From E-R Model to Relational Database

COM 3563: Database Implementation

Avraham Leff

Yeshiva University

avraham.leff@yu.edu

COM3563: Fall 2020

Today's Lecture: Overview

1. From E-R to Relational

2. Schema Redundancy

3. Review Of Common Relationship Set E-R $\Rightarrow$ RDB Mappings

Much material derived from lectures created by Professor Zvi Kedem, used with permission

Previous Lecture: The E-R Model

- E-R is valuable because it's a semi-formal design technique based upon informal semantics ☺
- When using E-R, our goal is to create schemas that accurately represent and formalize the information we acquired during the "requirements gathering" phase
  - The E-R modeling is valuable even though we know that complete accuracy isn't always possible
  - Tip-off: lots of "natural language" annotations
  - Tip-off: specialized E-R diagrams that are supported only by a given modeling tool
- The E-R modeling process
  1. Identify entity sets and their attributes
  2. Identify relationships between entity sets, and their attributes
  3. Identify maximum and minimum relationship cardinality
  4. Identify weak entity sets

Today's Plan

- The real benefit of E-R modeling stems from the ability to generate "real" RDB schema from the E-R model
  - Implication: we can iteratively refine the "high-level" E-R model, based on the concrete RDB schema that it generates
- The two data-models are sufficiently similar that we can "compile down" the higher-level model (E-R) to a lower-level (RDB) "executable" version
- Key point: this transformation can be applied in quasi-algorithmic fashion

From E-R to Relational

Basic Idea

- E-R in a nutshell: entity sets and relationship sets ☺
- Entity sets and relationship sets can both be transformed cleanly to RDB schemas
  - E-R diagram → collection of RDB schemas
  - With minimal "hackery" ☺
- 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
- The non-intuitive part is that we're converting relationship sets to schemas ("tables")!

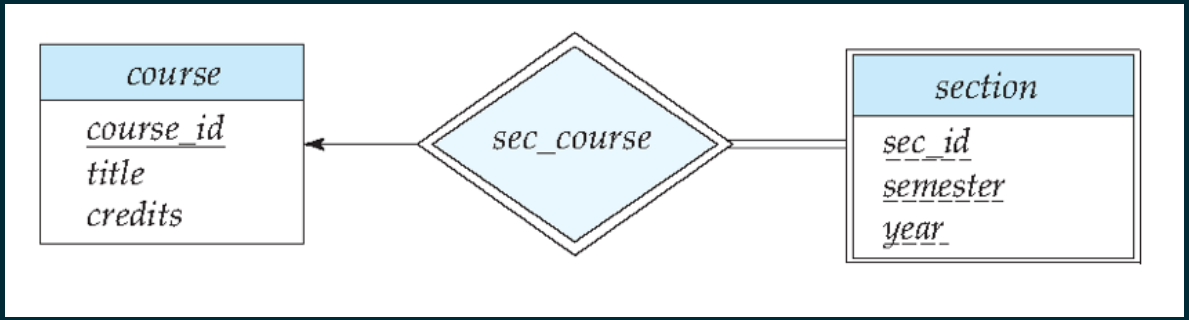
Representing Entity Sets

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

```
student(ID, name, tot_cred)
```

- Less obvious: a weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set
  - Example from last lecture: concept of a section doesn't have independent existence apart from a course
  - Discriminator for section: *(section_id, semester, year)*
  - Primary key of course is *course_id*

```
section (course_id, section_id, semester, year)
```



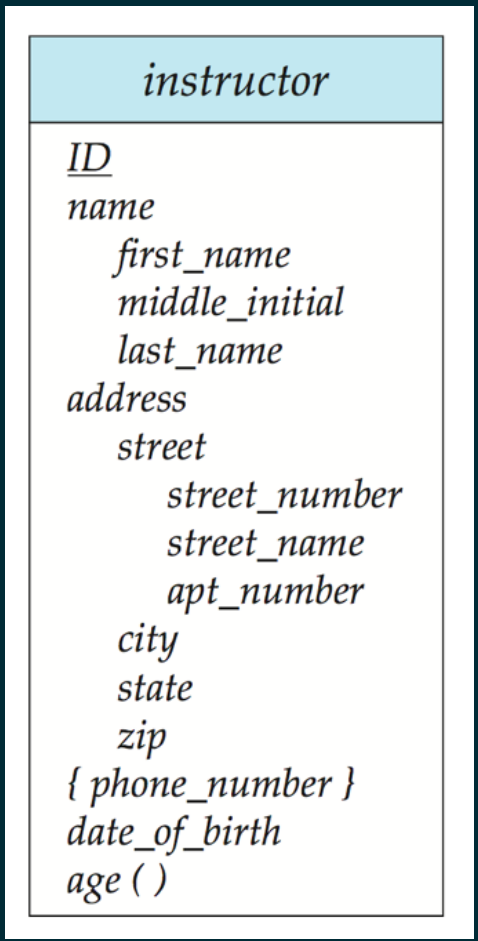
Weak Entity Sets

- Q: How does a relational database capture the concept that a "weak entity set" is existence dependent on some "strong entity set"?
- A: By creating a foreign-key constraint on the corresponding "weak" relation specifying that
  - Each of its attributes "derived from" the primary key of the "strong" relation reference the primary key of that relation
  - Example: *course_id* in schema *section* has a foreign key constraint on the primary key of the *course* schema
  - Can also add an integrity constraint on this *section* foreign-key constraint specifying an ON DELETE CASCADE option
    - This ensures that deleting a *course* tuple results in all associated *section* tuples being deleted as well

E-R ⇒ RDB Mapping (Steps 1 & 2)

- Step 1: mapping of strong entity sets
  - For each entity set *E* in the E-R schema, create a relation *R* that includes all the simple attributes of *E*
  - Choose one of the key attributes of *E* as the primary key for *R*
  - If the chosen key of *E* is composite, the set of simple attributes will form the primary key of *R*
- Step 2: mapping of weak entity sets
  - For each weak entity set *W* in the E-R schema with owner entity set *E*, create a relation *R* and include all simple attributes of *W* as attributes of *R*
  - Also, include as foreign key attributes of *R* the primary key attribute(s) of the relation(s) that correspond to the owner entity set(s)
  - The primary key of *R* is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity set *W*, if any

Entity Sets With Composite Attributes



- Composite attributes are "flattened" by creating a separate attribute for each component attribute
- Example: entity set instructor with composite attribute *name* composed of component attributes *first_name* and *last_name* → two schema attributes
  - *name_first_name* and *name_last_name*
- Can omit the "name prefix" if no resulting ambiguity
  - OK to transform *name_first_name* into *first_name*
- (Temporarily ignoring) the issue of multivalued attributes, extended *instructor* schema is shown below

```
1  instructor(ID, first_name, middle_initial, last_name, street_number,
2  street_name, apt_number, city, state, zip_code, date_of_birth)
```

MultiValued Attributes

- Example from last lecture: phone numbers (because *instructor* can have 0, 1, or more phone numbers)
- A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM*
- Schema *EM* has primary key 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

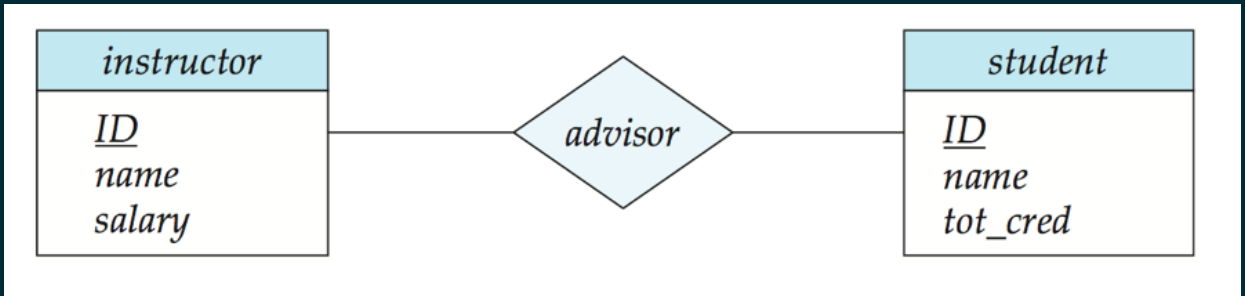
```
instructor_phone= (ID, phone_number)
```

- Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
- Example: instructor entity with primary key *22222* and phone numbers *456-7890* and *123-4567* maps to two tuples
  - (22222, 456-7890)
  - (22222, 123-4567)
- Much more fun to work with NoSQL databases for multivalued attributes ☺

Schema Generated From Multivalued Attributes

- Transformation process created a new schema from the multivalued attribute *phone number*
- We seem to have lost the information that *instructor_phone* was derived from the *instructor* entity set
- Q: can you suggest how we might somehow keep this information?
- A: Create a foreign-key constraint on the new schema
  - Attribute generated from primary key of entity set references the schema generated from that entity set
  - For *instructor_phone* schema, place foreign-key constraint such that attribute *ID* references the *instructor* relation

Representing Relationship Sets



- A relationship set (independently of the participating entity sets) is transformed into a schema
- Many-to-many relationship set is represented as a schema with
  - Attributes for the primary keys of the two participating entity sets and
  - All descriptive attributes of the relationship set itself
- Example from last lecture: relationship set *advisor* is transformed into

```
advisor = (s_id, i_id)
```

E-R ⇒ RDB Mapping (Steps 3 & 4)

- Step 3: mapping of multi-valued attributes
  - For each multi-valued attribute A, create a new relation R
  - This relation R will include an attribute corresponding to A, plus the primary key attribute K (as a foreign key in R) of the entity set that has A as an attribute
  - The primary key of R is the combination of A and K
    - If the multi-valued attribute is composite, include its simple components in the primary key
- Step 4: mapping of binary $M : N$ relationship types
  - For each binary M:N relationship set R, create a new ("relationship relation") relation S to represent R
  - The primary keys of the participating entity sets become foreign key attributes in S
    - The combination of these primary keys form the primary key of S
  - Include any attributes of the $M : N$ relationship set itself as attributes of S

E-R ⇒ RDB Mapping (Step 5): Mapping Binary 1 : *N* Relationships

- For each binary 1 : *N* relationship type *R*, create a relation *S* to represent the participating entity set at the "N-side" of the relation
  - Example: given an *Employee* ⇒ *Department* WORKS_IN relationship ...
  - ...*Employee* is the "N-side" of the relation
- Include the primary key of the relation *T* that represents the other entity set participating in *R* as a foreign key in *S*
  - In our example: include "department_number" as a foreign key in *Employee*
- Include any simple attributes of the 1 : *N* relation type as attributes of *S*

E-R ⇒ RDB Mapping (Step 6): Mapping Binary 1 : 1 Relationships (I)

- For each binary 1 : 1 relationship type R in the E-R schema, identify the relations S and T that correspond to the entity types participating in R
- There are three possible approaches:
  - Foreign key ("2 relations") approach: choose one of the relations (e.g., S) and include the primary key of T as a foreign key in S
    - Ideally: the entity type playing the role of S should have a total participation semantics in R
  - Merged relation ("1 relation") approach: merging S and T into a new single relation
    - This approach is reasonable when both participations are total
    - The relationship informations is now modeled implicitly

E-R ⇒ RDB Mapping (Step 6): Mapping Binary 1 : 1 Relationships (II)

- A third approach is possible as well
  - Cross-reference ("3 relations") approach: model the relationship explicitly by creating a third relation X
  - X "cross-references" the primary keys of S and T
    - This is sometimes called the "relationship relation" approach
    - The attributes of S and T are mapped directly from the participating entity types: the relationship between them is captured as "primary key" duples in X

Relationship Sets & Primary Keys: Summary (I)

- The naive approach for determining the primary key of the schema that implements the E-R relationship is to use the union of the participating entity sets
  - Use this as a "first approximation" approach for non-binary relationship
- But we can do better than "union of primary keys" by taking advantage of the fact that (many) binary relationships express a partial or total function between the two (participating) entity sets
- Binary One-to-One: primary key of either entity set becomes the primary key of the relation
  - Do you see why it doesn't matter which one we choose?
- Binary Many-to-One (or One-to-Many): primary key of the entity set on the "many" side of the relationship becomes the primary key of R'

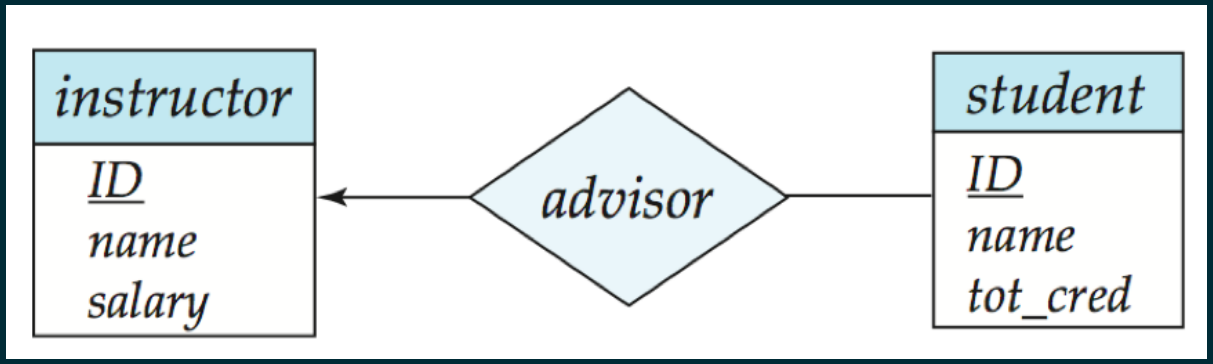
Relationship Sets & Primary Keys: Summary (II)

- Binary Many-to-Many: union of the primary keys of the participating entity sets becomes the primary key of R'
  - This is the "relationship relation" approach that we mentioned as a possibility for $1:1$ binary relationships
  - For $M:N$ relationships, this is the only approach possible
- Note: the above rules for binary relationships simply express the fundamental semantics of "primary keys"
  - A primary key uniquely identifies a tuple
  - By understanding "partial" and "total" participation semantics, we optimize the amount of primary key information that's needed to encode the relationship between the participating entity sets

Relationship Sets: Summary of Foreign-Key Constraints Semantics

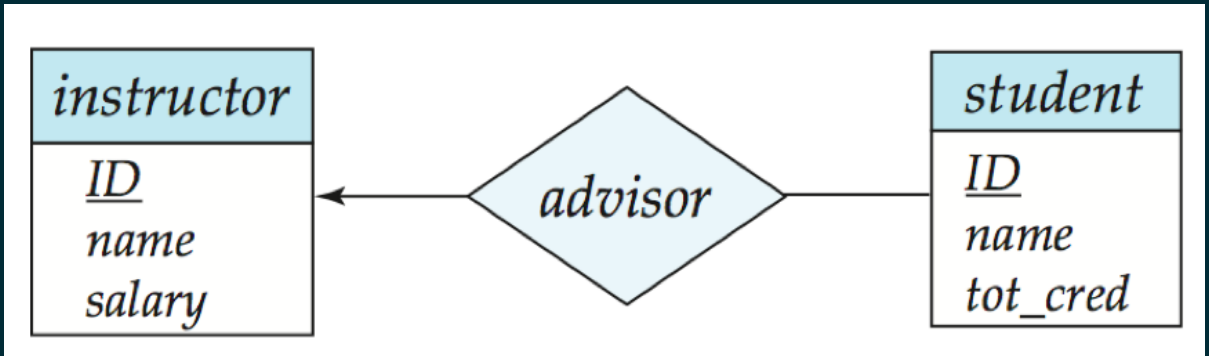
- Key point: these rules should be intuitive
  - A foreign-key is a column or group of columns that serve as a "cross-reference" or "link" between database tables
  - The "cross-reference" is implemented by specifying the primary key of another table
- Assume that each participating entity set E_i has a corresponding relational schema E'_i
- Recall: R' is the relation schema derived from relationship set R
- For every set of attributes in schema R' derived from primary key attributes of E'_i, create a foreign-key constraint from R' such that
  - The attributes of R' derived from the primary key attributes of E'_i ...
  - ...Are declared to reference the primary key of the relation schema E'_i

Relationship Set: Example (I)



- Relationship set *advisor* associates
  - *instructor* entity set with primary key *id*
  - *student* entity set with primary key *id*
  - In a one-to-many (instructor-to-student) binary relationship
- Relationship has no attributes of its own so *advisor* schema only has two attributes
  - Union of E_i primary key attributes
  - Must rename them as *instructor_id* and *student_id*
  - Primary key of *advisor* is the primary key of the "many side" of the relationship
    - Or: *student_id*

Relationship Set: Example (II)



- Create two foreign-key constraints in *advisor* schema
  - Attribute *instructor_id* references primary key of *instructor*
  - Attribute *student_id* references primary key of *student*

Relationship Set: Example (III)

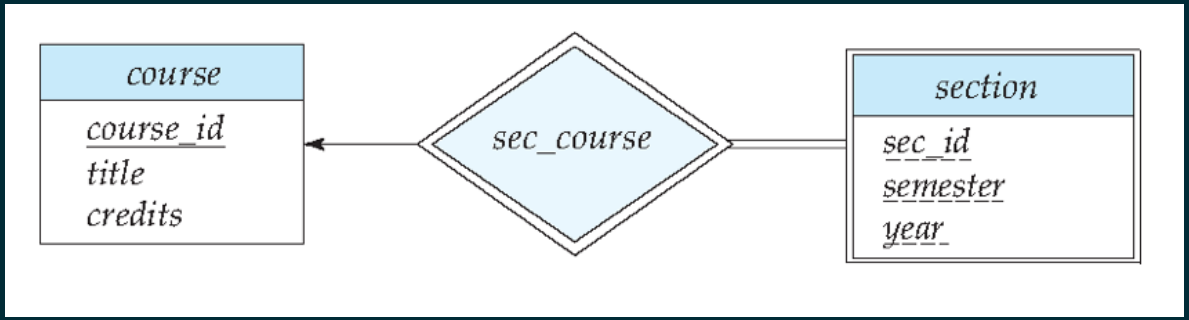
```
create table instructor(
  id integer not null,
    -- omitting other attributes

  primary key (id)
);

create table student(
    id integer not null,
    -- omitting other attributes
  primary key(id)
);

create table advisor(
  instructor_id integer not null,
  student_id integer not null,
  primary key(student_id),
  foreign key(student_id) references student(id),
  foreign key(instructor_id) references instructor(id)
);
```

Schema Redundancy

Introduction

- Previous slides presented an algorithm for E-R → relational mapping
  - The algorithm is definitely correct and straightforward to implement ...
  - But: it can generate many redundant schema
- What follows are some guidelines for *combining* or *removing* some of the redundant schema

Weak Entity Sets



- The schema corresponding to a relationship set that links a weak entity set to its identifying strong entity set is redundant
  - Relationship *sec_course* has no attributes of its own
  - The *section* schema already contains the primary-key attributes of the strong entity set (*course*) (see earlier discussion in today's lecture)
    - These (plus the attributes of the *section* entity-set) are precisely the attributes that would appear in the *sec_course* schema
  - Every tuple in *sec_course* has a matching tuple in *section* (and vice versa)
- Relationship schema is redundant: eliminate it!

Take Advantage of Total Participation

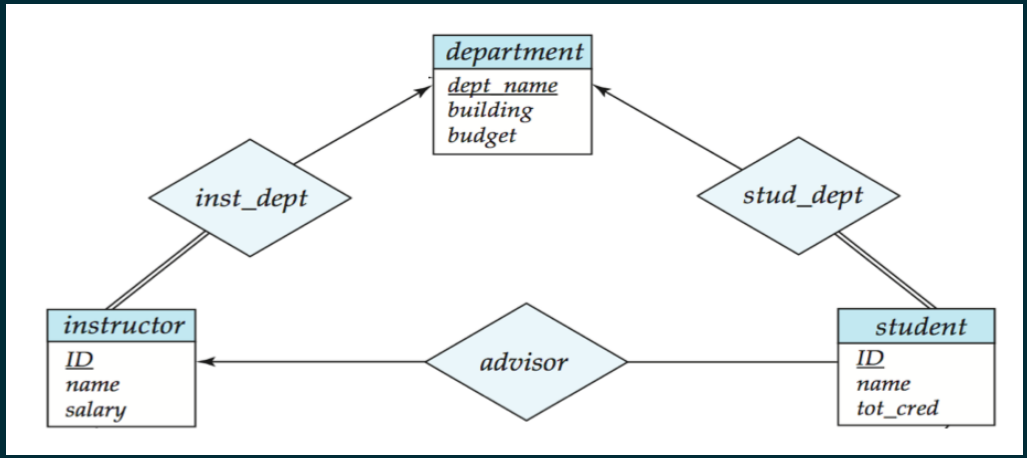
- Reminder: if every entity e_1 in E_1 must participate in a relationship set with E_2
  - We say that E_1 has a total participation in that *many-to-one* relationship
- Proposal: instead of three schemas: E_1, E_2, R ...
- ... Let's combine E_1 and R into a a single schema, leaving us with two schema
- Intuition: because of the total participation property, no need to keep E_1 information separate from the relationship information
  - Note: the primary key of R is already the primary key of E_1
    - Because E_1 is on the "many" side of the relationship
  - So: this proposal doesn't add new attributes to primary key of E_1
  - We only need to add relation-specific attributes (if any) to "enhanced" E_1

Total Participation ⇒ Combining Schema

- Conceptually: merge R into E_1
- Schema attributes of enhanced E_1 are union of E_1 and R attributes
  - Which is the union of the primary key attributes of E_1 and E_2 and descriptive attributes of R (if any)
- Primary key of enhanced E_1 is primary key of original E_1
- Only need one foreign-key constraint, to E_2
- Result: we've eliminated the R schema ☺

- Note: we've previously introduced this "merge" approach (for total participation) in the context of $1:1$ relationships!
- In that scenario, we can arbitrarily select either of the participating entity sets as the one that "assimilates the relationship set"
- The "merge" will not affect the primary key of the E_i into which the relationship is merged

Combining Schema: Example



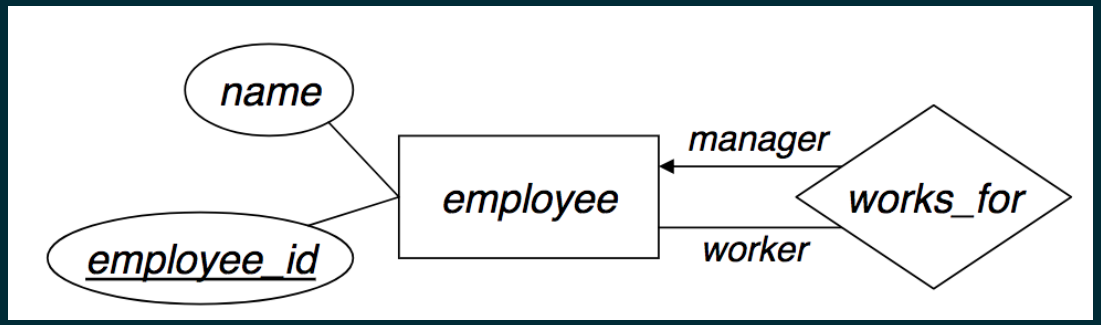
- This E-R diagram states that every instructor must be associated with some department
- Idea: instead of creating a schema for relationship set *inst_dept* ...
  - Add the *dept_name* attribute to the "enhanced" *instructor* schema
  - Because that's the primary key of the *other entity set (department)* (the "1-side" of the $1 : N$ relationship)
- *instructor* attributes are now *ID*, *name*, *dept_name*, *salary*
- *instructor* schema now has foreign-key constraint: attribute *dept_name* references the *dept* schema

Returning to Weak Entity Sets

- I hope that you now see that the approach for eliminating the weak entity set relationship is just a special case of the "total participation" approach that we just discussed
- In the case of a "weak entity set" we have …
  - A many-to-one relationship
  - The weak entity-set has total participation in the relationship
  - There are no descriptive attributes in the relationship
  - The weak entity-set's attributes already include the primary key attributes of the strong entity set

Must We Have The Total Participation Property?

- Even if participation is partial on the "many" side, can still combine schemas!
- Trick: E_1 (the "many side") must store NULL values for all E_2 primary key attributes whenever an entity in E_1 has no mapping to an E_2 entity



- Manager to employee mapping is one-to-many
- Relation schemas were
  - employee(employee_id, name)
  - works_for(employee_id, manager_id)
- Can combine into
  - employee(employee_id, name, manager_id)
  - But: Must store NULL for employees with no manager (the CEO))

Merge Approach: Advantages & Disadvantages

- Advantages of combining schema:
  - Eliminate a foreign-key constraint (improve performance)
  - Eliminate an extra JOIN operation in certain queries
- Disadvantages of combining schema:
  - May require use of NULL values (partial participation case)
  - Makes it harder to change mapping cardinality constraints *in the future*

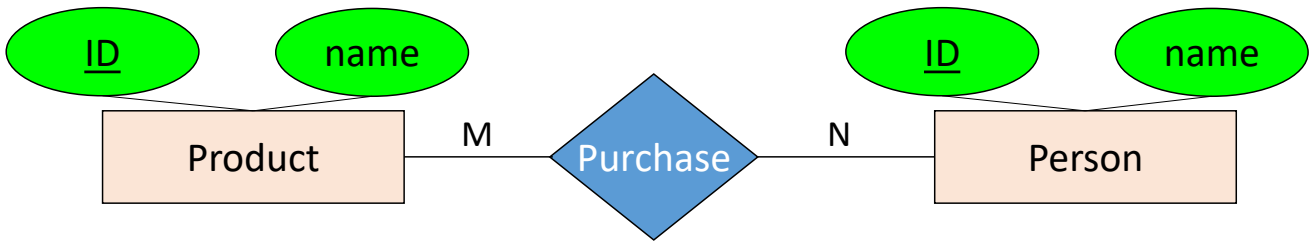
Summary of E-R-Relational Mapping Constructs

ER MODEL	RELATIONAL MODEL
Entity type	*Entity* relation
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and *n* foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

Table copied from “Fundamentals of Database Systems” (Elmasri & Navathe)

Review Of Common Relationship Set E-R ⇒ RDB Mappings

Many-to-many



Product

ID	name

Purchase

prod_ID	person_ID

Person

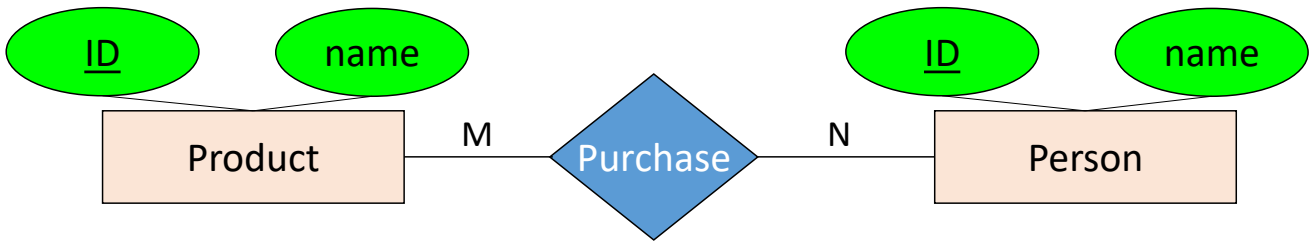
ID	name

```
CREATE TABLE Product (
   …
   PRIMARY KEY (ID)
);
```

```
CREATE TABLE Purchase (
   …
   PRIMARY KEY (prod_ID, person_ID),
   FOREIGN KEY (prod_ID) REFERENCES Product (ID),
   FOREIGN KEY (person_ID) REFERENCES Person (ID)
);
```

```
CREATE TABLE Person (
   …
   PRIMARY KEY (ID)
);
```

Many-to-many – a question



Purchase

prod_ID	person_ID

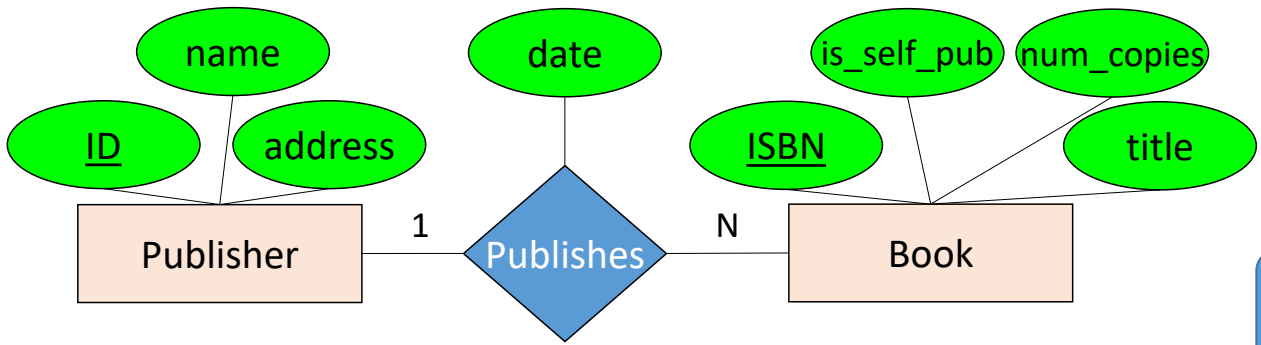
No. E.g., a **Product** never purchased simply never appears in **Purchase** table.

Primary key fields also NOT NULL.

Doesn't that force min. cardinality = 1?

```
CREATE TABLE Purchase (
   …
   PRIMARY KEY (prod_ID, person_ID),
   FOREIGN KEY (prod_ID) REFERENCES Product (ID),
   FOREIGN KEY (person_ID) REFERENCES Person (ID)
);
```

One-to-many / many-to-one



Publishes table optimized away.

Publisher

ID	name	address

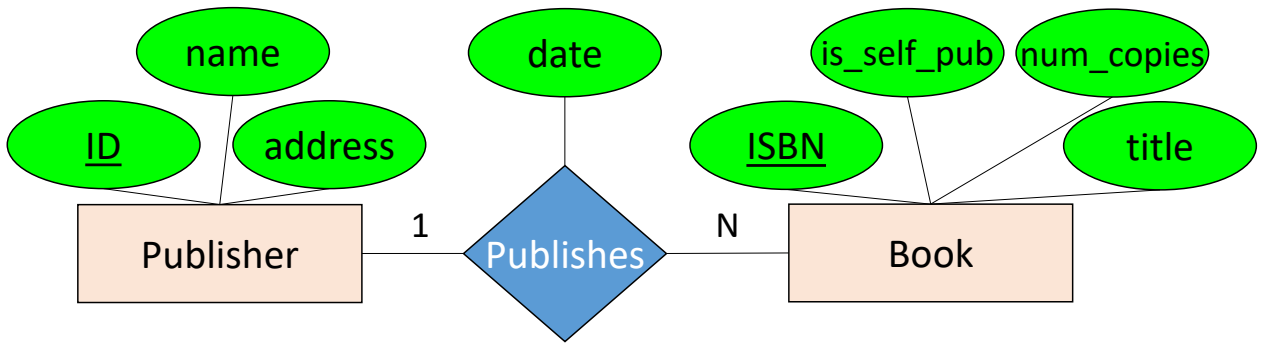
Book

ISBN	title	num_copies	is_self_pub	pub_ID	pub_date

```
CREATE TABLE Publisher (
   …
   PRIMARY KEY (ID)
);
```

```
CREATE TABLE Book (
   …
   PRIMARY KEY (ISBN),
   FOREIGN KEY (pub_ID) REFERENCES Publisher (ID)
);
```

One-to-many / many-to-one – a confusion



Publisher

ID	name	address

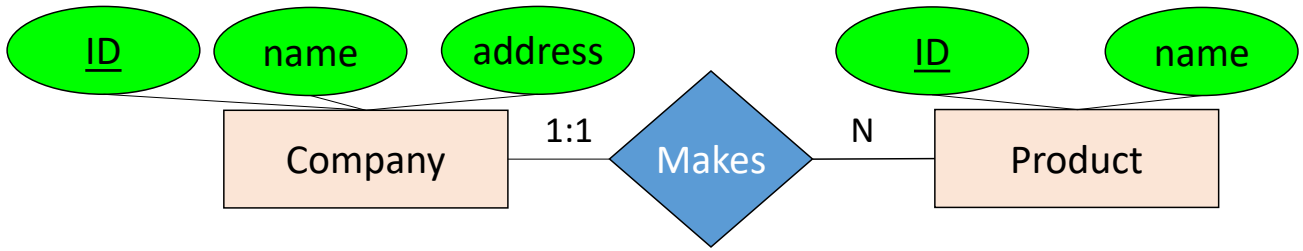
Book

ISBN	title	num_copies	is_self_pub	pub_ID	pub_date

Way to remember where the additional field is placed:

- **Book** has one publisher, so it can be a field.
- **Publisher** has many books, so it can't be a field.

One-to-many with minimum cardinality



Company

ID	name	address

Product

ID	name	co_ID

Companies with no products don't appear.

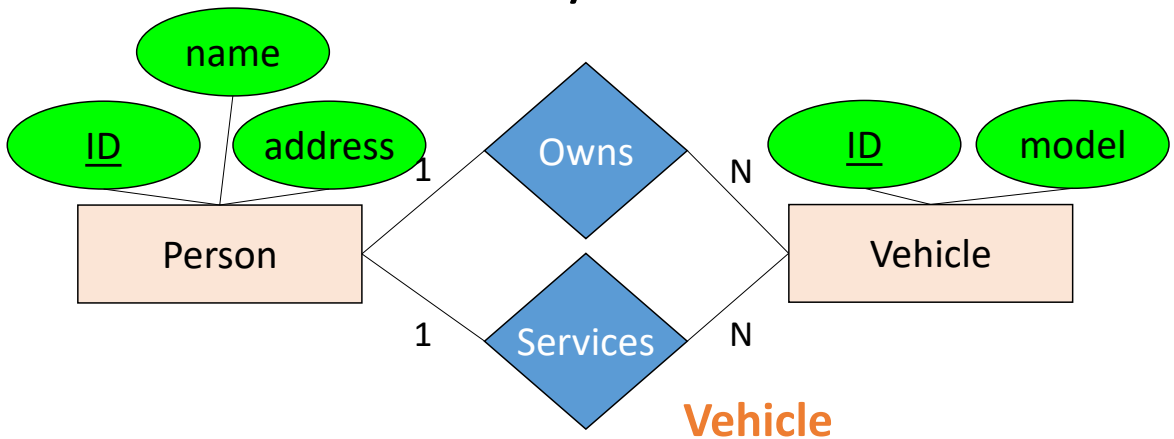
```
CREATE TABLE Company (
   …
   PRIMARY KEY (ID)
);
```

```
CREATE TABLE Product (
   …
   co_ID  … NOT NULL,
   PRIMARY KEY (ID),
   FOREIGN KEY (co_ID) REFERENCES Company (ID)
);
```

Product w/o Company isn't allowed.

FK doesn't imply NOT NULL.

Parallel one-to-many/one relationships



Same idea as before, for each relationship.

Person

ID	name	address

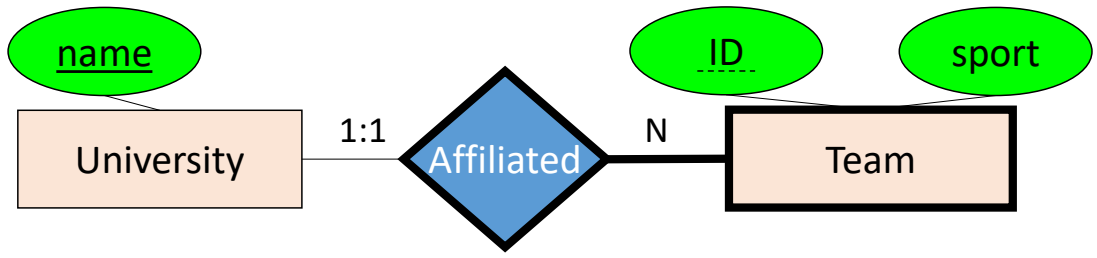
Vehicle

ID	model	owner_ID	mechanic_ID

```
CREATE TABLE Person (
   …
   PRIMARY KEY (ID)
);
```

```
CREATE TABLE Vehicle (
   …
   PRIMARY KEY (ID),
   FOREIGN KEY (owner_ID) REFERENCES Person (ID),
   FOREIGN KEY (mechanic_ID) REFERENCES Person (ID)
);
```

Weak entity



University

name

Team

univ_name	ID	sport

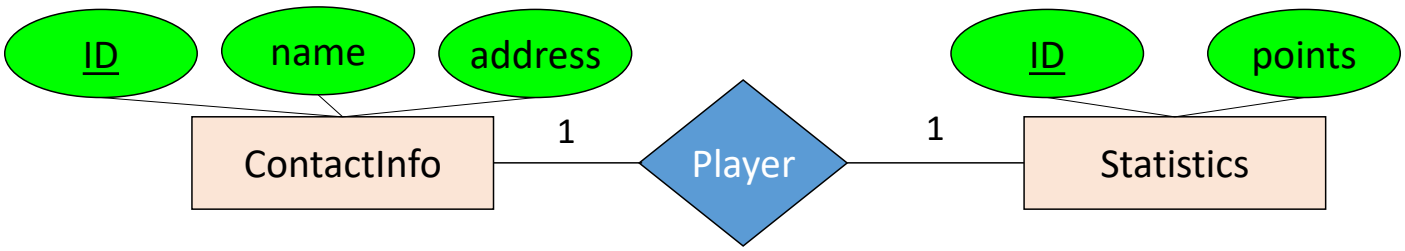
```
CREATE TABLE University (
   …
   PRIMARY KEY (name)
);
```

```
CREATE TABLE Team (
   …
   PRIMARY KEY (univ_name, ID),
   FOREIGN KEY (univ_name) REFERENCES University (name)
);
```

Primary key = keys of both entities. Must have value.

Foreign key = key of related table. Must have value.

One-to-one with semantically same key

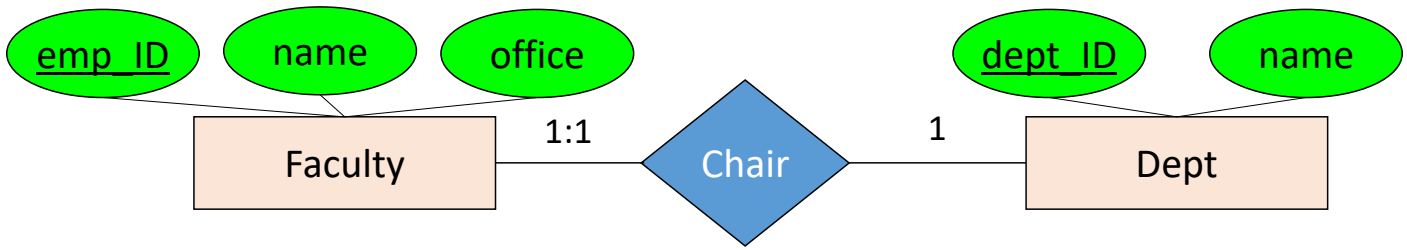


Usually indicates a poor design. Combine into one entity set.

Player

ID	name	address	points

One-to-one with different keys



Sometimes indicates a poor design.

Faculty

emp_ID	name	office	dept_id

Dept

dept_ID	name	chair_id

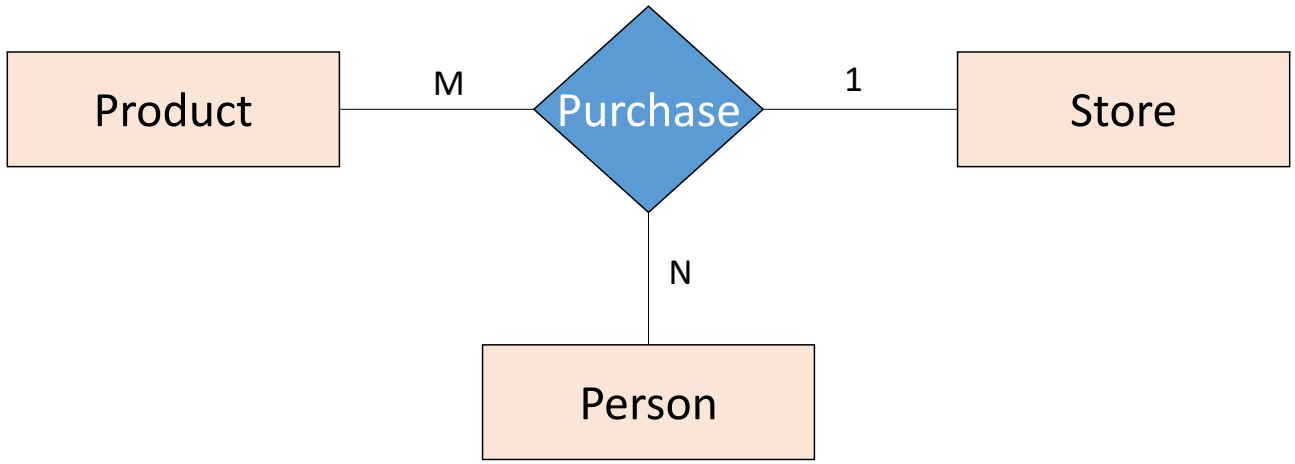
Same optimization as for one-to-many.

```
CREATE TABLE Faculty (
   …
   PRIMARY KEY (emp_ID),
   FOREIGN KEY (dept_ID) REFERENCES Dept (dept_ID)
);
```

```
CREATE TABLE Department (
   …
   chair_id … NOT NULL,
   PRIMARY KEY (dept_ID),
   FOREIGN KEY (chair_ID) REFERENCES Faculty (emp_ID)
);
```

Generalizing to n-ary relationships

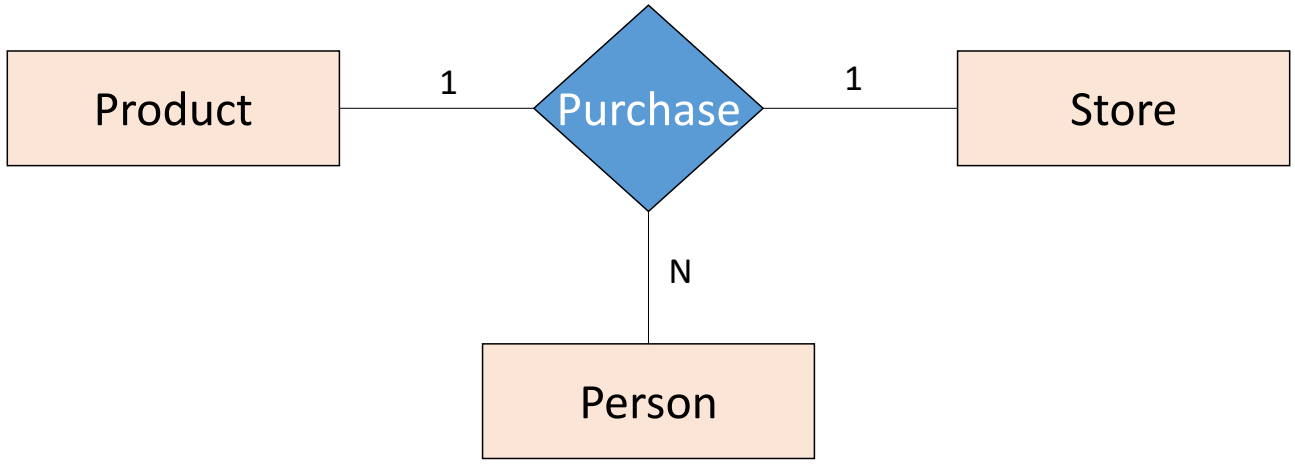
N-ary with multiple “many” sides



Junction table:

- Primary key = combination of PKs of all “many”-side tables
- Foreign keys to all related entity sets’ PKs

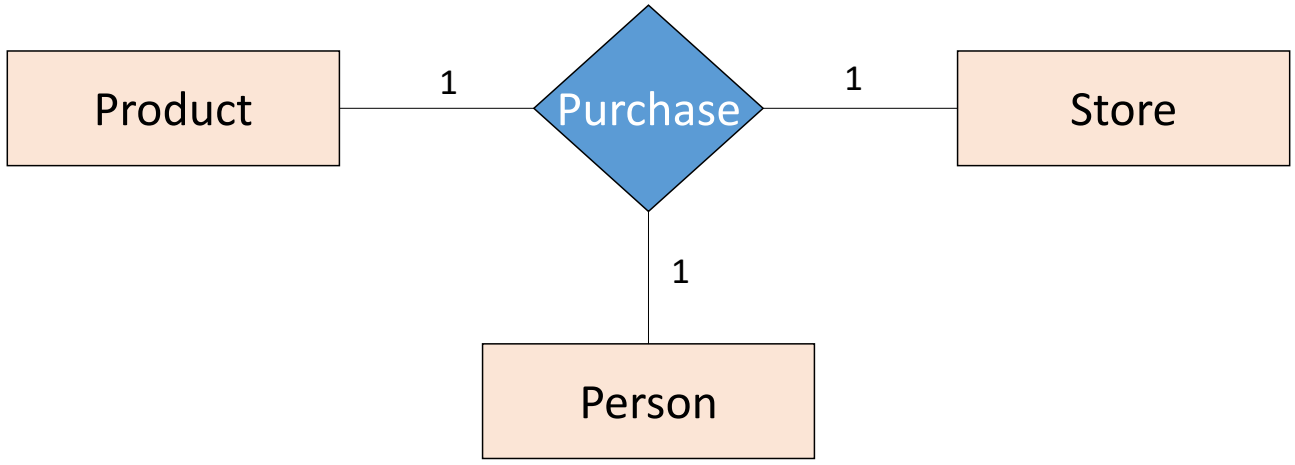
N-ary with only one “many” side



Implemented same as multiple binary one-to-many relationships:

- No junction table.
- “Many”-side table has FKs to other tables.

N-ary with no “many” side



Again, depends on whether tables should be combined or not.

Wrapping This Up

> *A relational database is "basically" a set of tables with values drawn from specified domains, and represented using a relational schema*

I hope that you now appreciate that this technically correct definition basically misses the point

The following captures more of the essence of our discussion

- A relational database consists of
  - A set of tables with identifiers (primary keys)
  - A set of many-to-one binary relationships between tables, each of which is "induced" by foreign-key constraints
  - Each binary relationship represents a function (partial or total) from $table_i \to table_j$

Today's Lecture: Wrapping it Up

From E-R to Relational

Schema Redundancy

Review Of Common Relationship Set E-R $\Rightarrow$ RDB Mappings

Readings

- Today's lecture covered Chapter 6.6 - 6.7 in the textbook
- Note: we didn't discuss "Extended E-R Features" (Chapter 6.8 in the textbook), nor did we discuss "Alternative Notations" (Chapter 6.10)

> - You are not responsible for this material, but as we leave this discussion, keep in mind that we definitely only "scratched the surface"
> - We didn't even get into the topic of E-R design tools ☹
> - At this point, you know enough to get by, and you know enough to be dangerous ☺

- Note: we've focused on binary relationships, and ignored the much more complicated issue of e.g., ternary relationships
  - One reason: not enough time ☹
  - Another reason: ternary (and higher) relationships can be converted into a set of binary relationships