



SQL Wrap Up & Introduction to DB Design

Week 4

Agent 7 ... which languages? skills?

```
select * from languagerel where agent_id = 7
```

lang_id	agent_id
3	7
14	7
19	7
20	7

agent 7 speaks 4 languages

```
select * from skillrel where agent_id = 7
```

skill_id	agent_id
17	7
42	7

agent 7 has 2 skills

Intersect vs. Intersect ALL

```
SELECT agent_id FROM languagerel WHERE agent_id = 7  
INTERSECT
```

```
SELECT agent_id FROM skillrel WHERE agent_id = 7
```

agent_id
7

Agent 7 has (at least one) language
and (at least one) skill

```
SELECT agent_id FROM languagerel WHERE agent_id = 7  
INTERSECT ALL
```

```
SELECT agent_id FROM skillrel WHERE agent_id = 7
```

agent_id
7
7

Agent 7 has at most 2 of either
skills or languages or both

Except vs. Except all

```
SELECT agent_id FROM languagerel WHERE agent_id = 7  
EXCEPT
```

```
SELECT agent_id FROM skillrel WHERE agent_id = 7
```

No rows found.

Agent 7 has both languages
and skills

```
SELECT agent_id FROM languagerel WHERE agent_id = 7  
EXCEPT ALL
```

```
SELECT agent_id FROM skillrel WHERE agent_id = 7
```

agent_id
7
7

Agent 7 has two more languages
than skills

Except vs. Except all (cont.)

```
SELECT agent_id FROM languagerel WHERE agent_id = 7  
EXCEPT ALL
```

```
SELECT agent_id FROM skillrel WHERE agent_id = 7
```

(repeated from previous slide)

Agent 7 has two more languages
than skills

agent_id
7
7

```
SELECT agent_id FROM skillrel WHERE agent_id = 7  
EXCEPT ALL
```

```
SELECT agent_id FROM languagerel WHERE agent_id = 7
```

No rows found.

Agent 7 has fewer skills than languages.

Union vs. Union ALL

```
SELECT agent_id FROM languagerel WHERE agent_id = 7  
UNION
```

```
SELECT agent_id FROM skillrel WHERE agent_id = 7
```

Agent 7 has languages or skills

agent_id
7

```
SELECT agent_id FROM languagerel  
WHERE agent_id = 7
```

```
UNION ALL
```

```
SELECT agent_id FROM skillrel  
WHERE agent_id = 7
```

Agent 7's languages + skills = 6

agent_id
7
7
7
7
7
7

Relational Algebra (as defined mathematically by Codd) vs. SQL

	Relational Algebra	SQL
$\sigma, \Pi, \bowtie, X, \rho$ for relations as sets	Yes	YES (with DISTINCT) Basic Select-From-Where
$\sigma, \Pi, \bowtie, X, \rho$ for tables as bags	NO	YES (it's the default) Basic Select-From-Where
$\cap, \cup, -$ for relations as sets	Yes	YES (UNION, INTERSECT, EXCEPT)
$\cap, \cup, -$ for tables as bags	NO	YES (UNION ALL, INTERSECT ALL, EXCEPT ALL)
GROUP BY/HAVING	NO	YES
Aggregate operators	NO	YES
\div (divide operator)	YES	NO (not directly)

Note that in a DBMS, there are algebraic operators for bags/sets and for grouping and aggregates. We use (only) the original rel. alg. operators defined for sets.

Hard to get the effect of divide in SQL

$(\pi_{\text{Owner, Type}} \text{Account}) \div \text{Account-types}$

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Maier		

Account-types	Type
	checking
	savings

```

SELECT A.Owner
FROM Account A
WHERE NOT EXISTS
  ((SELECT T.Type
    FROM Account-types T)
  EXCEPT
  (SELECT A1.Type
    FROM Account A1
    WHERE A1.Owner = A.Owner))
    
```

Integrity Constraints (ICs)

- An IC describes conditions that every *legal instance* of a relation must satisfy.
 - Inserts/deletes/updates that violate IC's are typically disallowed.
 - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)

Types of ICs

Domain constraints, primary key constraints, foreign key constraints, general constraints.

- *Domain constraints*: Field values must be of right type. Always enforced.

General Constraints

- Useful when more general ICs than keys are involved.
- Can use queries to express constraint.
- Constraints can be named, then dropped, suspended, etc.
- A very powerful mechanism

```
CREATE TABLE Account (...  
  
    PRIMARY KEY Number,  
    CHECK (CBalance <= CAmount))
```

```
CREATE TABLE Withdrawal (...  
  
    PRIMARY KEY (Tran-id),  
    CHECK (Amount <=  
        SELECT Balance FROM ACCOUNT A  
        WHERE A.Number = Account))
```

Domain Constraints

- You can create your own domains and use them in creating tables
- Joins are allowed between domains of the same BASE TYPE (Integer, in this case).
- SQL99 defines DISTINCT TYPES that can avoid this problem

```
CREATE DOMAIN AmtDom Integer
CHECK(VALUE >=1000 AND
       VALUE <=100000)
```

```
CREATE TABLE Account
( ..., CAmount AmtDom, ...)
```

Triggers

- Trigger: procedure that executes if specified changes occur to the DBMS
- Three parts:
 - Event (activates the trigger)
 - Condition (tests whether the triggers should run)
 - Action (what happens if the trigger runs)

Triggers: Example

```
CREATE TRIGGER IncrCredit ON Deposit AFTER INSERT AS
  IF new.Amount >= 10000
    UPDATE Account A
    SET A.CRating = A.CRating+1
    WHERE A.Account = new.Number
```

Note the special variable **new** that represents the newly inserted row.

Views, Levels of Abstraction and Data Independence

One database often supports multiple applications, which might have slightly different pictures of the world.

Views help accommodate this variation without storing redundant data.

Views

- A *view* is a named query stored in the database
 - Think of it as a table definition for future use
- Example view definition:

```
CREATE VIEW GStudent AS SELECT S.*  
FROM Student S WHERE s.gpa >= 2.5
```

- A view can be used like a *base table*, in a SELECT query or in another view. A kind of macro.

Example view use: Simpler queries

- Completed(StudID, Course)
- Queries are often about “good” students only

```
SELECT S.name, S.phone  
FROM GStudent S NATURAL JOIN Completed C  
WHERE C.course = 'CS386';
```

- Now it's easier to write the query.

Views for Security

Suppose schema is

Student(studID, name, address, major, gpa)

- This is a view of the Student table without the gpa field.

```
CREATE VIEW SecStudent AS
```

```
SELECT studID, name, address, major
```

```
FROM student
```

Views for Extensibility

- A company's database includes a relation: Part (PartID: Char(4), weight:real,...)
- Weight is stored in pounds
- Company is purchased by a firm that uses metric weights
- Databases must be integrated and use Kg.
- But there's much old software using pounds.
- Solution: views!

But there's one problem with views

- Views cannot always be updated unambiguously
- Consider Student(StudID, gpa, major,...)

```
CREATE VIEW majorgpa AS
SELECT major, AVG(gpa)
FROM Student
GROUP BY major
```

Majorgpa major gpa

CS	3.5
ECE	3.5

Updatability of views

- I want to change the GPA of CS majors from 3.5 to 3.6 .
- How can I do that?

The good news

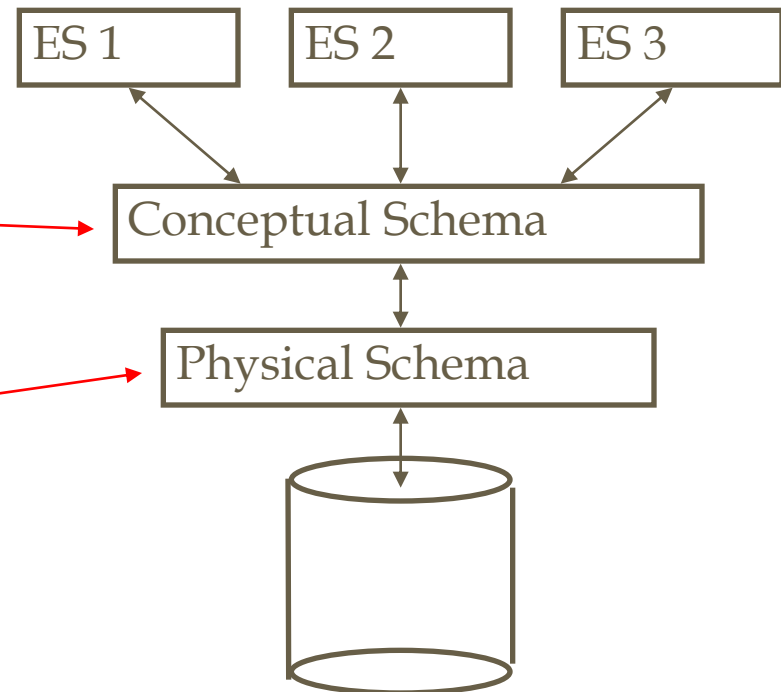
- A view can be updated if
 - It is defined on a single base table
 - Using only selection and projection
 - No aggregates
 - No DISTINCT

Levels of Abstraction

External view; user
and data designer

Logical storage; data
designer

Physical storage; DBA



Physical Schema

The *physical schema* is a description of how the data is physically stored in the database. It includes

- Where the data is located
- File structures
- Access methods
- Indexes

The physical schema is managed by the DBA.

Conceptual Schema

The conceptual schema is a logical description of what data is in the database. It consists of the schemas we have described with CREATE TABLE statements. It is managed by the data designer.

External Schemas

Each external schema is a combination of base tables and views, tailored to the needs of a single user. It is managed by the data designer and the user.

Data Independence

- A database model possesses *data independence* if application programs are protected from changes in the conceptual and physical schemas.
- Why is this important? Everything changes.
- How does the relational model achieve logical (conceptual) data independence?

Data Independence (ctd.)

- How does the relational model achieve physical data independence?
 1. Conceptual level contains no physical info
 2. SQL can program against the conceptual level
- Earlier DBMSs (network, hierarchical) did not have these properties.
 - Their languages had physical properties embedded in them.

Summary

- A view is a stored query definition
- Views can be very useful
 - Easier query writing, security, extensibility
- But not all views can be updated unambiguously
- Three levels of abstraction in a relational DBMS
 - Yields data independence: logical and physical

How do you like to see the Spy Database? As tables?

Agent(agent_id, first, middle, last, address, city, country, salary, clearance_id)

Mission(mission_id, name, access_id, team_id, mission_status)

Affiliation(aff_id, title, description)

AffiliationRel(aff_id, agent_id, affiliation_strength)

LanguageRel(lang_id, agent_id)

Language(lang_id, language)

SecurityClearance(sc_id, sc_level, description)

Skill(skill_id, skill)

SkillRel(skill_id, agent_id)

Team(team_id, name, meeting_frequency)

TeamRel(team_id, agent_id)

How do you like to see the Spy Database? As tables with foreign keys?

Agent(agent_id, first, middle, last, address, city, country, salary, clearance_id)

Mission(mission_id, name, access_id, team_id, mission_status)

Affiliation(aff_id, title, description)

AffiliationRel(aff_id, agent_id, affiliation_strength)

LanguageRel(lang_id, agent_id)

Language(lang_id, language)

SecurityClearance(sc_id, sc_level, description)

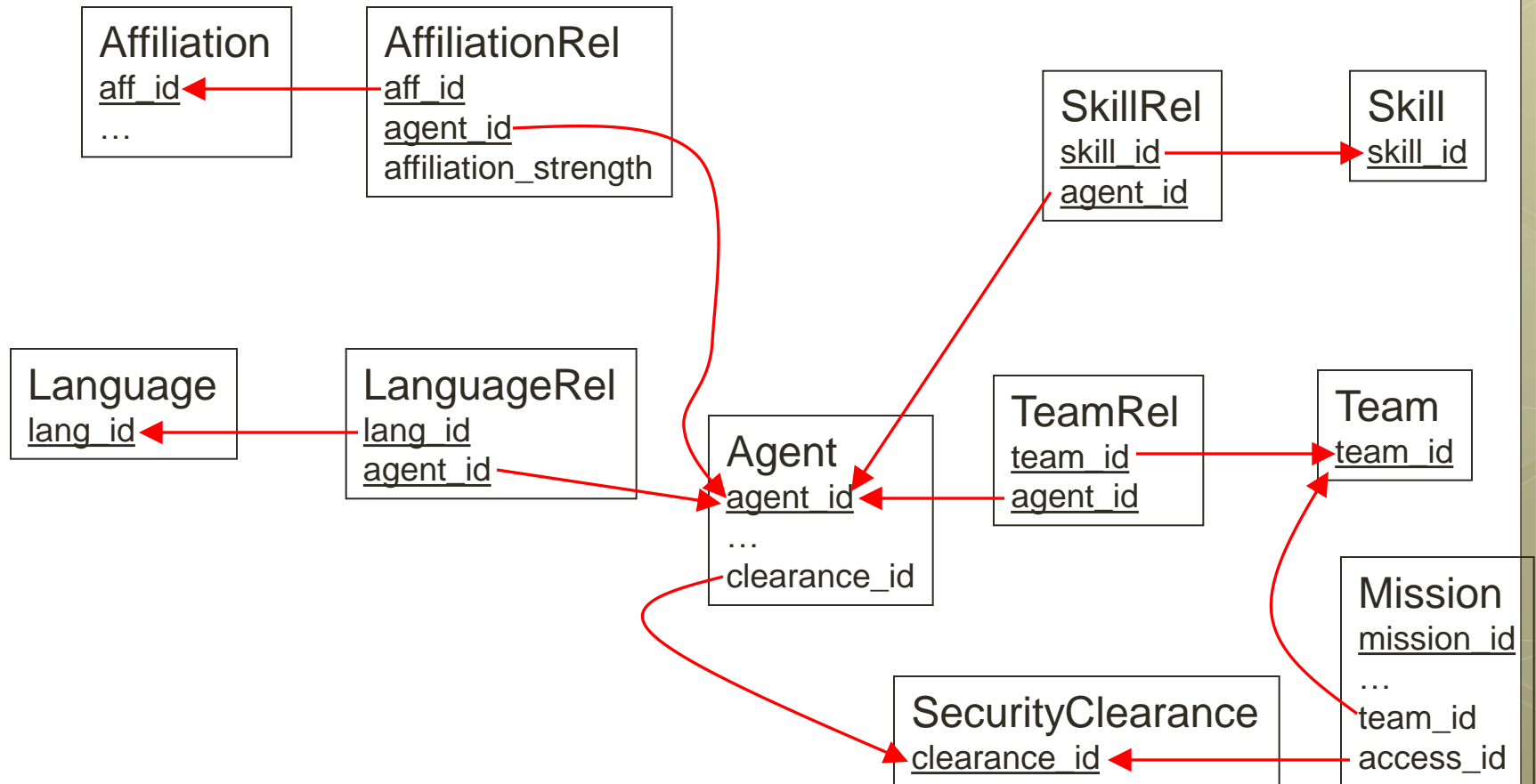
Skill(skill_id, skill)

SkillRel(skill_id, agent_id)

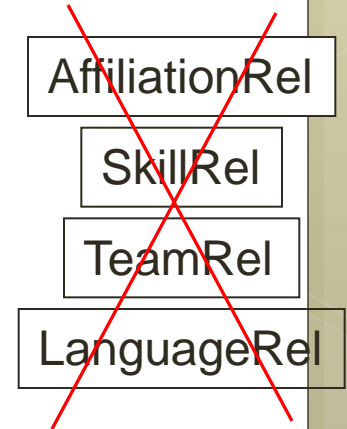
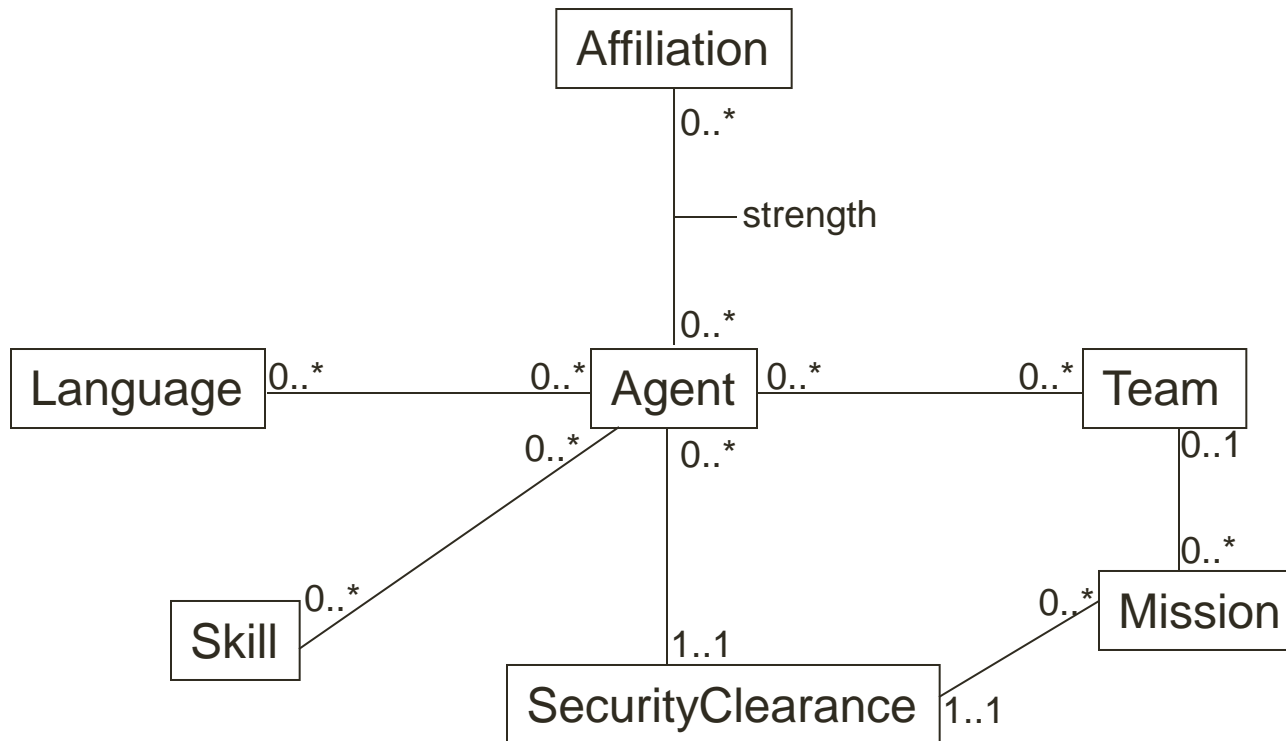
Team(team_id, name, meeting_frequency)

TeamRel(team_id, agent_id)

How do you like to see the Spy Database? MS[®] Access “Relationship Diagram” – of tables?



How do you like to see the Spy Database? Entity-Relationship Diagram (ERD)?

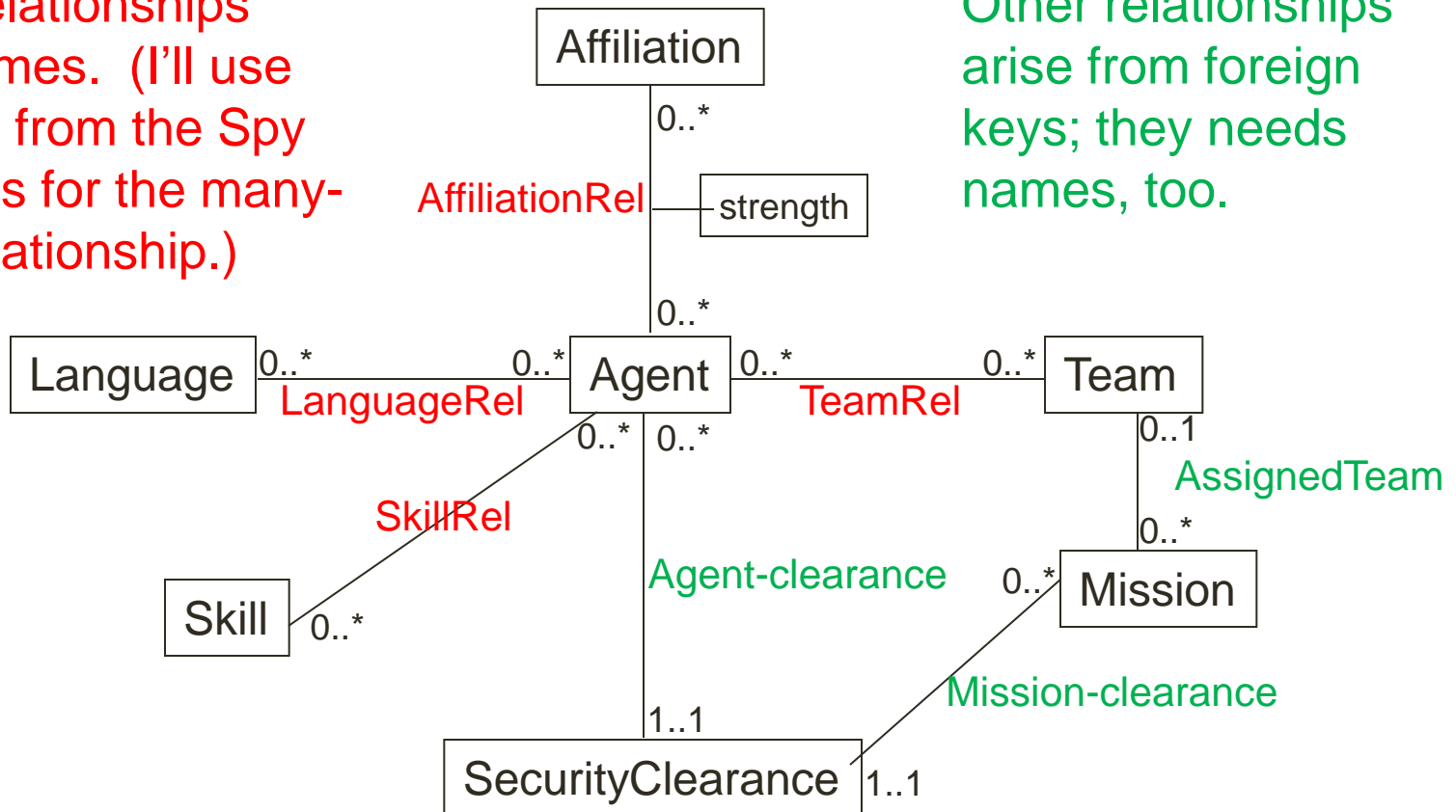


Rectangles: Entities Lines: Relationships (with cardinalities)

Entity-Relationship Diagram (ERD)

Note: Relationships need names. (I'll use the ones from the Spy DB tables for the many-many relationship.)

Other relationships arise from foreign keys; they need names, too.



Conceptual Database Design Using the Entity-Relationship (ER) Model

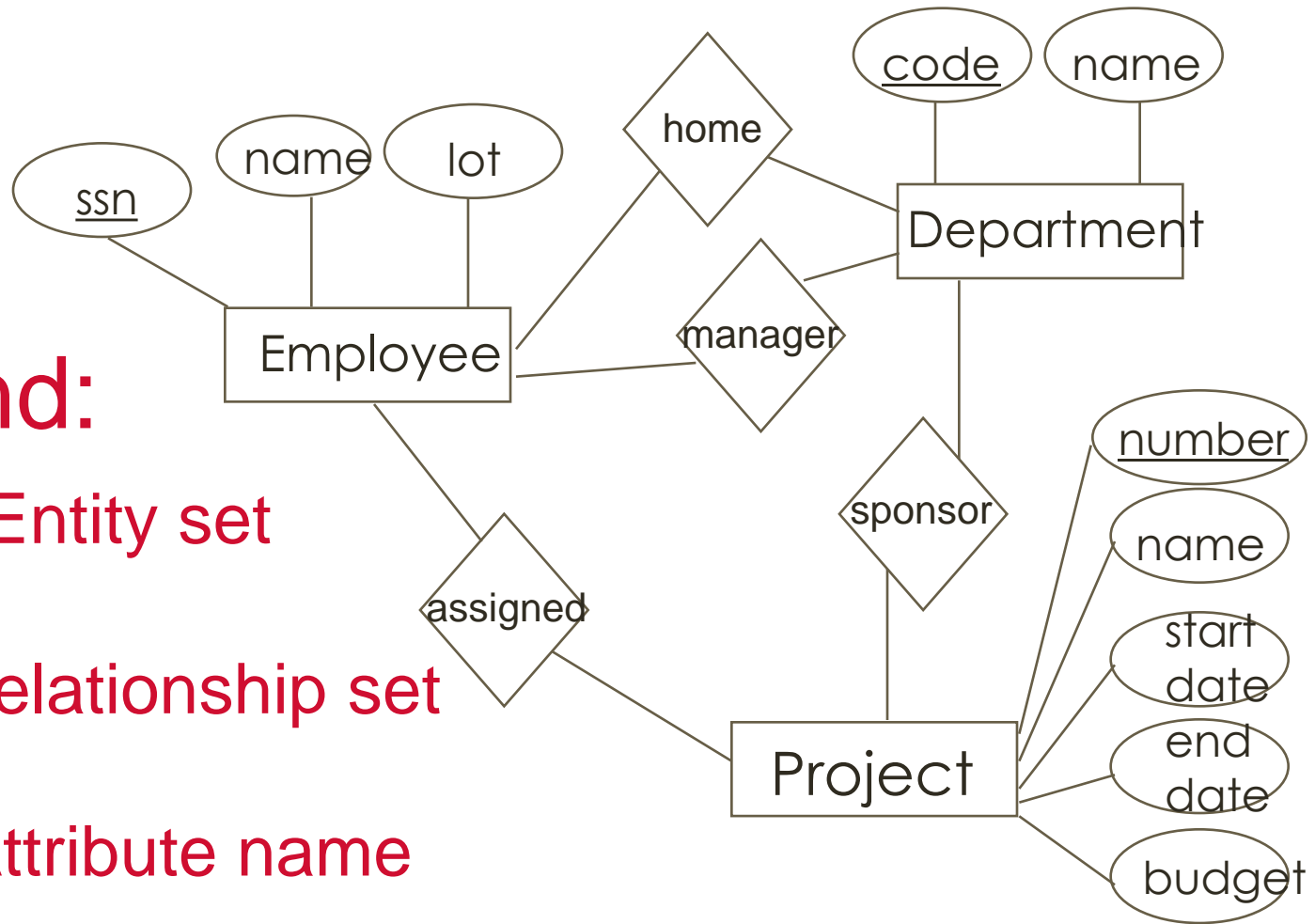
Overview of Database Design

- **Conceptual design:** (ER Model is used for this.)
 - What are the **entities** and **relationships** we need?
- **Logical design:**
 - Transform ER design to Relational Schema
- **Schema Refinement:** (Normalization)
 - Check relational schema for redundancies and related anomalies.
- **Physical Database Design and Tuning:**
 - Consider typical workloads; (sometimes) modify the database design; select file types and indexes.

Entity-Relationship Model is a different model than the Relational Model

- **Relation model** has:
 - **tables** (relations) with attributes, keys, foreign keys, domain definitions for attributes
- **Entity-Relationship model** has:
 - **Entities and entity sets** with attributes, keys, and domain definitions for attributes
 - **relationships among entities and relationship sets** with cardinality constraints (in the book they refer to key constraints and participation constraints)

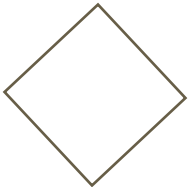
Entity-Relationship Diagram (original syntax)



Legend:



Entity set



Relationship set



Attribute name

Definitions

- *Entity*: Real-world object distinguishable from other objects. (An employee named **John Smith**, for example.)
 - An entity is described using a set of *attributes*.
- *Entity Set*: A collection of similar entities. (**All employees in the company**, for example.)

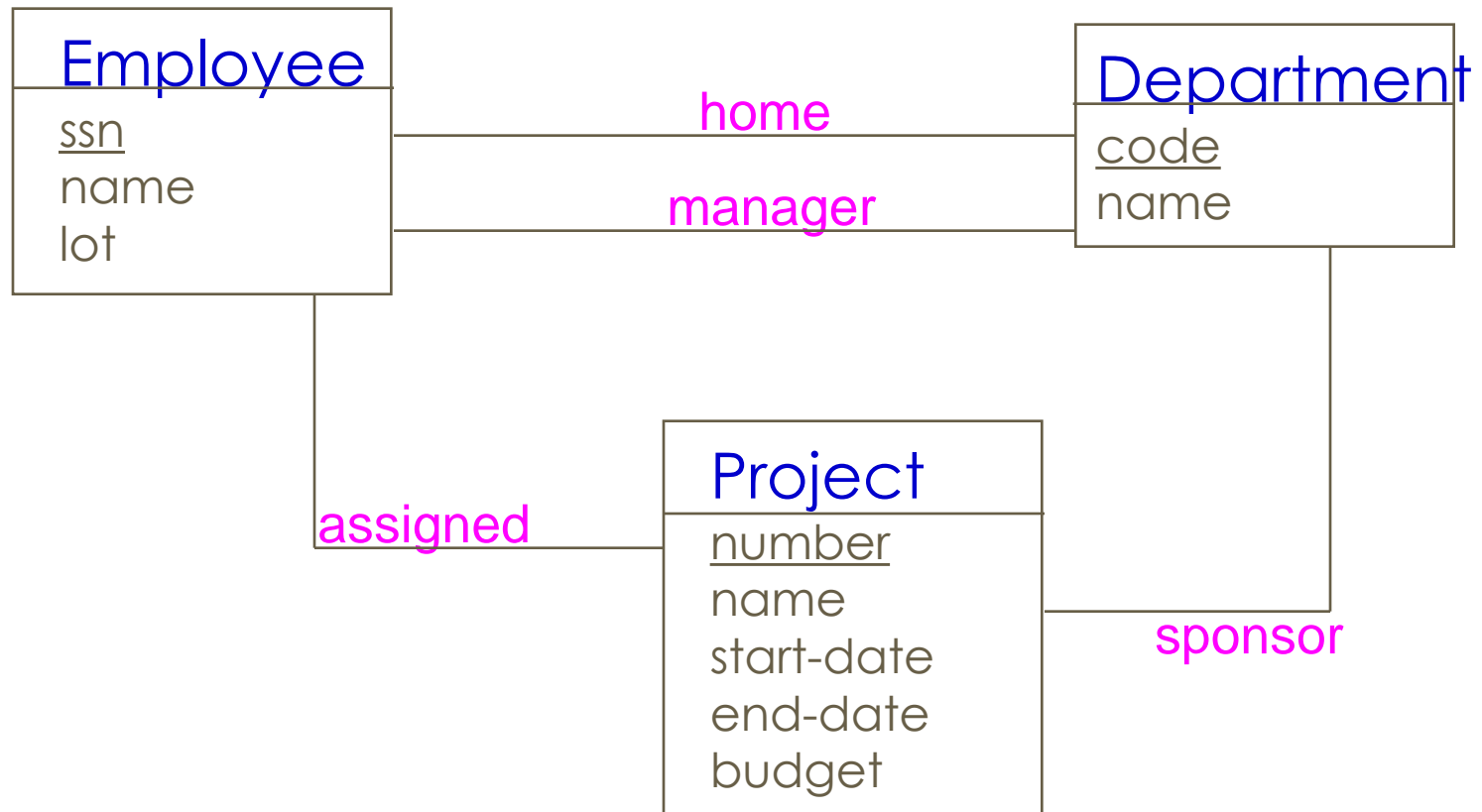
(An entity set is often referred to as just an entity if everyone knows we're talking about the schema rather than individual rows. An entity set may also be called entity type.)

Definitions

- Relationship: Association among 2 or more entities. John's **home department** is Pharmacy, for example. John is an entity. Pharmacy is an entity.
- Relationship Set: Collection of similar relationships. The **home department** relationship set consists of all indications of a home department for an employee. (A relationship set is often referred to as a relationship because everyone knows we are talking about the schema rather than individual rows. A relationship set may also be called a relationship type.)

UML version of the same E-R Diagram

UML: Unified Modeling Language – for OO Design



Equivalent Relational Schema

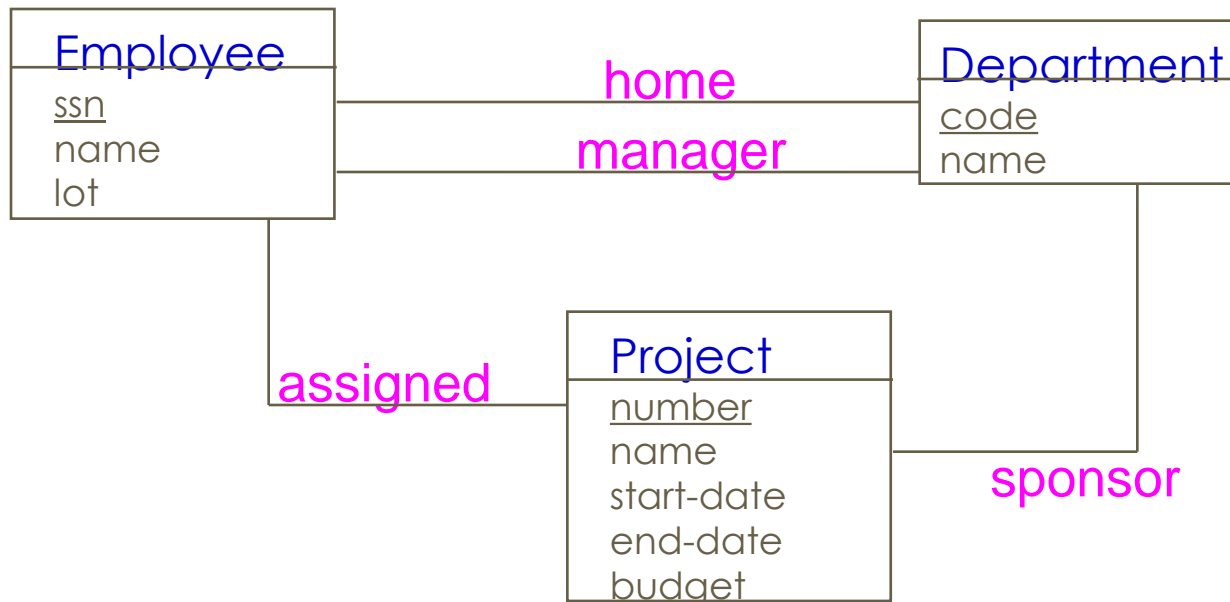
Employee (ssn, name, lot, home-dept)

Project-team(ssn, number)

Department (id, name, manager)

Project (number, name, start-date, end-date, budget, sponsor)

extra table to represent a many-to-many relationship



Equivalent Relational Schema - with foreign keys shown

Employee (ssn, name, lot, home-dept)

Project-team(ssn, number)

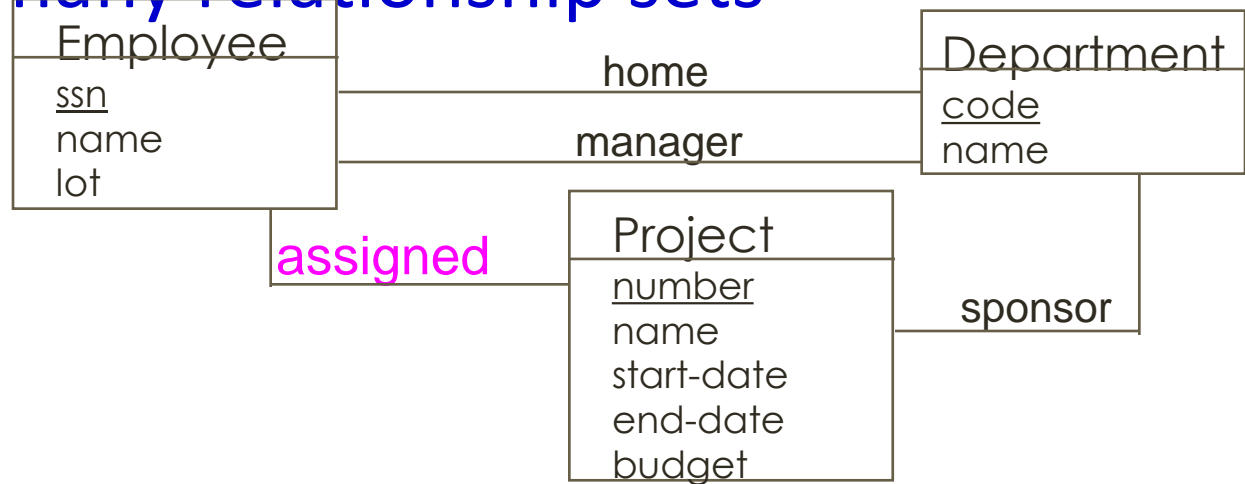
Department (id, name, manager)

Project (number, name, start-date, end-date, budget, sponsor)

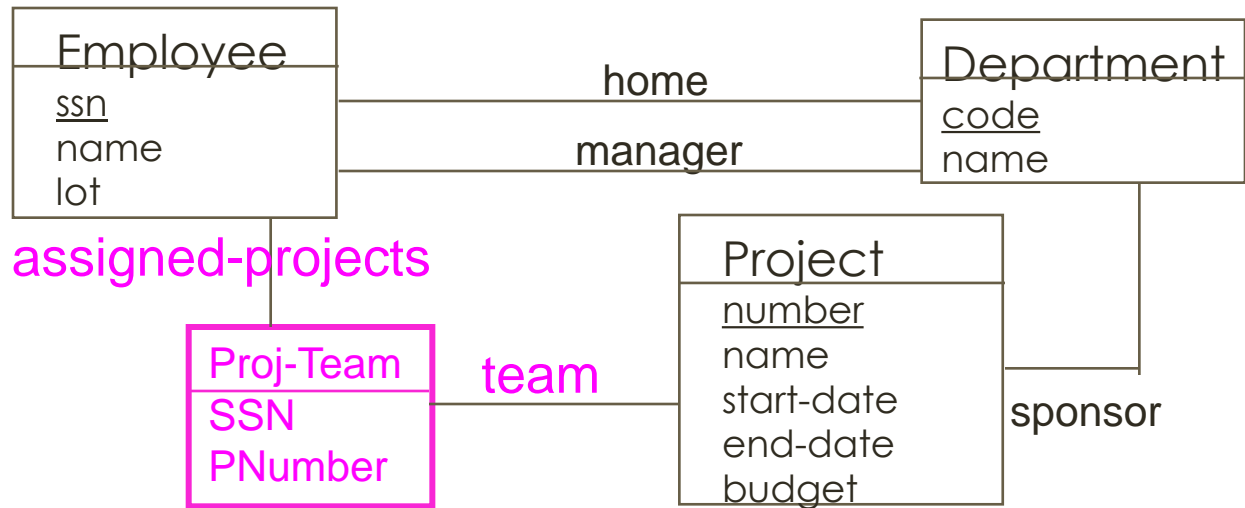
Notice that the many-to-many relationship set must be represented in a (new) table.

Many-to-many relationship sets

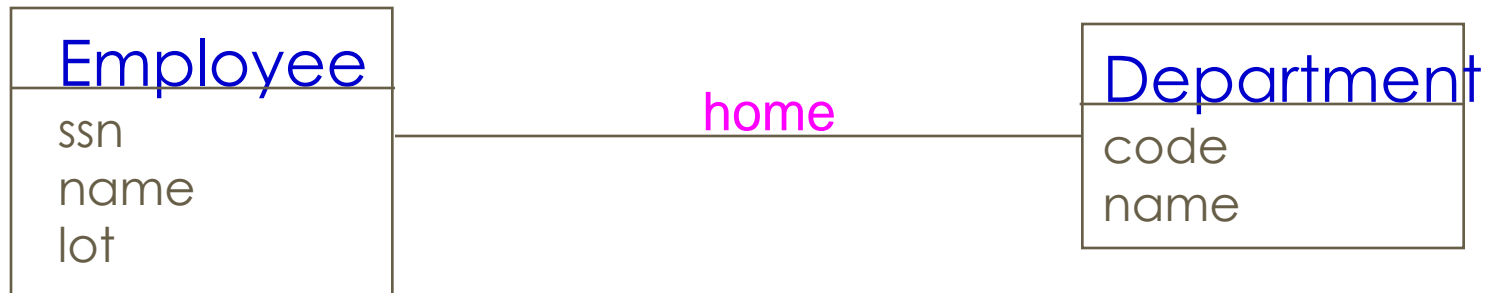
ERD



Relational
DB Diagram



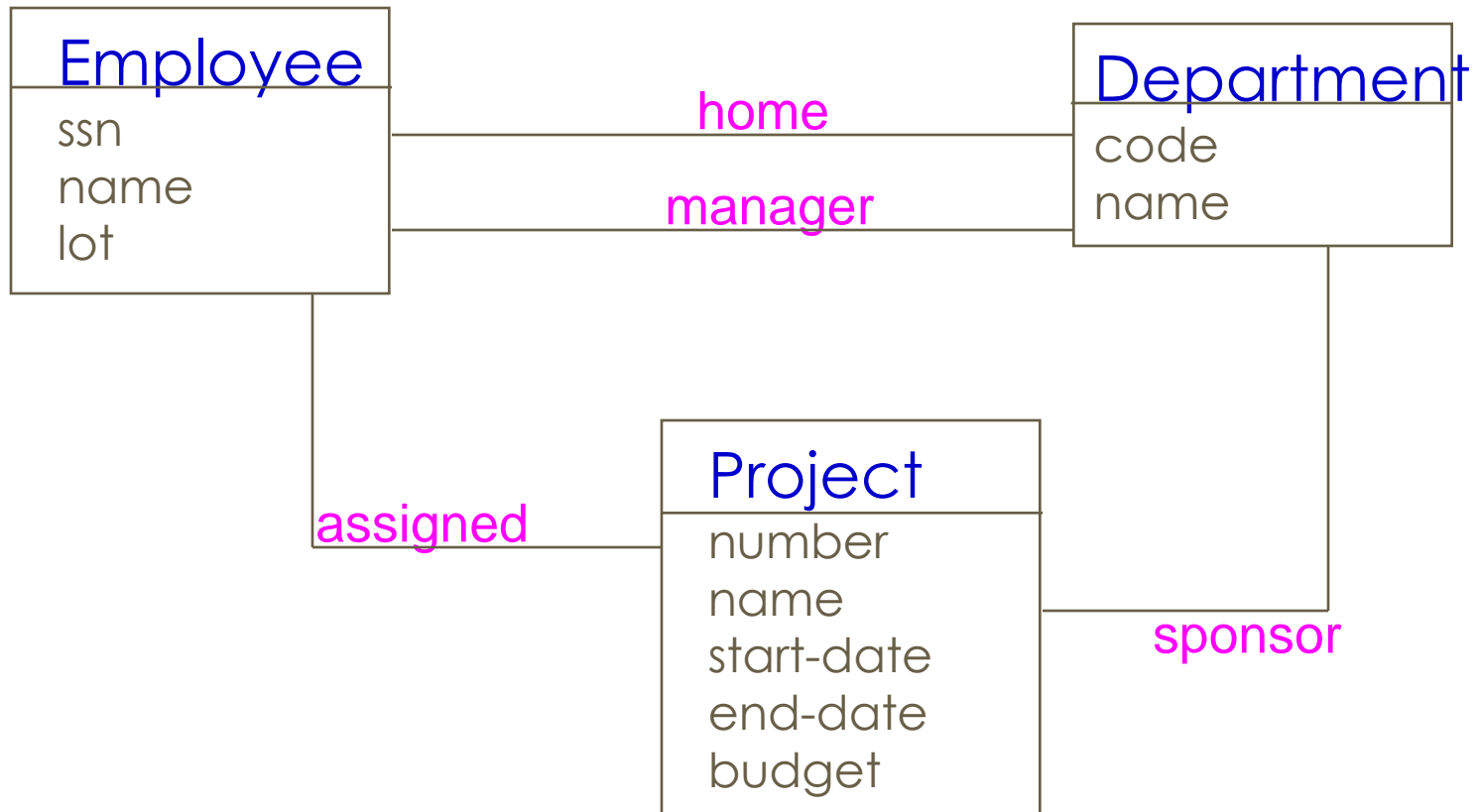
What data do we need to record a relationship?



We must indicate which employee and which department are to be connected (for each relationship).

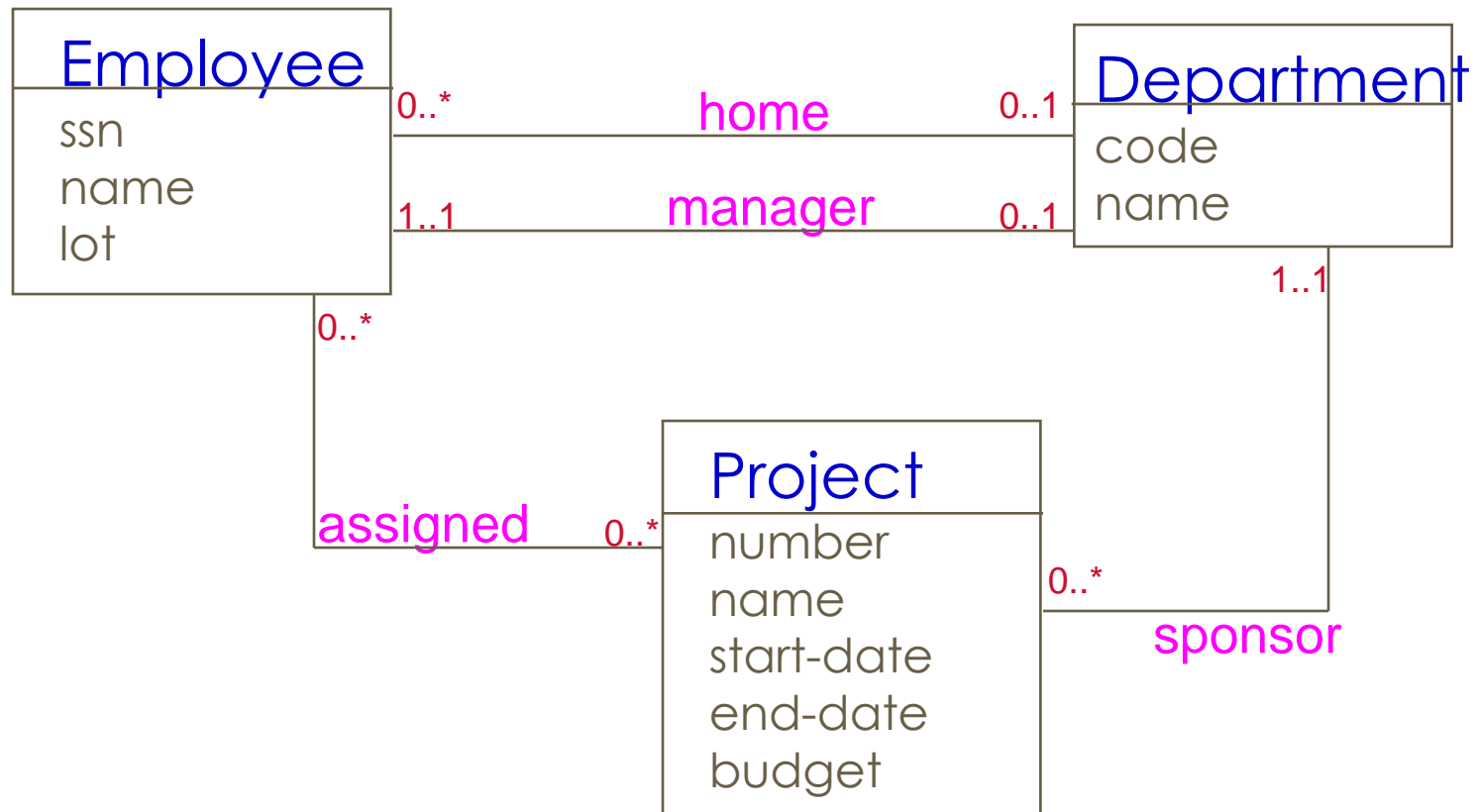
We need the key value for an employee and the key value for the department – stored together – to represent the relationship.

Cardinality Constraints on Relationship sets: How many entities can participate?

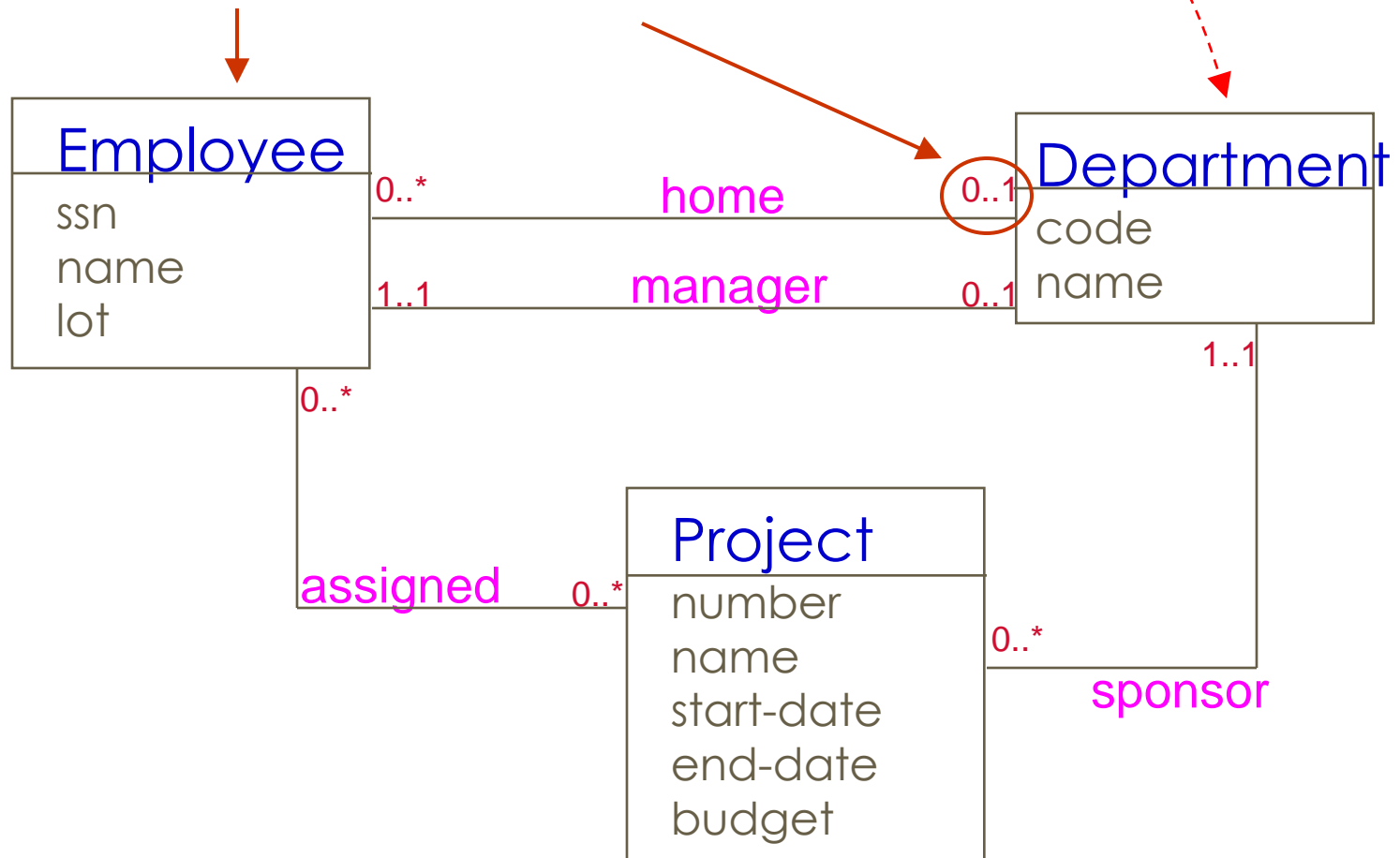


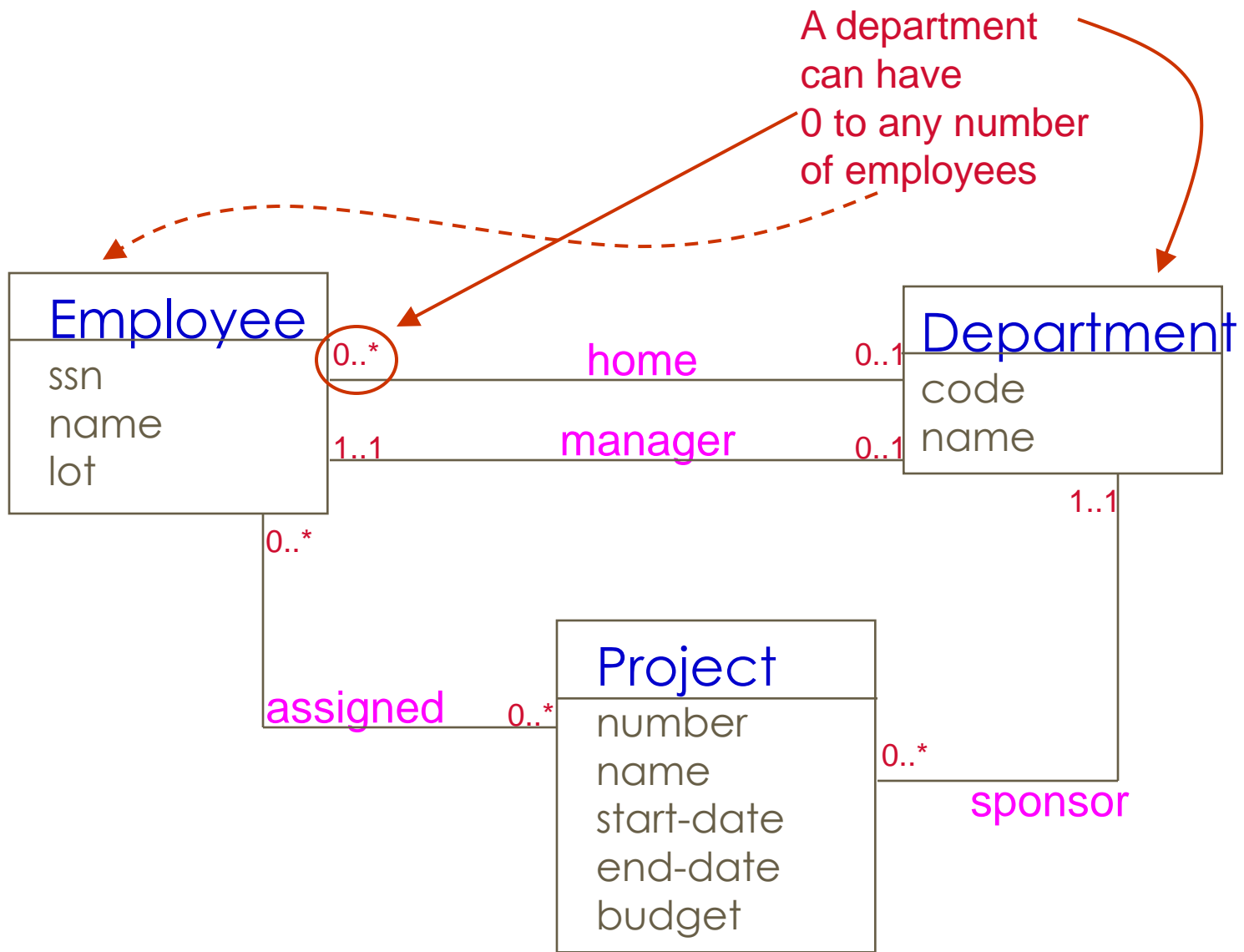
Cardinality Constraints on Relationship sets

How many entities can participate?



An employee can have 0 or 1 home departments



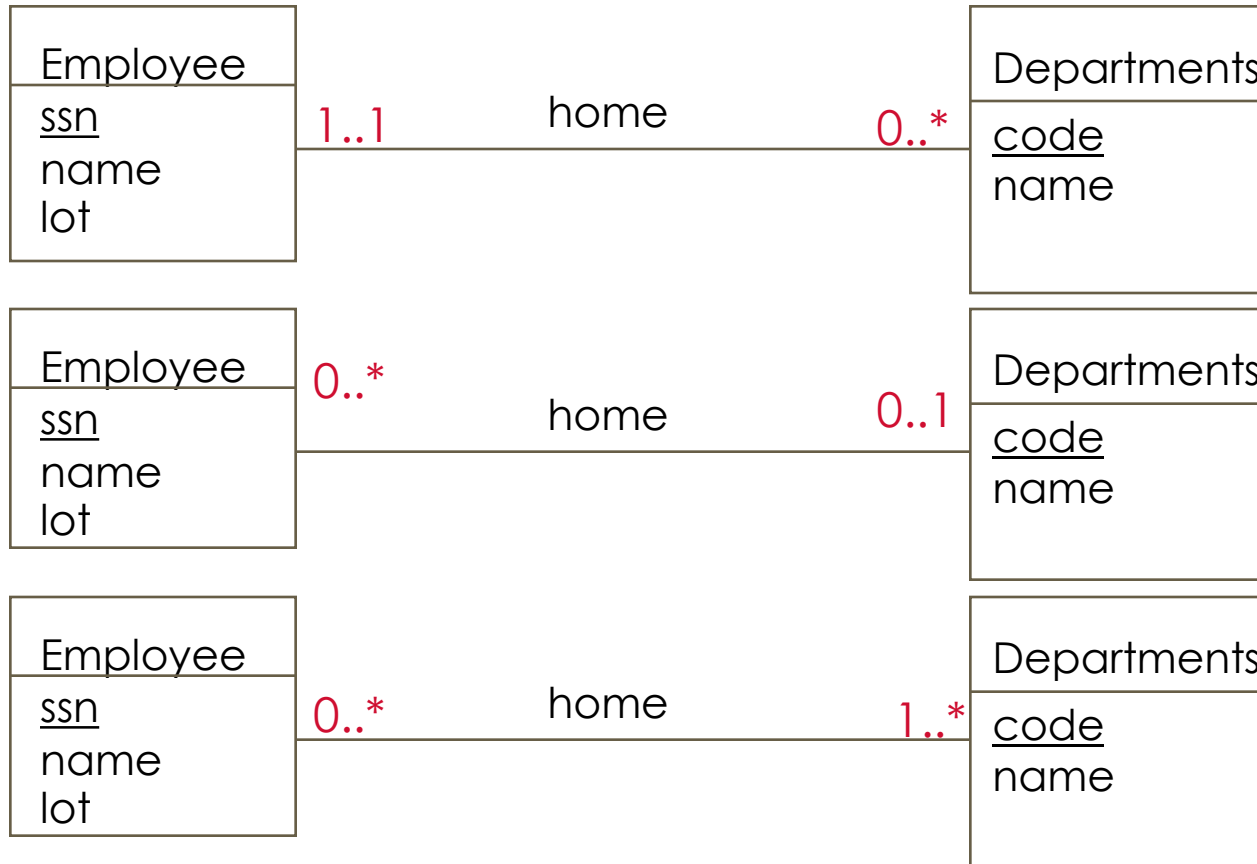


A department can have 0 to any number of employees



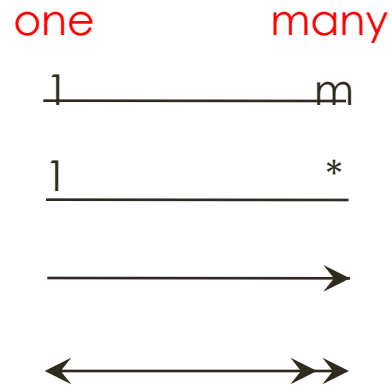
0..*

Unified Modeling Language (UML): Class Diagram



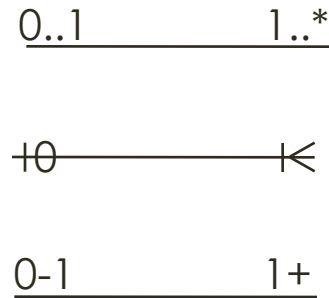
Which one is right? We must discover the semantics of the application!

Various notation for “one-to-many”



maximum cardinalities only

zero..one one..many



minimum and maximum cardinalities

Notice, UML notation allows any sort of cardinality. DBMSs are limited to enforcing a max of 1 or many and a min of 0 or 1.

Various notations for “many-to-many”

many many



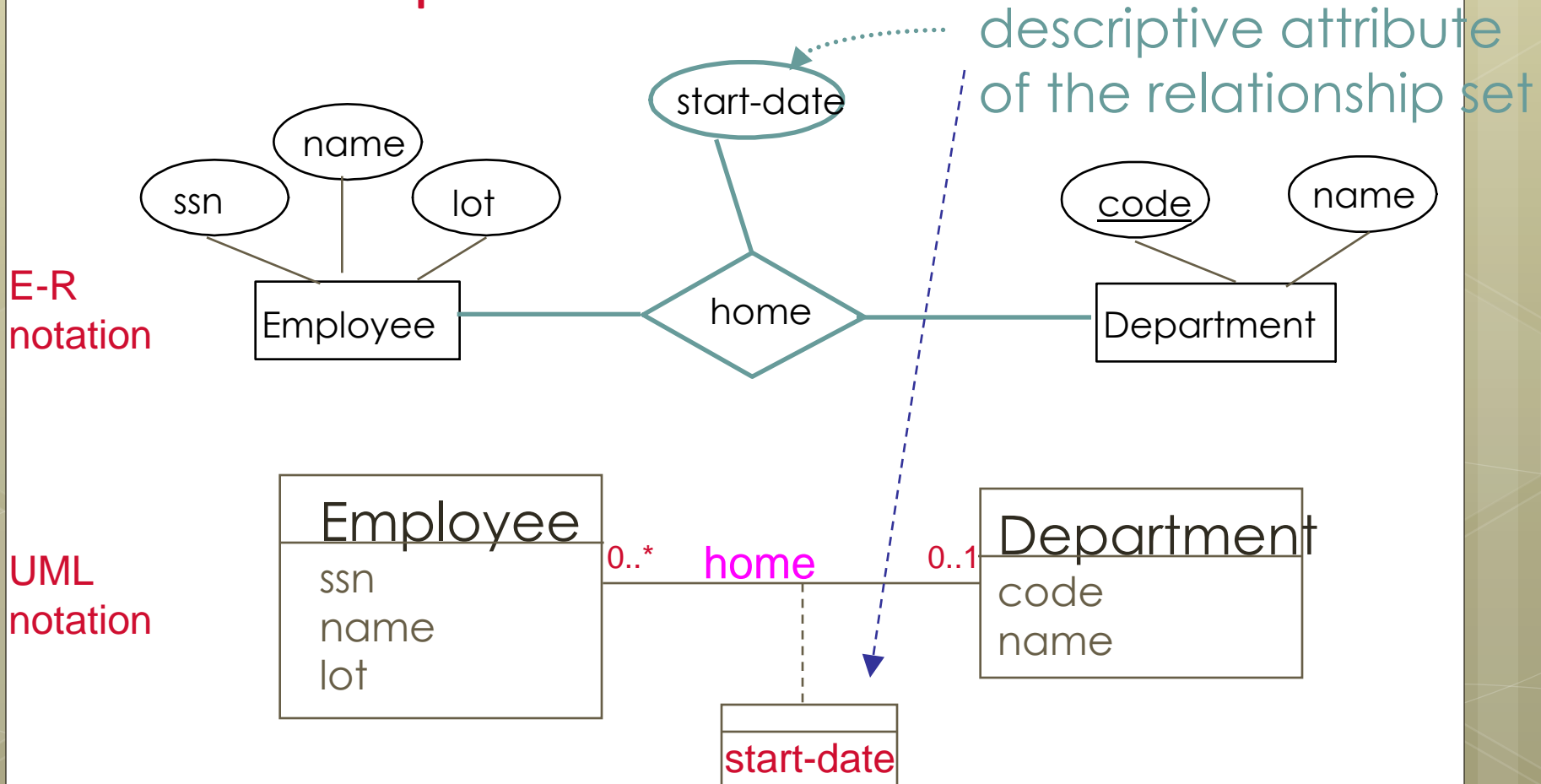
maximum cardinalities only

one..many one..many

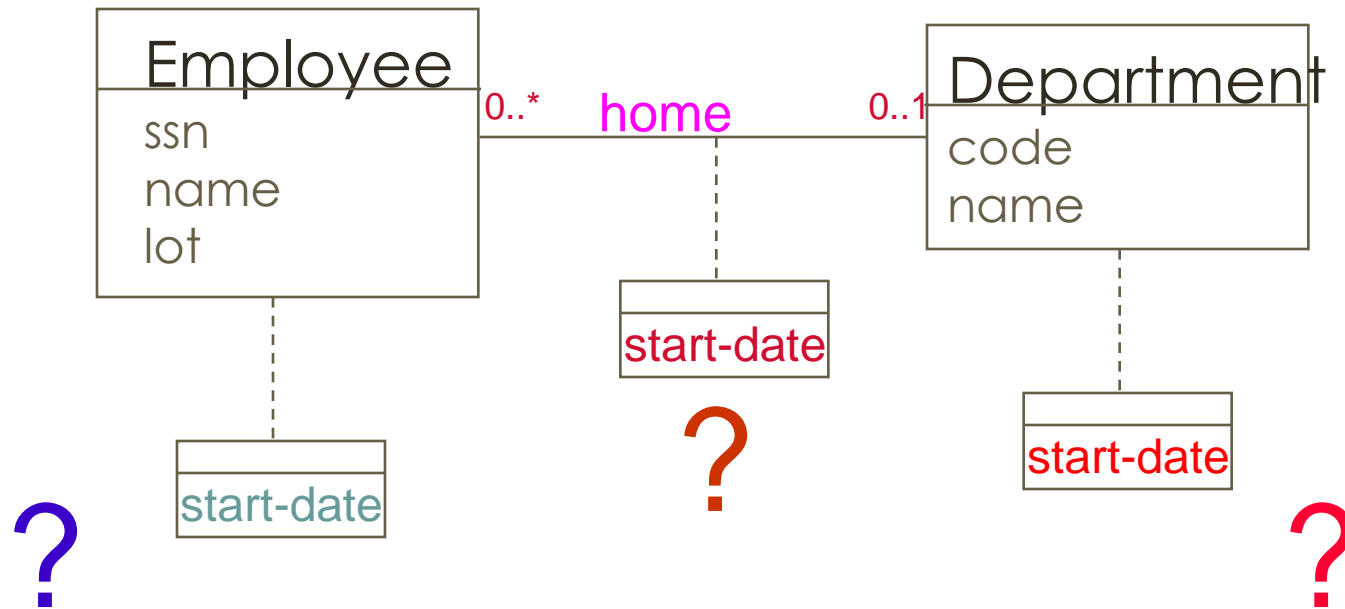


minimum and maximum cardinalities

Relationship sets can have attributes

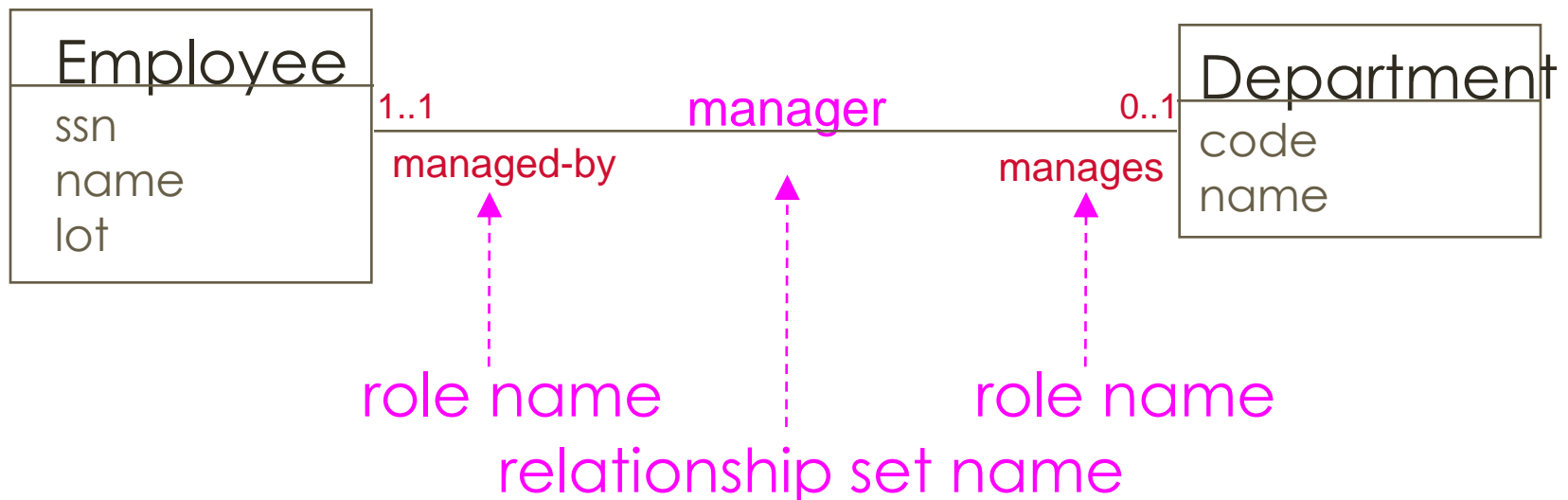


Try all three locations for the attributes:
which one makes sense?

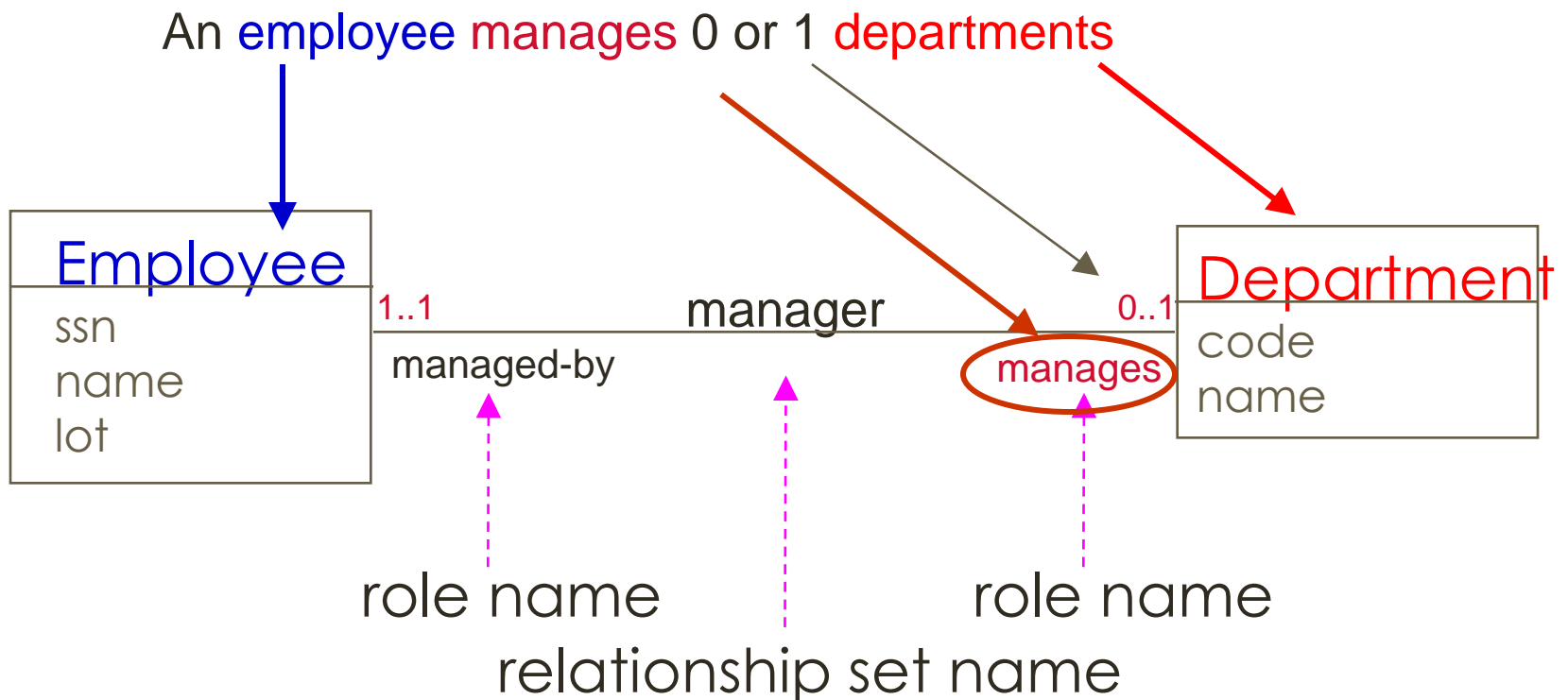


Relationship sets can have **role** names

(in addition to the name of the relationship set)

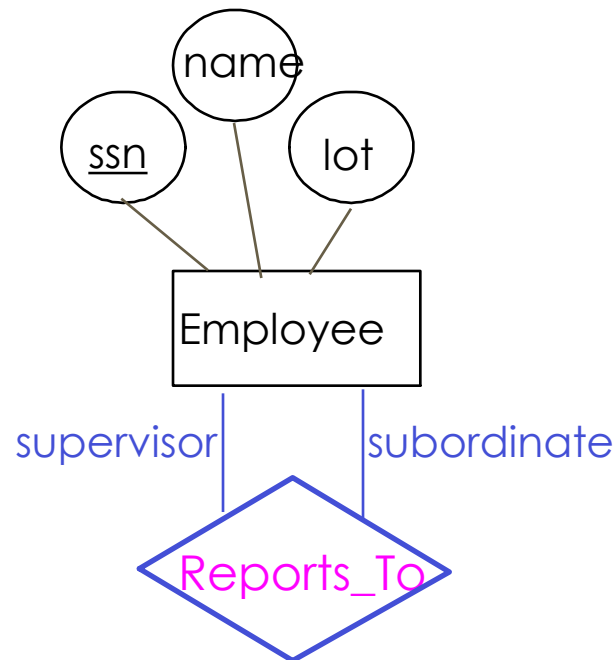


Example: reading **role** names

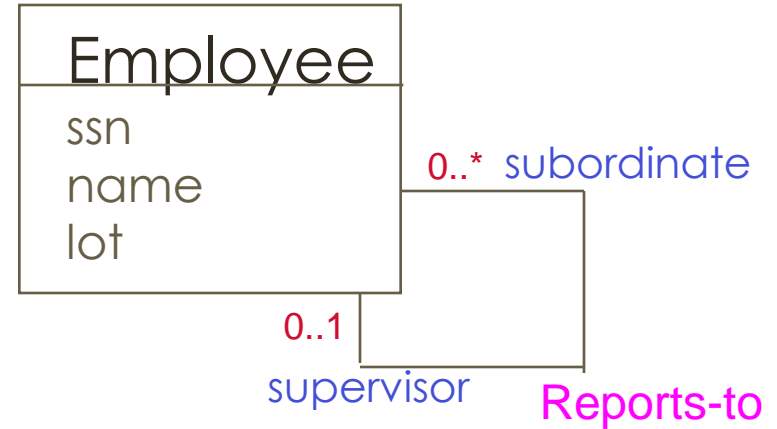


Same entity sets can participate in different “roles” for the same relationship set

E-R notation



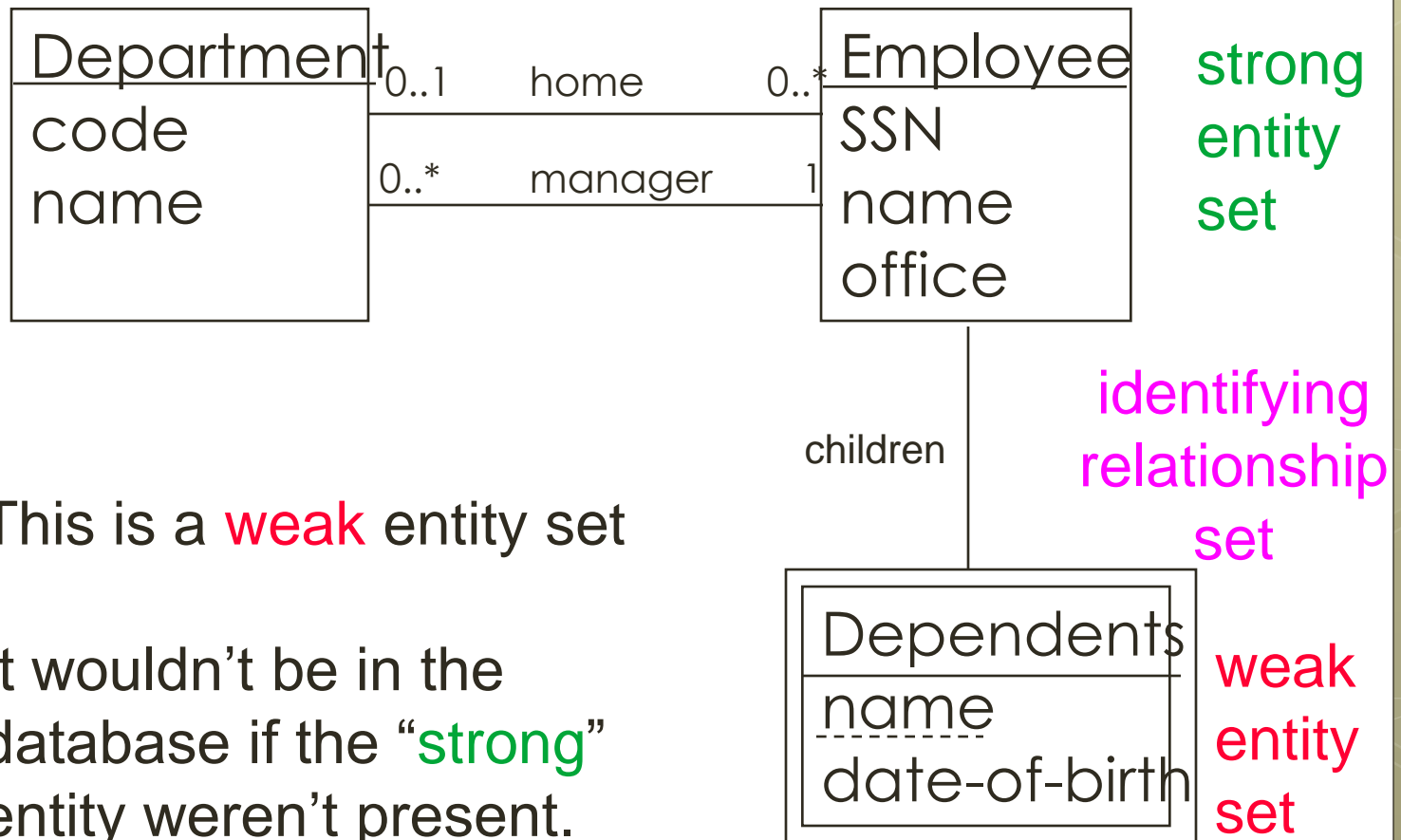
UML notation



Exercise: Draw an ERD

- Professors have a SSN, a name and an age and their SSNs uniquely identify them.
- Professors teach courses. Each course is supervised by one professor.
- Courses are uniquely identified by their courseID, and they have a name.
- Choose cardinalities for all your relationships.

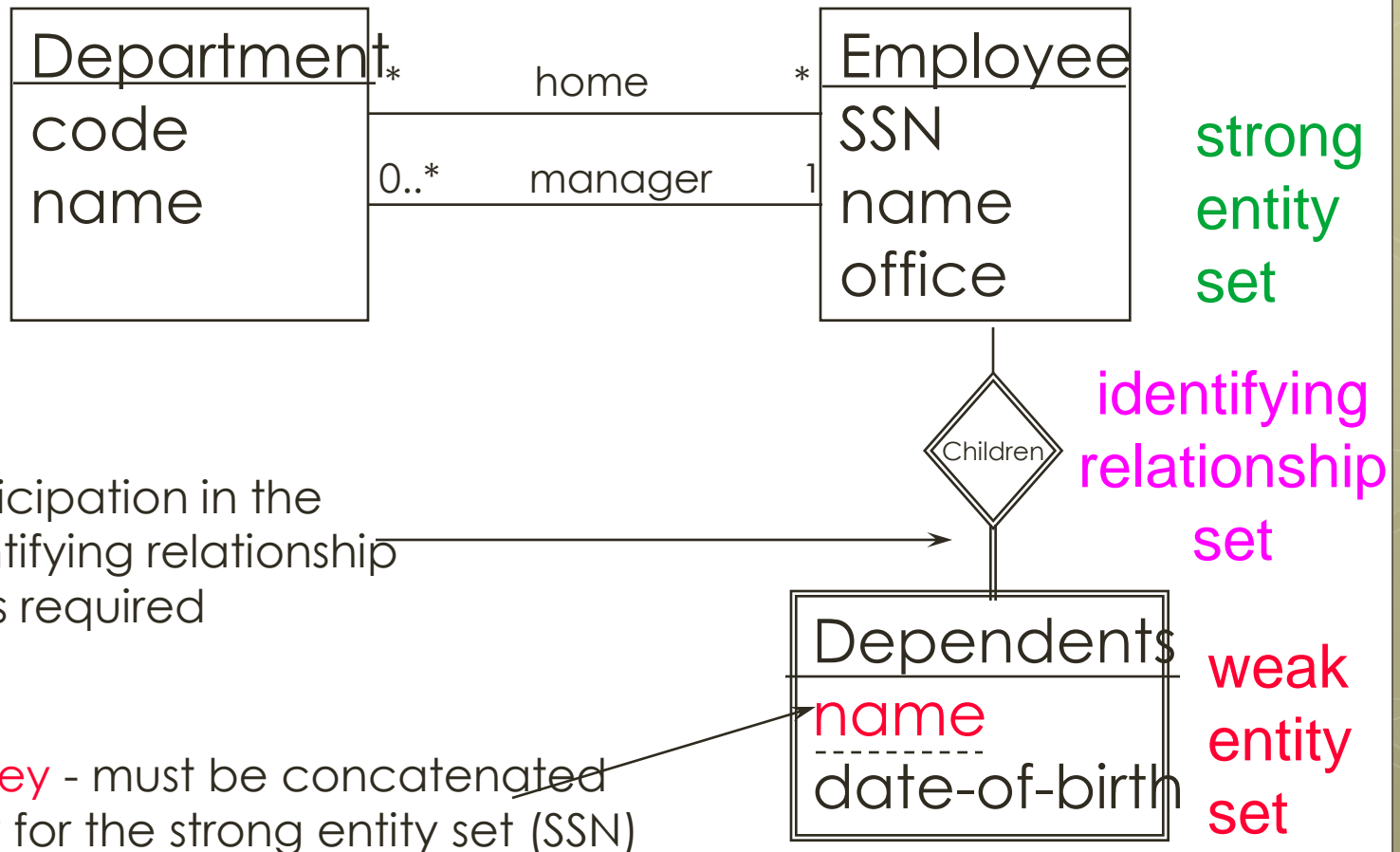
Weak Entity Sets (and Identifying Relationship sets)



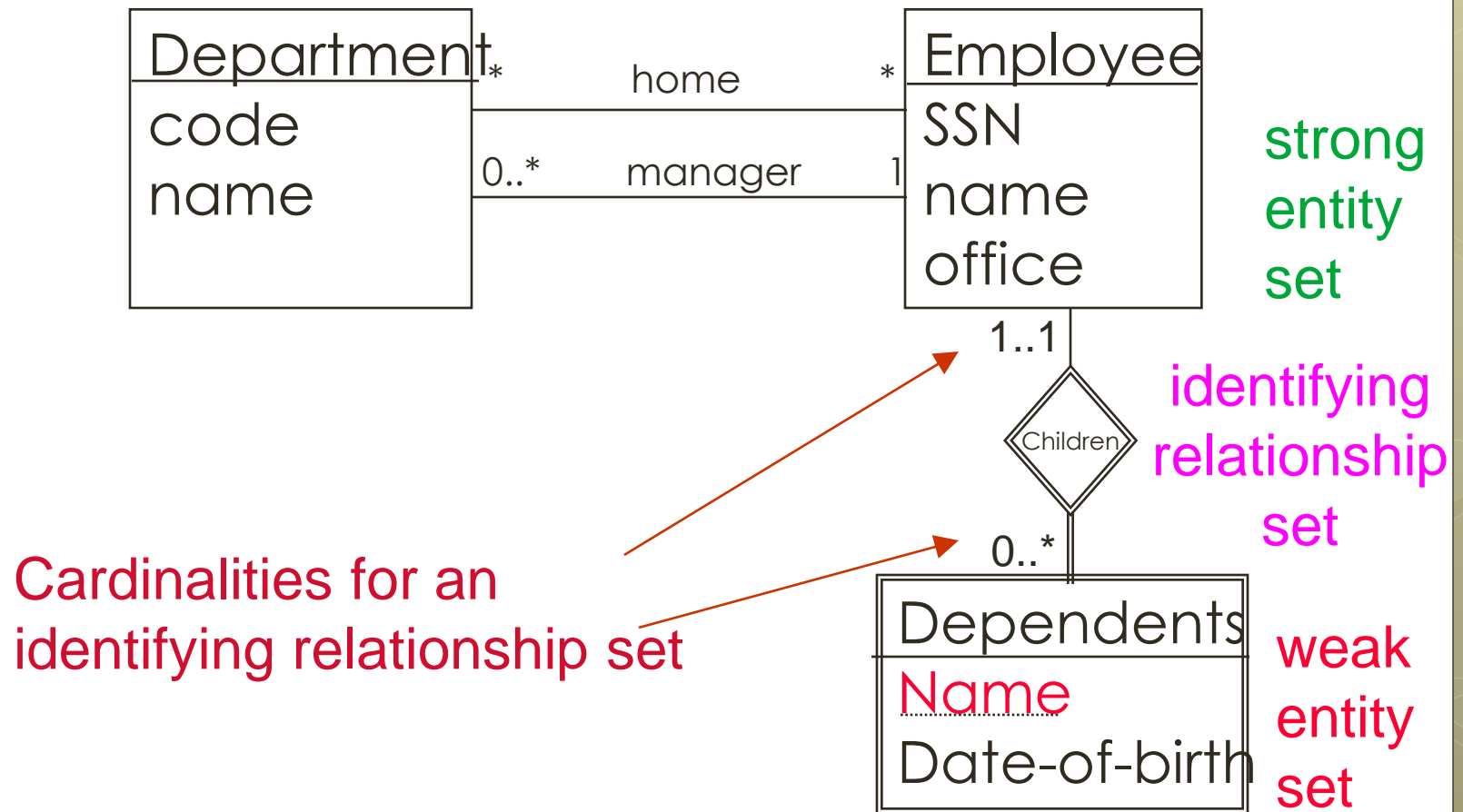
This is a **weak** entity set

It wouldn't be in the database if the “**strong**” entity weren't present.

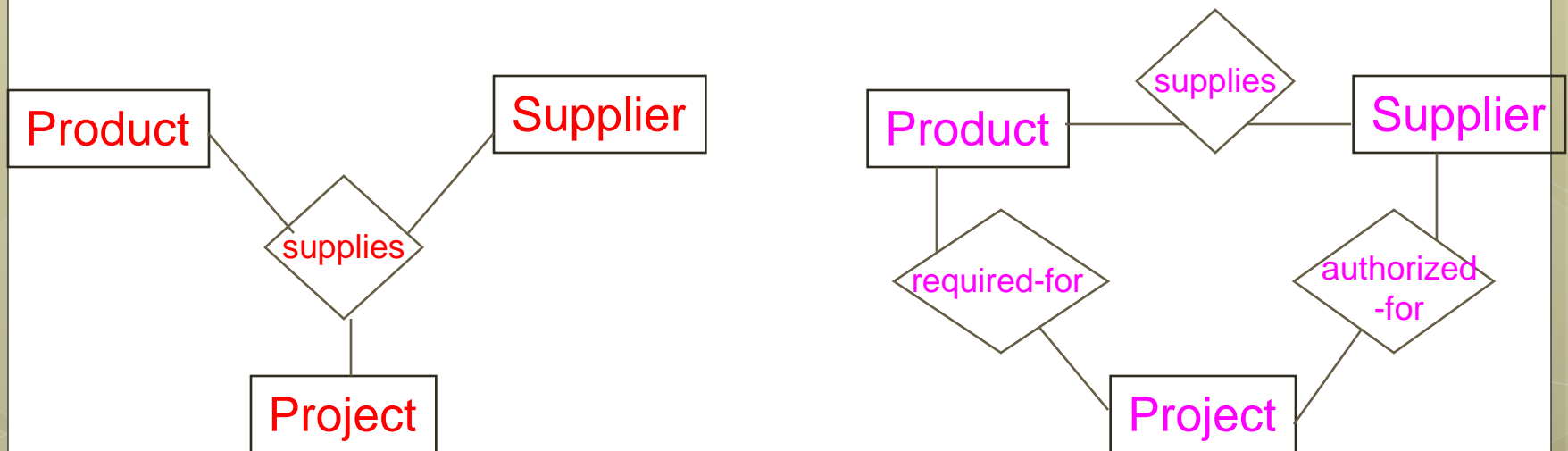
Weak Entity Sets and Identifying Relationship sets: Alternative Notation



Weak Entity Sets and Identifying Relationship sets: Alternative Notation (cont.)



Ternary vs. Binary Relationship sets



These two schemas are not equivalent!

When would we use a ternary relationship set?

When would we use three binary relationship sets?

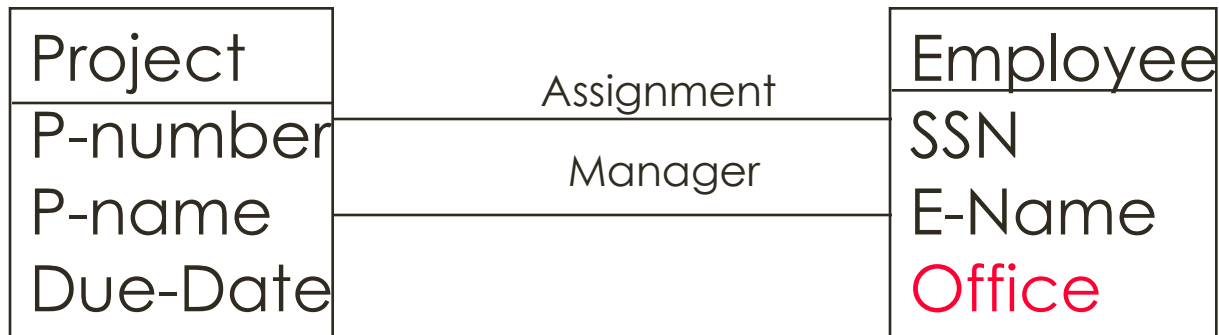
Binary vs. Ternary Relationship sets (Cont.)

- The ternary relationship set means that a Supplier must be authorized to supply a particular part to a particular project. e.g., **Office-Depot can supply laser printer paper to project 112.** **Office-Max can supply paper clips to Project 112.** **Office-Max can supply pencils to project 115.** (But based on that much information, **Office-Max can't supply pencils to 112.**)
- The three binary relationship sets each represent something distinct. A Supplier can be authorized to supply certain products (Office-Max can supply pencils). A Project can require certain products (112 needs pencils). And a Supplier can be authorized to supply a certain project. (Office-Max supplies 112)
Therefore: **Office-Max can supply pencils to 112.**

Duality:
and

entity
attribute

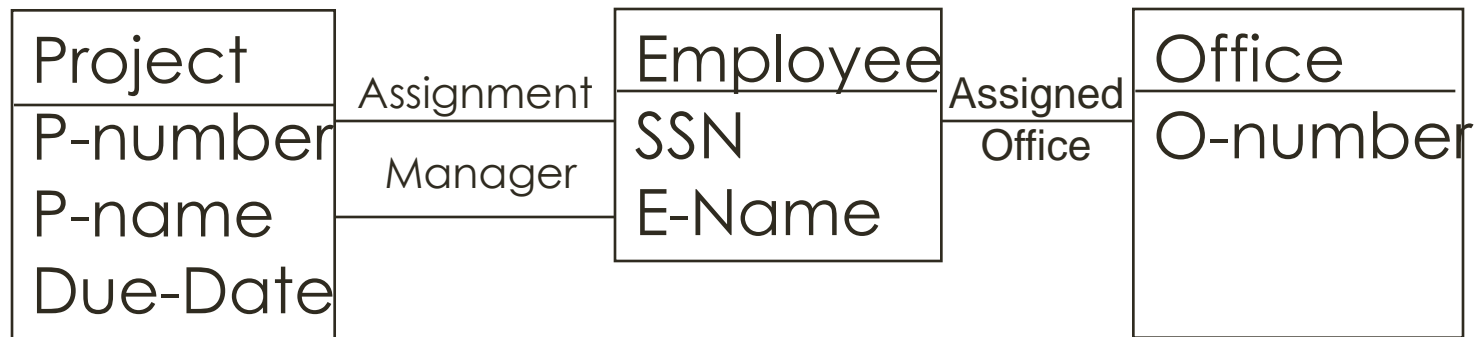
value
relationship



Should **Office** be an **attribute of Employee**? or a **separate entity set**? Most attributes can be “promoted” to an entity set and some entities can be “demoted” to an attribute value.

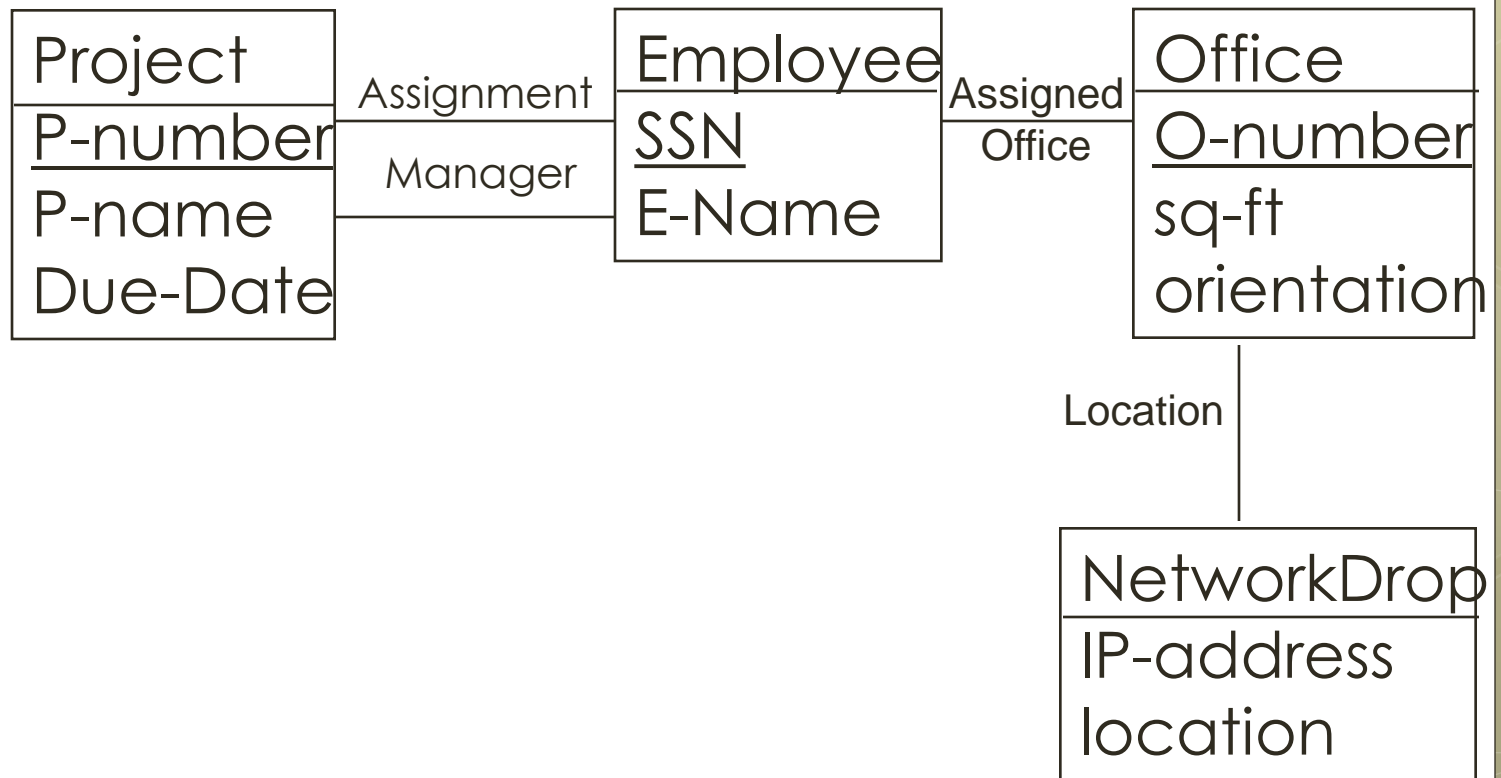
This explains why there are so many different ways to **design a schema.**

What are some reasons to model Office as an entity set?

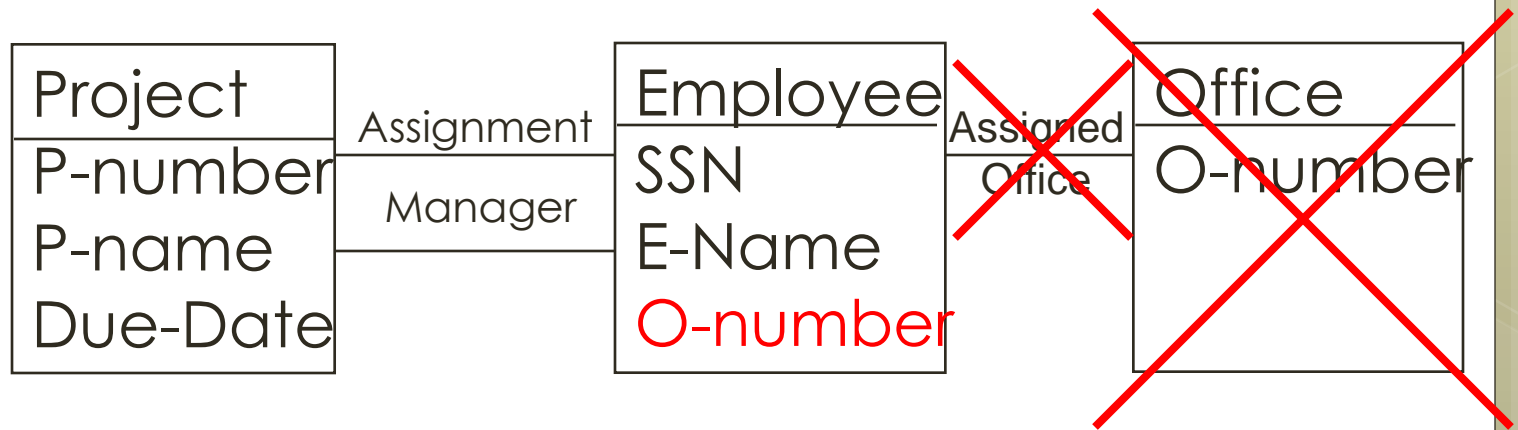


- an employee can have more than one office
- there are other attributes of Office
- Office needs to participate in other relationship sets such as a relationship set connecting to furniture or telephones or network drops (located in the office)

Office – as an entity (with attributes & relationships)



What are some reasons to model Office as an attribute of Employee?

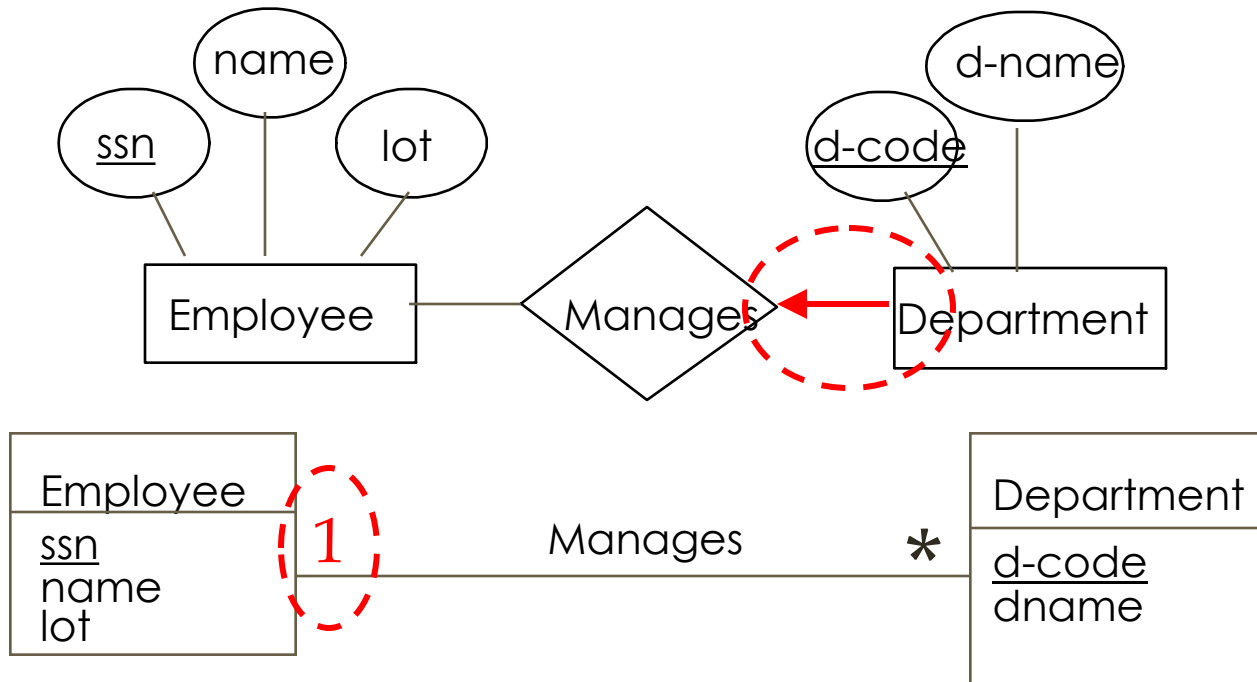


- it is faster to access office number; there is no join required.
- schema is a little simpler; one less table

Key Constraints - as described in the book

(limit participation in relationship set to at most 1)

same as maximum multiplicity of 1 in UML

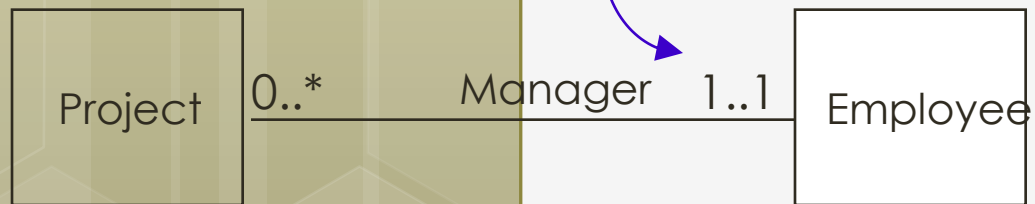


Each dept has at most one manager, according to the *key constraint* on Manages.

Participation Constraint - :

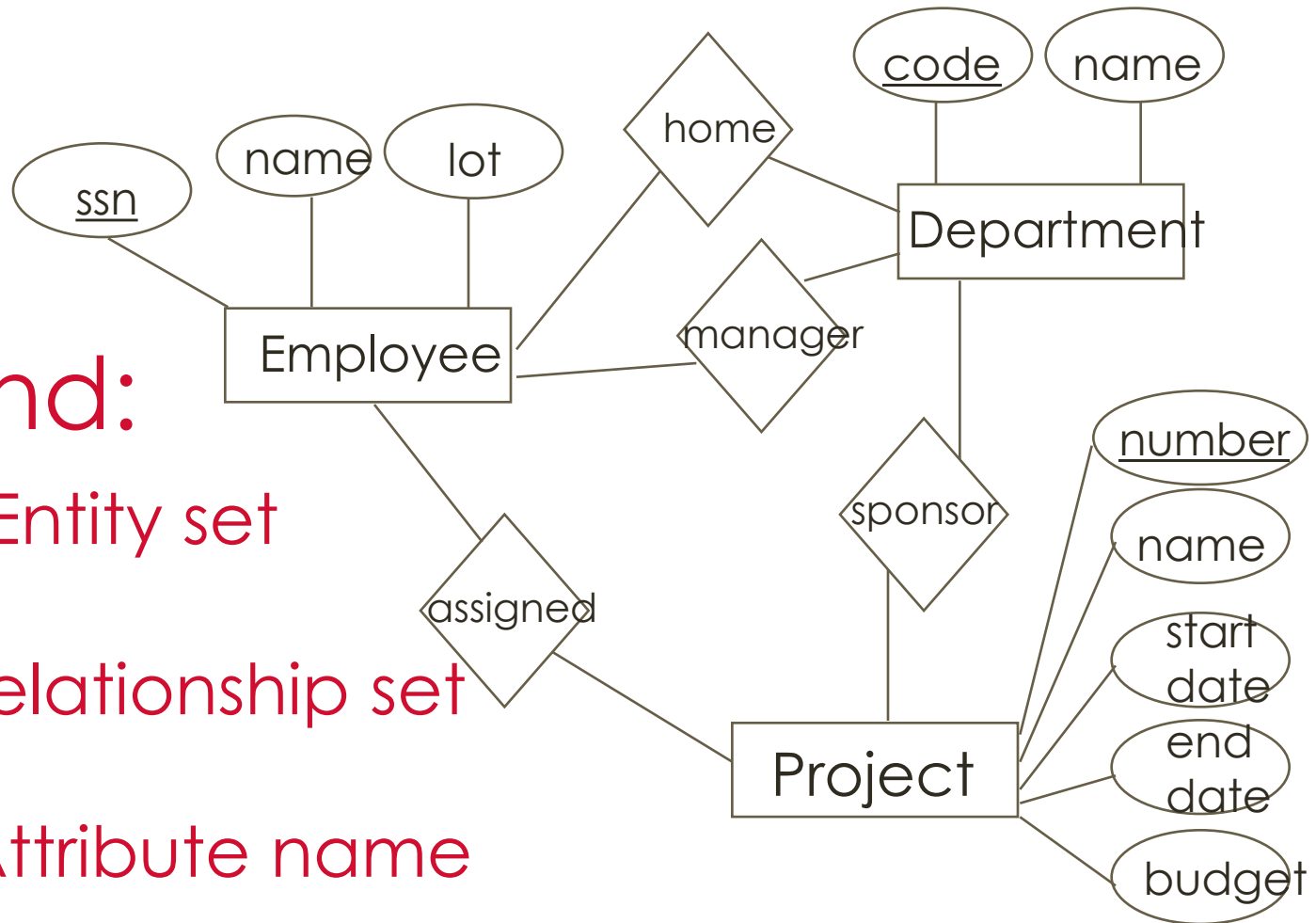
when every entity **MUST** participate in a relationship set

a Project has exactly one manager



**a Project MUST have a manager
and there is at most 1 employee who is manager**

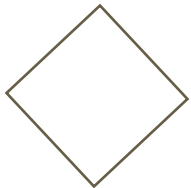
Entity-Relationship Diagram (original syntax)



Legend:



Entity set

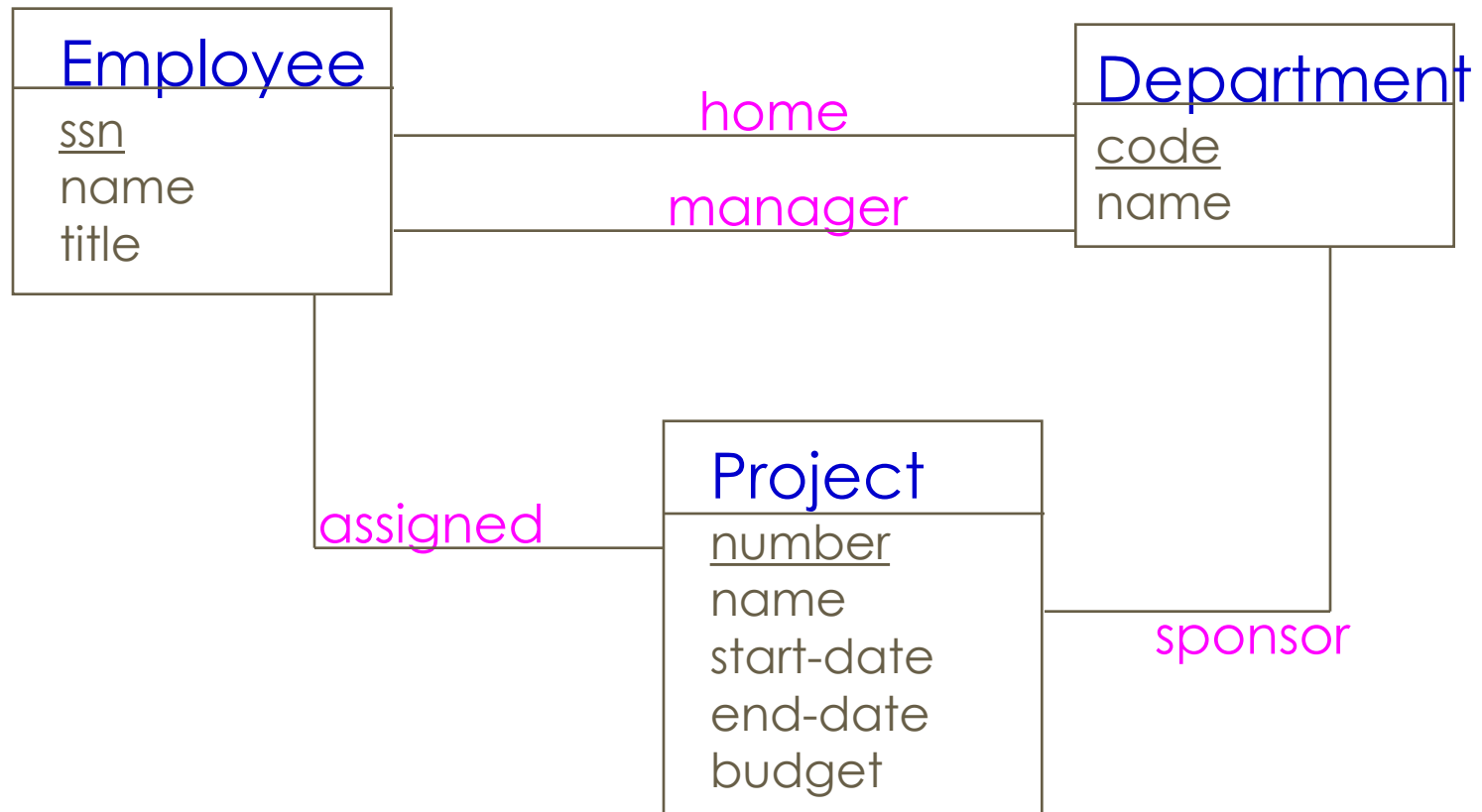


Relationship set



Attribute name

UML version of the same E-R Diagram



Equivalent Relational Schema - with foreign keys shown

Employee (ssn, name, title, home-dept)

Project-team(ssn, number)

Department (id, name, manager)

Project (number, name, start-date, end-date, budget, sponsor)

Notice that the many-to-many relationship set must be represented in a (new) table.

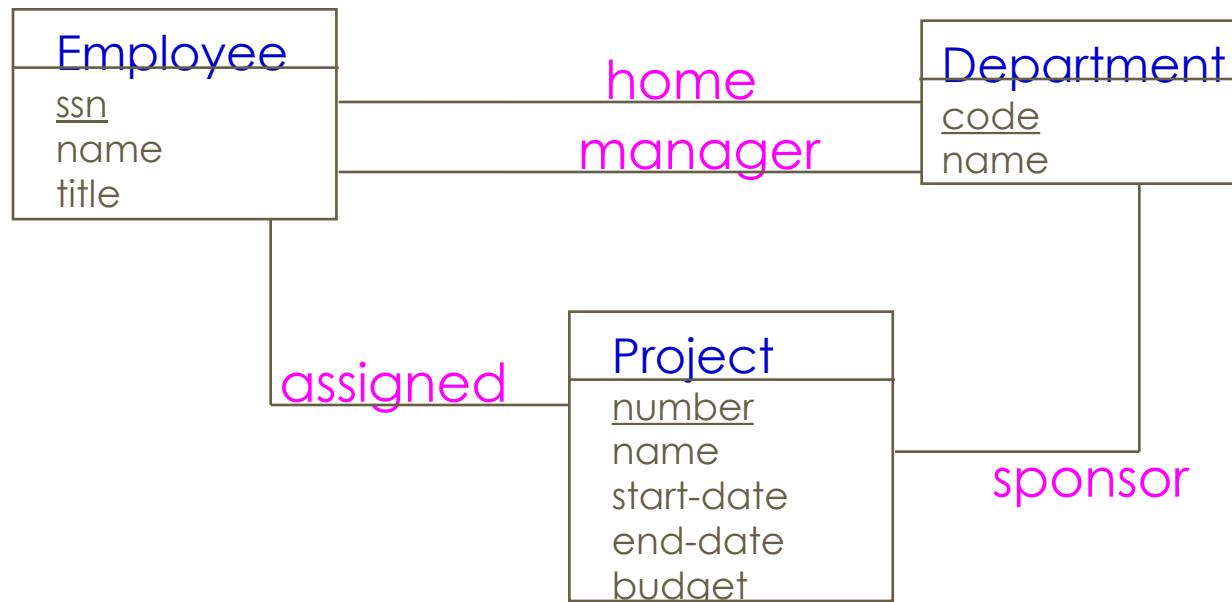
Equivalent Relational Schema

Employee (ssn, name, title, home-dept)

Project-team(ssn, number)

Department (id, name, manager)

Project (number, name, start-date, end-date, budget, sponsor)



Practice 1

- A club has a name, office and phone; it is uniquely identified by its name.
- Clubs sponsor events. Each event has one main sponsor, and may have co-sponsors.
- An event has a title, date, location and description; it is uniquely identified by the title and date.

Draw the Entity-Relationship Diagram (ERD).

Practice 2

- Draw the UML version of the previous ERD.