Midterm Review

EECS 339

Outline

- DB background and history
- Entity-relationship modeling
- Relational model
- Database normalization
- Relational algebra and calculus
- SQL

Background

- Databases enable users to efficiently and declaratively ask questions about their data
  - Non-computer scientists need to be able to work with their data too
- SQL has been the dominant language in data management since the 1970s

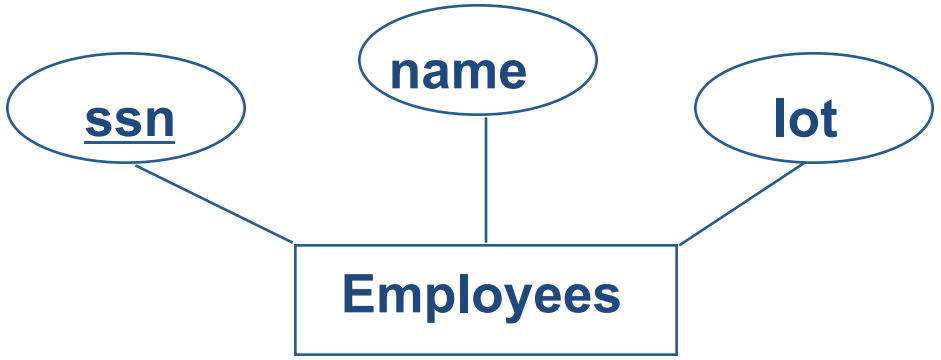
Main Advantages of DBMS

- Data independence
- Efficient querying
- Data integrity and security
- Data administration
- Concurrency and crash recovery
- Reduced developer bandwidth

Overview of Database Design

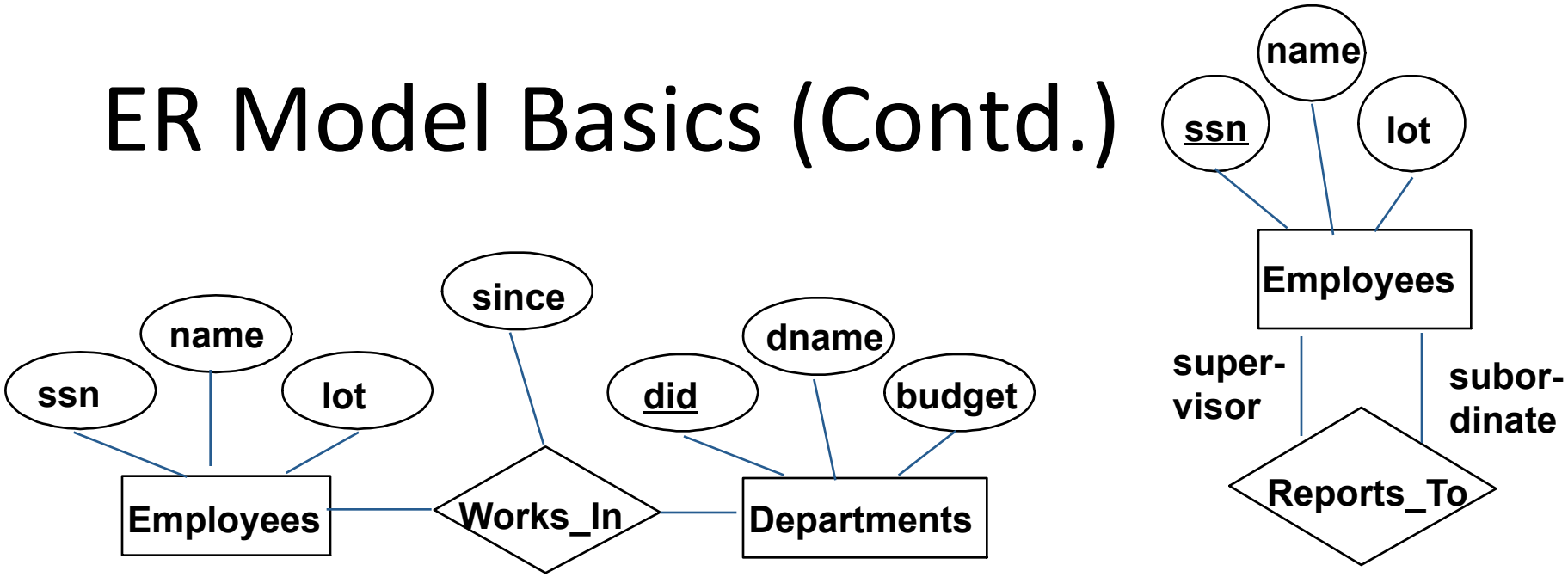
- *Conceptual design*: *(ER Model is used at this stage.)*
  - What are the *entities* and *relationships* in the enterprise?
  - What information about these entities and relationships should we store in the database?
  - What are the *integrity constraints* or *business rules* that hold?
  - A database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

ER Model Basics



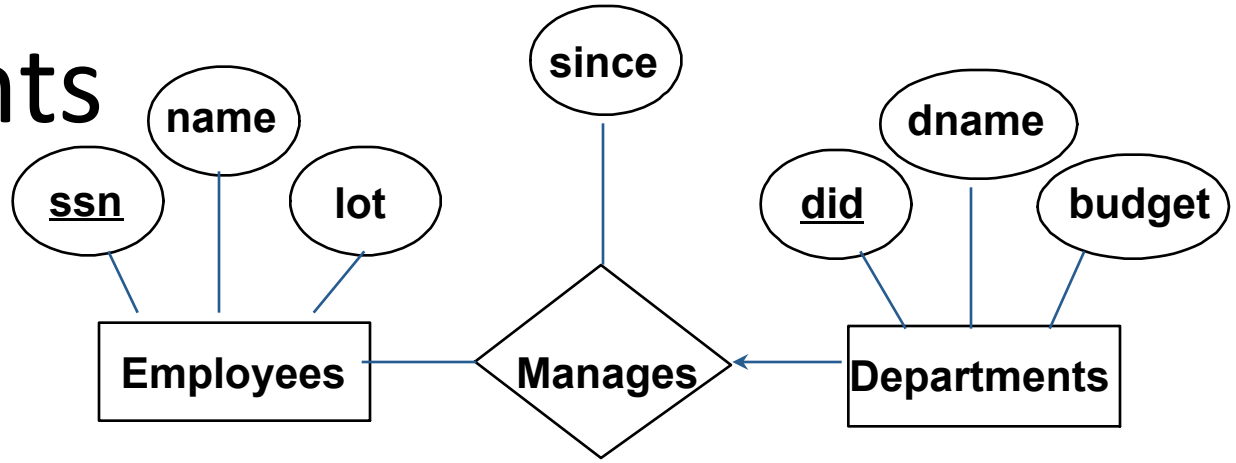
- *Entity:* Real-world object distinguishable from other objects. An entity is described (in DB) using a set of *attributes*.
- *Entity Set*: A collection of similar entities. E.g., all employees.
  - All entities in an entity set have the same set of attributes. (Until we consider ISA hierarchies, anyway!)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

ER Model Basics (Contd.)

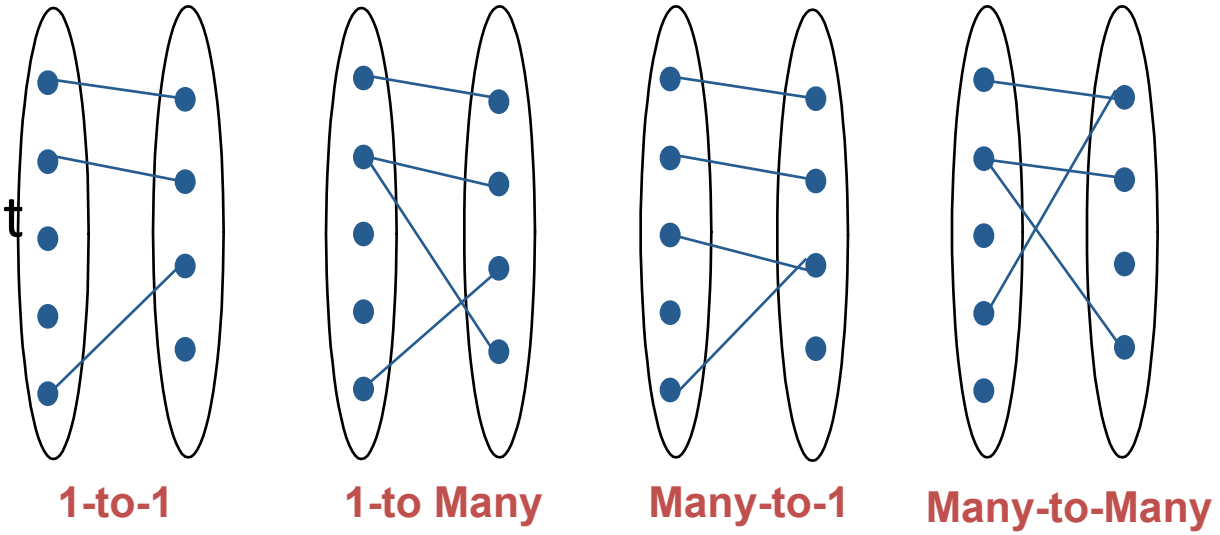


- *Relationship*: Association among two or more entities. E.g., Attishoo works in Pharmacy department.
- *Relationship Set*: Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1 E1, ..., en En
    - Same entity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



Representing “Multiplicity”

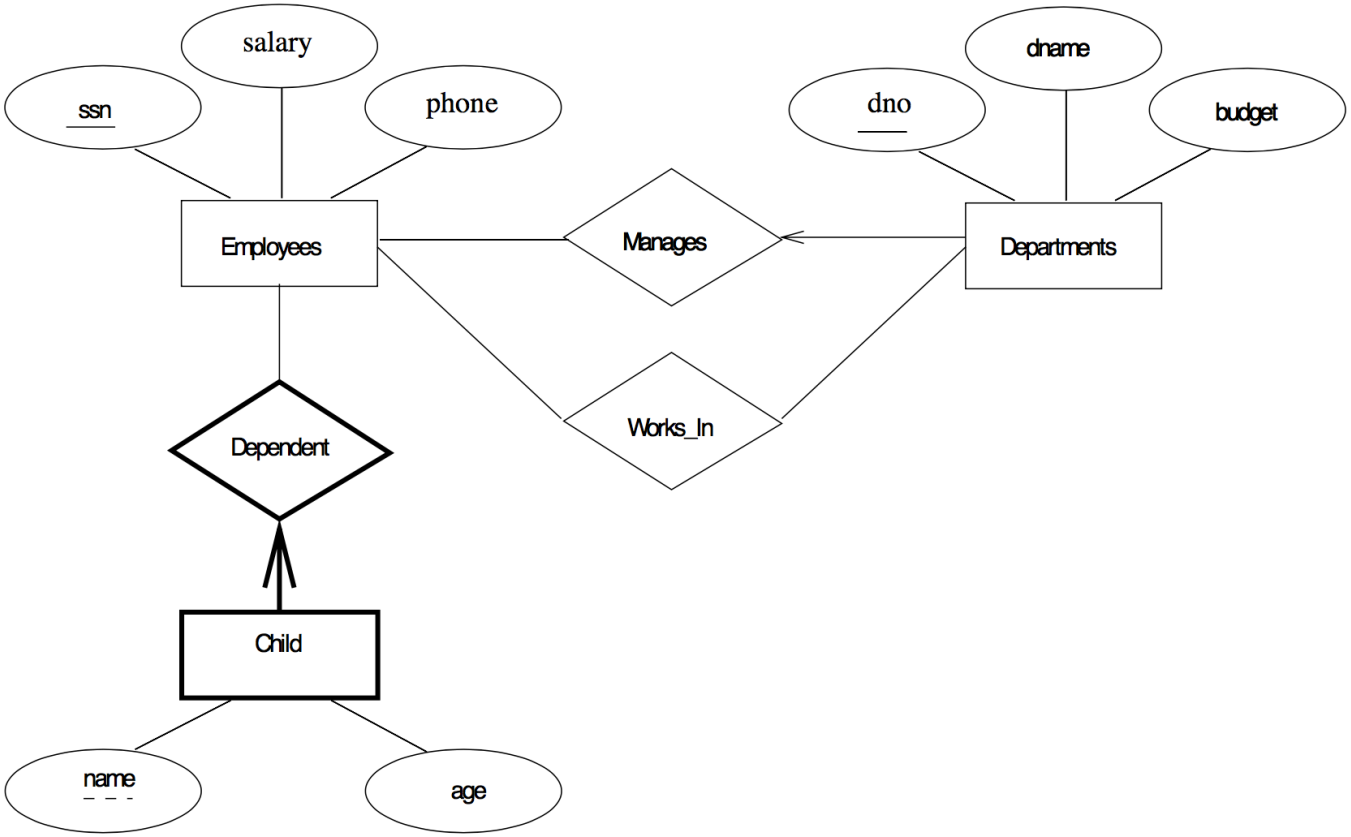
- Show a many-one relationship by an arrow entering the “one” side.
  - Remember: Like a functional dependency.
- Show a one-one relationship by arrows entering both entity sets.
- Rounded arrow = “exactly one,” i.e., each entity of the first set is related to exactly one entity of the target set.

Example Question

A company database needs to store information about employees (identified by ssn, with salary and phone as attributes), departments (identified by dno, with dname and budget as attributes), and children of employees (with name and age as attributes). Employees work in departments; each department is managed by an employee; a child must be identified uniquely by name when the parent (who is an employee; assume that only one parent works for the company) is known. We are not interested in information about a child once the parent leaves the company.

Draw an ER diagram that captures this information.

Solution


salary
ssn
phone
dname
dno
budget
Employees
Manages
Departments
Works_In
Dependent
Child
name
age

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).

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.

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, 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!

The Evils of Redundancy

- *Redundancy* is at the root of several problems associated with relational schemas:
  - redundant storage, insert/delete/update anomalies
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: *decomposition* (replacing ABCD with, say, AB and BCD, or ACD and ABD).
- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

Functional Dependencies (FDs)

- A functional dependency X → Y holds over relation R if, for every allowable instance *r* of R:
  - given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- An FD is a statement about *all* allowable relations.
  - Must be identified based on semantics of application.
  - Given some allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!
- K is a candidate key for R means that K → R
  - However, K → R does not require K to be *minimal*!

Converting a schema into Boyce-Codd Normal Form

- Start with one "universal relation"
- For each FD if X → Y violates BCNF (i.e., X is not a superkey over Y)
  - split up R into R1 and R2
    - R1 = X Union Y
    - R2 = R – (X U Y)
- Continue iterating until redundancy is eliminated

Recommended reading:
http://db.csail.mit.edu/6.830/lectures/lec3-notes.pdf

Example Question

- Consider the attribute set R = ABCDEG and the FD set F = {AB → C, AC → B, BC → A, E → G}.

- Decompose this into BCNF

Solution

- Start: R = {A, B, C, D, E, G}
  - {AB → C, AC → B, B → D, BC → A, E → G}
- R1 = {A, B, C, D} (covers AB→C, AC → B, BC→A, B → D)
- R2 = {E, G}
  - {E → G}

Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.
- Additional operations:
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)

Example Question

- Consider the following schema:

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

- Express the following queries with relational algebra:
  - Find the names of suppliers who supply some red part.
  - Find the sids of suppliers who supply some red or green part.
  - Find the sids of suppliers who supply some red part or are at 221 Packer Street.
  - Find the sids of suppliers who supply some red part and some green part.

Solution

- 1. πsname(πsid((πpidσcolor=′red′Parts) ◁▷ Catalog) ◁▷ Suppliers)
- 2. πsid(πpid(σcolor=′red′ ⋁ color=′green′ Parts) ◁▷ catalog)
- 3. ρ(R1,πsid((πpidσcolor=′red′Parts) ◁▷ Catalog))
- ρ(R2,πsidσaddress=′221PackerStreet′ Suppliers) R1 ∪ R2
- 4. ρ(R1,πsid((πpidσcolor=′red′Parts) ◁▷ Catalog)) ρ(R2,πsid((πpidσcolor=′green′ Parts) ◁▷ Catalog)) R1∩R2

SQL Syntax

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

- Select-project-join
- Aggregation

Example Questions

- Consider the following schema:

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

- Find the sids of suppliers who supply every red part or supply every green part.
- Find pairs of sids such that the supplier with the first sid charges more for some part than the supplier with the second sid.
- Find the pids of parts supplied by at least two different suppliers.
- Find the pids of the most expensive parts supplied by suppliers named Yosemite Sam.

Solutions

```
SELECT C.sid
FROM   Catalog C
WHERE  (NOT EXISTS (SELECT P.pid
                    FROM   Parts P
                    WHERE  P.color = 'red' AND
                    (NOT EXISTS (SELECT C1.sid
                                 FROM   Catalog C1
                                 WHERE  C1.sid = C.sid AND
                                        C1.pid = P.pid))))
       OR ( NOT EXISTS (SELECT P1.pid
                        FROM   Parts P1
                        WHERE  P1.color = 'green' AND
                        (NOT EXISTS (SELECT C2.sid
                                     FROM   Catalog C2
                                     WHERE  C2.sid = C.sid AND
                                            C2.pid = P1.pid))))
```

Solutions

```
SELECT C1.sid, C2.sid
FROM Catalog C1, Catalog C2
WHERE C1.pid = C2.pid AND C1.sid  >=C2.sid;
```

```
SELECT C.pid
FROM Catalog C
WHERE EXISTS (SELECT C1.sid
                FROM Catalog C1
                WHERE C1.pid = C.pid AND
                   C1.sid != C.sid )
```

Solutions (cont'd)

```
SELECT C.pid
FROM Catalog C, Suppliers S
WHERE S.sname = 'Yosemite Sam' AND C.sid = S.sid
    AND C.cost ≥ ALL (Select C2.cost
            FROM Catalog C2, Suppliers S2
            WHERE S2.sname = 'Yosemite Sam' AND
C2.sid = S2.sid)
```