Database Applications (15-415)

The Entity Relationship Model
Lecture 2, January 12, 2016

Mohammad Hammoud

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Today…

- Last Session:
  - Course overview and a brief introduction on databases and database systems

- Today's Session:
  - Introduction on databases and database systems (*Continue*)
  - Main steps involved in designing databases
  - Constructs of the entity relationship (ER) model
  - Integrity constrains that can be expressed in the ER model

- Announcements:
  - The first Problem Solving Assignment (PS1) is now posted on the course webpage. It is due on Jan 21st by midnight
  - Thursday, Jan 14th is the first recitation
    - A case study on the ER model will be solved together

Outline

- A Primer on Databases ✓
- Database Design
- ER Model: Constructs and Constraints

A Motivating Scenario

- Qatar Foundation (QF) has a large collection of data (say 500GB) on employees, students, universities, research centers, etc.,
- **Performance (Concurrency Control)**
- **Performance (Response Time)**
- **Correctness (Consistency)**
- **Correctness (Security)**
- **Correctness (Durability and Atomicity)**

Managing Data using File Systems

- What about managing QF data using local file systems?
  - Files of fixed-length and variable-length records as well as formats
  - Main memory vs. disk
  - Computer systems with 32-bit addressing vs. 64-bit addressing schemes
  - Special programs (e.g., C++ and Java programs) for answering user questions
  - Special measures to maintain atomicity
  - Special measures to maintain consistency of data
  - Special measures to maintain data isolation
  - Special measures to offer software and hardware fault-tolerance
  - Special measures to enforce security policies in which different users are granted different permissions to access diverse subsets of data

This becomes tedious and inconvenient, especially at large-scale, with evolving/new user queries and higher probability of failures!

Data Base Management Systems

- A special software is accordingly needed to make the preceding tasks easier

- This software is known as Data Base Management System (DBMS)

- DBMSs provide automatic:
  - Data independence
  - Efficient data access
  - Data integrity and security
  - Data administration
  - Concurrent accesses and crash recovery
  - Reduced application development and tuning time

Some Definitions

- A database is a collection of data which describes one or many real-world enterprises
  - E.g., a university database might contain information about entities like students and courses, and relationships like a student enrollment in a course

- A DBMS is a software package designed to store and manage databases
  - E.g., DB2, Oracle, MS SQL Server, MySQL and Postgres

- A database system = (Big) Data + DBMS + Application Programs

Data Models

- The user of a DBMS is ultimately concerned with some real-world enterprises (e.g., a University)

- The data to be stored and managed by a DBMS *describes* various aspects of the enterprises
  - E.g., The data in a university database describes students, faculty and courses entities and the relationships among them

- A data model is a collection of high-level data description constructs that hide many low-level storage details

- A widely used data model called the entity-relationship (ER) model allows users to pictorially denote entities and the relationships among them

The Relational Model

- The relational model of data is one of the most widely used models today
- The central data description construct in the relational model is the relation
- A relation is basically a table (or a set) with rows (or records or tuples) and columns (or fields or attributes)
- Every relation has a schema, which describes the columns of a relation
- Conditions that records in a relation must satisfy can be specified
  - These are referred to as integrity constraints

The Relational Model: An Example

- Let us consider the student entity in a university database

Students Schema

Students(*<u>sid</u>*: string, *name*: string, *login*: string, *dob*: string, *gpa*: real)

An attribute, field or column

Integrity Constraint: Every student has a unique *sid* value

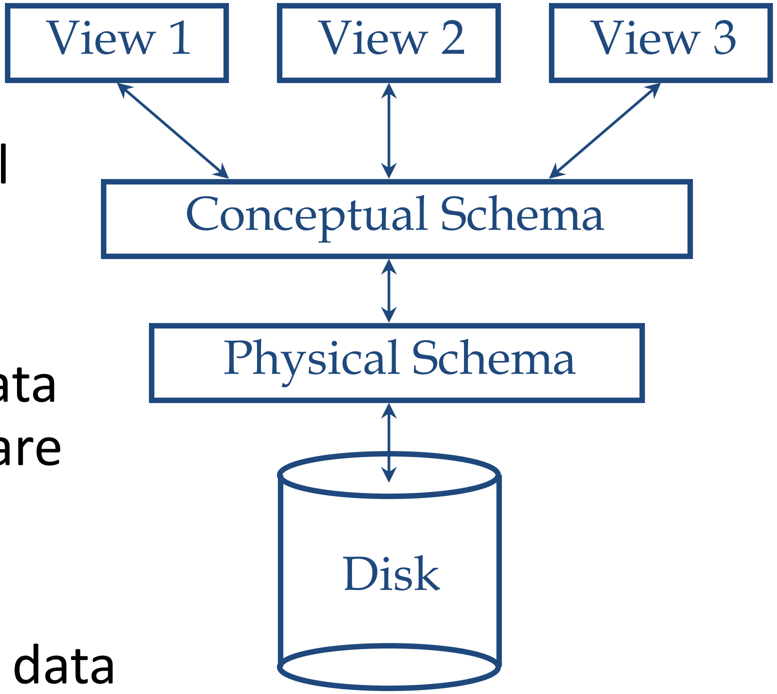
A record, tuple or row

sid	***name***	***login***	***dob***	***gpa***
512412	Khaled	khaled@qatar.cmu.edu	18-9-1995	3.5
512311	Jones	jones@qatar.cmu.edu	1-12-1994	3.2
512111	Maria	maria@qatar.cmu.edu	3-8-1995	3.85

An *instance* of a Students *relation*

Levels of Abstraction

- The data in a DBMS is described at three levels of abstraction, the conceptual (or logical), physical and external schemas



- The conceptual schema describes data in terms of a specific data model (e.g., the relational model of data)

- The physical schema specifies how data described in the conceptual schema are stored on secondary storage devices

- The external schema (or views) allow data access to be customized at the level of individual users or group of users (views can be 1 or many)

Views

- A view is conceptually a relation
- Records in a view are computed as needed and usually not stored in a DBMS
- Example: University Database

Conceptual Schema	Physical Schema	External Schema (View)
• Students(sid: string, name: string, login: string, dob: string, gpa:real) • Courses(cid: string, cname:string, credits:integer) • Enrolled(sid:string, cid:string, grade:string)	• Relations stored as heap files • Index on first column of Students	Students can be allowed to find out course enrollments: • Course_info(cid: string, enrollment: integer)

Can be computed from the relations in the conceptual schema (so as to avoid data redundancy and inconsistency).

Iterating: Data Independence

- One of the most important benefits of using a DBMS is data independence

- With data independence, application programs are insulated from how data are structured and stored

- Data independence entails two properties:
  - Logical data independence: users are shielded from changes in the conceptual schema (e.g., add/drop a column in a table)
  - Physical data independence: users are shielded from changes in the physical schema (e.g., add index or change record order)

Queries in a DBMS

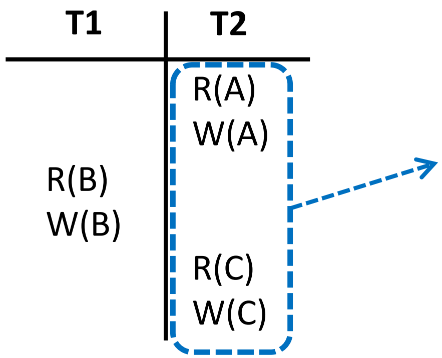
- The ease with which information can be queried from a database determines its value to users

- A DBMS provides a specialized language, called the query language, in which queries can be posed

- The relational model supports powerful query languages
  - Relational calculus: a formal language based on mathematical logic
  - Relational algebra: a formal language based on a collection of operators (e.g., selection and projection) for manipulating relations
  - Structured Query Language (SQL):
    - Builds upon relational calculus and algebra
    - Allows creating, manipulating and querying relational databases
    - Can be embedded within a host language (e.g., Java)

Concurrent Execution and Transactions

- An important task of a DBMS is to *schedule* concurrent accesses to data so as to improve performance



An *atomic* sequence of database actions (read/writes) is referred to as "*transaction*"

- When several users access a database *concurrently*, the DBMS must order their requests carefully to avoid conflicts
  - E.g., A check might be cleared while account balance is being computed!

- DBMS ensures that conflicts do not arise via using a locking protocol
  - Shared vs. Exclusive locks

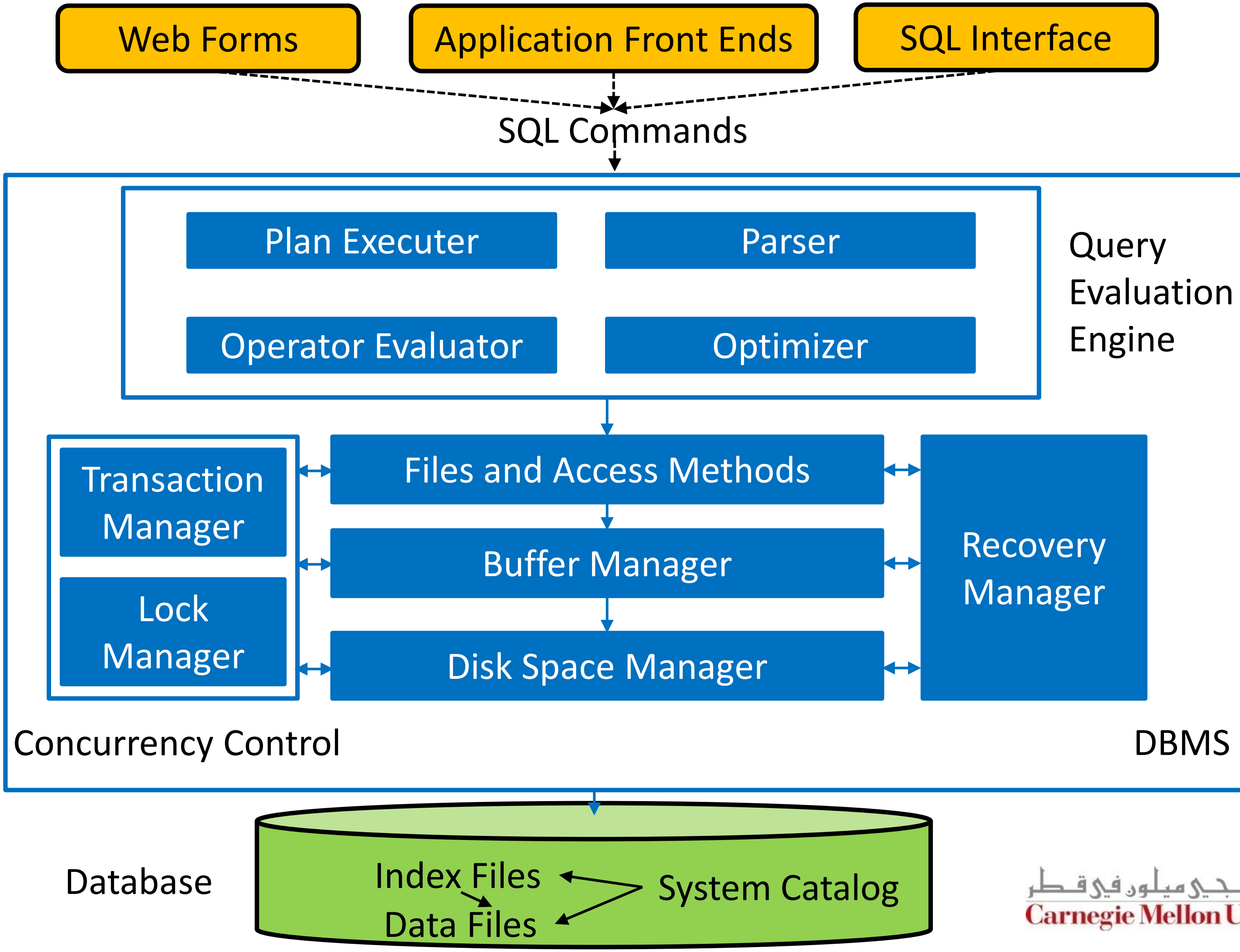
Ensuring Atomicity

- Transactions can be interrupted before running to completion for a variety of reasons (e.g., due to a system crash)

- DBMS ensures atomicity (all-or-nothing property) even if a crash occurs in the middle of a transaction

- This is achieved via maintaining a log (i.e., history) of all writes to the database
  - *Before* a change is made to the database, the corresponding log entry is forced to a safe location (this protocol is called Write-Ahead Log or WAL)
  - After a crash, the effects of partially executed transactions are *undone* using the log

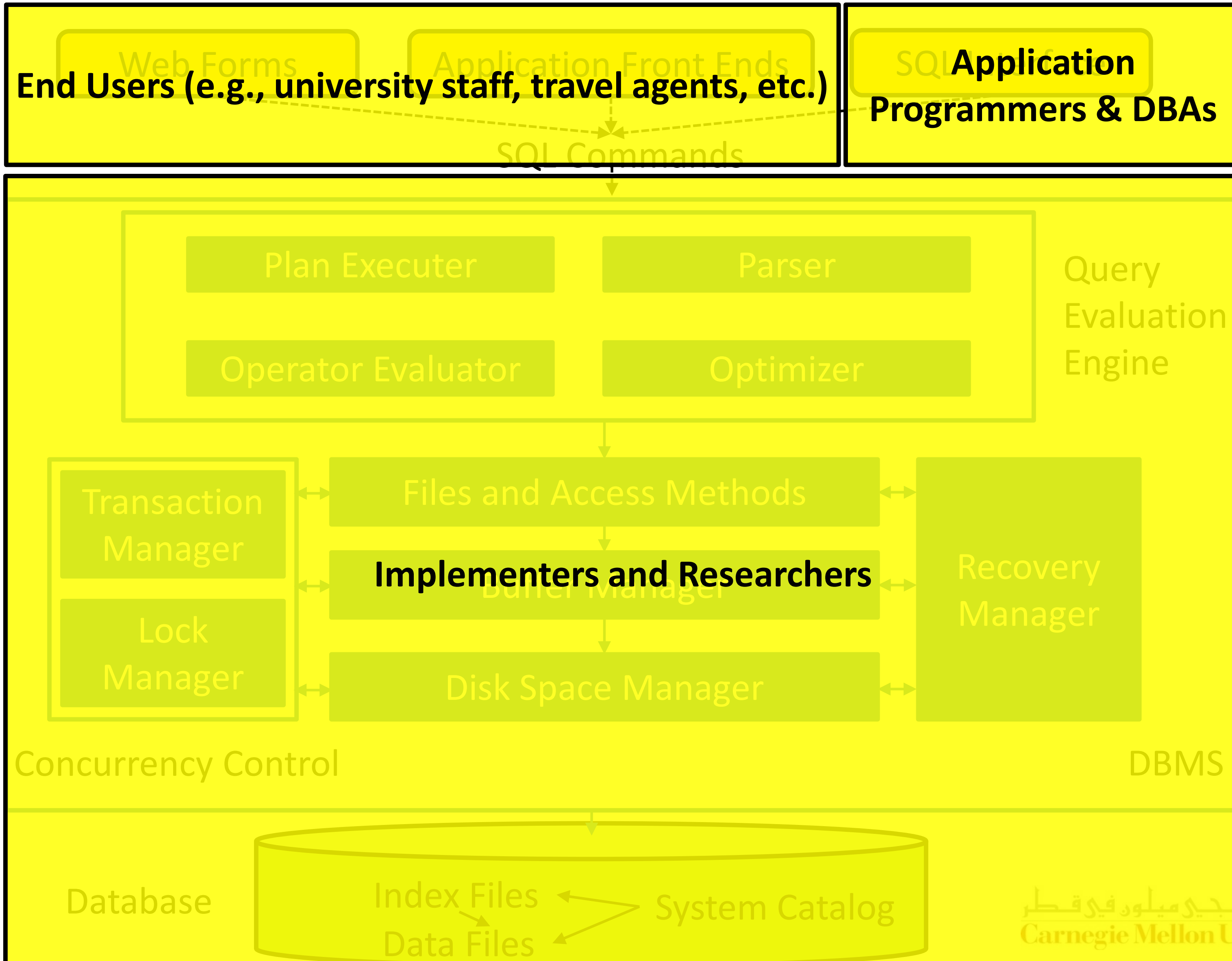
The Architecture of a Relational DBMS



People Who Work With Databases

- There are five classes of people associated with databases:
  1. End users
     - Store and use data in DBMSs
     - Usually not computer professionals
  2. Application programmers
     - Develop applications that facilitate the usage of DBMSs for end-users
     - Computer professionals who know how to leverage host languages, query languages and DBMSs altogether
  3. Database Administrators (DBAs)
     - Design the conceptual and physical schemas
     - Ensure security and authorization
     - Ensure data availability and recovery from failures
     - Perform database tuning
  4. Implementers
     - Build DBMS software for vendors like IBM and Oracle
     - Computer professionals who know how to build DBMS internals
  5. Researchers
     - Innovate new ideas which address evolving and new challenges/problems

The Architecture of a Relational DBMS



Summary

- We live in a world of data
- The explosion of data is occurring along the 3Vs dimensions
- DBMSs are needed for ensuring logical and physical data independence and ACID properties, among others
- The data in a DBMS is described at three levels of abstraction
- A DBMS typically has a layered architecture

Summary

- Studying DBMSs is one of the broadest and most exciting areas in computer science!

- This course provides an in-depth treatment of DBMSs with an emphasis on how to *design*, *create*, *refine*, *use* and *build* DBMSs and real-world enterprise databases

- Various classes of people who work with databases hold responsible jobs and are well-paid!

Outline

- A Primer on Databases
- Database Design
- ER Model: Constructs and Constraints

Database Design

- Requirements Analysis
  - Users needs
- Conceptual Design
  - A high-level description of the data (e.g., using the ER model)
- Logical Design
  - The conversion of an ER design into a relational database schema
- Schema Refinement
  - Normalization (i.e., restructuring tables to ensure some desirable properties)
- Physical Design
  - Building indexes and clustering some tables
- Security Design
  - Access controls

Outline

- A Primer on Databases
- Database Design
- ER Model: Constructs and Constraints ✓

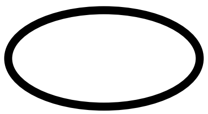
Entities and Entity Sets

- Entity:
  - A real-world object distinguishable from other objects in an enterprise (e.g., University, Students and Faculty)
  - Described using a set of *attributes*

- Entity set:
  - A collection of similar entities (e.g., *all* employees)
  - All entities in an entity set have the same set of attributes (until we consider ISA hierarchies, anyway!)
  - Each entity set has a *key*
  - Each attribute has a *domain*

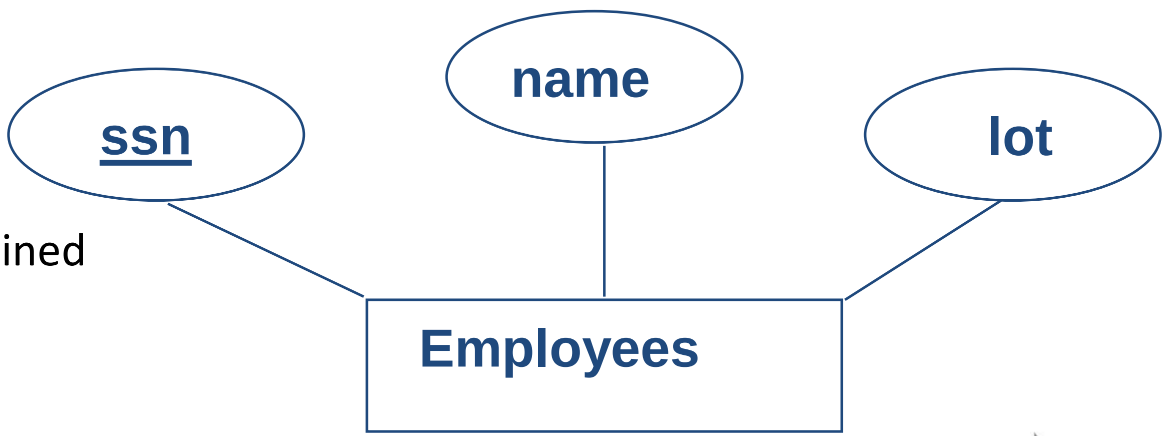
Tools and An ER Diagram

Entities ('Entity Sets')



Attributes

"ssn" is the *primary key*, hence, underlined

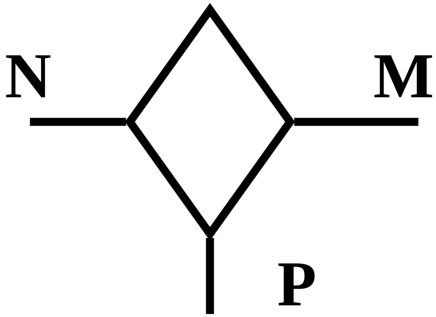


Relationship and Relationship Sets

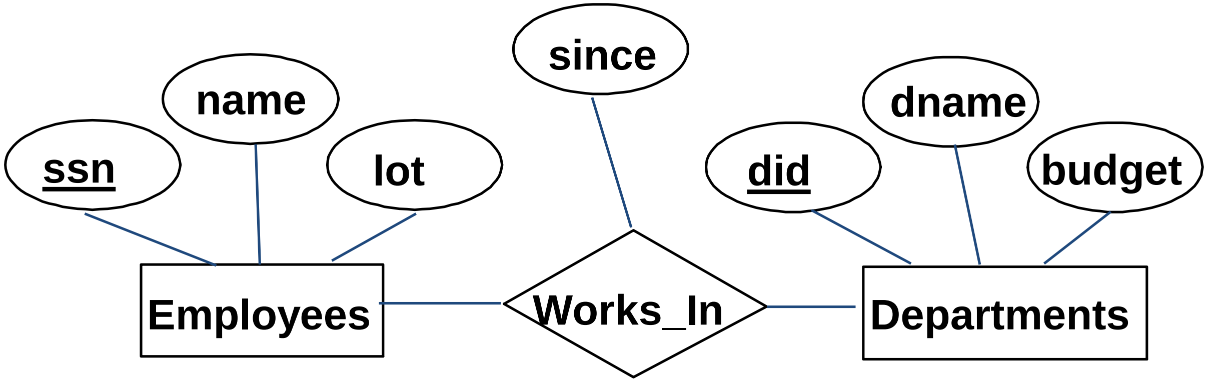
- Relationship:
  - Association among two or more entities (e.g., Mohammad is teaching 15-415)
  - Described using a set of attributes

- Relationship set:
  - Collection of similar relationships
  - Same entity set could participate in different relationship sets, or in different “roles” in the same set

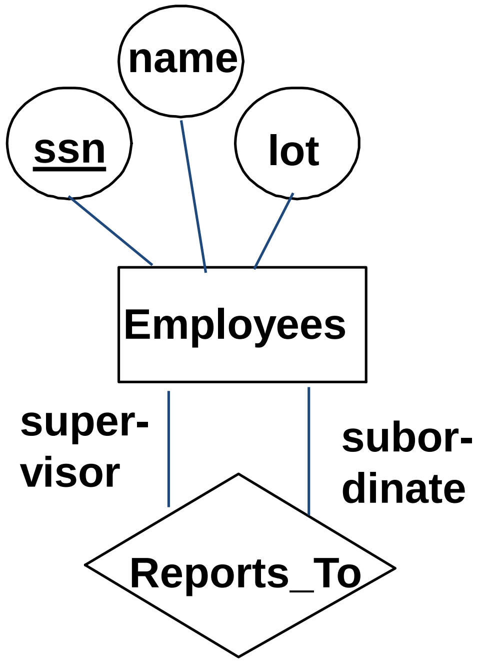
More Tools and ER Diagrams



Relationships ('rel. sets') and mapping constraints



A Binary Relationship

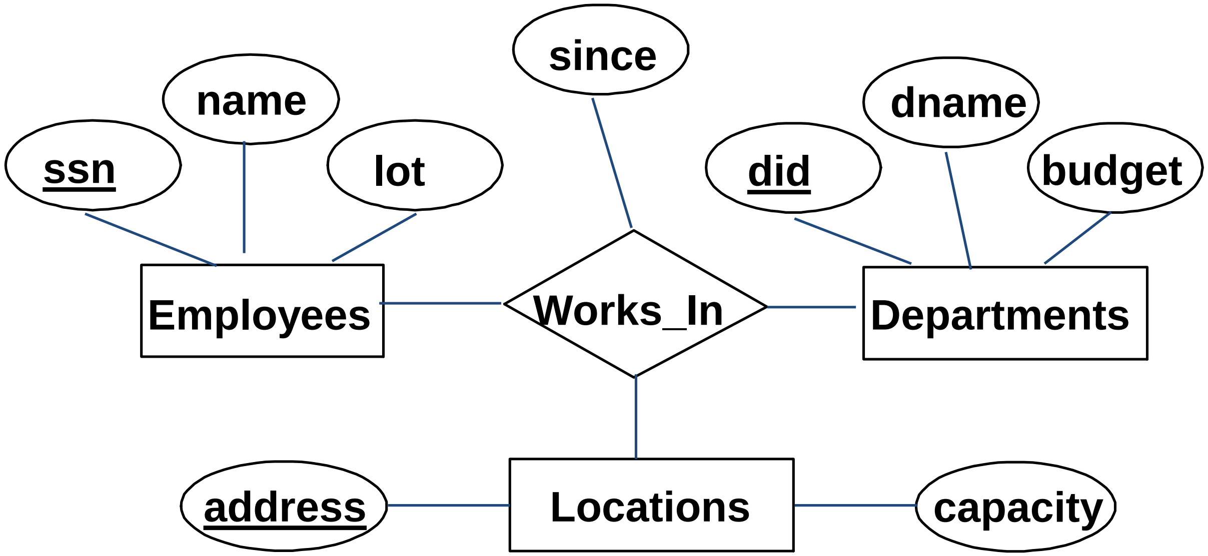


A Self-Relationship

Ternary Relationships

- Suppose that departments have offices at different locations and we want to record the locations at which each employee works

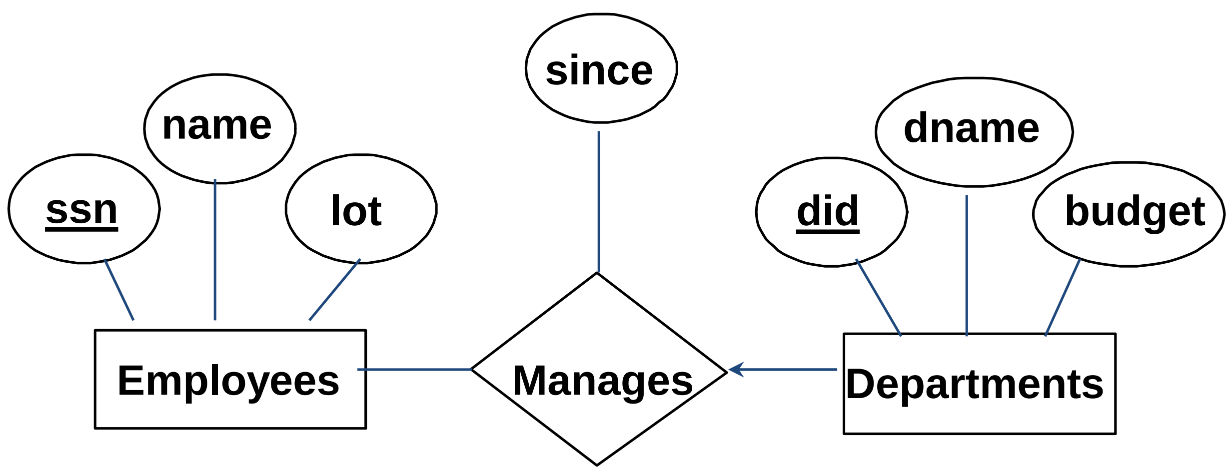
- Consequently, we must record an association between an employee, a department and a location



This is referred to as a "Ternary Relationship" (vs. Self & Binary Relationships)

Key Constraints

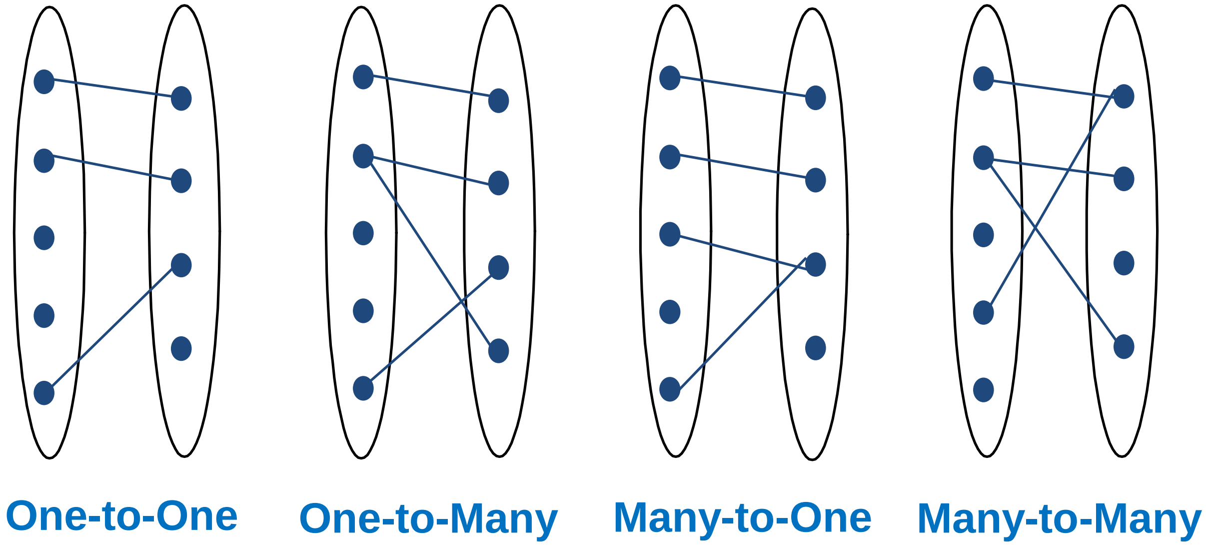
- Consider the “Employees” and “Departments” entity sets with a “Manages” relationship set
  - An employee can work in *many* departments
  - A department can have *many* employees
  - **This restriction is an example of a “key constraint”**

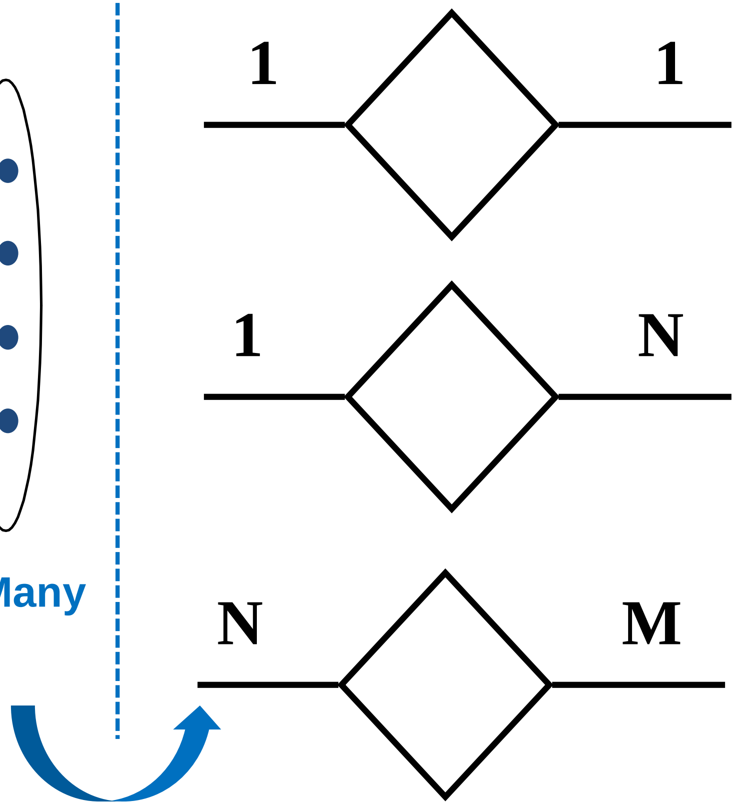


Key constraints are
denoted by *thin arrows*

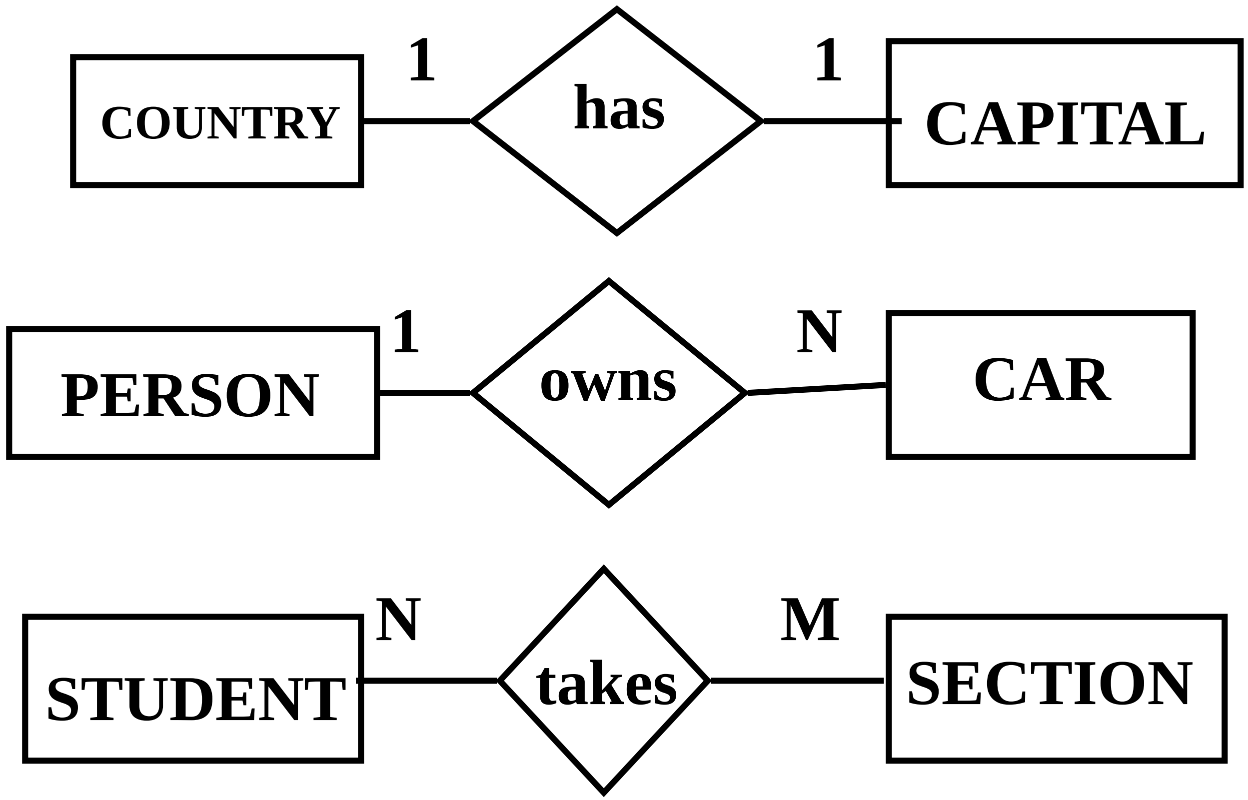
Cardinalities

- Entities can be related to one another as “one-to-one”, “one-to-many” and “many-to-many”
  - This is said to be the cardinality of a given entity in relation to another



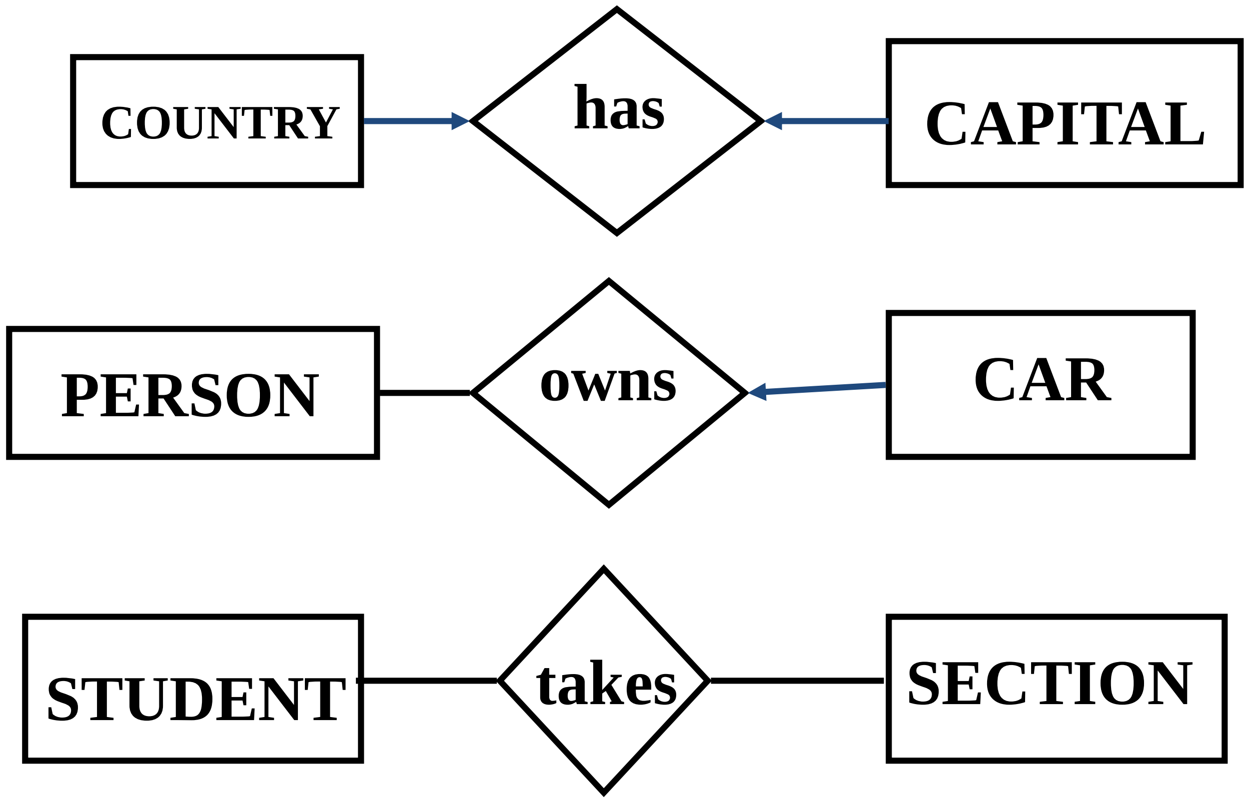


Cardinalities: Examples

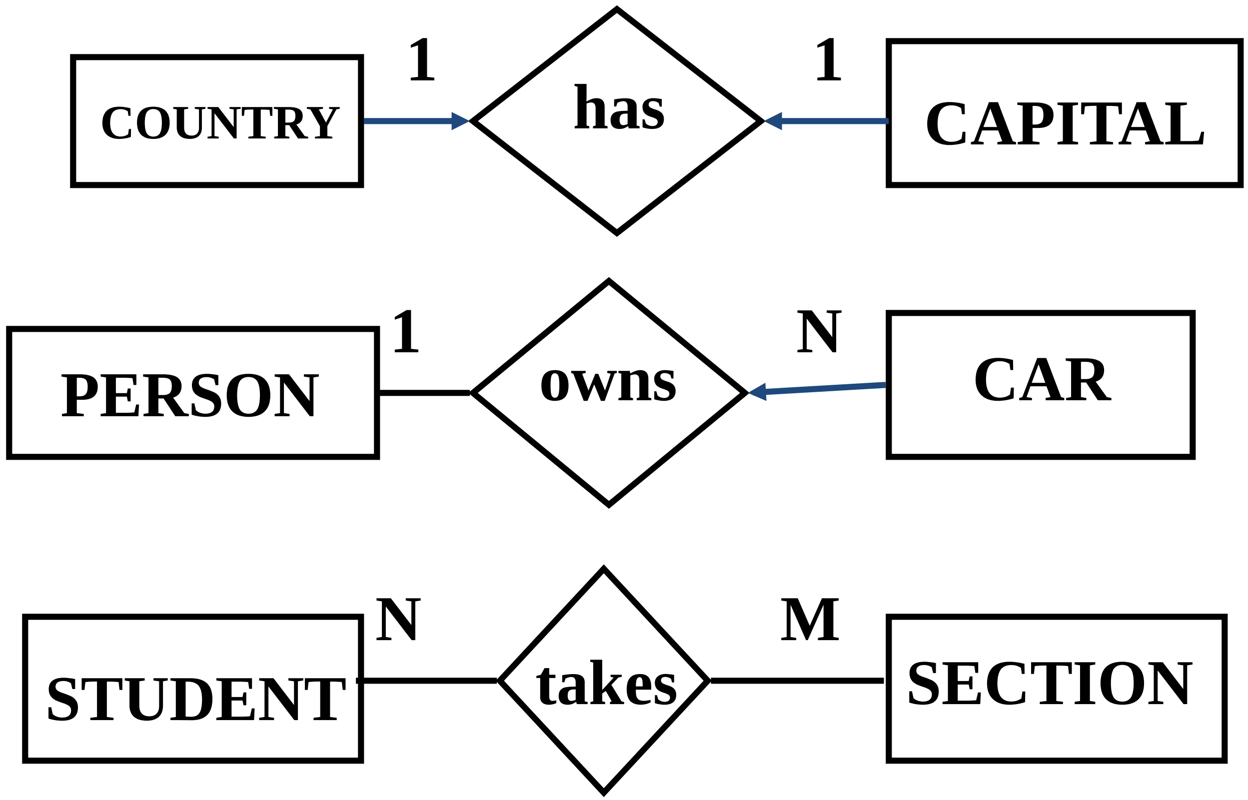


Cardinalities: Examples

Book's notation:



Cardinalities: Examples



Book's notation

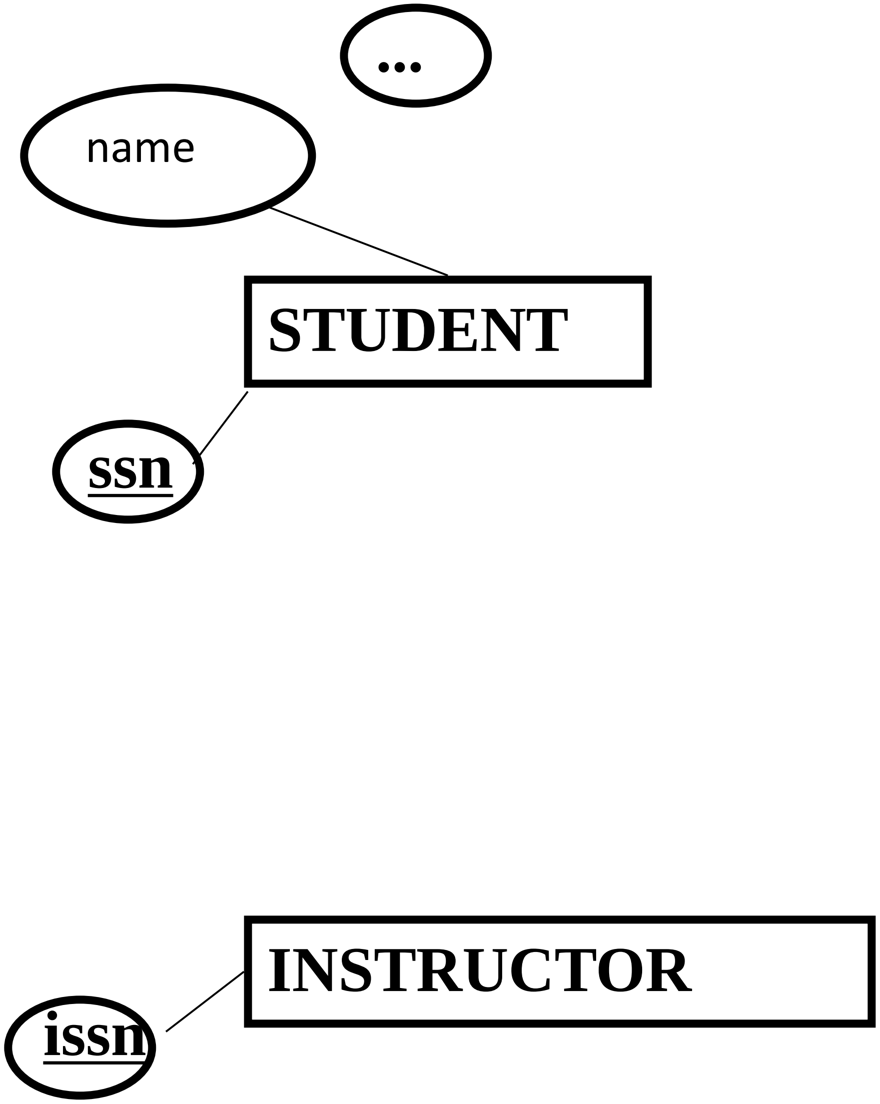
vs

***1 to N* notation**

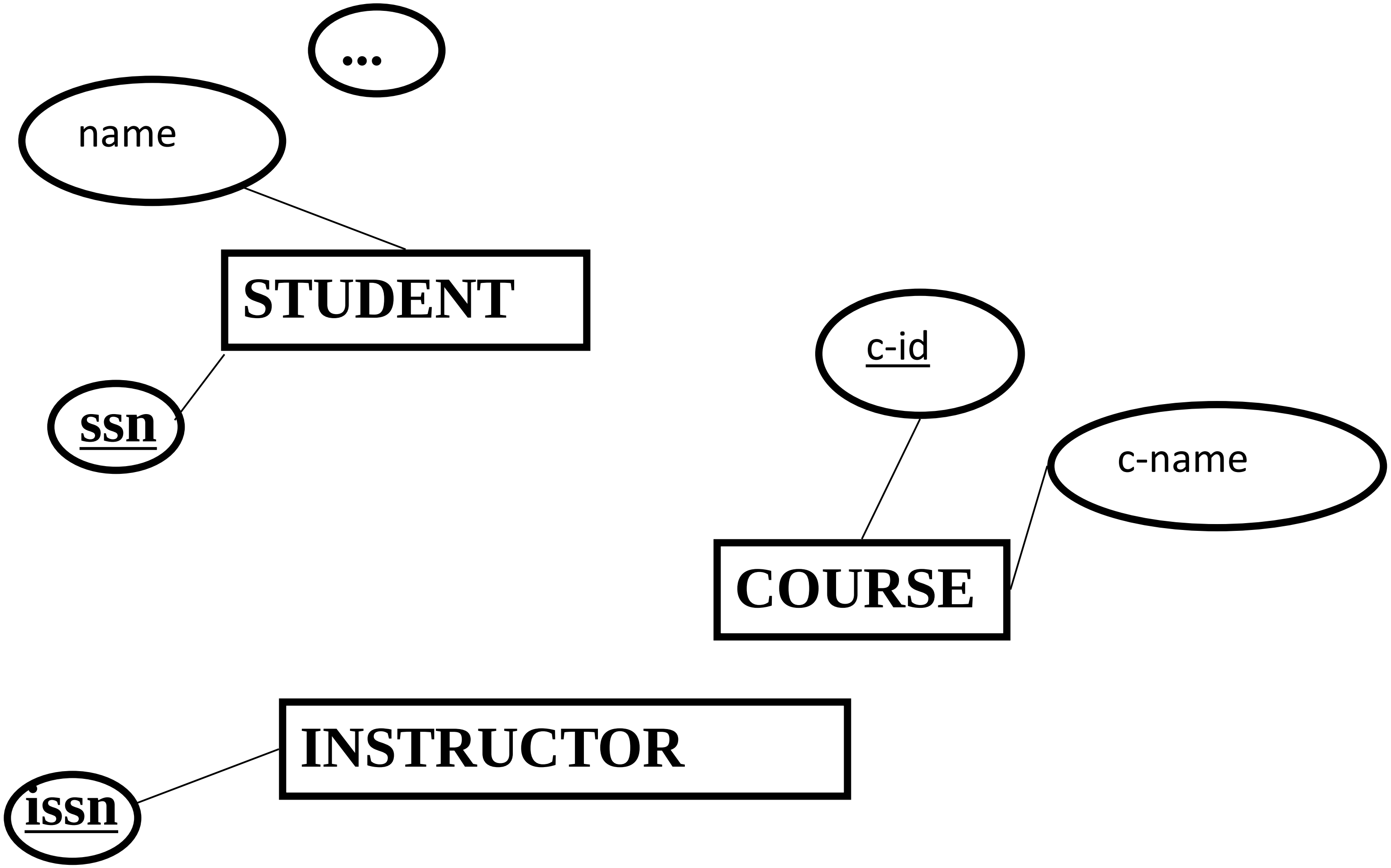
A Working Example

- Requirements: Students take courses offered by instructors; a course may have multiple sections; one instructor per section

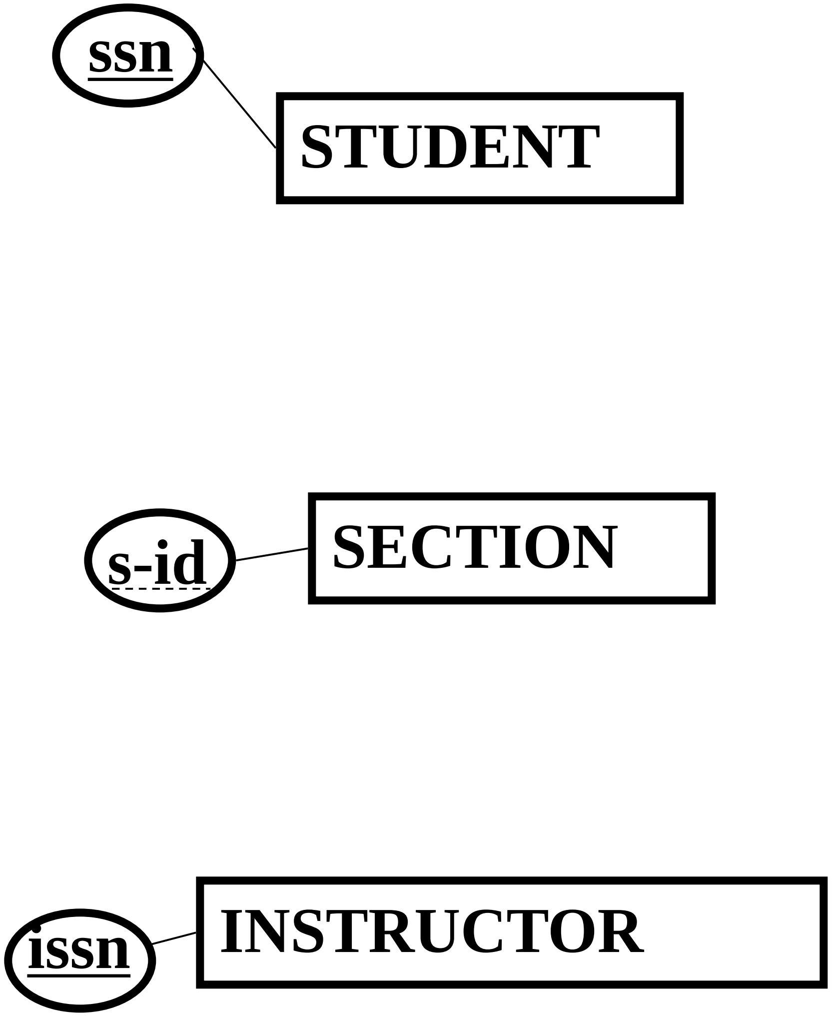
- How to start?
  - Nouns -> entity sets
  - Verbs -> relationship sets

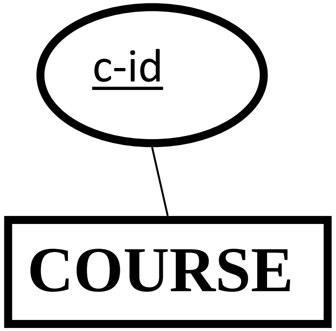


Primary key = unique identifier → <u>underline</u>

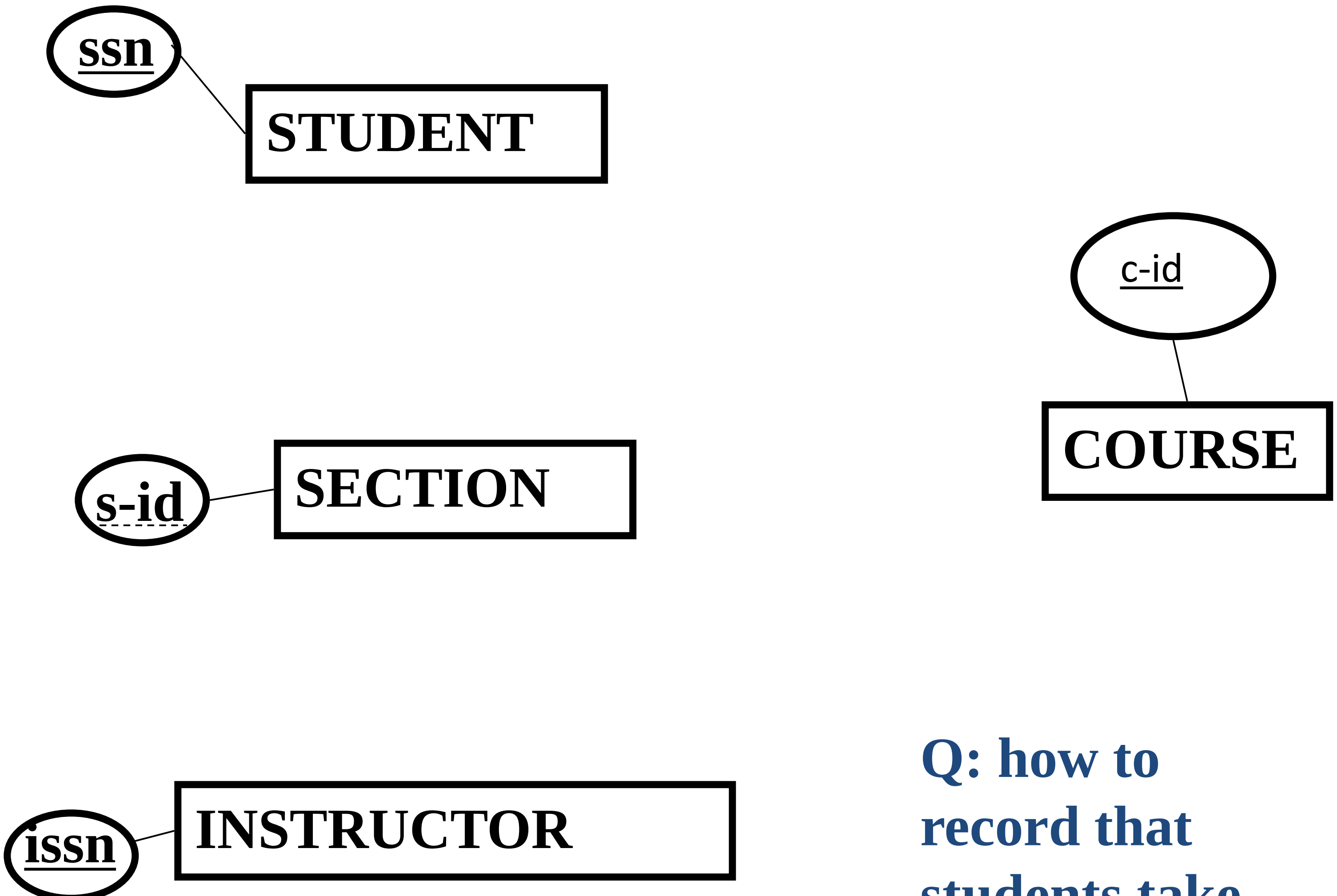


But: sections of a course (with different instructors)?

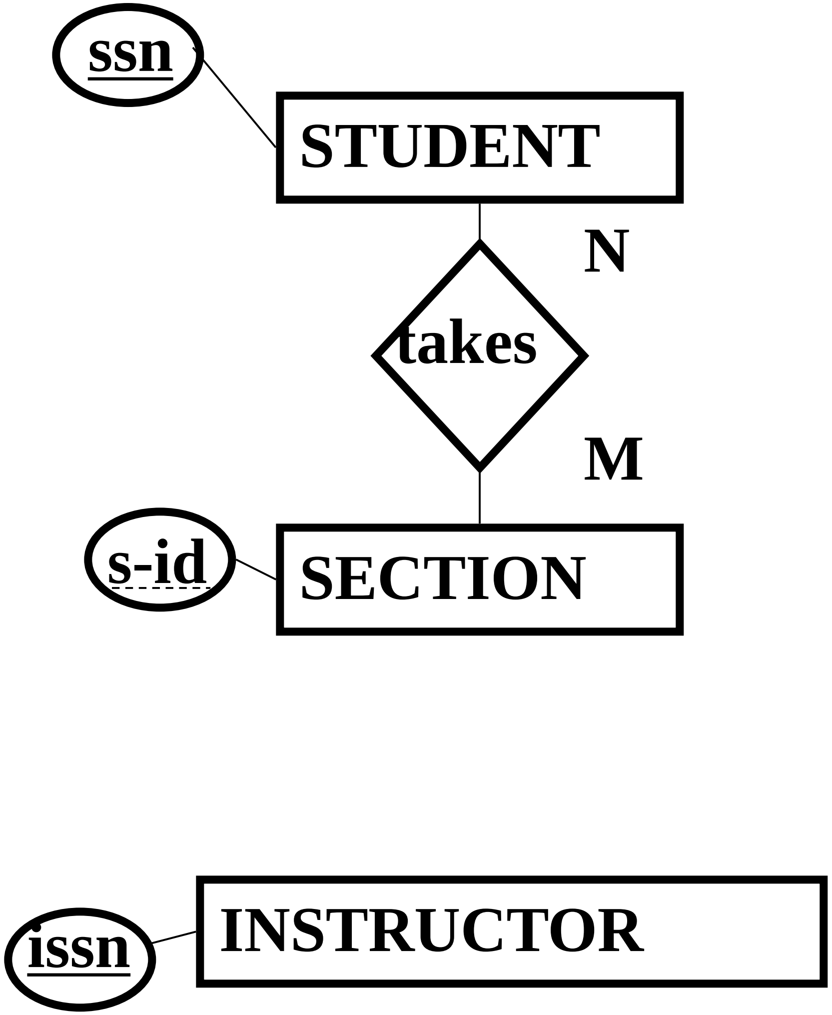


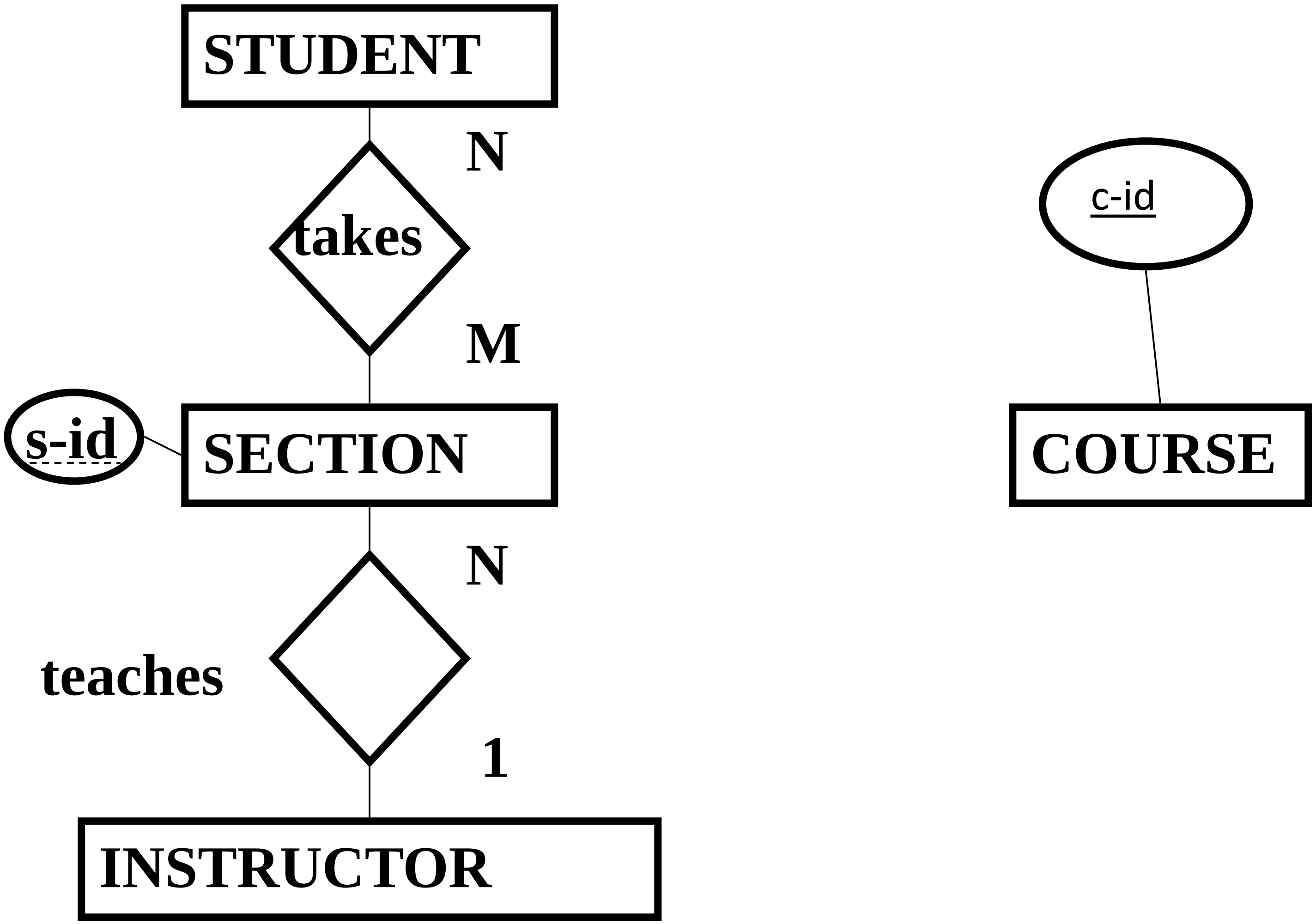


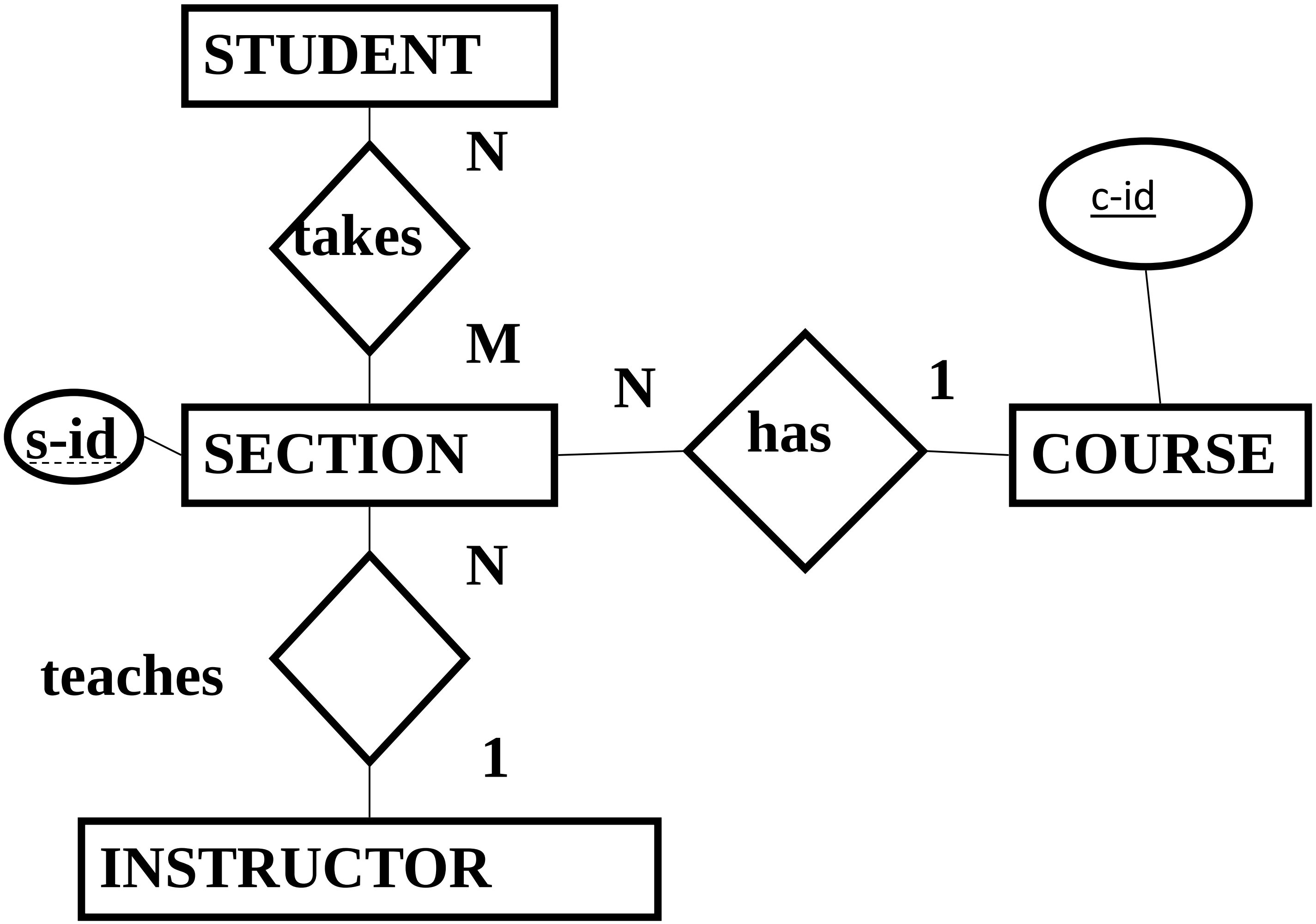
But: s-id is not unique... (see later)



Q: how to record that students take courses?

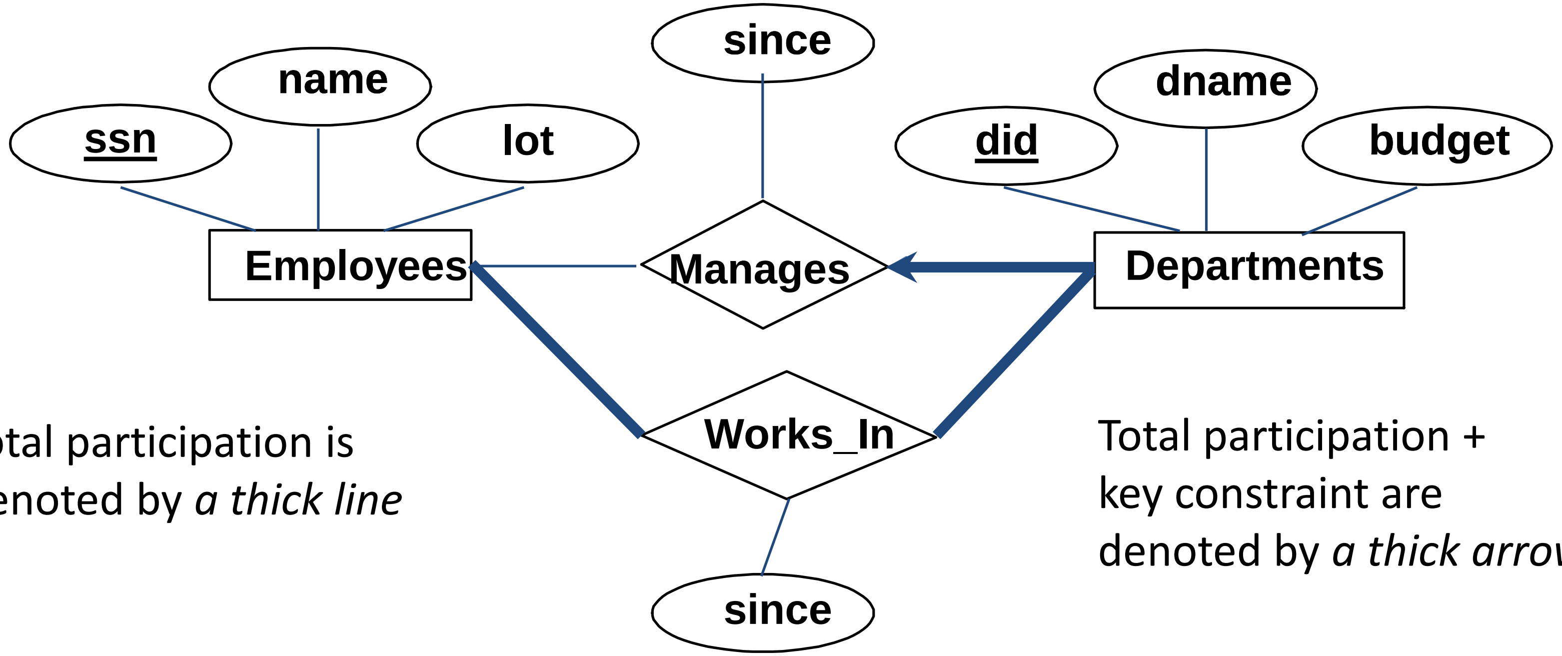

ssn
STUDENT
N
takes
M
s-id
SECTION
c-id
COURSE
issn
INSTRUCTOR


STUDENT
N
takes
M
c-id
s-id
SECTION
COURSE
N
teaches
1
INSTRUCTOR


STUDENT
N
takes
M
c-id
N
has
1
s-id
SECTION
COURSE
N
teaches
1
INSTRUCTOR

Participation Constraints

- Consider again the “Employees” and “Departments” entity sets as well as the “Manages” relationship set
  - Should every department have a manager?
  - If so, this is a participation constraint
  - Such a constraint entails that every Departments entity must appear in an instance of the Manages relationship
  - The participation of Departments in Manages is said to be total (vs. partial)



Total participation is denoted by *a thick line*

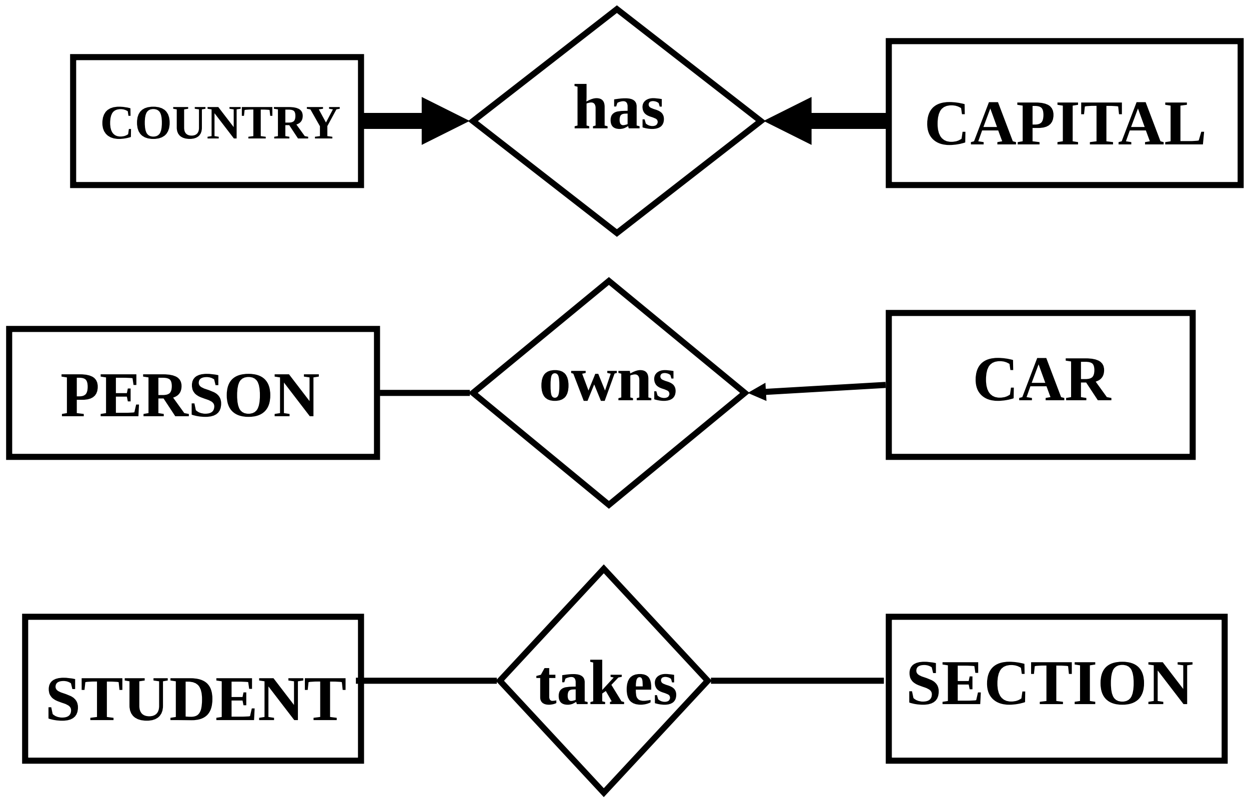
Total participation + key constraint are denoted by *a thick arrow*

Total vs. Partial Participations

Total, Total

??

??

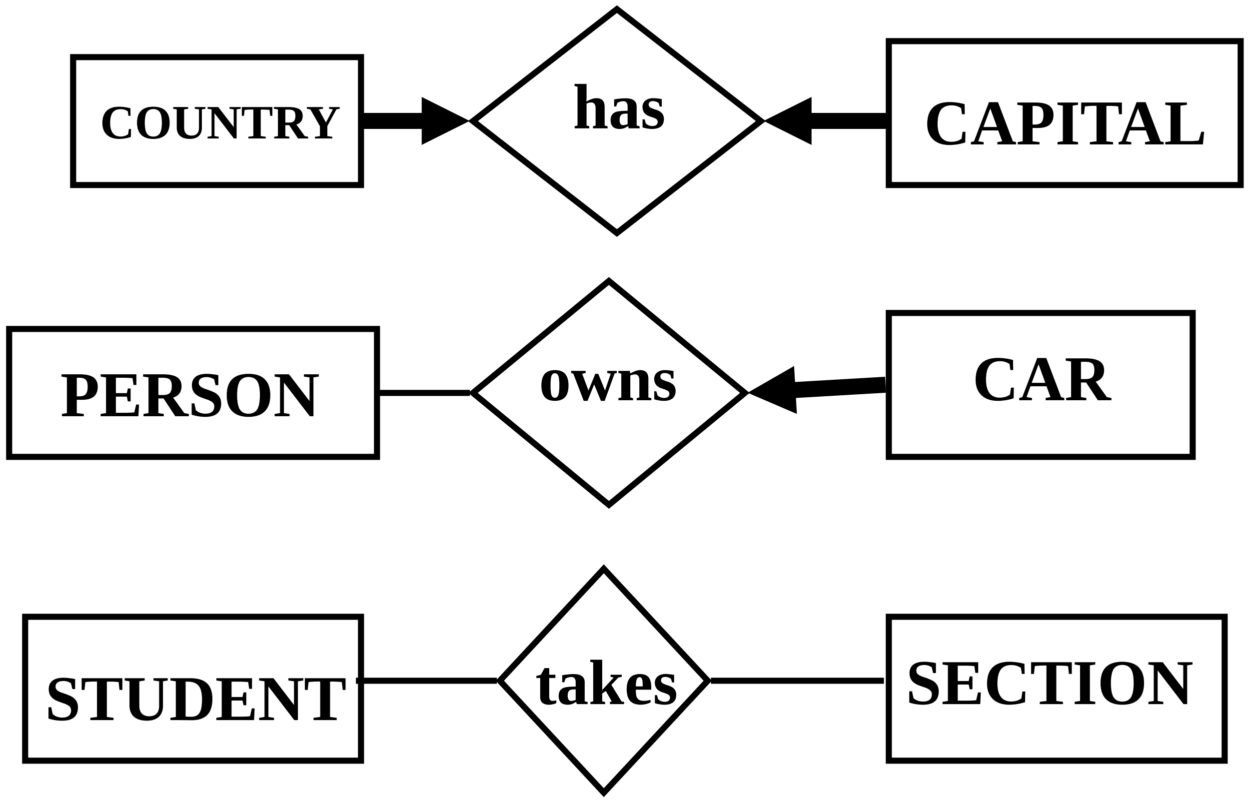


Total vs. Partial Participations

Total, Total

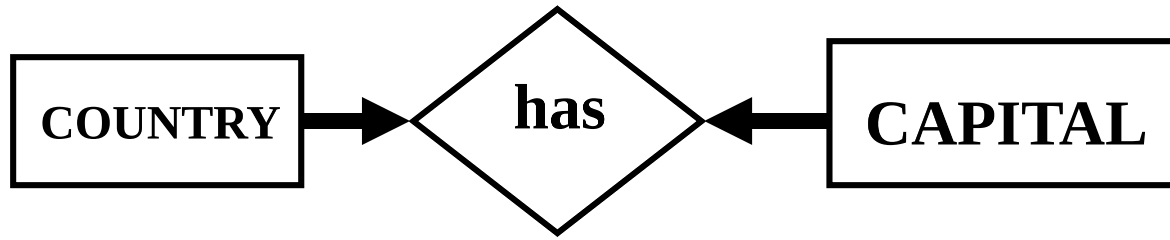
Partial, Total

??

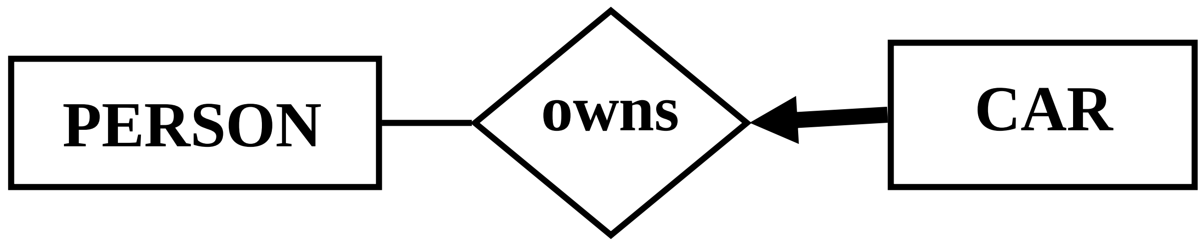


Total vs. Partial Participations

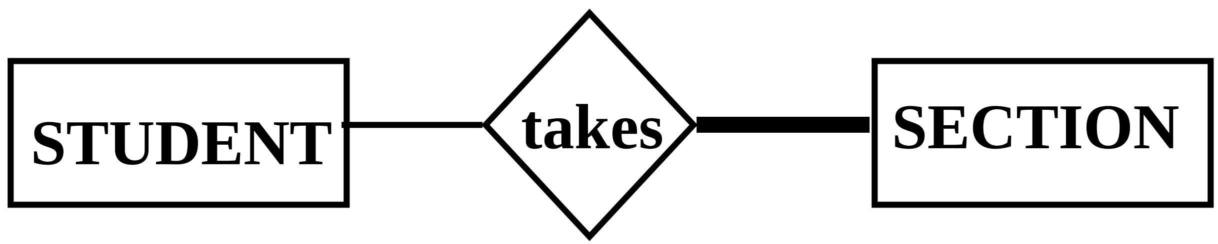
Total, Total



Partial, Total



Partial, Total



Weak Entities

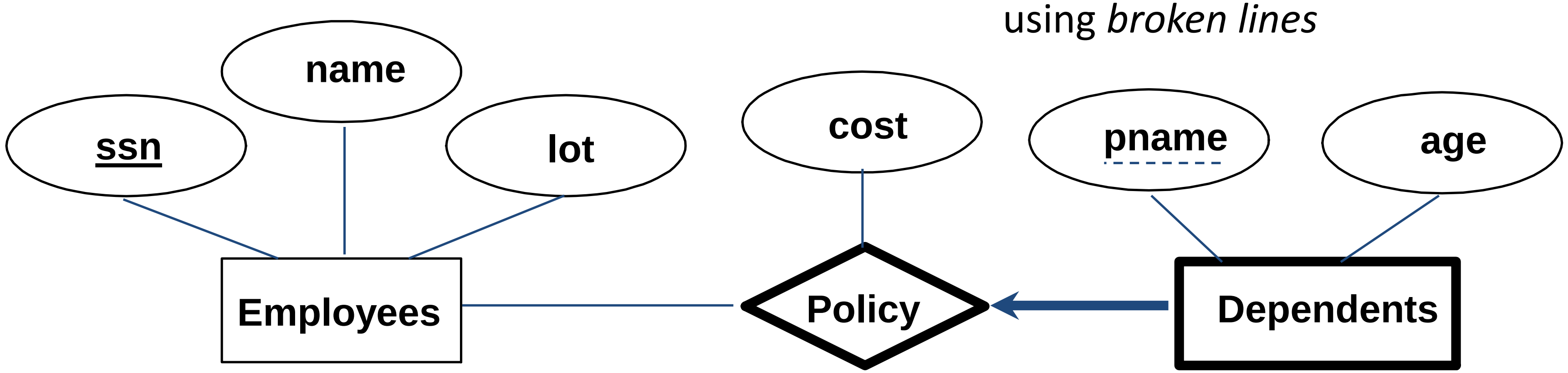
- A weak entity can be identified uniquely only by considering the primary key of another (*owner*) entity
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities)
  - Weak entity set must have total participation in this identifying relationship set

- The set of attributes of a weak entity set that uniquely identify a weak entity for a given owner entity is called partial key

Weak Entities: An Example

- "Dependents" has no unique key of its own
  - "Dependents" is a weak entity with partial key "pname"
  - "Policy" is an identifying relationship set
  - "pname" + "ssn" are the primary key of "Dependents"

Partial keys are underlined using *broken lines*



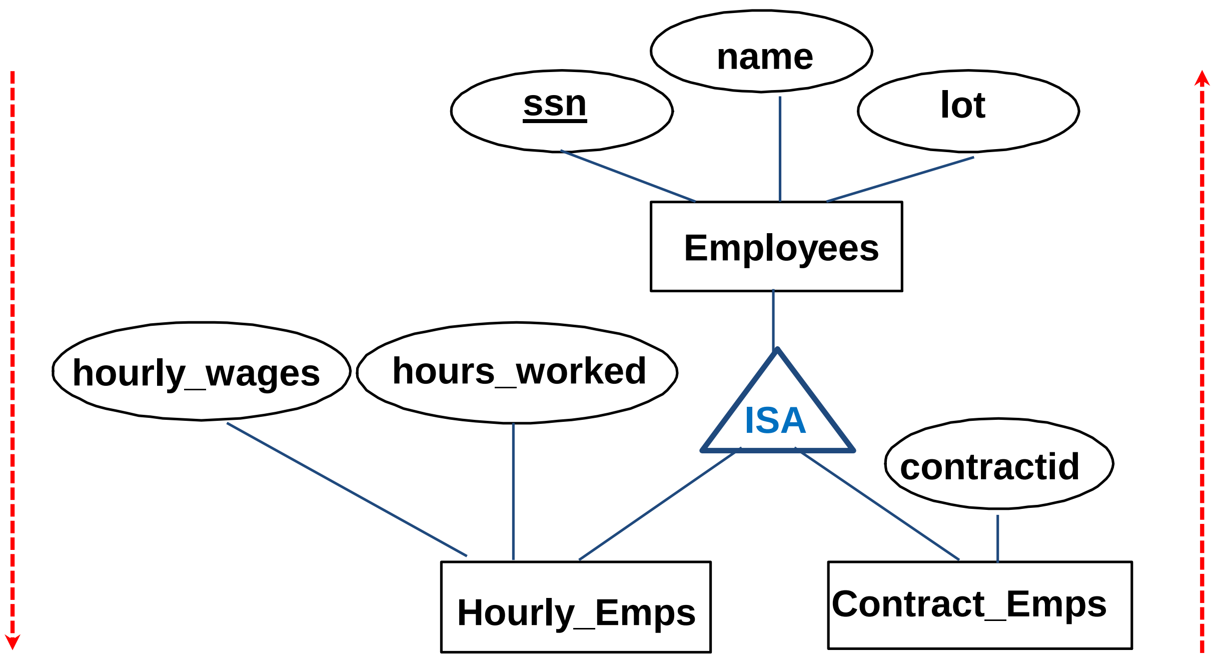
Weak entities and identifying relationships are drawn using *thick lines*

ISA (`is a’) Hierarchies

- Entities in an entity set can sometimes be classified into subclasses (this is “kind of similar” to OOP languages)

- If we declare B **ISA** A, every B entity is also considered to be an A entity

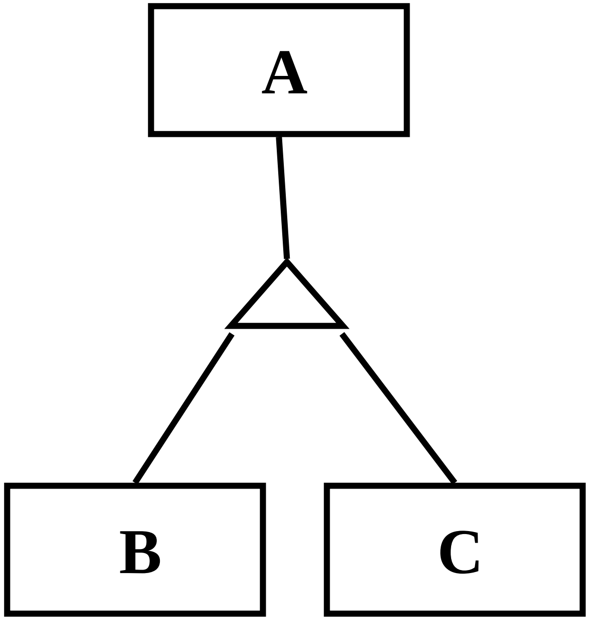
“Employees” is specialized into subclasses



“Hourly_Emps” and “Contract_Emps” are generalized by “Employees”

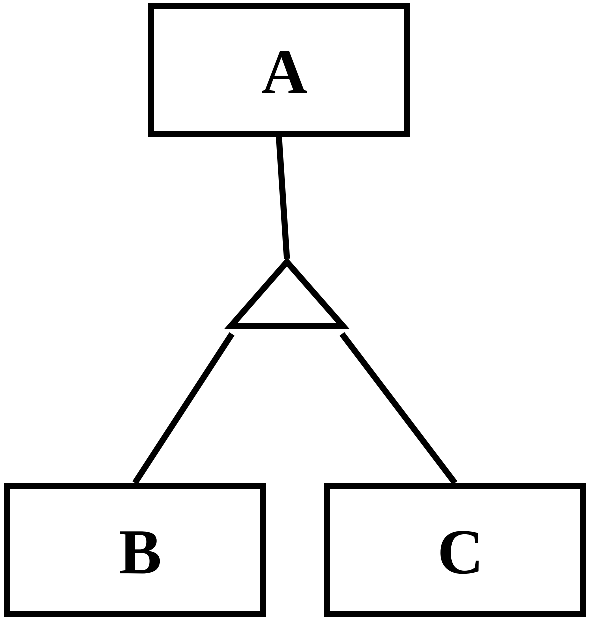
Overlap and Covering Constraints

- Overlap constraints
  - Can an entity belong to both ‘B’ and ‘C’?
- Covering constraints
  - Can an ‘A’ entity belong to neither ‘B’ nor ‘C’?



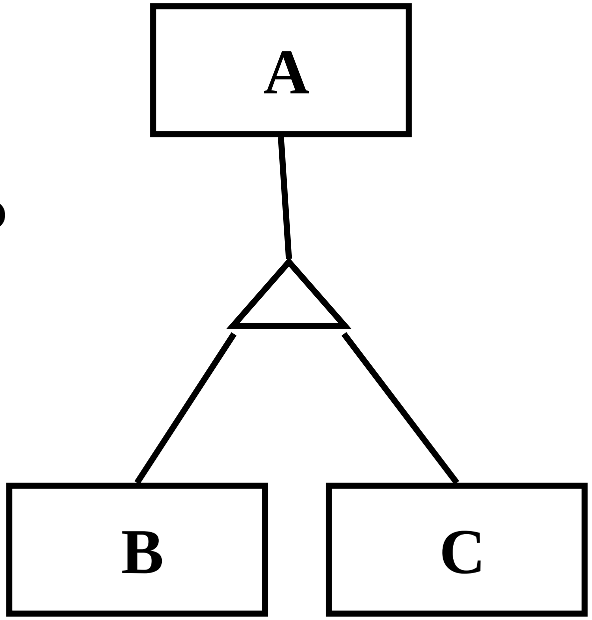
Overlap Constraints: Examples

- Overlap constraints
  - Can John be in Hourly_Emps and Contract_Emps? Intuitively, *no*
  - Can John be in Contract_Emps and in Senior_Emps? Intuitively, *yes* ➔ "Contract_Emps OVERLAPS Senior_Emps"



Covering Constraints: Examples

- Covering constraints
  - Does every one in Employees belong to a one of its subclasses? Intuitively, *no*
  - Does every Motor_Vehicles entity have to be either a Motorboats entity or a Cars entity? Intuitively, *yes* → "Motorboats AND Cars COVER Motor_Vehicles"



More Details on ISA Hierarchies

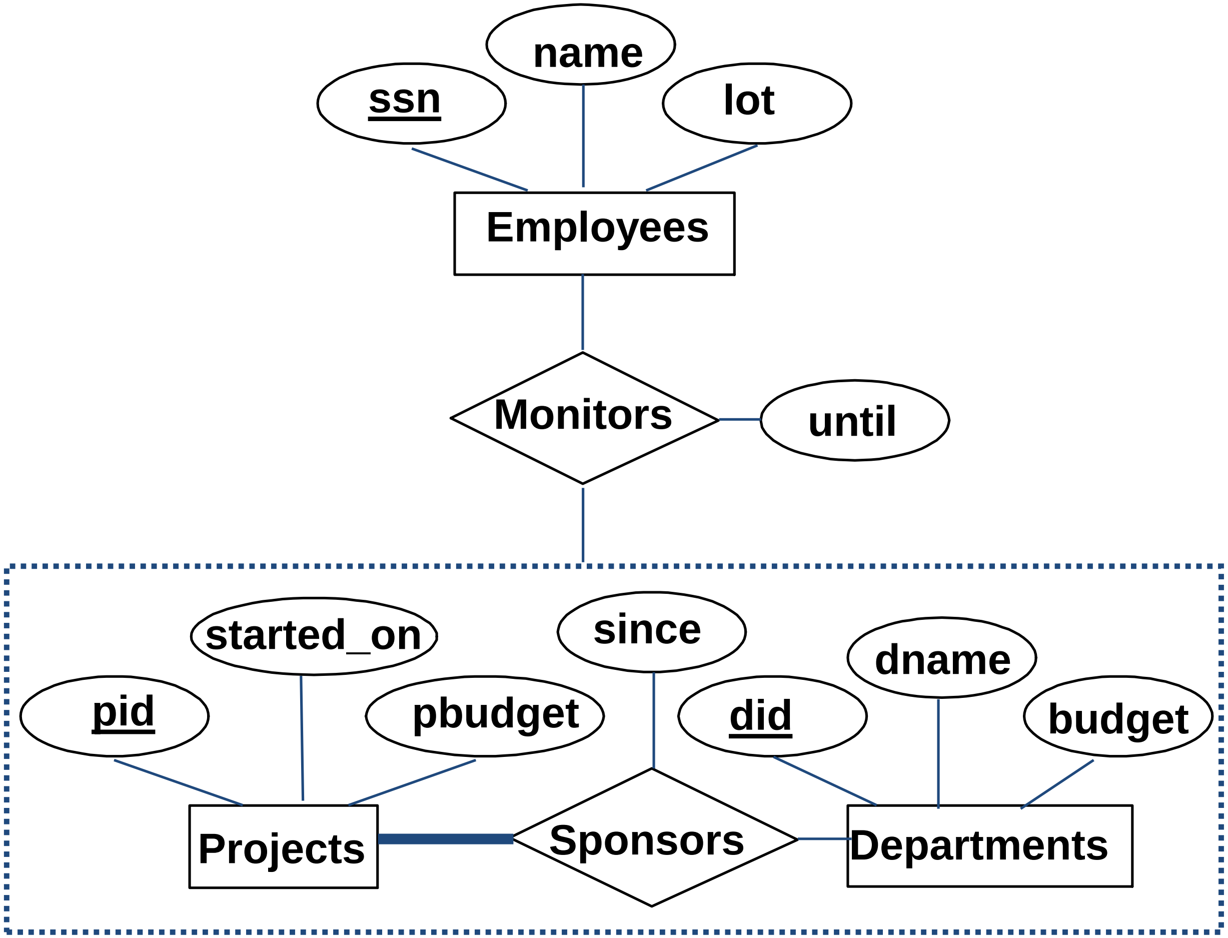
- Attributes are *inherited* (i.e., if B ISA A, the attributes defined for a B entity are the attributes for A *plus* B)

- We can have ***many*** levels of an ISA hierarchy

- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass
  - To identify entities that participate in a relationship

Aggregation

- Aggregation allows indicating that a relationship set (identified through a *dashed box*) participates in another relationship set



Next Class

Continue the ER Model and Start with the relational Model