Introduction to Database Systems I2DBS – Spring 2023

- Week 5:
- ER Diagrams
- Translation to SQL DDL

Jorge-Arnulfo Quiané-Ruiz

Readings:

PDBM 3.0-3.3, 6.3-6.4

Information

- **Homework: We will provide feedback ASAP**
  - Solution will be uploaded on LearnIT
  - Also review document on LearnIT
    - common errors for selected queries
- **Exercise 5: ER design and implementation**
  - It is large and comprehensive
  - More than 2 hrs, but use it to practice...
- **Homework 2 opens on Friday**
  - Try to finish pending exercises first

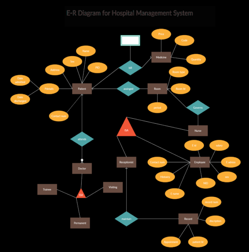
Profile of the Week

IT University of Copenhagen

Peter Chen

Inventor of the Entity-Relationships Model

- **1947:** Born in Taichung, Taiwan
- **1973:** PhD in Computer Science from Harvard University
- **1974-1978:** Assist. Prof. at MIT Sloan School of Management
- **1976:** Published the ER Model
- **1978-1983:** Assoc. Prof. at UCLA
- **1983-2011:** Distinguished Prof. at LSU



Modelling
(ER Diagram)

Entity-Relationships Model

Readings:

PDBM 3.0-3.3, 6.3-6.4

Why Conceptual Model?

- **The "boss" knows she wants a database…**
  … but not what should be in it!

- **Need an effective method to develop the schema and document its structure**

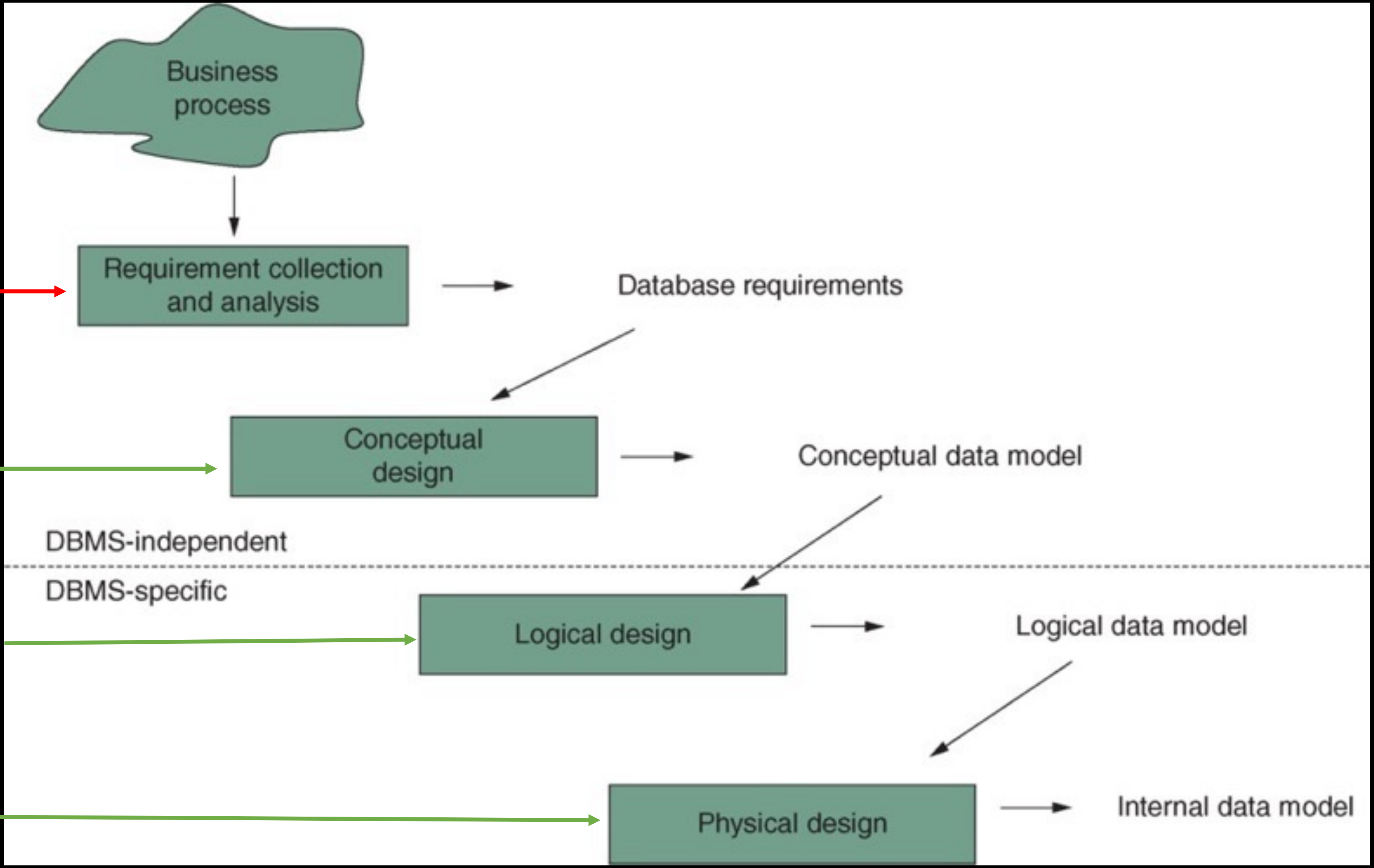
Design Process

Not in this course

Sketch the key components in an ER-diagram

Convert ER-diagram to DDL

Weeks 7 & 8



ER: Entity – Relationships

- **Conceptual Model Defined by Peter Chen (1976)**

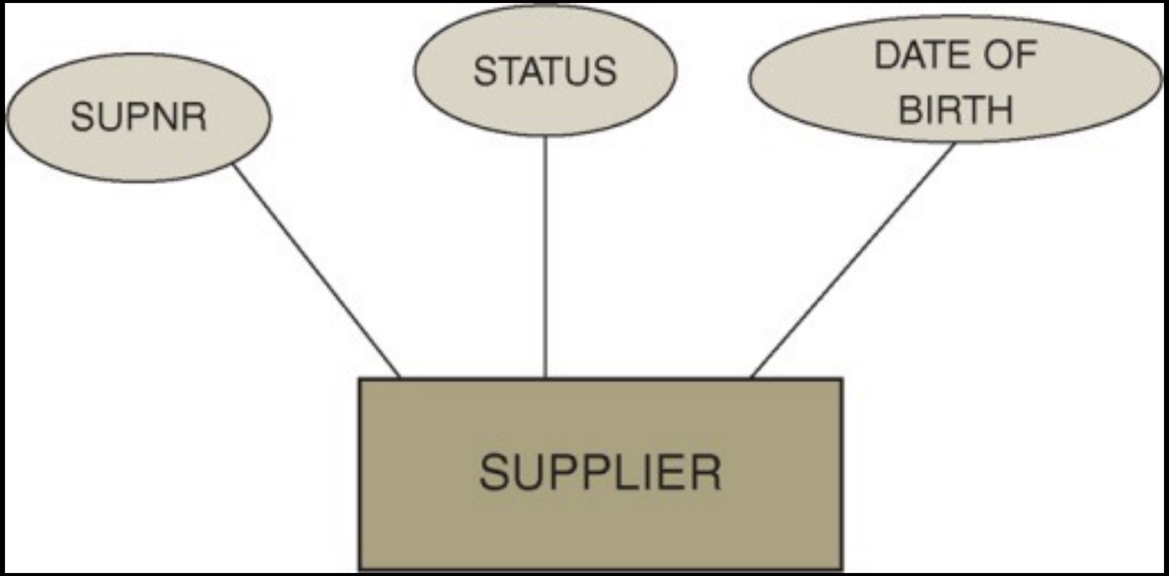
- **ER = modeling concepts + visual representation**
  - ER/EER notation is not standardized
    - Every textbook/company/tool has its own visual representation
  - Core concepts are universally accepted
    - EER vs ER rarely distinguished → we just call it ER!

- **UML can be used as design notation**
  - Slight differences – no UML in this course

- **Focus on ER notation from the book**
  - … plus, some minor extensions – made clear in the lectures
  - In exercises, project, exam: Use the notation in book, lectures

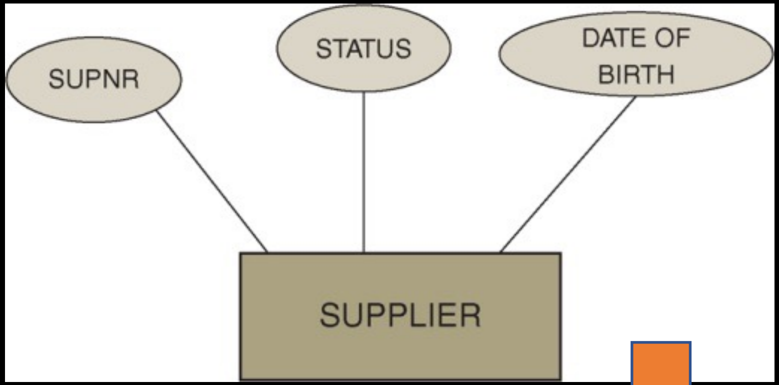
Entity Types and Attribute Types

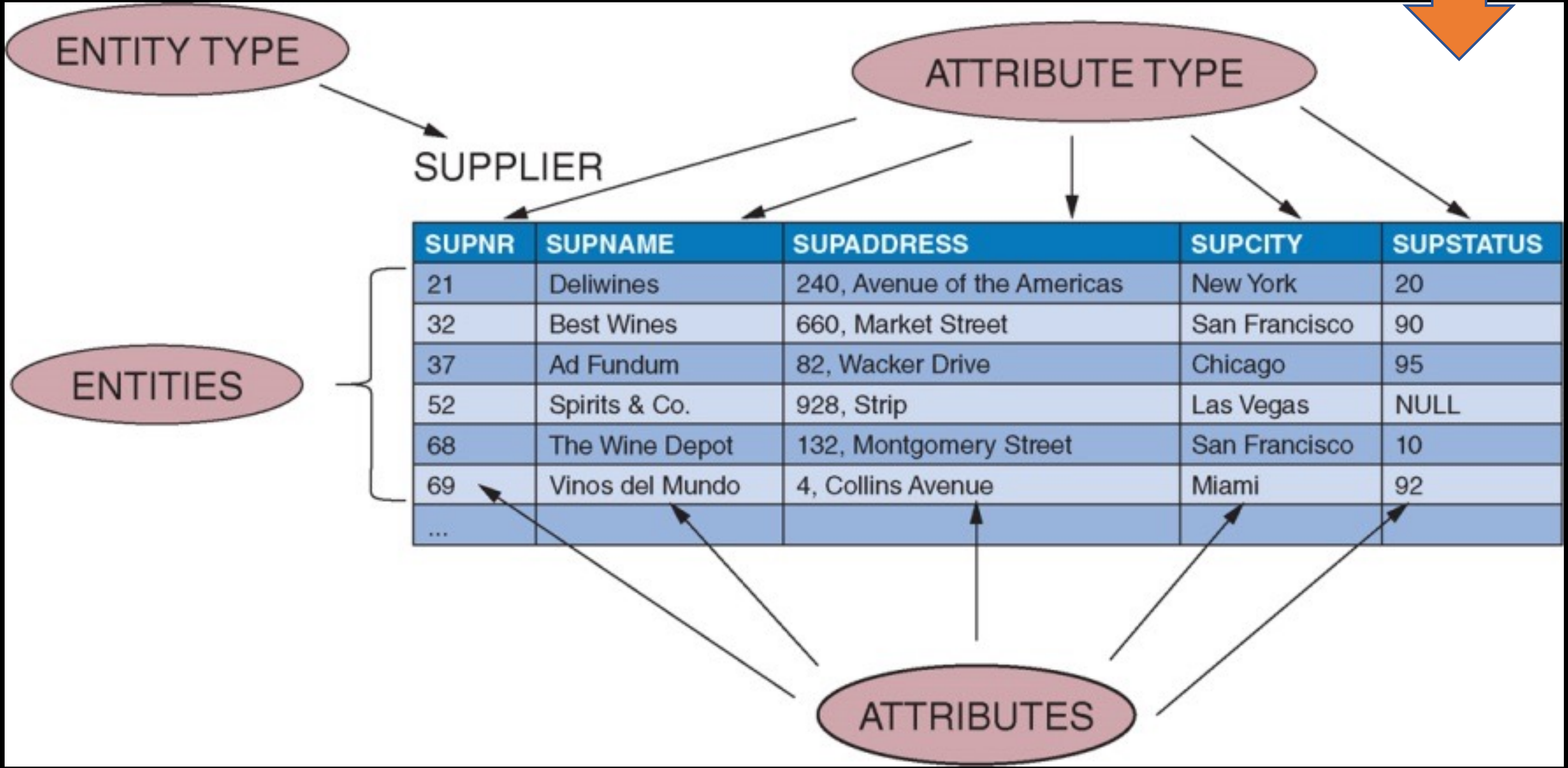
- **Entity Type**
  - Set of similar “things”
  - Ex: students, courses
  - Entity = instance
  - Ex: John, CSE305
- **Attribute Type**
  - Describes one aspect of an entity set
  - Ex: name, maximum enrolment



Entity Type as a Relation

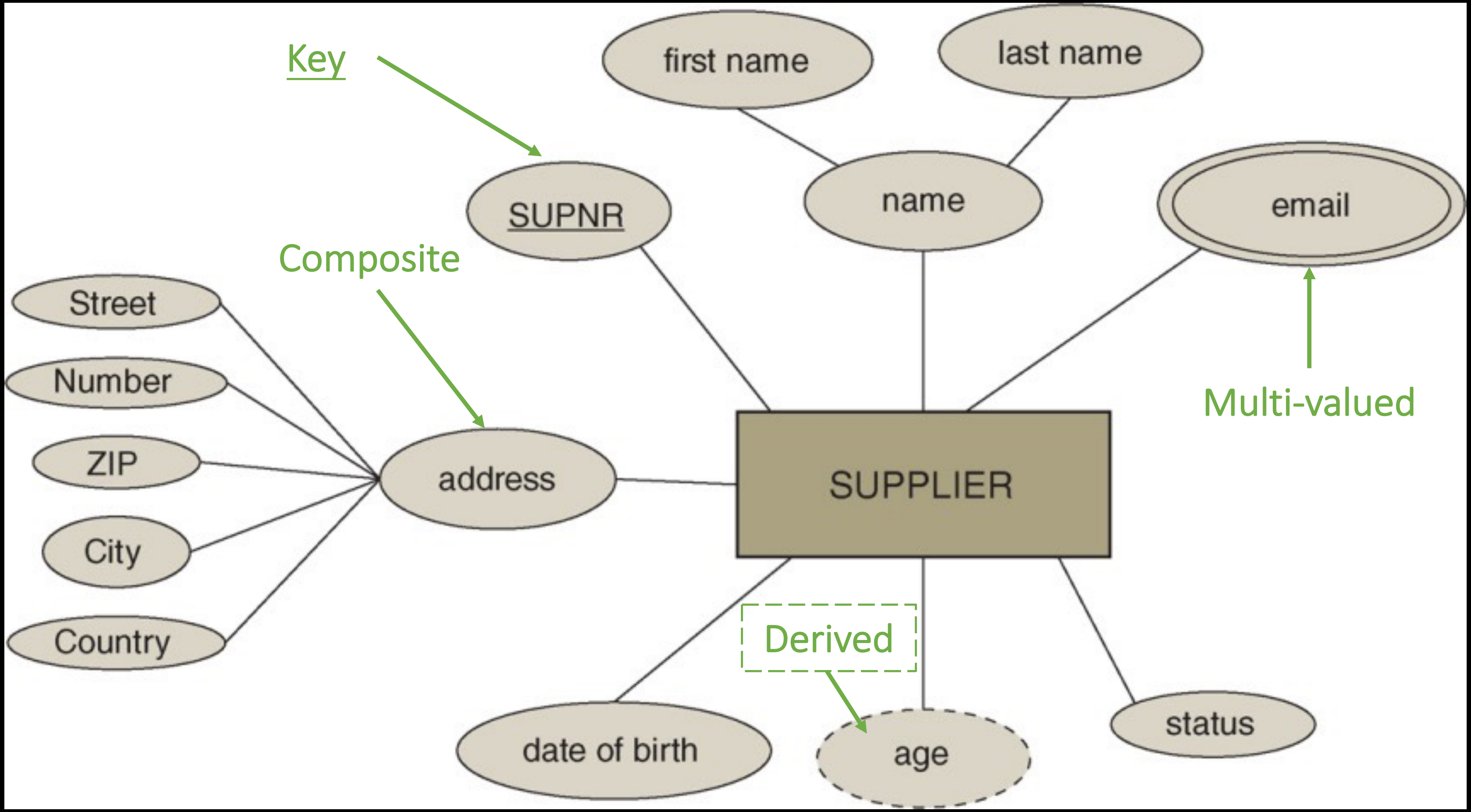
```
CREATE TABLE Supplier (
    SUPNR INT PRIMARY KEY,
    SUPNAME VARCHAR NOT NULL,
    …
);
```





SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS
21	Deliwines	240, Avenue of the Americas	New York	20
32	Best Wines	660, Market Street	San Francisco	90
37	Ad Fundum	82, Wacker Drive	Chicago	95
52	Spirits & Co.	928, Strip	Las Vegas	NULL
68	The Wine Depot	132, Montgomery Street	San Francisco	10
69	Vinos del Mundo	4, Collins Avenue	Miami	92
...				

More Attribute Types



Keys in ER Diagrams and SQL DDL

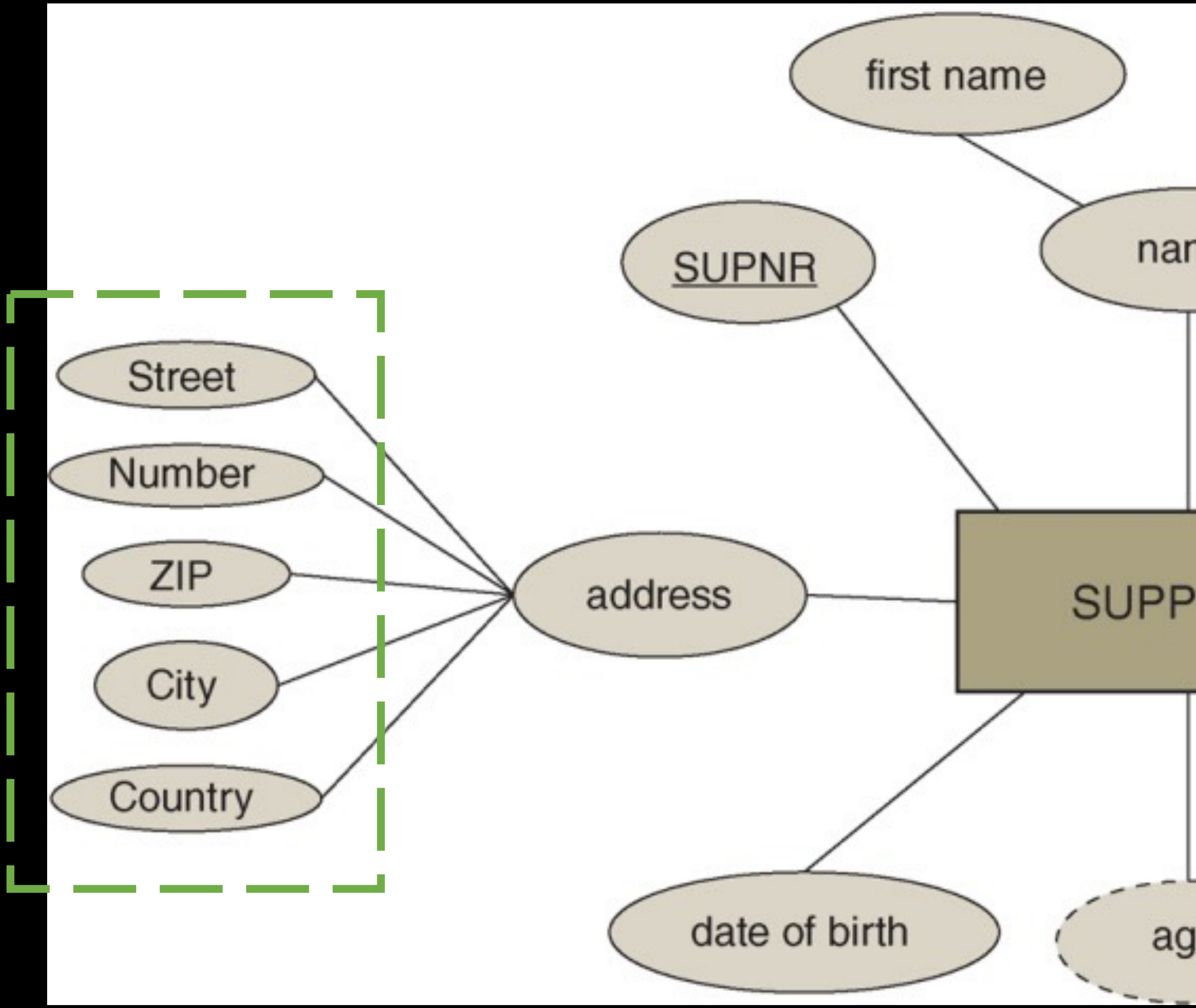
- **Underlined key = PRIMARY KEY**

- **ER diagrams cannot show secondary keys**
  - They must be noted somewhere else!
  - They must still be UNIQUE in the SQL table!

Composite Attributes in SQL DDL

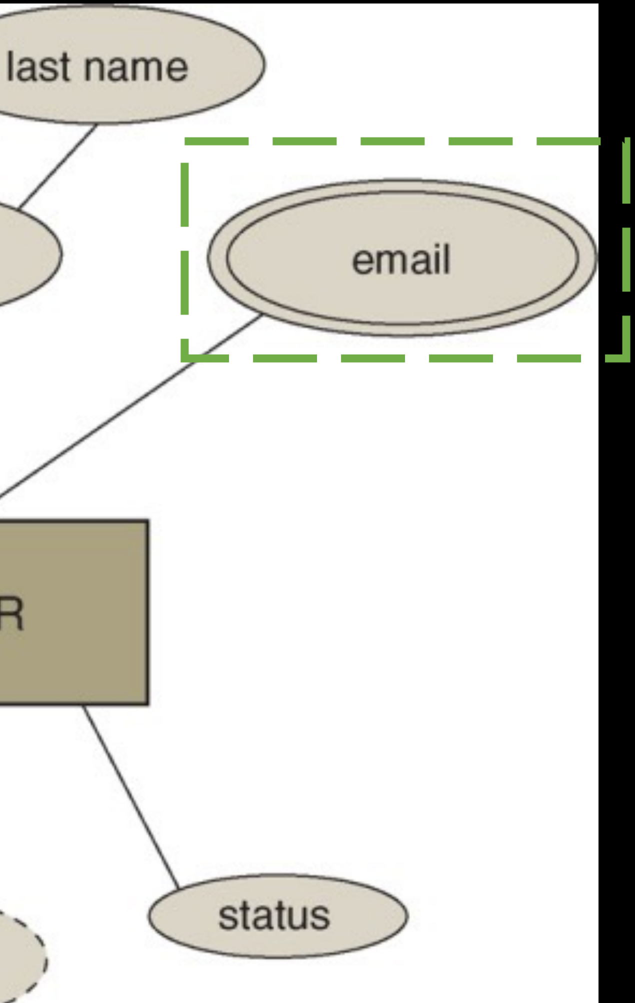
- **Simply use the detailed attributes**

```
CREATE TABLE Supplier  (
    ...
    Street VARCHAR NOT NULL,
    Number INTEGER NOT NULL,
    ZIP INTEGER NOT NULL,
    City VARCHAR NOT NULL,
    Country VARCHAR NOT NULL,
    ...
);
```



Multivalued Attributes in SQL DDL

- **Create a new table referencing the entity table**

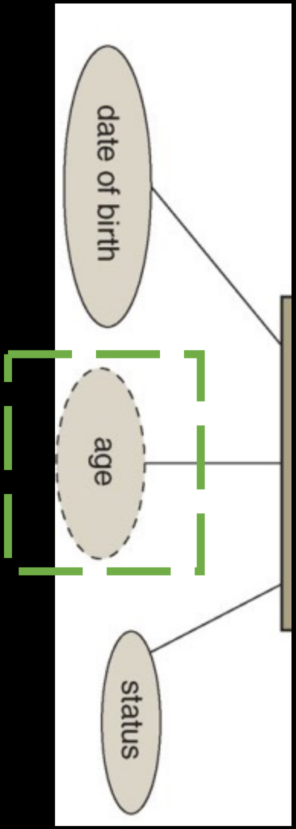


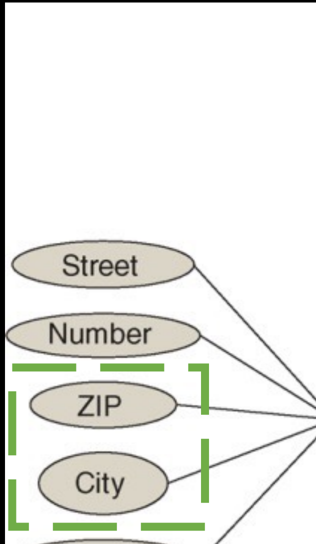
```
CREATE TABLE Emails  (
    Email VARCHAR,
    SupNR INTEGER
        REFERENCES Supplier(SupNR),
    PRIMARY KEY (Email, SupNR)
);
```

Derived Attributes in SQL DDL

- Not discussed in the PDBM book!
- Option 1: Create an attribute and maintain it
  - E.g. with a trigger, or regular update processes
- Option 2: Create a view that computes it
- Neither is very good!

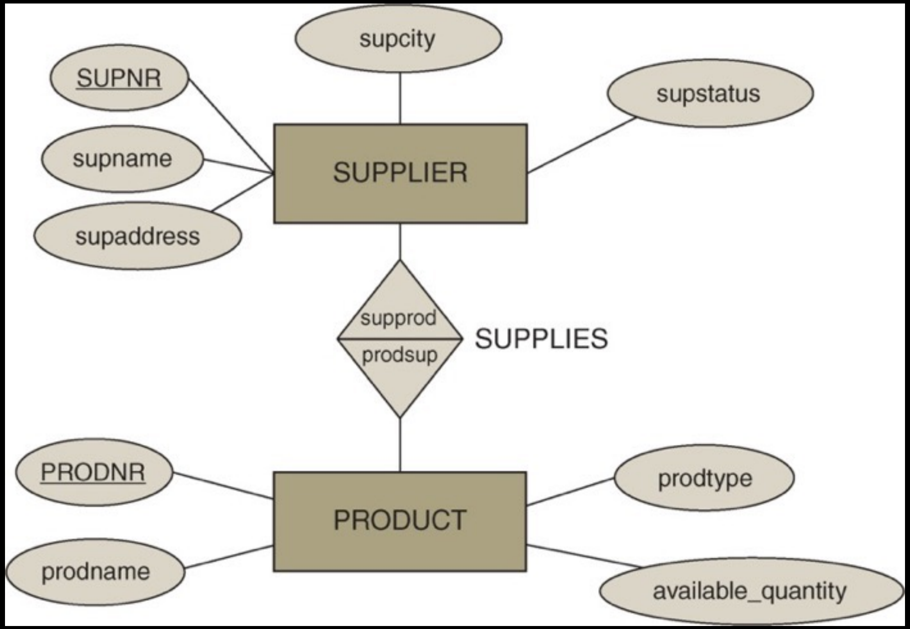
- Sometimes there is a second kind of inter-attribute relationship
  - Here: ZIP ⟶ City
  - Called functional dependencies (FDs)
  - ER diagrams may miss such relationships!
  - We fix this with normalization (week 6)

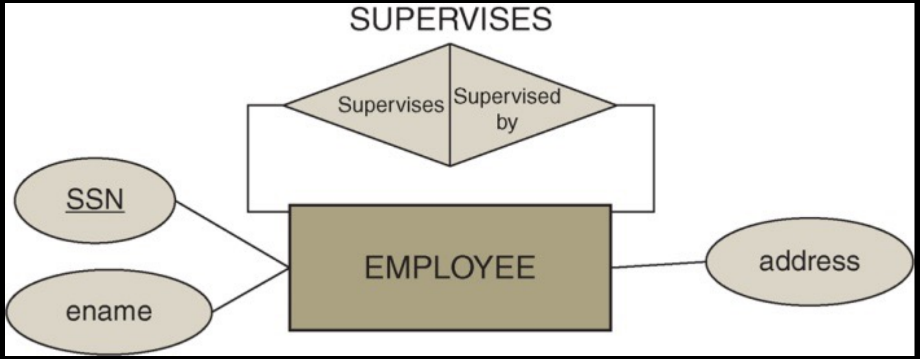




Relationship Types

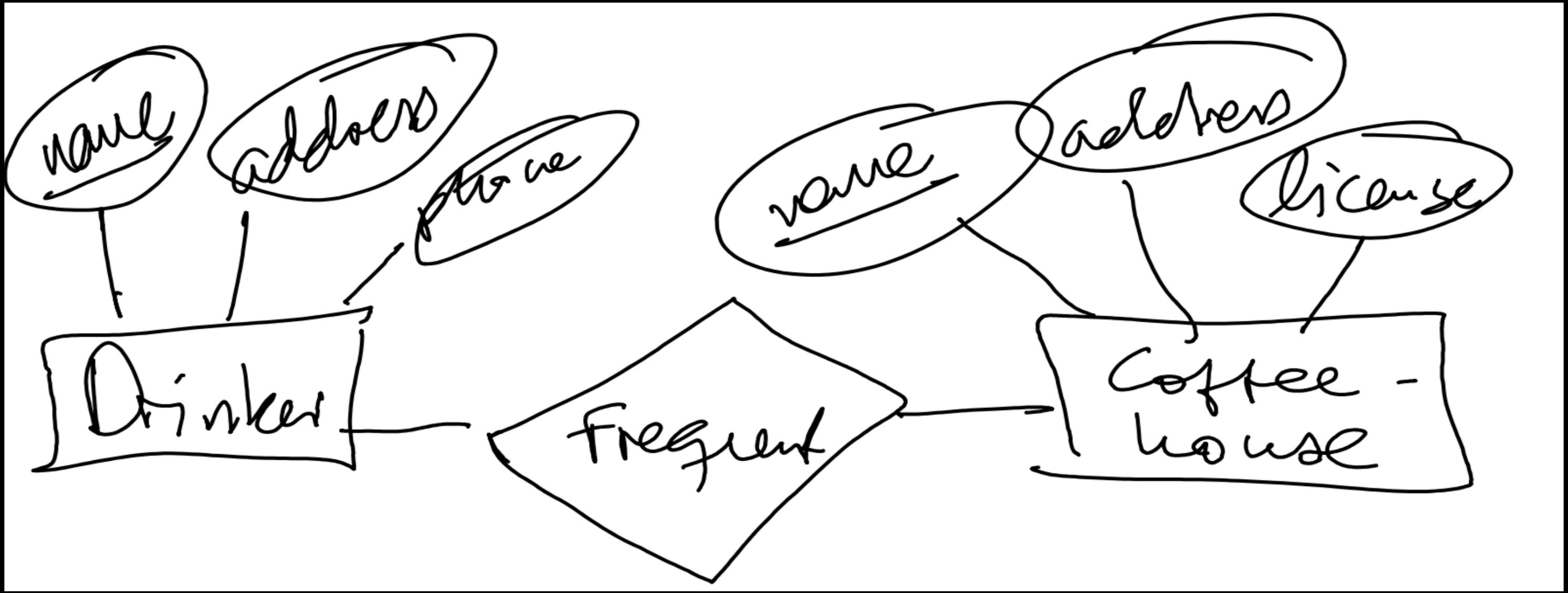
- **Relationship Types**
  - Relate two or more entities (with roles)
  - Ex: John majors in Computer Science
  - Roles may be omitted when obvious





Exercise

- **For drinker, store (unique) name, address, and phone.**
- **For coffeehouses, store (unique) name, address, and license.**
- **Store which drinkers frequent which coffeehouses.**

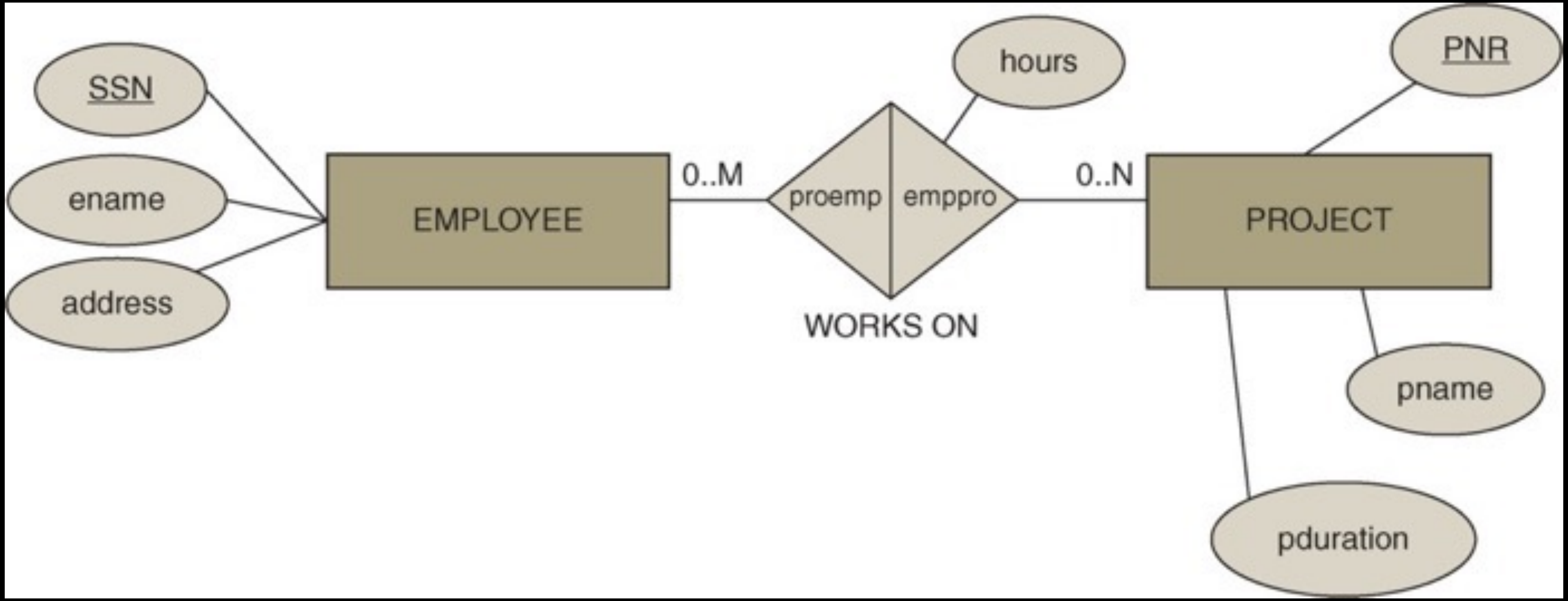


Relation vs. Relationship Type

- **Relation (relational model)**
  - set of tuples
- **Relationship type (ER model)**
  - describes relationship between entities of an enterprise
- **Both entity types and relationship types (from an ER model) will be represented by relations (in the relational model)**

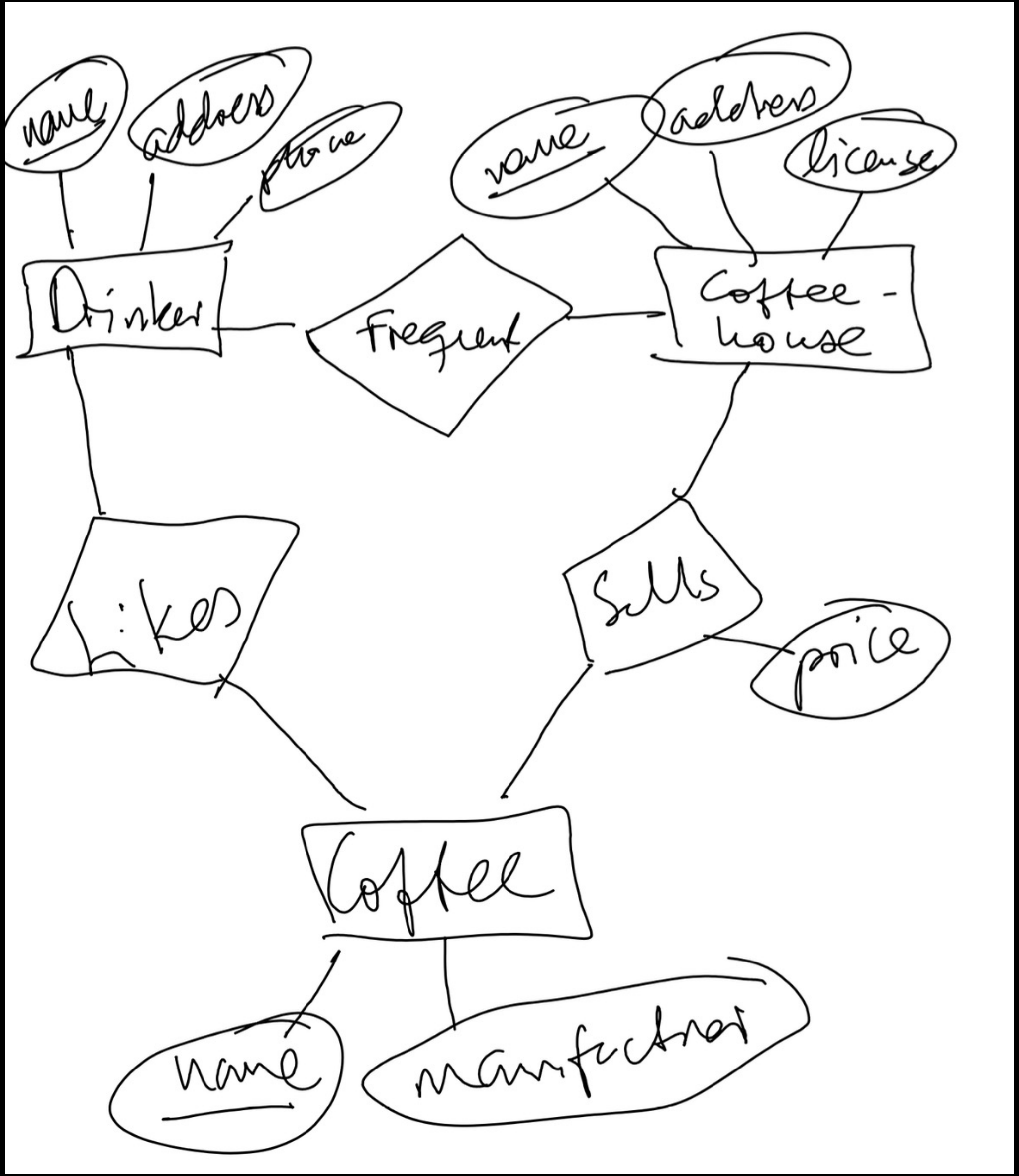
Relationship Attribute Types

- **Relationships may also have attributes**

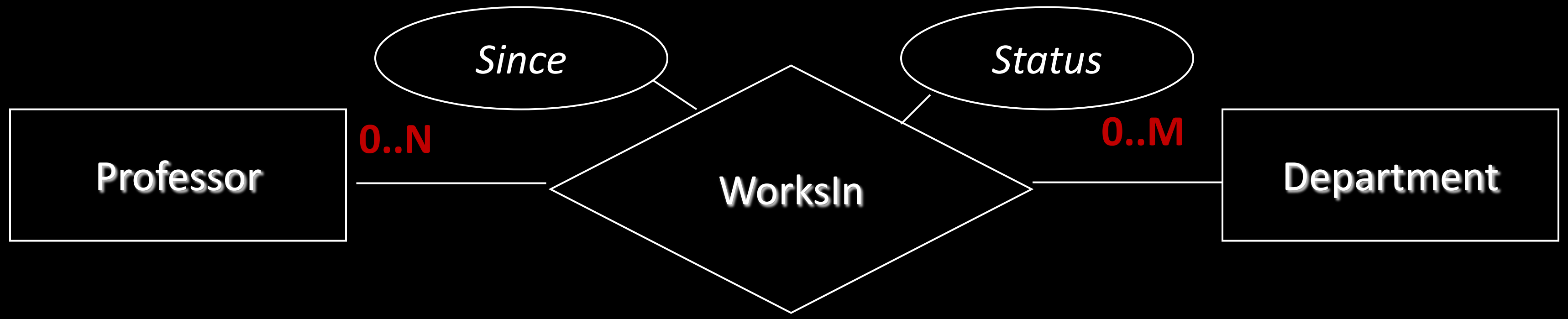


Example: Coffee Database

- **What if …**
  - price is an attribute of Coffee?
  - price is an attribute of Coffeehouse?



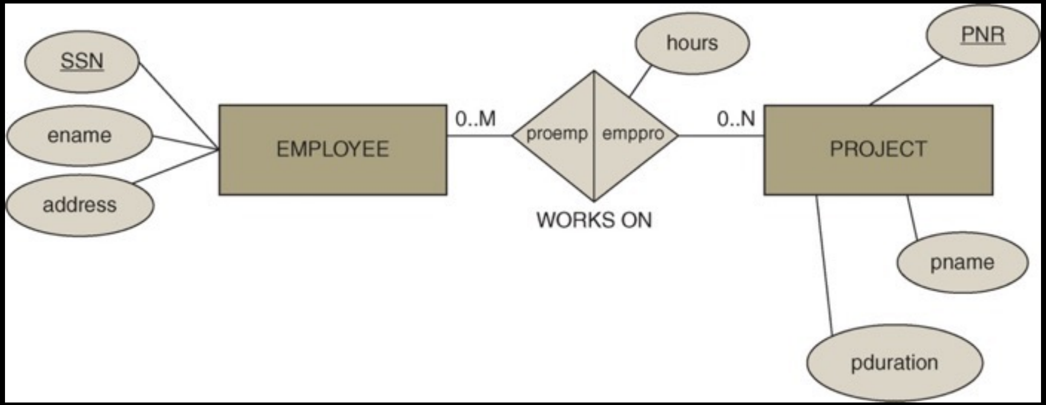
Basic Relationship Table in SQL DDL

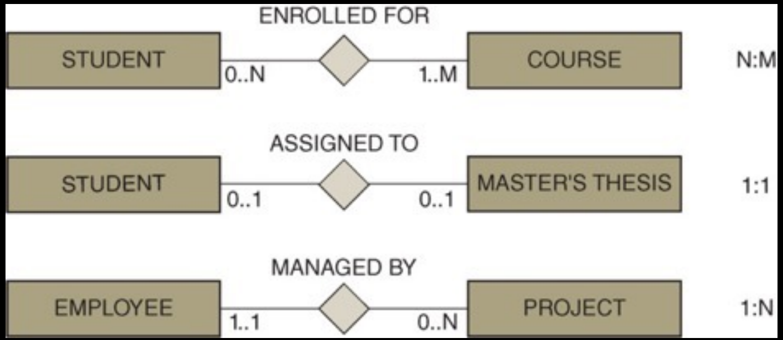


```
CREATE TABLE WorksIn  (
    ProfID  INTEGER,                        -- role (key of Professor)
    DeptID  CHAR (4),                       -- role (key of Department)
    Since DATE NOT NULL,                    -- attribute
    Status CHAR (10) NOT NULL,              -- attribute
    PRIMARY KEY (ProfID, DeptID),
    FOREIGN KEY (ProfID) REFERENCES Professor (ID),
    FOREIGN KEY (DeptID) REFERENCES Department (ID)
)
```

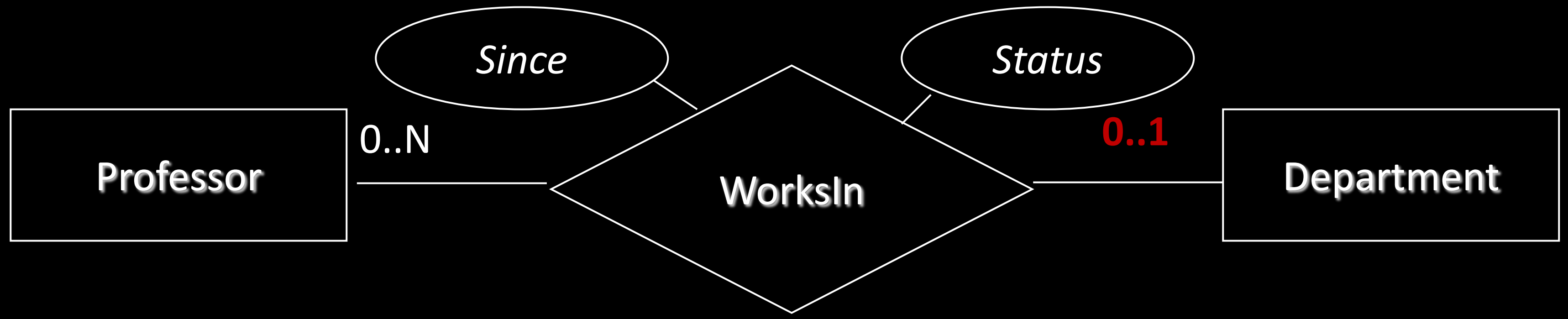
Cardinalities

- **Relationships always have cardinalities**
  - Minimum: 0 or 1
  - Maximum: 1 or N / M / L / * / …
- **Read: Entity (ignore) Relationship Cardinality Entity**
  - Student can enrol for 1 to M courses
  - Project is managed by exactly 1 employee
- **Do cardinalities impact the resulting table structure?**



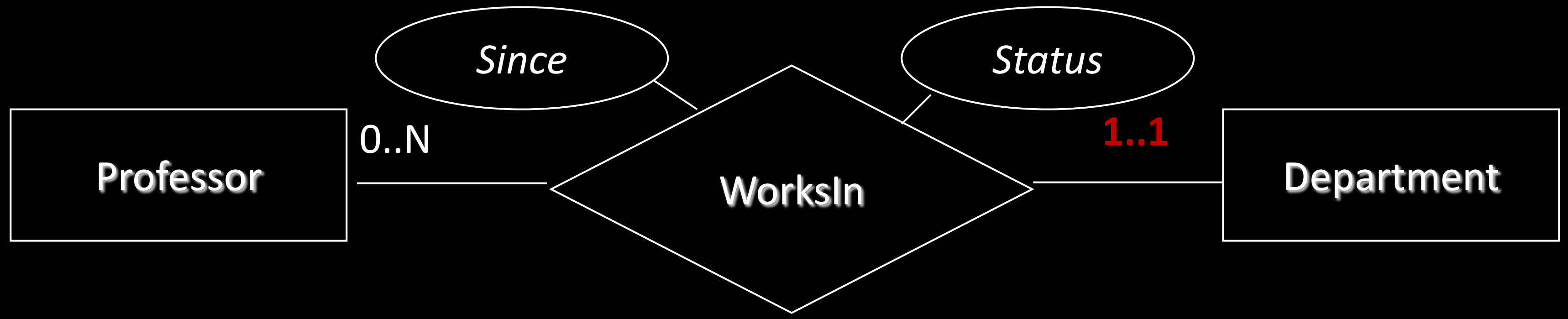


Maximum 1



```
CREATE TABLE WorksIn  (
    ProfID  INTEGER,              -- role (key of Professor)
    DeptID  CHAR (4) NOT NULL,  -- role (key of Department)
    Since DATE NOT NULL,        -- attribute
    Status  CHAR (10) NOT NULL, -- attribute
    PRIMARY KEY (ProfID),       -- each professor only once
    FOREIGN KEY (ProfID) REFERENCES Professor (ID),
    FOREIGN KEY (DeptID) REFERENCES Department (ID)
)
```

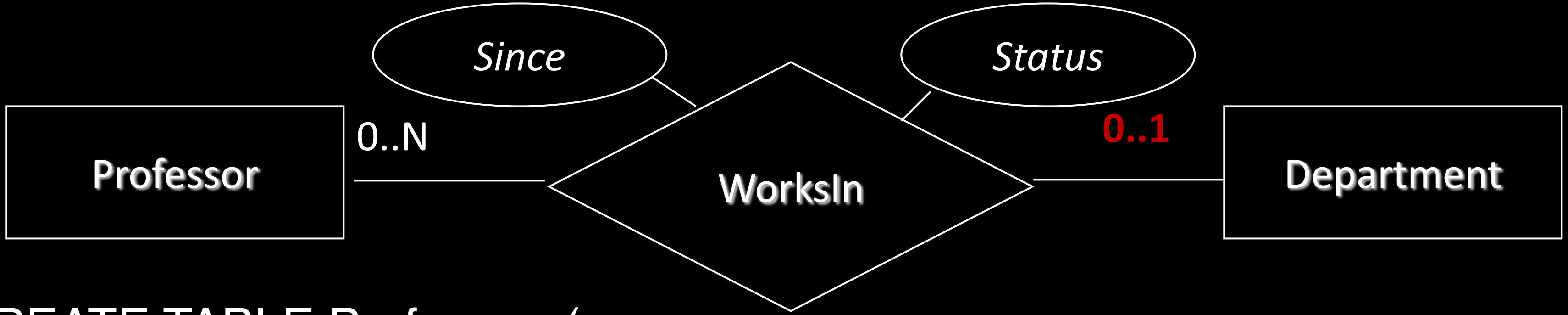
Exactly 1



```
~~CREATE TABLE WorksIN~~

CREATE TABLE Professor  (
    Id INTEGER PRIMARY KEY,
    ...
    deptId INTEGER NOT NULL,     -- foreign key
    Since DATE NOT NULL,         -- attribute
    Status  CHAR (10) NOT NULL, -- attribute
    FOREIGN KEY (deptId) REFERENCES Department(Id)
)
```

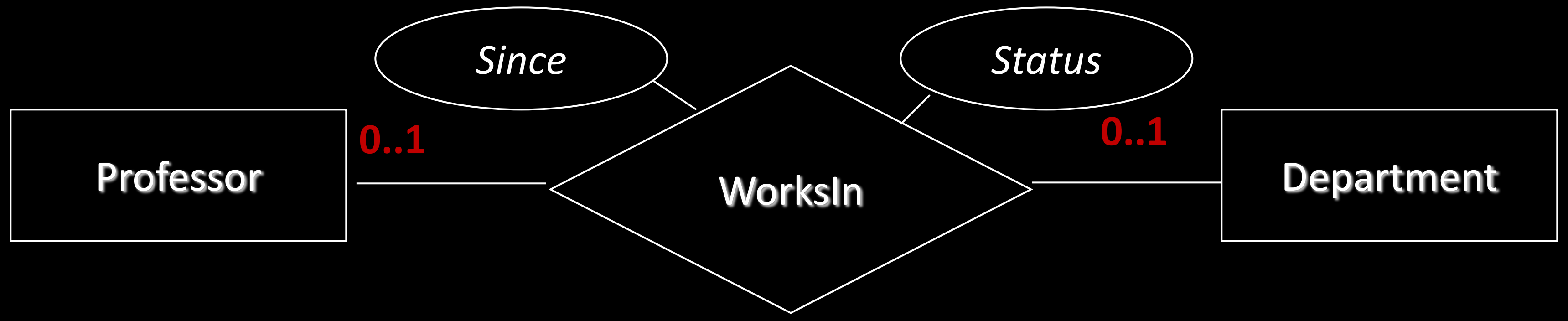
Maximum 1 – revisited

- **Could we do the same with 0..1?**
  - Yes, but three attributes must all be NULL or not NULL
  - We only do this if the relationship has NO attributes!!



```
CREATE TABLE Professor  (
    Id INTEGER PRIMARY KEY,
    ...
    ~~deptId INTEGER NULL,     -- foreign key~~
    ~~Since DATE NULL,          -- attribute~~
    ~~Status  CHAR (10) NULL, -- attribute~~
    ~~FOREIGN KEY (deptId) REFERENCES Department(Id)~~
)
```

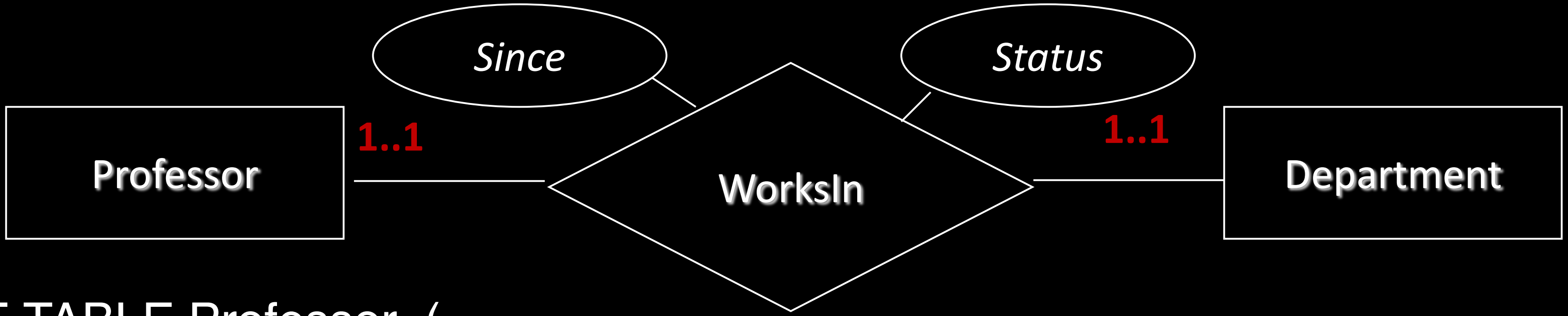
Maximum 1 – in both directions!



```
CREATE TABLE WorksIn  (
    ProfID  INTEGER,              -- role (key of Professor)
    DeptID  CHAR (4) NOT NULL,  -- role (key of Department)
    Since DATE NOT NULL,         -- attribute
    Status  CHAR (10) NOT NULL, -- attribute
    PRIMARY KEY (ProfID),        -- each professor only once
    UNIQUE (deptID),             -- each department only once
    FOREIGN KEY (ProfID) REFERENCES Professor (ID),
    FOREIGN KEY (DeptID) REFERENCES Department (ID)
)
```

Exactly 1 – in both directions!

- **Can we use FKs on both sides?**
  - Yes… but it is neither easy nor portable



```
CREATE TABLE Professor  (
   Id INTEGER PRIMARY KEY,
   ...
   deptId INTEGER NOT NULL,
   Since DATE NOT NULL,
   Status  CHAR (10) NOT NULL,
   FOREIGN KEY (deptId)
   REFERENCES Department(Id)
)
```

```
CREATE TABLE Department  (
   Id INTEGER PRIMARY KEY,
   ...
   profId INTEGER NOT NULL,
   FOREIGN KEY (profId)
   REFERENCES Professor(Id)
)
```

Exactly 1 – in both directions!

- **Can we use FKs on both sides?**
  - Yes… but it is neither easy nor portable
- **Think about inserting the first prof and dept**
  - Which comes first, the chicken or the egg?
- **Alternative 1: Deferred FK or trigger**
  - Runs at the end of a transaction – many systems support neither!

```
CREATE TABLE Professor  (
    Id INTEGER PRIMARY KEY,
    ...
    deptId INTEGER NOT NULL,
    Since DATE NOT NULL,
    Status  CHAR (10) NOT NULL,
    FOREIGN KEY (deptId)
    REFERENCES Department(Id)
)
```

```
CREATE TABLE Department  (
    Id INTEGER PRIMARY KEY,
    ...
    profId INTEGER NOT NULL,
    FOREIGN KEY (profId)
    REFERENCES Professor(Id)
)
```

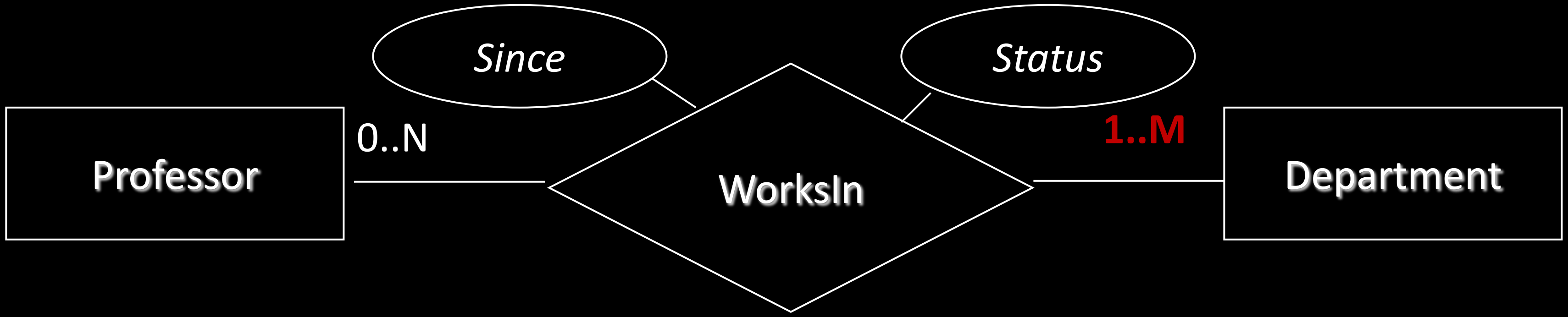
Exactly 1 – in both directions!

- **Alternative 2:**
  Merge tables
  - May work well in some cases
  - Depends on the entities
- **Alternative 3:**
  Pick one FK direction
  - Write down the other requirement
  - Do the best we can in software with the other direction

```
CREATE TABLE Professor  (
    Id INTEGER PRIMARY KEY,
    ...
    deptId INTEGER NOT NULL,
    deptName VARCHAR NOT NULL,
    ...
    Since DATE NOT NULL,
    Status  CHAR (10) NOT NULL,
)
```

Minimum 1 – Maximum N

- **What about 1..N or 1..M cardinalities?**
  - Or when requirements demand particular fixed numbers?
- **No support in SQL DDL**
  - Could use trigger code on Professor/WorksIn
- **Normally:**
  - Write down the requirement
  - Do the best we can in software



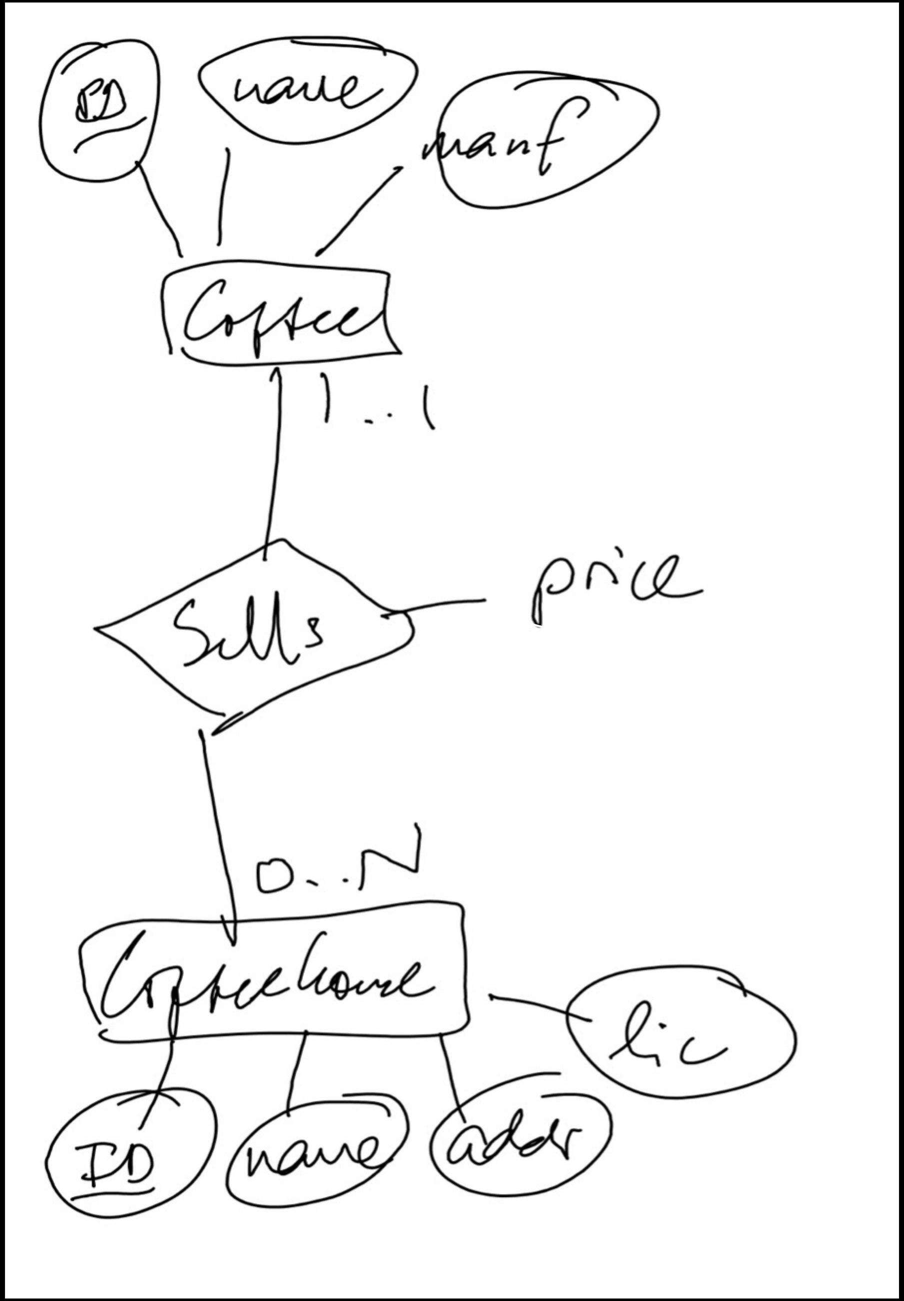
Exercise

- **Draw this schema as ER diagram … but:**
  - Use IDs
  - Assume each coffeehouse sells exactly one coffee
- **Write SQL DDL to create the tables**
  - How many tables?

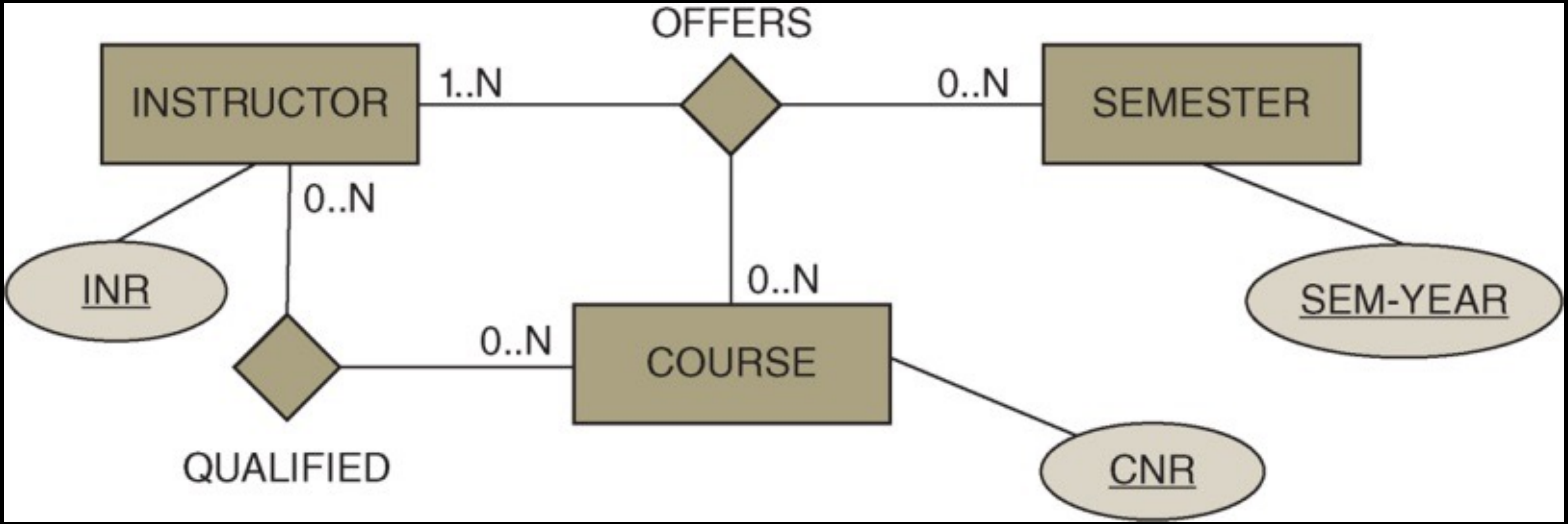
Coffees(name, manf)
Coffeehouses(name, addr, license)
Sells(coffeehouse, coffee, price)

Answer

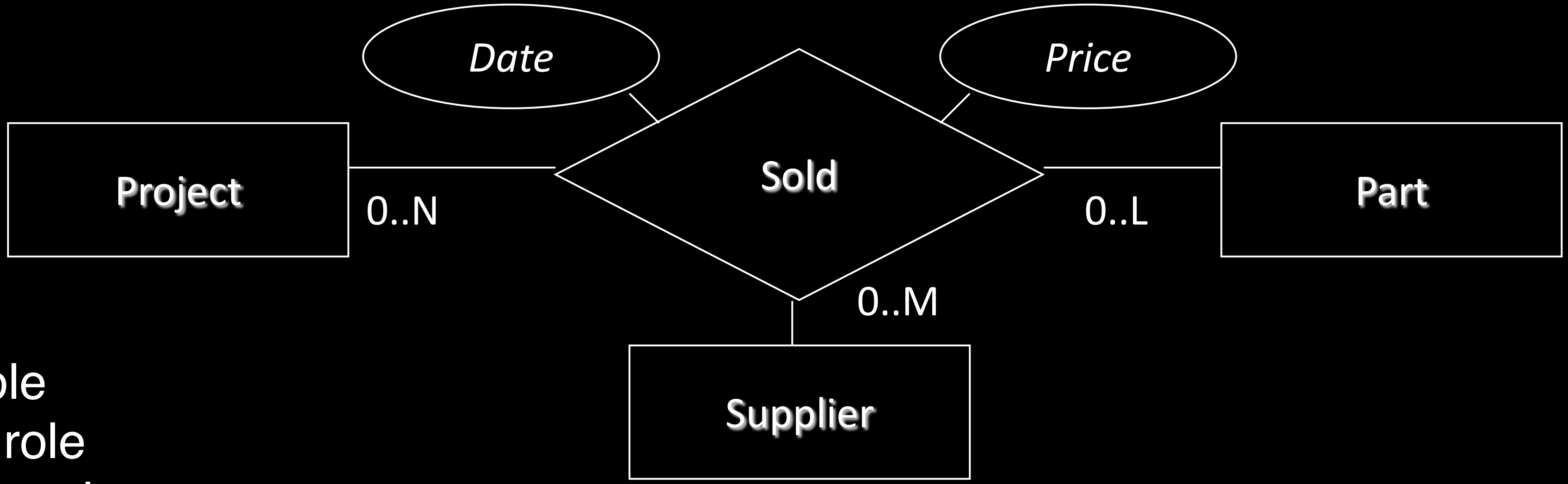
```
CREATE TABLE Coffee (
  ID INTEGER PRIMARY KEY,
  name VARCHAR NOT NULL,
  manf VARCHAR NOT NULL
);

CREATE TABLE Coffeehouse (
  ID INTEGER PRIMARY KEY,
  name VARCHAR NOT NULL,
  addr VARCHAR NOT NULL,
  lic VARCHAR NOT NULL,
  coffeeID INTEGER NOT NULL
        REFERENCES Coffee(ID),  price INTEGER NOT NULL
);
```



Tertiary Relationships (and Beyond)

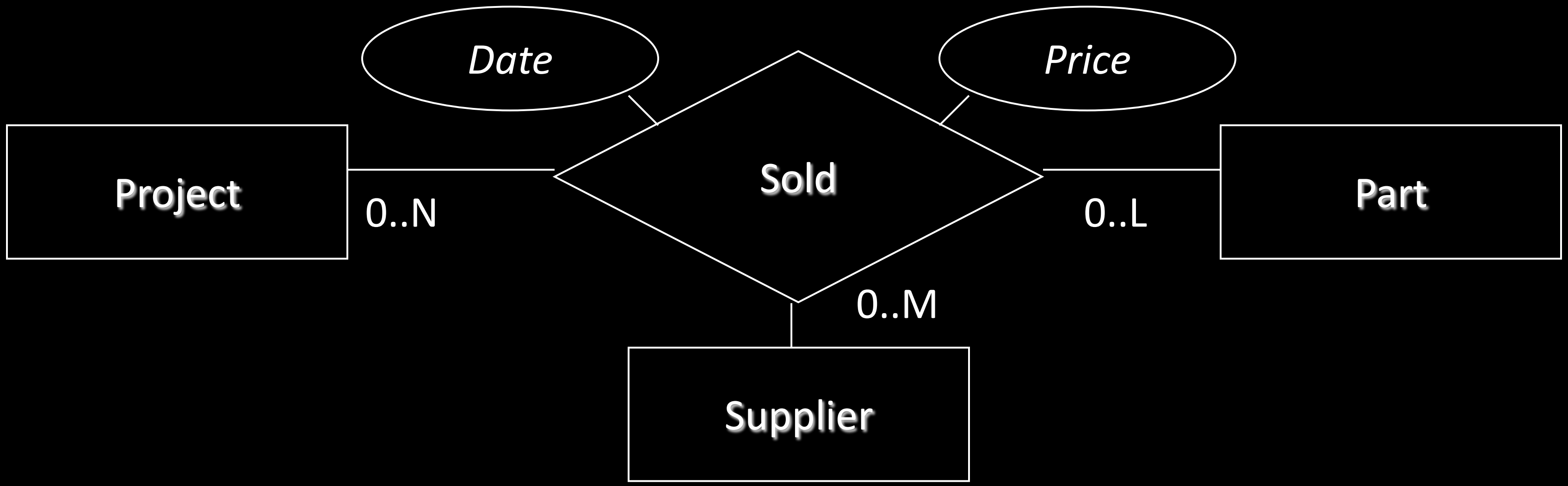


Example Tertiary Relationship Type



```
CREATE TABLE  Sold  (
    ProjID  INTEGER,             -- role
    SupplierID  INTEGER,         -- role
    PartNumber  INTEGER,         -- role
    Date  DATE NOT NULL,         -- attribute
    Price  INTEGER NOT NULL      -- attribute
    PRIMARY KEY (ProjID, SupplierID, PartNumber),
    FOREIGN KEY (ProjID) REFERENCES Project (ID),
    FOREIGN KEY (SupplierID) REFERENCES Supplier (ID),
    FOREIGN KEY (PartNumber) REFERENCES Part
(Number)
)
```

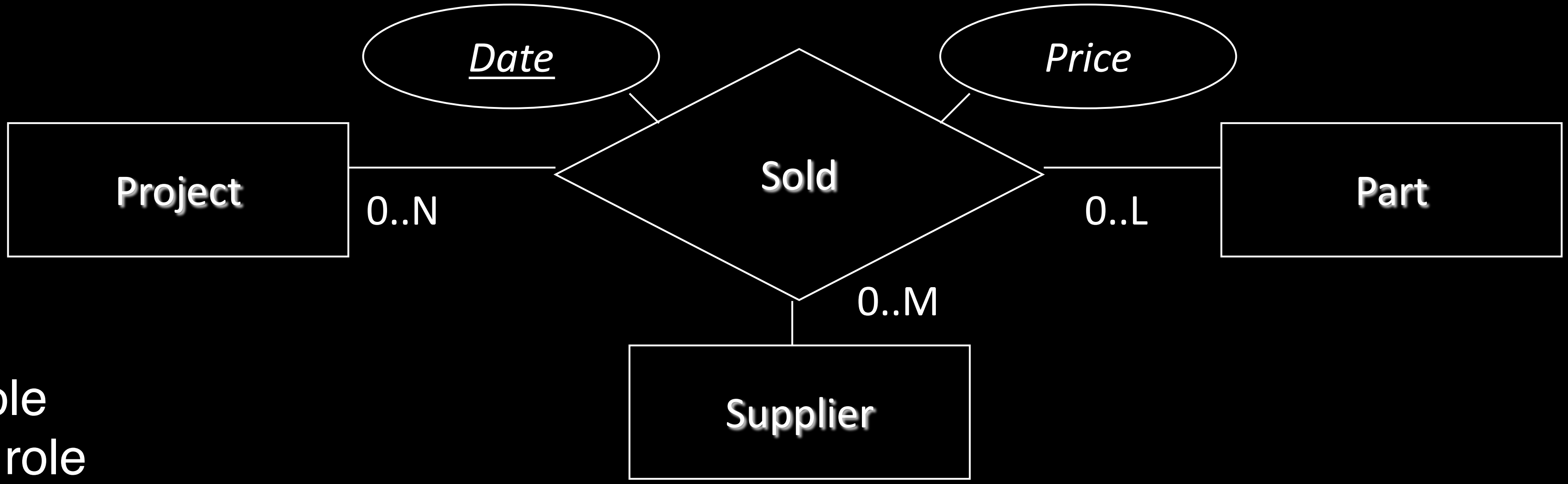
Extension: Partial Relationship Keys



PRIMARY KEY (ProjID, SupplierID, PartNumber)

- **What does the key of the relationship table mean?**
- **What if the part should be sold many times?**
- **The book: No discussion!**
- **Our notation: Partial relationship key**
  - Attribute is underlined

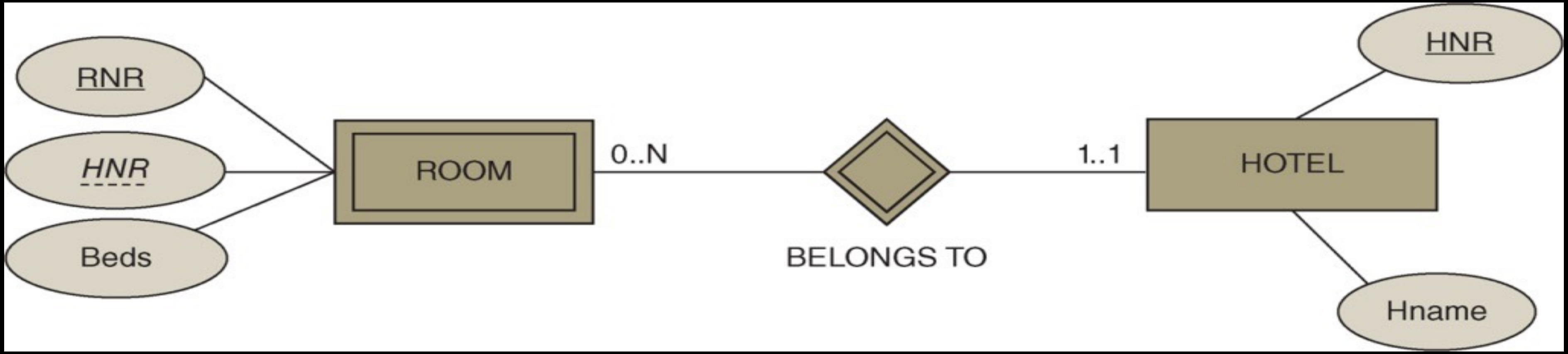
Partial Relationship Keys in SQL DDL



```
CREATE TABLE  Sold  (
   ProjID  INTEGER,              -- role
   SupplierID  INTEGER,          -- role
   PartNumber  INTEGER,          -- role
   Date  DATE,                   -- attribute
   Price  INTEGER NOT NULL,      -- attribute
   PRIMARY KEY (ProjID, SupplierID, PartNumber, Date),
   FOREIGN KEY (ProjID) REFERENCES Project (ID),
   FOREIGN KEY (SupplierID) REFERENCES Supplier (ID),
   FOREIGN KEY (PartNumber) REFERENCES Part
(Number)
)
```

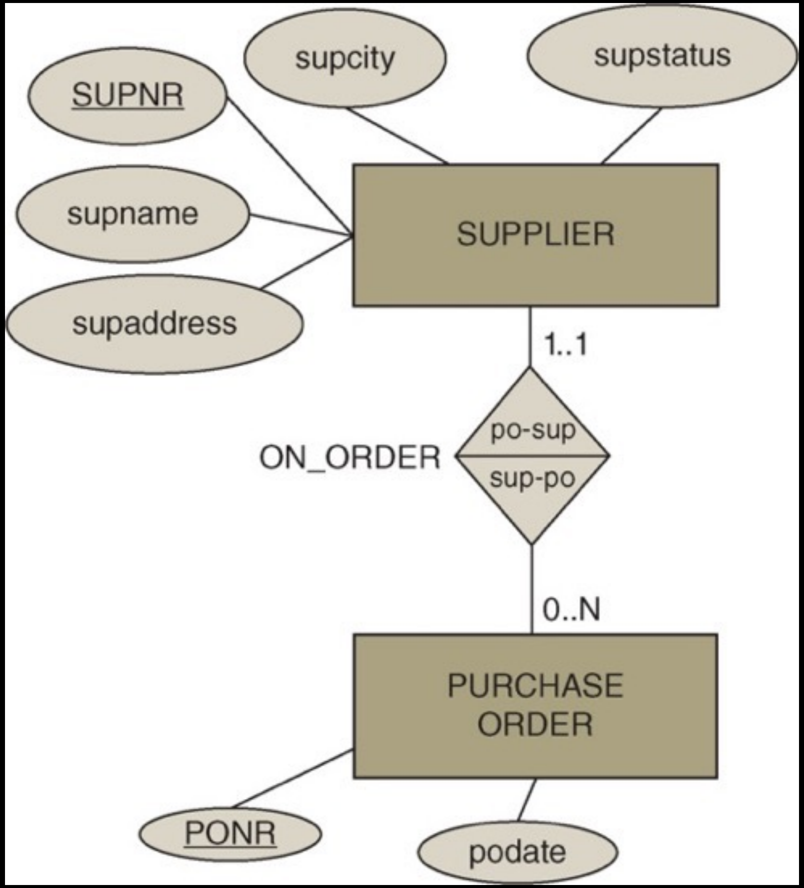
Weak Entity Types

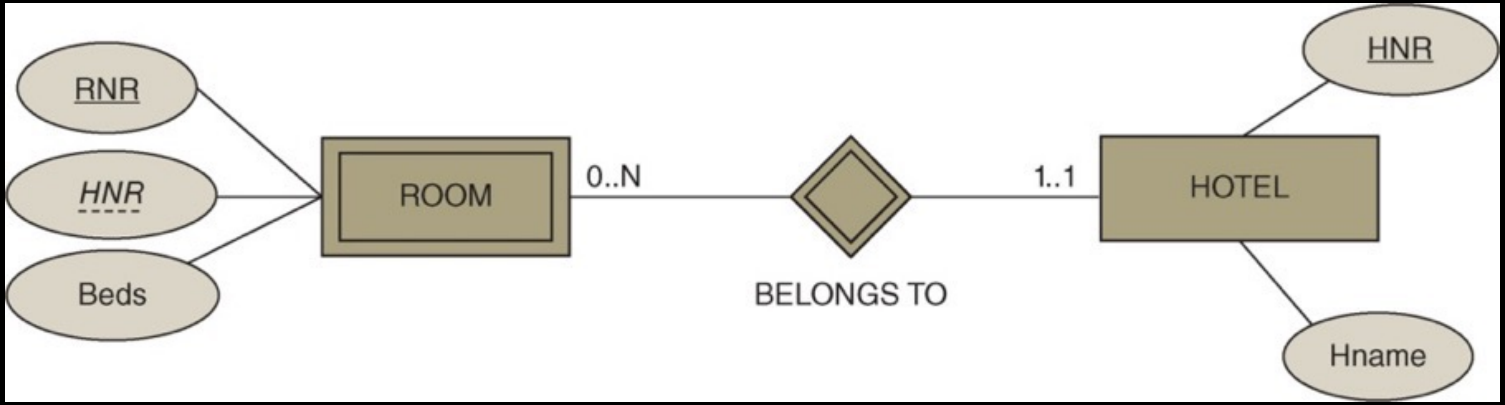
- **Weak entities belong to another entity**
  - They do not have a proper key, but include “parent” key
  - They have a 1 .. 1 participation in the relationship
  - If “parent” is deleted, so is the “child”
- **Representation**
  - Double outlines (entity, relationship)
  - Parent key is underlined with dashes



Weak Entities vs 1..1 Relationship Types

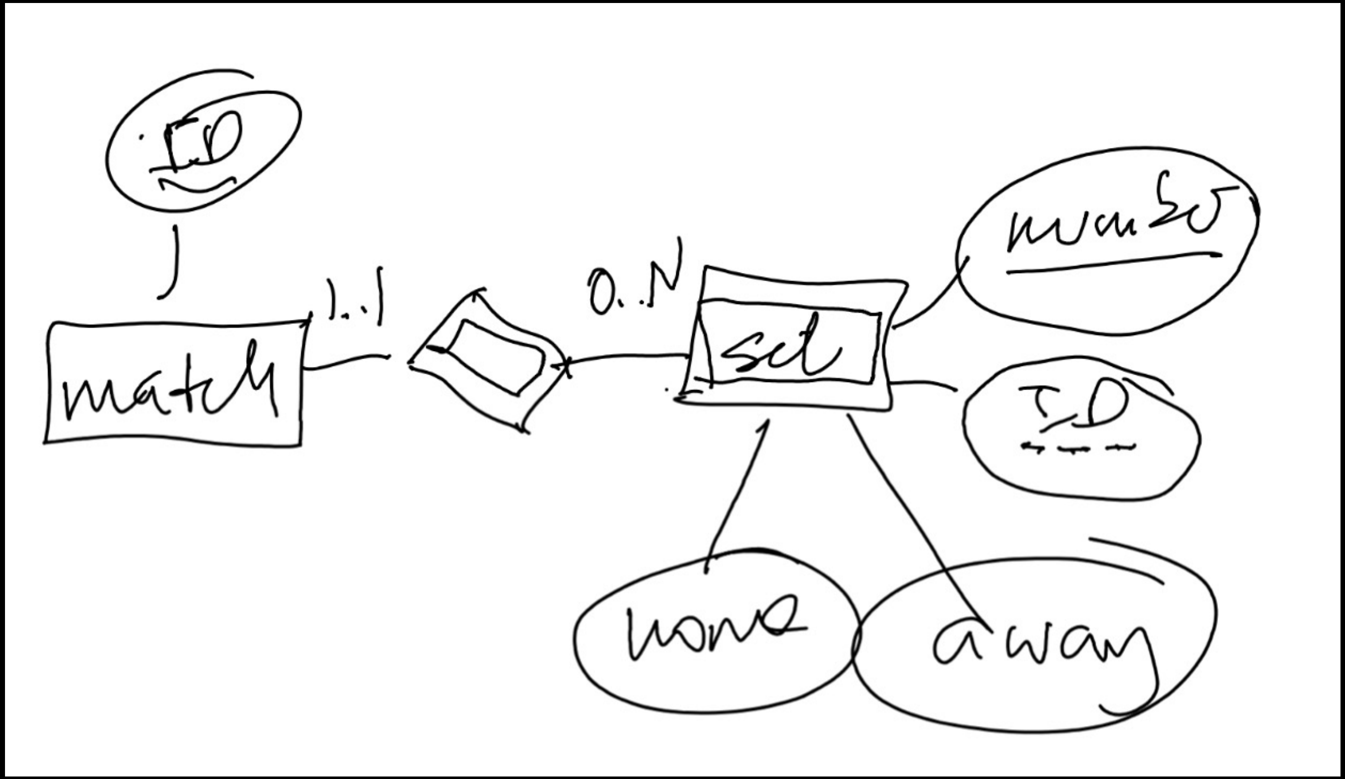
- **Weak entities have a 1..1 relationship type**
  - Some 1..1 relationship types represent weak entities
  - Most 1..1 relationship types do not represent weak entities
- **Main difference is presence of a natural key**
  - Weak entity:
    Only unique within the parent entity!





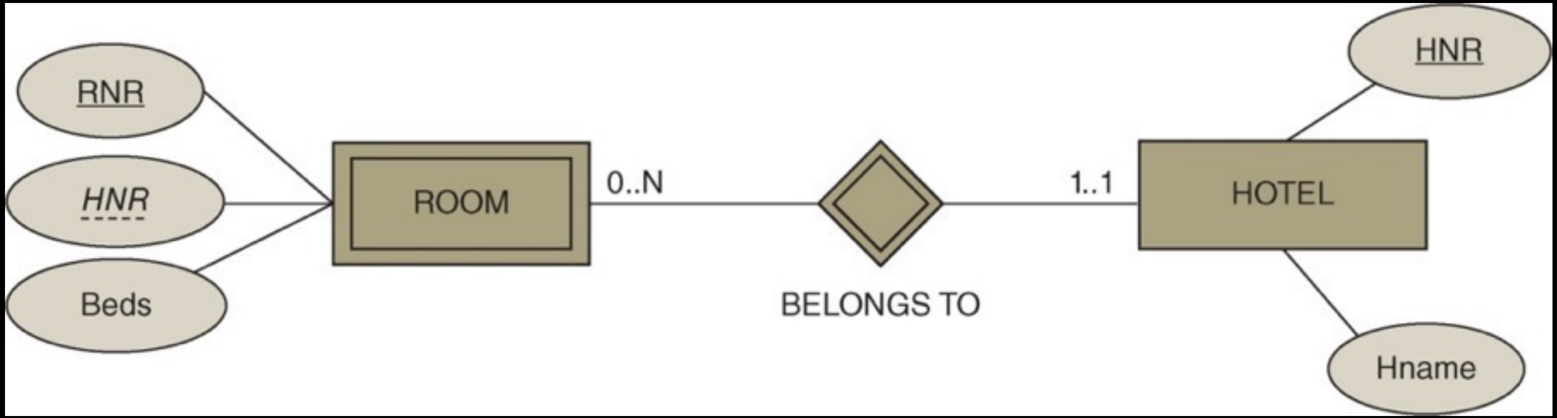
Practice: Weak Entity

- **Each volleyball match consists of sets, each with set number, home score and away score**



Weak Entities in SQL DDL

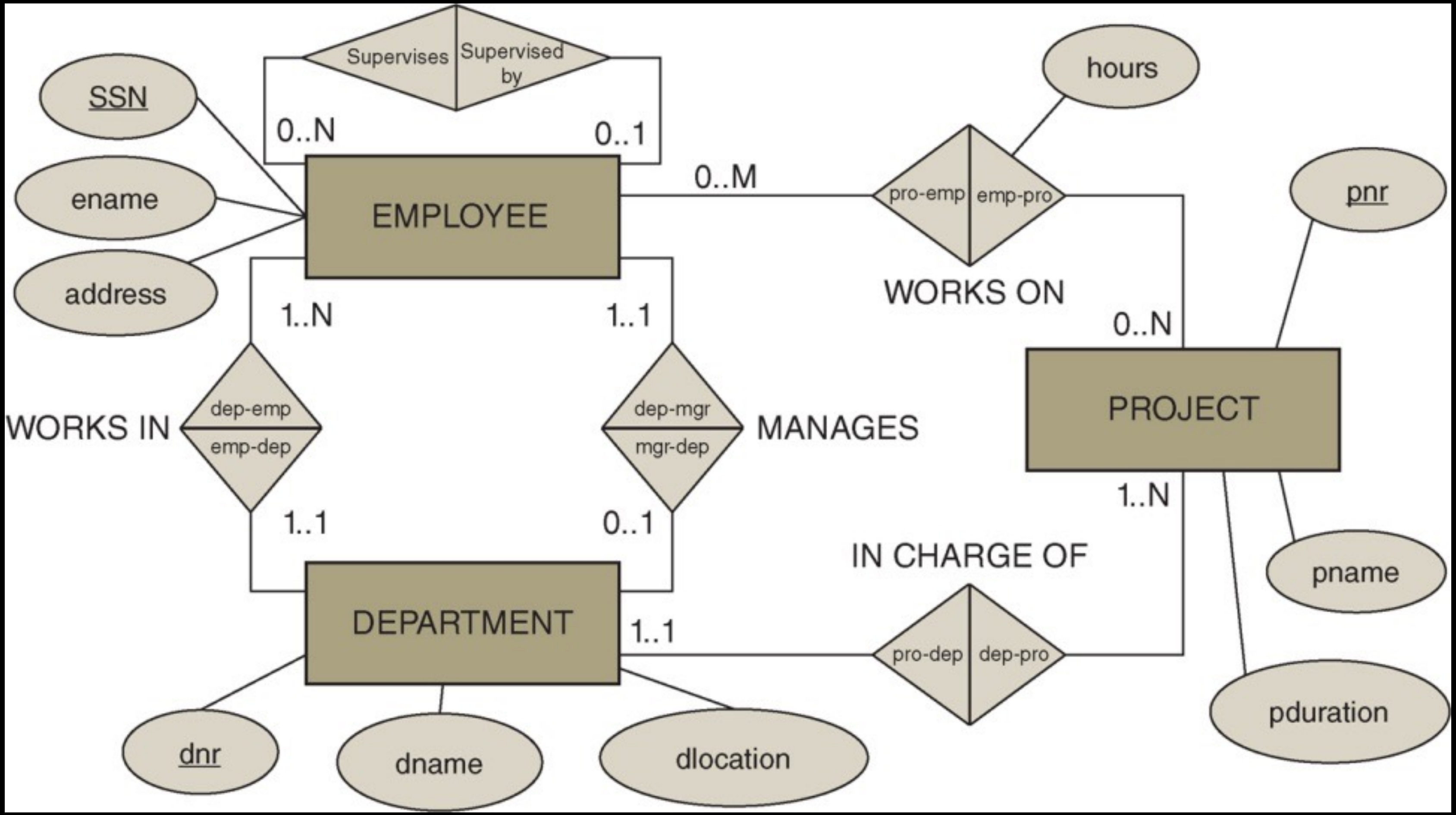
- **Create a new table referencing the entity table**
  - Primary key is parent key + partial key
  - Very similar to multi-valued attributes



```
CREATE TABLE Room  (
    RNR INTEGER,  -- Should not be SERIAL!
    HNR INTEGER   -- Must be a FOREIGN KEY!
        REFERENCES Hotel(HNR),
    Beds INTEGER NOT NULL,
    PRIMARY KEY (HNR, RNR)
);
```

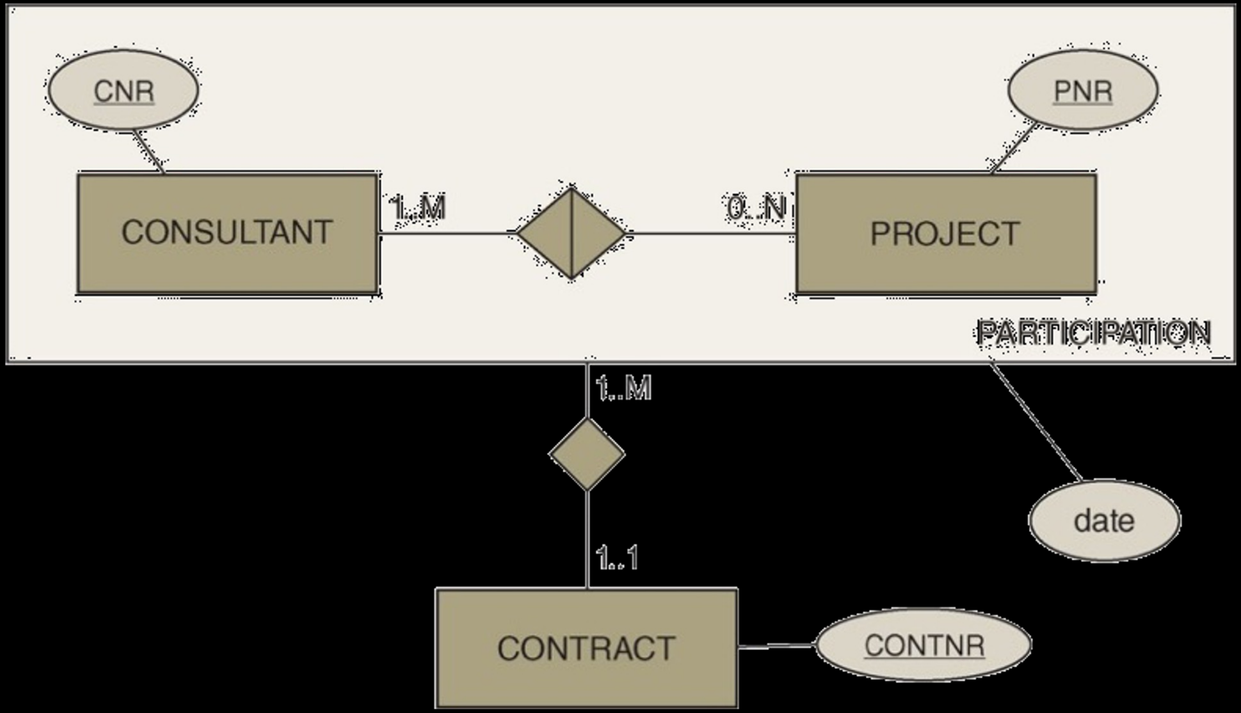
Exercise: Read ER Diagram

- **How many tables will be created? What are the keys?**

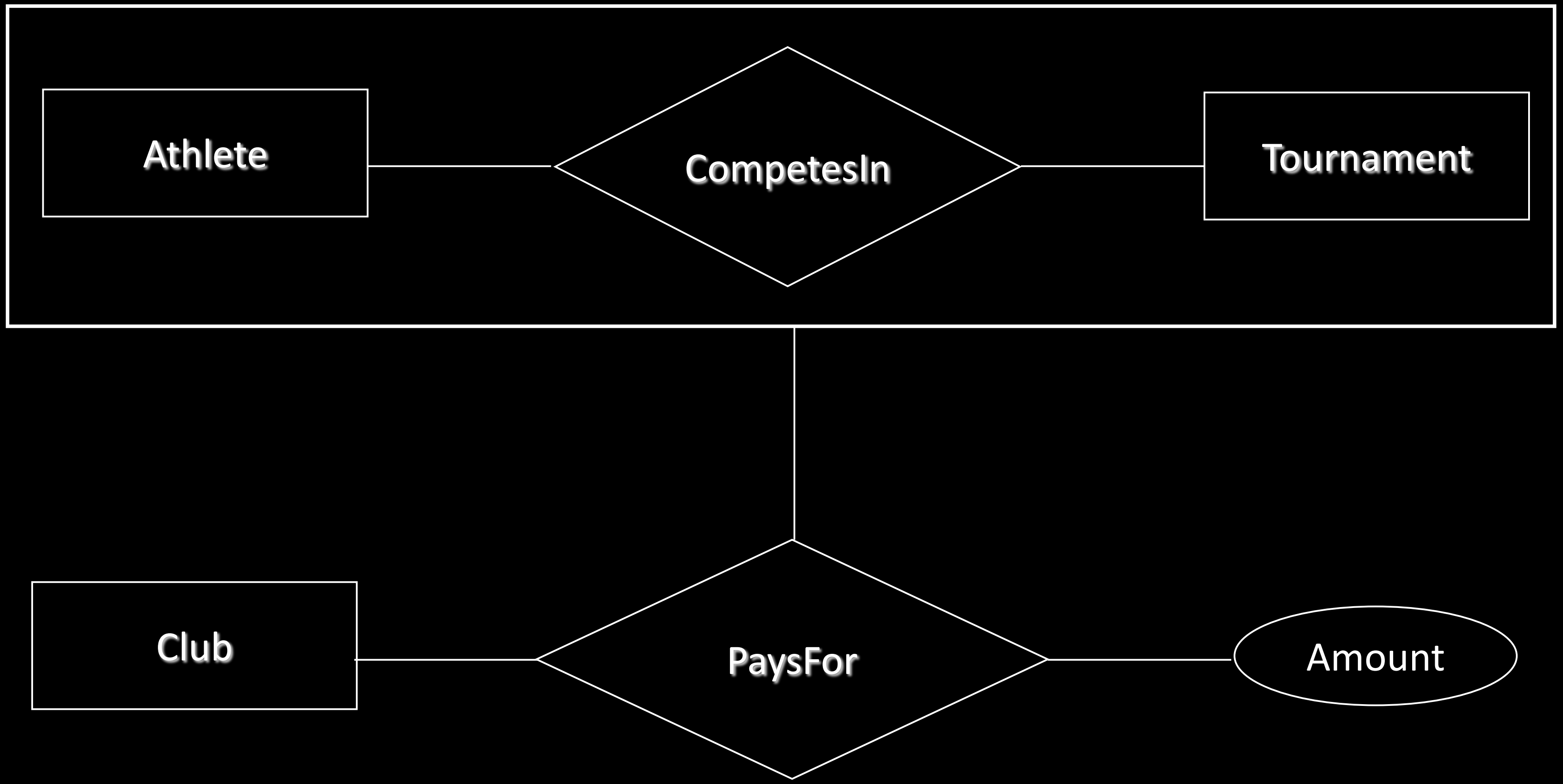


Aggregation

- Sometimes we need a relationship to a relationship
- Aggregation allows us to “convert” relationship types to entity types!
- Typical use: Monitoring, payments, contracts

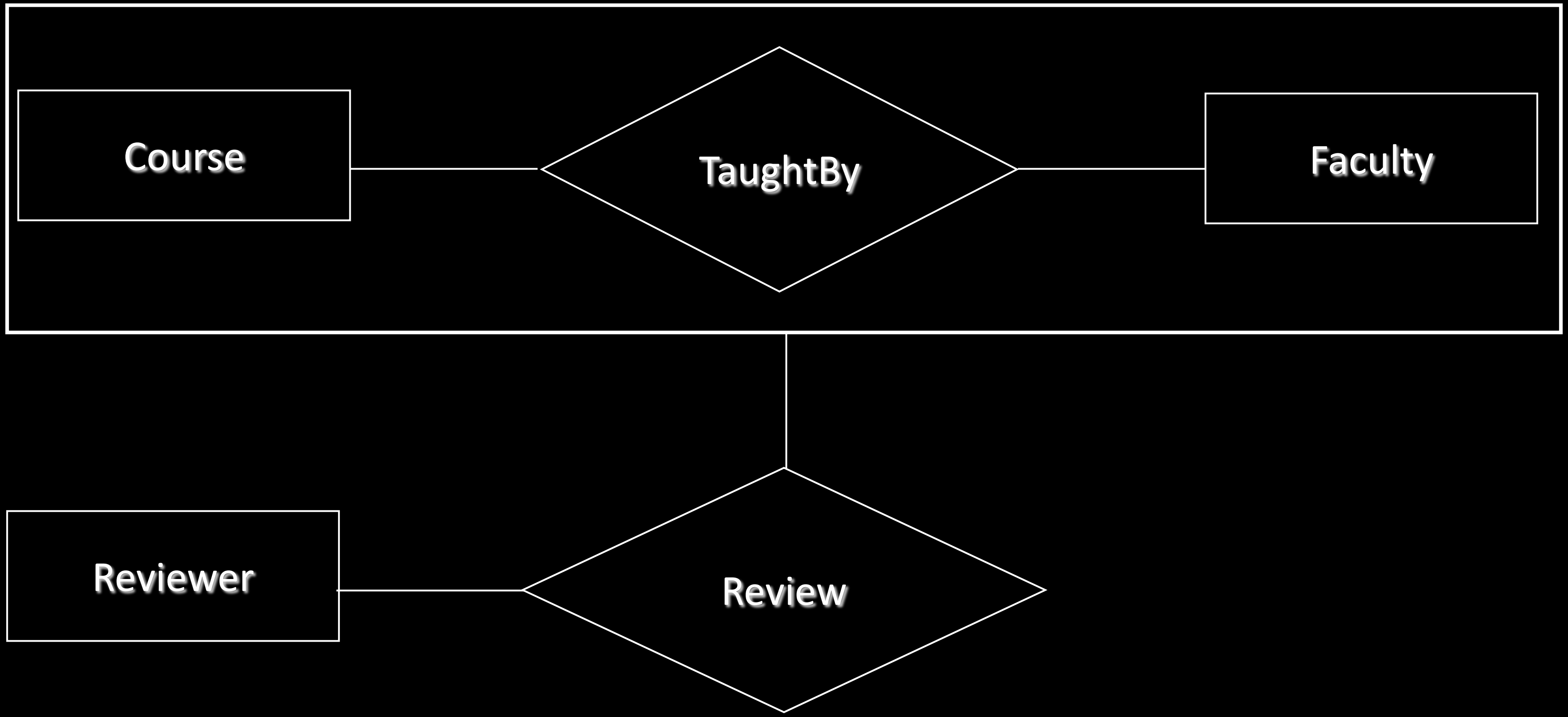


Aggregation: Relationship → Entity



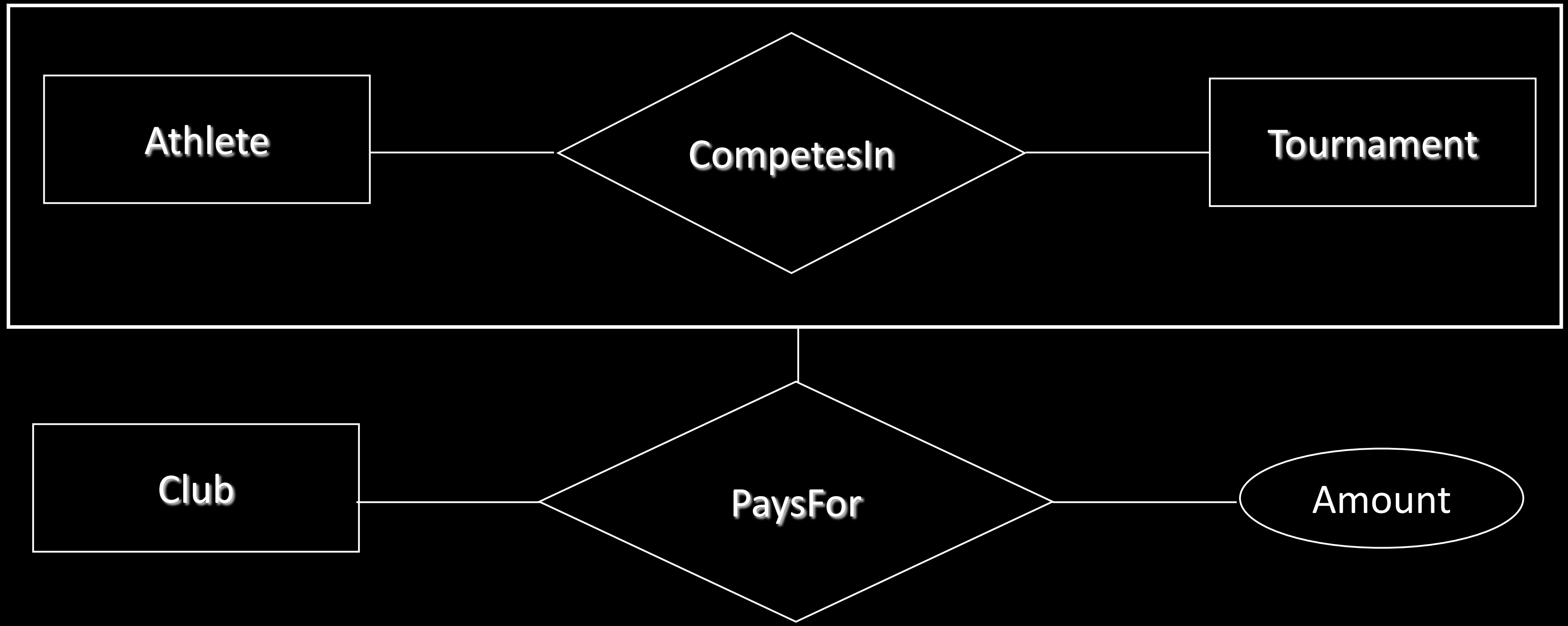
Aggregation Example

- **Courses are taught by faculty members. External reviewers review each instance of the course.**



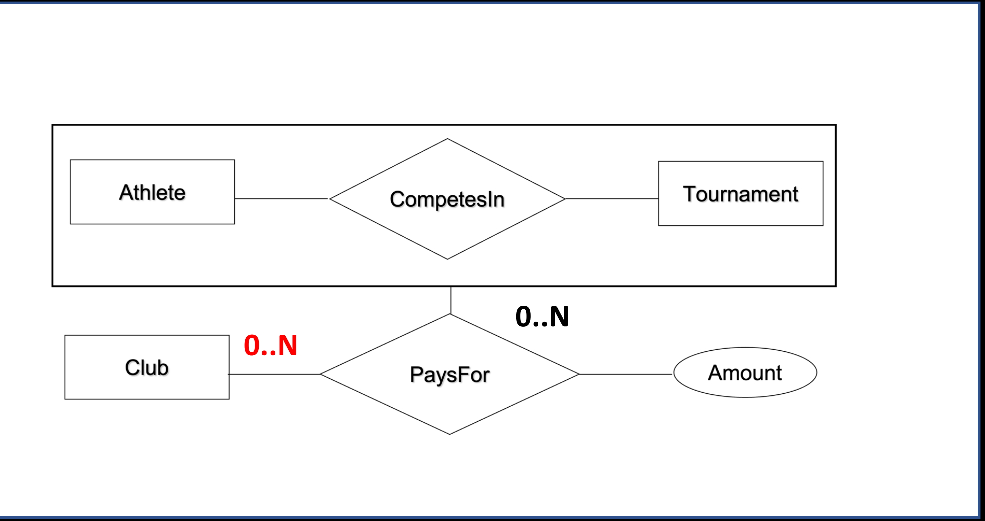
Translation to SQL DDL

- **Main observation: It is a relationship!**
  - Use the same method as for translating relationship
  - Treat the aggregation table as the entity!



Translation to SQL DDL: 0..N

- **Option 1: Use existing relationship key**

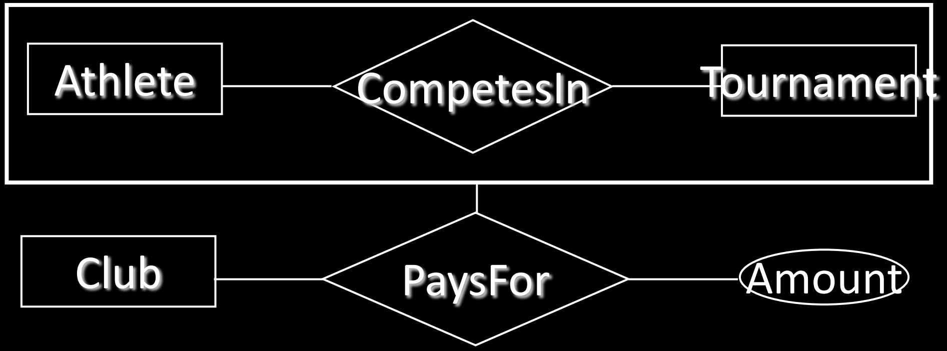


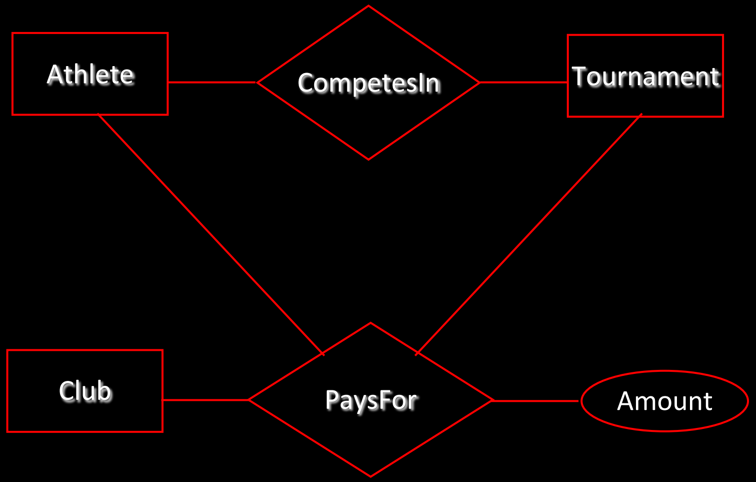
```
create table CompetesIn (
            AID integer references Athletes,
            TID integer references Tournaments,
            primary key (AID, TID)
);

create table PaysFor (
            AID integer,
            TID integer,
            CID integer references Clubs,
            amount integer NOT NULL,
            foreign key (AID, TID) references CompetesIn
(AID, TID),
            primary key (AID, TID, CID)
);
```

Option 1: Impact on Tables?

- **A common error is to use FKs to the entity tables!**
  - This is actually equivalent to the ER diagram on the right
  - See SQL code!

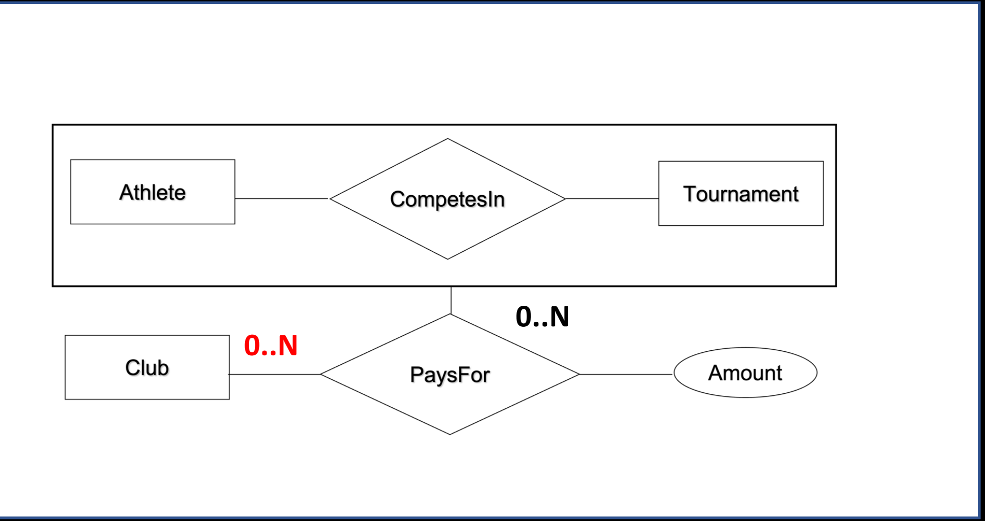
```
create table PaysFor (
        AID integer REFERENCES Athletes (AID),
        TID integer REFERENCES Tournaments (TID),
        CID integer references Clubs,
        amount integer NOT NULL,
        PRIMARY KEY (AID, TID, CID)
        FOREIGN KEY (AID, TID) REFERENCES CompetesIn
(AID, TID),
);
```





Translation to SQL DDL: 0..N

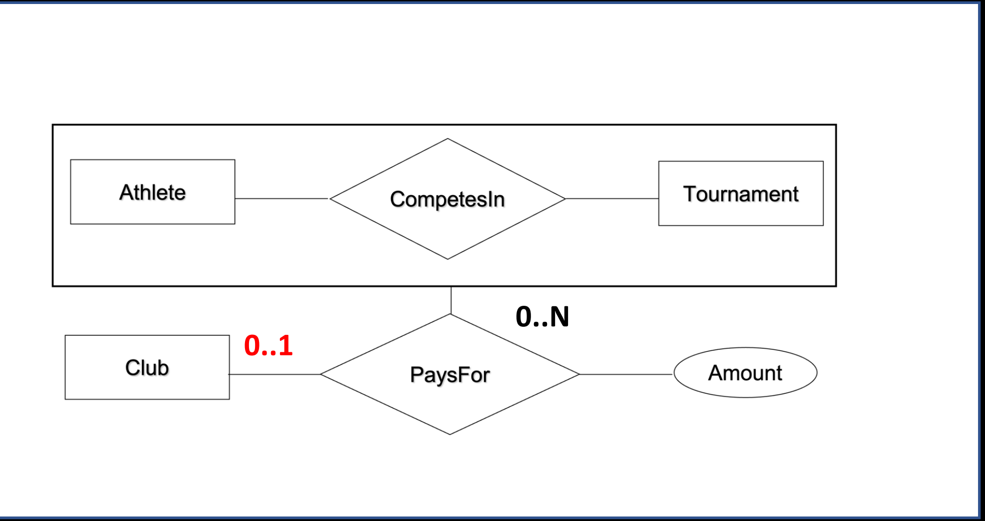
- **Option 2: Create a new relationship key**
  - A common error is to then forget about the existing key!



```
create table CompetesIn (
            CIID SERIAL primary key,
            AID integer NOT NULL references Athletes,
            TID integer NOT NULL references Tournaments,
            UNIQUE (AID, TID)
);

create table PaysFor (
            CIID integer references CompetesIn,
            CID integer references Clubs,
            amount integer NOT NULL,
            primary key (CIID, CID)
);
```

Translation to SQL DDL: 0..1

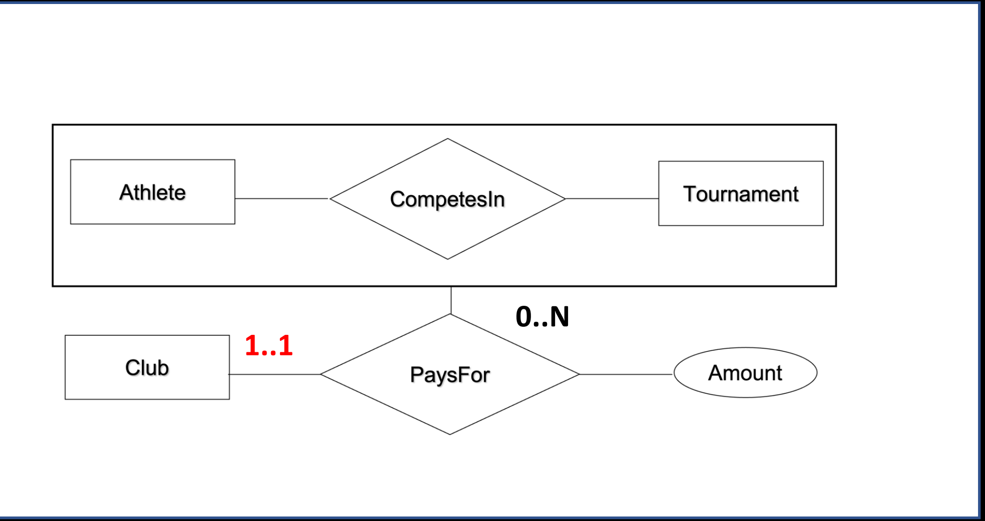
- **Change the primary key**
  - This might happen when payment comes later



```
create table CompetesIn (
        AID integer references Athletes,
        TID integer references Tournaments,
        primary key (AID, TID)
);

create table PaysFor (
        AID integer,
        TID integer,
        CID integer NOT NULL references Clubs,
        amount integer NOT NULL,
        foreign key (AID, TID) references CompetesIn (AID,
TID),
        primary key (AID, TID)
);
```

Translation to SQL DDL: 1..1

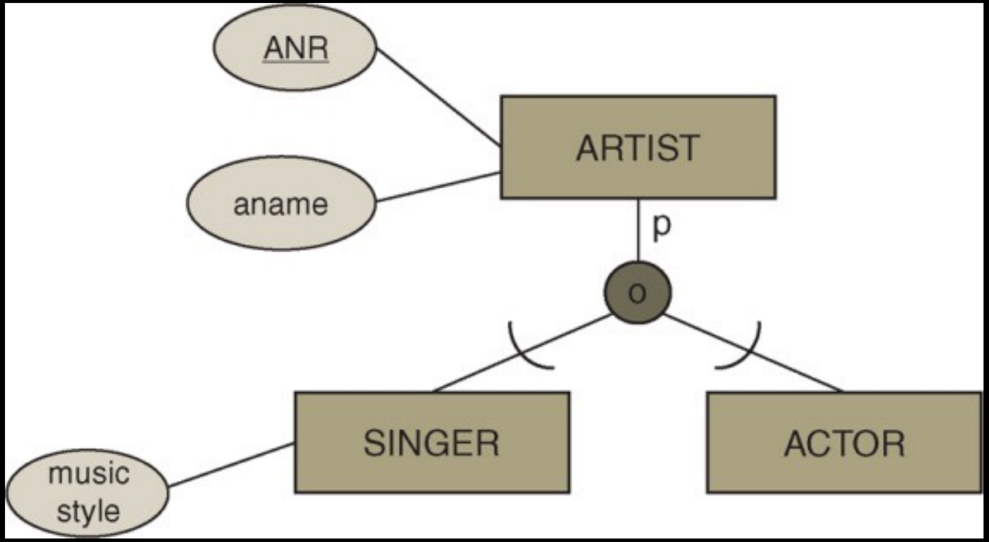
- **Change the relationship table**
  - This is the only case covered in the book!
  - Here: No PaysFor table!

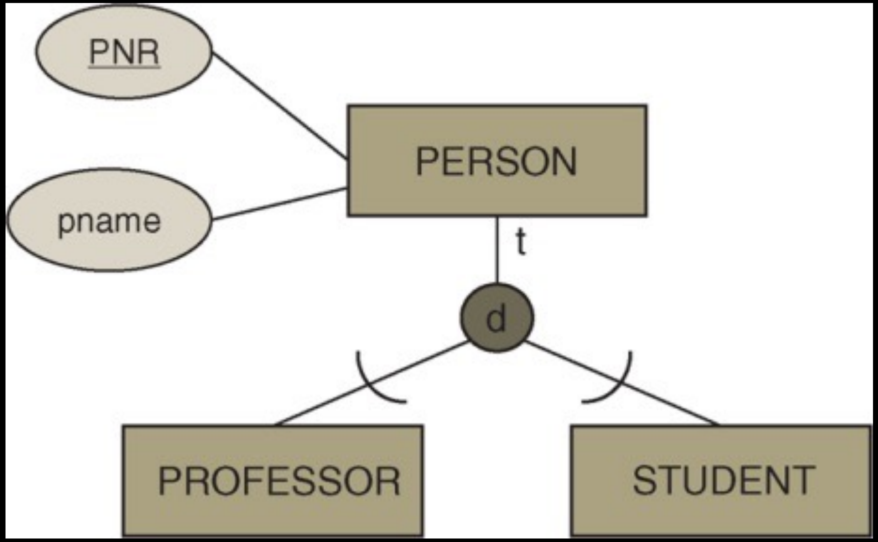


```
create table CompetesIn (
            AID integer references Athletes,
            TID integer references Tournaments,
            CID integer references Clubs NOT NULL,
            amount INTEGER NOT NULL,
            primary key (AID, TID)
);
```

Generalization/Specialization

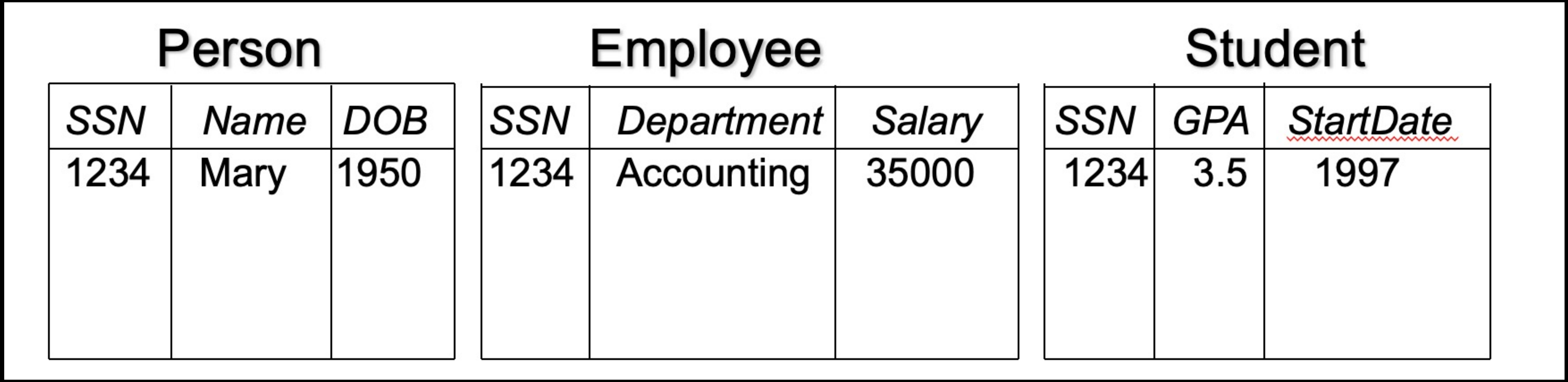
- **Like classes in Java**
  - Partial vs Total = p/t on the line
  - Overlapping vs Disjoint = o/d in the circle
  - Please don't use colour in homeworks/exam
  - Arcs matter → need to draw them (in the exam)!





Specialization in SQL DDL

- **One table for super-type, one per sub-type**
  - The PK of supertype is also PK for all subtypes
  - Each subtype has a FK to the supertype
- **Option 1 in the PDBM book – preferred option by far!**
  - Redundancy is eliminated:
    Name and DOB are stored only once
  - Adjusts well to hierarchies/lattices



Person

SSN	*Name*	*DOB*
1234	Mary	1950

Employee

SSN	*Department*	*Salary*
1234	Accounting	35000

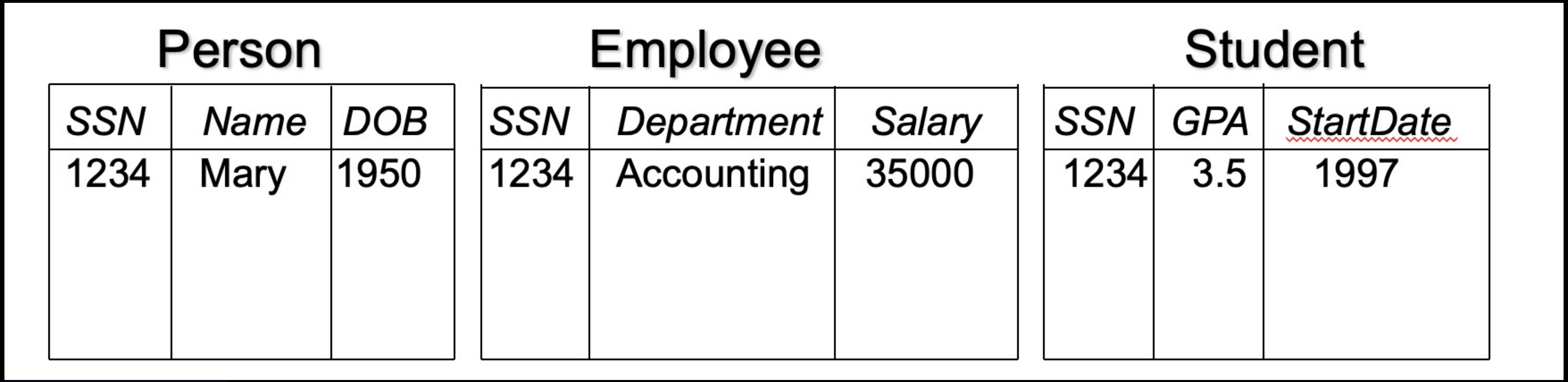
Student

SSN	*GPA*	*StartDate*
1234	3.5	1997

Specialization in SQL DDL: Option 1

```
CREATE TABLE Person (
  SSN INTEGER PRIMARY KEY,
  Name VARCHAR NOT NULL,
  DOB DATE NOT NULL
);

CREATE TABLE Employee (
  SSN INTEGER
    PRIMARY KEY
    REFERENCES Person(SSN),
  Department VARCHAR NOT NULL,
  Salary INTEGER NOT NULL
);
```

```
CREATE TABLE Student (
  SSN INTEGER
    PRIMARY KEY
    REFERENCES Person(SSN),
  GPA REAL NOT NULL,
  StartDate DATE NOT NULL
);
```



Person

SSN	*Name*	*DOB*
1234	Mary	1950

Employee

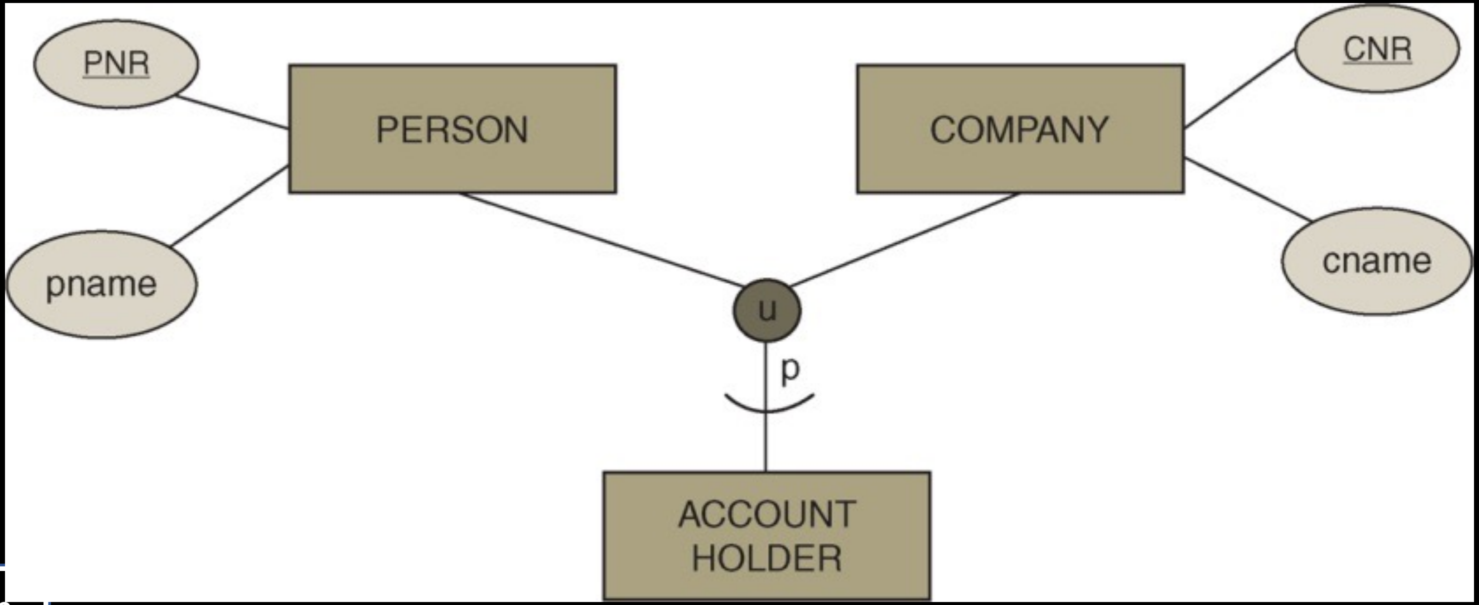
SSN	*Department*	*Salary*
1234	Accounting	35000

Student

SSN	*GPA*	*StartDate*
1234	3.5	1997

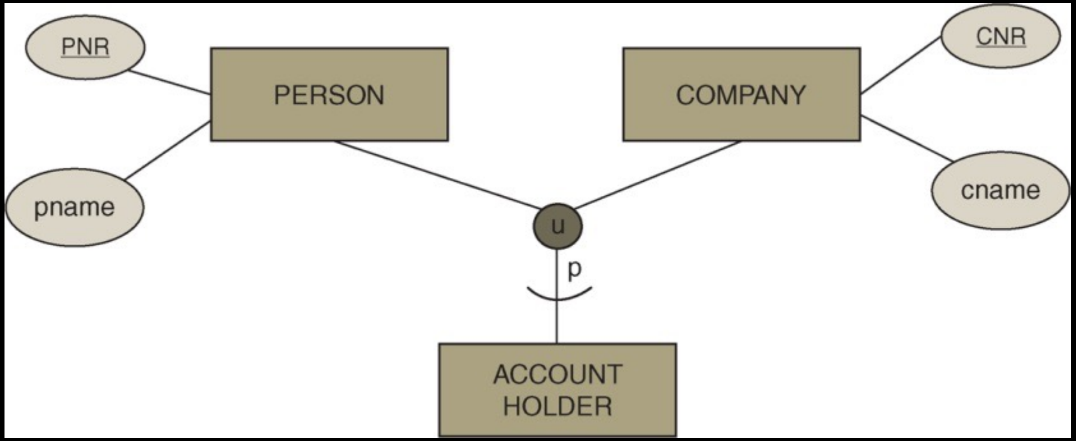
Categorization

- **Grouping of otherwise unrelated entities**
  - New entity is a union = u in the circle
    - Can be total or partial = p/t on the line
  - Can be checked with triggers similarly to specialization
  - Notice the direction of the arc to the union
    - Refers to flow of “inheritance” – or which comes first…
    - Arcs matter → need to draw them (in the exam)!



Categorization in SQL DDL

- **New table for the categorical entity**
  - New abstract PK with INTEGER / SERIAL
- **Add the PK as attribute to the other entities**
  - With FK to the categorical entity
- **Why is this important?**
  Sometimes AccountHolder participates in relationships…

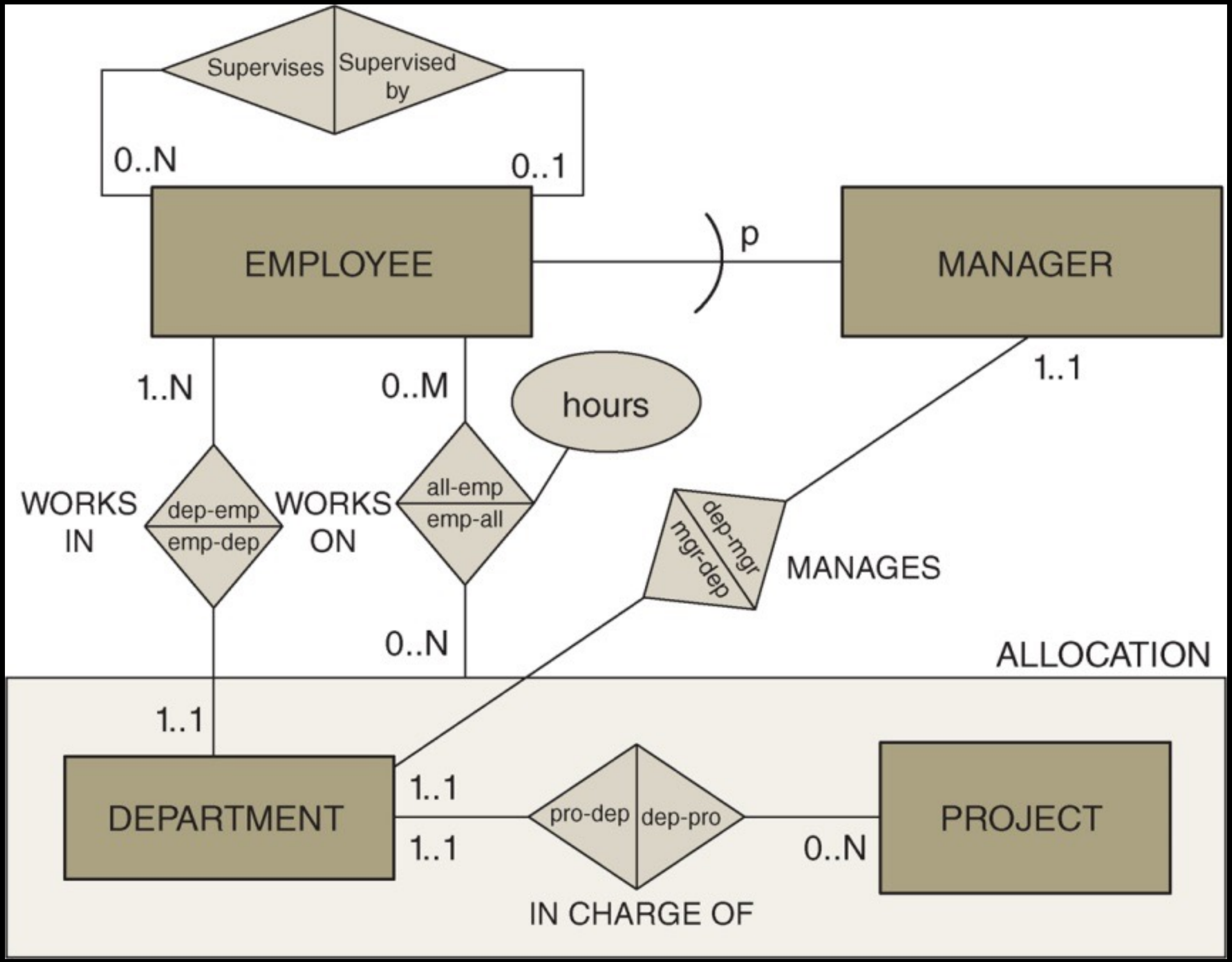


```
CREATE TABLE AccountHolder (
  AcctID SERIAL PRIMARY KEY
);

CREATE TABLE Person (
  PNR SERIAL PRIMARY KEY,
  pname VARCHAR NOT NULL,
  AcctID INTEGER [NOT NULL] REFERENCES AccountHolder(AcctID)
);
```

NOT NULL = total; NULL = partial

Practice: Read EER diagram

- How many tables will be created? What are the keys?



Getting Started?

- **Nouns = entities**
  - Descriptive elements = attributes
- **Verbs = relationships**
  - Descriptive elements = attributes
  - Look for words implying participation constraints
  - No words → 0..N

- Professors have an SSN, a name, an age, a rank, and a research specialty.
- Projects have a project number, a sponsor name (e.g., NSF), a starting date, ...
- Each project is managed by one professor (1..1 on profs, 0..N on projs)
- Professors may work on many projects (0..* on both sides)
- Each project must be reviewed by some professors (1..N on profs, 0..N on projs)

Dealing with Very Large ER diagrams

- **Method 1: Very large paper!**
  - One diagram with all the details
- **Method 2: Outline + Details**
  - One diagram with main entities and their relationships
  - One diagram per entity with attributes and weak entities
- **Method 3: Components**
  - Break the model into components
  - Details inside components
  - Some edge entities are repeated (details in one place)

Limitations of ER Designs

- **ER diagrams do not capture all design details**
  - Example: Multiple candidate keys
  - Must note missing details somewhere!
- **Some aspects do not map well to SQL DDL**
  - Example: 1..M cardinalities
  - Triggers (week 4, BSc) can be used to handle some problems
  - Normalization (week 6) provides a mechanism for fixing some problems
  - Some must simply be noted and addressed in code or ignored!

Summary of Notation Extensions

- **Relationship roles are generally unnecessary**
  - No need to label roles
    - Except for unary relationship types
  - May put relationship name inside rhombus
    - Except for unary relationship types
- **We allow partial keys of relationships**
  - Underlined relationship attribute
  - Part of the PK of the resulting relationship relation
- **Aggregation entity may cover only the relationship**
  - Much easier to read!
- **Allow 0..* in place of 0..N/M/L**
  - … or simply use 0..N everywhere!

Takeaways

- **ER design captures entities and relationships**
  - Weak entities, specialization, categorization and aggregation allow capturing more detailed model characteristics
  - This is hard – but also useful – so you must practice!
- **Conversion to SQL DDL**
  - Entities and relationships mapped to relations
  - Essentially an algorithmic process (with some options)
  - This is hard – but also useful – so you must practice!
- **Notation: MANY VARIANTS EXIST!**
  - We use the one from the book (as extended in lecture)
  - You must use this notation in the homework and exam!!!

What is next?

Next Lecture:

- Data Normalization