

CIS 330: Applied Database Systems

Lecture 3:
Introduction to ER Modeling
The Relational Model

Johannes Gehrke
johannes@cs.cornell.edu
<http://www.cs.cornell.edu/johannes>

Announcements

- How many laptops have problems?
 - If you do not have a laptop, contact Biswanath (bpanda@cs.cornell.edu)
 - If you have a laptop, try the following imaging process:
 - Image
 - Log in and then wait for 20 minutes (do not start any program, just let the computer "rest")
 - If it freezes, press power until it shuts off and then turn it on again
 - If your laptop is then still broken, contact Josh Gerner, jgerner@cs.cornell.edu. Third floor Upson.
 - Download www.zonealarm.com
- Lecture switch this week:
 - This week: ER model and relational model
 - Next Tuesday: Three-tier technologies
- Start reading the textbook

Goals of This Lecture

- Understand ER modeling
- Understand the basics of the relational model

Last Lecture

- Why Store Data in a DBMS?
 - Transactions (concurrent data access, recovery from system crashes)
 - High-level abstractions for data access, manipulation, and administration
 - Data integrity and security
 - Performance and scalability
- Some introductory remarks about keys etc. We will review this in today's lecture.

The Database Design Process

- Requirement analysis
- Conceptual design using the entity-relationship (ER) model
- Schema refinement
- Normalization
- Physical tuning

Overview of Database Design

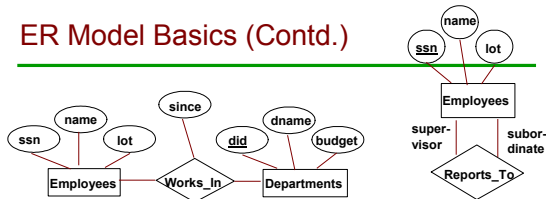
- *Conceptual design: (ER Model is used at this stage.)*
 - What are the *entities* and *relationships* in the enterprise?
 - What information about these entities and relationships should we store in the database?
 - What are the *integrity constraints* or *business rules* that hold?
 - A database `schema' 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. (Until we consider ISA hierarchies, anyway!)
 - Each entity set has a **key**.
 - Each attribute has a **domain**.

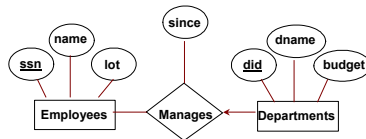
ER Model Basics (Contd.)



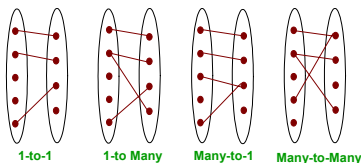
- **Relationship:** Association among two or more entities. E.g., Attishoo works in Pharmacy department.
- **Relationship Set:** Collection of similar relationships.
 - An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1 E1, ..., en En
 - Same entity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

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.

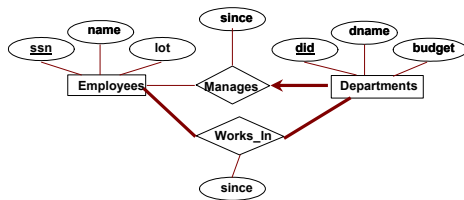


Key Constraints: Examples

- Example Scenario 1: An inventory database contains information about parts and manufacturers. Each part is constructed by exactly one manufacturer.
- Example Scenario 2: A customer database contains information about customers and sales persons. Each customer has exactly one primary sales person.
- What do the ER diagrams look like?

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)*.

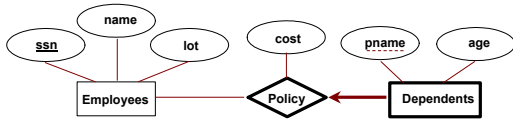


Participation Constraints: Examples

- Example Scenario 1 (Contd.): Each part is constructed by exactly one or more manufacturer.
- Example Scenario 2: Each customer has exactly one primary sales person.

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.

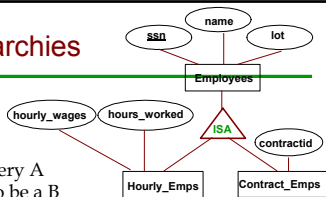


Exercise

- Give two real-life examples where each of the following would occur:
 - A key constraint
 - A participation constraint
 - A weak entity set

ISA ('is a') Hierarchies

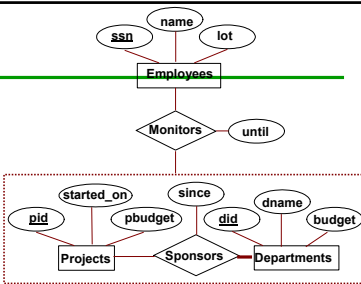
- ❖ As in Java or other PLs, attributes are inherited.
- ❖ If we declare A **ISA** B, every A entity is also considered to be a B entity.



- **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.

Aggregation

- Used when we have to model a relationship involving (entity sets and) a *relationship set*.



- Aggregation** allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.

- Aggregation vs. ternary relationship:**
 - ❖ Monitors is a distinct relationship, with a descriptive attribute.
 - ❖ Also, can say that each sponsorship is monitored by at most one employee.

ER Modeling: Case Study

Drugwarehouse.com has offered you a free life-time supply of prescription drugs (no questions asked) if you design its database schema. Given the rising cost of health care, you agree. Here is the information that you gathered:

- Patients are identified by their SSN, and we also store their names and age.
- Doctors are identified by their SSN, and we also store their names and specialty.
- Each patient has one primary care physician, and we want to know since when the patient has been with her primary care physician.
- Each doctor has at least one patient.

Conceptual Design Using the ER Model

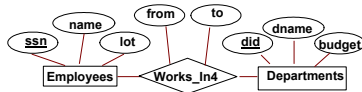
- Design choices:**
 - Should a concept be modeled as an entity or an attribute?
 - Should a concept be modeled as an entity or a relationship?
 - Identifying relationships: Binary or ternary? Aggregation?
- Constraints in the ER Model:**
 - A lot of data semantics can (and should) be captured.
 - But some constraints cannot be captured in ER diagrams. (What would be an example?)

Entity vs. Attribute

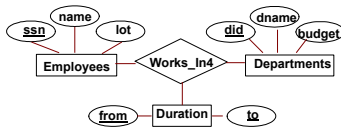
- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
- Depends upon the use we want to make of address information, and the semantics of the data:
 - If we have several addresses per employee, *address* must be an entity (since attributes cannot be 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.



- Similar to the problem of wanting to record several 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.

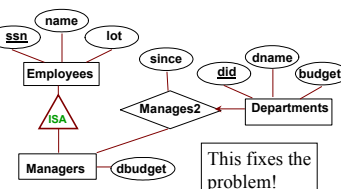


Entity vs. Relationship

- 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?



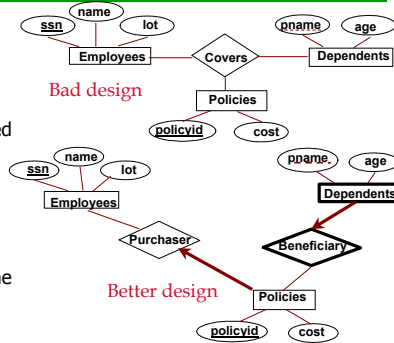
- **Redundancy:** *dbudget* stored for each dept managed by manager.
- **Misleading:** Suggests *dbudget* associated with department-mgr combination.



This fixes the problem!

Binary vs. Ternary Relationships

- If each policy is owned by just one 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*?

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*, and *aggregation*.
- Note: There are many variations on the 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, and whether or not to use aggregation.
- Ensuring good database design: resulting relational schema should be analyzed and refined further. FD information and normalization techniques are especially useful.

Case Study: The Internet Bookshop

Your customer:

"I would like my customers to be able to browse my catalog of books and place orders over the Internet. Currently, I take orders over the phone. I have mostly corporate customers who call me and give me the ISBN number of a book and a quantity; they often pay by credit card. I then prepare a shipment that contains the books they ordered. If I don't have enough copies in stock, I order additional copies and delay the shipment until the new copies arrive; I want to ship a customer's entire order together.

My catalog includes all the books I sell. For each book, the catalog contains its ISBN number, title, author, purchase price, sales price, and the year the book was published. Most of my customers are regulars, and I have records with their names and addresses. New customers have to call me first and establish an account before they can use my website.

On my new website, customers should first identify themselves by their unique customer identification number. Then they should be able to browse my catalog and to place orders online."

The Relational Data Model

A relational database is a set of relations. Turing Award (Nobel Prize in CS) for Codd in 1980 for his work on the relational model.

- Example relation:

Customers(cid: integer, name: string, byear: integer, state: string)

cid	name	byear	state
1	Jones	1960	NY
2	Smith	1974	CA
3	Smith	1950	NY

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 and object-relational data model, XML
 - OO: ObjectStore, Versant, Ontos
 - OR: Oracle, DB2, Microsoft
 - XML: Everybody

Relational Database: Definitions

- **Relational database**: a set of **relations**
- **Relation**: made up of 2 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.
 - **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

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

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

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)

The SQL Query Language

- To find all 18 year old students, we can write:

```
SELECT *
FROM Students S
WHERE S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

- To find just names and logins, replace the first line:

```
SELECT S.name, S.login
```

Querying Multiple Relations

- What does the following query compute?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade="A"
```

Given the following instance of Enrolled (is this possible if the DBMS ensures referential integrity?):

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

S.name	E.cid
Smith	Topology112

Creating Relations in SQL

- Creates the Students relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
(sid: CHAR(20),
name: CHAR(20),
login: CHAR(10),
age: INTEGER,
gpa: REAL)
```
- As another example, the Enrolled table holds information about courses that students take.

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

Destroying and Altering Relations

`DROP TABLE Students`

- Destroys the relation Students. The schema information *and* the tuples are deleted.

`ALTER TABLE Students`

`ADD COLUMN firstYear: integer`

- ❖ The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- ❖ Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

➤ *Powerful variants of these commands are available; more later!*

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!

Primary Key Constraints

- A set of fields is a *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.”

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

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),
lot INTEGER,
PRIMARY KEY (ssn))
```

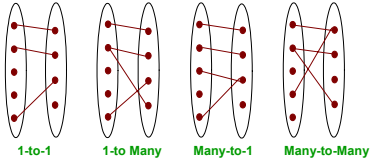
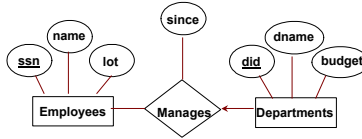
Relationship Sets to Tables

- 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(11),
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.



Translation to relational model?

Translating ER Diagrams with Key Constraints

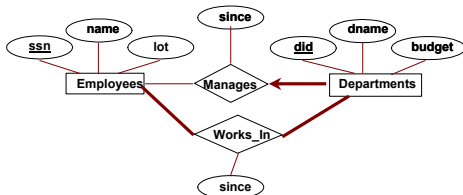
- Map relationship to a table:
 - Note that *did* is the key now!
 - Separate tables for Employees and Departments.
- Since each department has a unique manager, we could instead combine Manages and Departments.

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

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

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!)



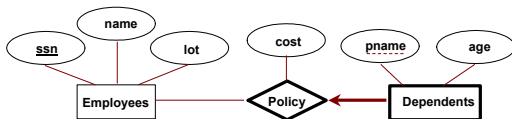
Participation Constraints in SQL

- 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.

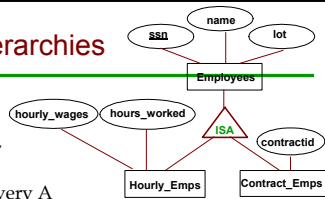


Translating Weak Entity Sets

- 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 $A \text{ ISA } B$, 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*)

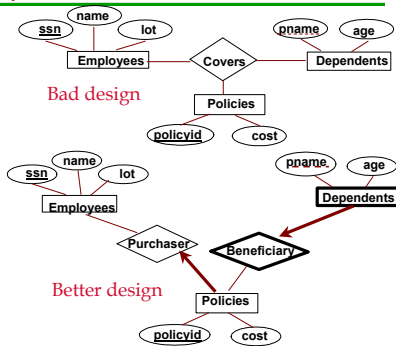
Translating ISA Hierarchies to Relations

• 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

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.

```
CREATE TABLE Dependents (  
  pname CHAR(20),  
  age INTEGER,  
  policyid INTEGER,  
  PRIMARY KEY (pname, policyid),  
  FOREIGN KEY (policyid) REFERENCES Policies,  
  ON DELETE CASCADE)
```
- What if Policies is a weak entity set?

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.

---

---

---

---

---

---

---

---

## Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given YoungStudents, but not Students or Enrolled, we can find students s who have are enrolled, but not the cid's of the courses they are enrolled in.

---

---

---

---

---

---

---

---

### 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

---

---

---

---

---

---

---

---

### Lecture Summary

---

- ER Model
- Relational Model
  
- Questions?

---

---

---

---

---

---

---

---