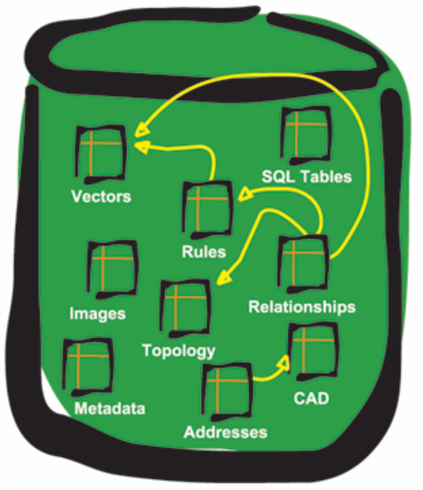
Database Management Systems



Session 2

Instructor: Vinnie Costa
vcosta@optonline.net

Beyond Relational Databases

- http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=299
- *Margo Seltzer, SleepyCat*
- *ACM Queue vol. 3, no. 3 - April 2005*

Term Paper

- Due Saturday, Oct 8
- Should be about 3-4 pages (9 or 10 font)
- Template http://www.acm.org/sigs/pubs/proceed/pubform.doc
- This should be an opportunity to ***explore*** a selected area

Term Paper

- Use Seltzer's Paper As A Launch Pad For Alternatives
- Possible topics:
  - XML Databases
  - Text Searches
  - Data Warehouses
  - Media Databases
  - Appliances
  - Federated Databases
  - Distributed
  - Peer-to-Peer Databases
- **Think Different!!!**

Homework

- Read Chapter One
- Exercises pp.23-24: 1.1, 1.4, 1.6, 1.9
- Read, “Beyond Relational Databases”

Exercise 1.1

- Why would you choose a database system instead of simply storing data in operating system files? When would it make sense *not* to use a database system?

Exercise 1.1

A *database* is an integrated collection of data, usually so large that it has to be stored on secondary storage devices such as disks or tapes. This data can be maintained as a collection of operating system files, or stored in a *DBMS* (database management system). The **advantages** of using a DBMS are:

Exercise 1.1

- Data independence and efficient access
- Reduced application development time
- Data integrity and security
- Data administration
- Concurrent access and crash recovery

If these advantages are not important for the application at hand, using a collection of files may be a better solution because of the increased cost and overhead of purchasing and maintaining a DBMS.

Exercise 1.4

- Explain the difference between external, internal, and conceptual schemas. How are these different schema layers related to the concepts of logical and physical data independence?

Exercise 1.4

- External schemas allows data access to be customized (and authorized) at the level of individual users or groups of users. Conceptual (logical) schemas describes all the data that is actually stored in the database. While there are several views for a given database, there is exactly one conceptual schema to *all* users. Internal (physical) schemas summarize how the relations described in the conceptual schema are actually stored on disk (or other physical media). External schemas provide logical data independence, while conceptual schemas offer physical data independence.

Exercise 1.6

- Scrooge McNugget wants to store information (names, addresses, descriptions of embarrassing moments, etc.) about the many ducks on his payroll. Not surprisingly, the volume of data compels him to buy a database system. To save money, he wants to buy one with the fewest possible features, and he plans to run it as a stand-alone application on his PC clone. Of course, Scrooge does not plan to share his list with anyone. Indicate which of the following DBMS features Scrooge should pay for; in each case, also indicate why Scrooge should (or should not) pay for thateature in the system he buys.

Exercise 1.6

- A security facility.
- A security facility is necessary because Scrooge does not plan to share his list with anyone else. Even though he is running it on his stand-alone PC, a rival duckster could break in and attempt to query his database. The database's security features would foil the intruder.
- Concurrency control.
- Concurrency control is not needed because only he uses the database.
- Crash recovery.
- Crash recovery is essential for any database; Scrooge would not want to lose his data if the power was interrupted while he was using the system.

Exercise 1.6

- A view mechanism.
- A view mechanism is needed. Scrooge could use this to develop “custom screens” that he could conveniently bring up without writing long queries repeatedly.
- A query language.
- A query language is necessary since Scrooge must be able to analyze the dark secrets of his victims. In particular, the query language is also used to define views.

Exercise 1.9

- What is a transaction?
- A transaction is any one execution of a user program in a DBMS. This is the basic unit of change in a DBMS.
- Why does a DBMS interleave the actions of different transactions instead of executing transactions one after the other?
- A DBMS is typically shared among many users. Transactions from these users can be interleaved to improve the execution time of users' queries. By interleaving queries, users do not have to wait for other user's transactions to complete fully before their own transaction begins. Without interleaving, if user A begins a transaction that will take 10 seconds to complete, and user B wants to begin a transaction, user B would have to wait an additional 10 seconds for user A's transaction to complete before the database would begin processing user B's request.

Exercise 1.9

- What must a user guarantee with respect to a transaction and database consistency? What should a DBMS guarantee with respect to concurrent execution of several transactions and database consistency?
- A user must guarantee that his or her transaction does not corrupt data or insert nonsense in the database. For example, in a banking database, a user must guarantee that a cash withdraw transaction accurately models the amount a person removes from his or her account. A database application would be worthless if a person removed 20 dollars from an ATM but the transaction set their balance to zero! A DBMS must guarantee that transactions are executed fully and independently of other transactions. An essential property of a DBMS is that a transaction should execute atomically, or as if it is the only transaction running. Also, transactions will either complete fully, or will be aborted and the database returned to it's initial state. This ensures that the database remains consistent.

Exercise 1.9

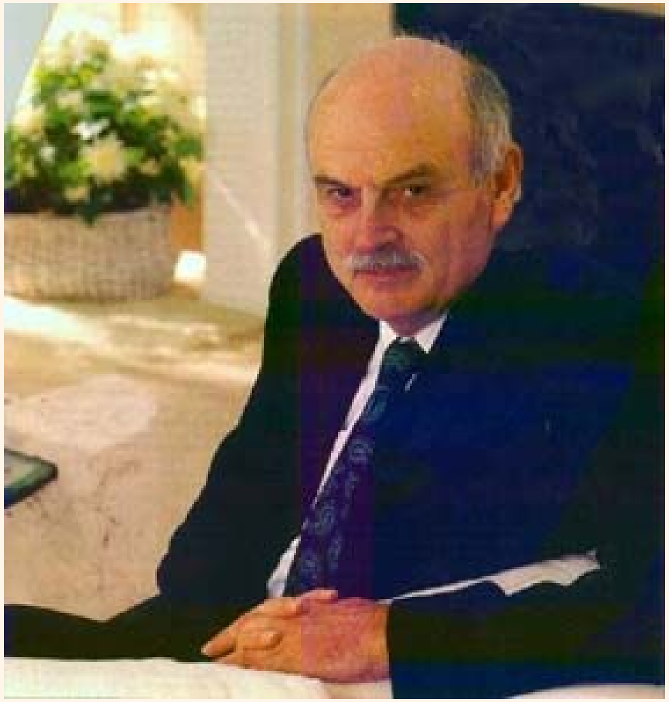
- Explain the strict two-phase locking protocol.
- Strict two-phase locking uses shared and exclusive locks to protect data. A transaction must hold all the required locks before executing, and does not release any lock until the transaction has completely finished.
- What is the WAL property, and why is it important?
- The WAL property affects the logging strategy in a DBMS. The WAL, Write-Ahead Log, property states that each write action must be recorded in the log (on disk) before the corresponding change is reflected in the database itself. This protects the database from system crashes that happen during a transaction's execution. By recording the change in a log before the change is truly made, the database knows to undo the changes to recover from a system crash. Otherwise, if the system crashes just after making the change in the database but before the database logs the change, then the database would not be able to detect his change during crash recovery.

The Entity-Relationship Model

Chapter 2

Edgar (Ted) Codd

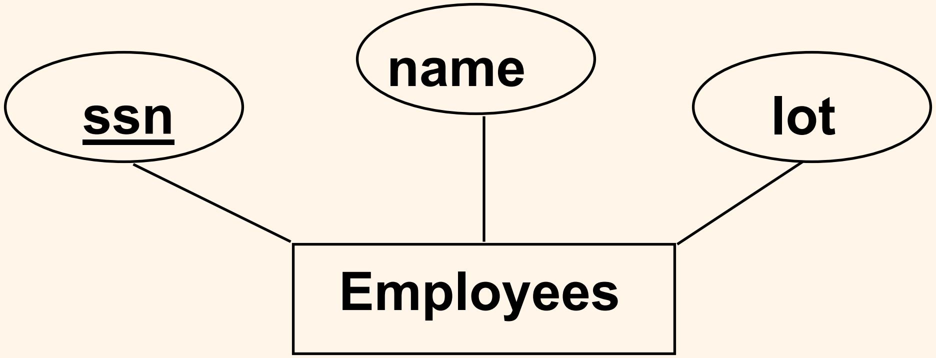
- In his landmark paper, "A Relational Model of Data for Large Shared Data Banks", Codd proposed replacing the hierarchical or navigational structure with simple tables containing rows and columns.
- Led to today's $12 billion database industry



Overview of Database Design

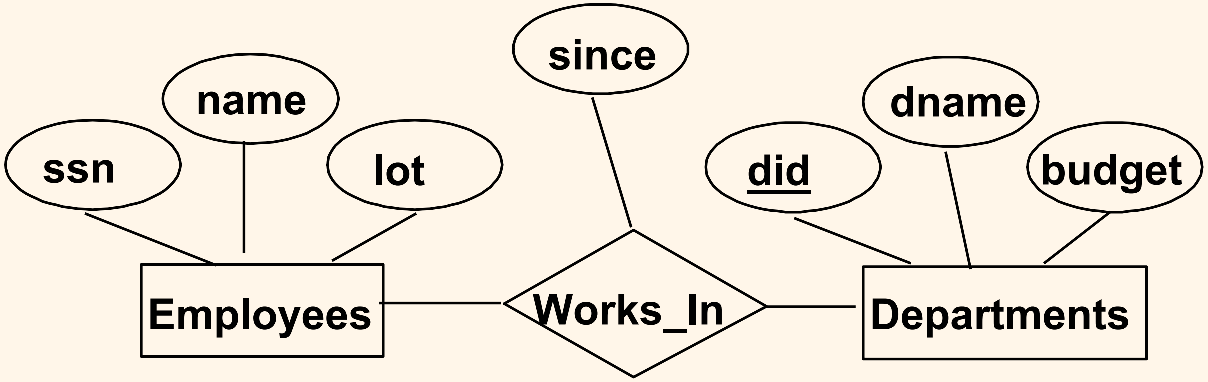
- *Conceptual design*: *(ER Model is used at this stage.)*
  - What are the *entities* and *relationships* in the enterprise?
  - What information about these entities and relationships should we store in the database?
  - What are the *integrity constraints* or *business rules* that hold?
  - A database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

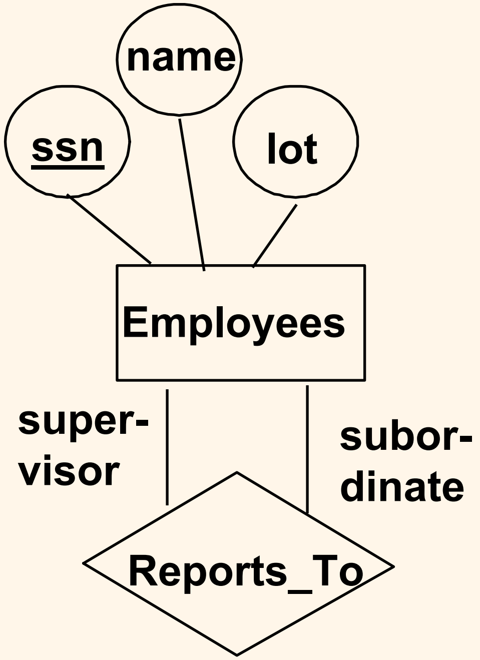
ER Model Basics



- *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*.

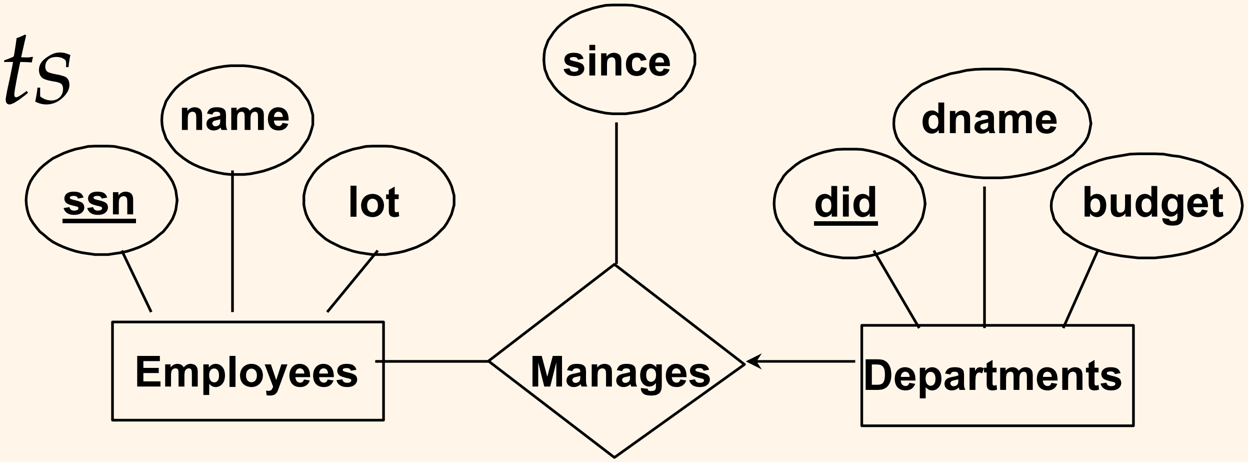
ER Model Basics (Contd.)



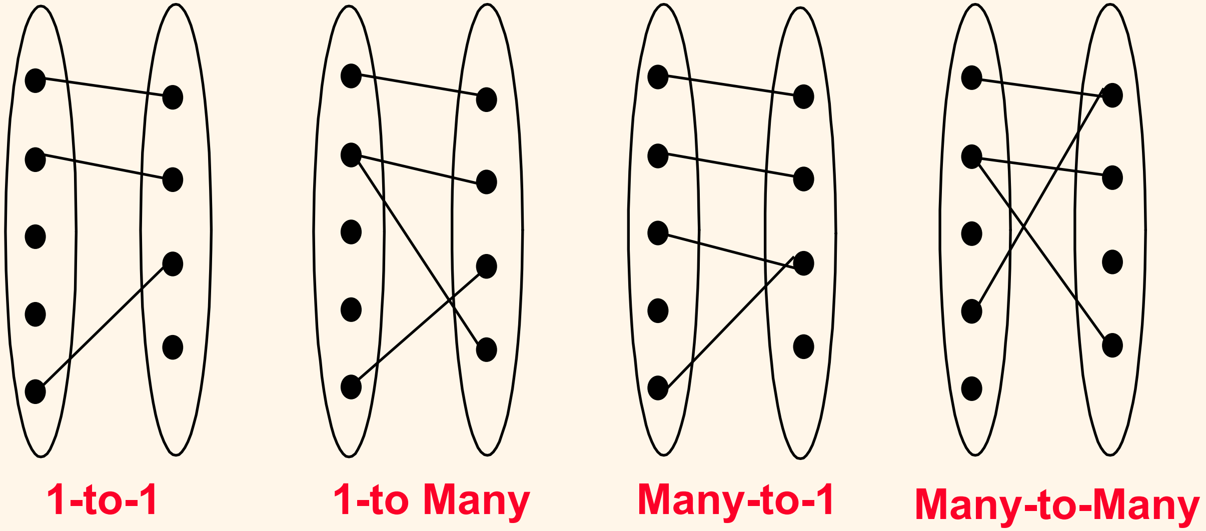


- *Relationship*: Association among two or more entities. E.g., Attishoo works in Pharmacy department.
- *Relationship Set*: Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1 E1, ..., en En
    - Same entity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

Key Constraints

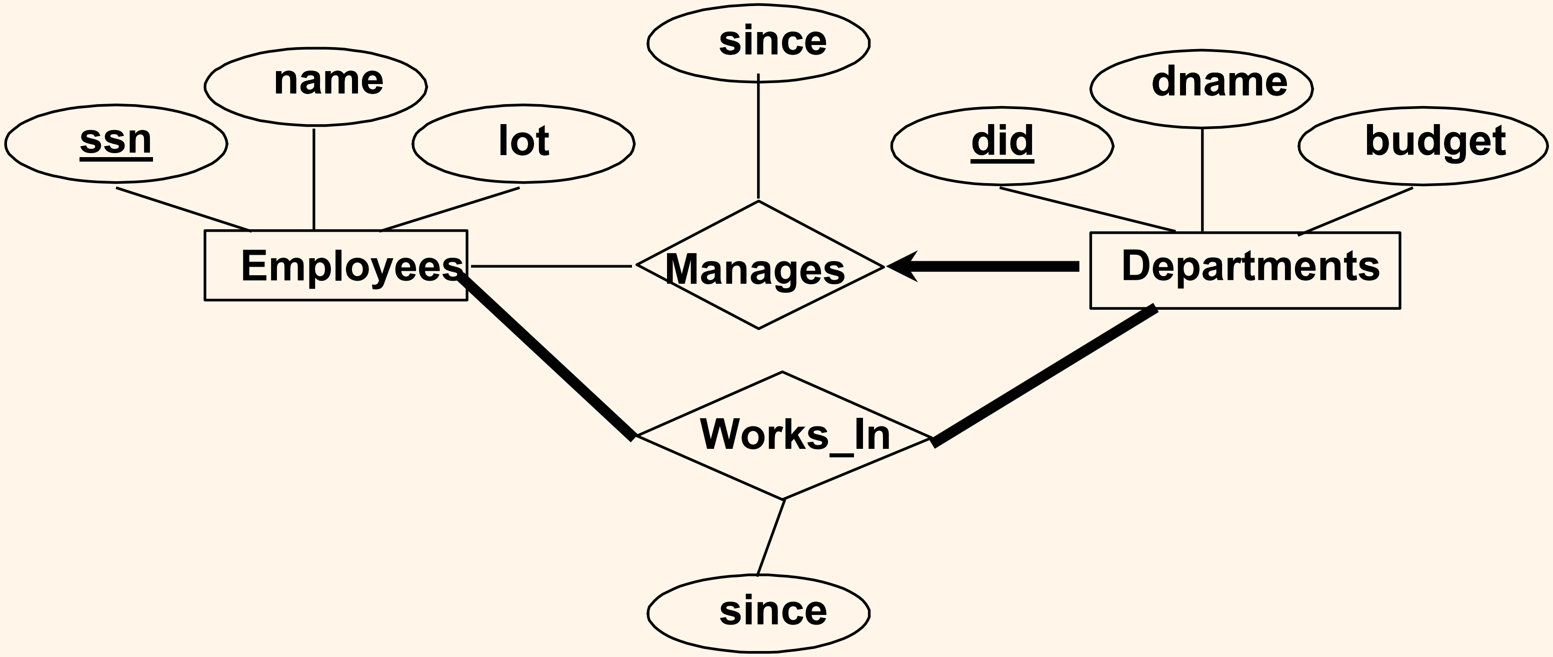


- Consider Works_In: An employee can work in many departments; a dept can have many employees.
- In contrast, each dept has at most one manager, according to the *key constraint* on Manages.



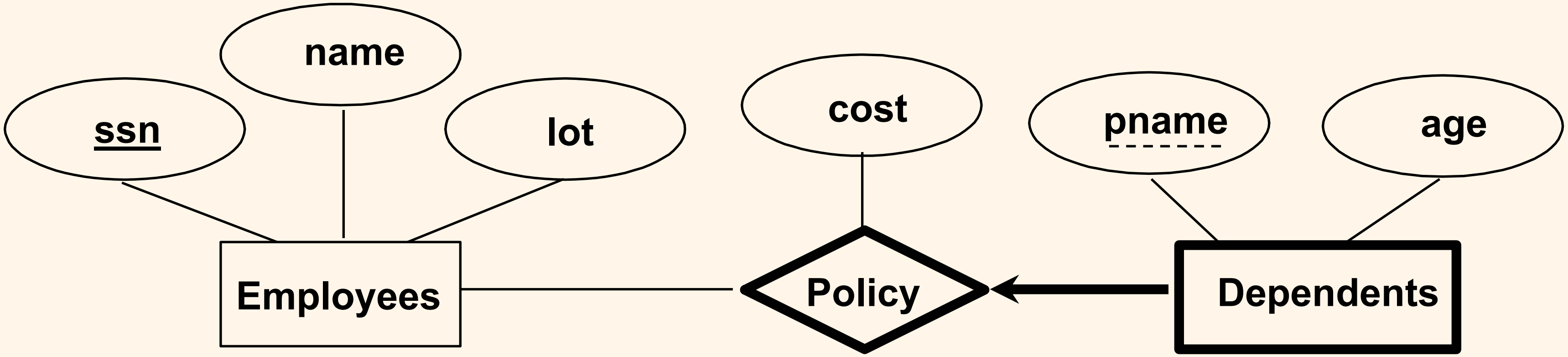
Participation Constraints

- 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 Departments entity must appear in an instance of the Manages relationship.

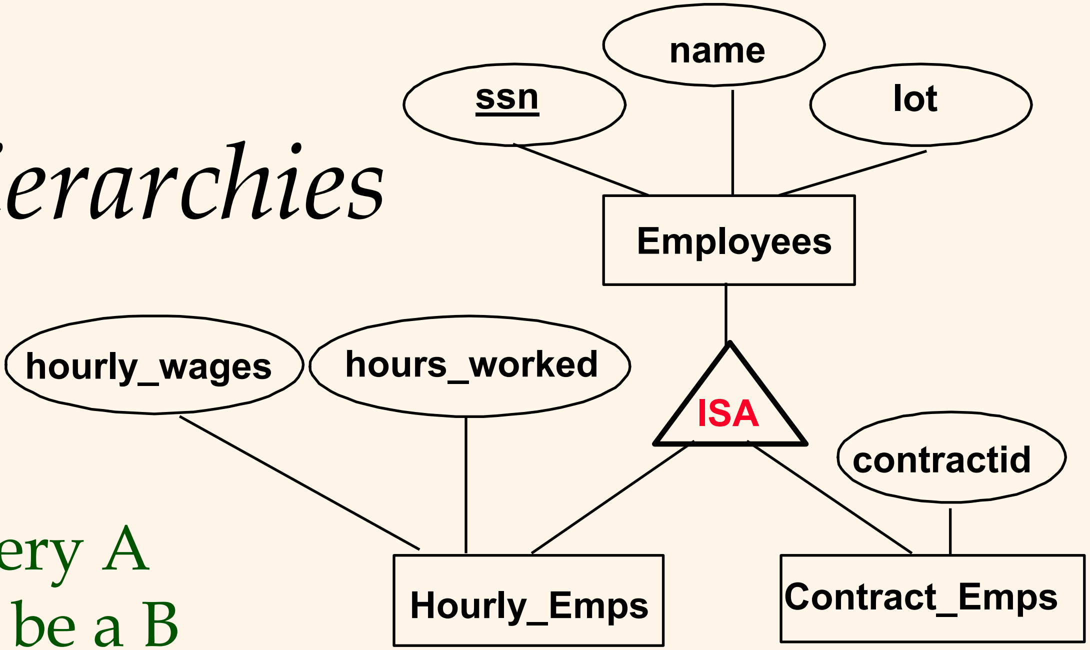


Weak Entities

- 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.



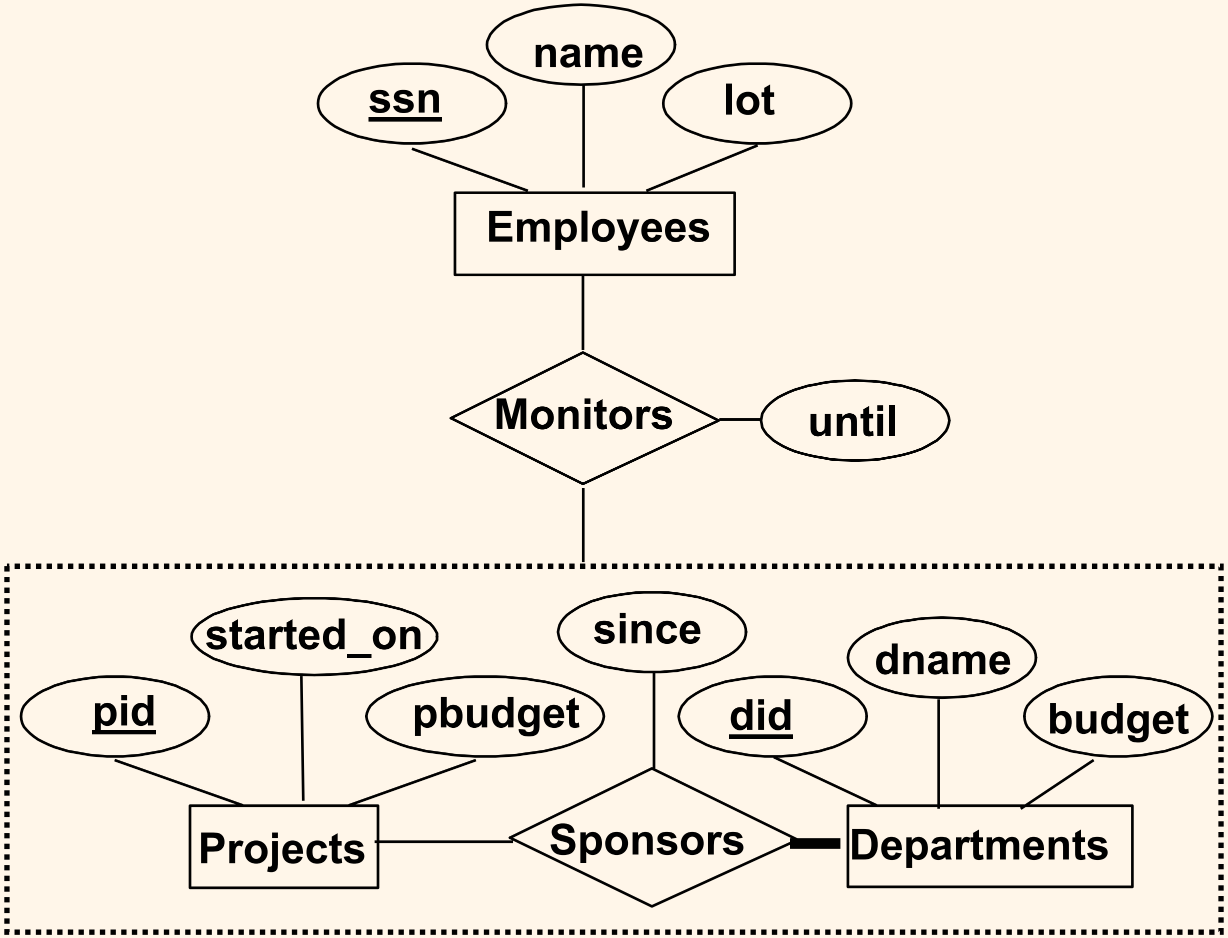
ISA (`is a') Hierarchies



- 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 entitities that participate in a relationship.

Aggregation

- Used when we have to model a relationship involving (entitity 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.

Conceptual Design Using the ER Model

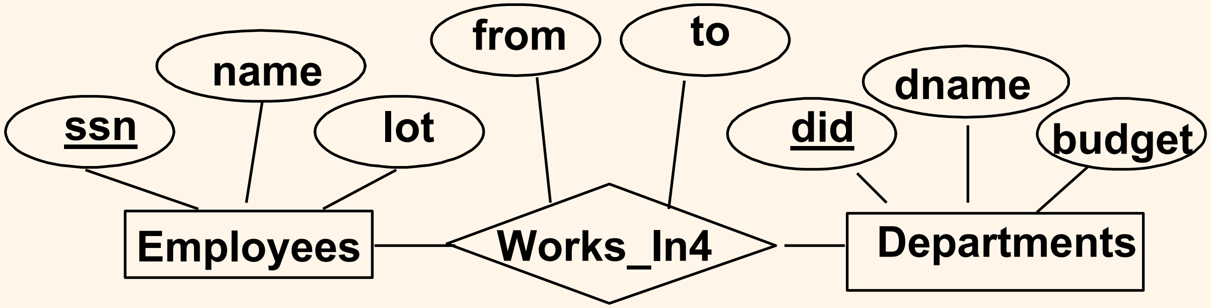
- Design choices:
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships: Binary or ternary? Aggregation?
- Constraints in the ER Model:
  - A lot of data semantics can (and should) be captured.
  - But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

Entity vs. Attribute

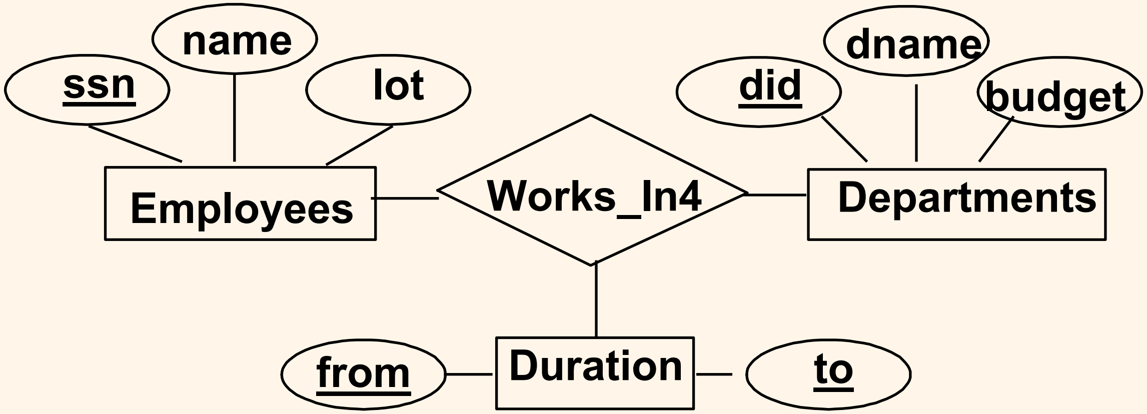
- Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
- Depends upon the use we want to make of address information, and the semantics of the data:
  - If we have several addresses per employee, *address* must be an entity (since attributes cannot be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

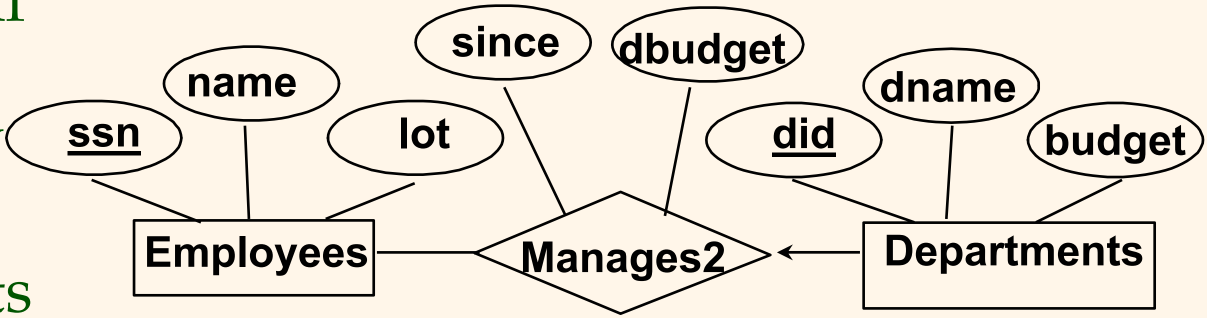


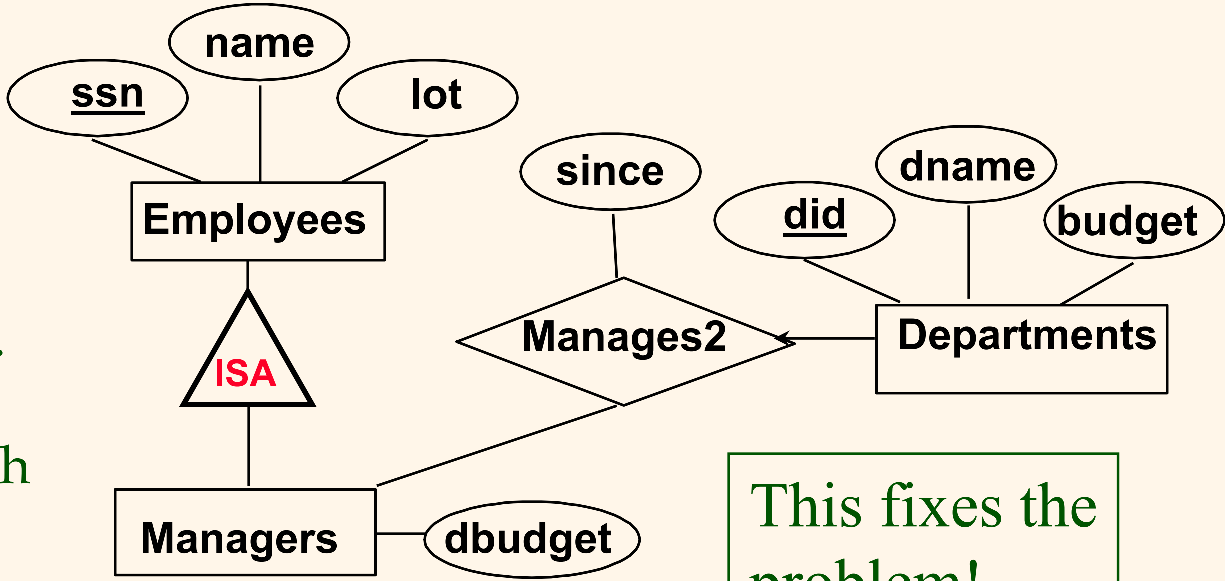
- Similar to the problem of wanting to record several addresses for an employee: We want to record *several values of the descriptive attributes for each instance of this relationship.* Accomplished by introducing new entity set, Duration.



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.

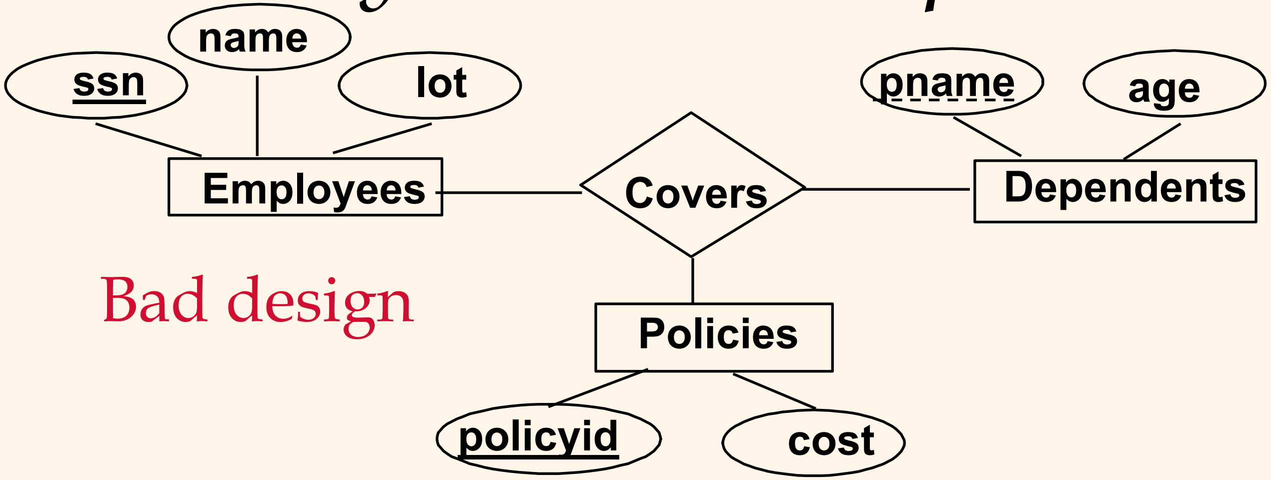




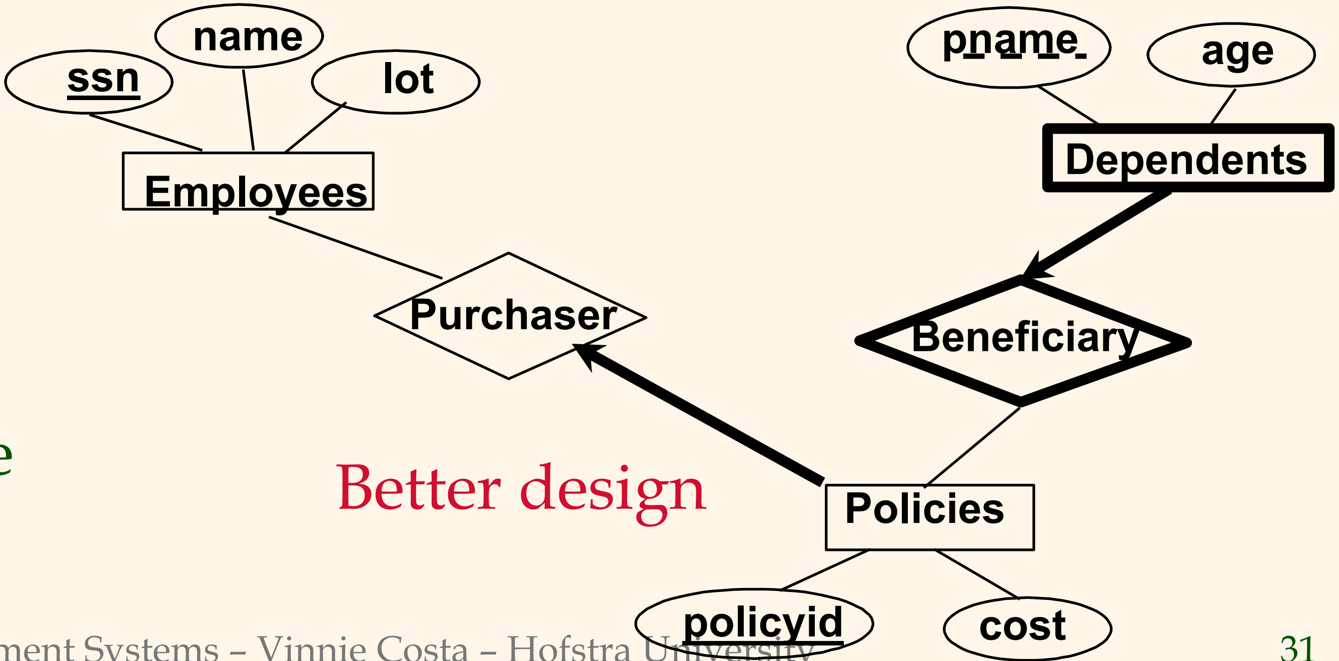
This fixes the problem!

Binary vs. Ternary Relationships

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



Bad design



Better design

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 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.

Useful Websites

- http://www.omg.org/ - information about UML
- Edgar (Ted) Codd – biographical sketch
- Modeling Tools – good list of available tools – checkout: DIA, ERwin, DBDesigner4, SmartDraw

Homework

- Read Chapter Two
- Exercises p.52: 2.1, 2.2 (1-5)

Practicum

- Install Apache
- Install Nvu
- …on our way to WAMP!!!

Apache

- The Apache Software Foundation
  http://www.apache.org/

- httpd.apache.org

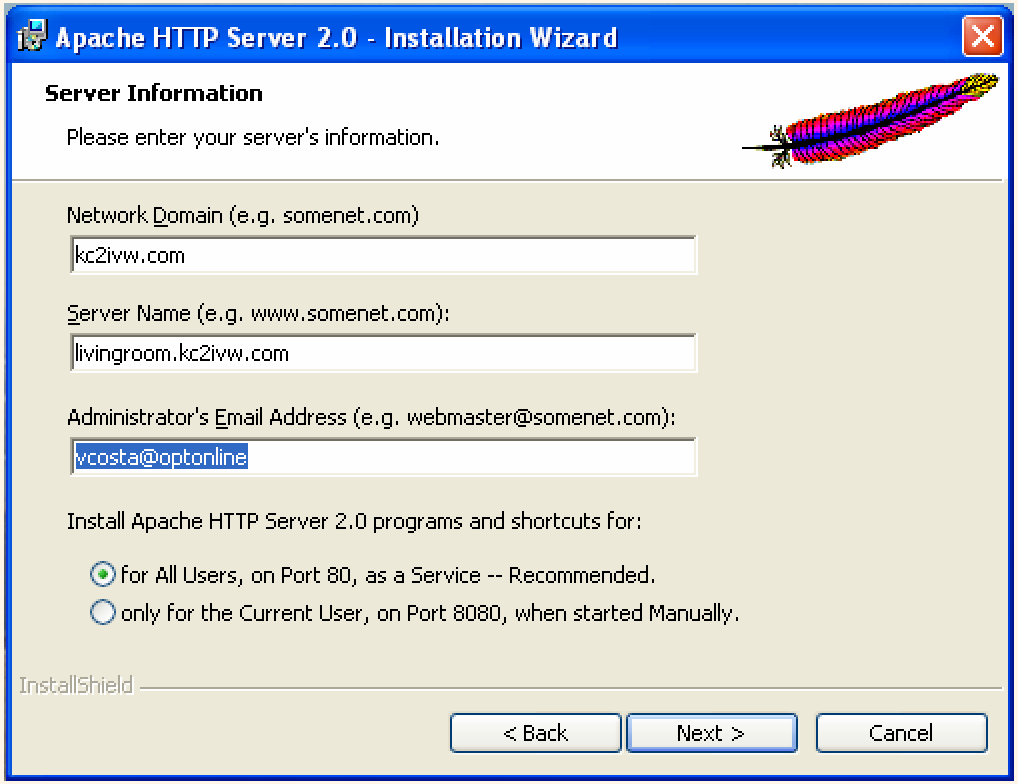
- The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards.

- Apache has been the **most popular web server on the Internet since April of 1996**. More than 68% of the web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined.

Install Apache

- http://httpd.apache.org/docs/2.0/platform/windows.html
- Installing apache is easy if you download the Microsoft Installer (.msi) package. Just double click on the icon to run the installation wizard. Click next until you see the Server Information window. You can enter localhost for both the Network Domain and Server Name. As for the administrator's email address you can enter anything you want.
- If using Windows XP, installed Apache as Service so every time I start Windows Apache is automatically started.

Installing Apache



Apache HTTP Server 2.0 - Installation Wizard

Server Information

Please enter your server's information.

Network Domain (e.g. somenet.com)

kc2ivw.com

Server Name (e.g. www.somenet.com):

livingroom.kc2ivw.com

Administrator's Email Address (e.g. webmaster@somenet.com):

vcosta@optonline

Install Apache HTTP Server 2.0 programs and shortcuts for:

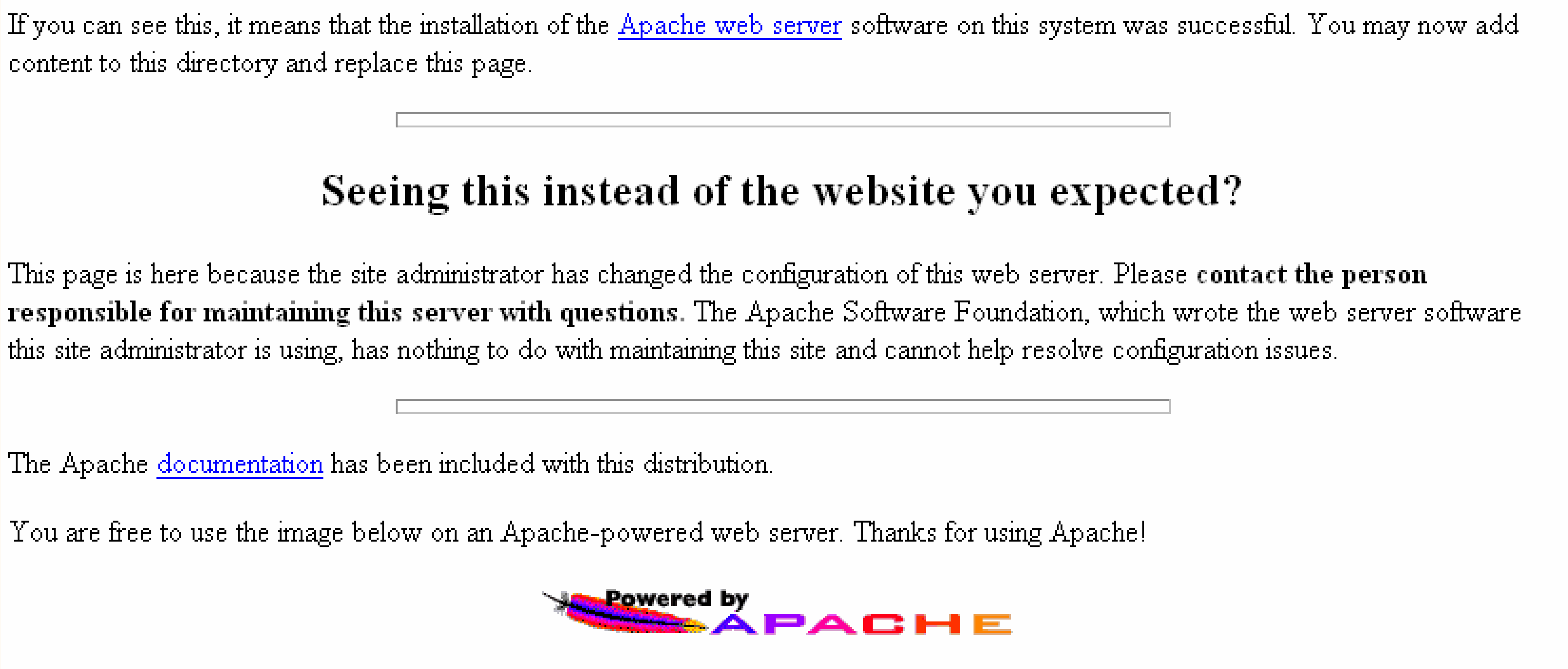
● for All Users, on Port 80, as a Service -- Recommended.

○ only for the Current User, on Port 8080, when started Manually.

- Click the **Next** button and choose **Typical installation**. Click Next one more time and choose where you want to install Apache (I installed it in the default location C:\Program Files\Apache Group). Click the Next button and then the Install button to complete the installation process.

Installing Apache

- To see if you Apache installation was successful open up you browser and type http://localhost (or http://127.0.0.1) in the address bar. You should see something like this :



If you can see this, it means that the installation of the Apache web server software on this system was successful. You may now add content to this directory and replace this page.

Seeing this instead of the website you expected?

This page is here because the site administrator has changed the configuration of this web server. Please **contact the person responsible for maintaining this server with questions.** The Apache Software Foundation, which wrote the web server software this site administrator is using, has nothing to do with maintaining this site and cannot help resolve configuration issues.

The Apache documentation has been included with this distribution.

You are free to use the image below on an Apache-powered web server. Thanks for using Apache!

Installing Apache

- By default Apache's **document root** is set to **htdocs** directory. The document root is where you must put all your PHP or HTML files so it will be process by Apache (and can be seen through a web browser). Of course you can change it to point to any directory you want. The configuration file for Apache is stored in `C:\Program Files\Apache Group\Apache2\conf\httpd.conf` (assuming you installed Apache in `C:\Program Files\Apache Group`) . It's just a plain text file so you can use Notepad to edit it.
- For example, if you want to put all your PHP or HTML files in `C:\www` just find this line in the `httpd.conf` :

  ```
  DocumentRoot "C:/Program Files/Apache Group/Apache2/htdocs"
  ```

  and change it to :

  ```
  DocumentRoot "C:/www"
  ```
- After making changes to the configuration file you have to restart Apache (Start > Programs > Apache HTTP Server 2.0 > Control Apache Server > Restart) to see the effect.

Installing Apache

- Another configuration you may want to change is the **directory index.** This is the file that Apache will show when you request a directory. As an example if you type `http://www.php-mysql-tutorial.com/` without specifying any file the `index.php` file will be automatically shown.
- Suppose you want apache to use `index.html, index.php or main.php` as the directory index you can modify the DirectoryIndex value like this :

  ```
  DirectoryIndex index.html index.php main.php
  ```
- Now whenever you request a directory such as `http://localhost/` Apache will try to find the `index.html` file or if it's not found Apache will use `index.php`. In case `index.php` is also not found then `main.php` will be used.

Installing Nvu

- www.nvu.com/
- A complete Web Authoring System for Linux Desktop users as well as Microsoft Windows and Macintosh users to rival programs like FrontPage and Dreamweaver.
- **Nvu** (pronounced N-view, for a "new view") makes managing a web site a snap. Now anyone can create web pages and manage a website with no technical expertise or knowledge of HTML.

Make A Home Page

- Create an `index.html` page with Nvu
- Copy `C:\Program Files\Apache Group\Apache2\htdocs` to `old_htdocs`
- Put the `index.html` into `htdocs`
- Test with `http://localhost` or `http://127.0.0.1`
- Explore Cascading Style Sheets (CSS)

Useful Websites

- www.w3.org/Style/CSS/ - the authoritative source
- http://www.w3.org/Style/Examples/011/firstcss – *Starting with HTML + CSS* – good beginners guide
- www.csszengarden.com – A demonstration of what can be accomplished visually through CSS-based design

Homework

- Install Apache On Your System
- Install Nvu
- Create your own home page
- Play with HTML
- Play with CSS
- Play, play, play, …

The Relational Model

Chapter 3

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

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.
    - 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

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"
```

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

Given the following instances of Enrolled and 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

we get:

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."

- 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) )
```

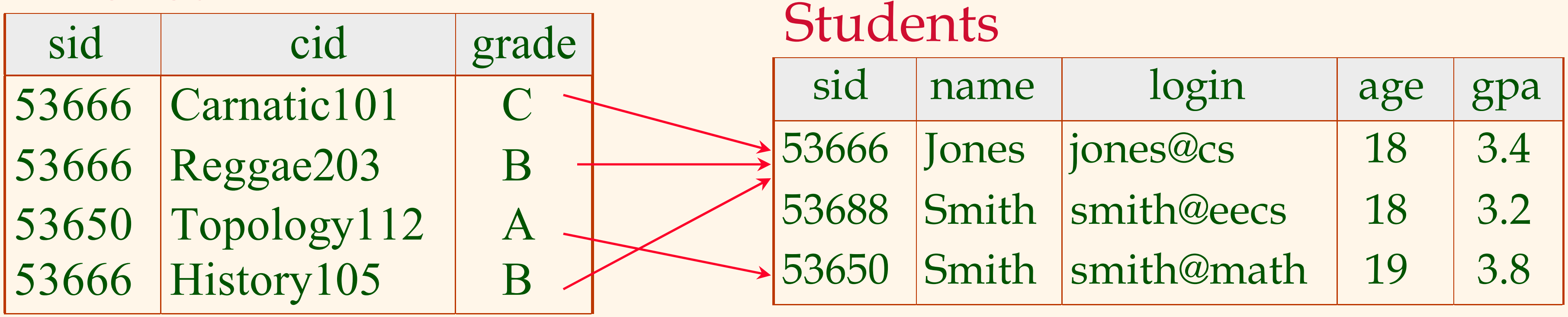
Foreign Keys, Referential Integrity

- *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

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)

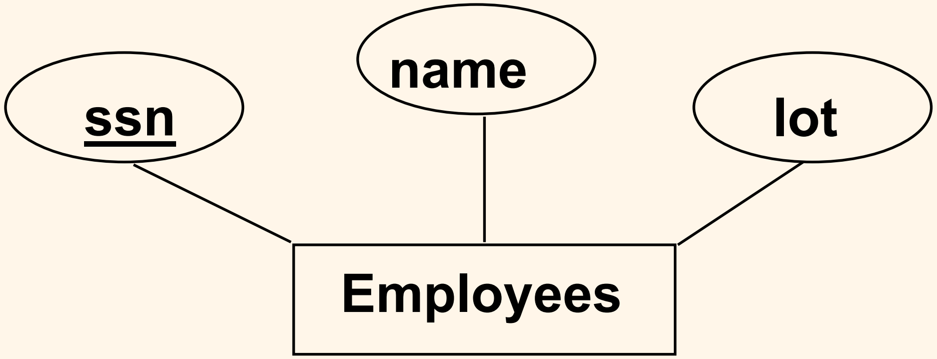
```
CREATE TABLE Enrolled
   (sid CHAR(20),
    cid CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid,cid),
    FOREIGN KEY (sid)
      REFERENCES Students
           ON DELETE CASCADE
           ON UPDATE SET DEFAULT )
```

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))
```

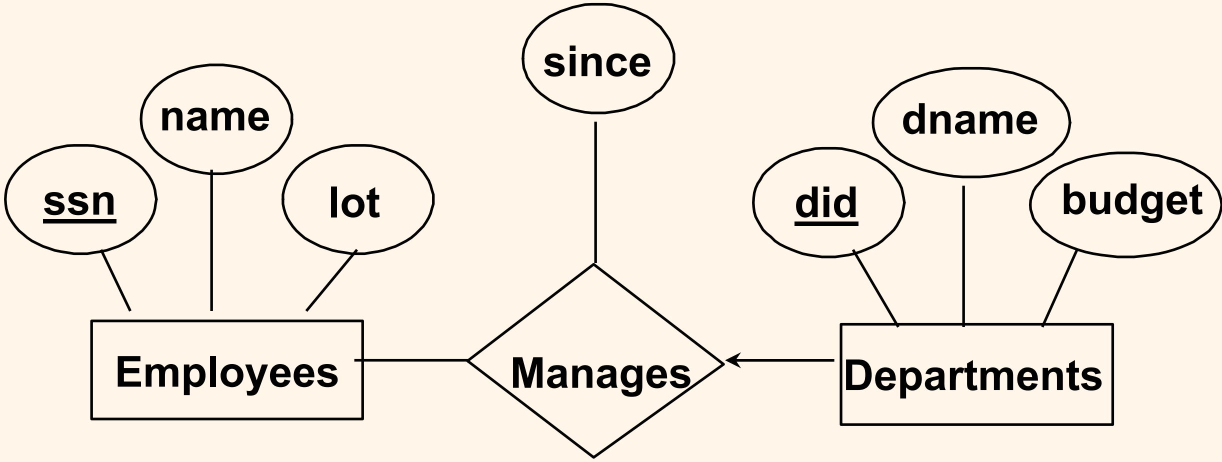
Relationship Sets to Tables

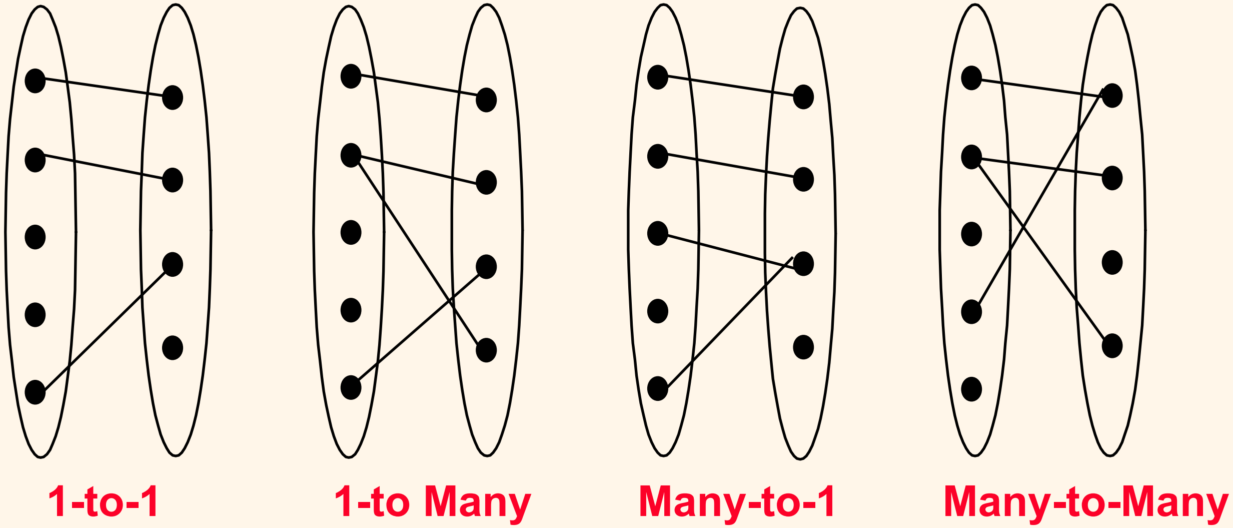
- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys).
    - This set of attributes forms a *superkey* for the relation.
  - All descriptive attributes.

```
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.




1-to-1

1-to Many

Many-to-1

Many-to-Many

Translation to relational model?

Translating ER Diagrams with Key Constraints

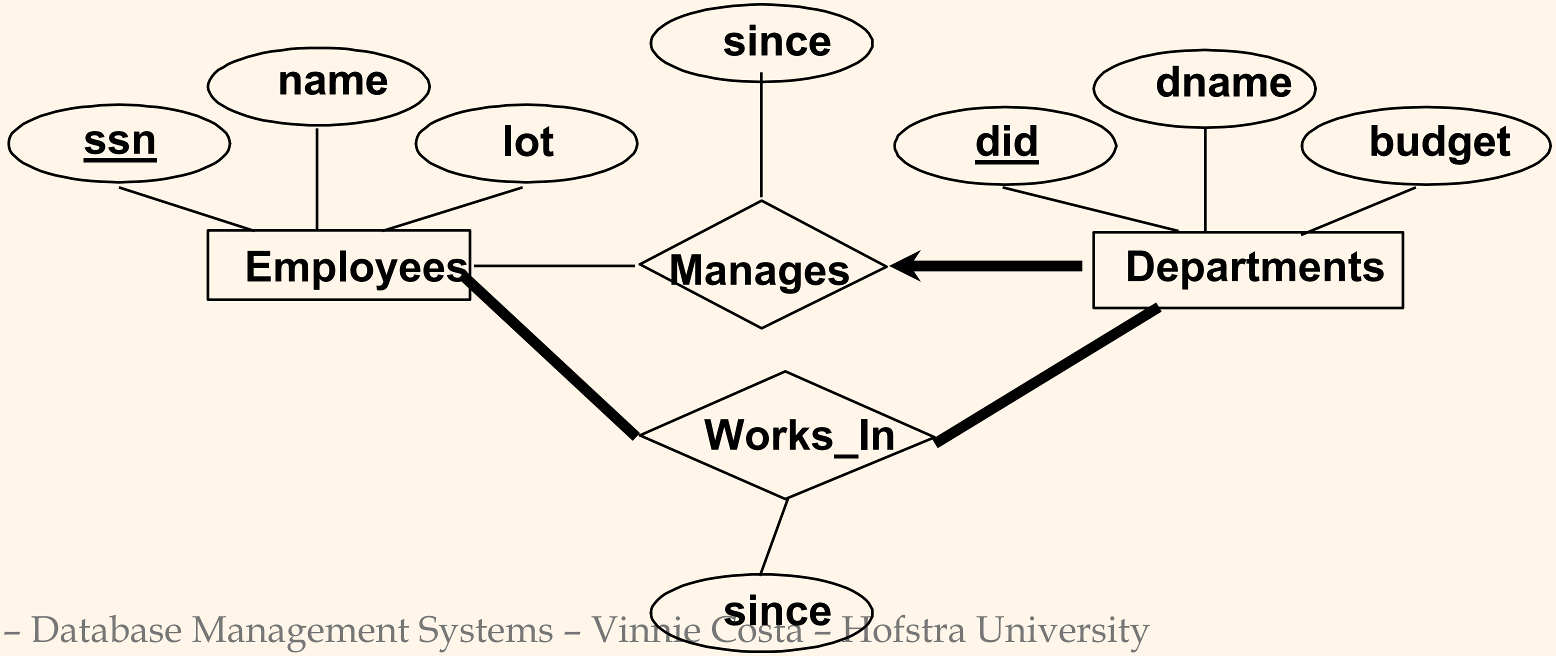
- Map relationship to a table:
  - Note that did is the key now!
  - Separate tables for Employees and Departments.
- Since each department has a unique manager, we could instead combine Manages and Departments.

```
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!)



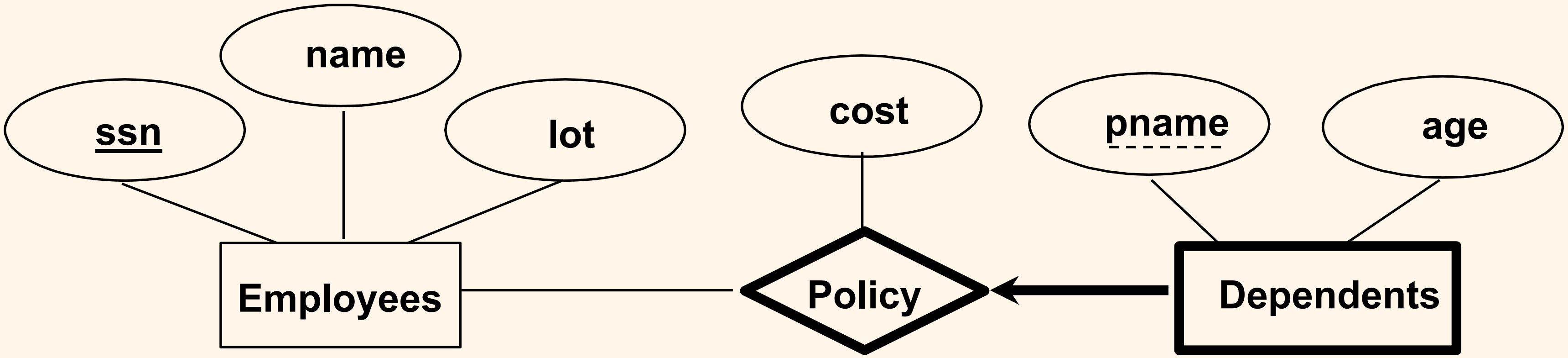
Participation Constraints in SQL

- 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.

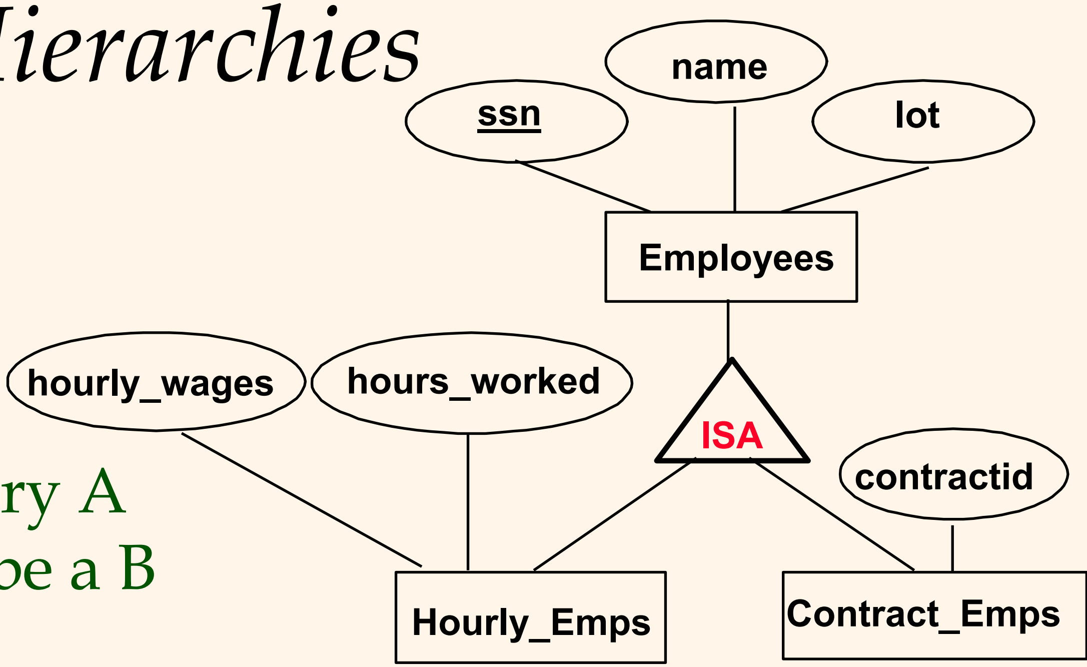


Translating Weak Entity Sets

- 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



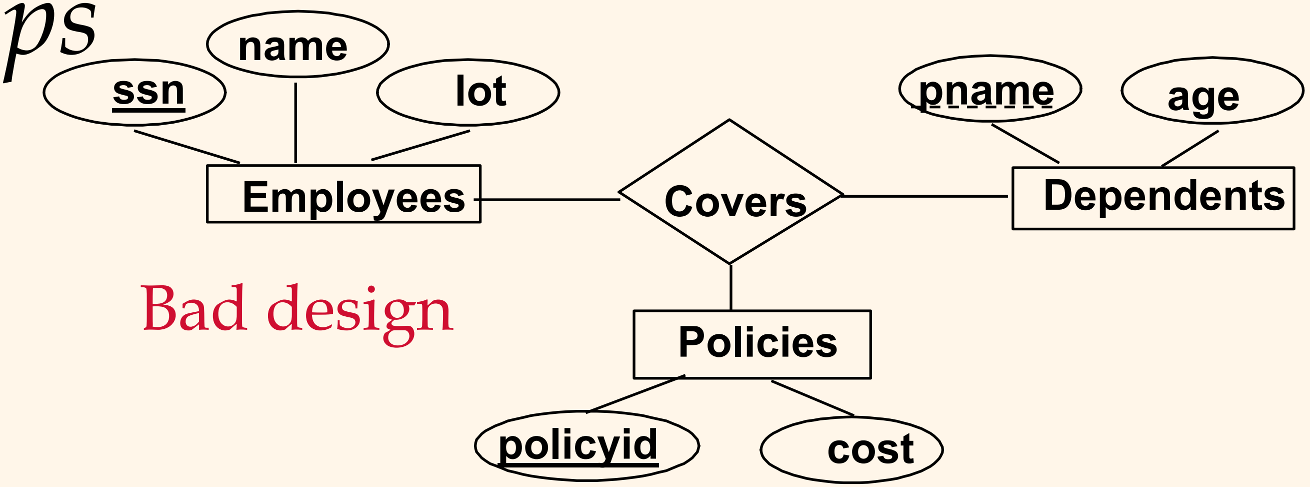
- As in C++, or other PLs, attributes are inherited.
- If we declare A **ISA** B, every A entity is also considered to be a B entity.

- *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

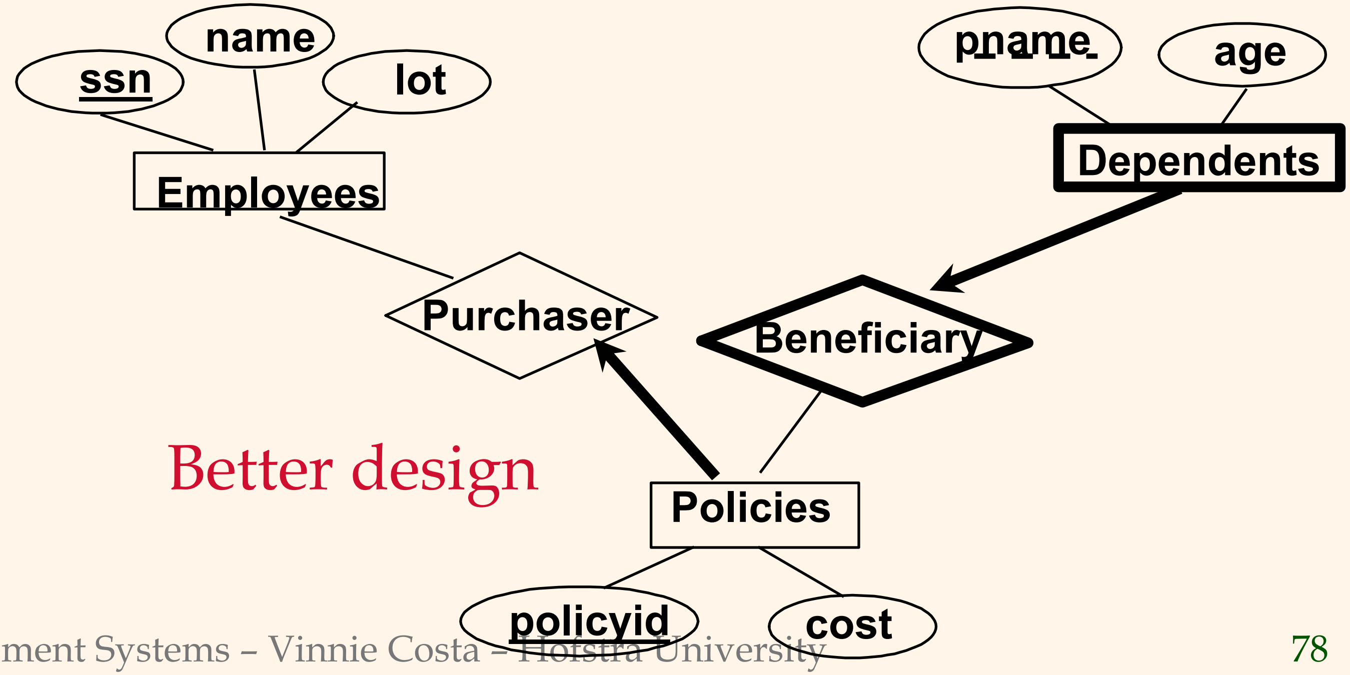
- ***General approach:***
  - 3 relations: Employees, Hourly_Emps and Contract_Emps.
    - *Hourly_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages, hours_worked, ssn*); must delete Hourly_Emps tuple if referenced Employees tuple is deleted).
    - 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



Bad design

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



Better design

Binary vs. Ternary Relationships (Contd.)

- The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.
- Participation constraints lead to NOT NULL constraints.
- What if Policies is a weak entity set?

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

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

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