ER to Relational Model

Professor Jessica Lin

Reduction to Relation Schemas

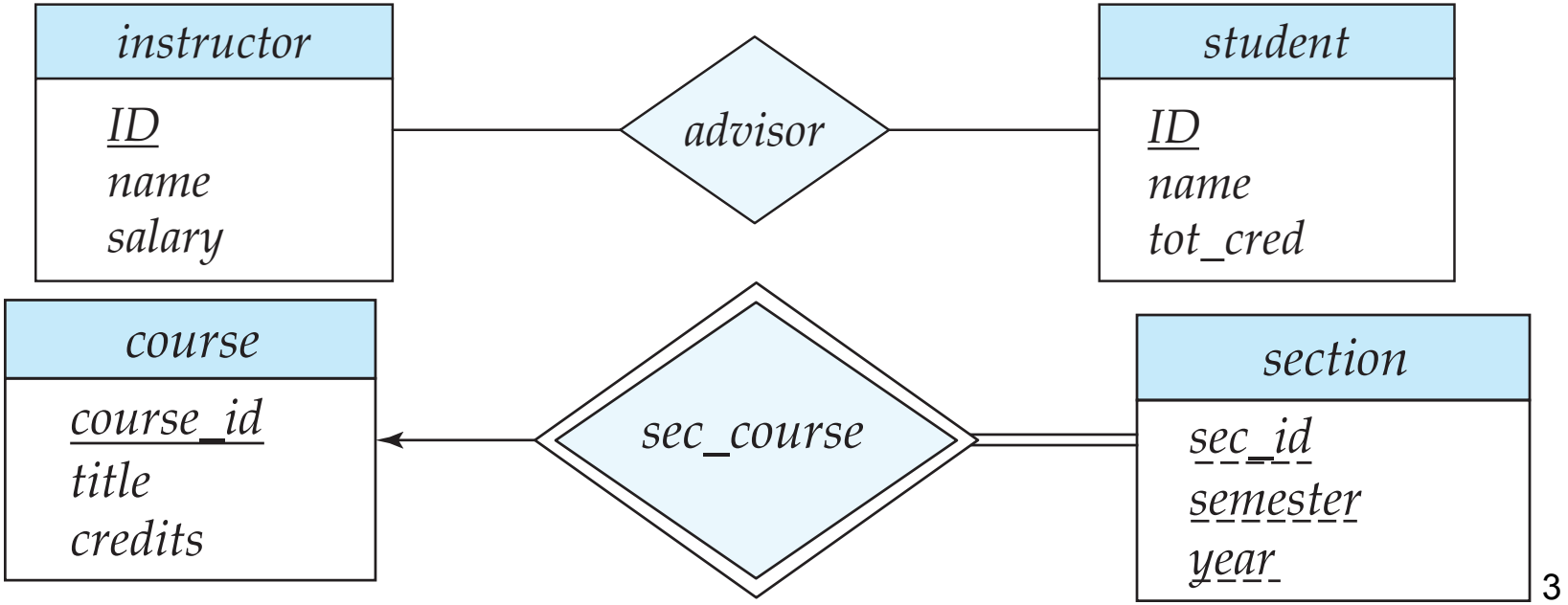
- Entity sets and relationship sets can be expressed uniformly as *relation schemas* that represent the contents of the database.
- A database which conforms to an E-R diagram can be represented by a collection of schemas.
- For each entity set and relationship set there is a unique schema that is assigned 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.

Translating ER diagrams into Relations

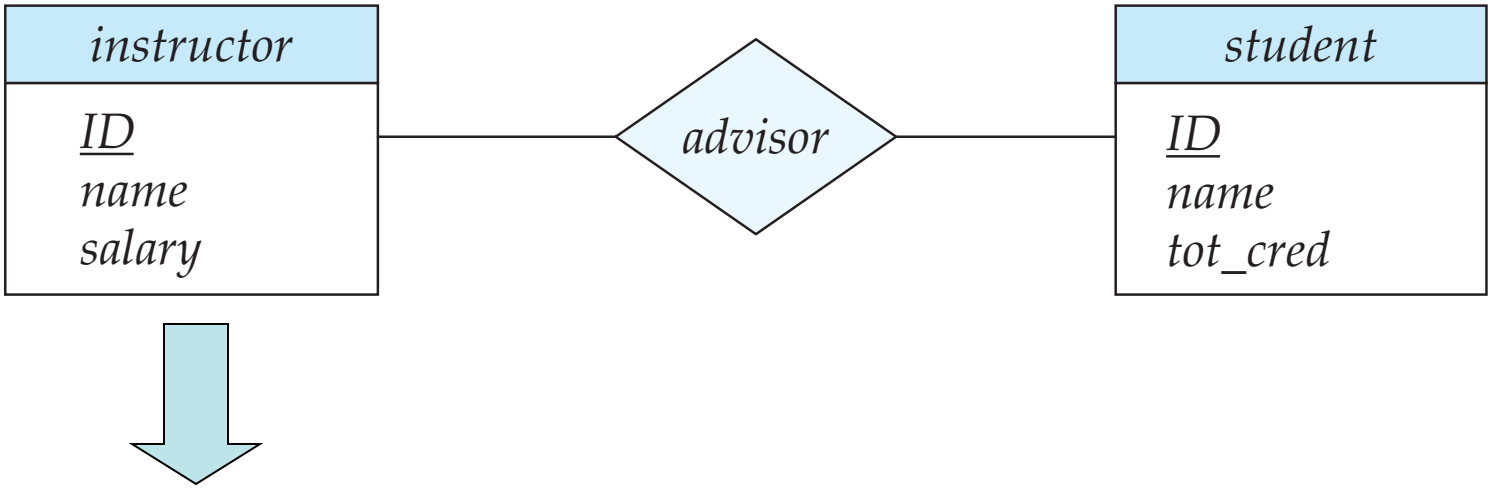
We need to figure out how to translate ER diagrams into relations.

There are three cases to worry about.

- Strong entity sets
- Weak entity sets
- Relationship sets



- **Strong entity sets**

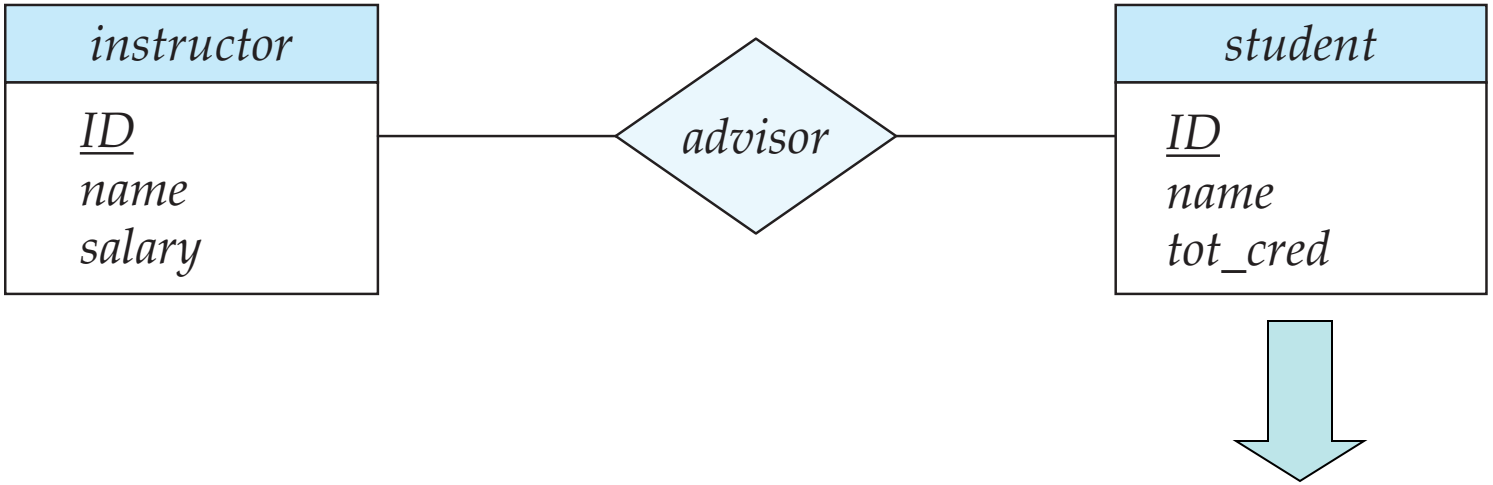


instructor(<u>*ID*</u>: string, *name*: string, *salary*: integer)

This is trivial, the primary key of the ER diagram becomes the primary key of the relation. All other fields are copied in (in any order)

ID	***name***	***salary***
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000

- **Strong entity sets**

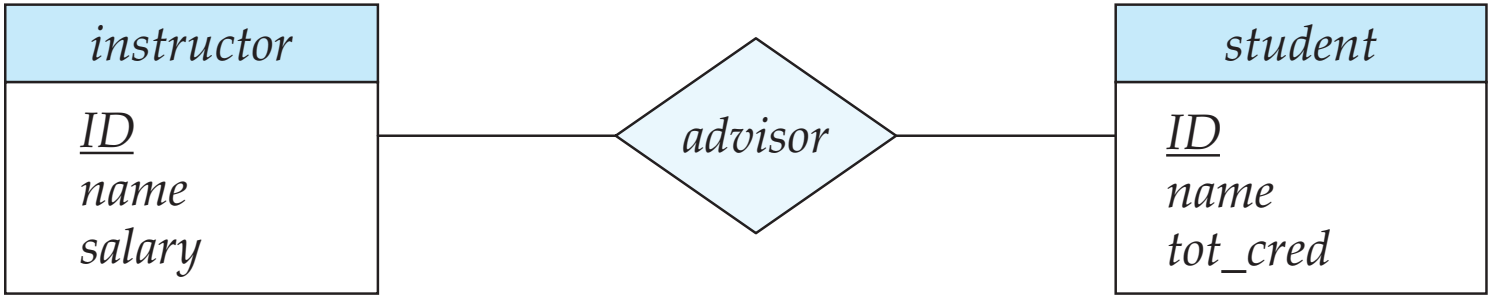


student(*ID*: string, *name*: string, *tot_cred*: integer)

This is trivial, the primary key of the ER diagram becomes the primary key of the relation. All other fields are copied in (in any order)

ID	***name***	***tot_cred***
98988	Tanaka	60
99001	Smith	51
90021	Jackson	12
95012	Lincoln	96

• Relationship entity sets



- For one-to-one relationship sets, the relation's primary key can be that of either entity set.
- For many-to-many relationship sets, the union of the primary keys becomes the relation's primary key
- For many-to-one or one-to-many relationship sets, the relation's primary key is the primary key of the "many" side of the relationship set.

The "imported" key from both entity sets is called the **foreign key**.

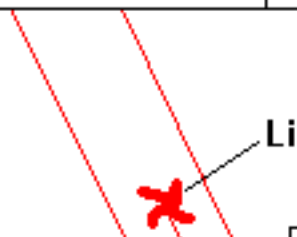
advisor(*i_ID*: string, *s_ID*: string)

i_ID	***s_ID***
10101	98988
10101	99001
12121	98988

Foreign Keys, Referential Integrity

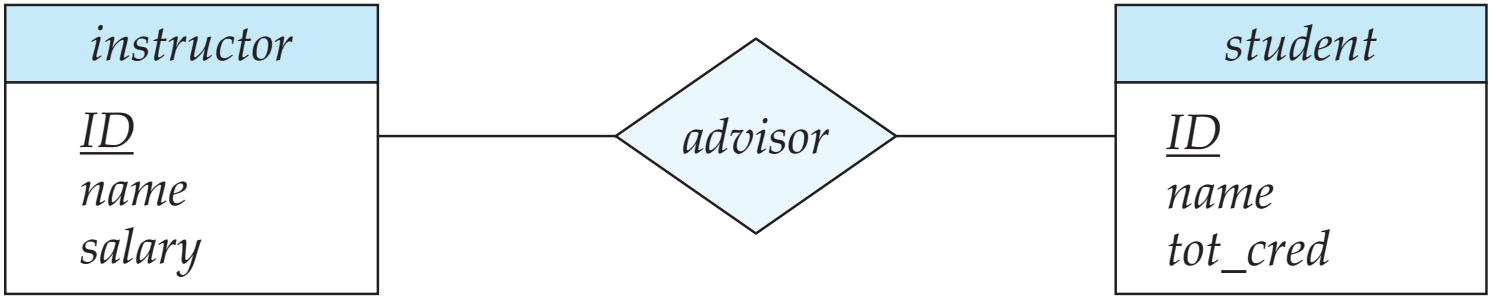
- *Foreign key* : Set of fields in one relation that is used to "refer" to a tuple in another relation. (Usually correspond to primary key of the second relation.) Like a `logical pointer' .
- e.g. *s_ID* is a foreign key referring to student(ID):
  - *advisor*(*i_ID*: string, *s_ID:* string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.
  - Can you name a data model w/o referential integrity?

artist_id	artist_name
1	Bono
2	Cher
3	Nuno Bettencourt



artist_id	album_id	album_name
3	1	Schizophonic
4	2	Eat the rich
3	3	Crave (single)

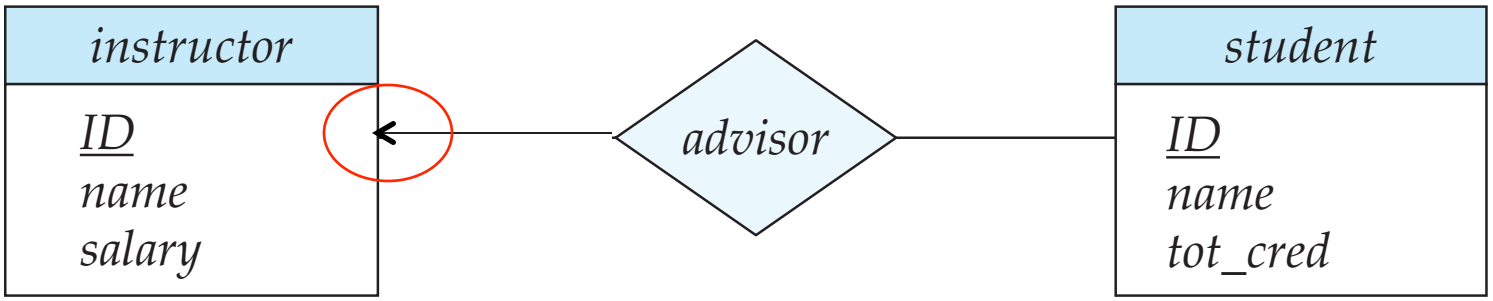
So, this ER Model…



… maps to this **database schema**

instructor(*ID*: string, *name*: string, *salary*: integer)
student(*ID*: string, *name*: string, *tot_cred*: integer)
advisor(*i_ID*: string, *s_ID:* string)

Suppose we add a constraint that a student can have at most one advisor…



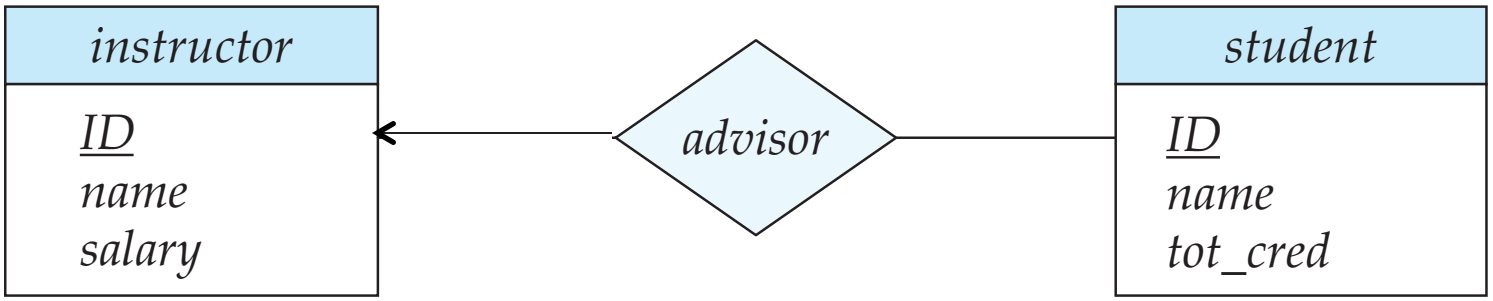
- For one-to-one relationship sets, the relation's primary key can be that of either entity set.
- For many-to-many relationship sets, the union of the primary sets, the union of the primary keys becomes the relation's primary key.
- For many-to-one or one-to-many relationship sets, the relation's primary key is the primary key of the "many" side of the relationship set.

The "imported" key from both entity sets is called the **foreign key**.

advisor(*s_ID*: string, *i_ID*: string)

s_ID	***i_ID***
98988	10101
99001	12121
90021	10101

The modified ER model…

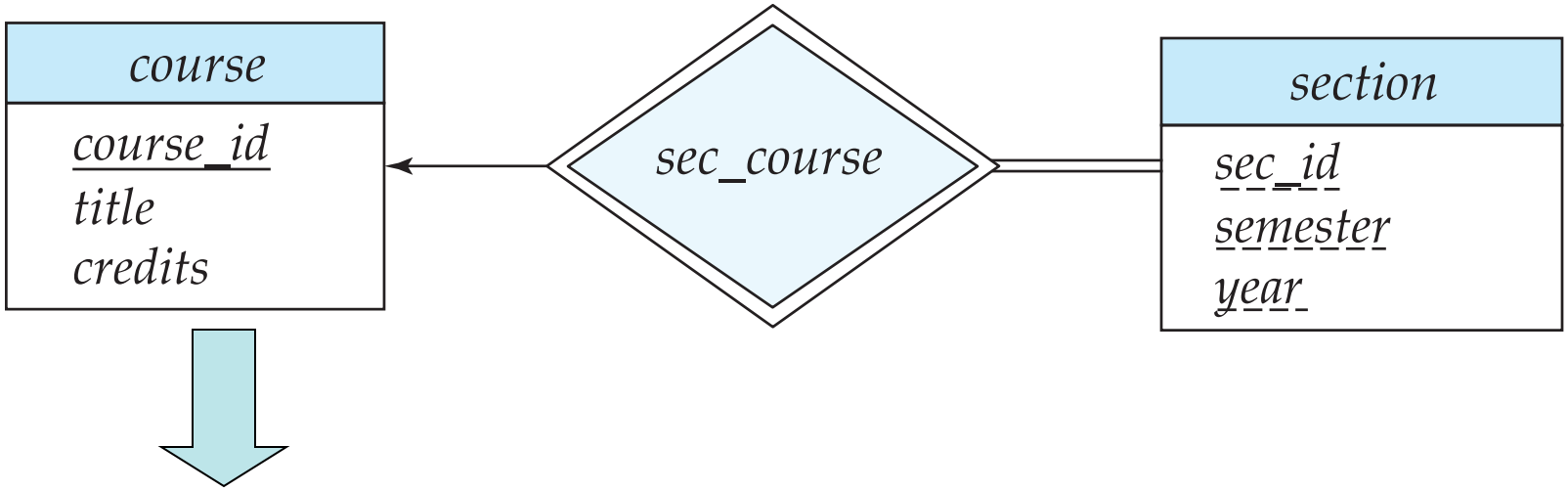


… maps to this **database schema**

instructor(*ID*: string, *name*: string, *salary*: integer)
student(*ID*: string, *name*: string, *tot_cred*: integer)
advisor(*s_ID*: string, *i_ID:* string)

Later we'll see how we can take advantage of the key constraint and come up with a better design.

Let's look at another example that involves a weak entity set.

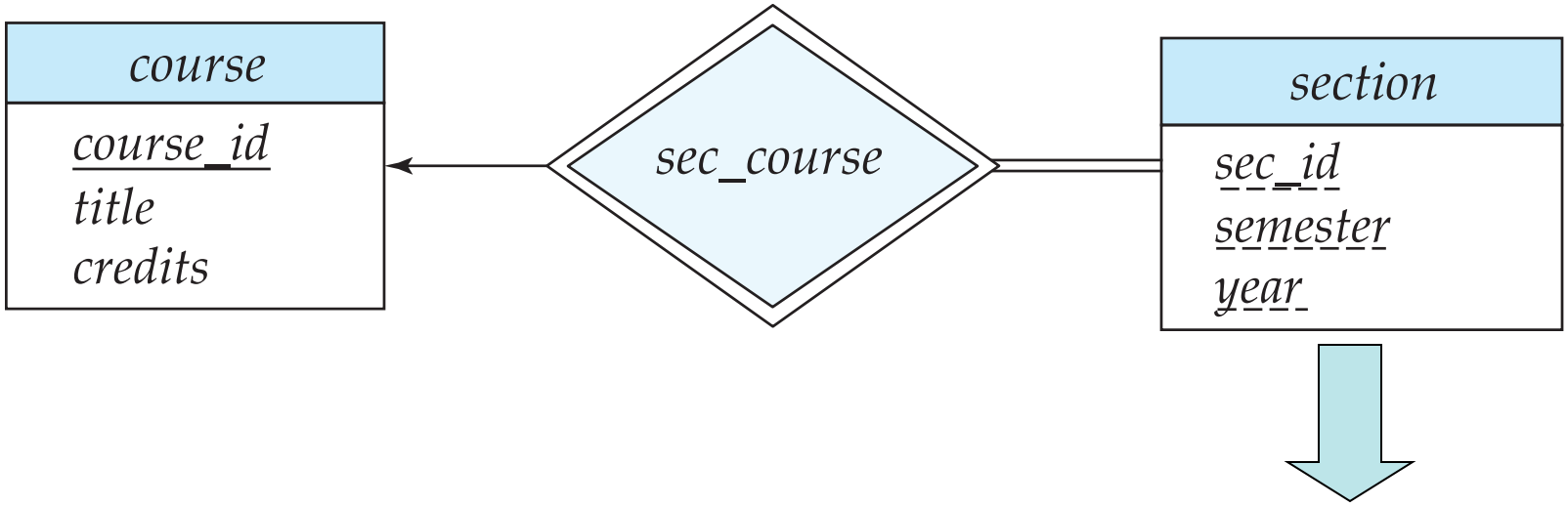


course(*course_id* : string, *title*: string, *credits*: integer)

course_id	***title***	***credits***
CS450	Database Concepts	3
CS484	Data Mining	3
CS310	Data Structures	3
CS101	Preview to CS	2

Strong entity set: trivial case

Let's look at another example that involves a weak entity set.



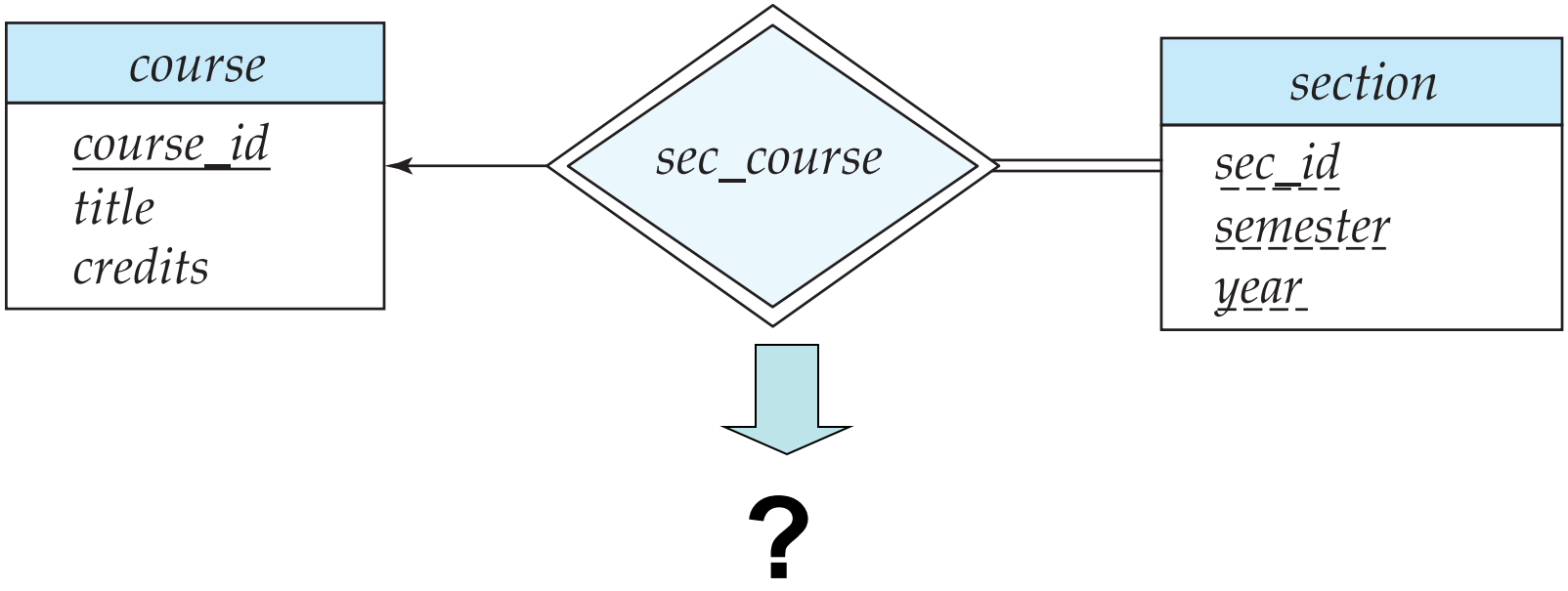
section(*course_id* : string, *sec_id*: string, *semester*: string, *year*: integer)

course_id	***sec_id***	***semester***	***year***
CS450	001	Fall	2015
CS450	002	Fall	2015
CS310	001	Fall	2015
CS450	001	Spring	2015

The primary key of the relation consists of the union of the primary key of the strong entity set and the discriminator of the weak entity set. The "imported" key from the strong entity set is called the **foreign key**.

All other fields are copied in (in any order)

- **Relationship entity sets**



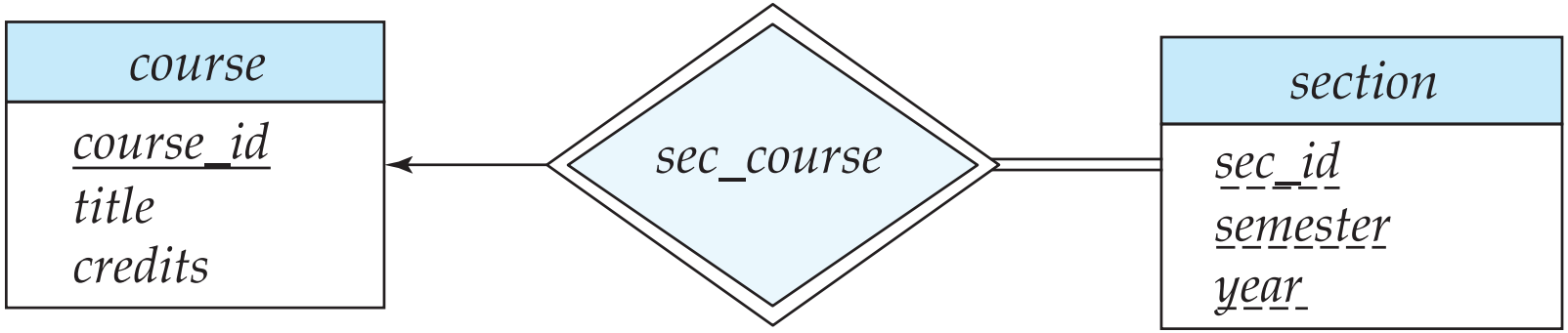
A relationship set involving a weak entity set is treated specially.

Since sec_course has no descriptiove attributes, the *sec_course* schema has attributes (*course_id, sec_id, semester, year)*.

Every (*course_id, sec_id, semester, year)* combination in a *sec_course* relation would also be present in the relation on schema section, and vice versa.

=> Combine them.

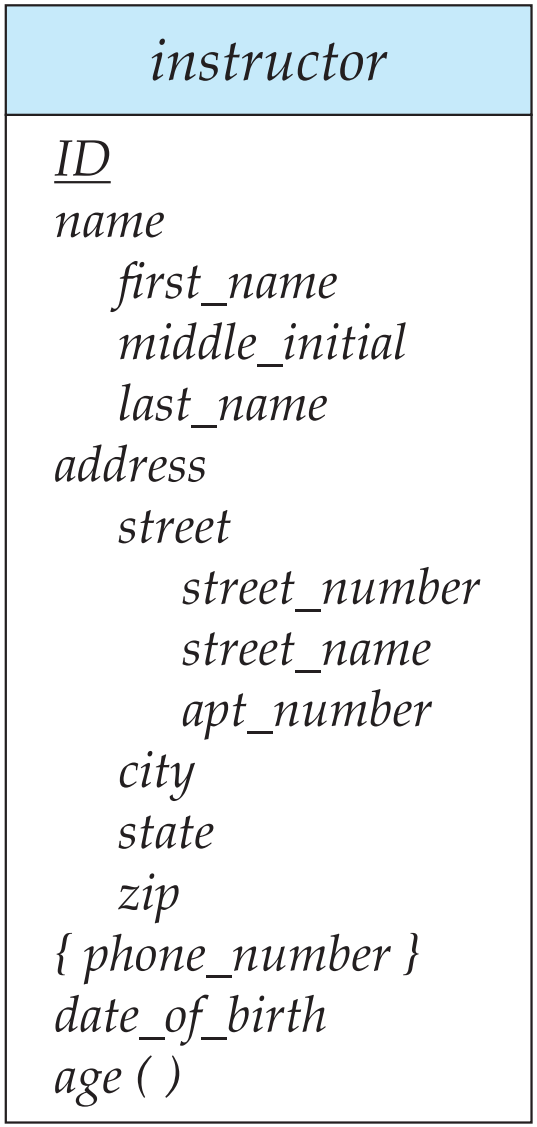
So, this ER Model…



… maps to this **database schema**

course(<u>*course_id*</u> : string, *title*: string, *credits*: integer)
section(<u>*course_id*</u> : string, <u>*sec_id*</u>: string, <u>*semester*</u>: string, <u>*year*</u>: integer)

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 Specialization via Schemas

- 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?

Representing Specialization as Schemas

- Method 2:
  - Form a schema for each entity set with all local and inherited attributes

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

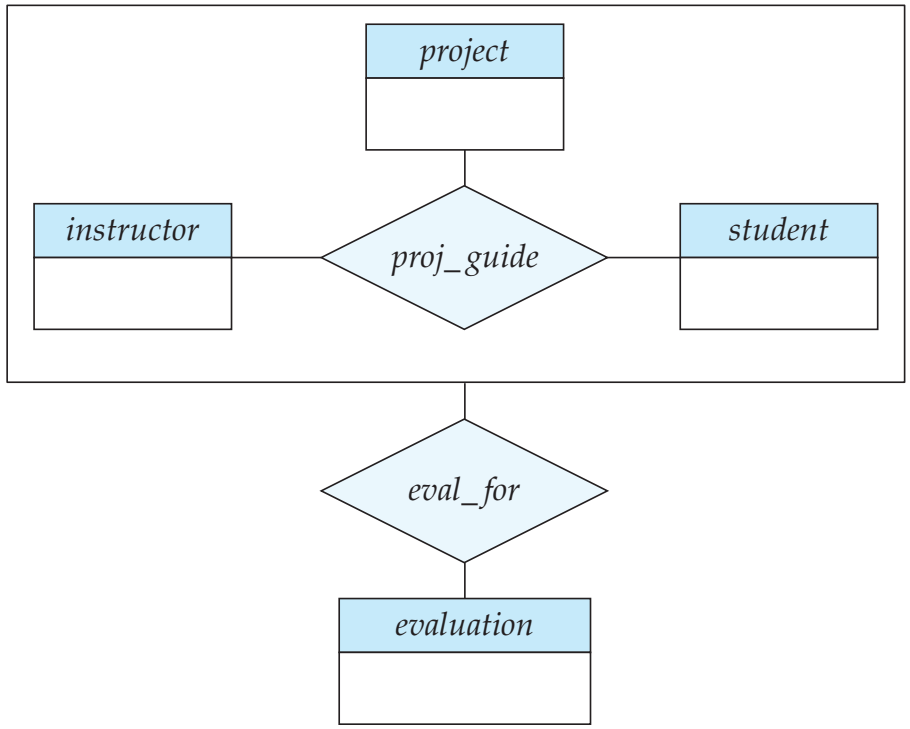
  - If specialization is total, the schema for the generalized entity set (*person*) not required to store information
    - Can be defined as a "view" relation containing union of specialization relations
    - But explicit schema may still be needed for foreign key constraints
  - Drawback?

Schemas Corresponding to Aggregation

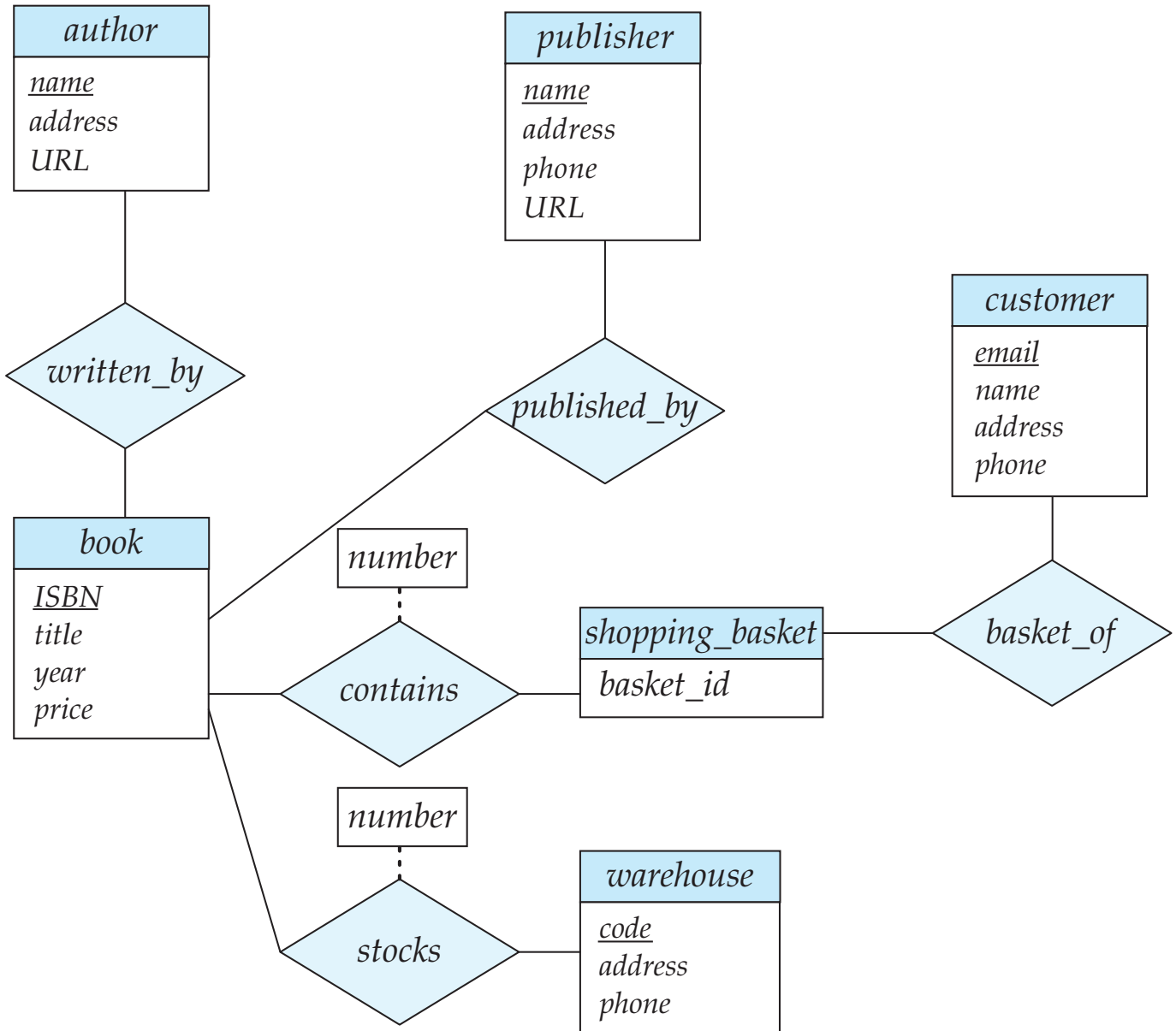
- To represent aggregation, create a schema containing
  - primary key of the aggregated relationship
  - the primary key of the associated entity set
  - any descriptive attributes

Schemas Corresponding to Aggregation

- For example, to represent aggregation manages between relationship works_on and entity set manager, create a schema
- *eval_for* (*s_ID, project_id, i_ID, evaluation_id*)
- Schema *proj_guide* is redundant provided we are willing to store null values for attribute *manager_name* in relation on schema *manages*



Practice


author
name
address
URL
publisher
name
address
phone
URL
customer
email
name
address
phone
written_by
published_by
book
ISBN
title
year
price
number
contains
shopping_basket
basket_id
basket_of
number
stocks
warehouse
code
address
phone

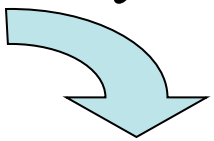
We have seen how to create a database schema, how do we create an actual database on our computers?

instructor(*ID*: string, *name*: string, *salary*: integer)
student(*ID*: string, *name*: string, *tot_cred*: integer)
advisor(*s_ID*: string, *i_ID:* string)

…how do we create an actual database on our computers?

We use SQL, a language that allows us to build, modify and query databases.

instructor(<u>*ID*</u> : string, *name* : string, salary: real)



```
SQL> CREATE TABLE instructor
  2  (ID CHAR(5),
  3  name CHAR(50),
  4  salary integer,
  5  PRIMARY KEY(ID));

Table created.
```

SQL (Structured Query Language)

- SQL is a language that allows us to build, modify and query databases.
- SQL is an ANSI standard language. American National Standards Institute
- SQL is the “engine” behind Oracle, Microsoft SQL Server, etc.
- Most of these systems have built GUIs on top of the command line interface, so you don’t normally write statements directly in SQL (although you can).

Creating Relations in SQL

- Creates a *student* relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
SQL> CREATE TABLE student
    (ID CHAR(5),
     name CHAR(50),
     tot_cred integer,
     PRIMARY KEY(ID));
```

- As another example, the *advisor* table holds information about the advising relationship between instructors and students.

```
SQL> CREATE TABLE advisor
    (s_ID CHAR(5),
     i_ID CHAR(5),
     PRIMARY KEY(s_ID));
```