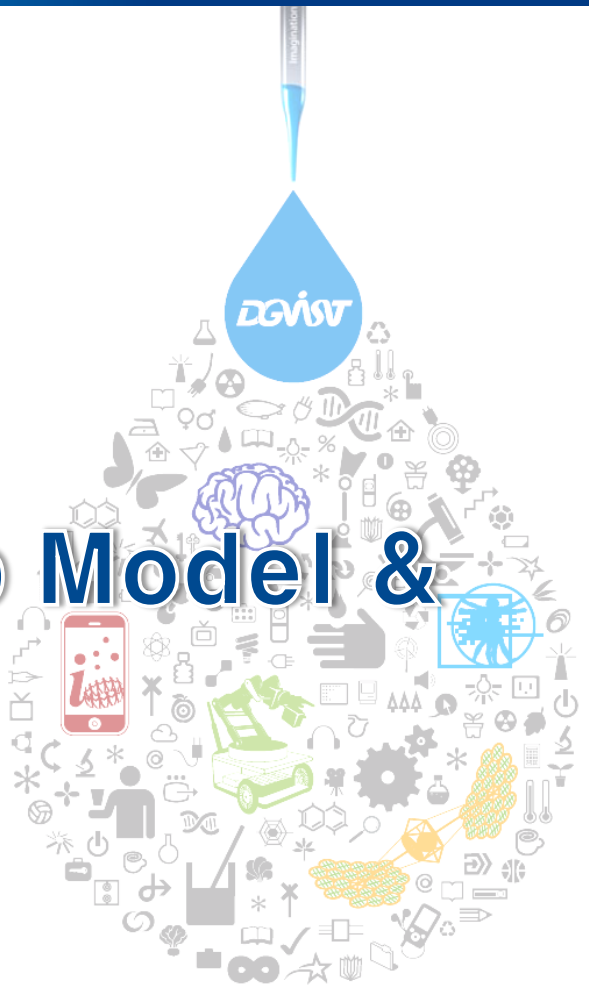
InfoLab

IC611: Database Systems Design

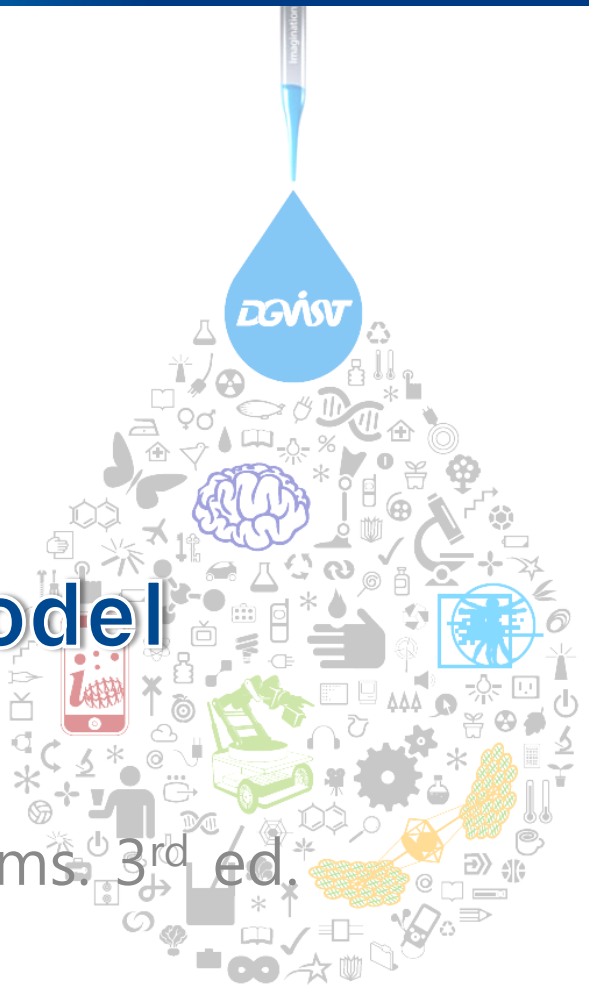
The Entity-Relationship Model & The Relational Model



Min-Soo Kim

DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology

InfoLab



IC611: Database Systems Design

The Entity-Relationship Model

from Database Management Systems, 3rd ed.

DGIST 대구경북과학기술원 Daegu Gyeongbuk Institute of Science & Technology

Database Design Process

1. Requirements Analysis
2. Conceptual Database Design
3. Logical Database Design
4. Schema Refinement
5. Physical Database Design
6. Application and Security Design

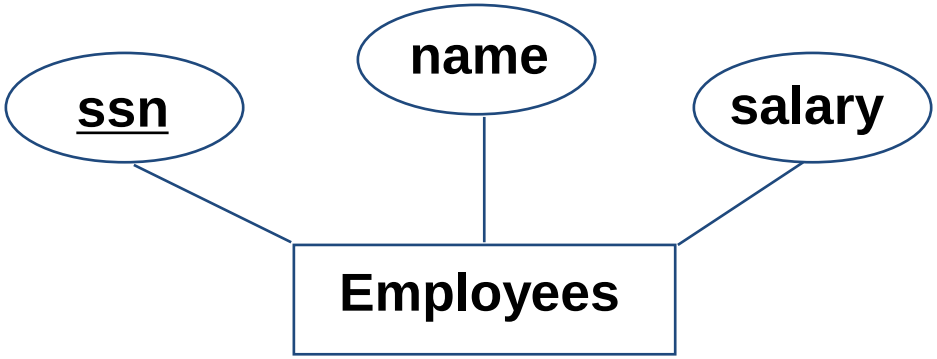
Overview of Database Design

- **Conceptual 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

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
  - Each entity set has a key
  - Each attribute has a domain

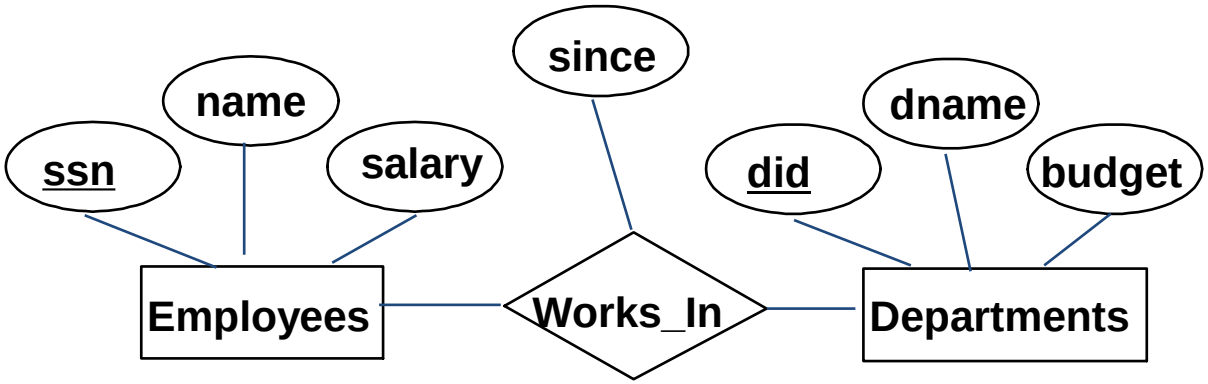


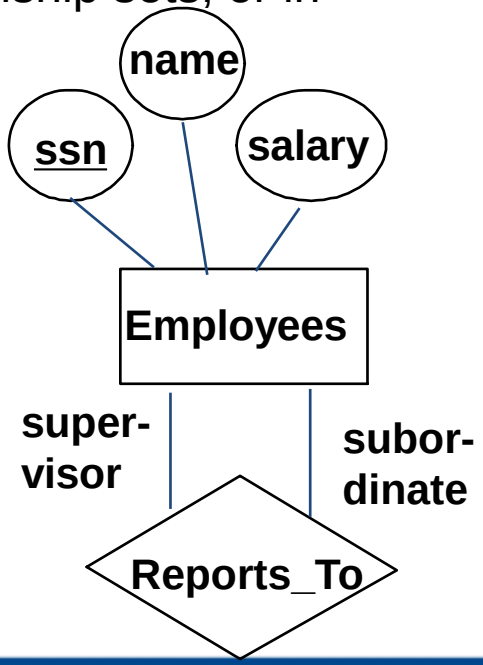
Time-series vs. Sequence data

Learning vs. Making Knowledge

ER Model Basics (Contd.)

- **Relationship**: **Association among two or more entities**
  - e.g., Bob works in Pharmacy department
- **Relationship Set**: **Collection of similar relationships**
  - An *n*-ary relationship set *R* relates *n* entity sets *E1*, ... , *En*
  - each relationship in *R* involves entities *e1* ∈ *E1*, ..., *en* ∈ *En*
    - Same entity set could participate in different relationship sets, or in different "roles" in the same set



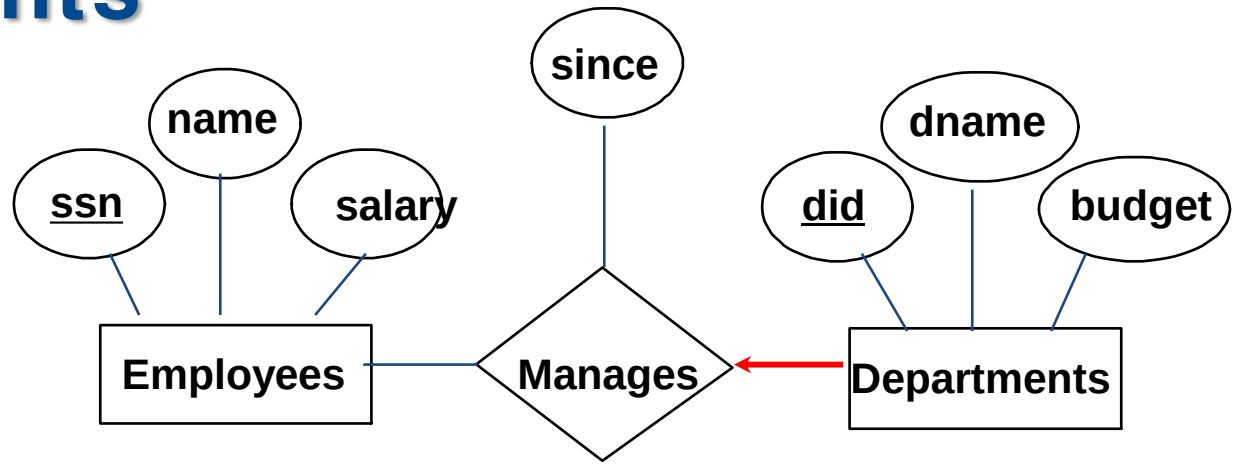


Examples

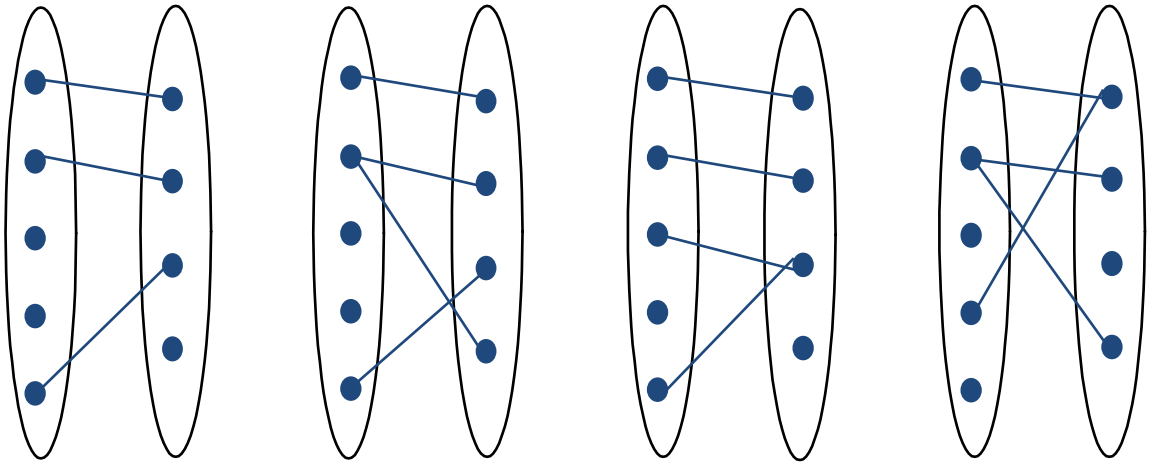
- People – Friendship
- Web page – Hyperlink
- Email address – Sending email
- Protein – Interaction
- Account – Transfer
- Consumer – Purchase – Goods

Key Constraints

- Consider Works_In: An employee can work in many departments; a dept can have many employees

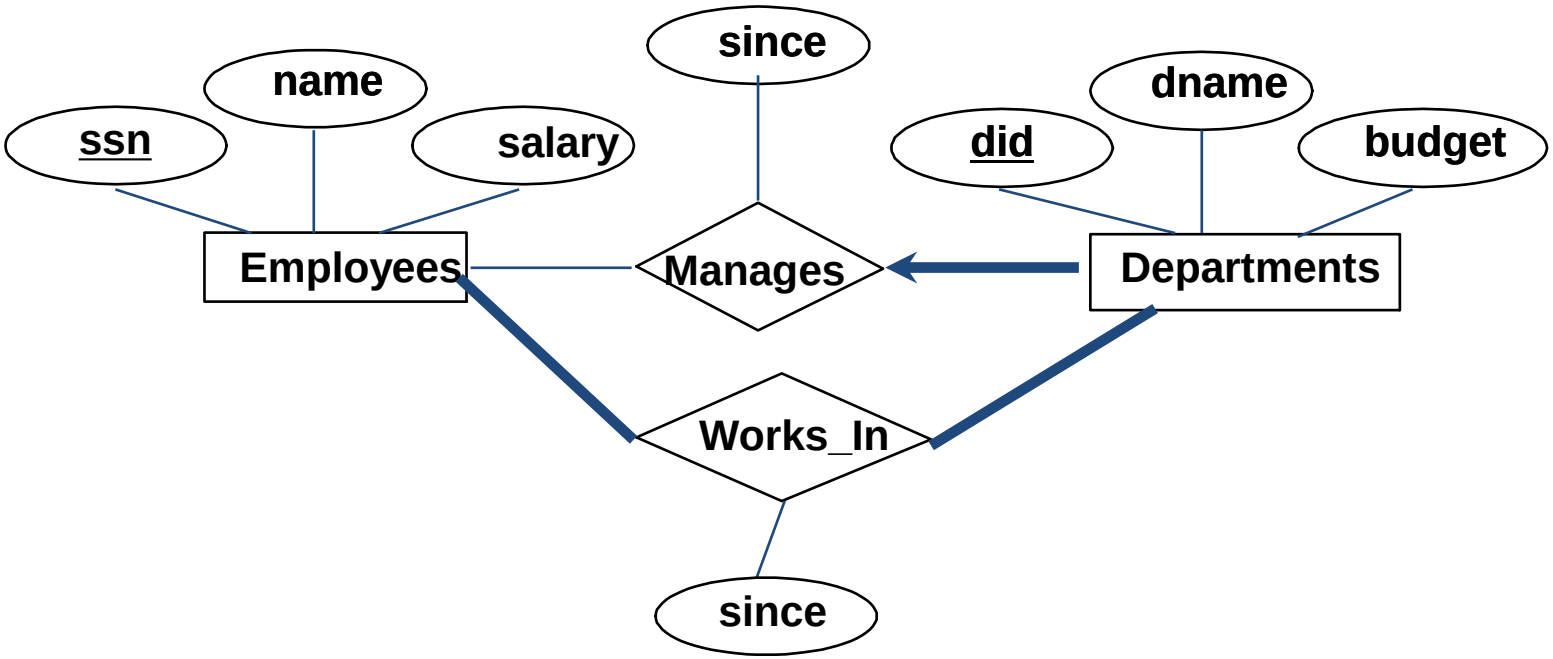


- In contrast, each dept. has at most one manager, according to the key constraint on Manages



Participation Constraints

- **Does every department have a manager?**
  - If so, this is a participation constraint: the participation of Departments in Manages is said to be total (vs. partial)
    - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)



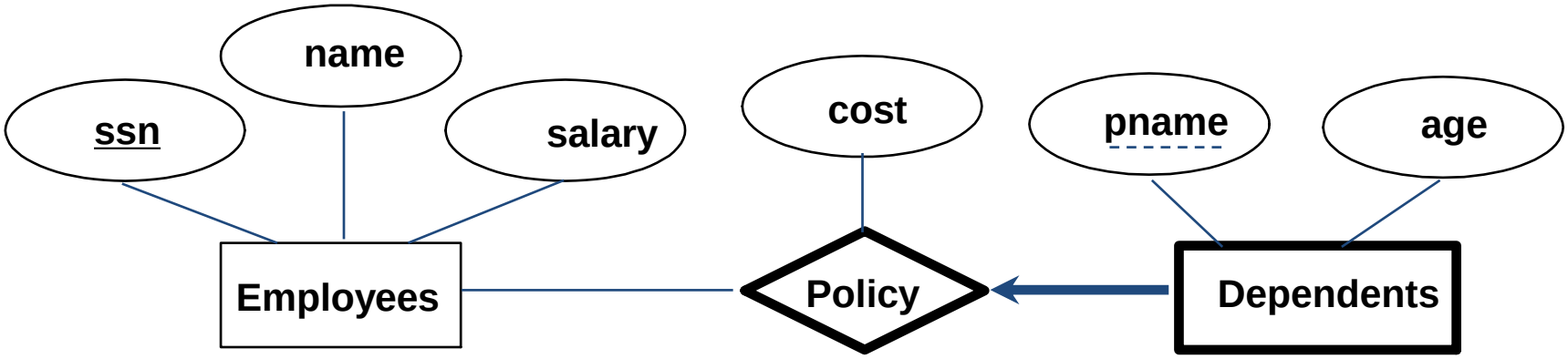
Student-Class Relationship

- **Relational model**

- **Graph model (bi-partitite)**

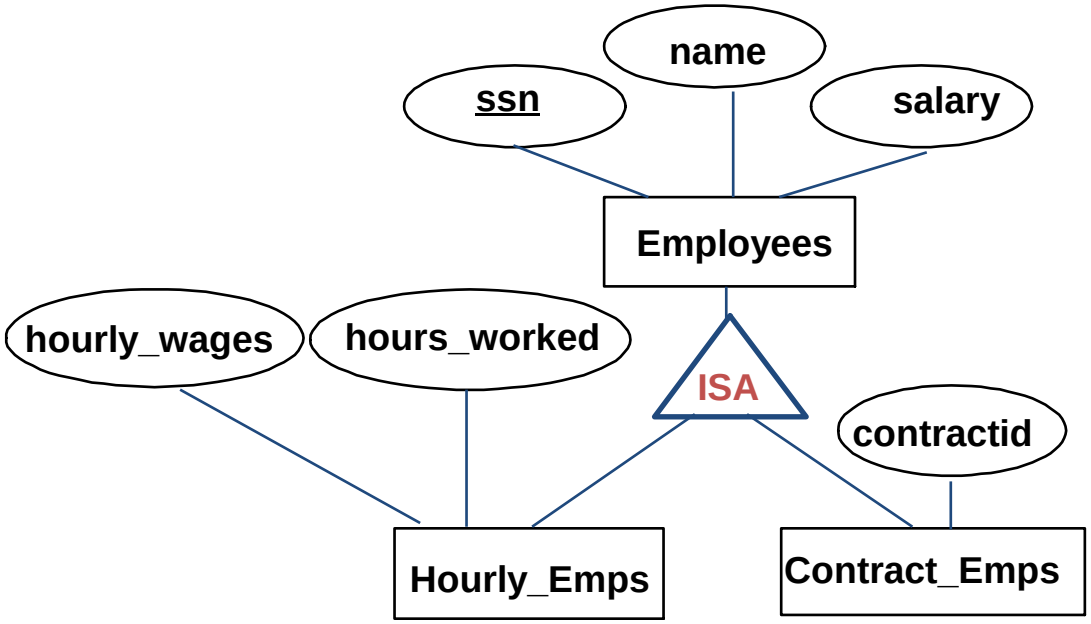
Weak Entities

- **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

- As in C++, or other PLs, attributes are inherited
- 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



Conceptual Design Using the ER Model

- **Design choices:**
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships: Binary or **ternary**?

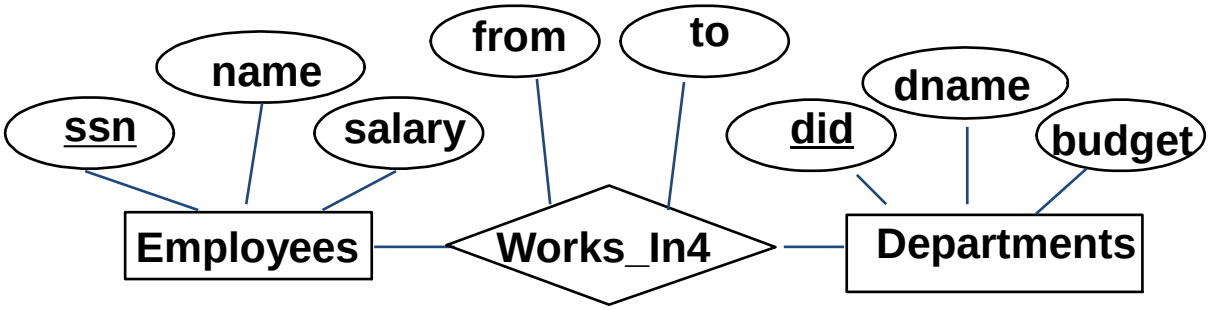
- **Constraints in the ER Model:**
  - A lot of data semantics can (and should) be captured
  - But some constraints cannot be captured in ER diagrams

Entity vs. Attribute

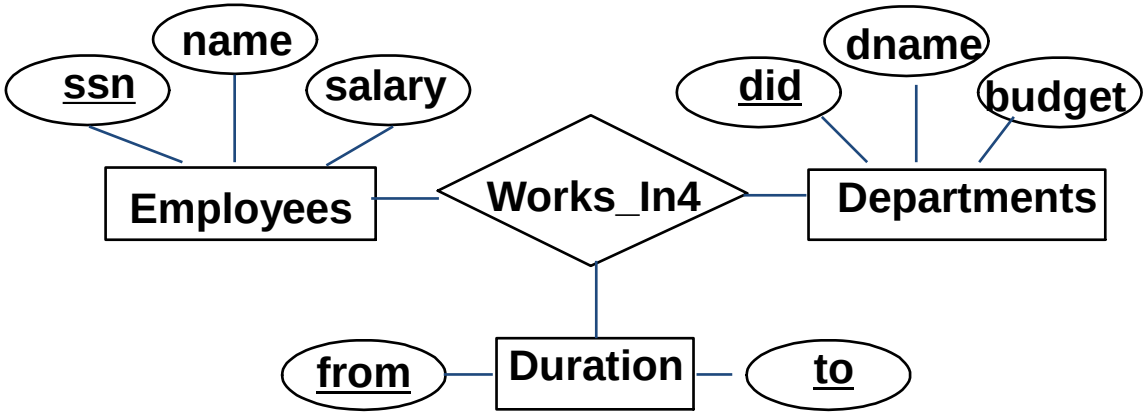
- **Should address be an attribute of Employees or an entity (connected to Employees by a relationship)?**
- **Depends upon the use we want to make of address information, and the semantics of the data:**
  - If we have several addresses per employee, address must be an entity (since attributes cannot be set-valued)
  - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, address must be modeled as an entity (since attribute values are atomic)

Entity vs. Attribute (Contd.)

- Works_In4 does not allow an employee to work in a department for two or more periods



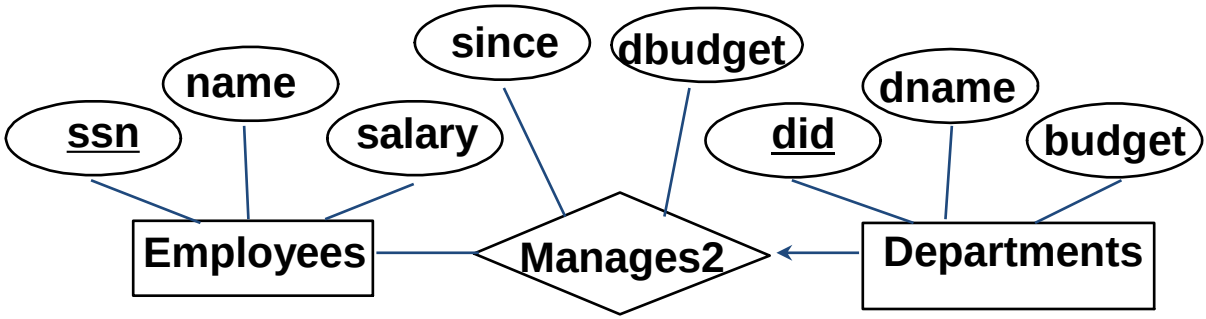
- We want to record several values of the descriptive attributes for each instance of this relationship
- Accomplished by introducing new entity set, Duration



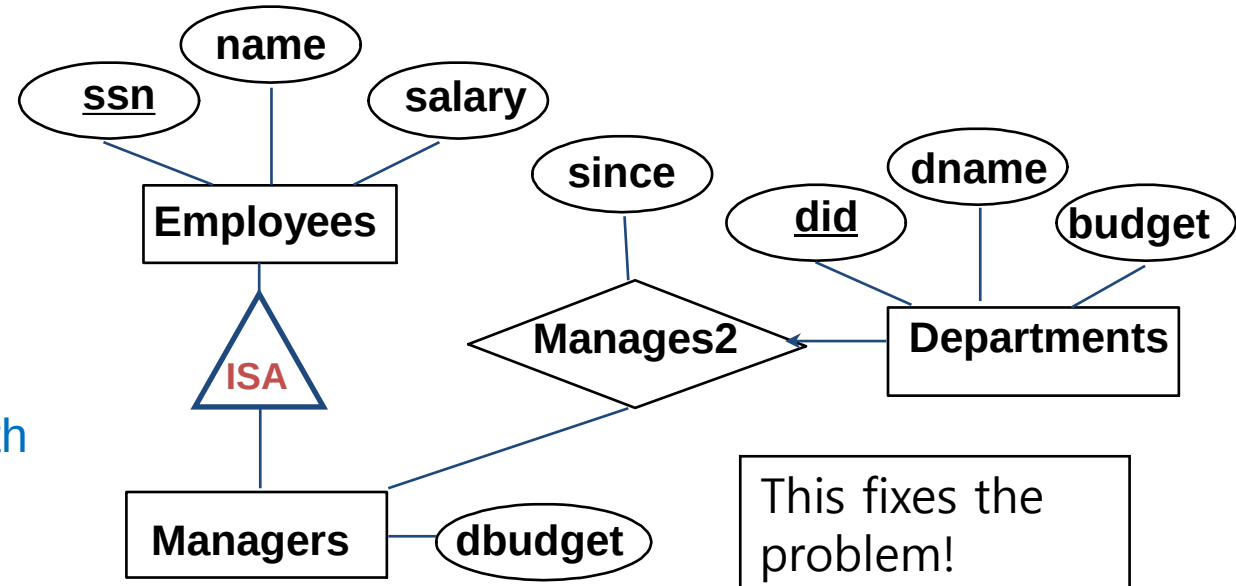
Design Choice: Generality vs. Efficiency

Entity vs. Relationship

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

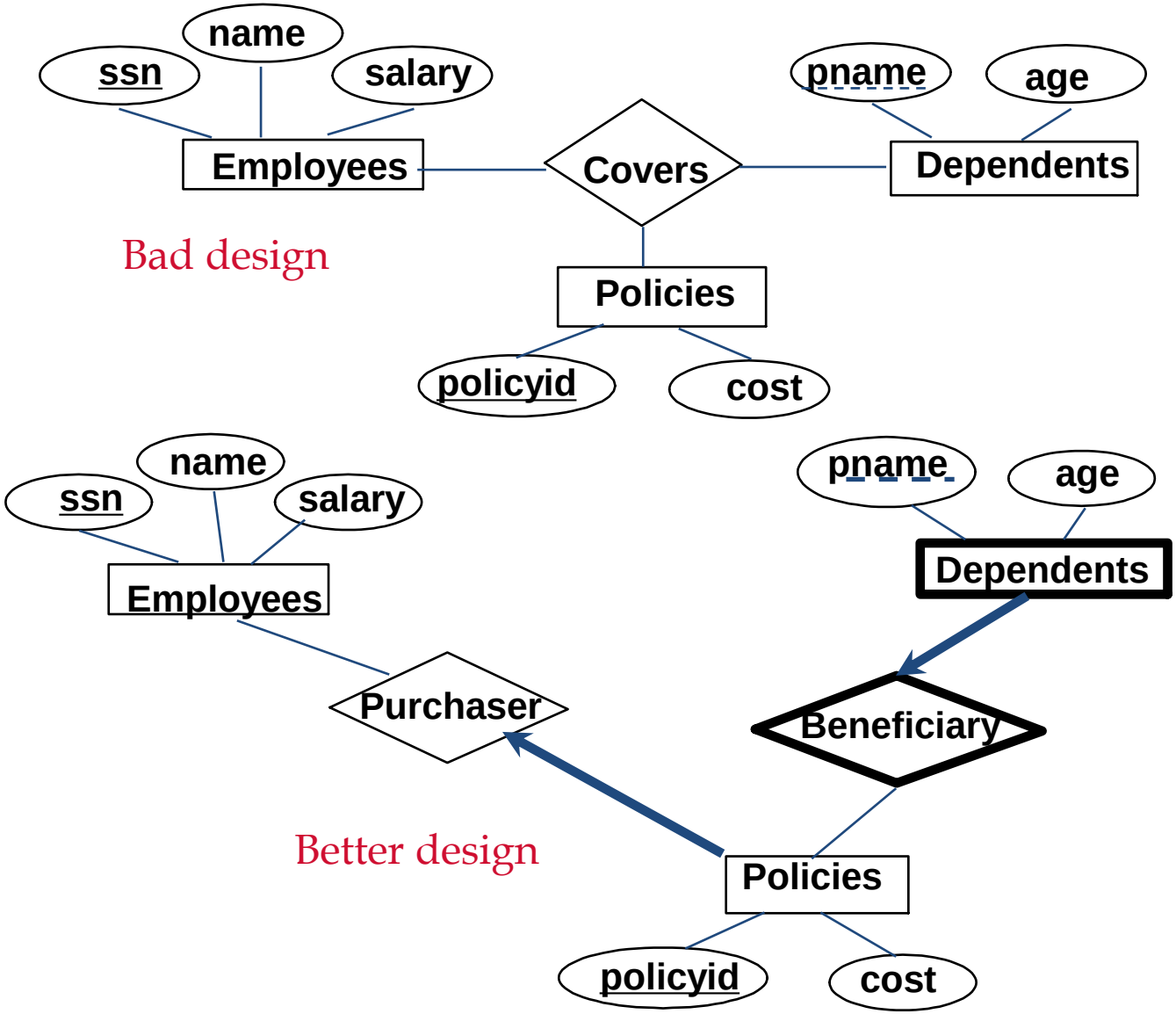


- **What if a manager gets a discretionary budget that covers all managed depts?**
  - Redundancy: *dbudget* stored for each dept managed by manager
  - Misleading: Suggests *dbudget* associated with department-mgr combination



Binary vs. Ternary Relationships

- If each policy is owned by just 1 employee, and each dependent is tied to the covering policy, first diagram is inaccurate
- What are the additional constraints in the 2nd diagram?



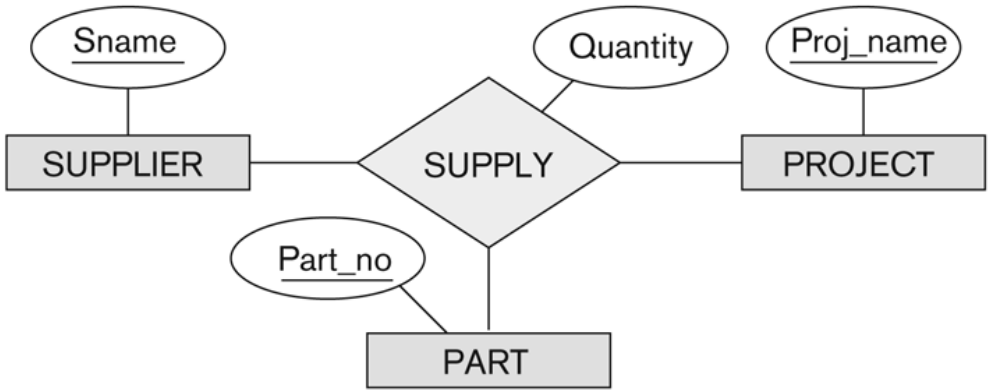
Binary vs. Ternary Relationships (Contd.)

- **Previous example illustrated a case when two binary relationships were better than one ternary relationship**

- **An example in the other direction: a ternary relation Contracts relates entity sets Parts, Departments and Suppliers, and has descriptive attribute qty. No combination of binary relationships is an adequate substitute:**
  - S "can-supply" P, D "needs" P, and D "deals-with" S does not imply that D has agreed to buy P from S
  - How do we record qty?

Examples of Ternary relationship

- **Project-Supplier-Part**



- **Other? (HW#1)**

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 foreign key constraints are also implicit in the definition of a relationship set**
  - Some constraints (notably, functional dependencies) cannot be expressed in the ER model
  - 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 n-ary relationship
  - whether or not to use ISA hierarchies
- **Ensuring good database design: resulting relational schema should be analyzed and refined further**
- **FD information and normalization techniques are especially useful**

Living database

HW#1: Design ER for your app-data

- **Platforms: Windows, MacOS, Linux, Android, iOS, …**
- **Two types of entities: app, data**
- **Suggest the way of collecting data**
- **Submit 3-page report**
  - p.1 : ER diagram (make it understandable!)
  - p.2 : the way of collecting the data
  - p.3 : one of more examples of the ternary relationship

InfoLab

IC611: Database Systems Design

The Relational Model

from Database Management Systems, 3rd ed.

DGIST 대구경북과학기술원
Daegu Gyeongbuk Institute of Science & Technology

Why Study the Relational Model?

- **Most widely used model**
  - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- **“Legacy systems” in older models**
  - e.g., IBM’s IMS
- **Recent competitor: object-oriented model**
  - ObjectStore, Versant, Ontos
  - A synthesis emerging: object-relational model
    - Informix Universal Server, UniSQL, O2, Oracle, DB2

Non-relational models

Relational Database: Definitions

- **Relational database: a set of relations**

- **Relation: made up of two parts:**
  - Instance : a table, with rows and columns
    - #Rows = cardinality
    - #fields = degree / arity
  - Schema : specifies name of relation, plus name and type of each column
    - e.g., Students(sid: string, name: string, login: string, age: integer, gpa: real)

- **Can think of a relation as a set of rows or tuples (i.e., all rows are distinct)**

Example Instance of Students Relation

- **Cardinality = 3, degree = 5, all rows distinct**
- **Do all columns in a relation instance have to be distinct?**

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Relational Query Languages

- **A major strength of the relational model: supports simple, powerful querying of data**

- **Queries can be written intuitively, and the DBMS is responsible for efficient evaluation**
  - The key: precise semantics for relational queries
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change

The SQL Query Language

- **Developed by IBM (system R) in the 1970s**
- **Need for a standard since it is used by many vendors**
- **Standards:**
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision)
  - SQL-99 (major extensions, current standard)

Basic SQL Query

- **relation-list:** A list of relation names (possibly with a tuple variable (range variable) after each name)
- **target-list:** A list of attributes of relations in relation-list
- **qualification:** Comparisons (Attr op const or Attr1 op Attr2, where op is one of $<, >, =, \leq, \geq, \neq$) combined using AND, OR and NOT
- **DISTINCT** is an optional keyword indicating that the answer should not contain duplicates
  - Default is that duplicates are not eliminated!

```
SELECT    [DISTINCT] target-list
FROM      relation-list
WHERE     qualification
```

Query vs. App

- Types of queries: structured vs. unstructured

- Queries for non-relational models

Basic SQL Query : create table

```
CREATE  TABLE  table_name
(
        list_of_attributes        attribute_domain

        [options:
                - PRIMARY KEY (attributes)
                - FOREIGN KEY (attributes)        REFERENCES (table_name)
                - ON DELETE/UPDATE
                        options:
                        - CASCADE
                        - NO ACTION (default)
                        - SET DEFAULT
        ]
);
```

Basic SQL Query: insert

INSERT INTO *table _name* (*attribute_list*)
VALUE (***value_of_attribute_list***)

- **For example**
  - INSERT INTO Students (sid, name, login, age, gpa)
    VALUES (53688, ‘Smith’, ‘smith@ee’, 18, 3.2)

Basic SQL Query: delete

```
DELETE
FROM    table _name    Alias_name
WHERE   qualification
```

- **For example**
  - DELETE
    FROM Students S
    WHERE S.name = 'Smith'

Basic SQL Query: update

```
UPDATE   table _name      Alias_name
SET      adjust_equation_list
WHERE    qualification
```

- **For example**
  - UPDATE Students S
    SET S.age = S.age + 1, S.gpa = S.gpa –1
    WHERE S.sid = 53688

Read-only vs. Write

- **Read and write?**

- **Counterintuitive idea**

Integrity Constraints (ICs)

- **IC: condition that must be true for any instance of the database; e.g., domain constraints**
  - ICs are specified when schema is defined
  - ICs are checked when relations are modified

- **A legal instance of a relation is one that satisfies all specified ICs**
  - DBMS should not allow illegal instances

- **If the DBMS checks ICs, stored data is more faithful to real-world meaning**
  - Avoids data entry errors, too!

Type checking in Programming Language

- Type casting in C/C++
- In Python?
- Data Model and Data Type
- How about Artificial Intelligence (AI)?

Primary Key Constraints

- **A set of fields is a (candidate) key for a relation if :**
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key
     - Part 2 false? A superkey
     - If there's >1 key for a relation, one of the keys is chosen (by DBA) to be the primary key

- **e.g., *sid* is a key for Students**
  - What about name?
  - The set {sid, gpa} is a superkey

Primary and Candidate Keys in SQL

- Possibly many candidate keys (specified using UNIQUE), one of which is chosen as the primary key
- "For a given student and course, there is a single grade."

vs.

- "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."
- Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid) )
```

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid),
   UNIQUE (cid, grade) )
```

Foreign Keys, Referential Integrity

- **Foreign key : Set of fields in one relation that is used to `refer' to a tuple in another relation**
  - Must correspond to primary key of the second relation
  - Like a logical pointer

- **e.g., *sid* is a foreign key referring to Students:**
  - Enrolled(sid: string, cid: string, grade: string)
  - If all foreign key constraints are enforced, **referential integrity** is achieved, i.e., **no dangling references**
  - Can you name a data model w/o referential integrity?
    - Links in HTML!

Foreign Keys in SQL

- **Only students listed in the Students relation should be allowed to enroll for courses**

```
CREATE TABLE Enrolled
  (sid CHAR(20), cid CHAR(20), grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Physical vs. Logical Pointer

- **Pointer and Indirection**

Enforcing Referential Integrity

- **Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students**
- **What should be done if an Enrolled tuple with a non-existent student *id* is inserted? (Reject it!)**
- **What should be done if a Students tuple is deleted?**
  - Also delete all Enrolled tuples that refer to it
  - Disallow deletion of a Students tuple that is referred to
  - Set *sid* in Enrolled tuples that refer to it to a default *sid*
  - (In SQL, also: Set *sid* in Enrolled tuples that refer to it to a special value null, denoting `unknown' or `inapplicable')
- **Similar if primary key of Students tuple is updated**

Referential Integrity in SQL

- **SQL/92 and SQL:1999 support all 4 options on deletes and updates**
  - Default is NO ACTION (delete/update is rejected)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
   (sid CHAR(20),
    cid CHAR(20),
    grade CHAR(2),
    PRIMARY KEY (sid,cid),
    FOREIGN KEY (sid)
      REFERENCES Students
            ON DELETE CASCADE
            ON UPDATE SET DEFAULT )
```

Where do ICs Come From?

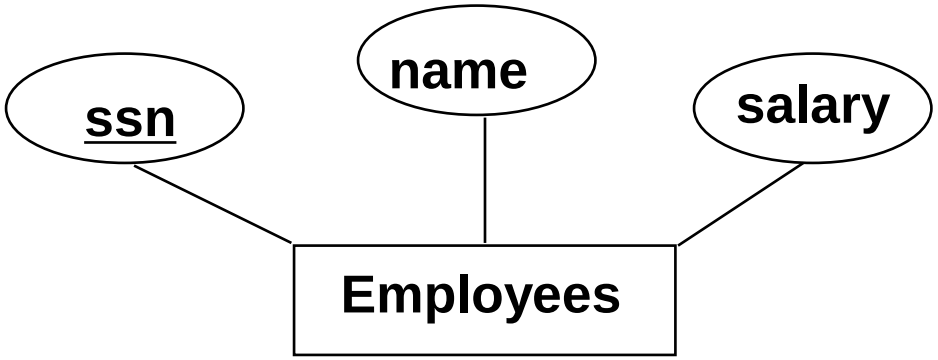
- **ICs are based upon the semantics of the real-world enterprise that is being described in the database relations**

- **We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance**
  - An IC is a statement about all possible instances!
  - From example, we know *name* is not a key, but the assertion that *sid* is a key is given to us

- **Key and foreign key ICs are the most common; more general ICs supported too**

Logical DB Design: ER to Relational

- **Entity sets to tables:**




```
CREATE TABLE Employees
    (ssn CHAR(11),
    name CHAR(20),
    salary INTEGER,
    PRIMARY KEY  (ssn))
```

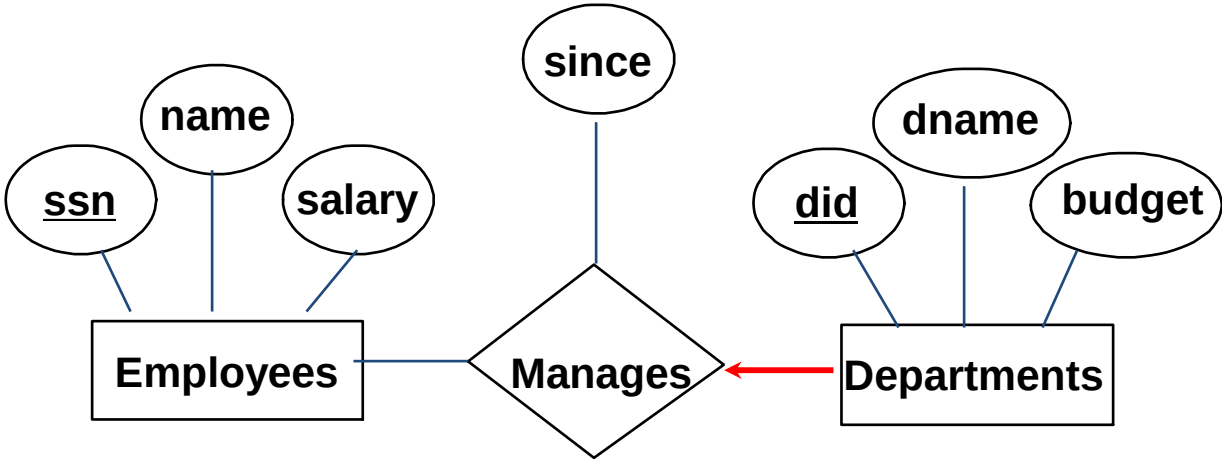
Relationship Sets to Tables

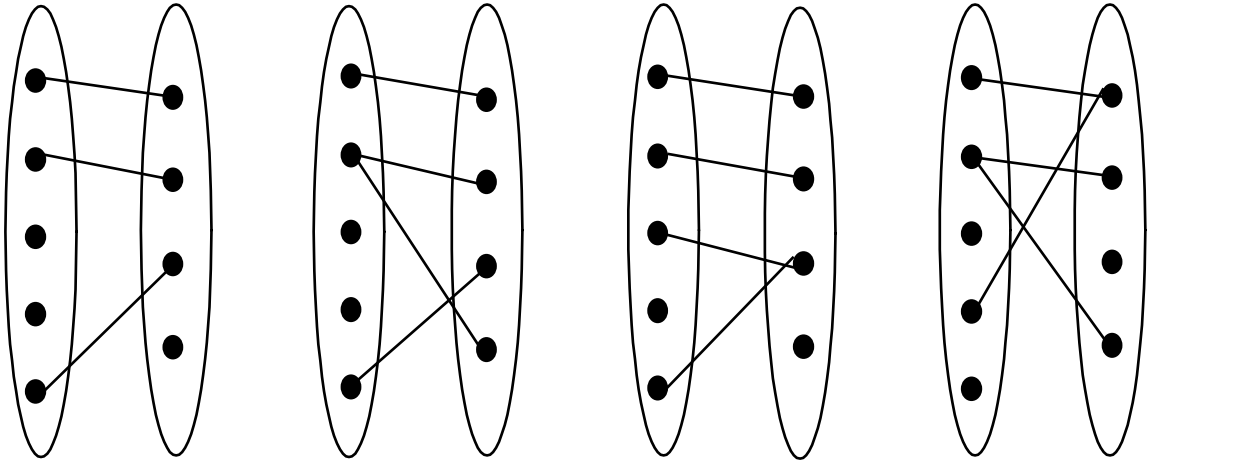
- **In translating a relationship set to a relation, attributes of the relation must include:**
  - Keys for each participating entity set (as foreign keys)
    - This set of attributes forms a superkey for the relation
  - All descriptive attributes

```
CREATE TABLE Works_In(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
      REFERENCES Employees,
  FOREIGN KEY (did)
      REFERENCES Departments)
```

Review: Key Constraints

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





1-to-1 | 1-to Many | Many-to-1 | Many-to-Many

Translation to relational model?

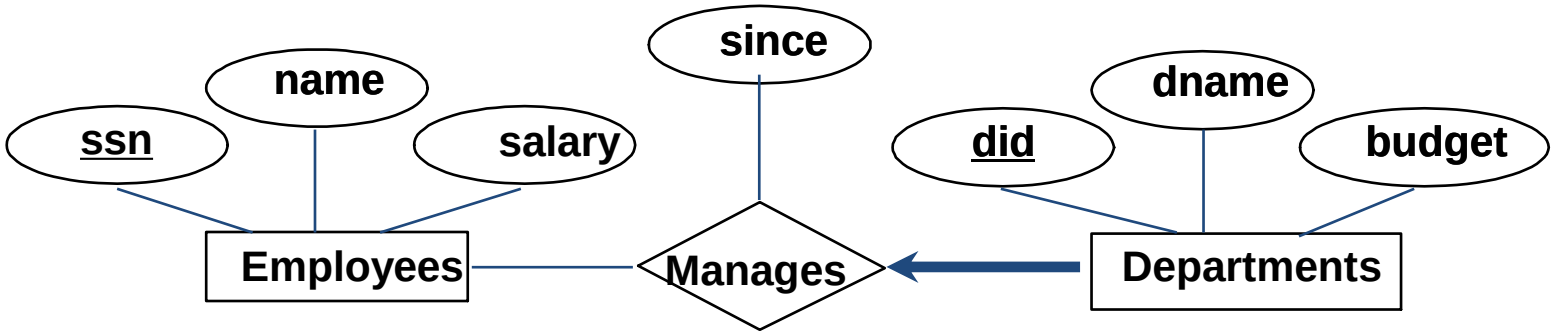
Translating ER Diagrams with Key Constraints

- **Map relationship to a table:**
  - Note that *did* is the key now!
  - Separate tables for Employees and Departments

```
CREATE TABLE Manages(
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments)
```

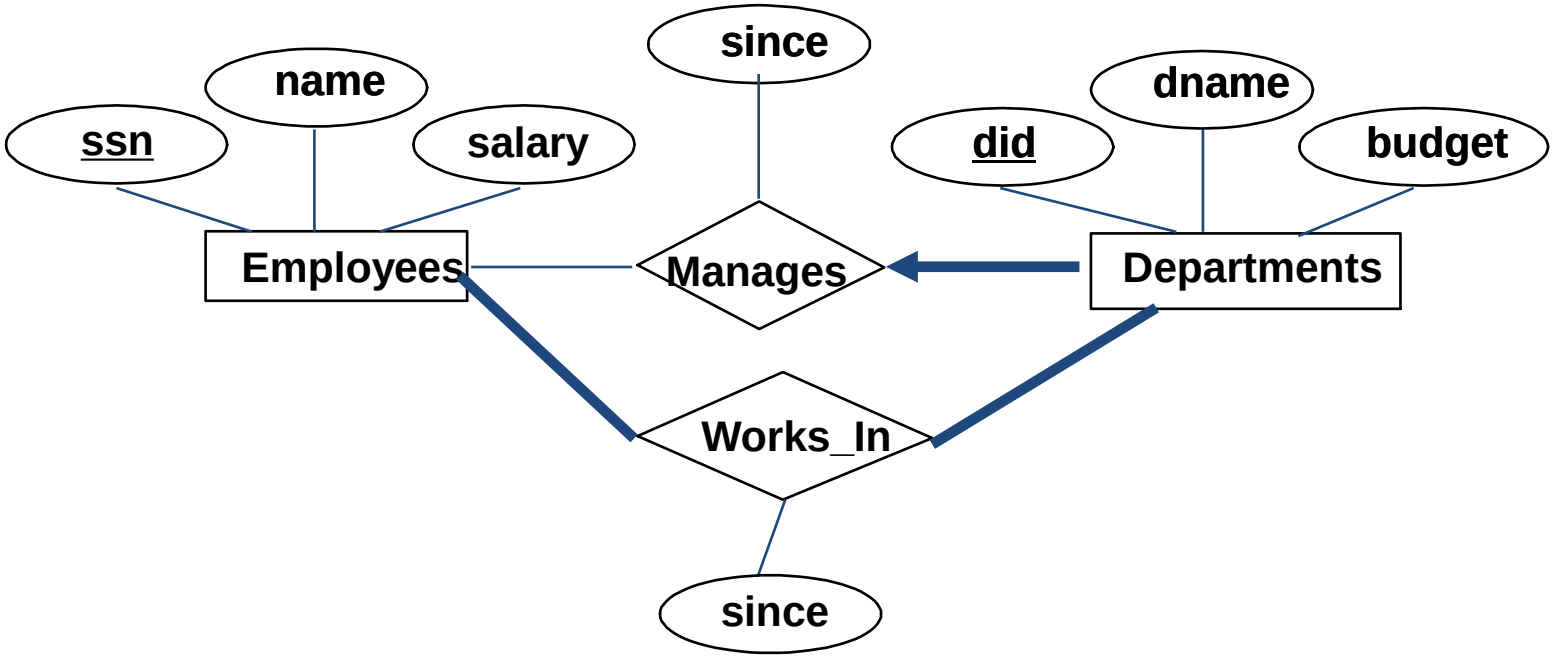
- **Since each department has a unique manager, we could instead combine Manages and Departments**

```
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees)
```


since
name
dname
ssn
salary
did
budget
Employees
Manages
Departments

Review: Participation Constraints

- **Does every department have a manager?**
  - If so, this is a participation constraint: the participation of Departments in Manages is said to be total (vs. partial)
    - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)



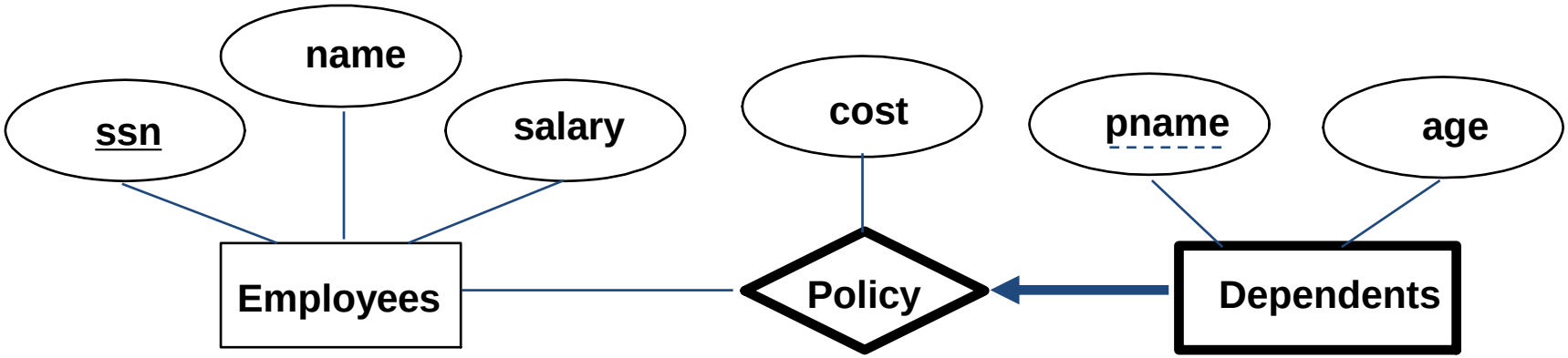
Participation Constraints in SQL

- **We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints)**

```
CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
      ON DELETE NO ACTION)
```

Review: 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 (1 owner, many weak entities)
  - Weak entity set must have total participation in this identifying relationship set



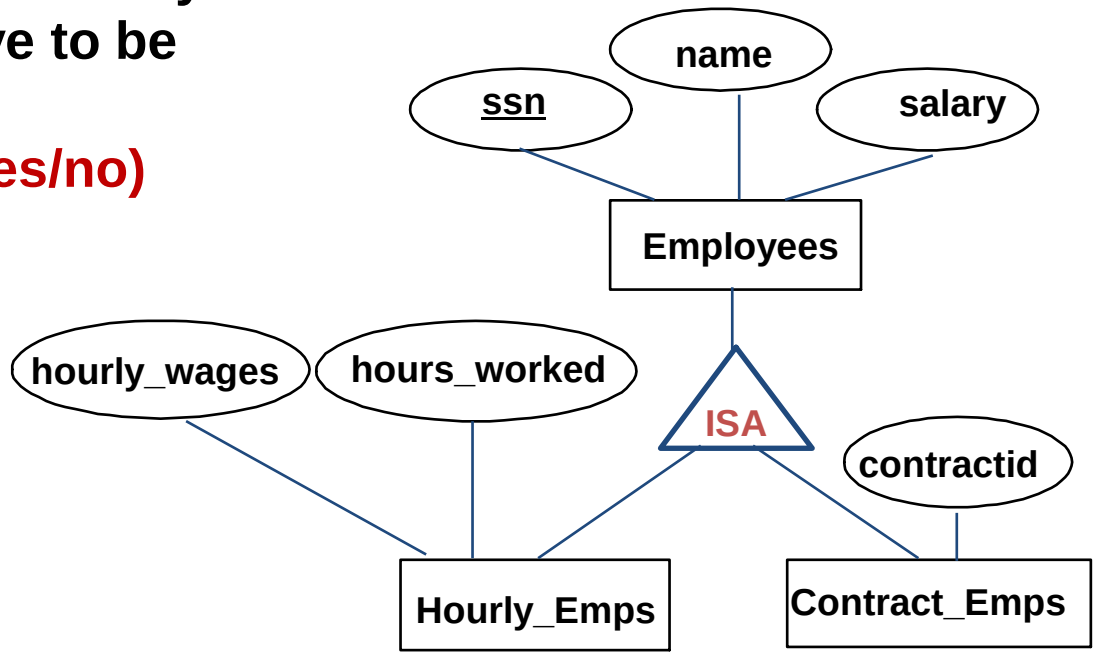
Translating Weak Entity Sets

- **Weak entity set and identifying relationship set are translated into a single table**
  - When the owner entity is deleted, all owned weak entities must also be deleted

```
CREATE TABLE Dep_Policy (
   pname  CHAR(20),
   age  INTEGER,
   cost  REAL,
   ssn  CHAR(11) NOT NULL,
   PRIMARY KEY  (pname, ssn),
   FOREIGN KEY  (ssn) REFERENCES Employees,
     ON DELETE CASCADE)
```

Review: ISA Hierarchies

- As in C++, or other PLs, attributes are inherited.
- 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)



Translating ISA Hierarchies to Relations

- **General approach:**
  - 3 relations: Employees, Hourly_Emps and Contract_Emps
    - Hourly_Emps:
      - Every employee is recorded in Employees
      - For hourly emps, extra info recorded in Hourly_Emps (hourly_wages, hours_worked, ssn);
      - must delete Hourly_Emps tuple if referenced Employees tuple is deleted)
    - Queries involving all employees easy, those involving just Hourly_Emps require a join to get some attributes

- **Alternative: Just Hourly_Emps and Contract_Emps**
  - Hourly_Emps: ssn, name, lot, hourly_wages, hours_worked
  - Each employee must be in one of these two subclasses

Review: Binary vs. Ternary Relationships

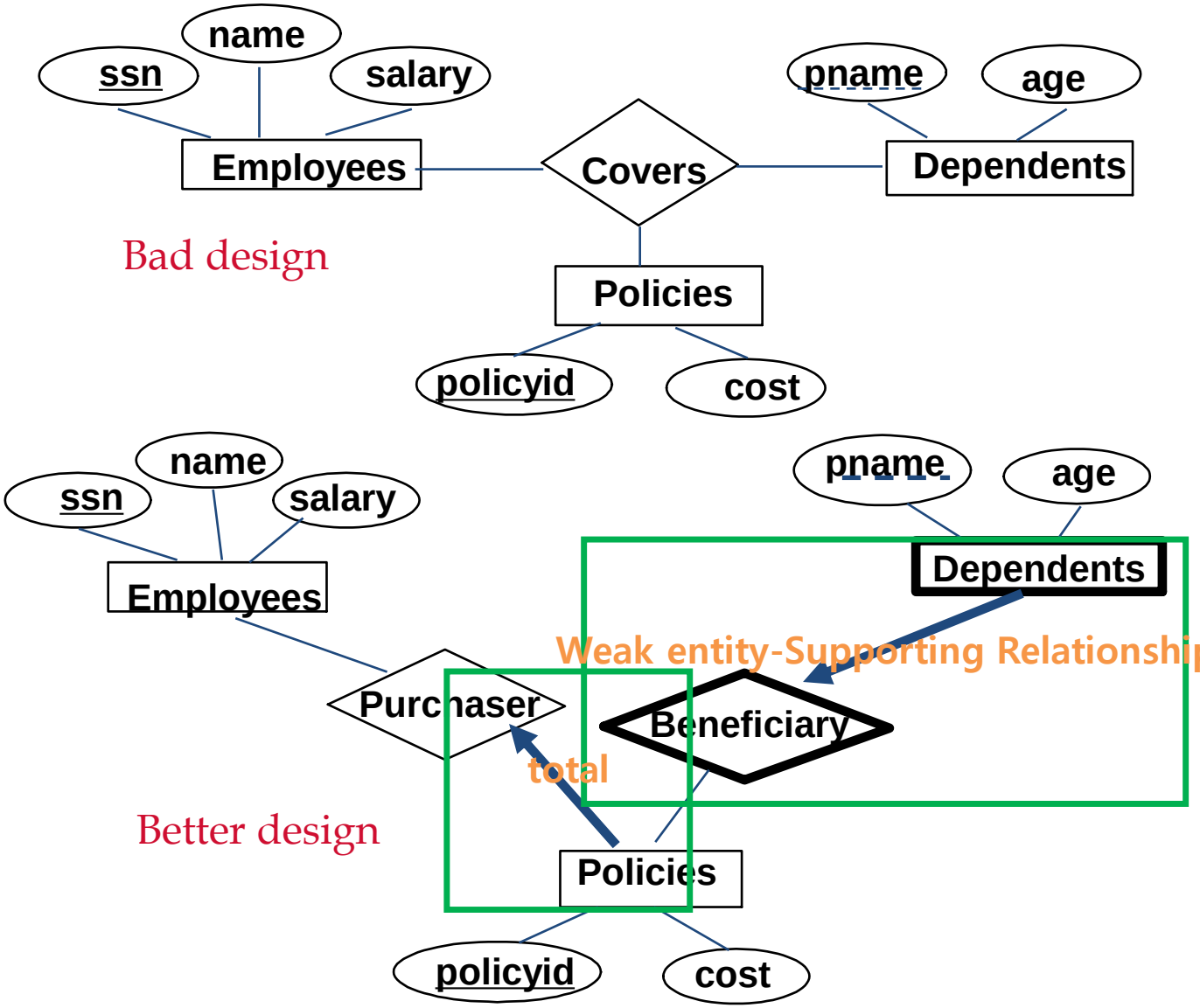
- An Employee can own several policies
- Each policy can be owned by several employees
- Each dependent can be covered by several policies

vs.

- A policy cannot be owned jointly by two or more employees
  - 1-to-many relationship for Employee-Policy
- Every policy must be owned by some employee
  - Total participation
- Dependents is a **weak entity set**, and each dependent entity is uniquely identified by taking *pname* in conjunction with the *policyid* of a policy
  - Weak entity-supporting relationship

Review: Binary vs. Ternary Relationships

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



Binary vs. Ternary Relationships (Contd.)

- The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents

```
CREATE TABLE Policies (
  policyid INTEGER,
  cost REAL,
  ssn CHAR(11) NOT NULL,
  PRIMARY KEY (policyid),
  FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE CASCADE)
```

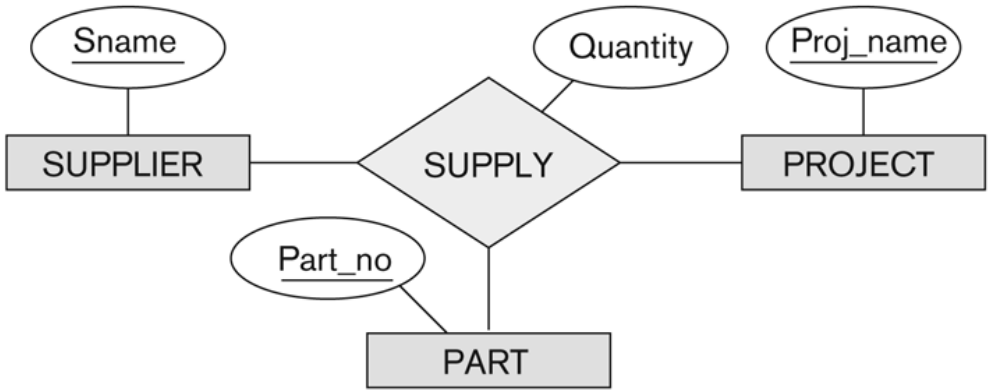
- Participation constraints lead to NOT NULL constraints

- What if Policies is a weak entity set?

```
CREATE TABLE Dependents (
  pname CHAR(20),
  age INTEGER,
  policyid INTEGER,
  PRIMARY KEY (pname, policyid),
  FOREIGN KEY (policyid) REFERENCES Policies,
    ON DELETE CASCADE)
```

Examples of Ternary relationship

- **Project-Supplier-Part**



Views

- **A view is just a relation, but we store a definition, rather than a set of tuples**

```
CREATE VIEW YoungActiveStudents (name, grade)
        AS SELECT S.name, E.grade
        FROM Students S, Enrolled E
        WHERE S.sid = E.sid and S.age<21
```

- **Views can be dropped using the DROP VIEW command**
  - How to handle DROP TABLE if there's a view on the table?
    - DROP TABLE command has options to let the user specify this

Relational Model: Summary

- **A tabular representation of data**
- **Simple and intuitive, currently the most widely used**
- **Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations**
  - Two important ICs: primary and foreign keys
  - In addition, we always have domain constraints
- **Powerful and natural query languages exist**
- **Rules to translate ER to relational model**

Thinking a lot

- **Density (Concentration)**
- **Volume (Time)**
- ***Third factor??***