Entity-Relationship Diagrams and the Relational Model

CS 186, Fall 2007, Lecture 2
R & G, Chaps. 2&3

A relationship, I think, is like a shark, you know? It has to constantly move forward or it dies. And I think what we got on our hands is a dead shark.

Woody Allen (from Annie Hall, 1979)

Review

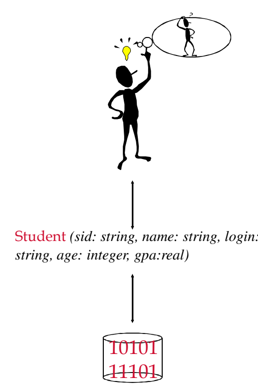
- Why use a DBMS? OS provides RAM and disk

Review

- Why use a DBMS? OS provides RAM and disk
  - Concurrency
  - Recovery
  - Abstraction, Data Independence
  - Query Languages
  - Efficiency (for most tasks)
  - Security
  - Data Integrity

Data Models

- DBMS models real world
- *Data Model* is link between user's view of the world and bits stored in computer
- Many models exist
- We will ground ourselves in the Relational Model
  - clean and common
- But use the Entity-Relationship model as a middle ground for design



Why Study the Relational Model?

- Most widely used model.
- "Legacy systems" in older models
  - e.g., IBM's IMS
- Object-oriented concepts merged in
  - "Object-Relational" – two variants
    - Object model known to the DBMS
    - Object-Relational Mapping (ORM) outside the DBMS
      - A la Rails
- XML features in most relational systems
  - Can export XML interfaces
  - Can provide XML storage/retrieval

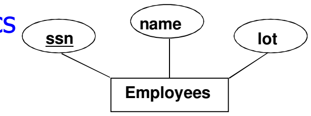
Steps in Database Design

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

Conceptual Design

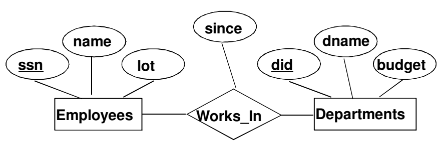
- What are the *entities* and *relationships* in the enterprise?
- What information about these entities and relationships should we store in the database?
- What *integrity constraints* or *business rules* hold?
- A database `schema' in the ER Model can be represented pictorially (*ER diagrams*).
-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

ER Model Basics



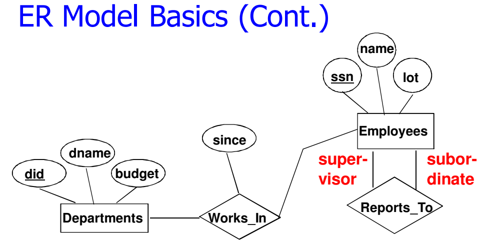
- *Entity:* Real-world object, distinguishable from other objects. An entity is described using a set of *attributes*.
- *Entity Set*: A collection of similar entities. E.g., all employees.
  - All entities in an entity set have the same set of attributes. (Until we consider hierarchies, anyway!)
  - Each entity set has a *key (underlined)*.
  - Each attribute has a *domain*.

ER Model Basics (Contd.)



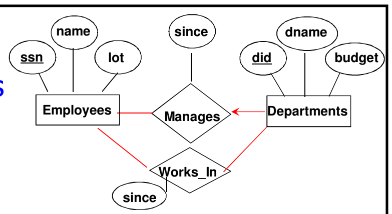
- *Relationship*: Association among two or more entities. E.g., Attishoo works in Pharmacy department.
  - relationships can have their own attributes.
- *Relationship Set*: Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets $E_1 \ldots E_n$; each relationship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

ER Model Basics (Cont.)



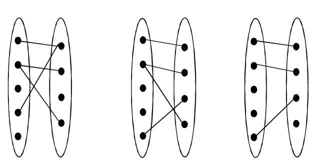
- Same entity set can participate in different relationship sets, or in different "roles" in the same set.

Key Constraints



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

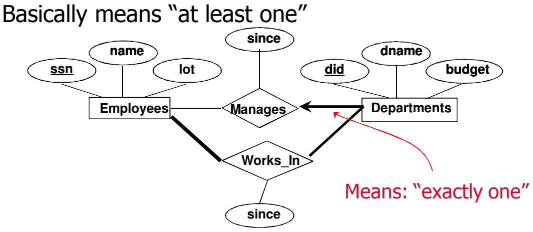
In contrast, each dept has at most one manager, according to the *key constraint* on Manages.



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

Participation Constraints

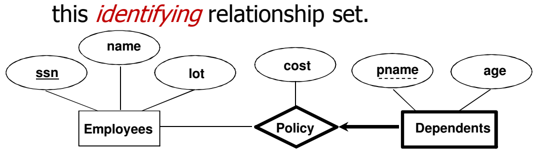
- Does every employee work in a department?
- If so, this is a *participation constraint*
  - the participation of Employees in Works_In is said to be *total* (vs. *partial*)
  - What if every department has an employee working in it?
- Basically means "at least one"



Weak Entities

A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.

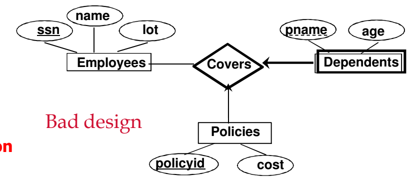
- Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
- Weak entity set must have total participation in this *identifying* relationship set.



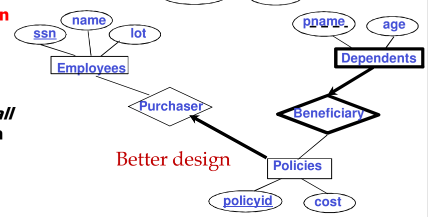
Weak entities have only a "partial key" (dashed underline)

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!

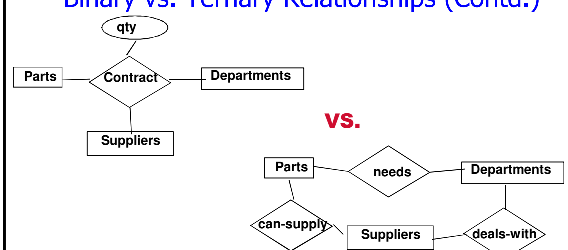


- **Think through *all* the constraints in the 2nd diagram!**

Binary vs. Ternary Relationships (Contd.)

- Previous example 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.

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*?

Summary so far

- Entities and Entity Set (boxes)
- Relationships and Relationship sets (diamonds)
  - binary
  - n-ary
- Key constraints (1-1,1-N, M-N, arrows)
- Participation constraints (bold for Total)
- Weak entities - require strong entity for key

Administrivia

- Blog online
- Syllabus & HW calendar coming on-line
  - Schedule and due dates may change (check frequently)
  - Lecture notes are/will be posted
- HW 0 posted -- due Friday night!
  - Accts forms!
- Other textbooks
  - Korth/Silberschatz/Sudarshan
  - O'Neil and O'Neil
  - Garcia-Molina/Ullman/Widom

Other Rails Resources

- Rails API: http://api.rubyonrails.org
- Online tutorials
  - E.g. http://poignantguide.net/ruby
  - Screencasts: http://www.rubyonrails.org/screencasts
  - Armando Fox's daylong seminar: http://webcast.berkeley.edu/event_details.php?webcastid=20854
- There are *tons* of support materials and fora on the web for RoR

Relational Database: Definitions

- Relational database: a set of relations.
- Relation: made up of 2 parts:
  - 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)
  - Instance : a table, with rows and columns.
    - #rows = cardinality
    - #fields = degree / arity
- Can think of a relation as a set of rows or tuples.
  - i.e., all rows are distinct

Ex: 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, arity = 5 , all rows distinct
- Do all values in each column of a relation instance have to be distinct?

SQL - A language for Relational DBs

- SQL (a.k.a. "Sequel"), standard language
- Data Definition Language (DDL)
  - create, modify, delete relations
  - specify constraints
  - administer users, security, etc.
- Data Manipulation Language (DML)
  - Specify queries to find tuples that satisfy criteria
  - add, modify, remove tuples

SQL Overview

- CREATE TABLE <name> (<field> <domain>, …)
- INSERT INTO <name> (<field names>)
  VALUES (<field values>)
- DELETE FROM <name>
  WHERE <condition>
- UPDATE <name>
  SET <field name> = <value>
  WHERE <condition>
- SELECT <fields>
  FROM <name>
  WHERE <condition>

Creating Relations in SQL

- Creates the Students relation.
  - Note: 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 FLOAT)
```

Table Creation (continued)

- Another example: the Enrolled table holds information about courses students take.

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

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!

Keys

- Keys are a way to associate tuples in different relations
- Keys are one form of integrity constraint (IC)

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

FOREIGN Key

PRIMARY Key

Primary Keys

- A set of fields is a superkey if:
  - No two distinct tuples can have same values in all key fields
- A set of fields is a key for a relation if :
  - It is a superkey
  - No subset of the fields is a superkey
- what if >1 key for a relation?
  - One of the keys is chosen (by DBA) to be the primary key. Other keys are called candidate keys.
- 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*.
- Keys must be used carefully!
- "For a given student and course, there is a single grade."

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

vs.

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

"Students can take only one course, and no two students in a course receive the same 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 the primary key of the other relation.
  - Like a `logical pointer'.
- If all foreign key constraints are enforced, *referential integrity* is achieved (i.e., no dangling references.)

Foreign Keys in SQL

- **E.g. Only students listed in the Students relation should be allowed to enroll for courses.**
  - *sid* is a foreign key referring to Students:

```
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
~~11111~~	~~English102~~	~~A~~

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 issues arise if primary key of Students tuple is updated.

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!

Where do ICs Come From?

- ICs are based upon the semantics of the real-world 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.

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

- The most widely used relational query language.
  - Current std is SQL:2003; SQL92 is a basic subset
- 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
53650	Smith	smith@math	19	3.8

- 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

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

we get:

S.name	E.cid
Smith	Topology112

Semantics of a Query

- A *conceptual evaluation method* for the previous query:
  1. do FROM clause: compute *cross-product* of Students and Enrolled
  2. do WHERE clause: Check conditions, discard tuples that fail
  3. do SELECT clause: Delete unwanted fields
- Remember, this is *conceptual*. Actual evaluation will be *much* more efficient, but must produce the same answers.

Cross-product of Students and Enrolled Instances

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
53666	Jones	jones@cs	18	3.4	53831	Carnatic101	C
53666	Jones	jones@cs	18	3.4	53832	Reggae203	B
53666	Jones	jones@cs	18	3.4	53650	Topology112	A
53666	Jones	jones@cs	18	3.4	53666	History105	B
53688	Smith	smith@ee	18	3.2	53831	Carnatic101	C
53688	Smith	smith@ee	18	3.2	53831	Reggae203	B
53688	Smith	smith@ee	18	3.2	53650	Topology112	A
53688	Smith	smith@ee	18	3.2	53666	History105	B
53650	Smith	smith@math	19	3.8	53831	Carnatic101	C
53650	Smith	smith@math	19	3.8	53831	Reggae203	B
53650	Smith	smith@math	19	3.8	**53650**	Topology112	**A**
53650	Smith	smith@math	19	3.8	53666	History105	B

Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used
  - Object-relational support in most products
  - XML support added in SQL:2003, most systems
- 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 query languages exist.
  - SQL is the standard commercial one
    - DDL - Data Definition Language
    - DML - Data Manipulation Language

Internet Moment

Databases for Programmers

- Programmers think about objects (structs)
  - Nested and interleaved
- Often want to "persist" these things
- Options
  - encode opaquely and store
  - translate to a structured form
    - relational DB, XML file
  - pros and cons?

Remember the Inequality!

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

- If storing indefinitely…use a flexible representation

But YUCK!!

- How do I "relationalize" my objects?
- Have to write a converter for each class?
- Think about when to save things into the DB?

- Good news:
  - Can all be automated
  - With varying amounts of trouble

Object-Relational Mappings

- Roughly:
  - Class ~ Entity Set
  - Instance ~ Entity
  - Data member ~ Attribute
  - Reference ~ Foreign Key

Details, details

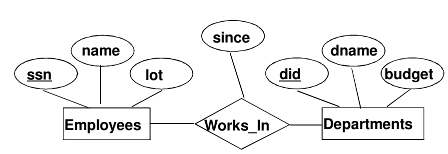
- We have to map this down to tables
- Which table holds which class of object?
- What about relationships?
- Solution #1: Declarative Configuration
  - Write a description file (often in XML)
    - E.g. Enterprise Java Beans (EJBs)
- Solution #2: Convention
  - Agree to use some conventions
    - E.g. Rails

Ruby on Rails

- Ruby: an OO scripting language
  - and a pretty nice one, too
- Rails: a framework for web apps
  - "convention over configuration"
    - great for standard web-app stuff!
  - allows overriding as needed
- Very ER-like

Rails and ER

- Models
  - Employees
  - Departments



Some Rails "Models"

app/models/state.rb

```
class State < ActiveRecord::Base
  has_many :cities
end
```

app/models/city.rb

```
class City < ActiveRecord::Base
  belongs_to :state
end
```

A More Complex Example

Further Reading

- Chapter 18 (through 18.3) in *Agile Web Development with Rails*