



# SDSC 5003


## Introduction

Instructor:

Yu Yang [yuyang@cityu.edu.hk](mailto:yuyang@cityu.edu.hk)



[Terence Chan](mailto:CITYUDS.terenceChan@cityu.edu.hk) CITYUDS.terenceChan@cityu.edu.hk



# SDSC5003 Topics

- DB specification and implementation
  - Database design – the relational model and the ER model
  - Creating and accessing a database
    - Relational algebra
    - Creating and querying a DB using SQL
    - Query optimization, transaction processing, ...
  - *Database application development*
- Spark/Hadoop (more of data processing systems)
  - Solution to parallel massive data manipulation
  - Basic programming model



# Tentative Schedule:

## Terence CHAN

Week 1	Introduction & ER Model
Week 2	Relational Model
Week 3	Relational Algebra
Week 4	SQL Queries I
Week 5	SQL Queries II & SQL Constraints
<b>Week 6</b>	<b>No teaching due to CityU Info Day</b>
Week 7	Design Theory

## Yu YANG

Week 8	Time Complexity & Storage and Indexing I
Week 9	Storage and Indexing II
Week 10	Storage and Indexing III & Query Evaluation
Week 11	Parallel Data Processing & MapReduce
Week 12	Spark & Overview of Transaction Management
Week 13	Vector DB & Course Review



# Textbook

- **[RG] *Database Management Systems (3rd edition)***, Raghu Ramakrishnan and Johannes Gehrke.

The website for the book is <http://www.cs.wisc.edu/~dbbook/>.  
The website contains solutions to odd-numbered exercises and other resources.

- **[GUW] *Database Systems: The Complete Book (2nd edition)***, Hector Garcia-Molina, Jeffrey Ullman, and Jennifer Widom.



# Method of Assessment

- [Academic Calendar 2025/26](#)
- **Individual** Assignments: 30%
- Final Exam: 50% (Dec 8-20)
- **Group** Project:
  - **Track 1: Database Application Development**
  - **Track 2: Research Paper Replication**
  - 3-5 students per team
  - **Assessment weighing**
    - Source Code (30%)**
    - Video Demonstration (20%)**
    - Written Report (50%)**



# A Survey

- [HKQF](#)

- [Qualifications Search](#)

- What is your undergraduate major?

- Computer Science (including Software Engineering, IoT and other closed related majors)

- Engineering (Industrial/System/Electrical & Electronic/Mechanical/...)

- Math/Statistics/Physics

- Business

- Others



# What is Data Storing and Retrieving?

- Data Management (Both **Engineering** and Science)
- Database
- How to Design and Use Databases



# What is a Database?

- A **database** contains information that is relevant to some enterprise
- The main goal of a database is to **store** and **retrieve** this information, e.g. CityU (student, faculty, course, classroom, office...)
- Databases typically contain **large amounts** of information
- It should be possible to access this information efficiently and securely, e.g. Amazon has 40+ categories of products, 10K+ sub-categories, > 12 million products, over 100 million users ...





# What is a Database System?

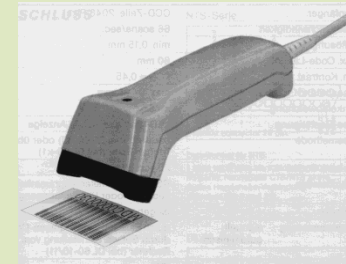
- A database system consists of two components
- Database (DB) and
- Database Management System (DBMS)
- The DB contains the data
- The DBMS is software that stores, manages and retrieves the information in the DB



# Why We Need Databases?

# Data in the Current Millenium

- Consider Walmart
- Which handles more than 1 million customer transactions per hour
- Imported into databases estimated to contain more than 2.5 petabytes (2,560 terabytes) of data
- Gigabyte –  $2^{30}$  bytes
- Terabyte –  $2^{40}$  bytes
- Petabyte –  $2^{50}$  bytes



Customer Transactions

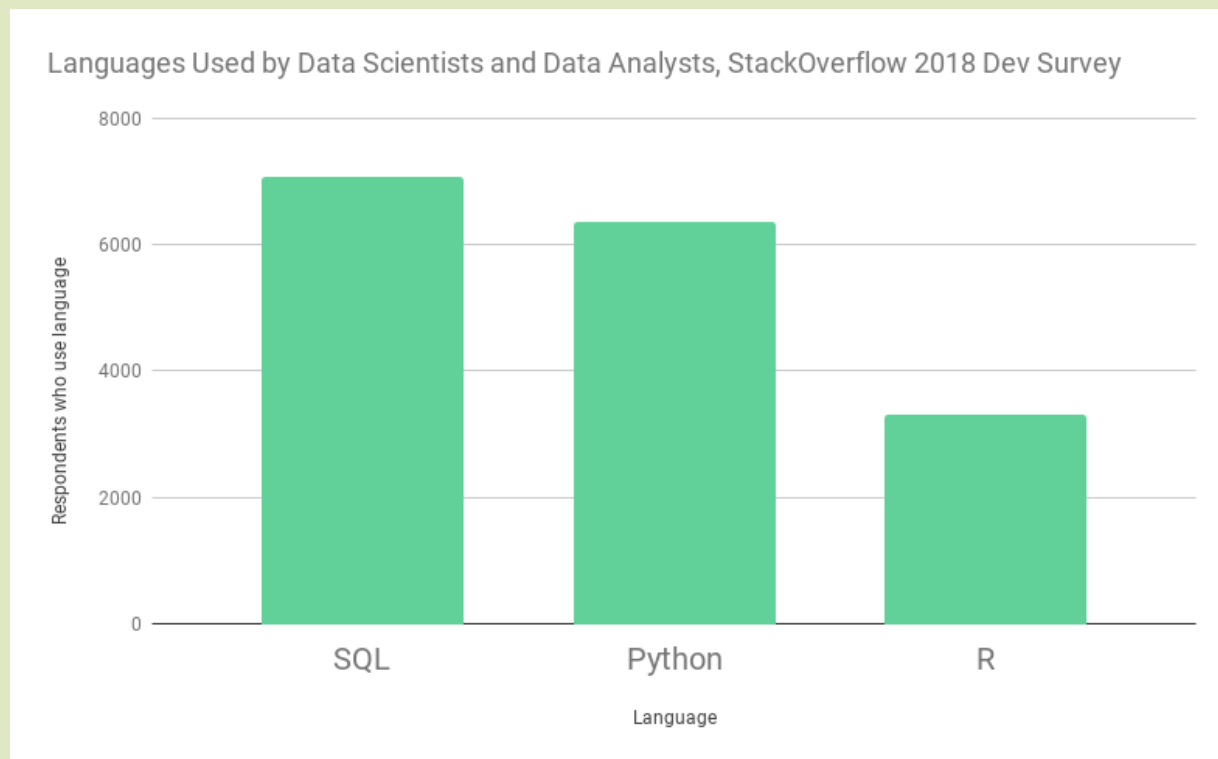


# Database Applications

- Any application that has to store large amounts of data probably needs a database
- Banking
- Airlines
- Universities
- Credit card transactions
- Finances
- Sales
- On-line retailers
- Manufacturing
- Human resources
- MMORPGs
- ...

# Database to Data Scientists

- Massive data
- Preparing Data often takes 80% of time



# Data Storage Without DBMS

- Different data sources and data formats
  - Parsing different data formats is tedious!
  - [DATA.GOV.HK](https://data.gov.hk)

## CSV

	A	B	C	D
1	ID	Gender	City	Monthly_
2	ID000002C	Female	Delhi	20000
3	ID000004E	Male	Mumbai	35000
4	ID000007F	Male	Panchkula	22500
5	ID000008I	Male	Saharsa	35000
6	ID000009J	Male	Bengaluru	100000
7	ID000010K	Male	Bengaluru	45000
8	ID000011L	Female	Sindhudurg	70000
9	ID000012N	Male	Bengaluru	20000
10	ID000013N	Male	Kochi	75000
11	ID000014C	Female	Mumbai	30000
12	ID000016C	Male	Mumbai	25000
13	ID000018S	Female	Surat	25000
14	ID000019T	Female	Pune	24000
15	ID000021V	Male	Bhubanes	27000
16	ID000022V	Female	Howrah	28000

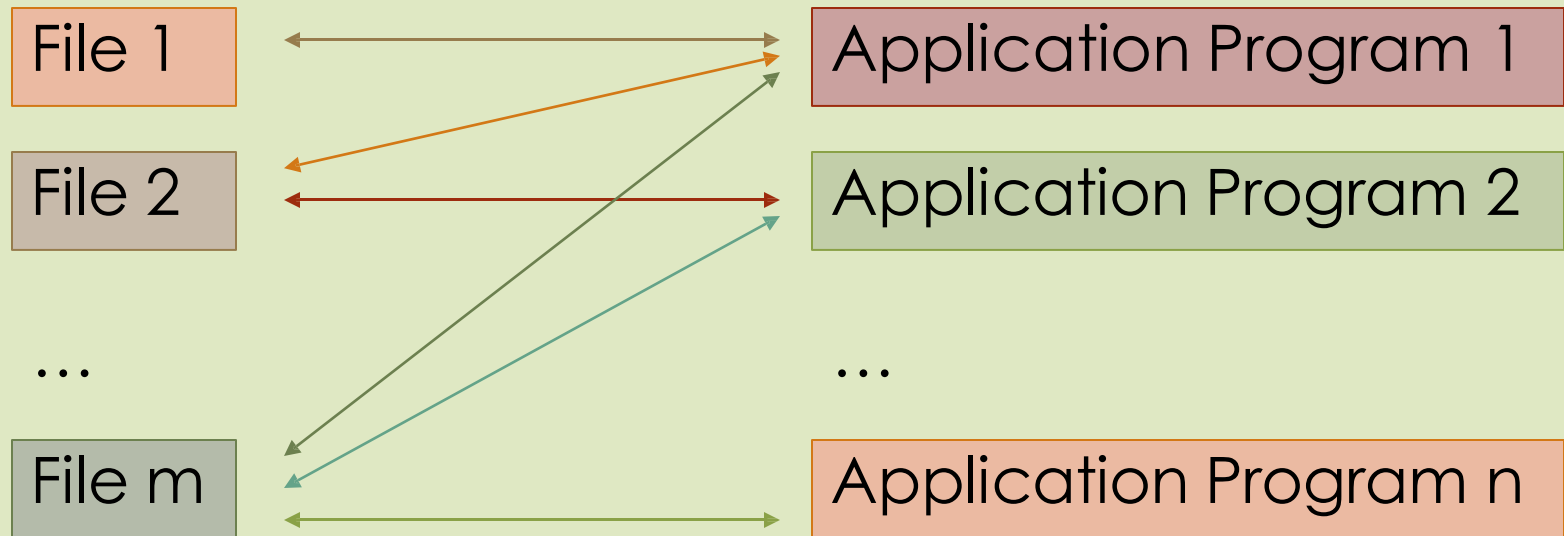
## JSON

```
{  
  "Employee": [  
    {  
      "id": "1",  
      "Name": "Ankit",  
      "Sal": "1000",  
    },  
    {  
      "id": "2",  
      "Name": "Faizy",
```

```
<?xml version="1.0"?>  
  
<contact-info>  
  
  <name>Ankit</name>  
  
  <company>Analytics Vidhya</company>  
  
  <phone>+9187654321</phone>  
  
</contact-info>
```

# Data Storage Without DBMS

- Data would be collected in many different files and
- Used by many application programs



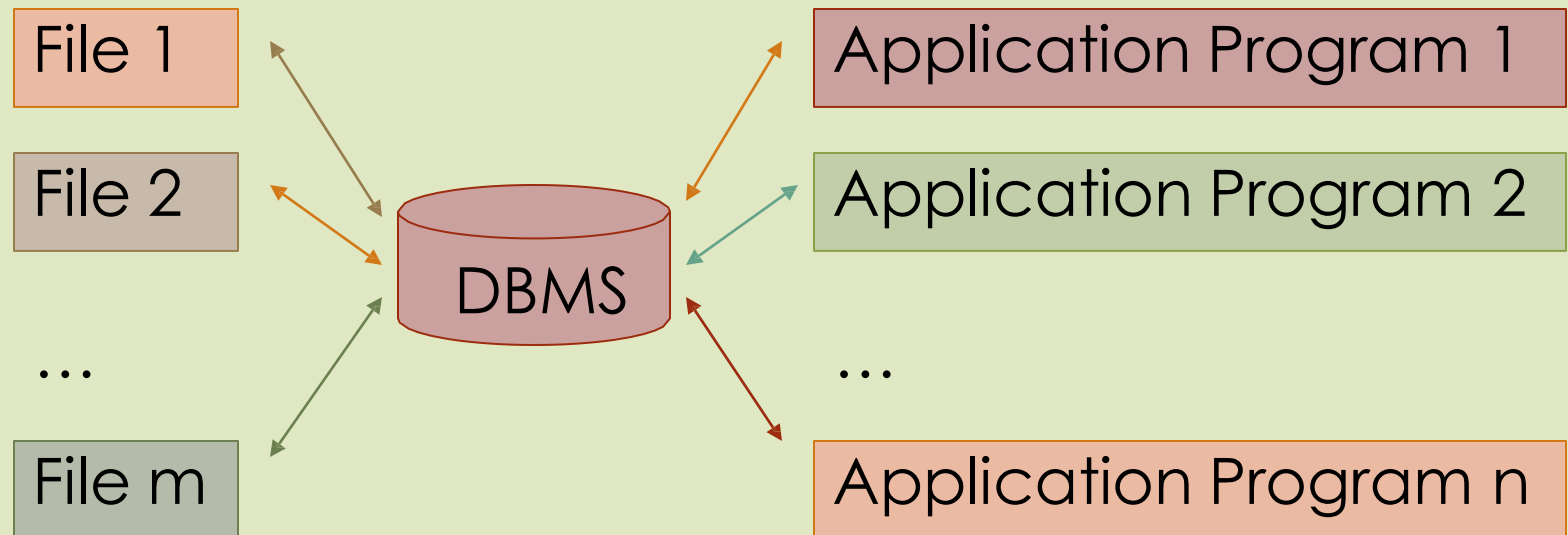


# What Happens If ...

- An attribute is added to one of the files?
- Information that is in more than one file is changed by a program that only interacts with one file?
- We need to access a single record out of millions of records?
- Several programs need to access *and modify* the same record at the same time?
- The system crashes while one of the application programs is running?



# Data Storage with a DBMS





# DBMS Advantages

- All access to data is centralized and managed by the DBMS which provides
- Logical data independence
- Physical data independence
- Reduced application development time
- Efficient access
- Data integrity and security
- Concurrent access and concurrency control
- Crash recovery



# Database Overview

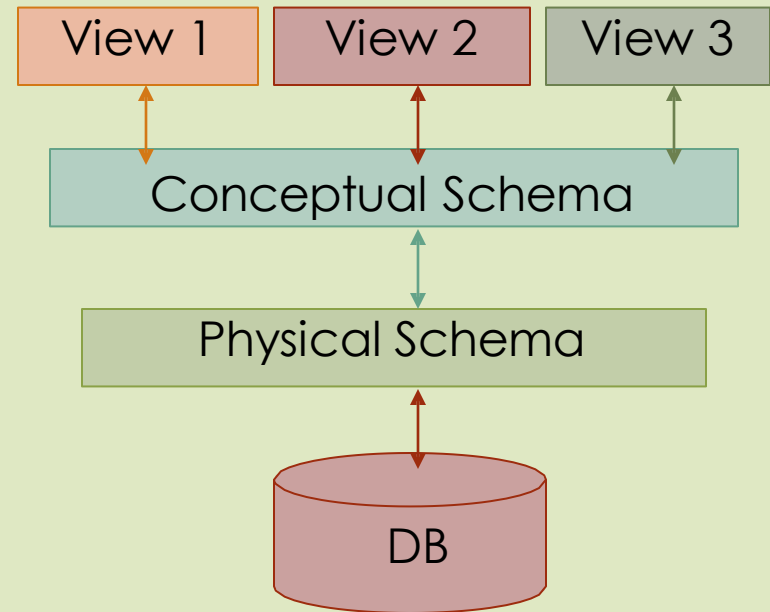


# Data Models

- A **database** models a real-world enterprise, e.g. CityU
- A **data model** is a formal language for describing data
- A **schema** is a description of a particular collection of data using a particular data model
- The most widely used data model is the **relational data model**
- The main concept **relation**, essentially a table with rows and columns
- Each relation has a schema, e.g. `Enrolled(sid: string, cid: string, gpa: real)`

# Data Abstraction

- Physical schema
- Describes how data are stored and indexed
- Conceptual (or logical) schema
- Describes data in terms of the data model
- External (or **view**) schema
- Describes how some users access the data
- There can be many different views, e.g., students who takes SDSC5003 and is older than 20 years old.





# Data Independence

- Physical data independence
  - Allows the physical schema to be modified without rewriting application programs
  - e.g. adding or removing an index or moving a file to a different disk
- Logical data independence
  - Allows the logical schema to be modified without rewriting application programs
  - e.g. adding an attribute to a relation
  - This results in reduced application development and maintenance time

# Example: University Database

- Conceptual schema:
  - *Students(sid: string, name: string, login: string, age: integer, gpa: real)*
  - *Courses(cid: string, cname: string, credits: integer)*
  - *Enrolled(sid: string, cid: string, grade: string)*
- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.
- External Schema (View):
  - *Course\_info(cid: string, enrollment: integer)*



# Efficient Access

- What happens when a user wants to find one record out of millions?
- An index structure maps the desired attribute values to the address of the record
- The desired records can be retrieved without scanning the whole relation





# Concurrency Control

- What happens if two users try to change the same record at the same time?
- A DBMS system ensures that concurrent transactions leave the DB in a consistent state
- While still allowing for maximal possible access of the data
- e.g. many users can read the same record at the same time but only one user at a time can modify a record



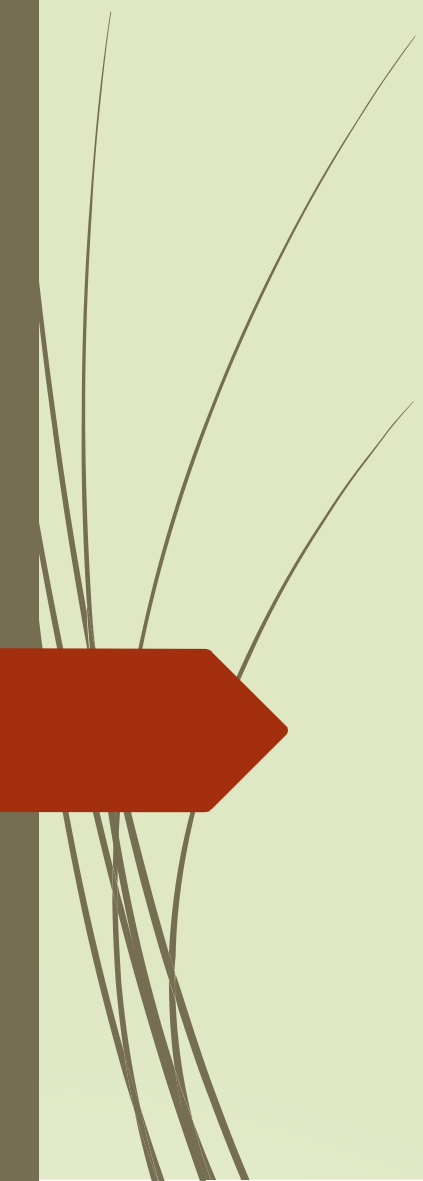
# Data Integrity

- Data should be consistent with the information that it is modeling
- A DBMS cannot actually understand what data represents
- Users can specify integrity constraints on data and a DBMS will then enforce these constraints
- e.g. not allowing ages to be negative



# Transactions

- Changes to a DB occur as a result of *transactions*
- A transaction is a sequence of reads and writes to the DB caused by one execution of a user program
- Transactions must have the **ACID** properties:
- **A**tomic: all or nothing
- **C**onsistent: the DB must be in a consistent state after the transaction
- **I**solated: transactions are performed serially
- **D**urable: the effects of a transaction are permanent
- Log transactions for redo or undo (**crash recovery**)



# Database Languages



# Database Languages

- A database language is divided into two parts
- Data definition language (DDL)
- Data manipulation language (DML)
- Structured query language (SQL) is both a DDL and a DML
- Most commercial databases use SQL and we will cover it in detail in this course



# Data Definition Language

- The DDL allows entire databases to be created, and allows integrity constraints to be specified
- Domain constraints
- Referential integrity
- Assertions
- Authorization
- The DDL is also used to modify existing DB schema
- Addition of new tables
- Deletion of tables
- Addition of attributes



# Data Manipulation Language

- The DML allows users to access or change data in a database
- Retrieve information stored in the database
- Insert new information into database
- Delete information from the database
- Modify information stored in the database



# Database Users





# Database Users

- End users
  - May have specialized knowledge (CAD etc.) and may be familiar with SQL
  - The majority have no DB knowledge
- DB Administrators
  - Have central control over data and programs that access that data
- Database Application Programmers
  - Write programs that need to interact with the DB
- DB Implementers and Vendors
  - Build and sell DB products

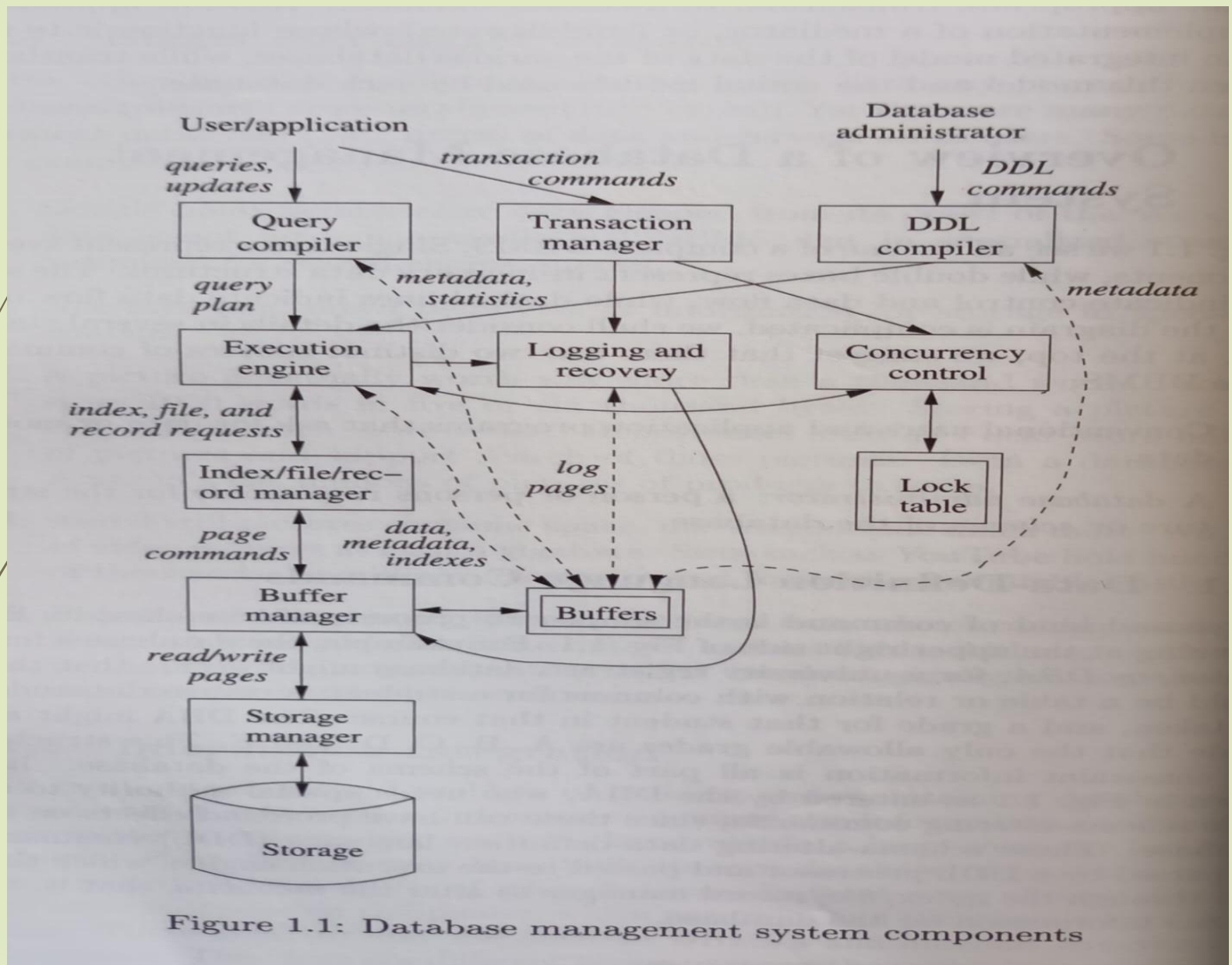


Figure 1.1: Database management system components



# Exercise

Which of the following plays an important role in *representing* information about the real world in a database?

1. The data definition language.
2. The data manipulation language.
3. The buffer manager.
4. The data model.



# The Entity Relationship Model

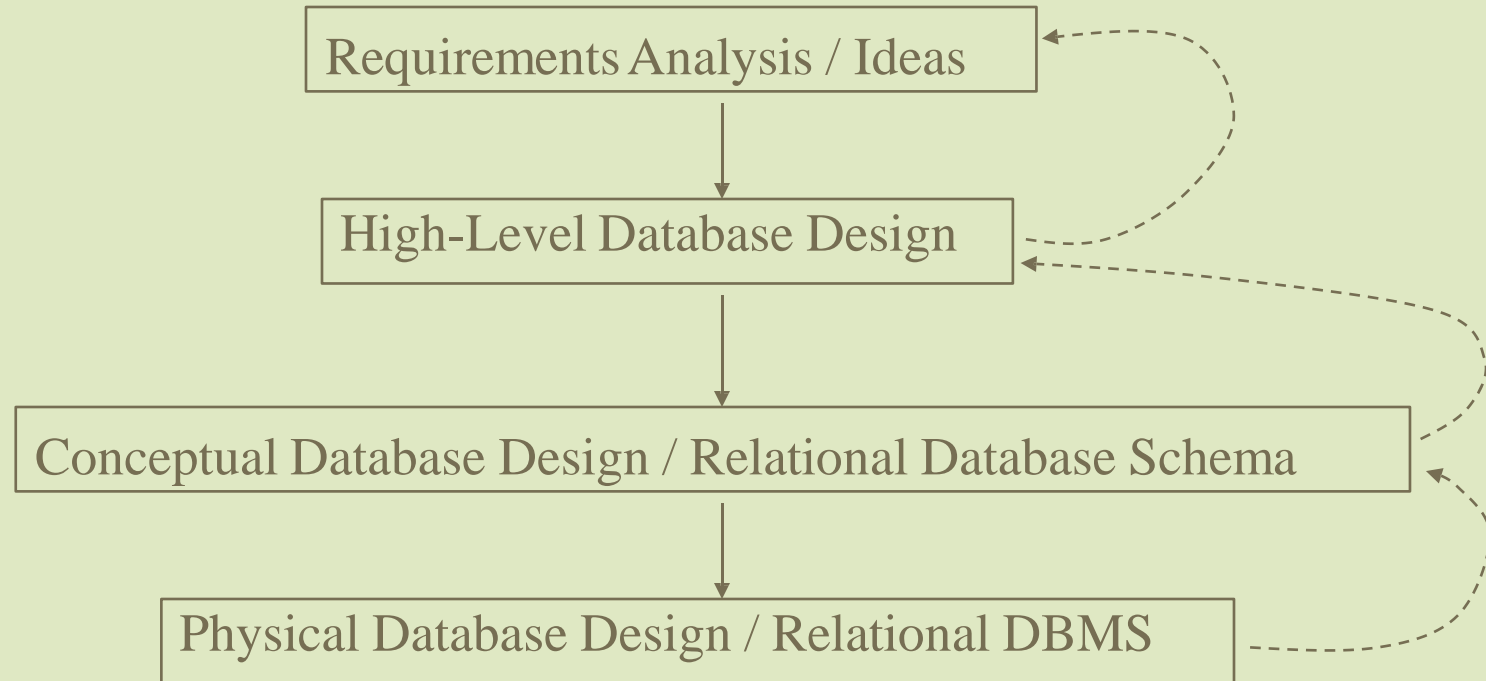
- Database design
- The ER model
  - Key Constraints.
  - Participation Constraints.
  - Weak Entities.
  - ISA Hierarchy.



# Database Design

- **RG Chapter 2**
- G UW Chapter 4.1-4.6

# Design Overview



→ Similar to software development



# Requirements Analysis

- To model a real-world enterprise
- What data are to be stored in the database?
- What applications are required to work with the database?
- Which are the most frequent, and the most important operations?



# Conceptual Database Design

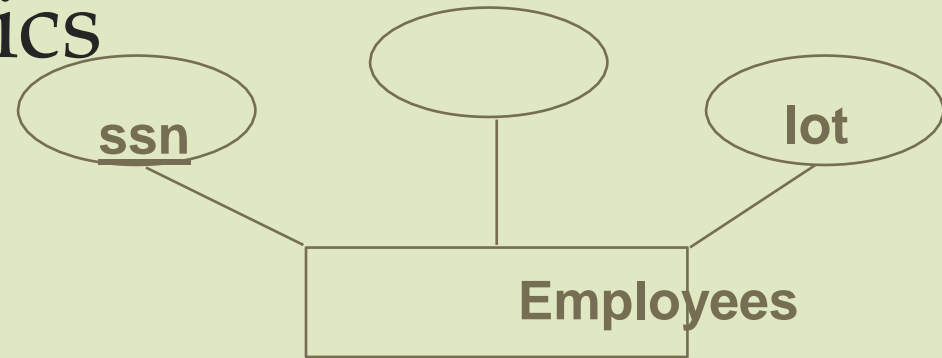
- *ER Model* is used at this stage
- What are the *entities* and *relationships* in the enterprise?
- What information about these entities and relationships should we store in the database?
- What are the *integrity constraints* or *business rules* that hold?
- A database 'schema' in the ER Model can be represented pictorially (*ER diagrams*).
- Can map an ER diagram into a relational schema.



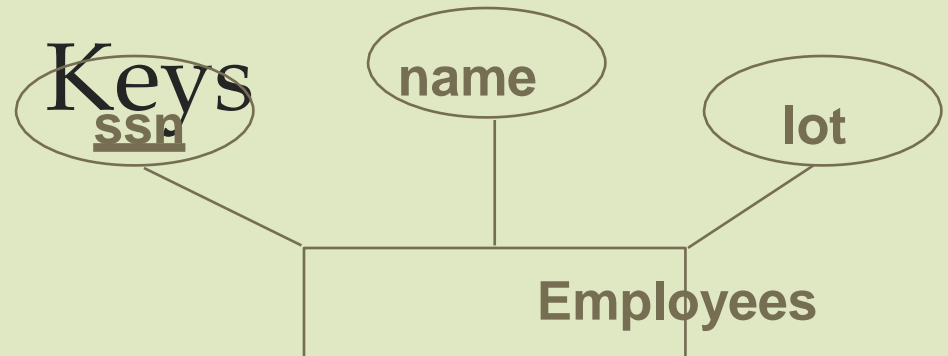


# Entities


# ER Model Basics



- Entity: Real-world object distinguishable from other objects. An entity is described (in DB) 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 (A set of all possible attribute values, no value indicated by NULL).



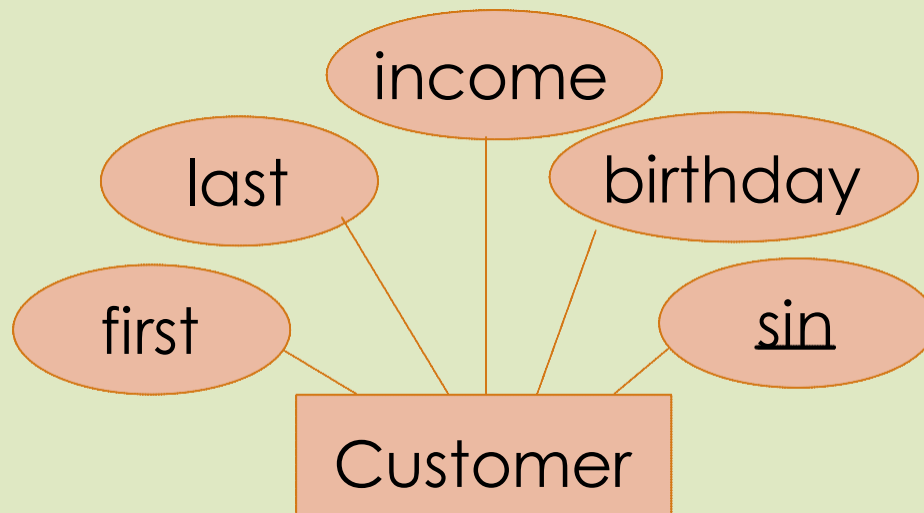
- Differences between entities in an entity set are expressed in terms of their attributes
- That is, you cannot have two different entities that have the same values for each of their attributes
- As they would represent the same real-world entity (but see weak entity sets later)
- A *key* is a set of attributes whose values ***uniquely*** identify an entity **in an entity set**



# Types of Keys

- Superkey
- Any set of attributes whose values uniquely identify an entity in an entity set
- Candidate key
- A minimal superkey, that is a superkey with no *extraneous* attributes
- A relation can have more than one candidate key
- Primary key (must be a candidate key)
- The candidate key designated by the database designer to refer to tuples (rows) in the relation (table)
- Should never (or very rarely) change

# Primary Key



Candidate Key 1: {first, last, birthday}<sup>\*</sup>

Candidate Key 2: {sin}

Primary Key: {sin}

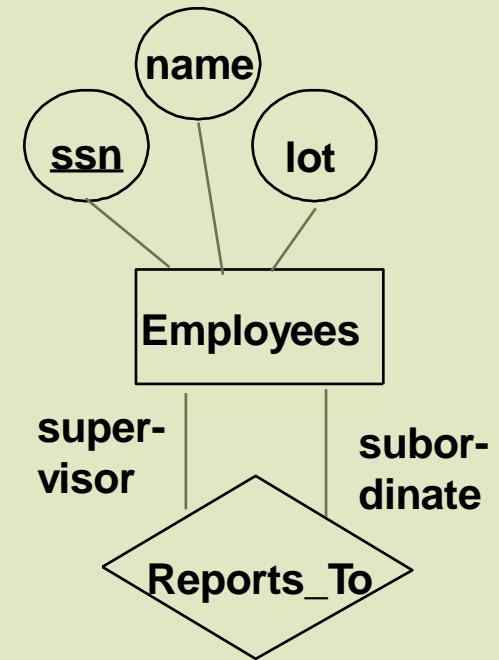
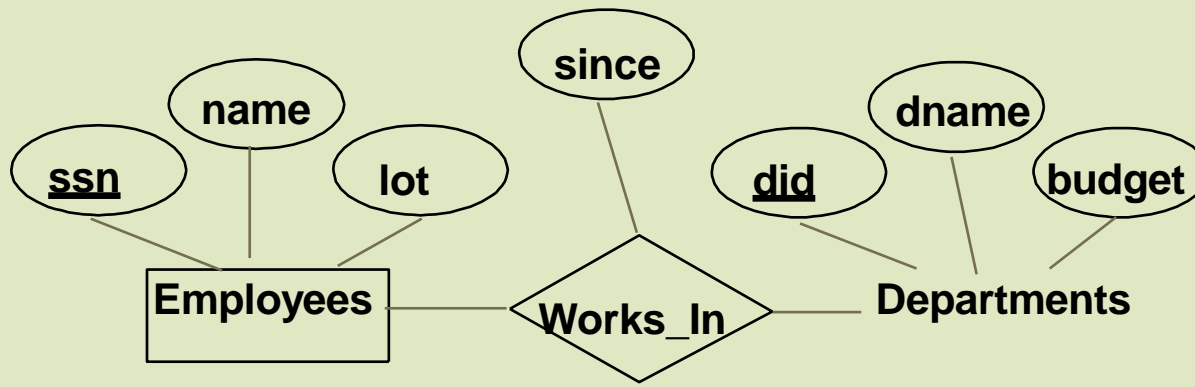
<sup>\*</sup>assuming (unrealistically) that there are no two people with the same first name, last name and birthday

If {sin} is the only candidate key, how many superkeys do we have?



# Relationships

## ER Model Basics (Contd.)



- ◉ **Relationship**: Association among two or more entities.  
E.g., Kris works in Pharmacy department.
- ◉ **Relationship Set**: Collection of similar relationships.
- ◉ An n-ary relationship set  $R$  relates  $n$  entity sets  $E_1 \dots E_n$ ; each relationship in  $R$  involves entities  $e_1, \dots, e_n$ .
- ◉ Same entity set could participate in different relationship sets, or in different “roles” in same set.
- ◉ It may have descriptive attributes



# Cartesian or Cross-Products

- A **tuple**  $\langle a_1, a_2, \dots, a_n \rangle$  is just a list with  $n$  elements in order.
- A binary tuple  $\langle a, b \rangle$  is called an **ordered pair**.
- Given two sets  $A, B$ , we can form a new set  $A \times B$  containing all ordered pairs  $\langle a, b \rangle$  such that  $a$  is a member of  $A$ ,  $b$  is a member of  $B$ .
- In set notation:  $A \times B = \{ \langle a, b \rangle \mid a \text{ in } A, b \text{ in } B \}$ .
- Example:  $\{1, 2, 3\} \times \{x, y\} = \{ \langle 1, x \rangle, \langle 1, y \rangle, \langle 2, x \rangle, \langle 2, y \rangle, \langle 3, x \rangle, \langle 3, y \rangle \}$





# Exercise

- Let  $A = \{1,2,3\}$ ,  $B = \{x,y\}$ .
- Compute  $B \times A$ .
- Compute  $A \times A$ .

# N-fold cross products

- Given  $n$  sets  $A_1, A_2, \dots, A_n$ , the cross product  $A_1 \times A_2 \times A_3 \dots \times A_n$  is a new set defined by

$$A_1 \times A_2 \times A_3 \dots \times A_n = \{ \langle a_1, a_2, a_3, \dots, a_n \rangle : a_1 \text{ in } A_1, a_2 \text{ in } A_2, \dots, a_n \text{ in } A_n \}.$$

- Example:  $\{1,2,3\} \times \{x,y\} \times \{1,2,3\}$  has as members  $\langle 1,x,1 \rangle$  and  $\langle 1,y,3 \rangle$ .



# Exercise

- Let  $A = \{1,2,3\}$ ,  $B = \{x,y\}$ .
- Compute  $B \times A \times B$ .
- Compute  $B \times B \times B$ .
- If a set  $C$  has  $n$  members, how many are there in  $C \times C$ ? How many in  $C \times C \times C$ ? In general, how many in  $C \times C \times \dots \times C$  where we take  $k$  cross-products?

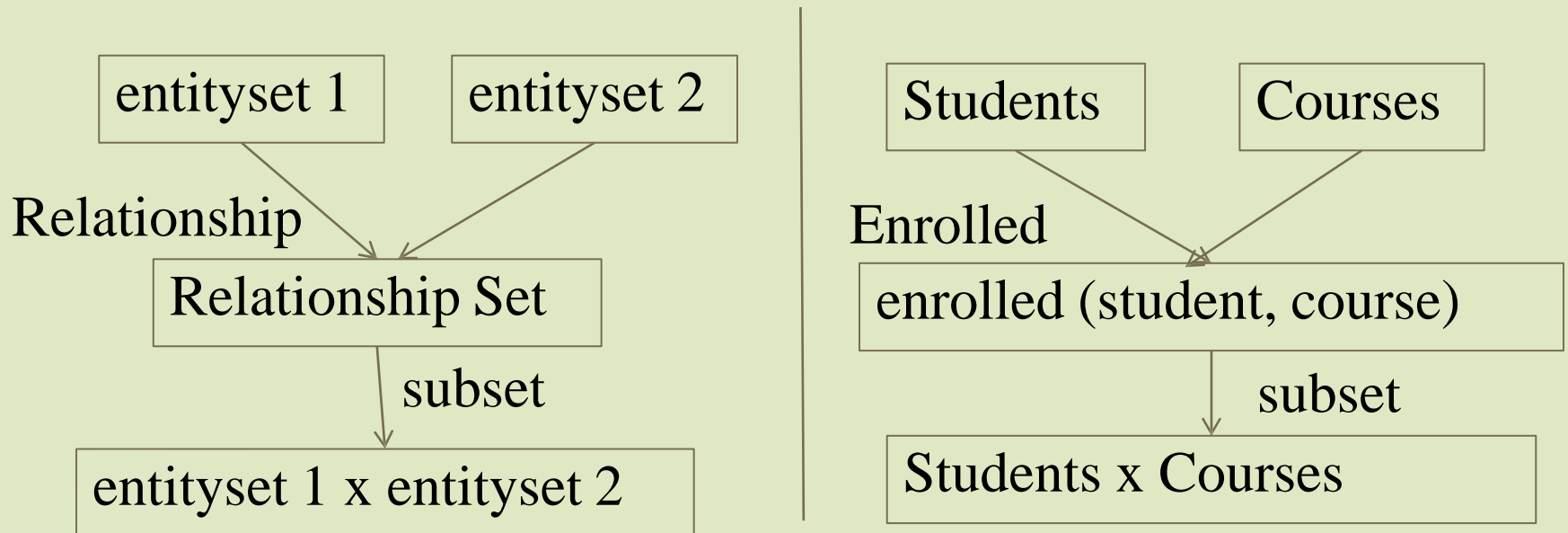


# The Cross Product

- ▶ Let  $E1, E2, E3$  be three entity sets.
- ▶ A **relationship** among  $E1, E2, E3$  is a tuple in  $E1 \times E2 \times E3$ .
- ▶ A **relationship set** is a set of relationships. So if  $R$  is a relationship set among  $E1, E2, E3$ , then  $R$  is a *subset* of  $E1 \times E2 \times E3$ .
- ▶ *Relationship set = subset of cross product.*

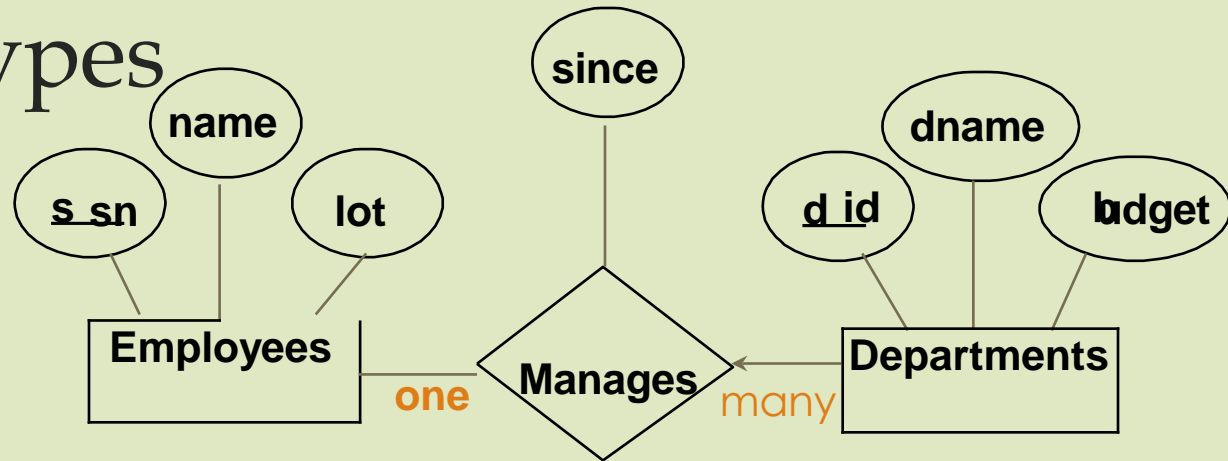
# Relationships

- A relationship, or simply relation, returns a relationship set given entity sets.
- Informally, the relationship specifies which entities are related.



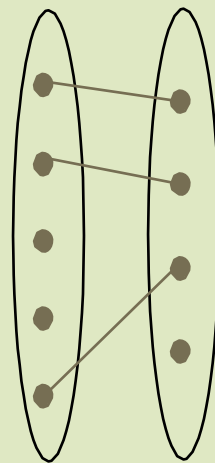
# Relation Types

- Consider Works\_In:  
An employee can work in many departments; a dept can have many employees.

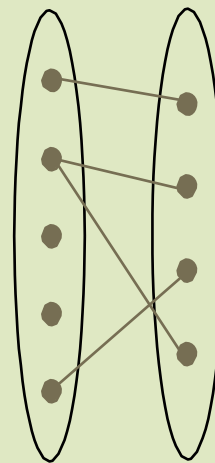


Why is the arrow on the many side?

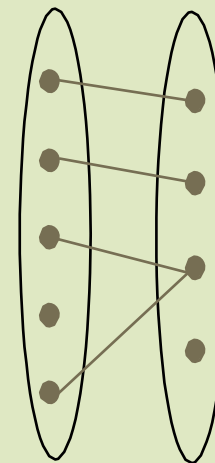
- In contrast, each dept has at most one manager, according to the key constraint on **Manages**.
- Key constraint is represented by an arrow →



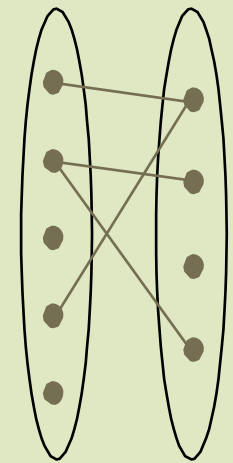
1-to-1



1-to Many



Many-to-1



Many-to-Many



# Relationship Set Primary Keys

- The primary key of a relationship depends on the key constraints in the relationship
- *Many-to-many* – all the non-descriptive attributes of the relationship set
- *One-to-many* – the primary key for the *many* entity
- *One-to-one* – the primary key of *either* entity



## Exercise 2.2

A university database contains information about professors (identified by SSN) and courses (identified by courseid). Professors teach courses; each of the following situations concerns the Teaches relationship set. For each diagram, draw an ER diagram that describes it (assuming no further constraints hold).

1. Professors can teach the same course in several semesters, and each offering must be recorded.
2. Professors can teach the same course in several semesters, and only the most recent such offering needs to be recorded.



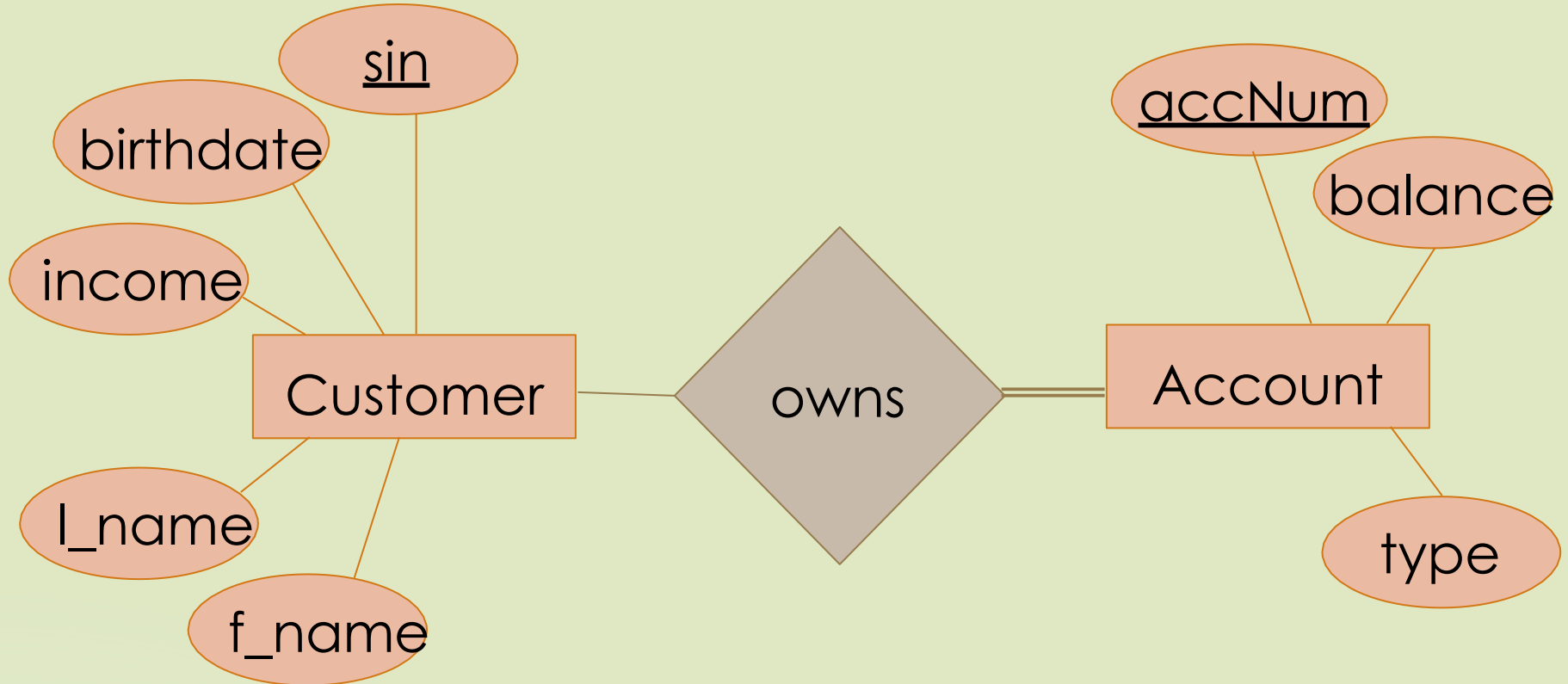


# Participation Constraints

- Indicate that each entity in an entity set *must* be involved in at least one relationship
- Participation is said to be either *total* (there is a constraint) or *partial* (no constraint)
- If there is no participation constraint all the entities *may* still be involved in the relationship
- Total participation is indicated by a double line from the relationship to the entity
- Or a thick line

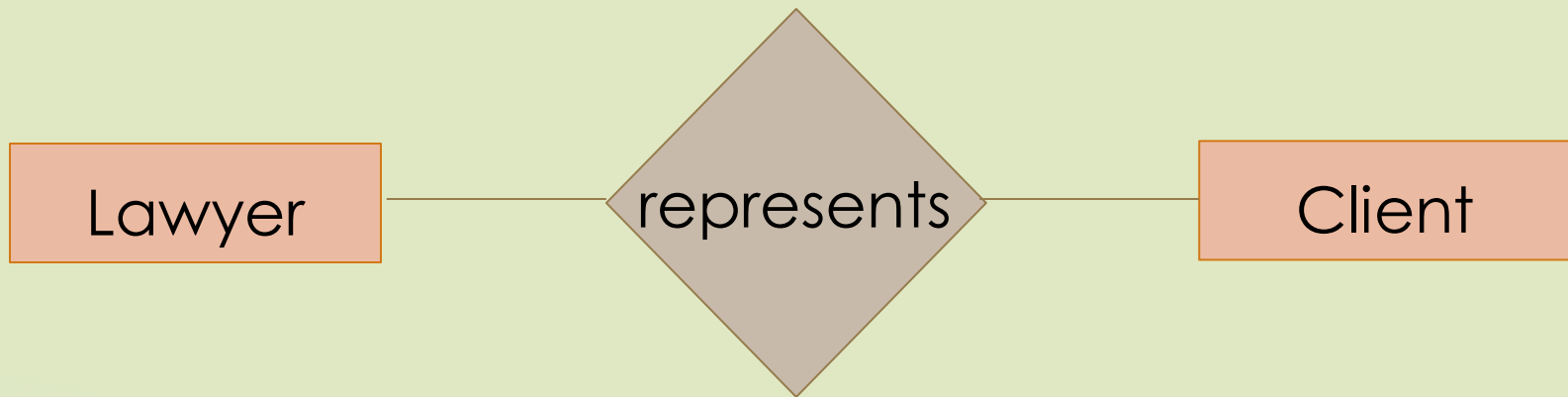
# Participation Example

- Each account must be owned by at least one customer



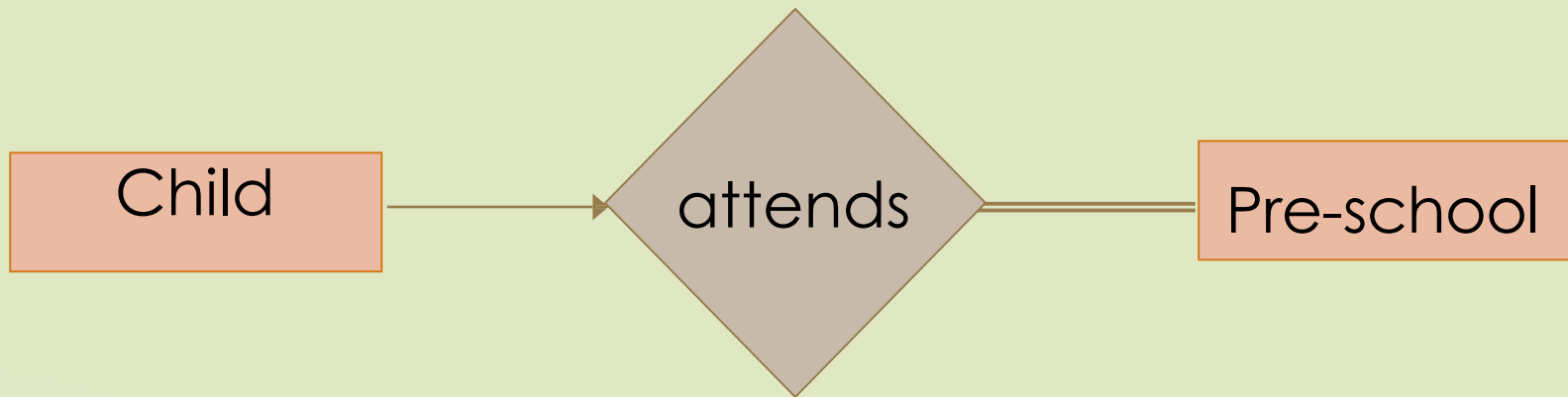
# Lawyers

- ▶ A law firm's lawyers are assigned to represent clients' interests



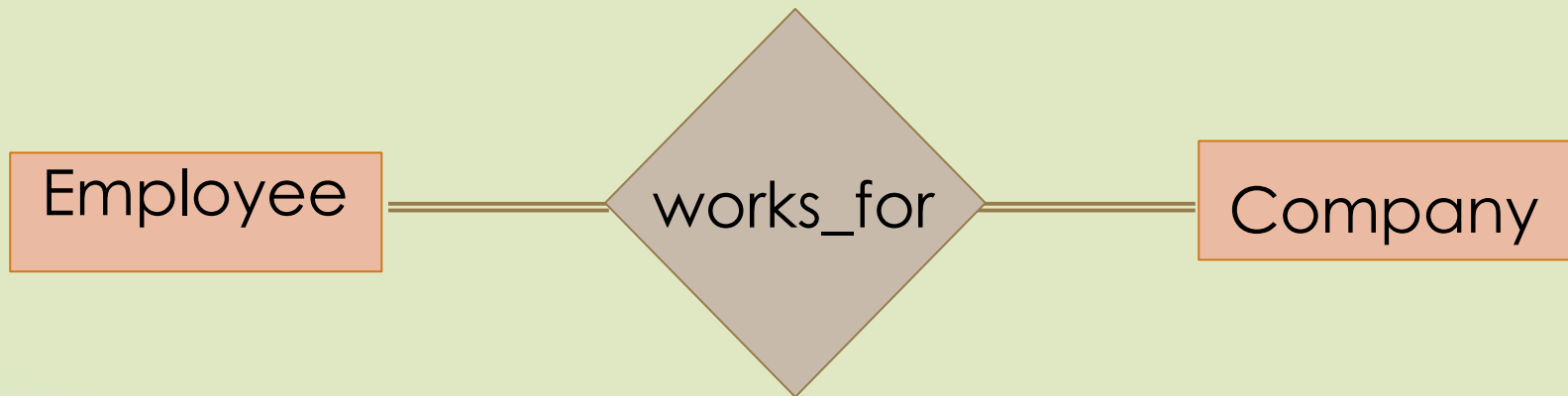
# Kids

- Children attend pre-school, assume that
  - Children can only attend one pre-school
  - A pre-school must have children



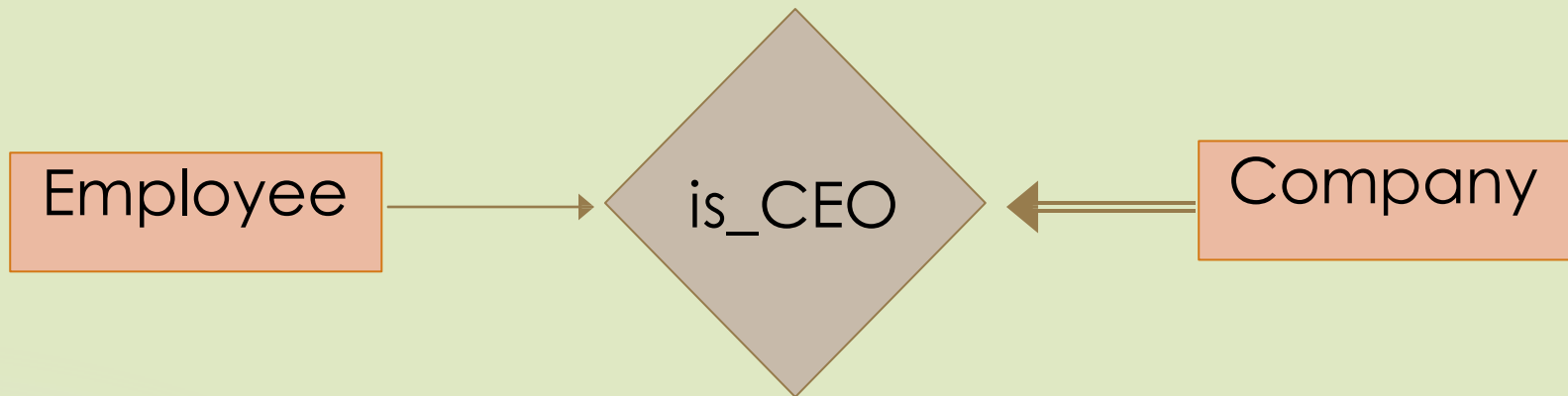
# Wage Slaves

- Employees work for companies
  - An employee must be employed by someone (or they wouldn't be an employee)
  - People often have more than one job



# The Man

- ▶ Chief Executive Officer (CEOs) of a company
- ▶ Assume that being a CEO is a full time position and
- ▶ That a company can only have one CEO
- ▶ And that one woman or man must be in charge
- ▶ So no cooperatives or workers' collectives





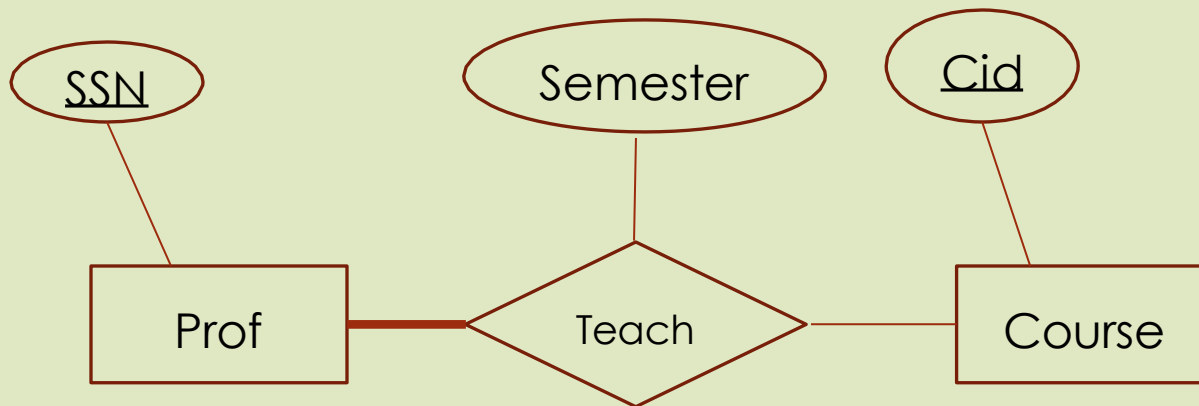
## Exercise 2.2 ctd.

Same scenario as before, where we need only the current semester. Draw E-R diagrams for the following constraints.

3. Every professor must teach some course.
4. Every professor teaches exactly one course.
5. Every professor teaches exactly one course, and every course must be taught by some professor.

## Exercise 2.2

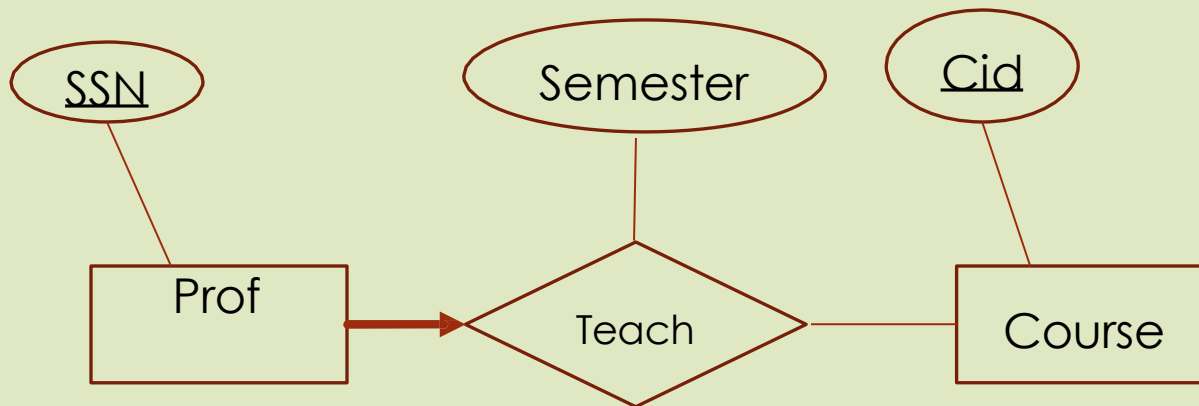
Every professor must teach some course.





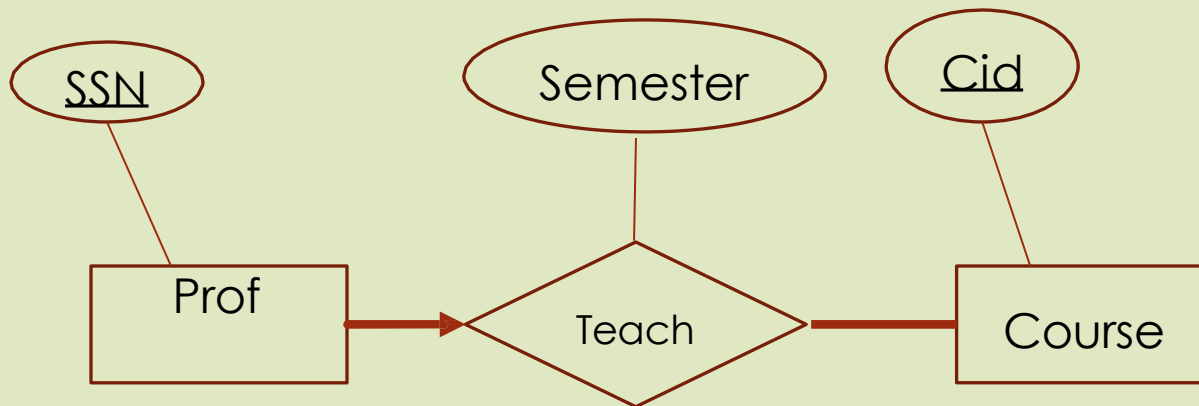
## Exercise 2.2

Every professor teaches exactly one course.



## Exercise 2.2

Every professor teaches exactly one course, and every course must be taught by some professor.

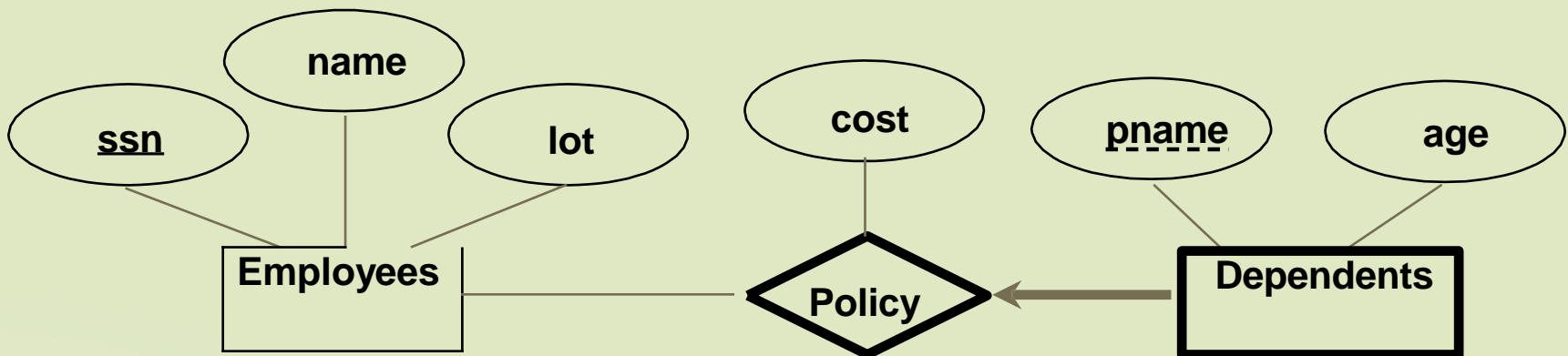




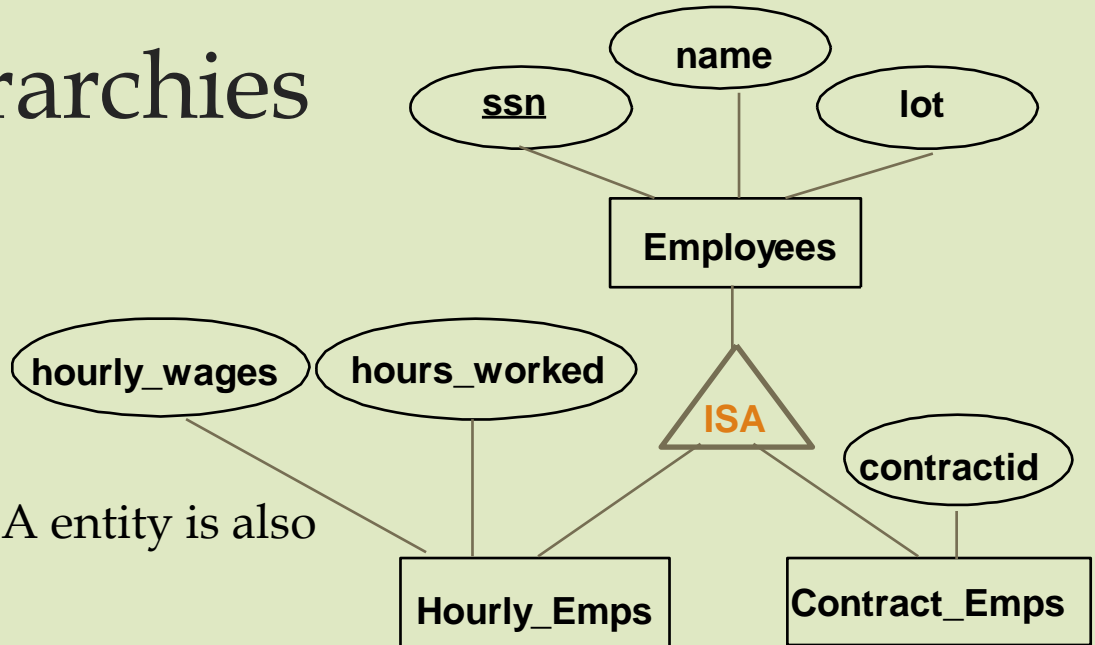
# Extended ER Model

# 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.



# ISA ('is a') Hierarchies



If we declare A **ISA** B, every A entity is also considered to be a B entity.

- *Overlap constraints*: Can Joe be an Hourly\_Emps as well as a Contract\_Emps entity? (*Allowed/disallowed*)
- *Covering constraints*: Does every Employees entity also have to be an Hourly\_Emps or a Contract\_Emps entity? (*Yes/no*)
- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify entities that participate in a relationship.

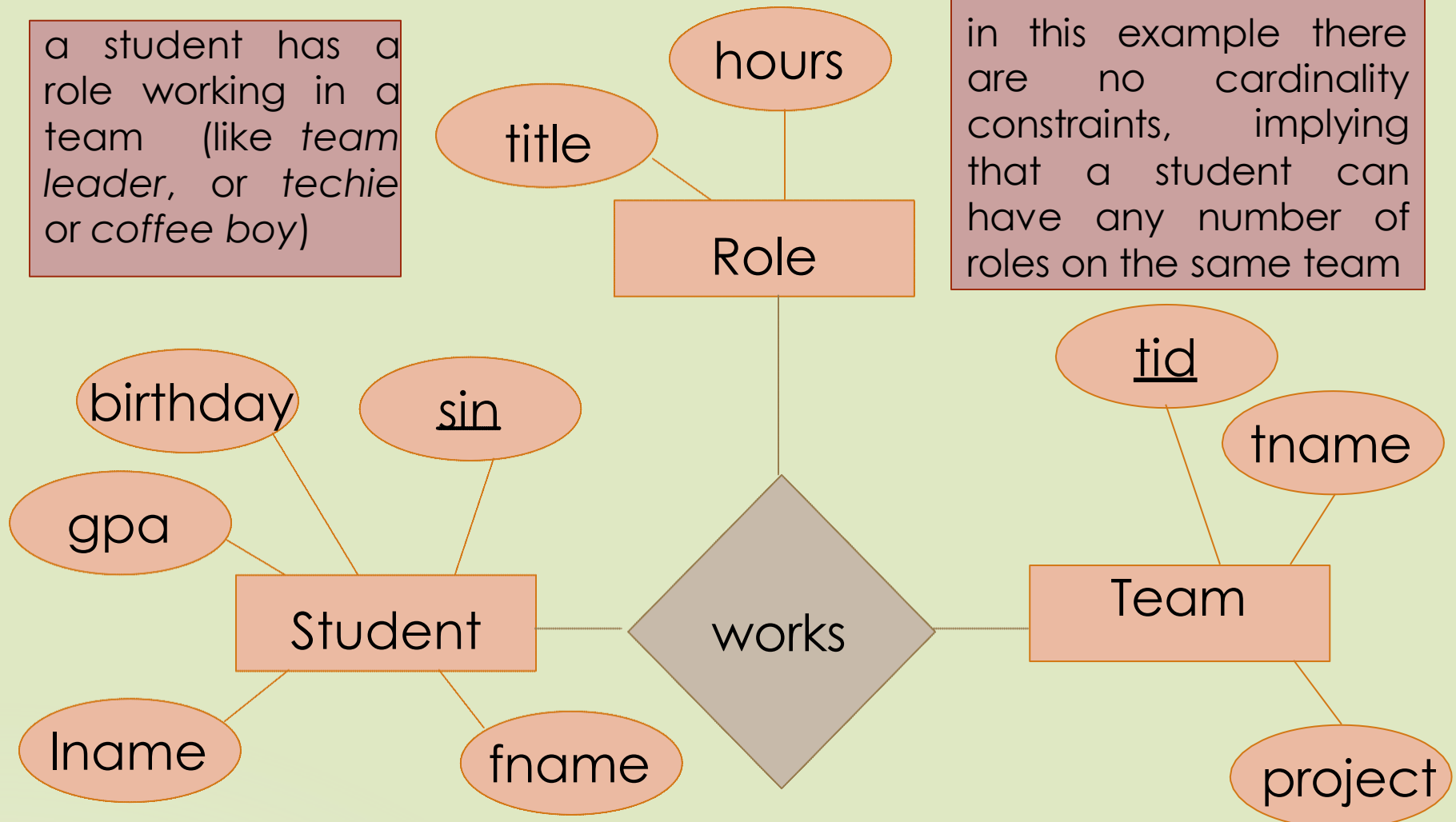


# Non-Binary Relationships

- A relationship does not have to be binary but can include any number of entity sets
- Ternary relationships are not uncommon
- Specifying the mapping cardinalities of non-binary relationships can be complicated
- A key constraint indicates that an entity can participate in only one relationship
- Just like with binary relationships

# Ternary Relationships

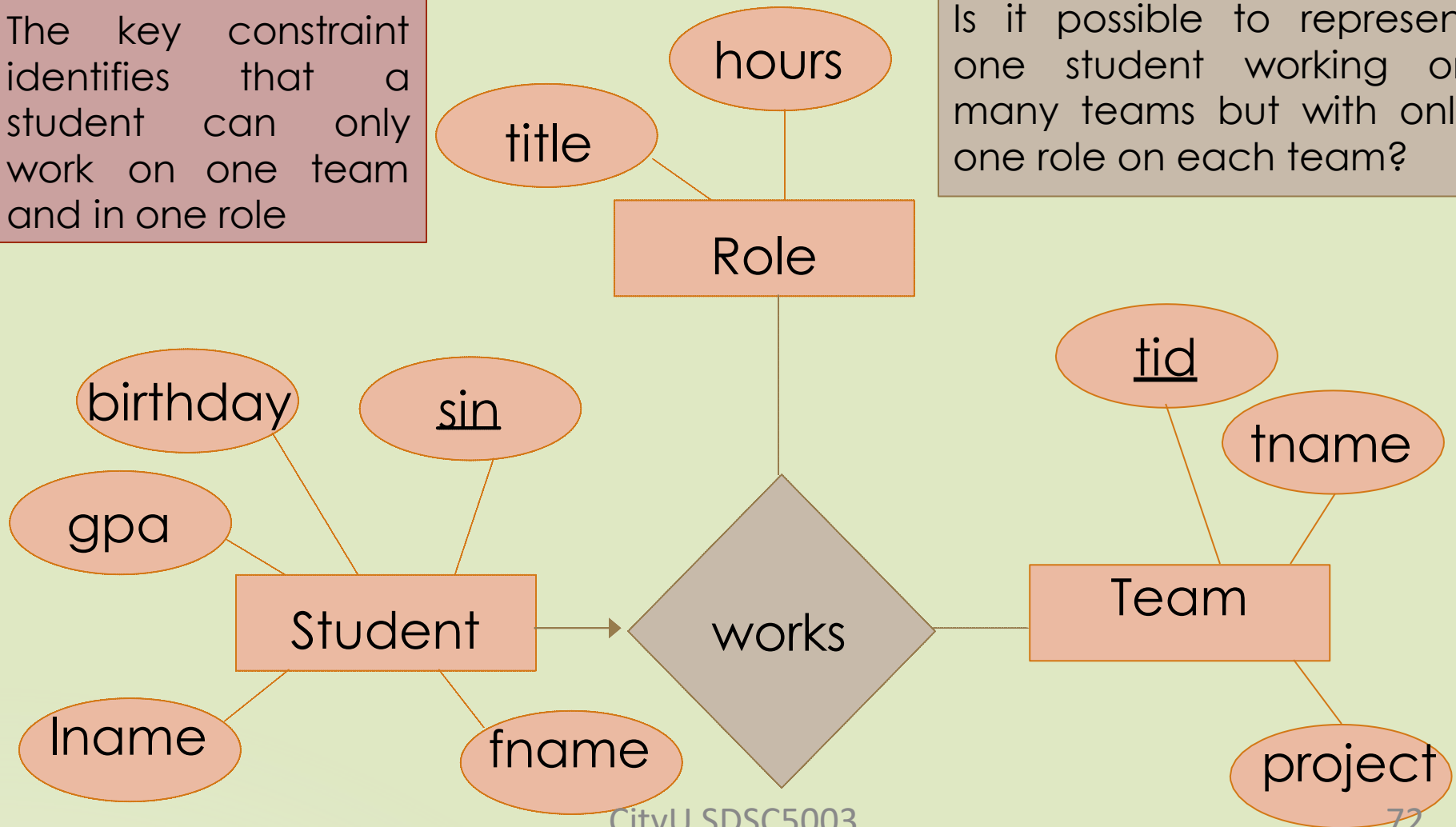
a student has a role working in a team (like *team leader*, or *techie* or *coffee boy*)



# Ternary Relationships

The key constraint identifies that a student can only work on one team and in one role

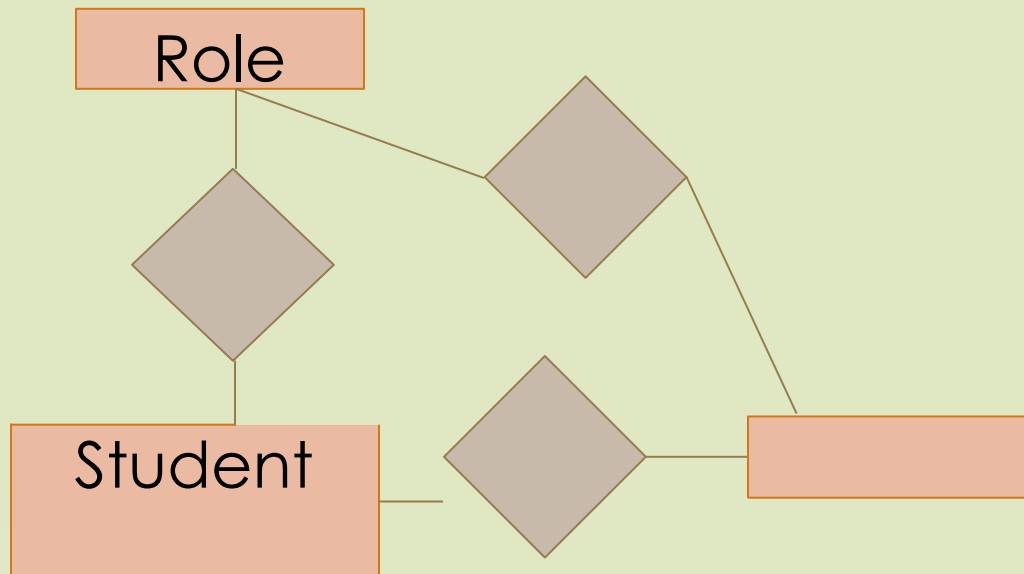
Is it possible to represent one student working on many teams but with only one role on each team?





# Replace Ternary Relationship

- We can replace the ternary relationships with binary relationships
- But can have only one role on each team?
- And the relationship pairs are not forced to relate to each other
- Bob may have a manager role and Bob may work on the Harmony OS project with a team
- But the Harmony OS project may not have a manager



$(\text{Student}, \text{Role}, \text{Team}) \rightarrow$   
 $(\text{Student}, \text{Role}) + (\text{Student}, \text{Team})$   
 $+ (\text{Role}, \text{Team})$

Do  $(\text{Bob}, \text{Manager}) +$   
 $(\text{Bob}, \text{Harmony}) +$   
 $(\text{Manager}, \text{Harmony})$  imply  
 $(\text{Bob}, \text{Manager}, \text{Harmony})$ ?

Team

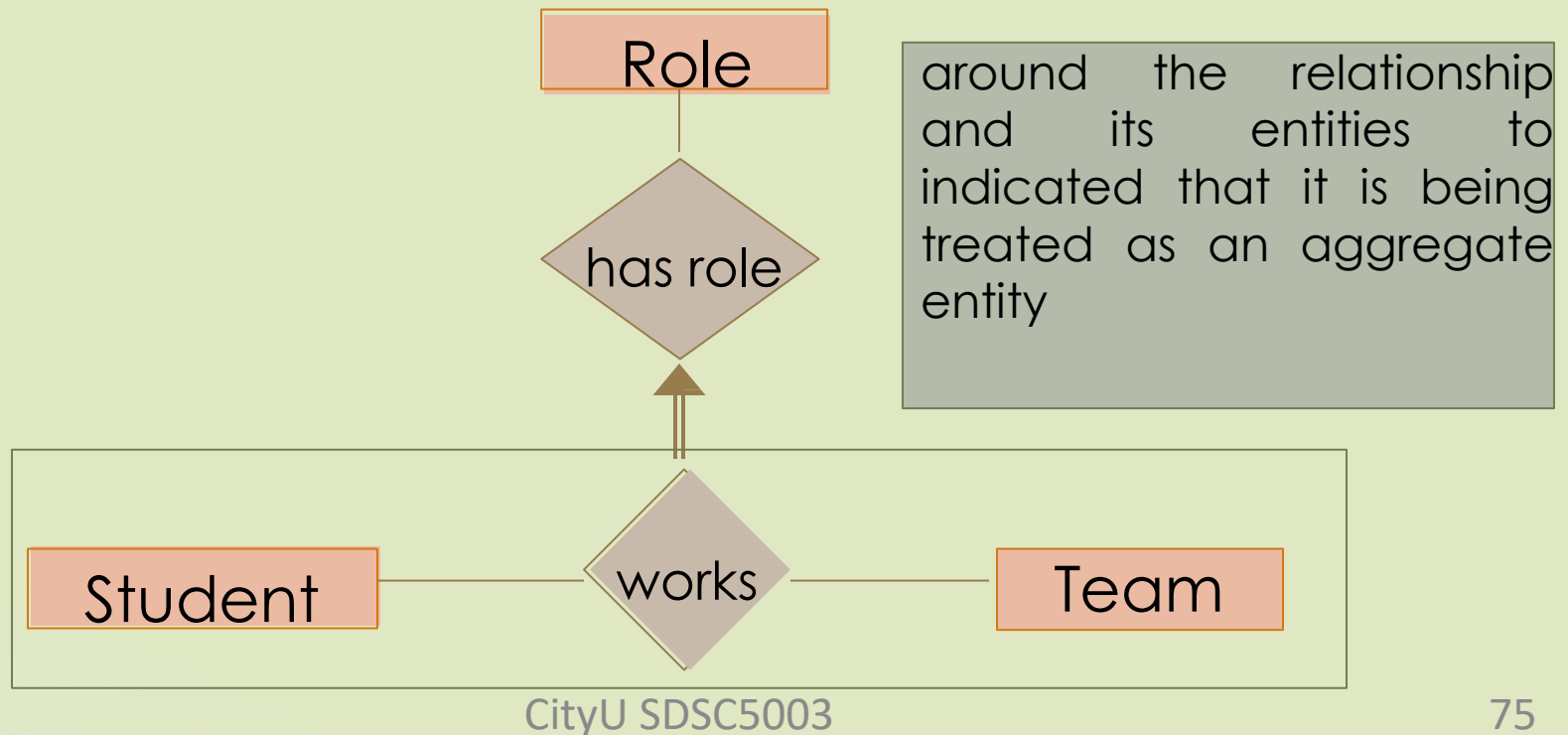


# Aggregation

- Indicates that a relationship set participates in another relationship set
- An abstraction that treats relationship sets as higher-level entities
- For the purpose of participation in other relationships
- When should aggregation be used?
- When there is a relationship between an entity set and another relationship
- Aggregation is often used with non-binary relationships

# Aggregation Example

- ▶ Treat the works relationship as an aggregate entity
- ▶ Each works relationship can be associated with just one role
- ▶ Different works relationships with the same project or employee can be associated with other roles





# ER Design Principles

- Faithfulness
  - The design must be faithful to the specification (and the enterprise that is being represented)
  - All the relevant aspects of the enterprise should be represented in the model
- Avoid redundancy
  - Redundant representation makes the ER diagram harder to understand
  - Redundancy wastes storage in the DB and
  - May lead to inconsistencies



# More ER Design Principles

- Simplicity
  - The simpler it is, the easier it is to understand
  - Avoid introducing unnecessary entities or relationships
  - Where possible use attributes rather than entity sets or relationships
- Specify as many constraints as possible
  - Ensure that all key constraints and participation constraints are specified
  - Some constraints cannot be shown in ER diagrams
    - Example: if a team has more than 10 members, it should have one manager.

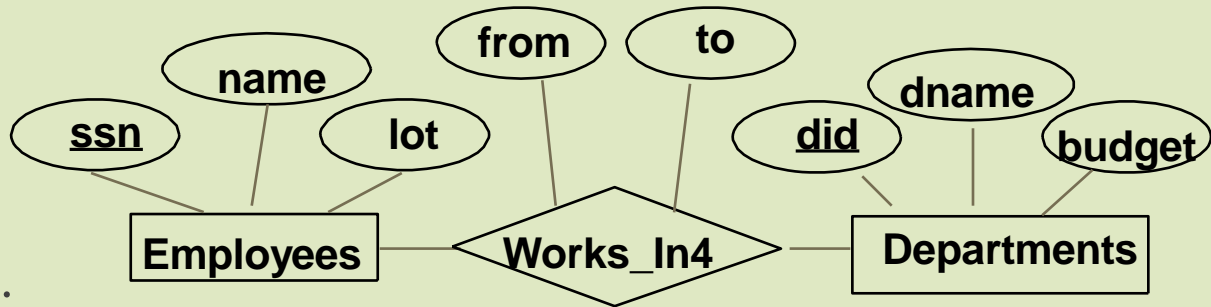


# Design Choices

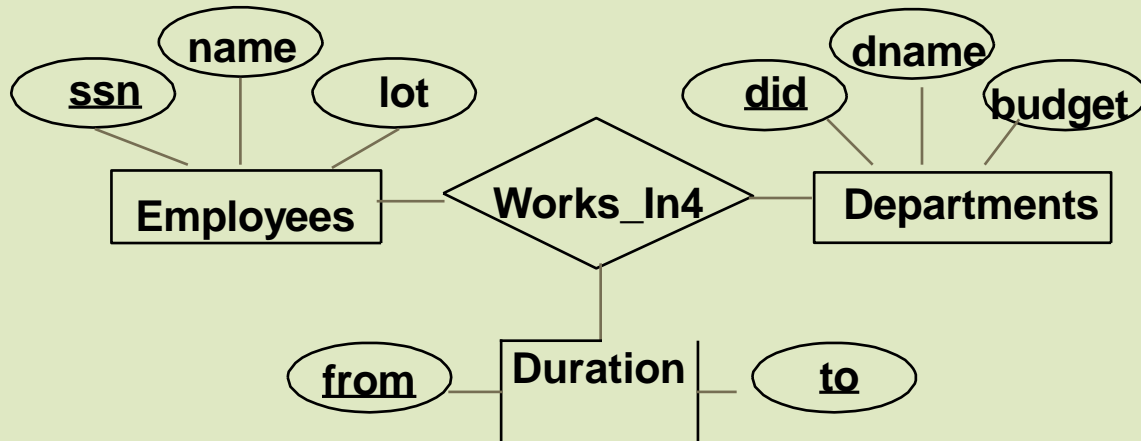
- Entity set or attribute?
- Entity set or relationship set?
- What sort of relationship is it?
  - Multiple binary relationships
  - Ternary relationship
  - Aggregation

# Entity vs. Attribute

- Works\_In4 does not allow an employee to work in a department for two or more periods.

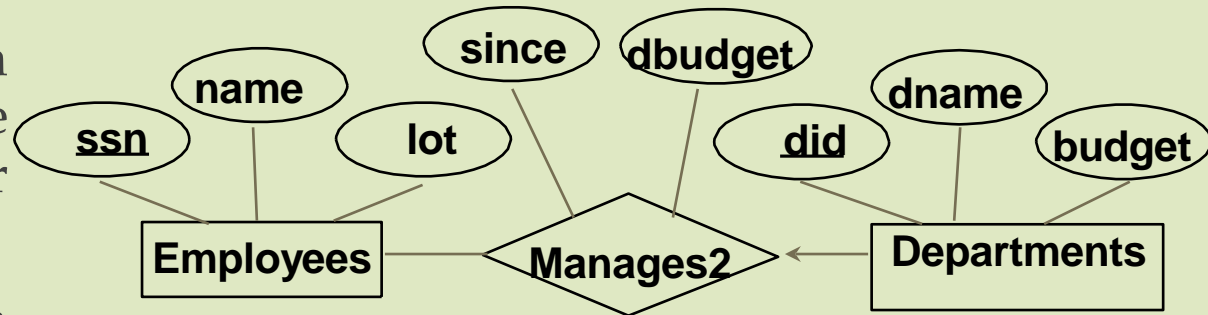


- Similar to the problem of addresses for an employee: We want to record *several values of the descriptive attributes for each instance of this relationship*. Accomplished by introducing new entity set, **Duration**.



# Entity vs. Relationship

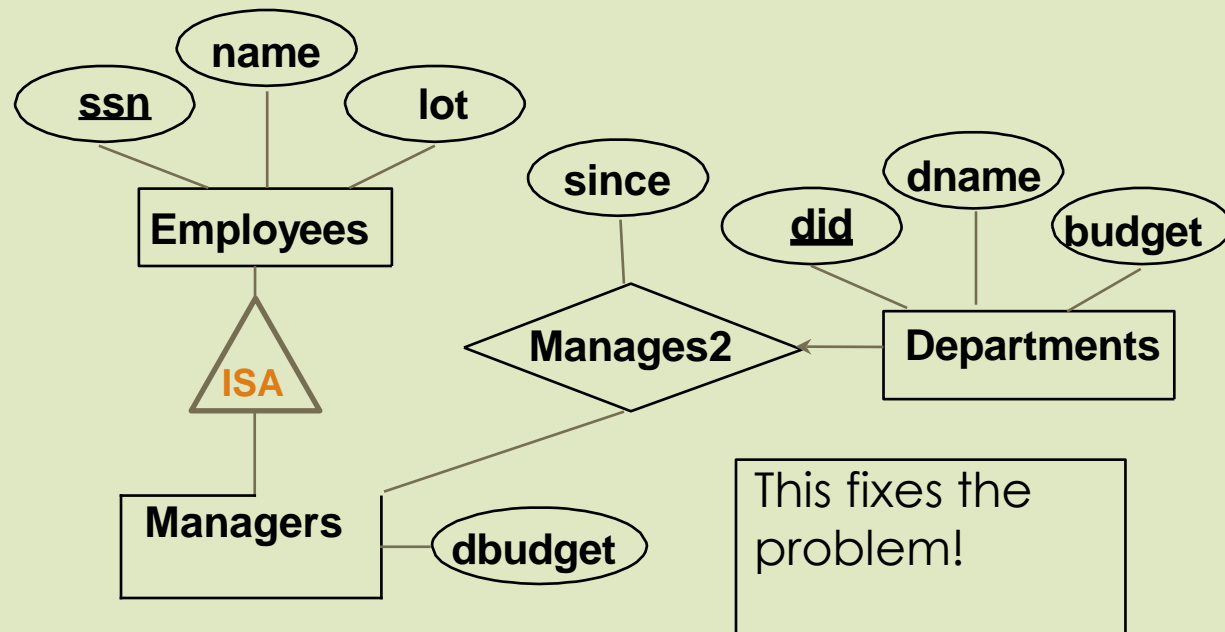
- First ER diagram OK if a manager gets a separate discretionary budget for each dept.



- What if a manager gets a discretionary budget that covers *all* managed depts?

- Redundancy:** *dbudget* stored for each dept managed by manager.

- Misleading:** Suggests *dbudget* associated with department-mgr combination.



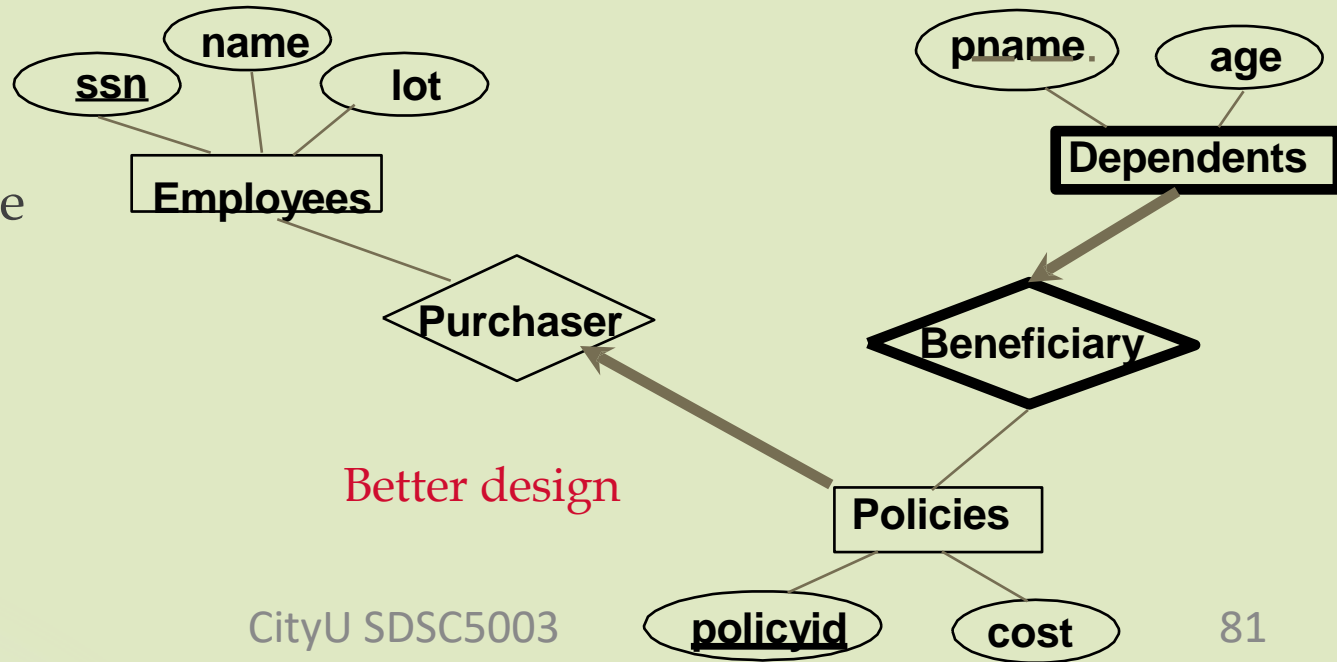
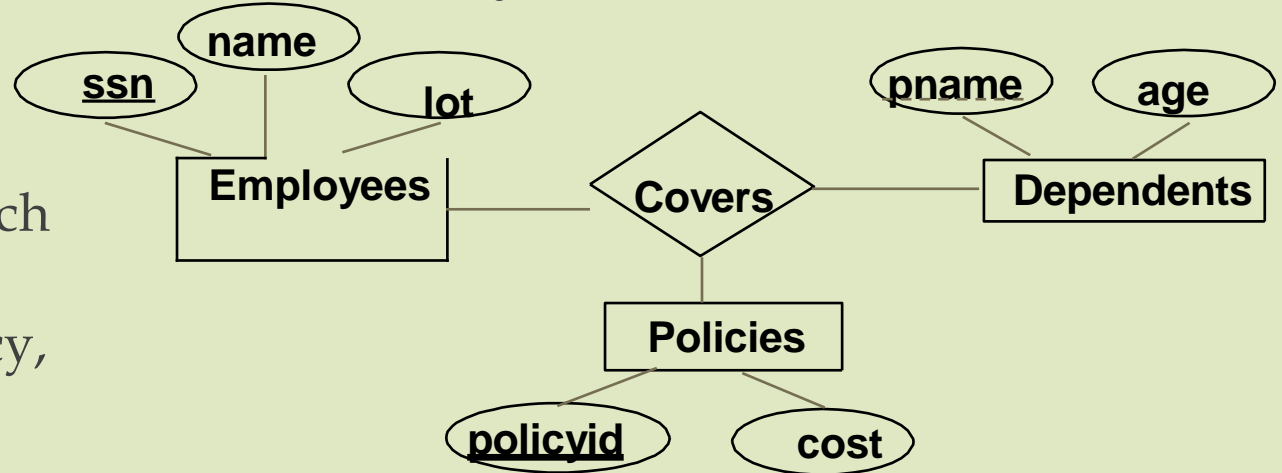


# Binary vs. Ternary Relationships

- If each policy is owned by just 1 employee, and each

the covering policy, first diagram is inaccurate.

- What are the additional constraints in the 2nd diagram?





# Summary of Conceptual Design

- *Conceptual design follows requirements analysis,*
- Yields a high-level description of data to be stored
- ER model popular for conceptual design
- Constructs are expressive, close to the way people think about their applications.
- Basic constructs: *entities, relationships, and attributes* (of entities and relationships).
- Some additional constructs: *weak entities, ISA hierarchies.*
- Note: There are many variations on ER model.



# Summary of ER (Contd.)

- Several kinds of integrity constraints can be expressed in the ER model: *key constraints, participation constraints, and overlap/covering constraints* for ISA hierarchies.
- Some constraints (notably, *functional dependencies*) cannot be expressed in the ER model. (e.g.,  $z = x + y$ )
- Constraints play an important role in determining the best database design for an enterprise.



# Summary of ER (Contd.)

➤ ER design is *subjective*. There are often many ways to model a given scenario! Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:

- Entity vs. attribute.
- entity vs. relationship
- binary or Ternary relationship
- ISA hierarchies.
- Ensuring good database design: resulting relational schema should be analyzed and refined further. See Ch. 19