Introduction to Database Systems IDBS - Fall 2024

Lecture 5 - Designing Databases

ER Diagrams
Translation to SQL DDL

Readings: PDBM 3.0-3.3, 6.3-6.4

Omar Shahbaz Khan

General Info

EXERCISE 5: ER Design and Implementation

- Large and comprehensive
- More than 2hrs, but use it to practice

HOMEWORK 2 - OUT NOW!

- DEADLINE: 14. October 2024 23:59

-- TODO

- Conceptual Data Modeling (ER Diagram)
  - Entities and Attributes
  - Relationships
    - Cardinalities
    - Partial Relationship Keys
  - Weak Entities
  - Aggregation
  - Generalization/Specialization
  - Categorization

- Translation to SQL DDL

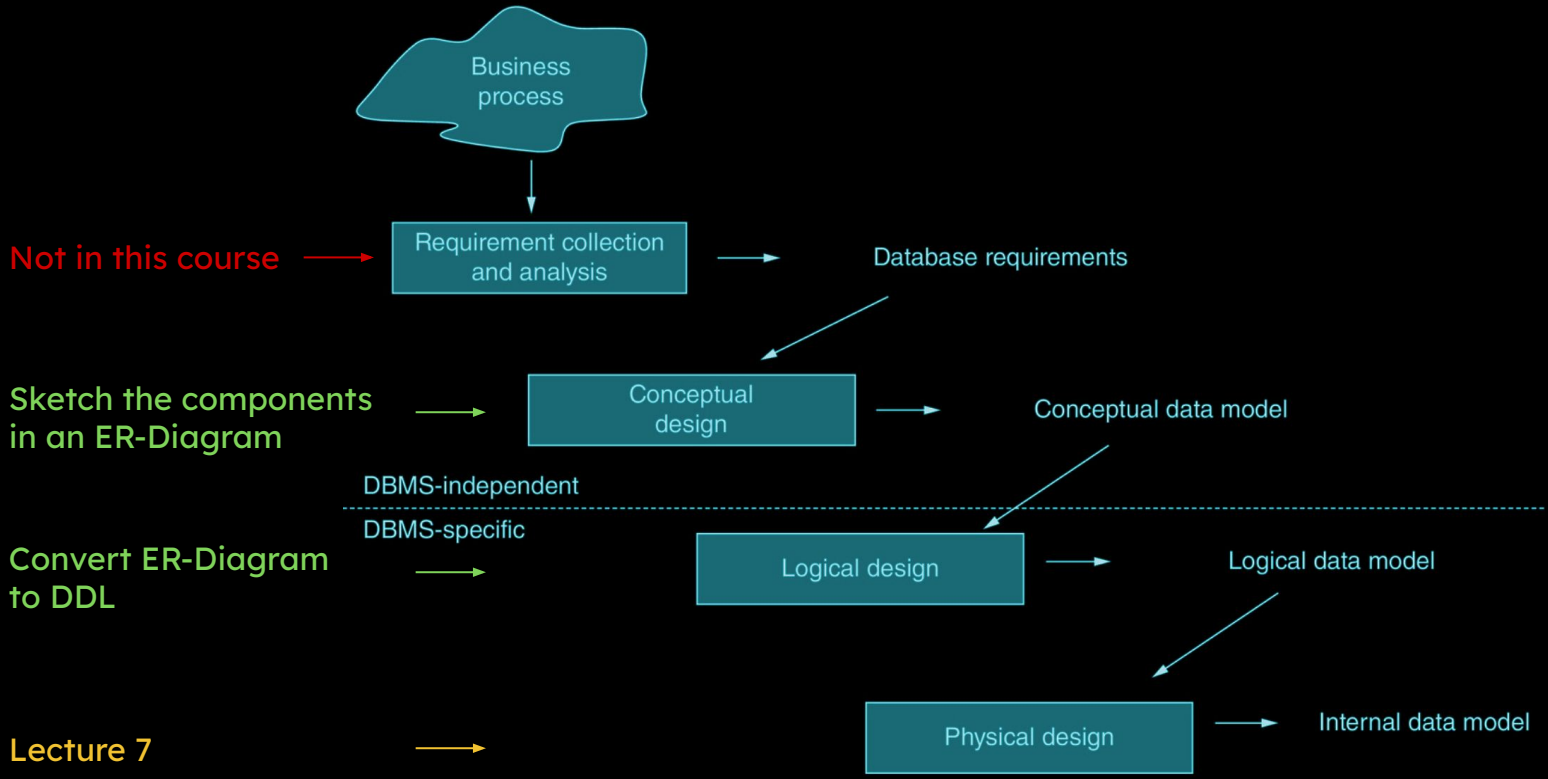
Conceptual Data Modeling

Why do we need a Conceptual Model?

- The “higher-ups” know they want a database…
  … but not what should be in it!

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

Design Process



ER: Entity-Relationships

- Conceptual Data Modeling technique was 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

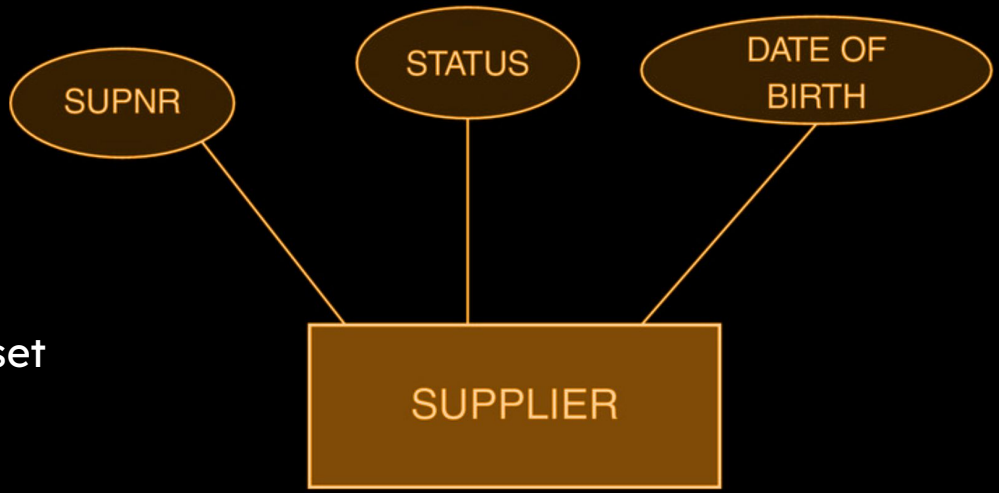
Entity and Attribute Types

Entity Type:

- Set of similar “things”
  - Ex: Movie, Supplier
- Entity = Instance
  - Ex: Interstellar, 2014

Attribute Type:

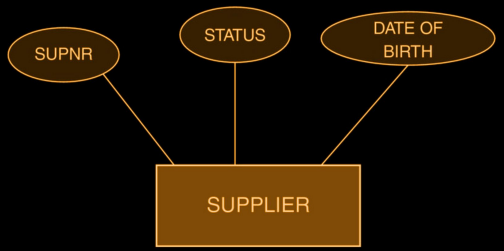
- Describes one aspect of an entity set
  - Ex: name, year, address

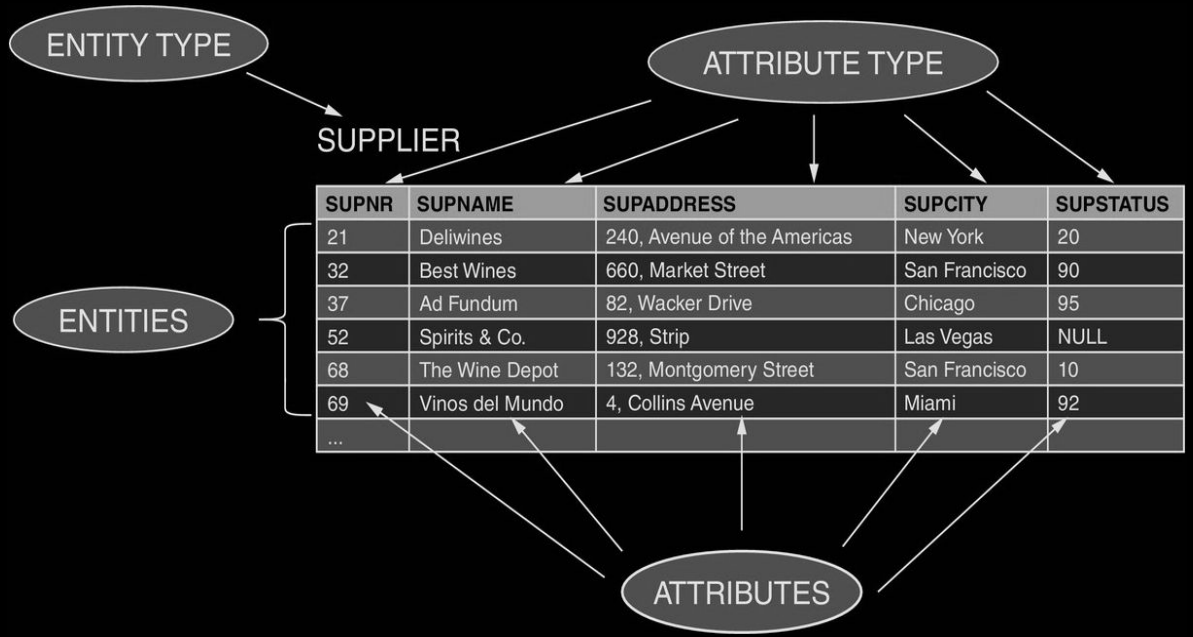


Is there something suspicious about this entity?

Entity and Attribute Types

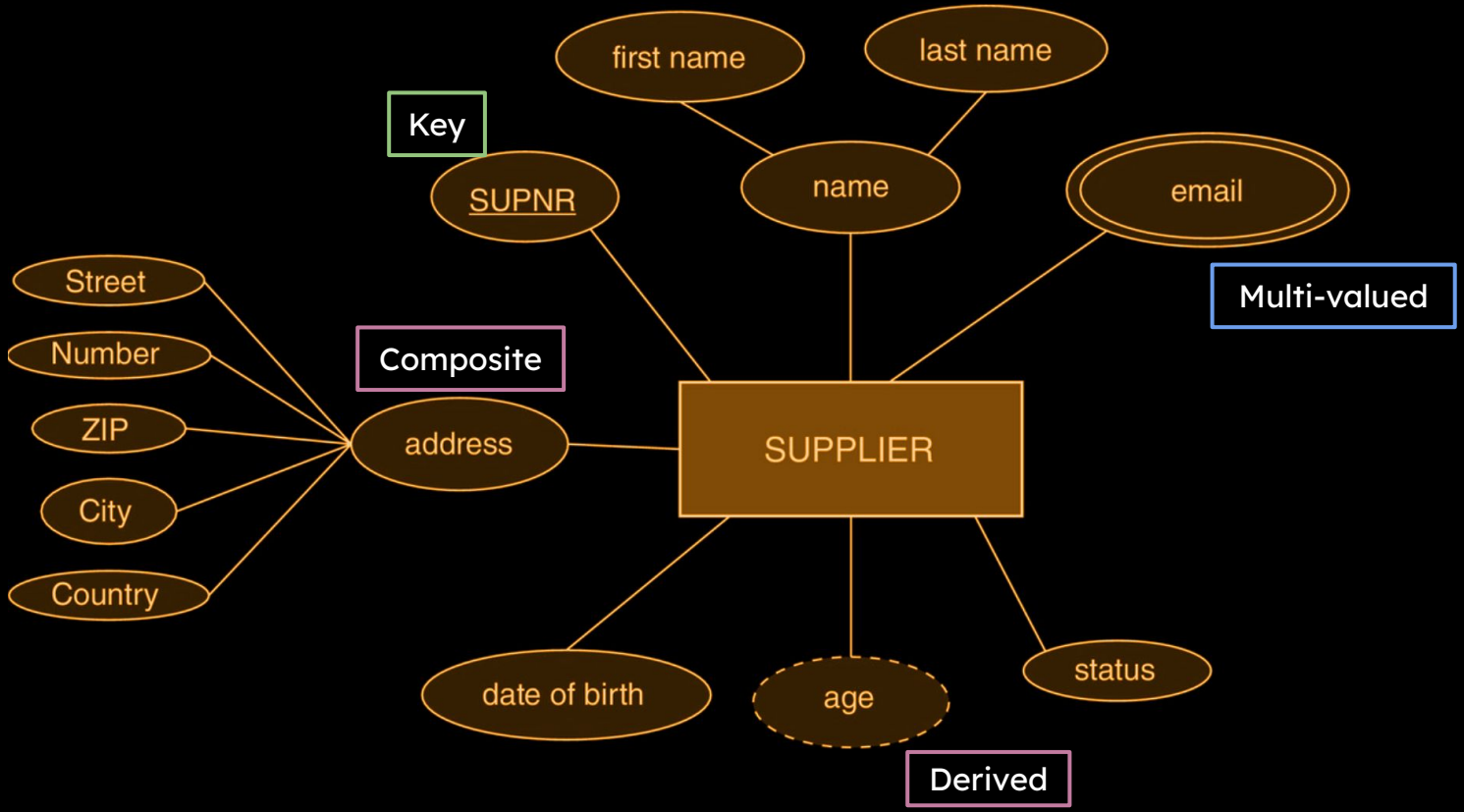
```
CREATE TABLE Supplier (
      SUPNR INT PRIMARY KEY,
      SUPNAME VARCHAR NOT NULL,
      ...
);
```



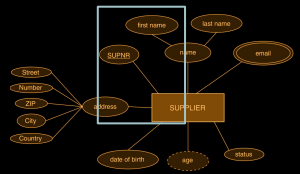


SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS
21	Deliwines	240, Avenue of the Americas	New York	20
32	Best Wines	660, Market Street	San Francisco	90
37	Ad Fundum	82, Wacker Drive	Chicago	95
52	Spirits & Co.	928, Strip	Las Vegas	NULL
68	The Wine Depot	132, Montgomery Street	San Francisco	10
69	Vinos del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

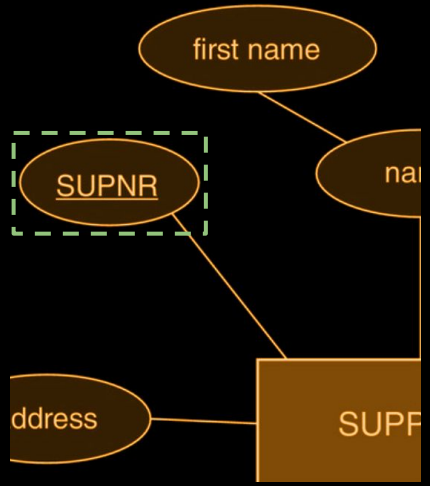
More Attribute Types



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!

```
CREATE TABLE Supplier (
      SUPNR INT PRIMARY KEY,
      ...
);
```

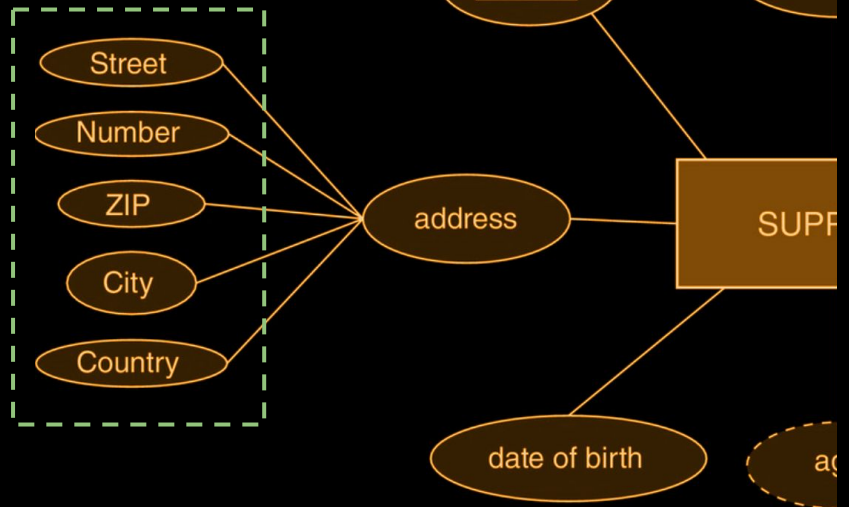
```
CREATE TABLE Supplier (
      SUPNR INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
      ...
);
```

In case you need an incremental sequence

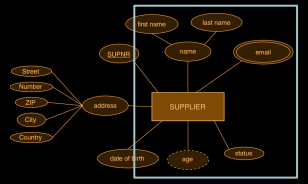
Composite Attributes in SQL DDL

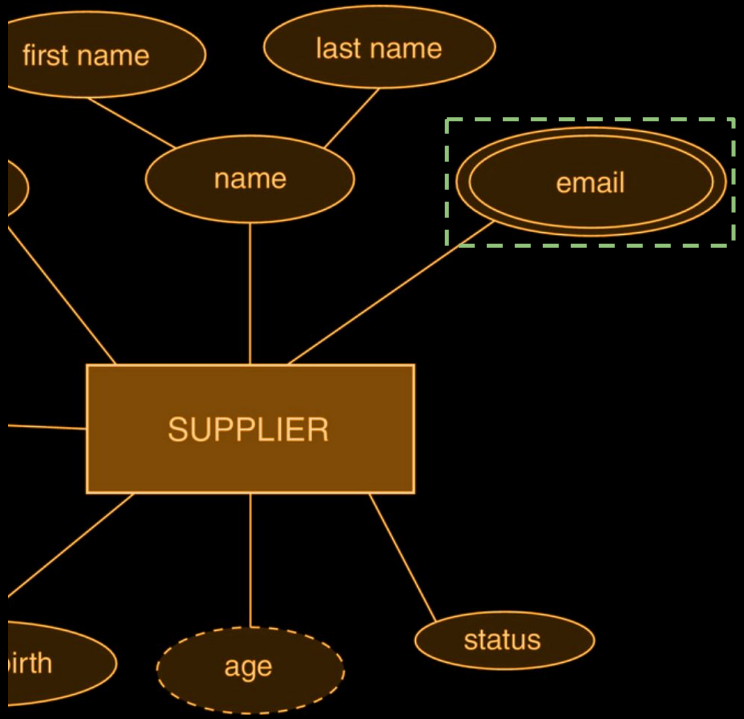
- Simply add the detailed attributes:

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



Multi-Valued Attributes in SQL DDL

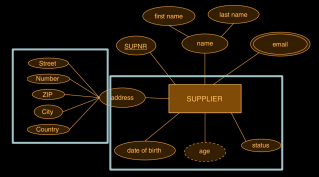




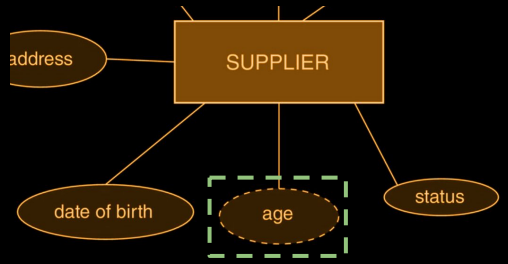

```
CREATE TABLE Emails (
    Email VARCHAR,
    SupNR INTEGER REFERENCES Supplier(SupNR),
    PRIMARY KEY (Email, SupNR)
);
```

- Could add `ON DELETE CASCADE` to SupNR
- Generally a good idea to add to foreign keys, if the related rows should not exists independent of that key

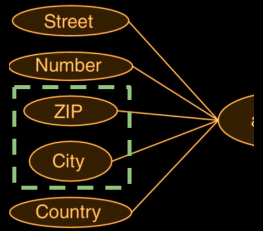
Derived Attributes in SQL DDL



- 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 (Lecture 6)



The CHECK Statement

- We can add constraints that check the value of a field directly in the DDL
- These can be on the column or on the table
  - Postgresql does not care but other database system may

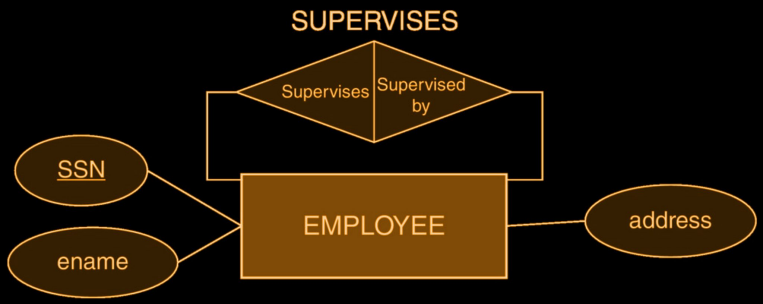
```
CREATE TABLE Supplier (
    Id INTEGER PRIMARY KEY,
    ...
    status CHAR(10) NOT NULL
    CHECK (status IN ('Active','Inactive')),
    ...
);

CREATE TABLE Product (
    Id INTEGER PRIMARY KEY,
    ...
    price FLOAT NOT NULL,
    ...
    CHECK (price > 0),
    ...
);
```

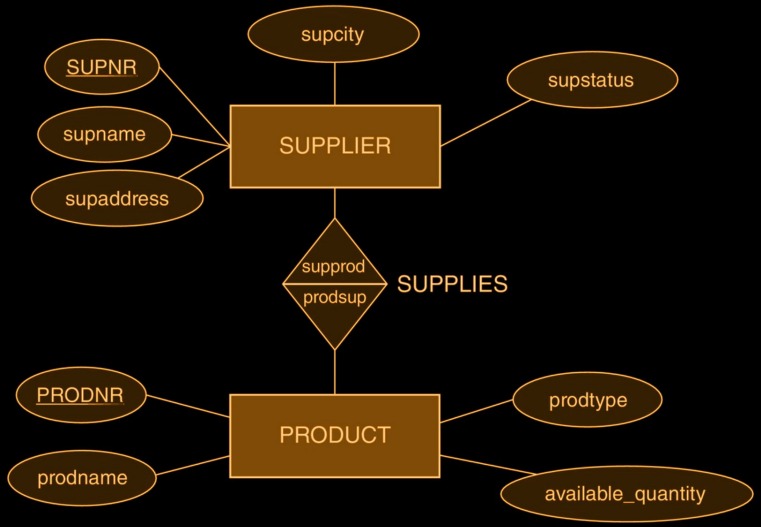
https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-check-constraint/

Relationships

Relationship Types

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






```
CREATE TABLE Supplies (
        SupNR INT REFERENCES Supplier,
        ProdNR INT REFERENCES Product,
        PRIMARY KEY (SupNR, ProdNR)
);
```

ER-Diagram Exercise

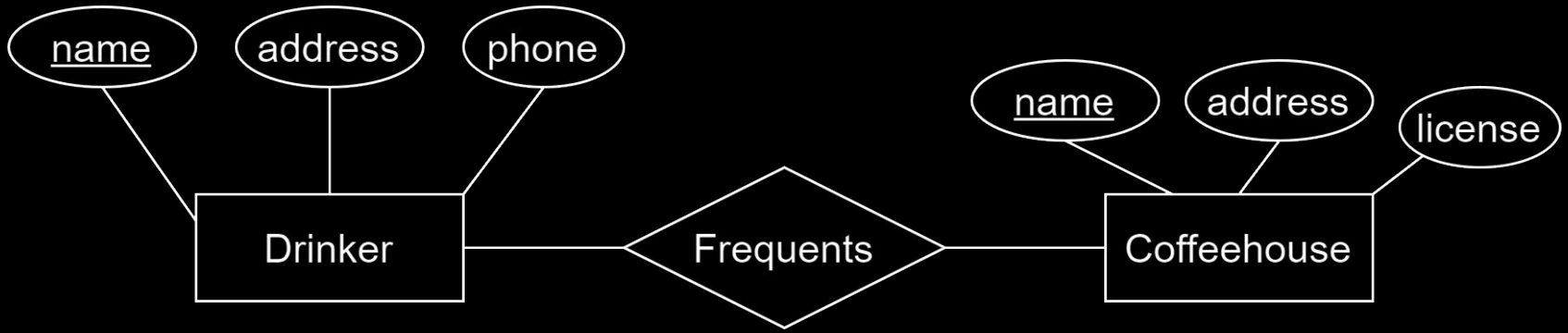
DB Requirements:

- A drinker has a (unique) name, address, and phone.
- A coffeehouse has a (unique) name, address, and license.
- Store which drinkers frequent which coffeehouses.

ER-Diagram Exercise

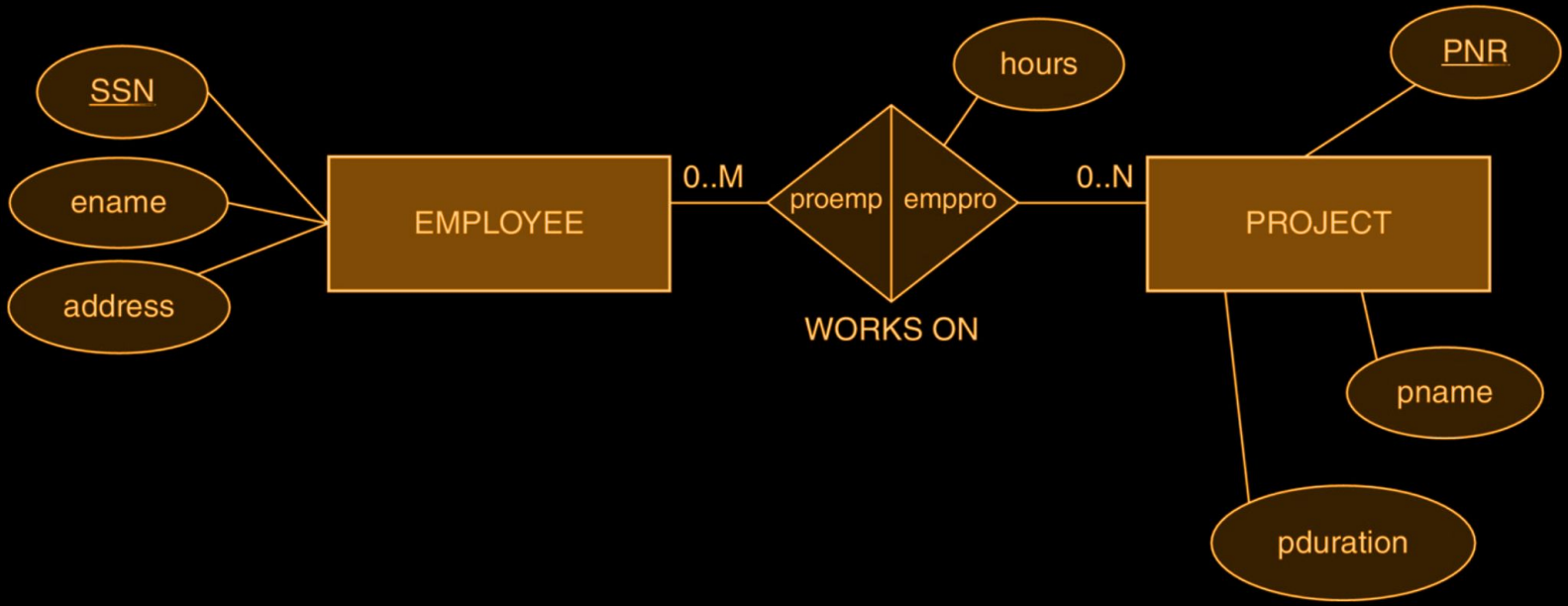
DB Requirements:

- A drinker has a (unique) name, address, and phone.
- A coffeehouse has a (unique) name, address, and license.
- Store which drinkers frequent which coffeehouses.



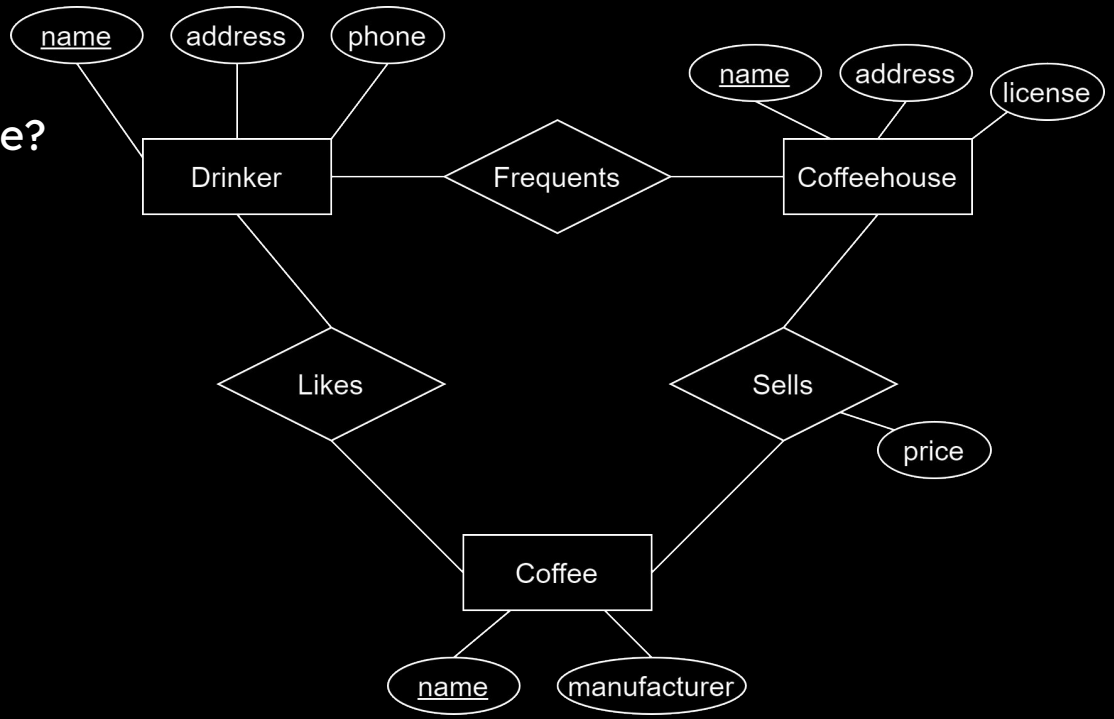
Relationship Attribute Types

- Relationships can also have attribute types



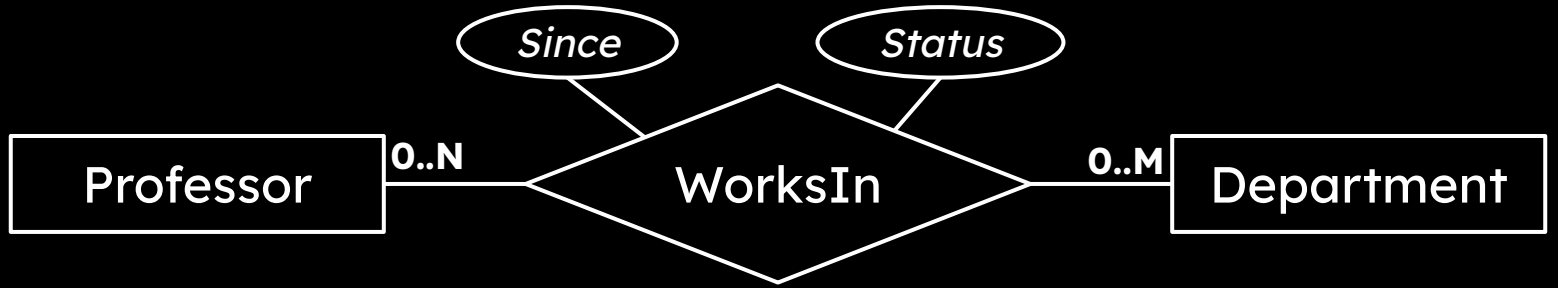
Example: Coffee DB

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



Coffees (name, manufacturer)
Coffeehouses (name, address, license)
Drinkers (name, address, phone)
Likes (drinker, coffee)
Sells (coffeehouses, coffee, price)
Frequents (drinker, coffeehouse)

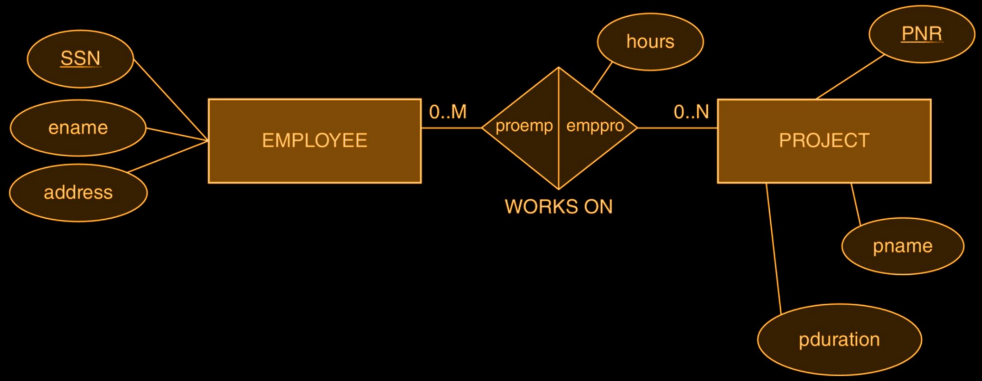
Basic Relationship Table in SQL DDL

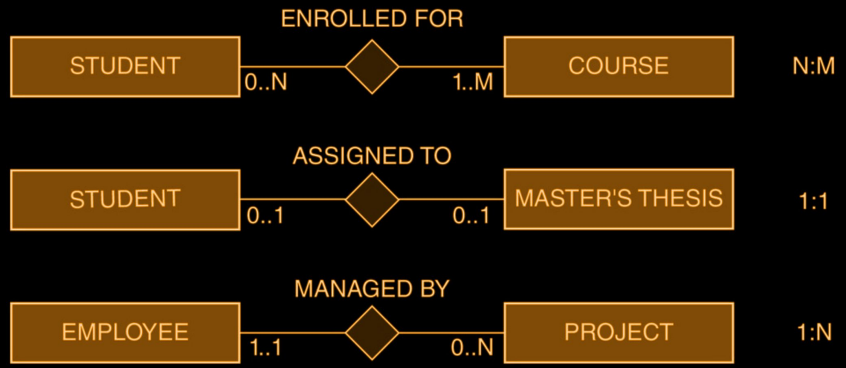



```sql
CREATE TABLE WorksIn  (
    ProfID INT,                 -- role (key of Professor)
    DeptID CHAR(4),             -- role (key of Department)
    Since DATE NOT NULL,        -- attribute
    Status CHAR(10) NOT NULL, -- attribute
    PRIMARY KEY (ProfID, DeptID),
    FOREIGN KEY (ProfID) REFERENCES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

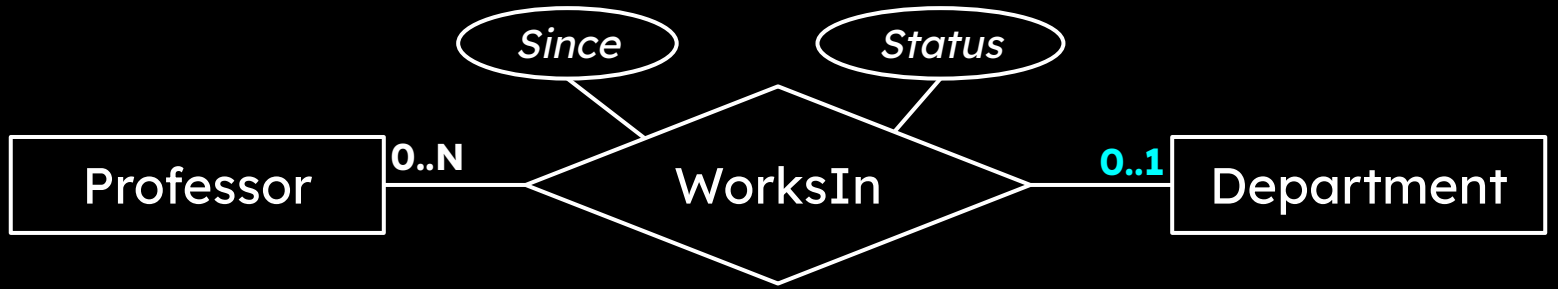
Cardinalities

- Relationships always have cardinalities
  - Minimum: 0 or 1
  - Maximum: 1 or N / M / P / * / …
- Read:
  - Entity (ignore) Relationship Cardinality Entity
  - Employee can work on zero to many projects
- Do cardinalities impact the resulting table structure?





Maximum 1

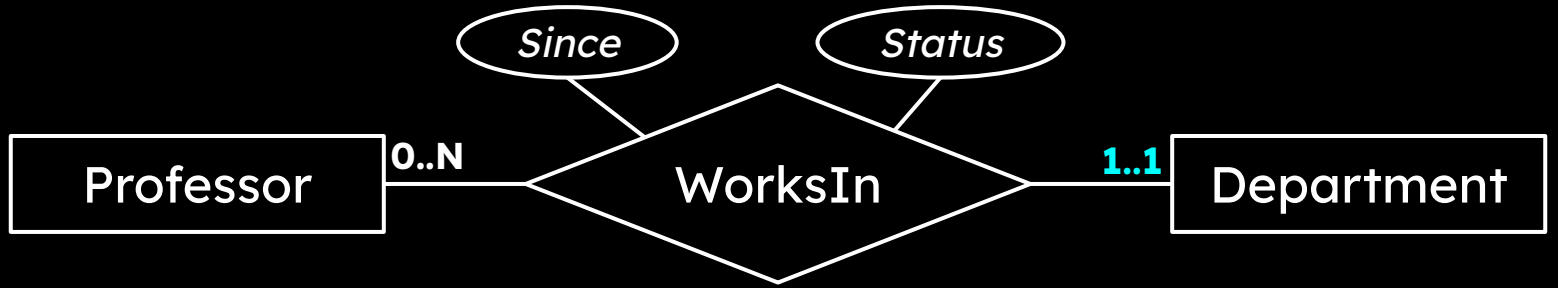



```
CREATE TABLE WorksIn  (
     ProfID INT,                  -- role (key of Professor)
     DeptID CHAR(4) NOT NULL,     -- role (key of Department)
     Since DATE NOT NULL,         -- attribute
     Status CHAR(10) NOT NULL,    -- attribute
     PRIMARY KEY (ProfID),        -- each professor only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

Maximum 1 - No Attributes

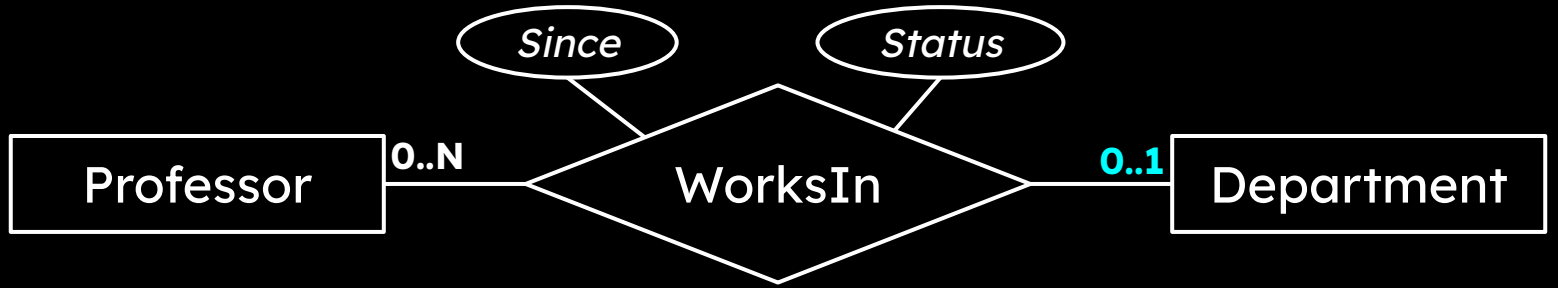

```
CREATE TABLE WorksIn (...)
CREATE TABLE Professor (
     Id INT PRIMARY KEY,
     ...
     deptId INT,  -- foreign key
     FOREIGN KEY (deptId) REFERENCES Department(Id)
)
```

Exactly 1




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

Maximum 1 - Revisited

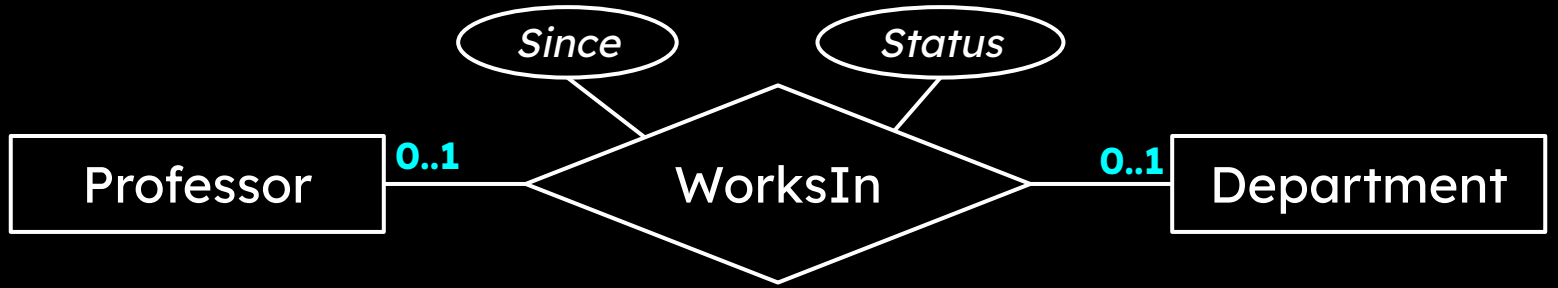


Could we do the same for 0..1?

- Yes, but the three attributes must be able to be NULL
- **We only do this if the relationship has no attributes**

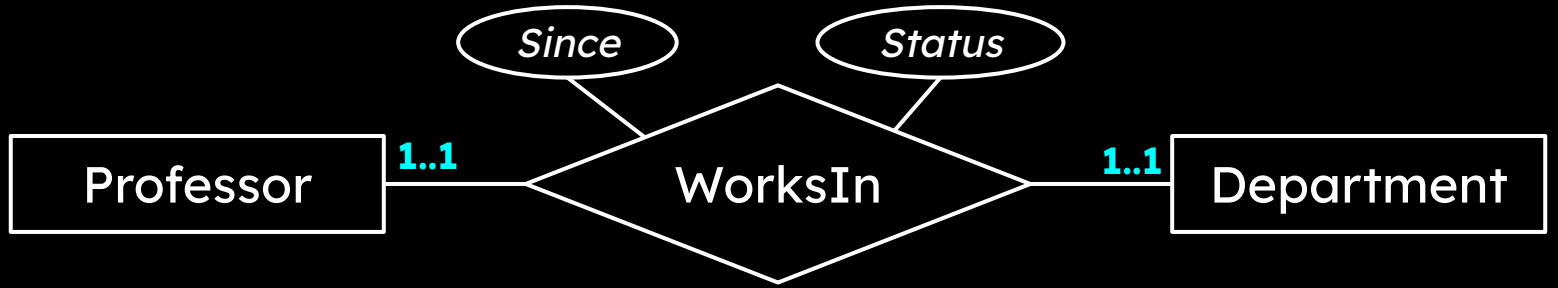
```
CREATE TABLE Professor (
      Id INT PRIMARY KEY,
      ...
      deptId INTEGER,          foreign key
      Since DATE,              attribute
      Status CHAR(10),         attribute
      FOREIGN KEY (deptId) REFERENCES Department(Id)
)
```

Maximum 1 - Both directions




```
CREATE TABLE WorksIn  (
     ProfID INT,                -- 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 - Both direction




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

```sql
CREATE TABLE Department (
    Id INT PRIMARY KEY,
    ...
    ProfId INT NOT NULL,
    FOREIGN KEY (ProfId)
    REFERENCES Professor(Id)
)
```

Exactly 1 - Both direction

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

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

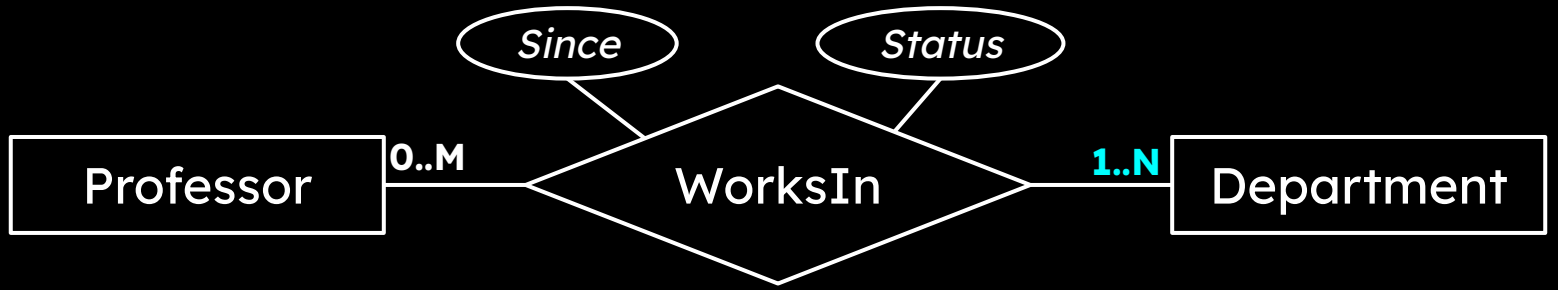
```
CREATE TABLE Department (
     Id INT PRIMARY KEY,
     ...
     ProfId INT NOT NULL,
     FOREIGN KEY (ProfId)
     REFERENCES Professor(Id)
)
```

Exactly 1 - Both directions

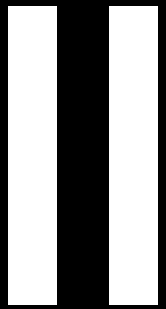
- Alternative 2: Merge Tables
  - May work well in some cases
  - Depends on entities
- Alternative 3: Pick one FK direction
  - Write down the other requirement
  - Do the best in software with the other direction

```
CREATE TABLE Professor  (
    Id INT PRIMARY KEY,
    ...
    deptId INT 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 a trigger on Professor/WorksIn
- Normally:
  - Write down the requirement
  - Do the best in software

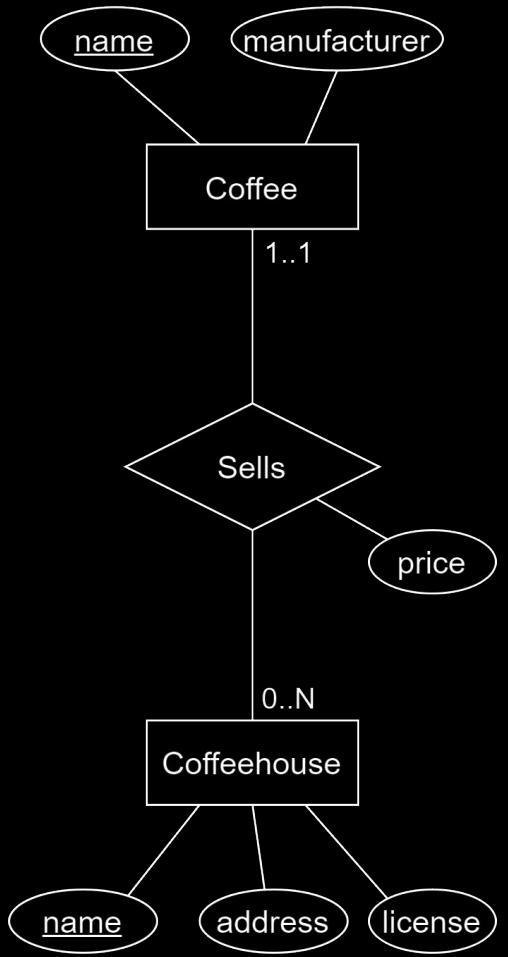


Exercise

- Draw this schema as ER diagram… BUT:
  - Use IDs
  - Assume each coffeehouse sells exactly one coffee

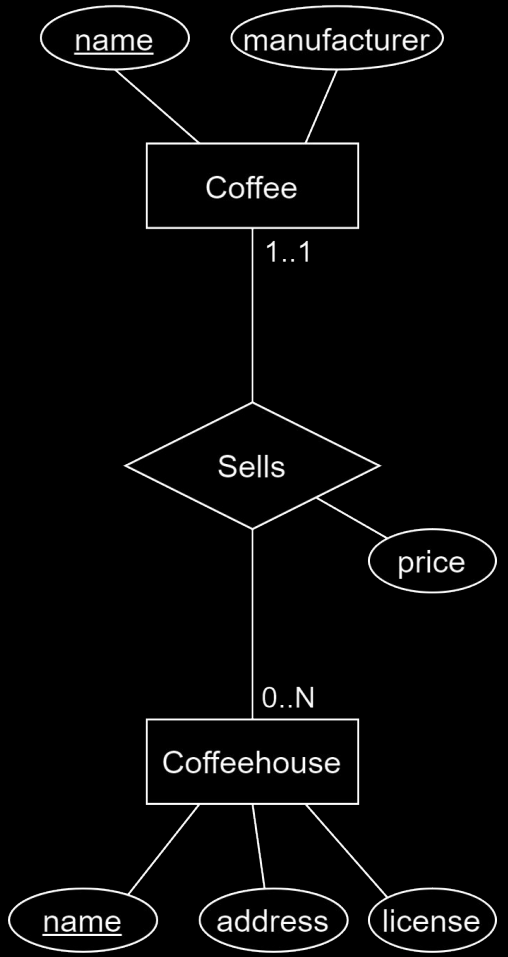
```
Coffees(name, manf)
Coffeehouses(name, addr, license)
Sells(coffeehouse, coffee, price)
```

- Write SQL DDL to create the tables
  - How many tables?

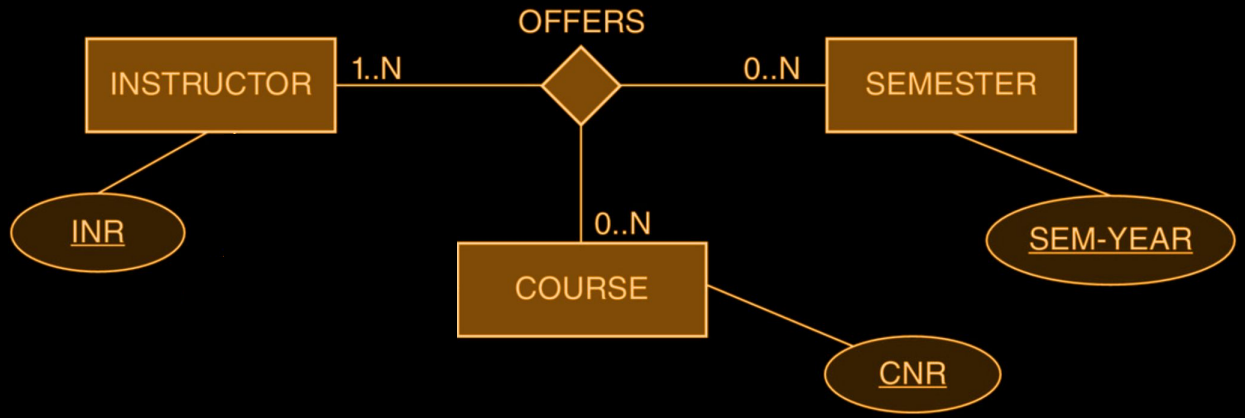


Exercise Answer DDL

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

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

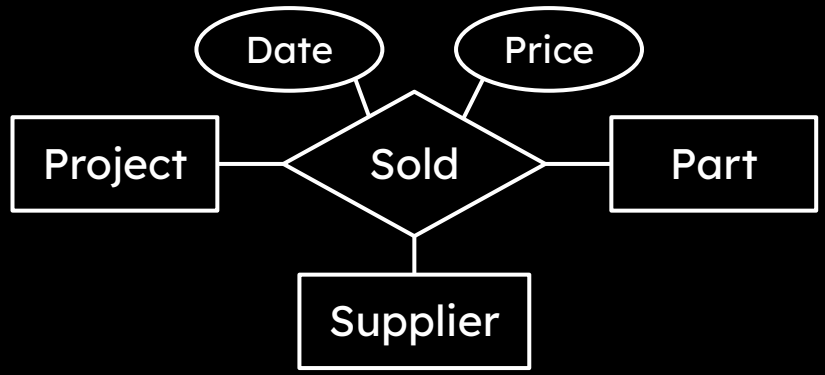


Ternary Relationships (and Beyond)



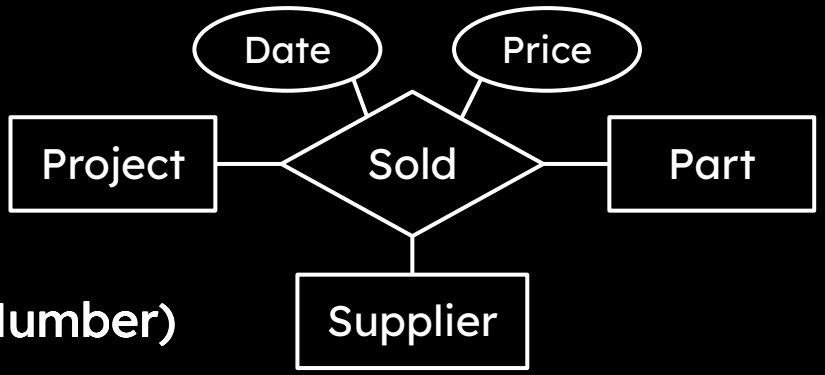
- An Instructor can offer many Courses during a Semester
- A Course offered in a given Semester must have at least one Instructor involved
- An Instructor can offer a given Course for many Semesters

Example: DLL of Ternary Relationship




```
CREATE TABLE Sold (
    ProjID INT, -- role
    SupplierID INT, -- role
    PartNumber INT, -- role
    Date DATE NOT NULL, -- attribute
    Price FLOAT 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)
)
```

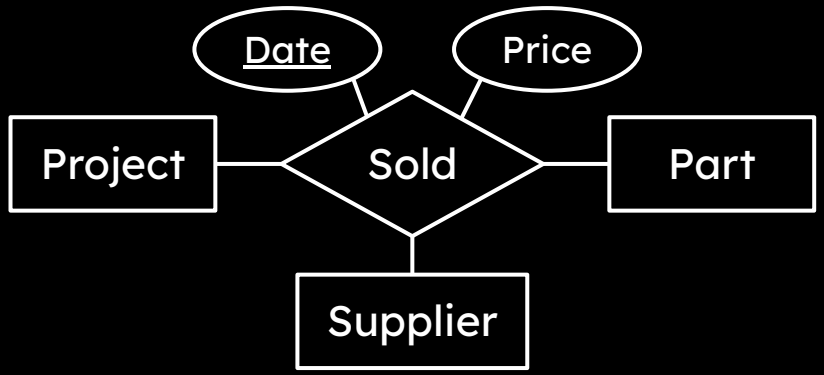
Extension: Partial Relationship Keys



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 keys*
  - Attribute is underlined

Extension: Partial Relationship Keys in SQL DDL




```
CREATE TABLE Sold (
    ProjID INT, -- role
    SupplierID INT, -- role
    PartNumber INT, -- role
    Date DATE NOT NULL, -- attribute
    Price FLOAT 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)
)
```

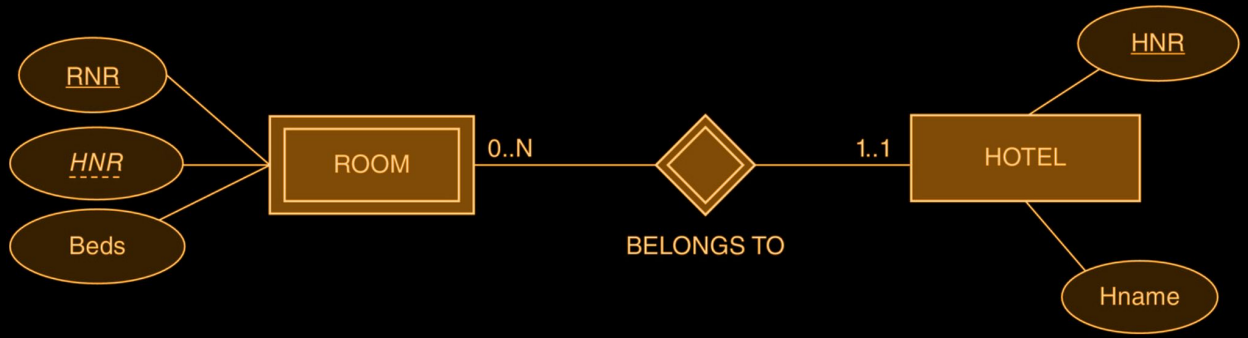
-- TODO

- Conceptual Data Modeling (ER Diagram)
  - ✓ Entities and Attributes
  - ✓ Relationships
    - ✓ Cardinalities
    - ✓ Partial Relationship Keys
  - ○ Weak Entities
  - ○ Aggregation
  - ○ Generalization/Specialization
  - ○ Categorization

- Translation to SQL DDL

Weak Entity Types

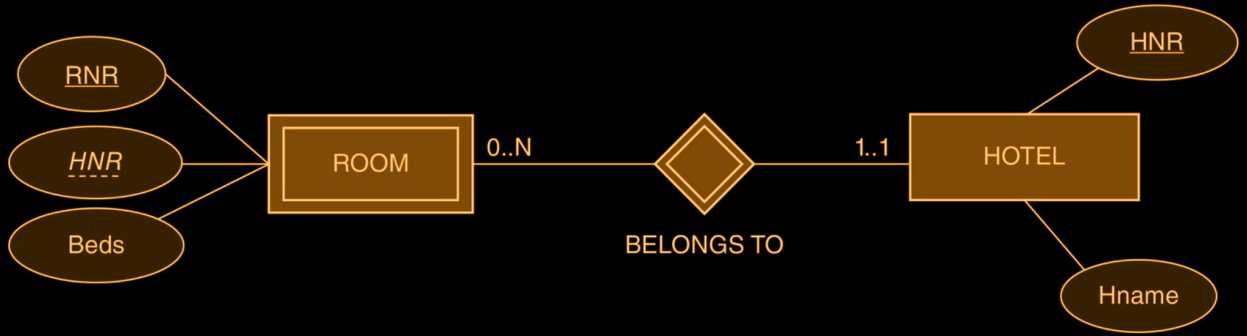
- Weak entities belong to another entity
  - They do not have a proper key, but include “parent” key
  - They have a 1 .. 1 participation in the relationship
  - If “parent” is deleted, so is the “child”
- Representation
  - Double outlines (entity, relationship)
  - Parent key is underlined with dashes

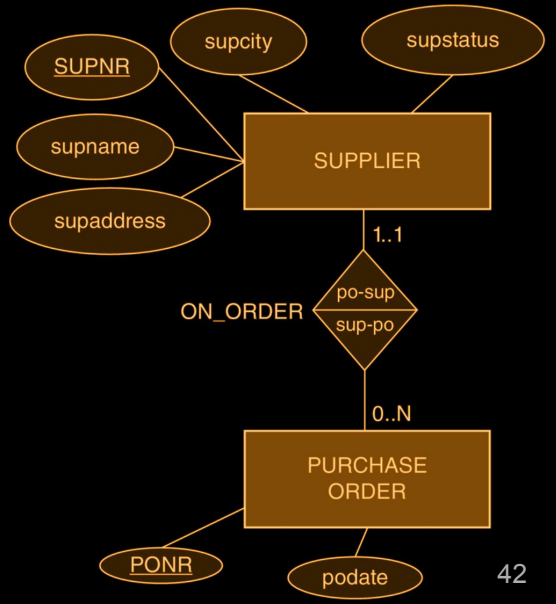


Weak Entity Types vs 1..1 Relationship Types

- 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!



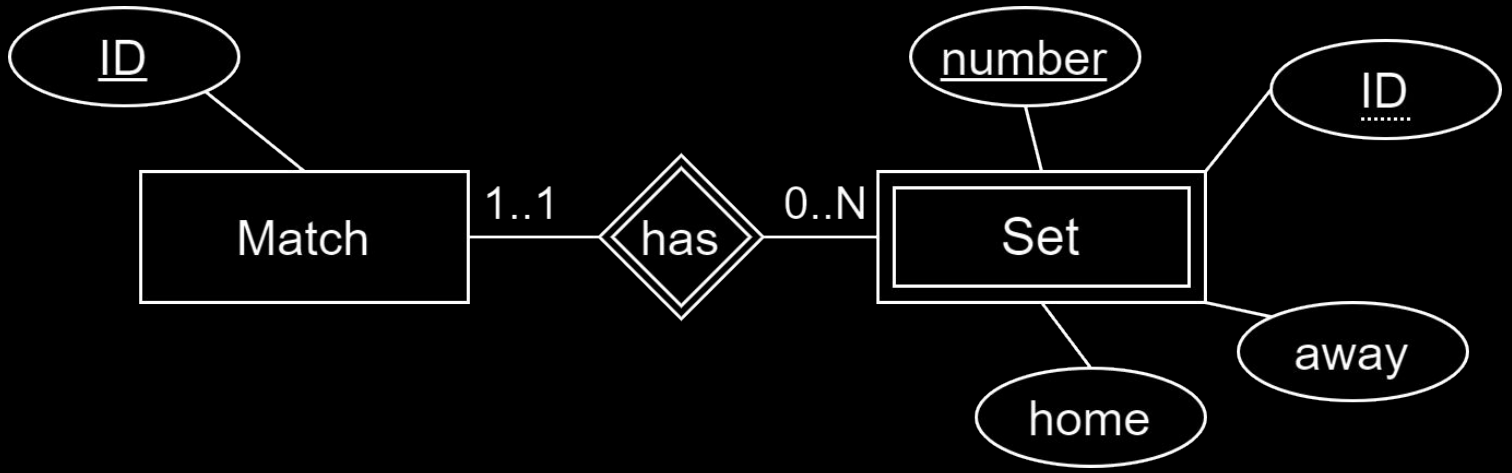


Practice Weak Entities

- Each Volleyball Match consists of Sets, each set has a number, home score and away score

Practice Weak Entities

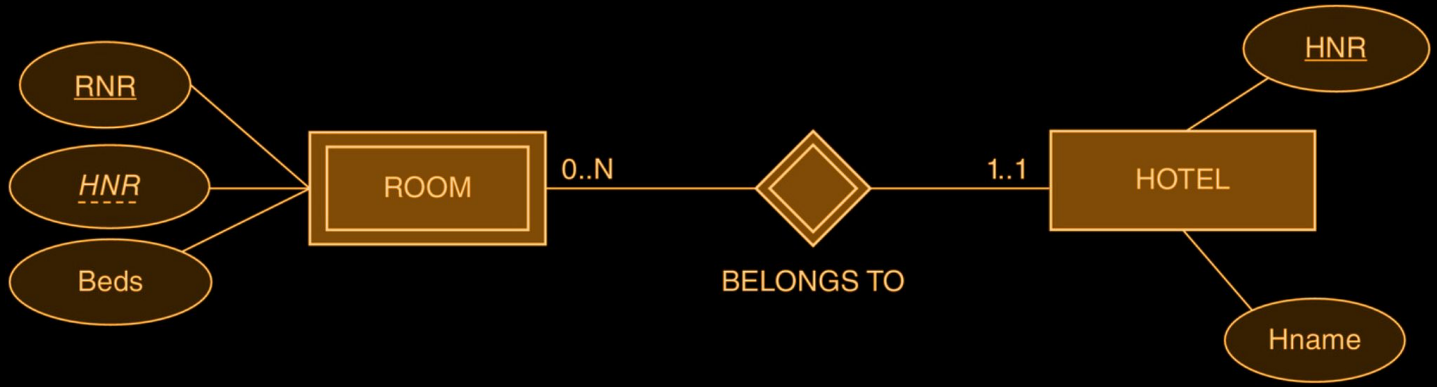
- Each volleyball match consists of sets, each with set number, home score and away score



Weak Entities in SQL DDL

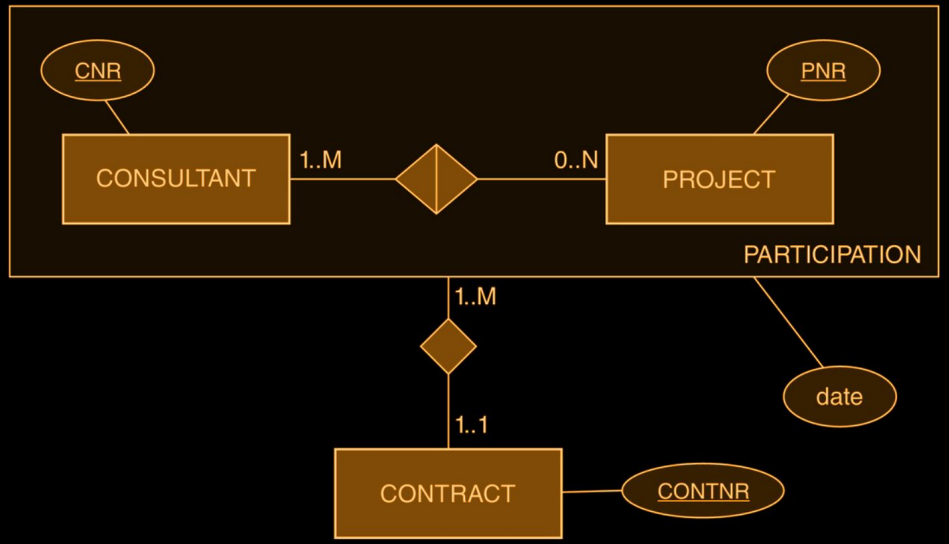
- Create a new table referencing the entity table
  - Primary key is parent key + partial key
  - Very similar to multi-valued attributes

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

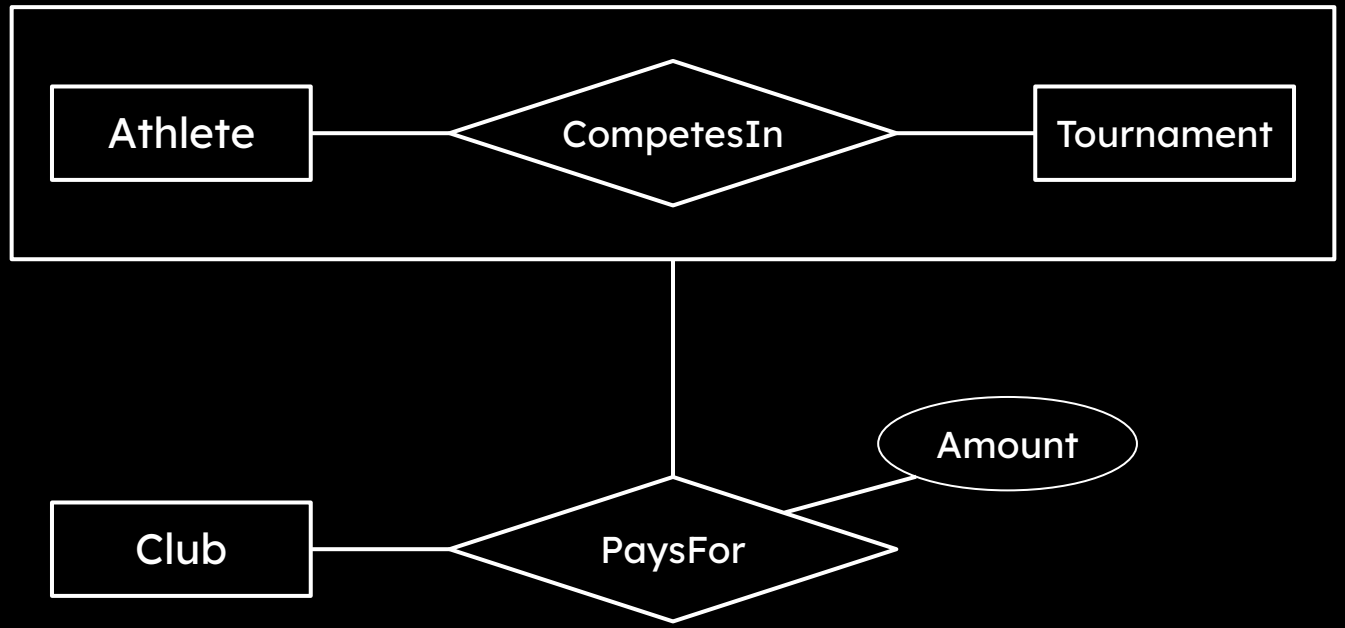


Aggregation

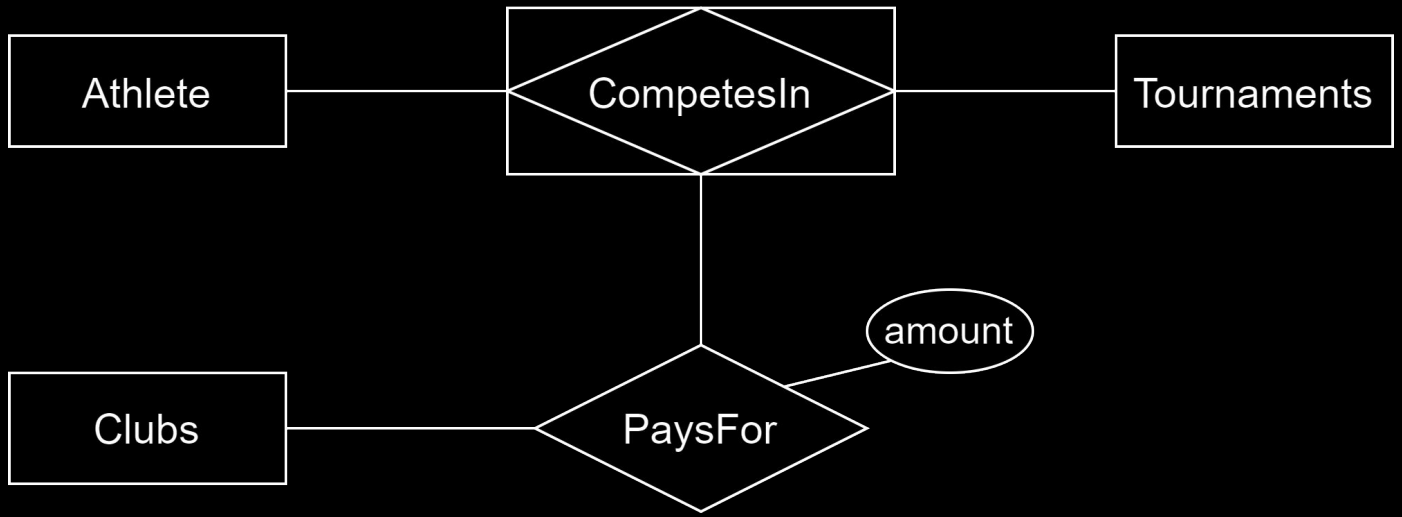
- Sometimes we need a relationship to a relationship
- Aggregation allows us to “convert” relationship types to entity types!
- Typical use: Monitoring, payments, contracts



Relationship → Entity

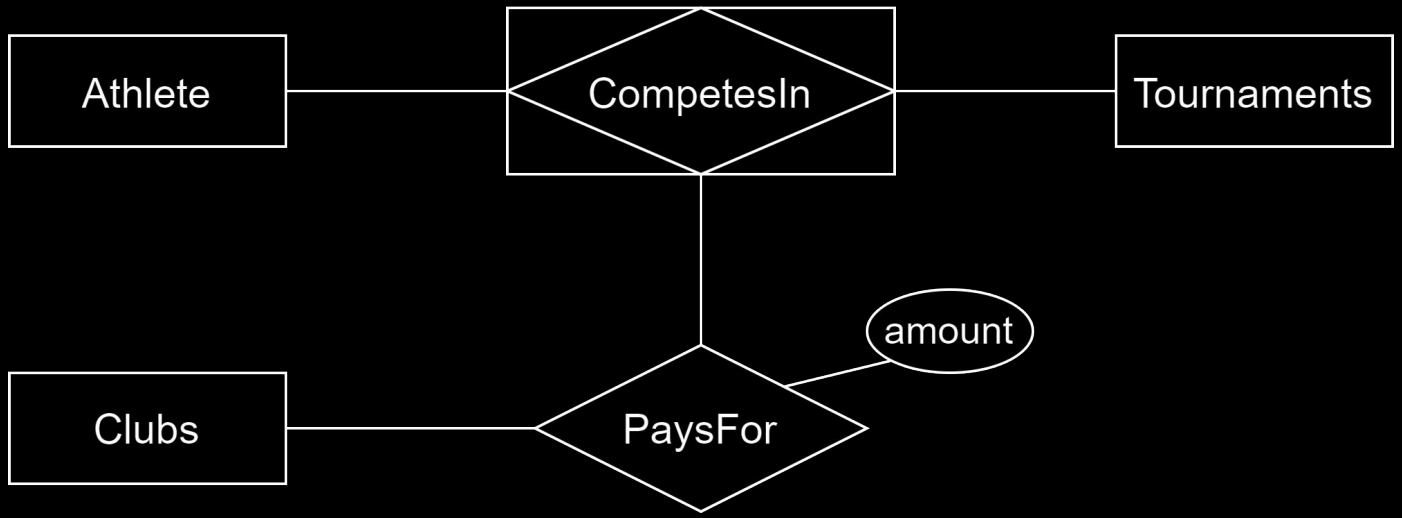


Relationship → Entity



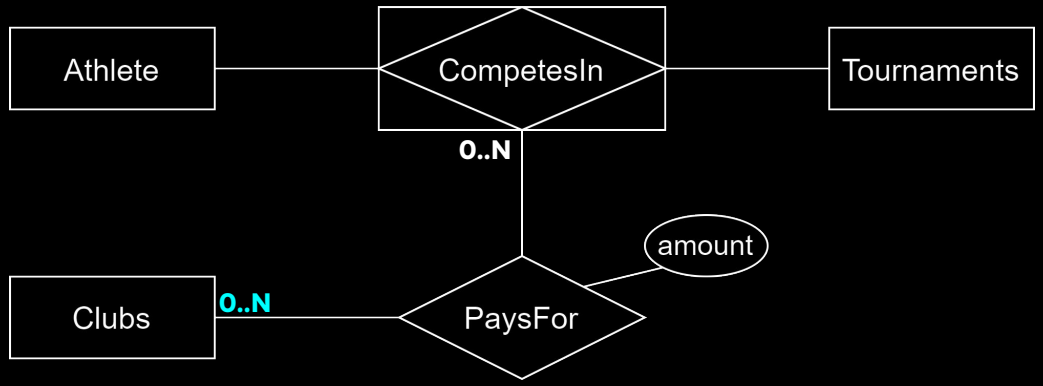
Translation to SQL DDL

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



Translation to SQL DDL: 0..N

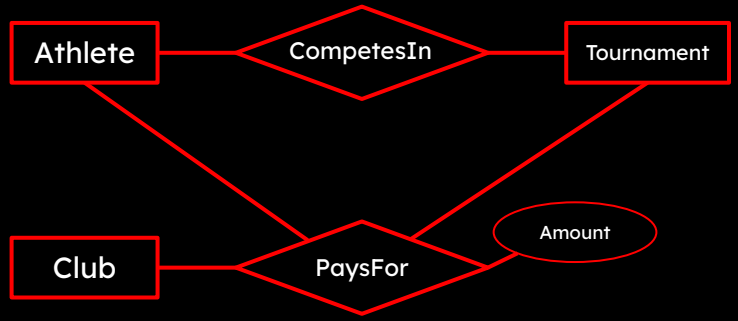
- Option 1: Use existing relationship key

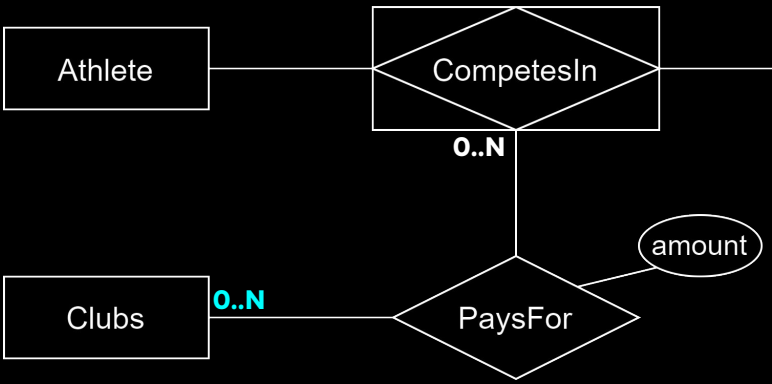



```sql
CREATE TABLE CompetesIn (
    AID INT REFERENCES Athlete,
    TID INT REFERENCES Tournament,
    PRIMARY KEY (AID, TID)
);

CREATE TABLE PaysFor (
    AID INT,
    TID INT,
    CID INT REFERENCES Clubs,
    amount INT NOT NULL,
    FOREIGN KEY (AID, TID)
    REFERENCES CompetesIn (AID, TID),
    PRIMARY KEY (AID, TID, CID)
);
```

Entities and Aggregation: Impact on Tables

- A common error is to use FKs to entity tables

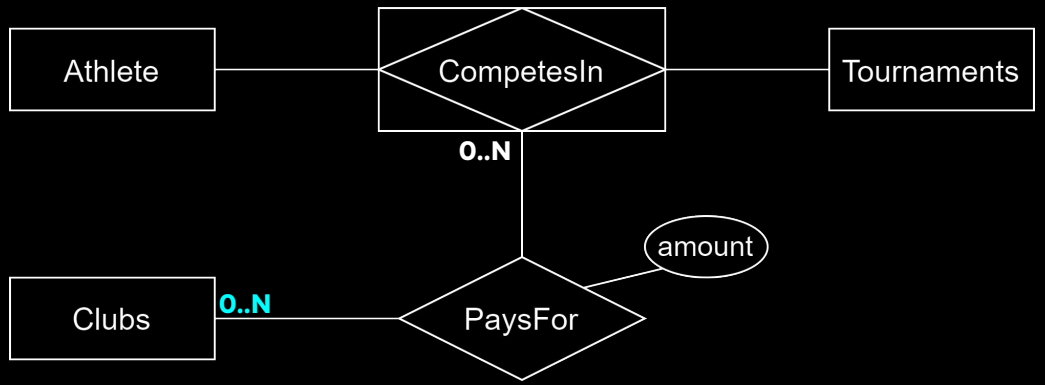





```
CREATE TABLE PaysFor (
    AID INT REFERENCES Athlete,
    TID INT REFERENCES Tournament,
    CID INT REFERENCES Clubs,
    amount INT NOT NULL,
    FOREIGN KEY (AID, TID)
    REFERENCES CompetesIn (AID, TID),
    PRIMARY KEY (AID, TID, CID),
);
```

Translation to SQL DDL: 0..N

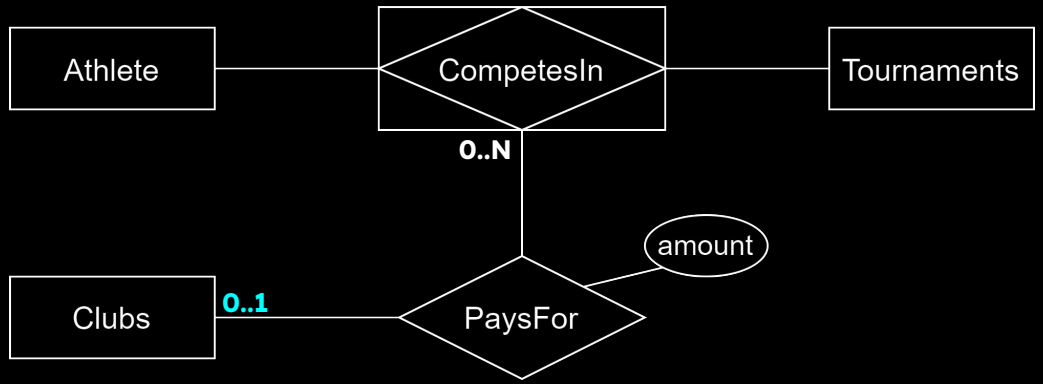
- Option 2: Create a new relationship key
  - Common error is to forget the existing key




```sql
CREATE TABLE CompetesIn (
    CIID INT ALWAYS GENERATED AS IDENTITY
     PRIMARY KEY,
     AID INT REFERENCES NOT NULL Athletes,
     TID INT REFERENCES
     NOT NULL Tournament,
    UNIQUE (AID, TID)
);

CREATE TABLE PaysFor (
    CIID INT REFERENCES CompetesIn,
     CID INT REFERENCES Clubs,
     amount INTEGER NOT NULL,
     PRIMARY KEY (CIID, CID)
);
```

Translation to SQL DDL: 0..1

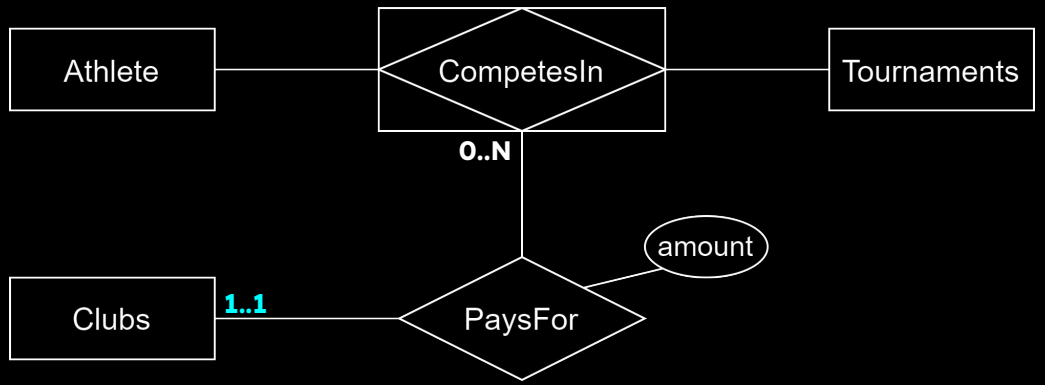
- An Athlete competing in a Tournament can receive payment from maximum one Club
  - Change the Primary Key




```sql
CREATE TABLE CompetesIn (
    AID INT REFERENCES Athletes,
    TID INT REFERENCES Tournament,
    PRIMARY KEY (AID, TID)
);

CREATE TABLE PaysFor (
    AID INT,
    TID INT,
    CID INT NOT NULL REFERENCES Clubs,
    amount INT 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 INT REFERENCES Athletes,
    TID INT REFERENCES Tournament,
    CID INT NOT NULL REFERENCES Clubs,
    amount INT NOT NULL
    PRIMARY KEY (AID, TID)
);
```

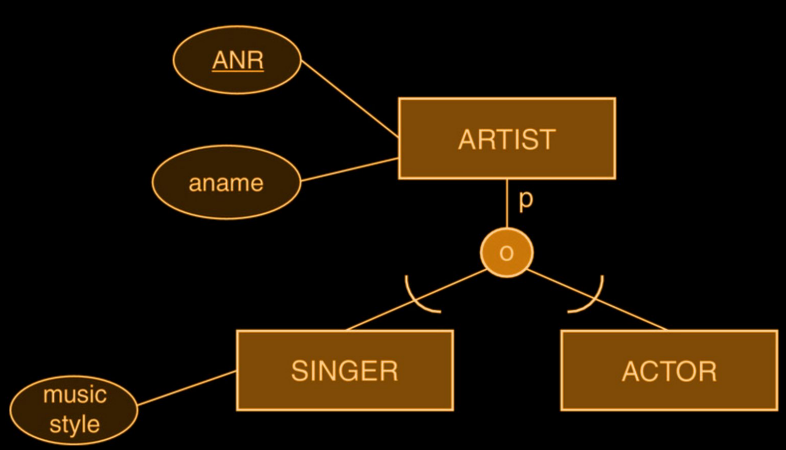
-- TODO -> DONE

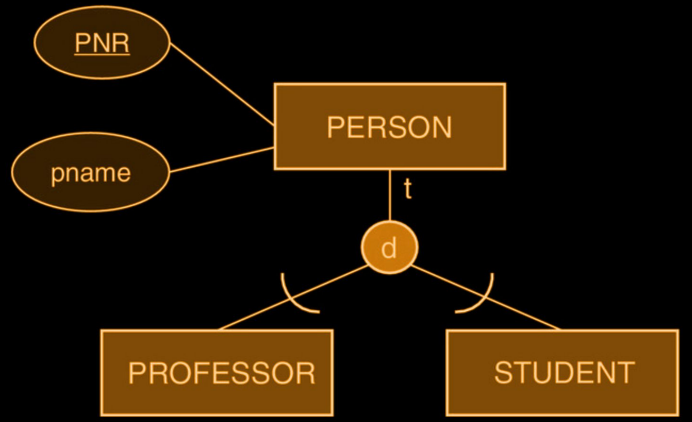
- ✓ Conceptual Data Modeling (ER Diagram)
  - ✓ Entities and Attributes
  - ✓ Relationships
    - ✓ Cardinalities
    - ✓ Partial Relationship Keys
  - ✓ Weak Entities
  - ✓ Aggregation
  - ○ Generalization/Specialization
  - ○ Categorization

- ● Translation to SQL DDL

Generalization/Specialization

- Like inheritance in Object-Oriented Programming (Java/C#/etc.)
- 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

- 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

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

Person

SSN	*Name*	*DOB*
1234	Mary	1950

Employee

SSN	*Department*	*Salary*
1234	Accounting	35000

Student

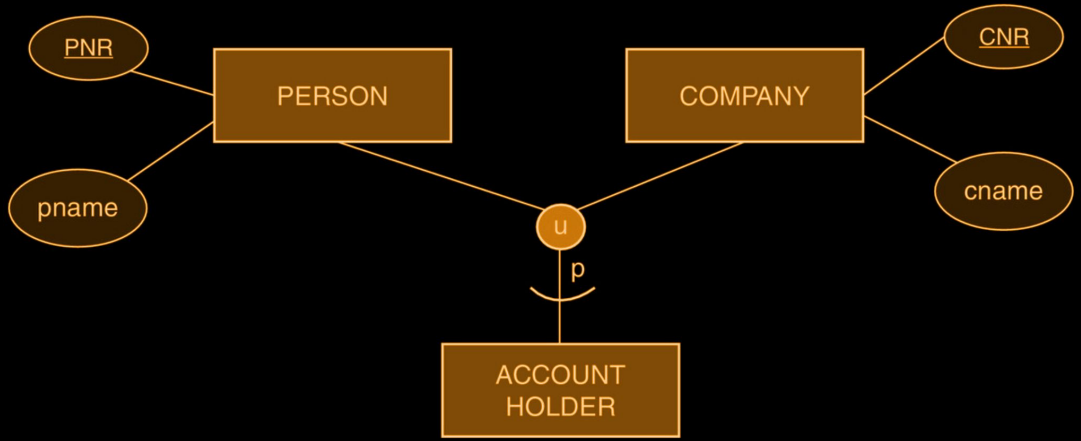
SSN	*GPA*	*StartDate*
1234	3.5	1997

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

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

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)!

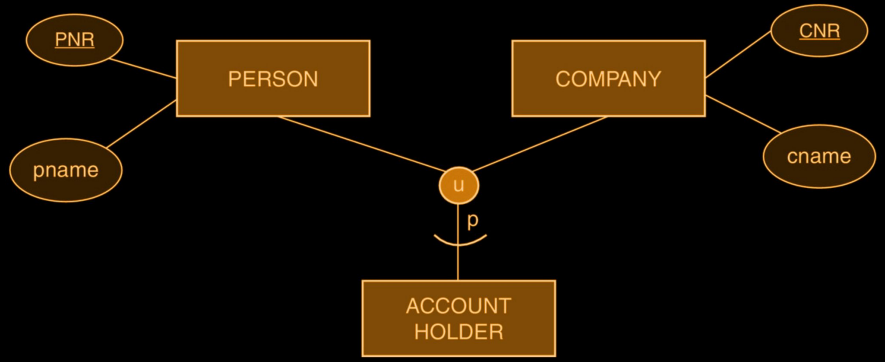


Categorization in SQL DDL

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

```sql
CREATE TABLE AccountHolder (
    AcctID INT PRIMARY KEY
);

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

NOT NULL = Total
NULL = Partial

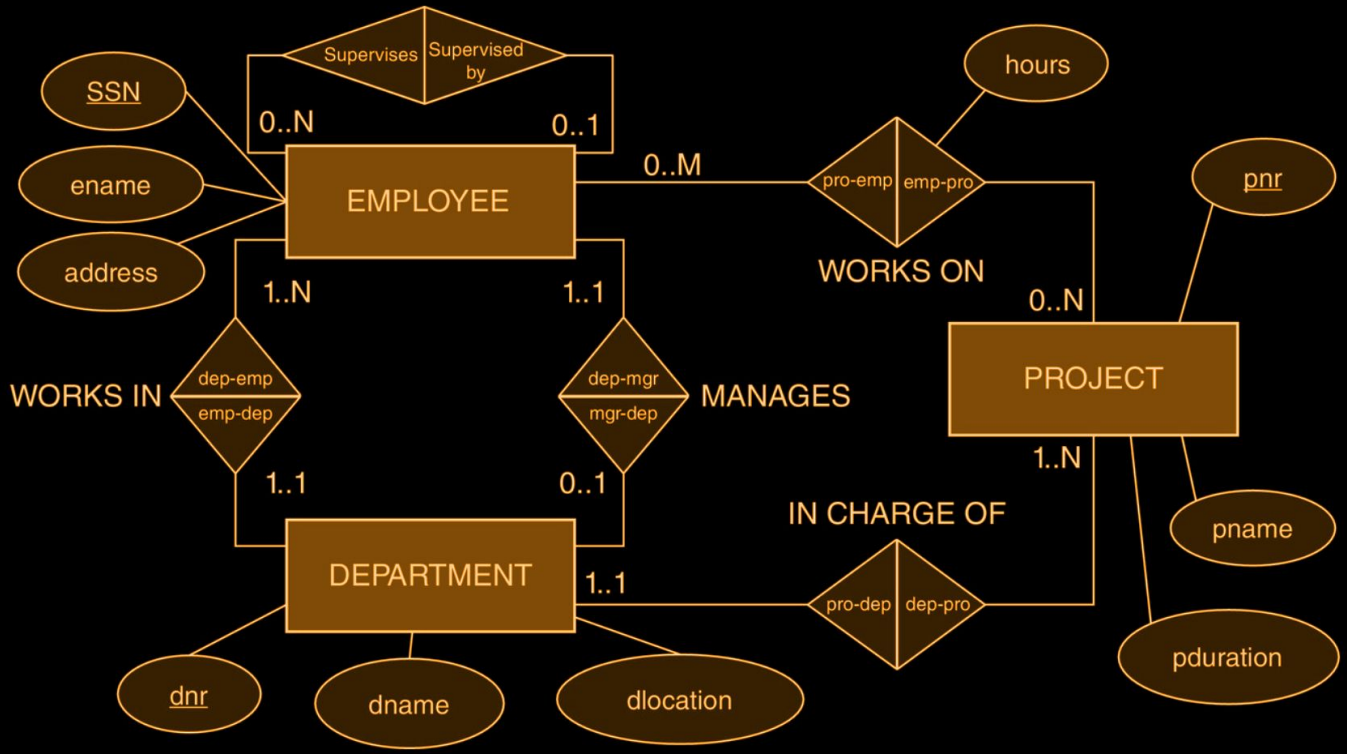


-- TODO -> DONE

- ✓ Conceptual Data Modeling (ER Diagram)
  - ✓ Entities and Attributes
  - ✓ Relationships
    - ✓ Cardinalities
    - ✓ Partial Relationship Keys
  - ✓ Weak Entities
  - ✓ Aggregation
  - ✓ Generalization/Specialization
  - ✓ Categorization

- ✓ Translation to SQL DDL

Exercise: How many tables? What are the keys?



Requirements to ERD

- Nouns = entities
  - Descriptive elements = attributes

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

- Example:
  - 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 Design

- 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 (Lecture 4) can be used to handle some problems
  - Normalization (Lecture 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

one-to-zero
one-to-one
one-to-many
many-to-many

ER diagram 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!!!

Next Time in IDBS...

Introduction to Database Systems IDBS - Fall 2024

Lecture 6

Normalization

Readings: PDBM 6.2-6.4

Eleni Tzirita Zacharatou