Entity-Relationship Diagrams and the Relational Model

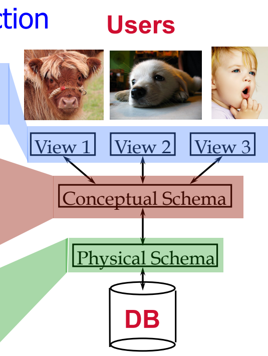
R & G, Chaps. 2&3

Describing Data: Data Models

- *Data model* : collection of concepts for describing data.
- *Schema:* description of a particular collection of data, using a given data model.
- *Relational model of data*
  - Main concept: *relation* (table), rows and columns
  - Every relation has a *schema*
    - describes the columns
    - column names and domains

Levels of Abstraction

- Views describe how users see the data.
- Conceptual schema defines logical structure
- Physical schema describes the files and indexes used.



Example: University Database

- Conceptual schema:
```
– Students(sid text, name text,
           login text, age integer,
           gpa float)
– Courses(cid text, cname text,
          credits integer)
– Enrolled(sid text, cid text,
           grade text)
```
- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.
- External Schema (View):
```
– Course_info(cid text,
              enrollment integer)
```

Data Independence

- Insulate apps from structure of data
- Logical data independence:
  - Protection from changes in *logical* structure
- Physical data independence:
  - Protection from changes in *physical* structure
- Q: Why particularly important for DBMS?

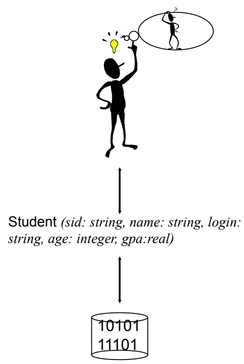
Because databases and their associated applications persist.

Hellerstein's Inequality

$$\frac{dapp}{dt} << \frac{denv}{dt}$$

Data Models

- Connect concepts to bits!
- Many models exist
- We will ground ourselves in the *Relational* model
  - clean and common
  - generalization of key/value
- *Entity-Relationship* model also handy for design
  - Translates down to Relational



Why Focus on Relational Model?

- Most widely used model.
- Other models exist (and co-exist)
  - "Legacy systems" in older models
    - e.g., IBM's IMS
  - Object-Relational mergers
    - Object-Oriented features provided by DBMS
    - Object-Relational Mapping (ORM) outside the DBMS
      - A la Rails (Ruby), Django (Python), Hibernate (Java)
  - XML, JSON, etc.
    - Nested and "semi-structured" data
    - Languages like Xquery, XSLT, JSONic
    - Most relational engines now handle these to a degree

Entity-Relationship Model

- Relational model is a great formalism
  - and a clean system framework
- But a bit detailed for design time
  - a bit fussy for brainstorming
  - hard to communicate to customers
- Entity-Relationship model is a popular "shim" over relational model
  - graphical, slightly higher level

Steps in Traditional Database Design

- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - high level description (often done w/ER model)
  - *Rails encourages you to program here*
- Logical Design
  - translate ER into DBMS data model
  - *Rails requires you to help here too*
- Schema Refinement
  - consistency, normalization
- Physical Design - indexes, disk layout
- Security Design - who accesses what, and how

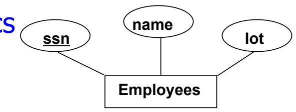
Conceptual Design

- What are the entities and relationships?
- What info about E's & R's should be in DB?
- What *integrity constraints* (*business rules*) hold?
- *ER diagram* is the "schema"
- Can map an ER diagram into a relational schema.
- Conceptual design is where the SW/data engineering *begins*
  - Rails "models"

Modern pattern: "Schema on Use"

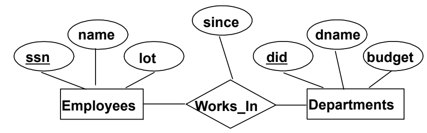
- What about more agile, less governed environments?
- Don't let the lack of schema prevent storing data!
  - Just use binary, text, CSV, JSON, xlsx, etc.
  - Can shove into a DBMS, or just a filesystem (e.g. HDFS)
  - Most database engines can query files directly these days
- Wrangle the data into shape as needed
  - Essentially defining views over the raw data
  - This amounts to database design, at the view level
  - What about integrity constraints?
    - Instead, define "anomaly indicator" columns – or queries
- Fits well with read/append-only data
  - E.g. Big Data, a la Hadoop
  - Less of a fit with update-heavy data
- Analogies to strong vs. loose typing in PL

ER Model Basics



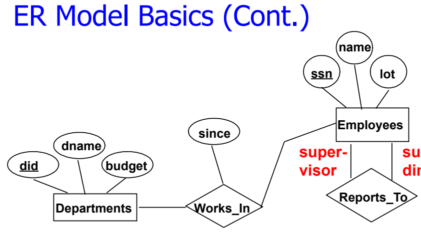
- *Entity:*
  - A real-world object described by a set of *attribute* values.
- *Entity Set*: A collection of similar entities.
  - E.g., all employees.
  - All entities in an entity set have the same attributes.
  - Each entity set has a *key* (underlined)
  - Each attribute has a *domain*

ER Model Basics (Contd.)



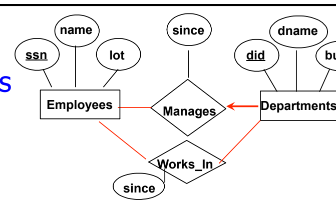
- *Relationship*: Association among two or more entities.
  - E.g., Attishoo works in Pharmacy department.
  - relationships can have their own attributes.
- *Relationship Set*: Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets E_1 ... E_n; each relationship in R involves entities $e_1 \in E_1$, ..., $e_n \in E_n$

ER Model Basics (Cont.)



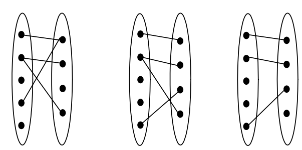
- Same entity set can participate in different relationship sets, or in different "roles" in the same relationship set.

Key Constraints



An employee can work in many departments; a dept can have many employees.

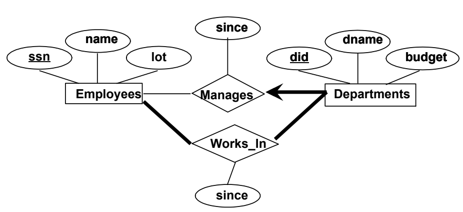
In contrast, each dept has at most one manager, according to the *key constraint* on Manages.



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

Participation Constraints

- Does every employee work in a department?
- If so: a *participation constraint*
  - participation of Employees in Works_In is *total* (vs. *partial*)
  - What if every department has an employee working in it?
- Basically means "at least one"



Alternative: Crow's Foot Notation

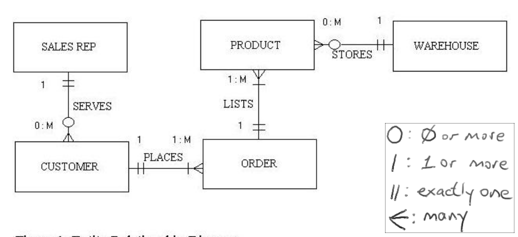


Figure 1. Entity-Relationship Diagram

- 1 INSTANCE OF A SALES REP SERVES 1 TO MANY CUSTOMERS
- 1 INSTANCE OF A CUSTOMER PLACES 1 TO MANY ORDERS
- 1 INSTANCE OF AN ORDER LISTS 1 TO MANY PRODUCTS
- 1 INSTANCE OF A WAREHOUSE STORES 0 TO MANY PRODUCTS

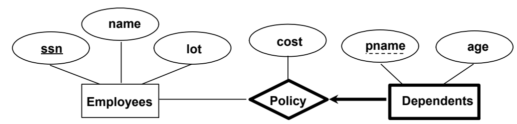
Summary so far

- Entities and Entity Set (boxes)
- Relationships and Relationship sets (diamonds)
- Key constraints (arrows)
- Participation constraints (bold for Total)

These are enough to get started, but we'll need more…

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.



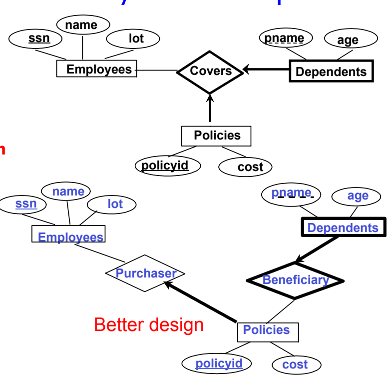
Weak entities have only a "partial key" (dashed underline)

Binary vs. Ternary Relationships

If each policy is owned by just 1 employee:

Key constraint on Policies would mean policy can only cover 1 dependent!

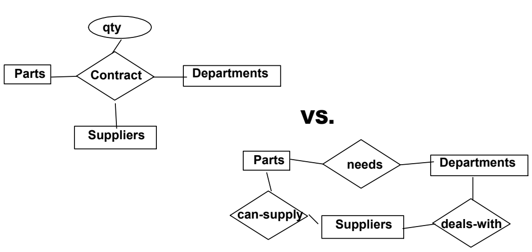
- **Think through *all* the constraints in the 2nd diagram!**



Binary vs. Ternary Relationships (Contd.)

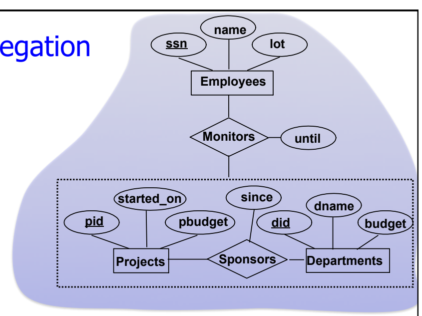
- Previous example:
  - 2 binary relationships better than 1 ternary relationship.
- An example in the other direction:
  - ternary relationship set Contracts relates entity sets Parts, Departments and Suppliers
  - relationship set has descriptive attribute *qty*.
  - no combo of binary relationships is a substitute!
    - See next slide…

Binary vs. Ternary Relationships (Contd.)



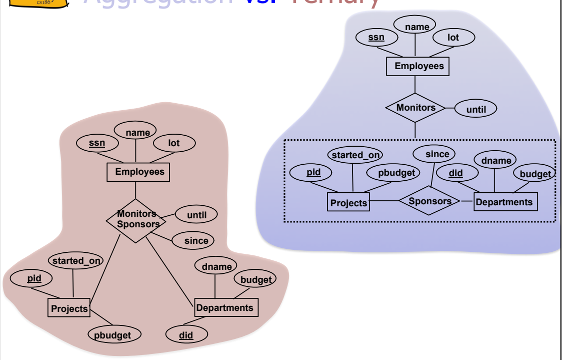
- S "can-supply" P, D "needs" P, and D "deals-with" S does not imply that D has agreed to buy P from S.
- How do we record *qty*?

Aggregation



Allows relationships with *relationship sets*.

Aggregation vs. Ternary



Conceptual Design Using the ER Model

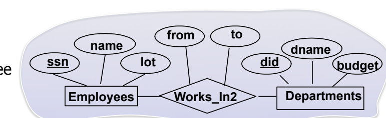
- ER modeling *can* get tricky!
- Design choices:
  - Entity or attribute?
  - Entity or relationship?
  - Relationships: Binary or ternary? Aggregation?
- ER Model goals and limitations:
  - Lots of semantics can (and should) be captured.
  - Some constraints *cannot* be captured in ER.
    - We'll refine things in our logical (relational) design

Entity vs. Attribute

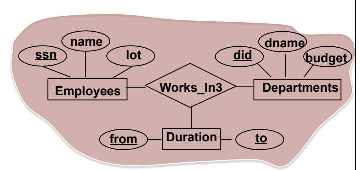
- "Address":
  - attribute of Employees?
  - Entity of its own?
- It depends! Semantics and usage.
  - Several addresses per employee?
    - must be an entity
    - atomic attribute types (no set-valued attributes!)
  - Care about structure? (city, street, etc.)
    - must be an entity!
    - atomic attribute types (no tuple-valued attributes!)

Entity vs. Attribute (Cont.)

- Works_In2: employee cannot work in a department for >1 period.

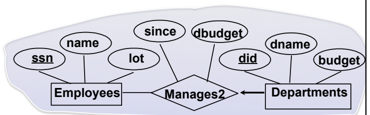


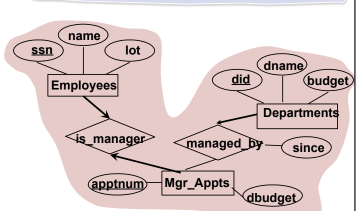
- Like multiple addresses per employee!



Entity vs. Relationship

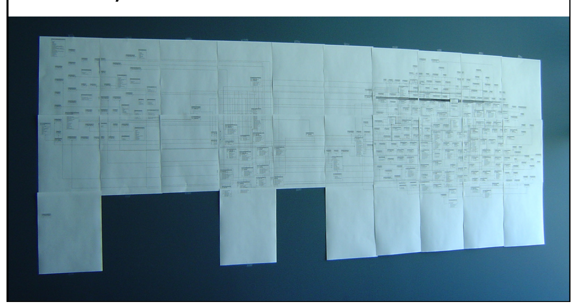
- Separate discretionary budget (dbudget) for each dept.
- What if manager's dbudget covers all managed depts
  - Could repeat value
  - But redundancy = problems
- Better design:





E-R Diagram as Wallpaper

- Very common for them to be wall-sized

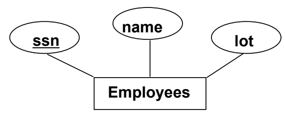


Converting ER to Relational

- Fairly analogous structure
- But many simple concepts in ER are subtle to specify in relations

Logical DB Design: ER to Relational

- Entity sets to tables.



ssn	name	lot
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

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

Relationship Sets to Tables

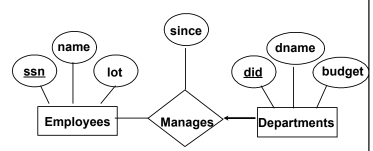
- In translating a many-to-many relationship set to a relation, attributes of the relation must include:
  1) Keys for each participating entity set (as foreign keys). This set of attributes forms a *superkey* for the relation.
  2) All descriptive attributes.

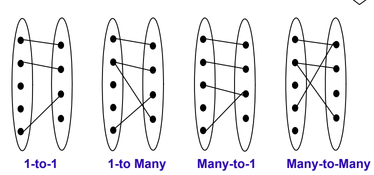
```
CREATE TABLE Works_In(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
 PRIMARY KEY (ssn, did),
 FOREIGN KEY (ssn)
    REFERENCES Employees,
 FOREIGN KEY (did)
  REFERENCES Departments)
```

ssn	did	since
123-22-3666	51	1/1/91
123-22-3666	56	3/3/93
231-31-5368	51	2/2/92

Review: Key Constraints

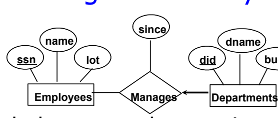
- Each dept has at most one manager, according to the *key constraint* on Manages.





Translation to relational model?

Translating ER with Key Constraints



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

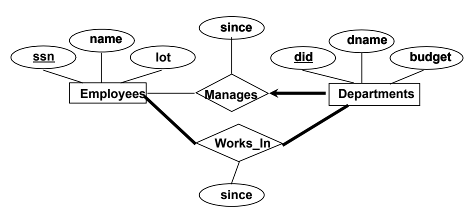
```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
REFERENCES Employees,
   FOREIGN KEY (did)
REFERENCES Departments)
```

Vs.

```
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!)



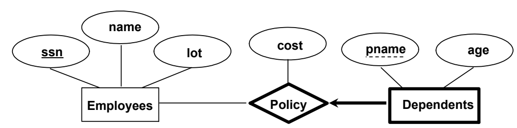
Participation Constraints in SQL

- 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)
```

Summary of Conceptual Design

- *Conceptual design* follows *requirements analysis*,
  - Yields a high-level description of data to be stored
  - You may want to postpone it for read-only "schema on use"
- ER model popular for conceptual design
  - Constructs are expressive, close to the way people think about their applications.
  - Note: There are many variations on ER model
    - Both graphically and conceptually
- Basic constructs: *entities, relationships*, and *attributes* (of entities and relationships).
- Some additional constructs: *weak entities, ISA hierarchies* (see text if you're curious), and *aggregation*.

Summary of ER (Cont.)

- Several kinds of integrity constraints:
  - *key constraints*
  - *participation constraints*
- Some *foreign key constraints* are also implicit in the definition of a relationship set.
- Many other constraints (notably, *functional dependencies*) cannot be expressed.
- Constraints play an important role in determining the best database design for an enterprise.

Summary of ER (Cont.)

- 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, aggregation.
- Ensuring good database design: resulting relational schema should be analyzed and refined further.
  - Functional Dependency information and normalization techniques are especially useful.