

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



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



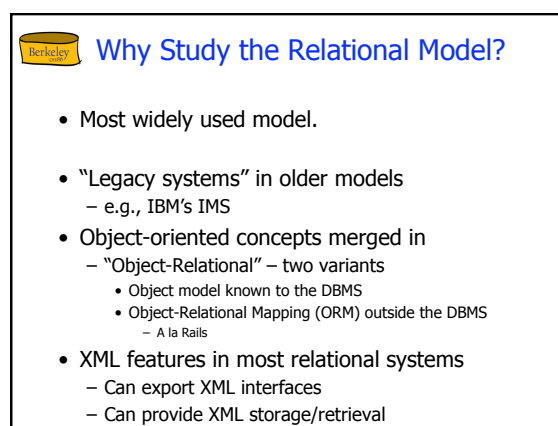
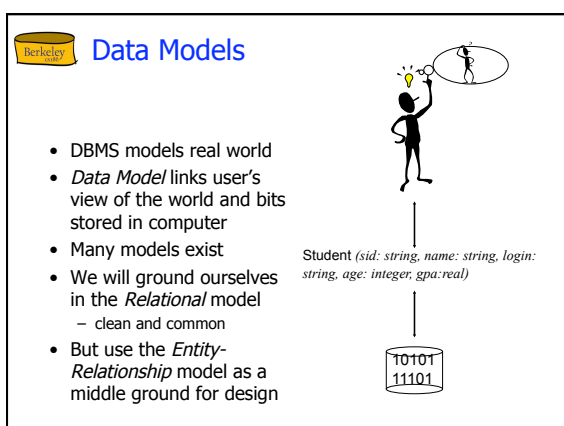
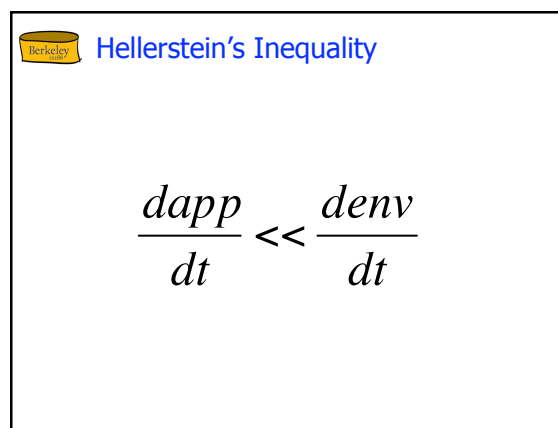
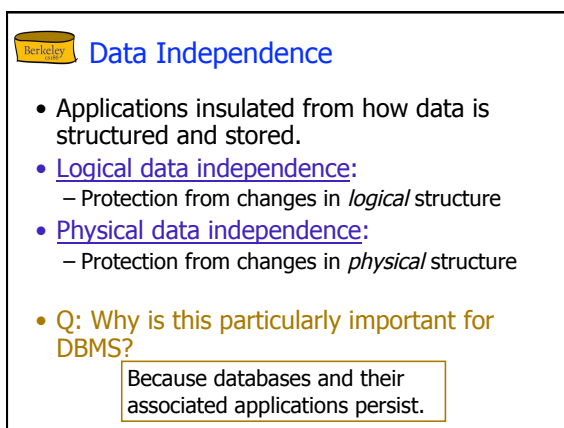
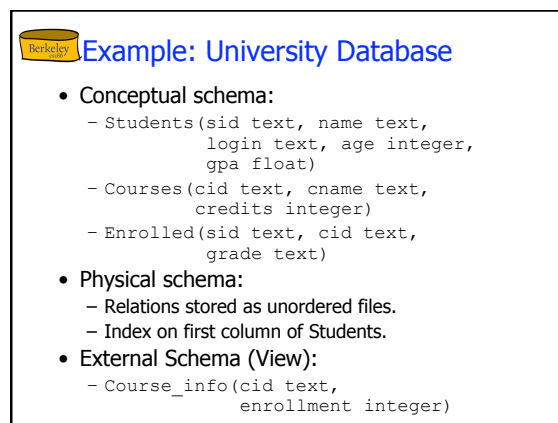
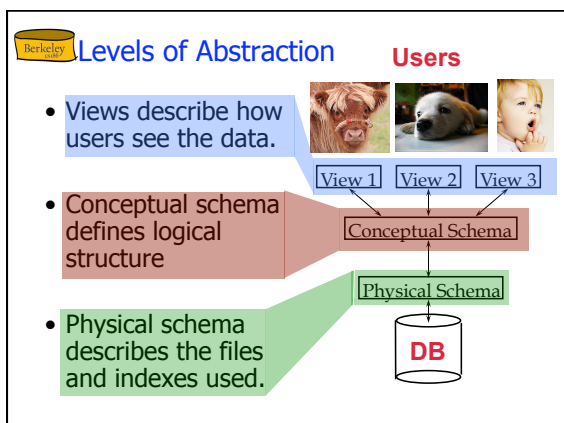
Some Synonyms

Formal	Not-so-formal 1	Not-so-formal 2
Relation	Table	
Tuple	Row	Record
Attribute	Column	Field



Example: University Database

- 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)





An Aside: Ruby on Rails

- Ruby
 - An object-oriented scripting language
- Rails
 - An open source web framework
 - *Active Record*
 - An *Object-Relational Mapping (ORM)*
 - Very similar to Entity-Relationship modeling
 - *Migrations*
 - A methodology for database evolution
 - A Model-View-Controller (MVC) design pattern for websites
 - *Convention over Configuration*



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 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

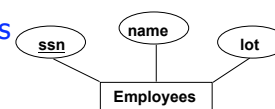


Conceptual Design

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



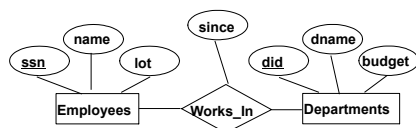
ER Model Basics



- **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*



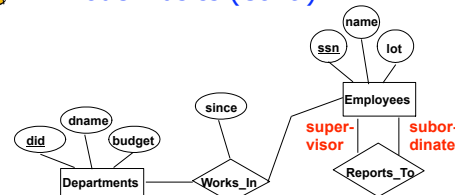
ER Model Basics (Contd.)



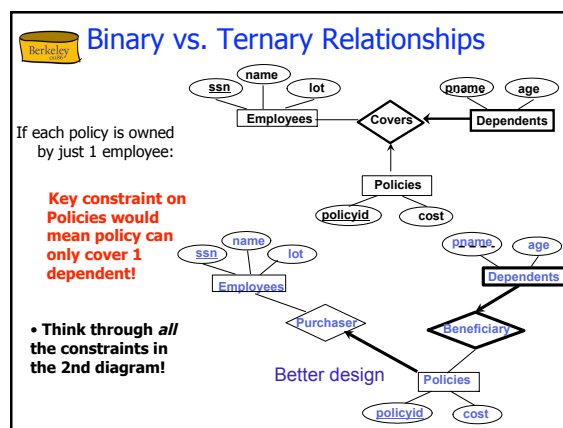
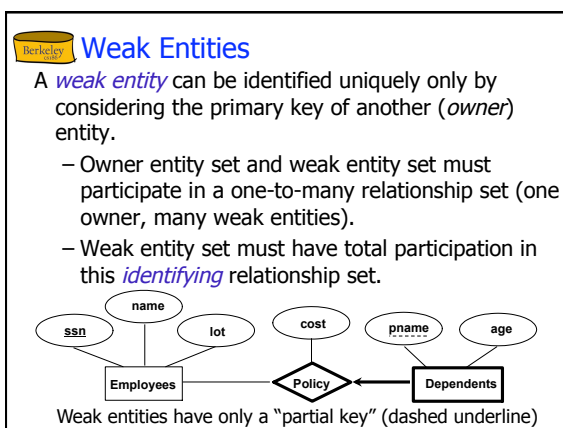
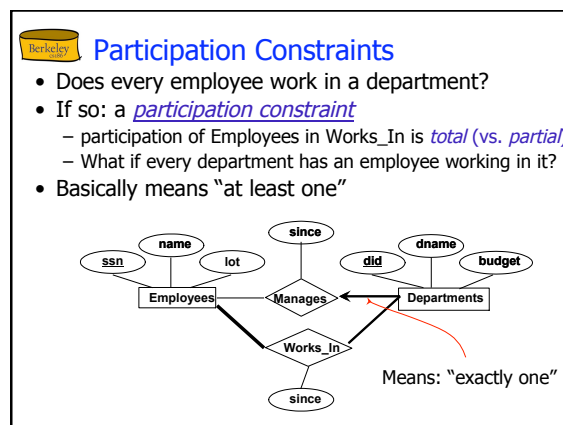
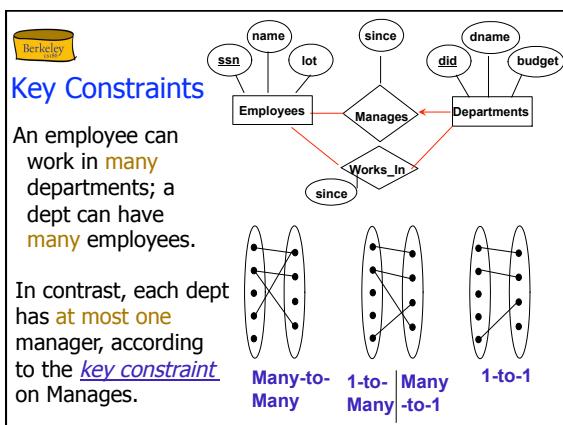
- **Relationship:** Association among two or more entities.
 - E.g., Attishoo works in Pharmacy department.
 - relationships can have their own attributes.
- **Relationship Set:** Collection of similar relationships.
 - An n -ary relationship set R relates n entity sets $E_1 \dots E_n$; each relationship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



ER Model Basics (Contd.)

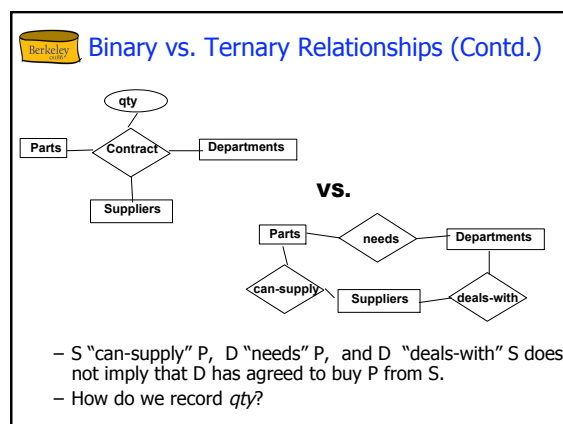


- Same entity set can participate in different relationship sets, or in different "roles" in the same relationship set.



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!





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



Online Resources

- Website
 - <http://www.cs186berkeley.net/> or
 - <http://inst.eecs.berkeley.edu/~cs186>
- Class blog
- Class bulletin board
 - <http://tech.groups.yahoo.com/group/cs186-09/>



Administrivia, cont.

- Office Hours and Sections on the website
 - One more section will be added soon
 - Discussion Sections meet next Monday
- Syllabus & HW calendar coming on-line
 - Schedule and due dates may change (check frequently)
 - Lecture notes are/will be posted
- **Acct forms!**
- Other textbooks
 - Korth/Silberschatz/Sudarshan
 - O'Neil and O'Neil
 - Garcia-Molina/Ullman/Widom



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)
```

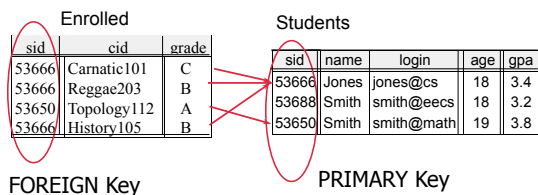
- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
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)**



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.

Berkeley 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))
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."

Berkeley 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.)

Berkeley 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			Students				
sid	cid	grade	sid	name	login	age	gpa
53666	Carnatic101	C	53666	Jones	jones@cs	18	3.4
53666	Reggae203	B	53688	Smith	smith@eecs	18	3.2
53650	Topology112	A	53650	Smith	smith@math	19	3.8
53666	History105	B					
11111	English102	A					

Berkeley Enforcing Referential Integrity

- *sid* in Enrolled: foreign key referencing Students.
- Scenarios:
 - Insert Enrolled tuple with non-existent student id?
 - Delete a Students tuple?
 - Also delete Enrolled tuples that refer to it? (Cascade)
 - Disallow if referred to? (No Action)
 - Set sid in referring Enrolled tuples to a *default* value? (Set Default)
 - Set sid in referring Enrolled tuples to *null*, denoting 'unknown' or 'inapplicable'. (Set NULL)
- Similar issues arise if primary key of Students tuple is updated.

Berkeley 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!

Berkeley Where do ICs Come From?

- Semantics of the real world!
- Note:
 - We can check IC violation in a DB instance
 - 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

- Feature: Simple, powerful *ad hoc* querying
- Declarative languages
 - Queries precisely specify *what* to return
 - DBMS is responsible for efficient evaluation (*how*).
 - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.
 - Key to data independence!



The SQL Query Language

- The most widely used relational query language.
 - Current std is SQL:2008; 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 features 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