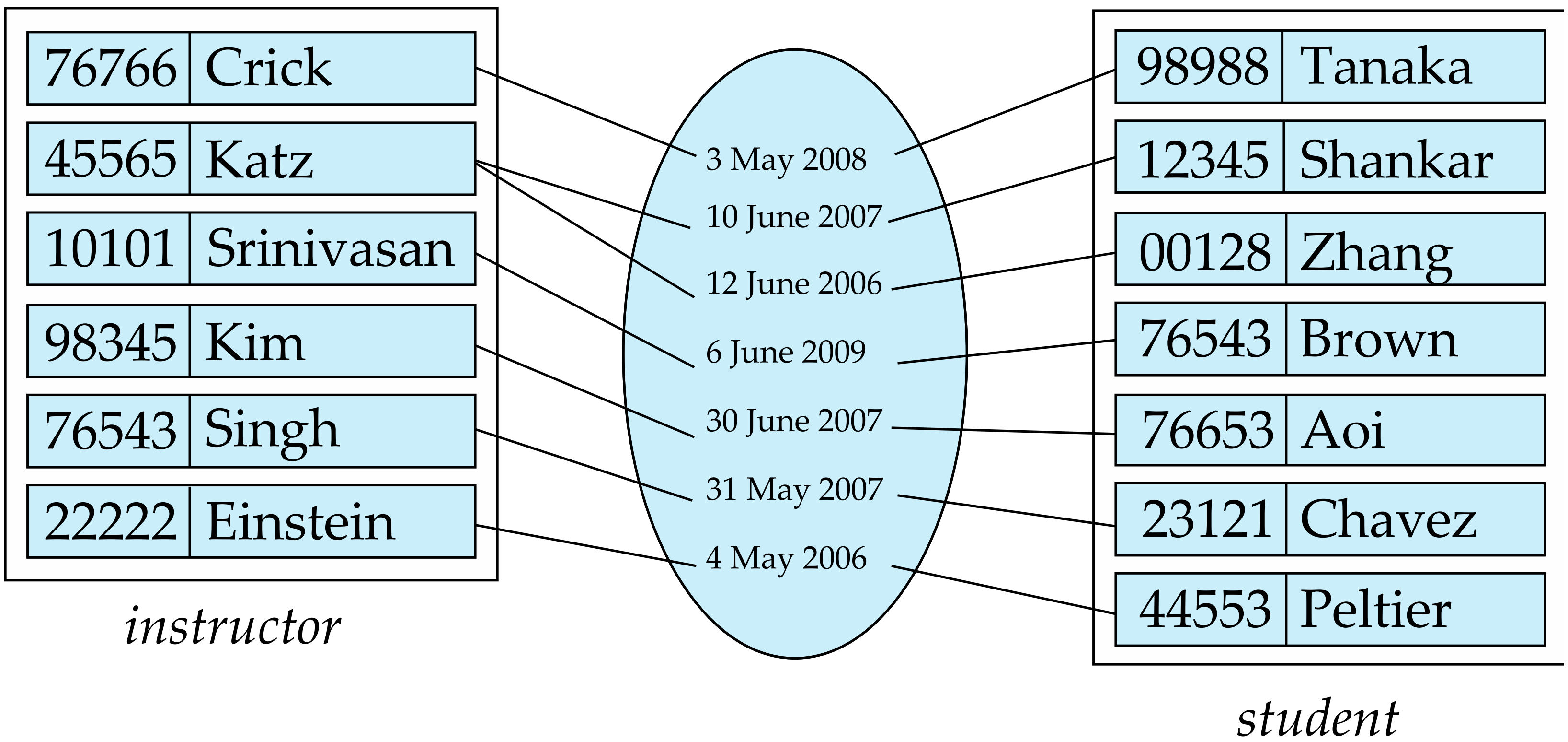
- Bascially, a *relationship* is a tuple, while a *relationship set* is a set of tuples

Relationship Set *advisor*



Relationship Sets (Cont.)

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



Degree of a Relationship Set

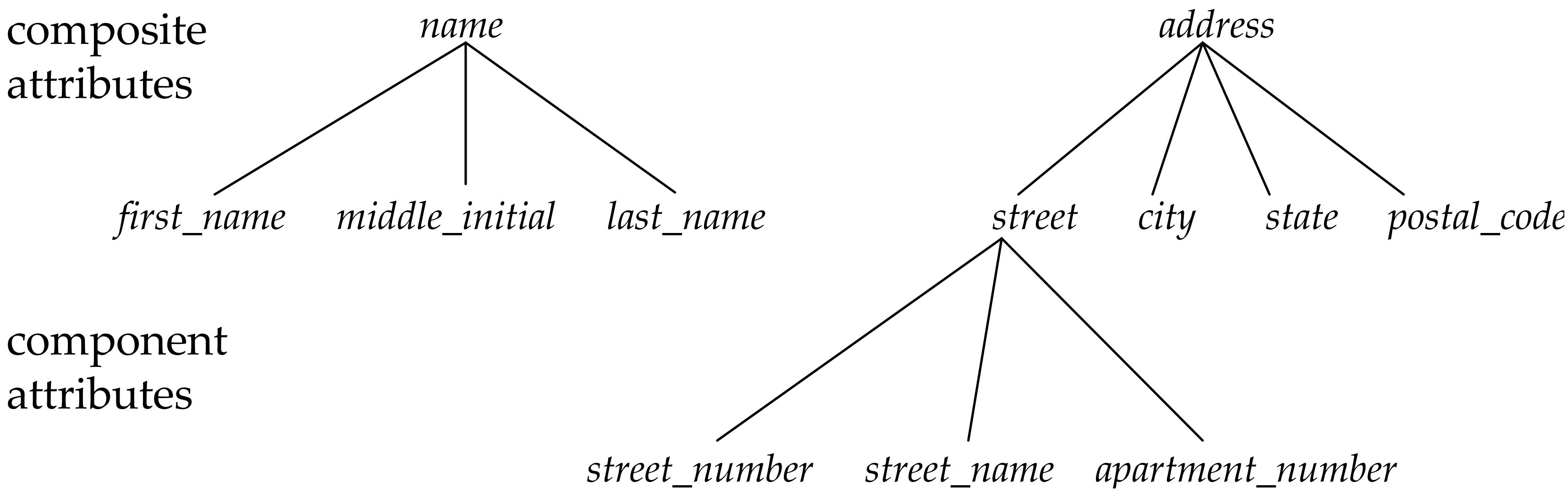
- **binary relationship**
  - involve two entity sets (or degree two).
  - most relationship sets in a database system are binary.
- Relationships between more than two entity sets are rare. Most relationships are binary. (More on this later.)
  - Example: *students* work on research *projects* under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

Attributes

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

    instructor = (*ID, name, street, city, salary*)
    course= (*course_id, title, credits*)
- **Domain** – the set of permitted values for each attribute
- Attribute types:
  - **Simple** and **composite** attributes.
  - **Single-valued** and **multivalued** attributes
    - Example: multivalued attribute: *phone_numbers*
  - **Derived** attributes
    - Can be computed from other attributes
    - Example: age, given date_of_birth

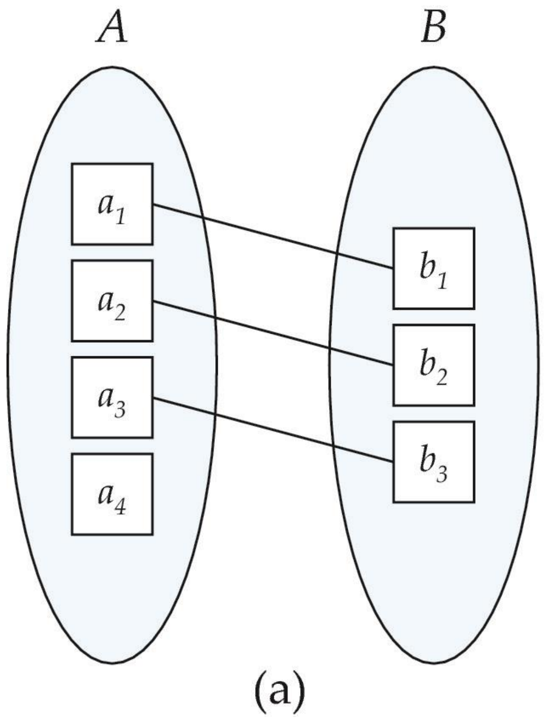
Composite Attributes



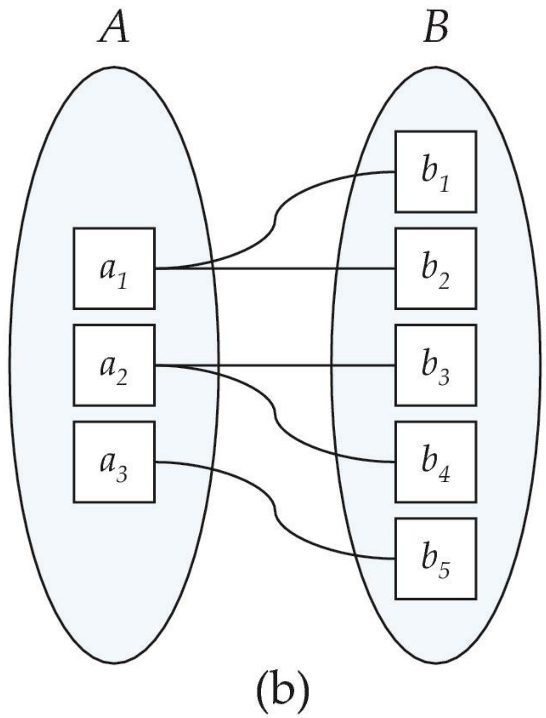
Mapping Cardinality Constraints

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

Mapping Cardinalities



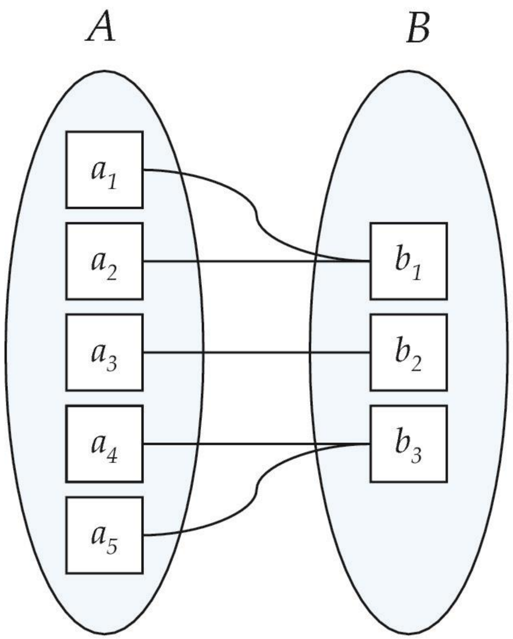
One to one



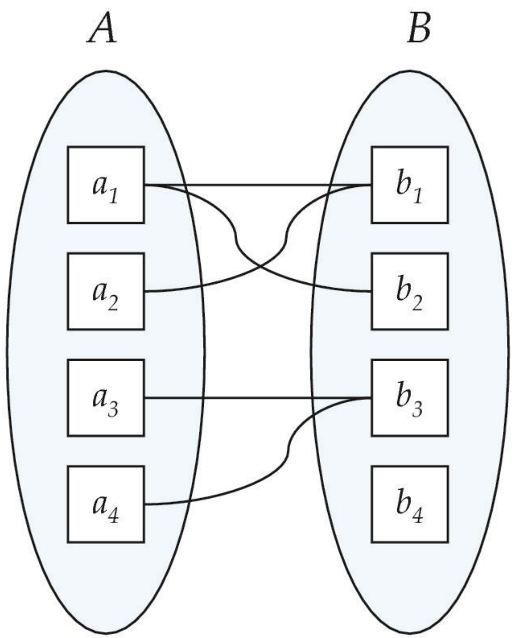
One to many

Note: Some elements in *A* and *B* may not be mapped to any elements in the other set (partial mapping)

Mapping Cardinalities



(a)



(b)

Many to one

Many to many

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

Keys

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

Keys for Relationship Sets

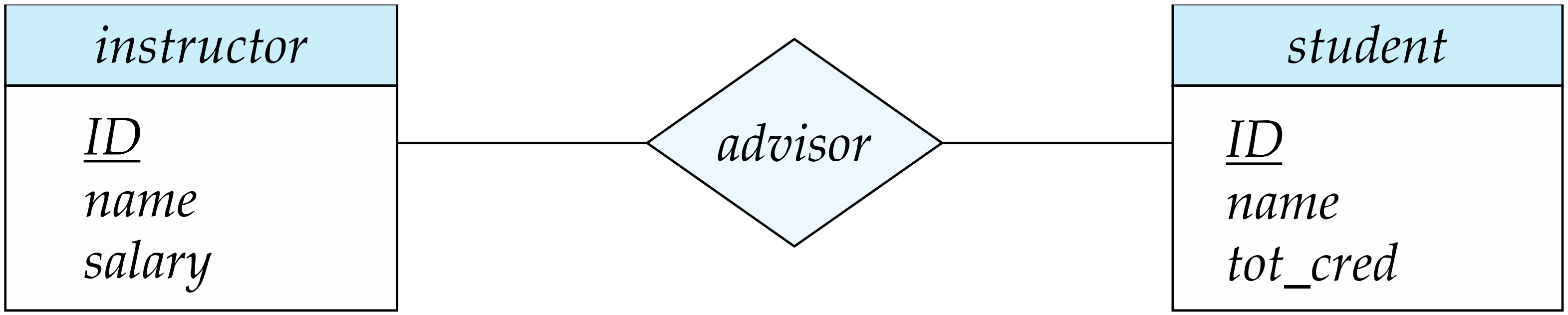
- The combination of primary keys of the participating entity sets forms a super key of a relationship set.
  - (*s_id, i_id*) is the super key of *advisor*
  - *NOTE: this means* ***a pair of entities can have at most one relationship in a particular relationship set.***
    - Example: if we wish to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

Redundant Attributes

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

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

E-R Diagrams



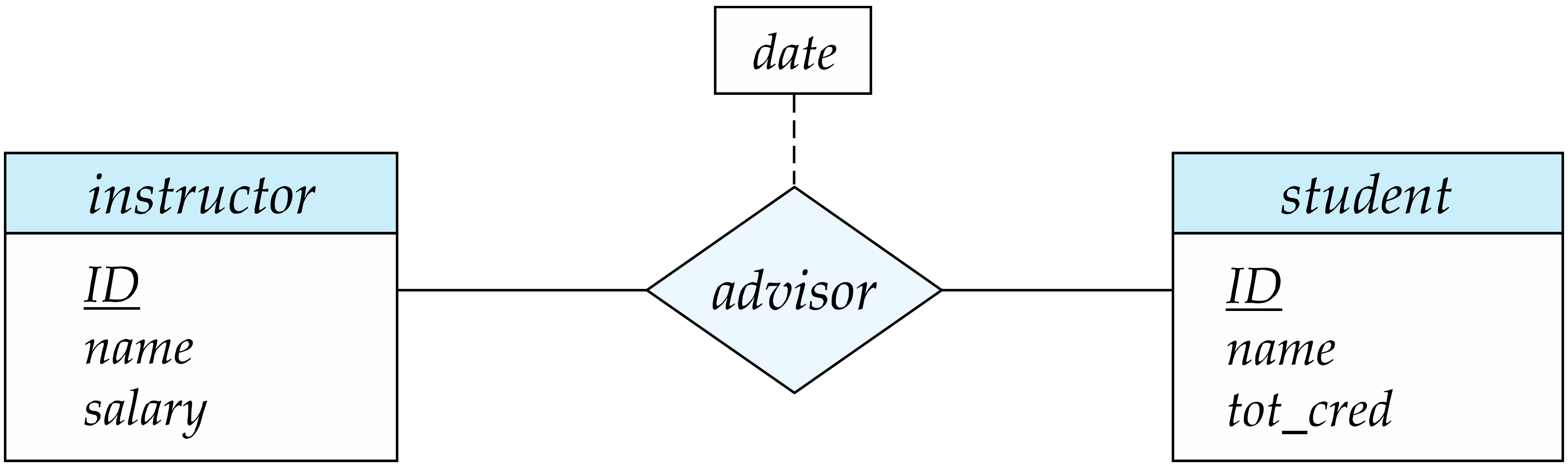
- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Attributes listed inside entity rectangle
- Underline indicates primary key attributes

- Note: we use a slight different (and simplified) notation for entity sets and attributes here!

Entity With Composite, Multivalued, and Derived Attributes

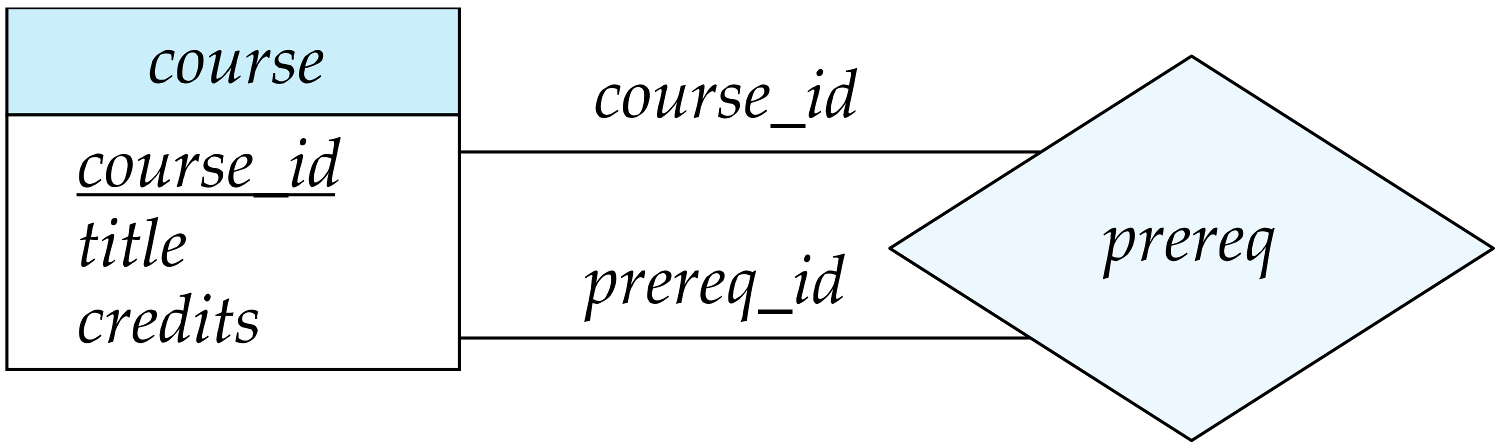
instructor
<ins>*ID*</ins> *name*   *first_name*   *middle_initial*   *last_name* *address*   *street*   *street_number*   *street_name*   *apt_number*   *city*   *state*   *zip* *{ phone_number }* *date_of_birth* *age ( )*

Relationship Sets with Attributes



Roles

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

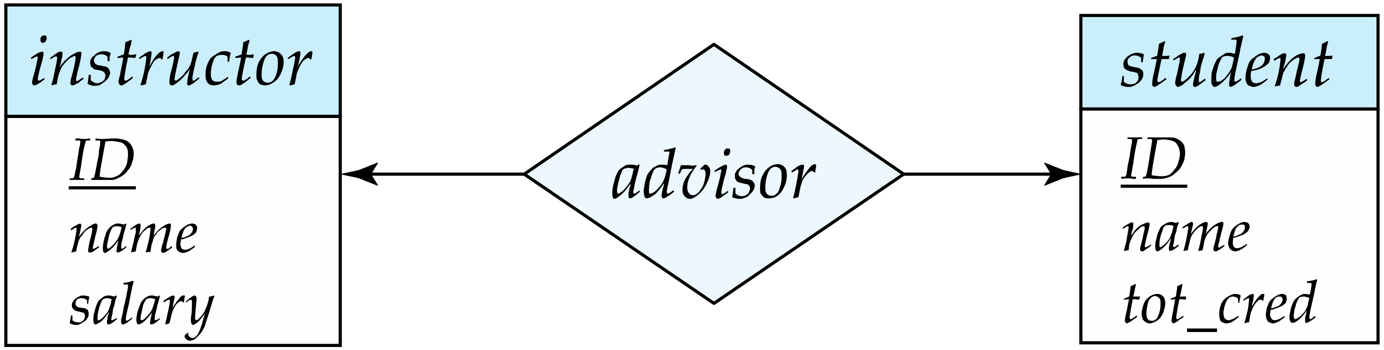


Cardinality Constraints

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

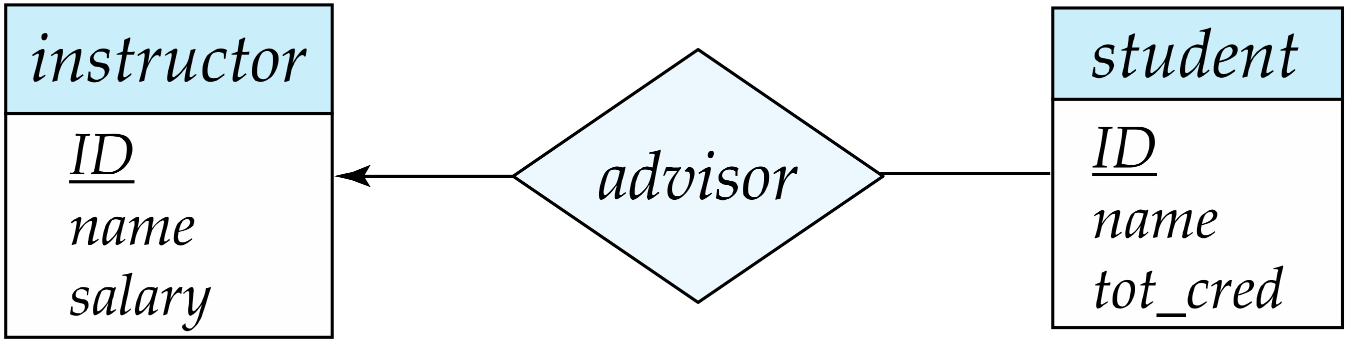
One-to-One Relationship

- one-to-one relationship between an *instructor* and a *student*
  - an instructor is associated with at most one student via *advisor*
  - and a student is associated with at most one instructor via *advisor*



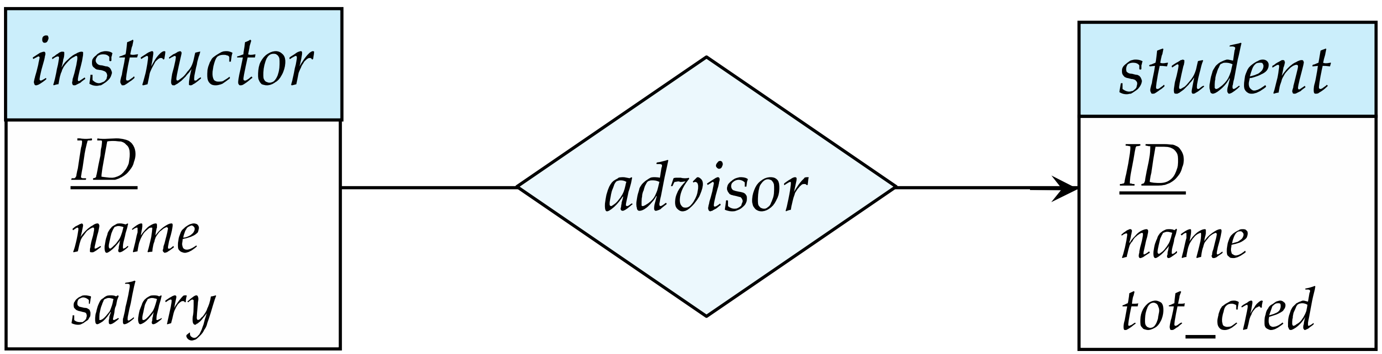
One-to-Many Relationship

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



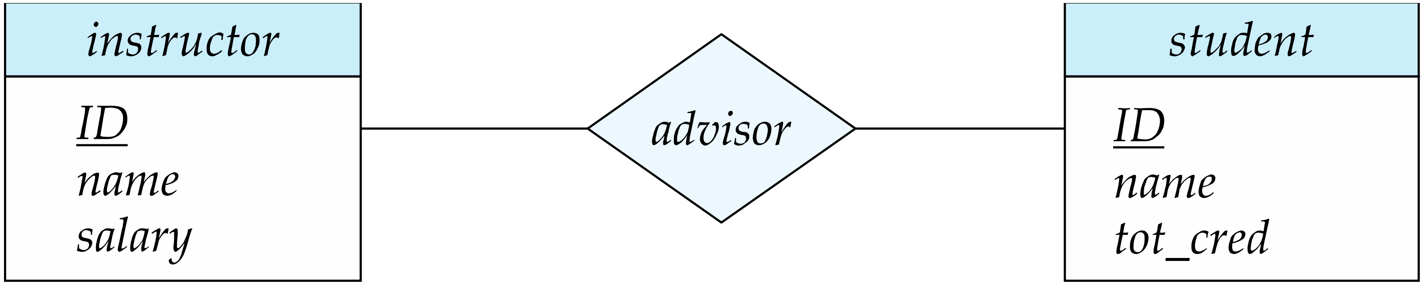
Many-to-One Relationships

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



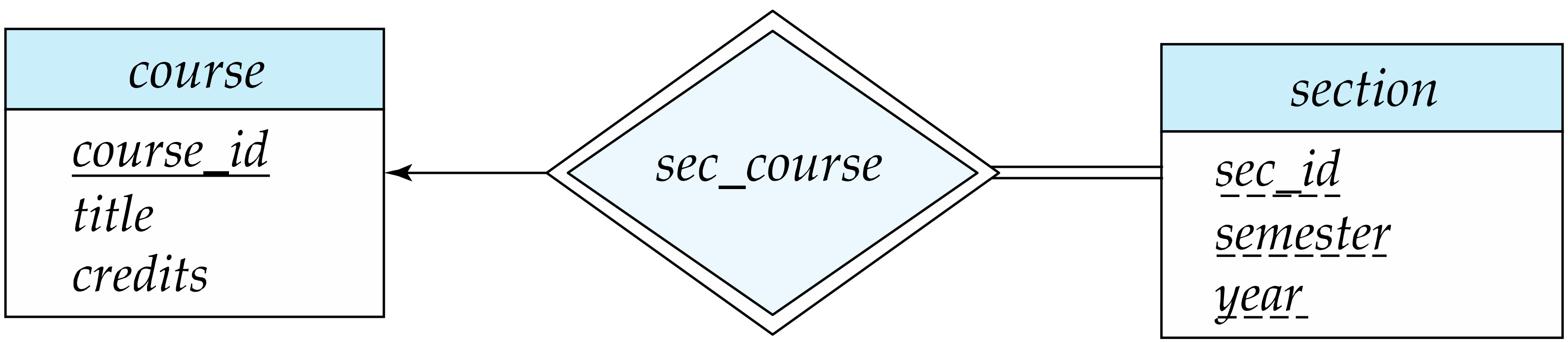
Many-to-Many Relationship

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



Participation of an Entity Set in a Relationship Set

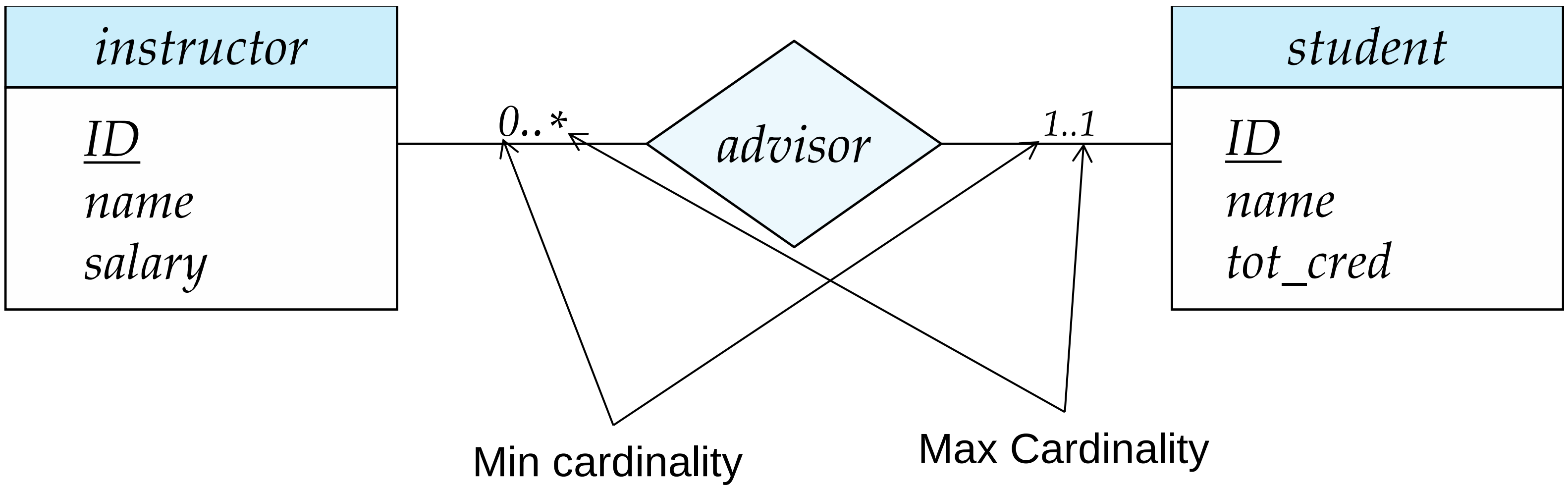
- Total participation (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set
  - E.g., participation of *section* in *sec_course* is total
    - every *section* must have an associated course
- Partial participation: some entities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



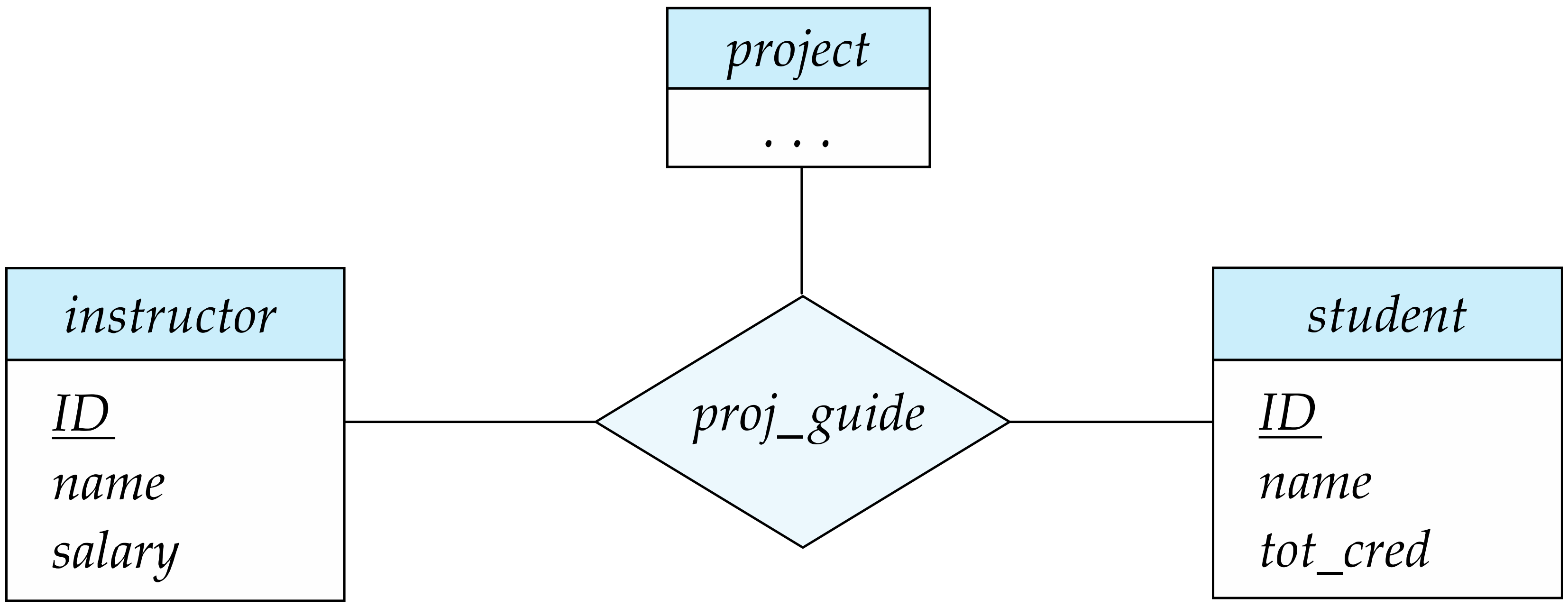
Note: doubly outlined diamond is a relationship set that identifies a weak entity set

Alternative Notation for Cardinality Limits

- Cardinality limits can also express participation constraints



E-R 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
- E.g., 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.
  - E.g., a ternary relationship *R* between *A*, *B* and *C* with arrows to *B* and *C* could mean

    1. each *A* entity is associated with a unique entity from *B* and *C* or

    2. each pair of entities from (*A, B*) is associated with a unique *C* entity, and each pair (*A, C*) is associated with a unique *B*
  - Each alternative has been used in different formalisms
  - To avoid confusion we outlaw more than one arrow

End