Review Session

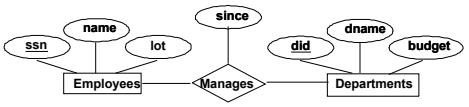
- ER and Relational
  - ER ⇔ Relational
  - Constraints, Weak Entities, Aggregation, ISA
- Relational Algebra ⇔ Relational Calculus
  - Selections/Projections/Joins/Division
- SQL (Division, Outer-Joins, Constraints)
- Your questions

ER & Relational Review

Employees (ssn CHAR(11), name CHAR(20), lot INTEGER)

Departments (did INTEGER, dname CHAR(20), budget INTEGER)

Manages (did INTEGER, ssn CHAR(11), since DATE)
-FOREIGN KEY (ssn) REFERENCES Employees
-FOREIGN KEY (did) REFERENCES Employees



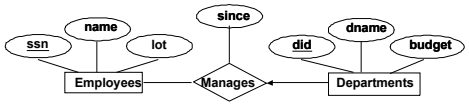
Each employee can manage **zero, one or more** departments.

Each department has **zero, one or more** managers.

Key Constraint

Employees (ssn CHAR(11), name CHAR(20),lot INTEGER)

Dept_Mgr (did INTEGER, ssn CHAR(11), dname CHAR(20), budget INTEGER, since DATE)
-FOREIGN KEY (ssn) REFERENCES Employees



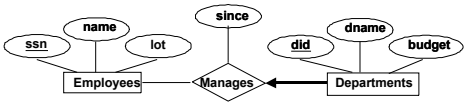
Each employee can manage **zero, one or more** departments.

Each department has **at most one** manager.

Key + Participation Constraint

Employees (ssn CHAR(11), name CHAR(20),lot INTEGER)

Dept_Mgr (did INTEGER, ssn CHAR(11), dname CHAR(20), budget INTEGER, since DATE)
- FOREIGN KEY (ssn) REFERENCES Employees
- ssn **NOT NULL**



Each employee can manage **zero, one or more** departments.

Each department has **exactly one** manager.

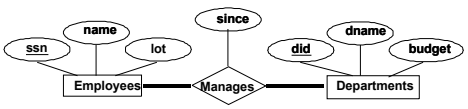
Participation Constraint

Employees (ssn CHAR(11), name CHAR(20),lot INTEGER)

Departments (did INTEGER, dname CHAR(20), budget INTEGER)

Manages (did INTEGER, ssn CHAR(11), since DATE)
-FOREIGN KEY (ssn) REFERENCES Employees
-FOREIGN KEY (did) REFERENCES Employees

Insuffcient! Additional checks required.

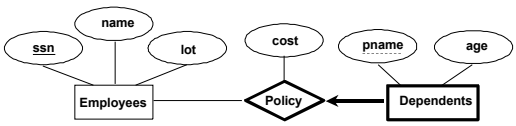


Each employee manages **at least one** department.

Each department has **at least one** manager.

Weak Entities

Dep_Policy (pname CHAR(20), ssn CHAR(11), age INTEGER, cost REAL)
- FOREIGN KEY (ssn) REFERENCES Employees
  -NOT NULL
  -ON DELETE CASCADE

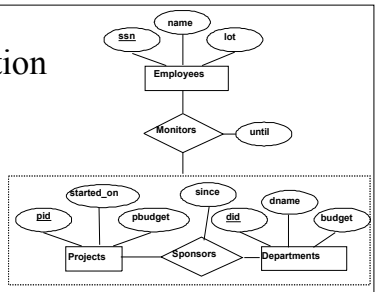


Weak entities have only a "partial key" (dashed underline)

Aggregation

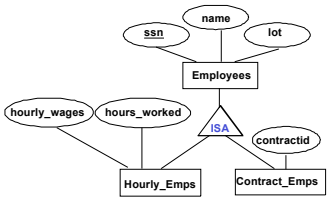
Used to model a relationship involving a *relationship set*.

Allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.



Employees(ssn, name, lot)
Projects(pid, pbudget, started_on)
Departments (did, dname, budget)
Sponsors (pid_FK, did_FK, since)
Monitors (pid_FK, ssn_FK, did_FK, until)

ISA (`is a') Hierarchies



Employees (ssn, name, lot)
Hourly_Emps(ssn_FK, hourly_Wages, hours_worked)
Contract_Emps(ssn_FK, contractid)

Relational Algebra: 5 Basic Operations

- ***Selection*** ( ) **Selects a subset of *rows* from relation (horizontal).**
- ***Projection*** ( ) **Retains only wanted *columns* from relation (vertical).**
- ***Cross-product*** ( ) **Allows us to combine two relations.**
- ***Set-difference*** (—) **Tuples in r1, but not in r2.**
- ***Union*** ( ) **Tuples in r1 and/or in r2.**
- ***Renaming*** ( ) **E.g.** (C(1 -> sid1, 5 -> sid2), S1 x R1)

Compound Operations

- **Intersect** ( )
  - R S = R (R S)
- Join
  - **Condition Join** $S1 \bowtie_{S1.sid < R1.sid} R1$
  - **Equijoin:** Special case where c contains only equalities
  - **Natural join**
    - Compute R S
    - Select rows where attributes that appear in both relations have equal values
    - Project all unique attributes and **one copy** of each of the common ones.

Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Tuple Relational Calculus

- ***Query* has the form: {*T* | *p*(*T*)}**
  - ***p(T)*** denotes a formula in which tuple variable ***T*** appears.

{S | S Sailors S.rating > 7
R(R Reserves R.sid = S.sid
R.bid = 103)}

Only one free variable

Bounded variable

Algebra ⇔ Calculus

Find the names of sailors who've reserved a red boat

$\pi_{sname}((\sigma_{color='red'}, Boats) \bowtie Reserves \bowtie Sailors)$

Projection

{S1 | ∃S (S∈Sailors ∧ S1.name = S.name ∧ Join

∃R(R∈Reserves ∧ R.sid = S.sid ∧

∃B(B∈Boats ∧ B.bid = R.bid ∧ B.color = 'red'))}

Join Selection

Division example

Find the names of sailors who've reserved ALL red boats

$\pi_{sname}((\pi_{sid, bid} Reserves) / (\pi_{bid}(\sigma_{color=red} Boats)) \bowtie Sailors)$

{S1 | ∃S (S∈Sailors ∧S.sname = S1.sname ∧ ∃B ∈ Boats (B.color = 'red' ∧ ∃R(R∈Reserves ∧ S.sid = R.sid ∧ B.bid = R.bid))))}

SQL Query

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

The *target-list* contains (i) list of column names & (ii) terms with aggregate operations (e.g., MIN (*S.age*)).

- column name list (i) **can contain only attributes from the *grouping-list*.**

Conceptual Evaluation

- Compute Cartesian product (×) of all tables in FROM clause
- Discard rows not satisfying WHERE clause (Selection σ)
- Group the remaining rows according to Grouping-List
- Apply HAVING clause
- Apply SELECT list (Projection π)
- If there is DISTINCT, eliminate duplicates
- Order remaining tuples according to ORDER BY

Division in SQL

Find sailors who've reserved ALL boats.

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
                  FROM Boats B
                  WHERE NOT EXISTS (SELECT R.bid
                                    FROM Reserves R
                                    WHERE R.bid=B.bid
                                      AND R.sid=S.sid))
```

Simpler →

```
SELECT S.name
FROM Sailors S, reserves R
WHERE S.sid = R.sid
GROUP BY S.name, S.sid
HAVING
      COUNT(DISTINCT R.bid) =
       ( Select COUNT (*) FROM Boats)
```

```
SELECT s.sid, s.name, r.bid
FROM Sailors s LEFT OUTER JOIN Reserves r
ON s.sid = r.sid
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

sid	bid	day
22	101	10/10/96
95	103	11/12/96

s.sid	s.name	r.bid
22	Dustin	101
95	Bob	103
31	Lubber	

Constraints Over Multiple Relations

```
CREATE TABLE Sailors
( sid INTEGER,
  sname CHAR(10),
  rating INTEGER,
  age REAL,
  PRIMARY KEY (sid),
  CHECK
  ( (SELECT COUNT (S.sid) FROM Sailors S)
  + (SELECT COUNT (B.bid) FROM
        Boats B) < 100 )
```

Number of boats plus number of sailors is < 100

- Awkward and wrong!
- Only checks sailors!
- Only required to hold if the associated table is non-empty.

- ASSERTION is the right solution; not associated with either table.

```
CREATE ASSERTION smallClub
CHECK
( (SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid)
 FROM Boats B) < 100 )
```