# Logical Database Design: Mapping ER to Relational

Chapter 3, Section 3.5

# ER Model vs. Relational Model

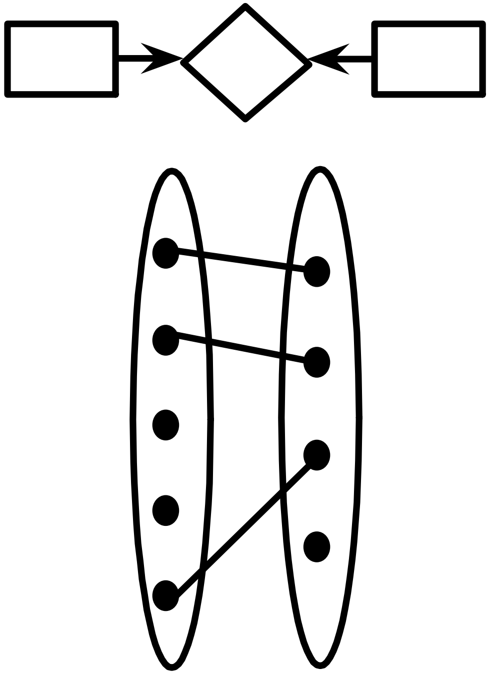
- ER Model used for conceptual design
- Relational Model implemented by modern DBMS
- Important Step: Translate ER diagram to Relational schema

# Recall ER Constructs

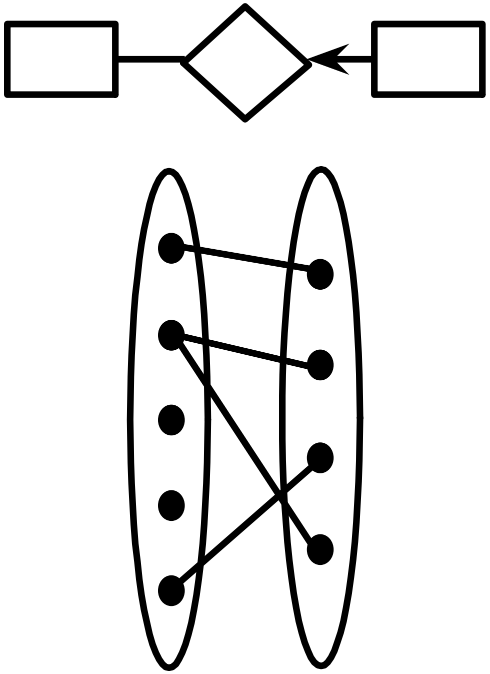
- Basic Constructs
  - Entity Sets
  - Relationship Sets
  - Attributes (of entities and relationships)
- Additional Constructs
  - ISA Hierarchies
  - Weak Entities
  - Aggregation
- Integrity Constraints
  - Key constraints
  - Participation constraints
  - Overlap / Covering constraints for ISA hierarchies

# Review: ER relationship types

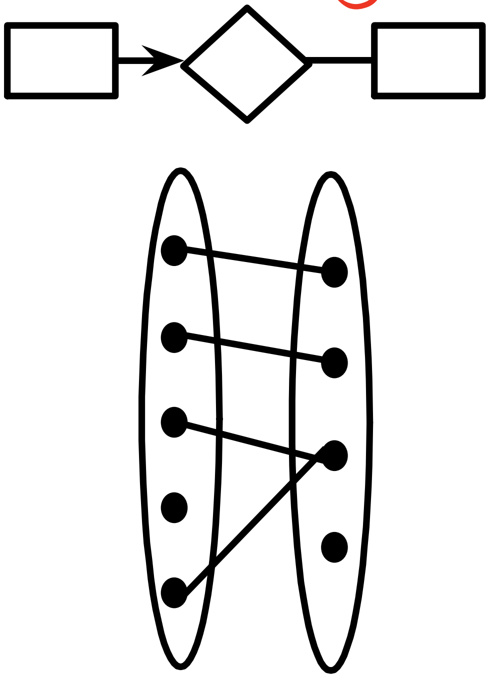
!!! key constraints affect how



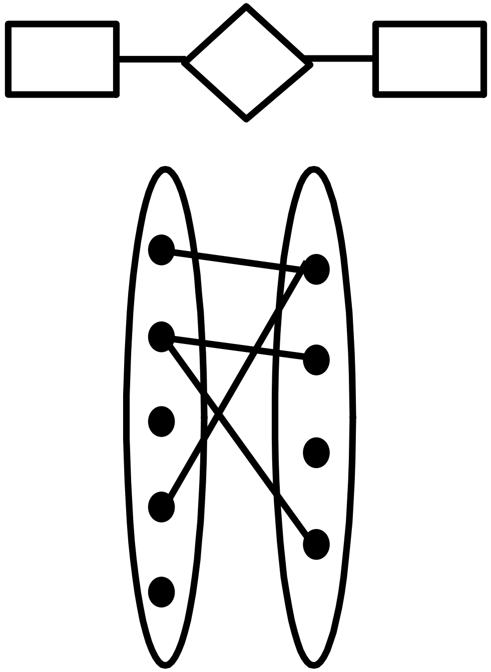
1-to-1



1-to Many
1-to-N



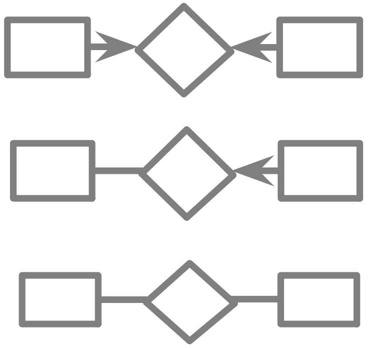
Many-to-
1
N-to-1



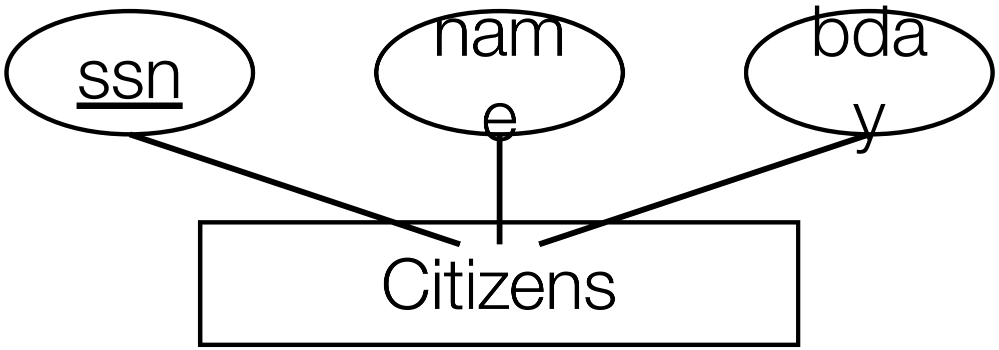
Many-to-Many
M-to-N

# ER to Tables: Basics

- **Strong entities:**
  - key = primary key
- **Relationships:** keys come from the participating entities. E.g. for binary relationships:
  - **1-to-1:** either key (other = candidate key)
  - **1-to-N:** the key of the ‘many’ (N) part
  - **M-to-N:** both keys

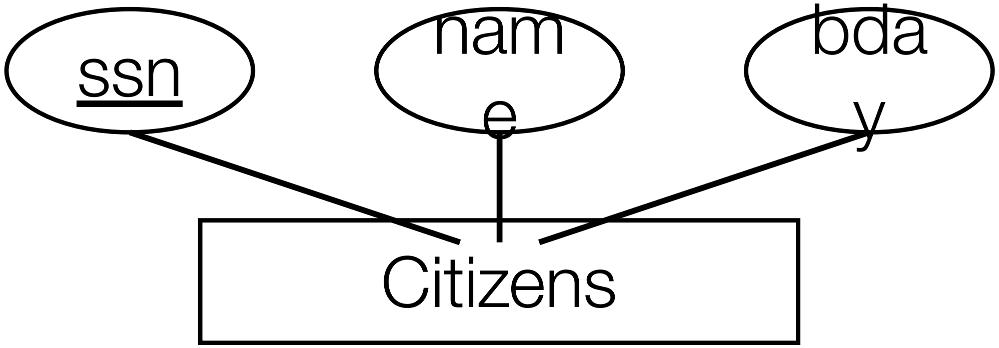


# Entity Sets to Tables



```
CREATE TABLE Citizens
      (ssn    CHAR(11),
    name   CHAR(20),
    bday   DATE,
    PRIMARY KEY  (ssn))
```

# Question??



```
CREATE TABLE Citizens
      (ssn    CHAR(11),
    name   CHAR(20),
    bday   DATE,
    PRIMARY KEY   (ssn))
```

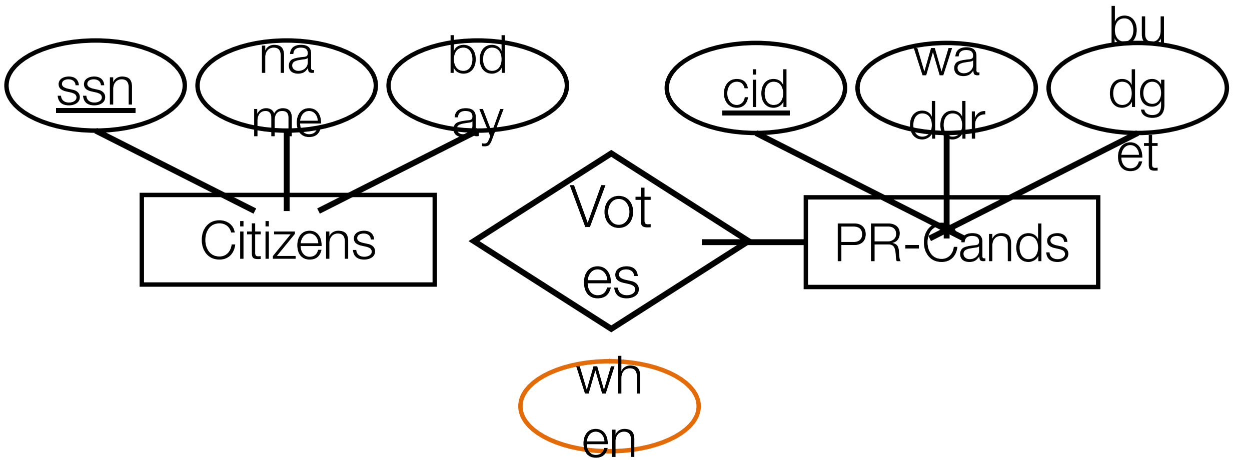
Can ssn have a null value?

A. Yes

[✓ B. No]

C. Depends

# Relationship Sets to Tables



```
CREATE TABLE Votes(
    ssn      CHAR(11),
    cid      INTEGER,
    when     DATE,
PRIMARY KEY (ssn, cid),
FOREIGN KEY (ssn) REFERENCES Citizens,
FOREIGN KEY (cid) REFERENCES PR-Cands)
```

- Foreign key: keys from participating entity sets
- Descriptive attributes

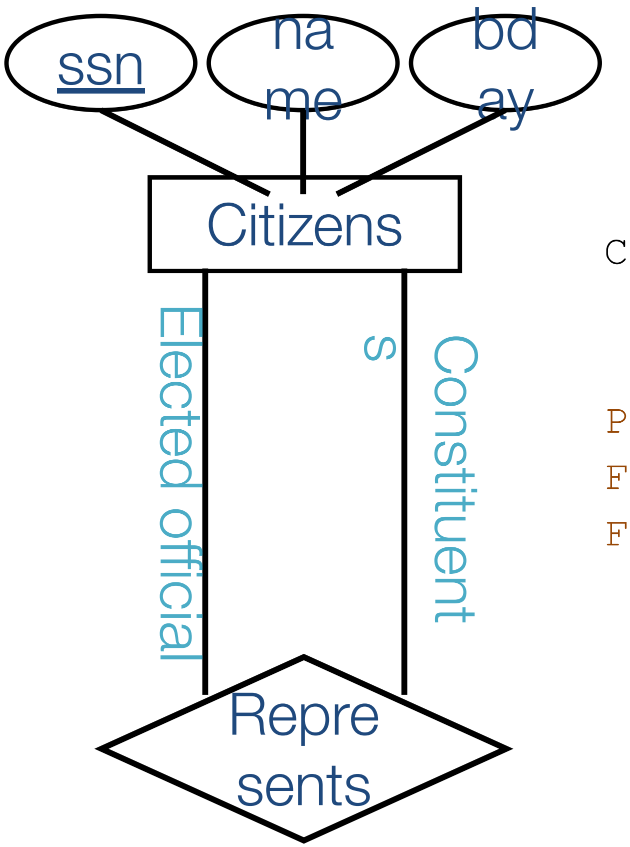
Generalizes to n-ary relationships

(we will see example later)

same col name => no need specify

# Relationship Sets to Tables

same entity participate in two different roles

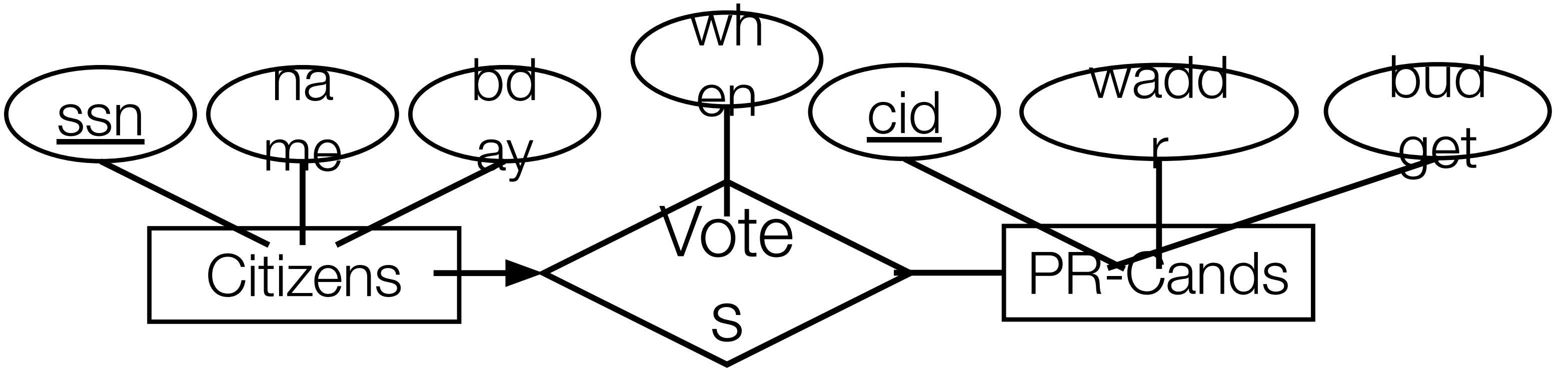


```
CREATE TABLE Represents(
    elected_ssn  CHAR(11),
    cons_ssn     CHAR(11),
PRIMARY KEY (elected_ssn, cons_ssn),
FOREIGN KEY (elected_ssn) REFERENCES Citizens(ssn),
FOREIGN KEY (cons_ssn) REFERENCES Citizens(ssn))
```

(ssn, ssn) ⇒ make no sense!!!

Note that you need to specify the column that you are referring to in the `Citizens` table, as `Citizens` does not have “`elected_ssn`” nor “`cons_ssn`”.

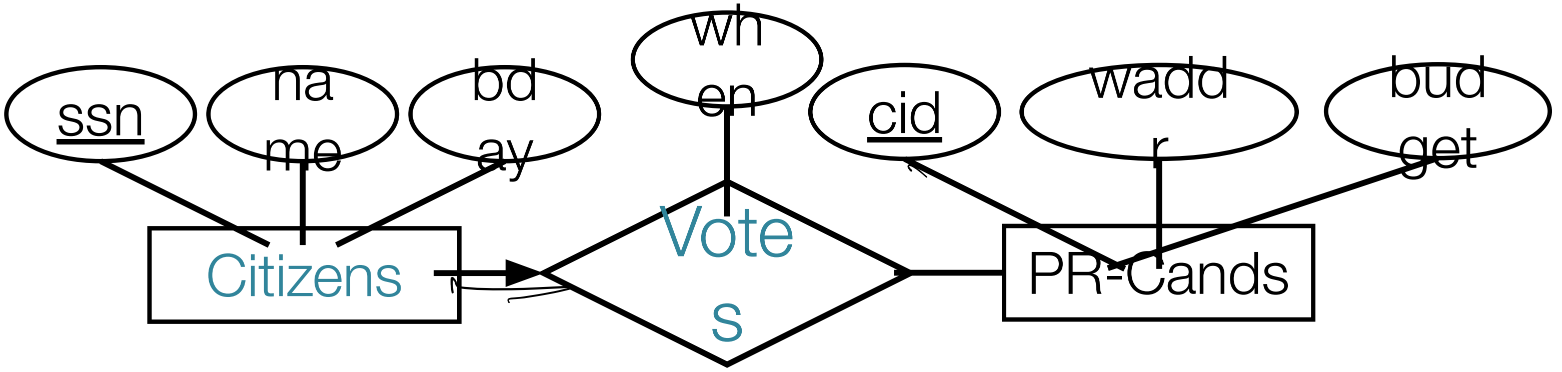
# Key Constraints



- Approach 1: **Three Tables** (Citizens, Votes, PR-Cands)

```
CREATE TABLE  Votes
(   ssn      CHAR(11),
    cid      INTEGER,
    when     DATE,
     PRIMARY KEY (ssn),
     FOREIGN KEY (ssn)  REFERENCES Citizens,
     FOREIGN KEY (cid)  REFERENCES PR-Cands)
```

# Key Constraints

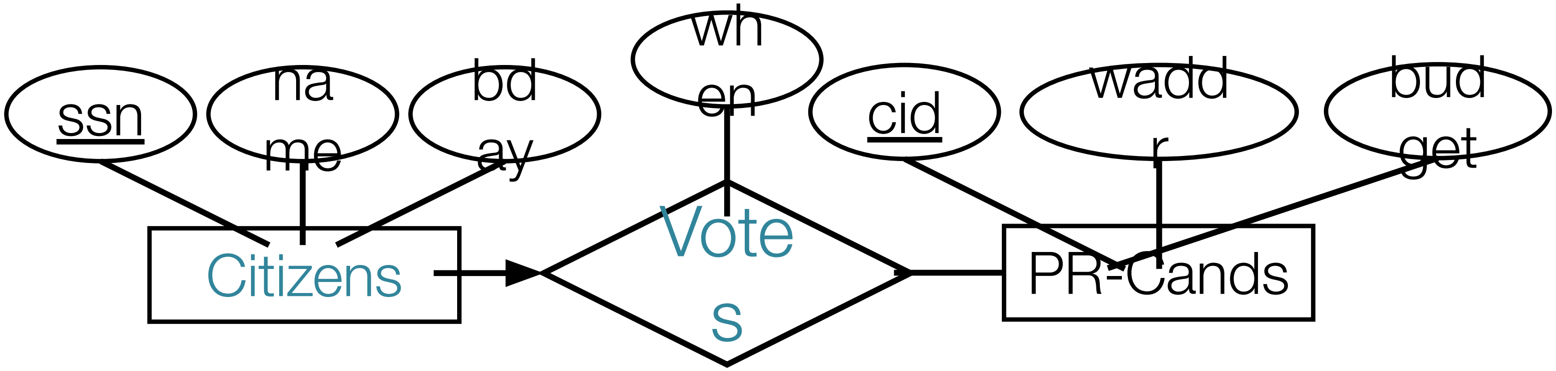


*Each citizen can only vote once, so OK to fold 'Votes' relationship into 'Citizens' entity* !!!

- Approach 2: **Two Tables** (Citizen_Votes, PR-Cands)

```
CREATE TABLE  Citizen_Votes (
   ssn     CHAR(11),
   name    CHAR(20),
   bday    DATE,
   when    DATE,
   cid     INTEGER,
PRIMARY KEY  (ssn),
FOREIGN KEY (cid) REFERENCES PR-Cands)
```

# Question??



*Each citizen can only vote once, so OK to fold 'Votes' relationship into 'Citizens' entity*

- Approach 2: **Two** Tables (Citizen_Votes, PR-Cands)

```
CREATE TABLE  Citizen_Votes (
    ssn     CHAR(11),
    name    CHAR(20),
    bday    DATE,
    when    DATE,
    cid     INTEGER,
PRIMARY KEY  (ssn),
FOREIGN KEY (cid) REFERENCES PR-Cands)
```

PR-cand table can't be null

cid in

primary never null

foreign key may be null

Q: Can cid be null?

A – YES; B- NO

Q: Which approach is better?

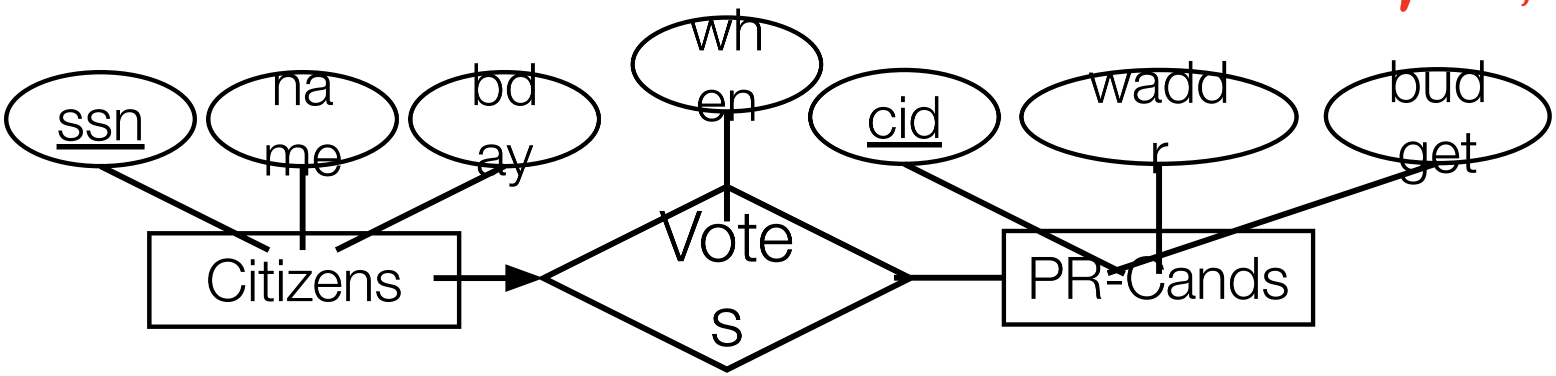
A – 3 Tables

B – 2 Tables

# Key Constraints

preferred with high partici rate

partici Constraint

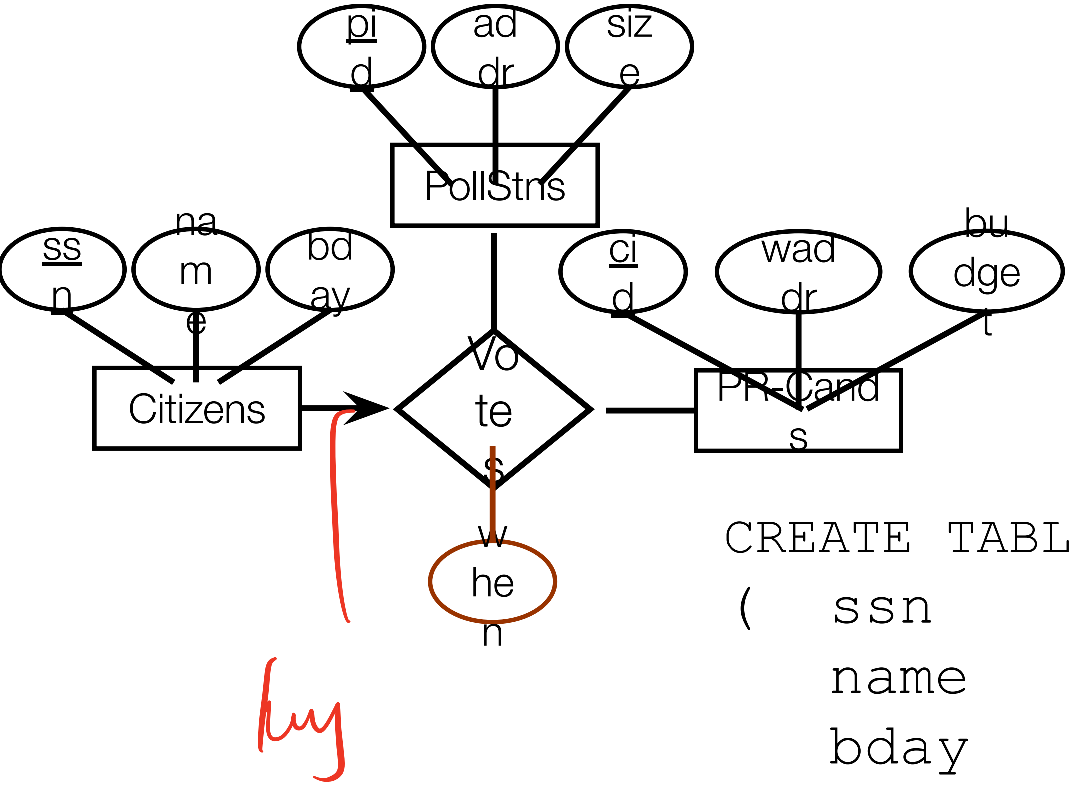


- What about **one** table?
- No!  This is bad design.

e.g., For each citizen that votes for Candidate-X, we have to store Candidate-X information

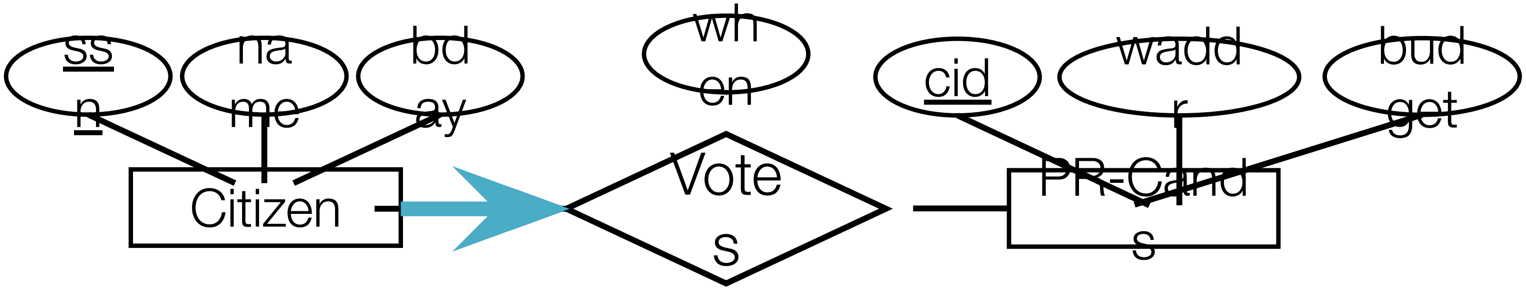
(cid, waddr, budget) => REDUNDANCY!

# Key Constraints: N-ary relationships



```
CREATE TABLE Citizens_Votes
(   ssn      CHAR(11),
    name     CHAR(20),
    bday     DATE,
    when     DATE,
    pid      INTEGER,
    cid      INTEGER,
    PRIMARY KEY  (ssn),
    FOREIGN KEY (cid) REFERENCES PR-Cands,
    FOREIGN KEY (pid) REFERENCES PollStns)
```

# Participation Constraints



## Using Approach 2

```
CREATE TABLE  Citizen_Votes(
    ssn      CHAR(11),
    name     CHAR(20),
    bday     DATE,
    when     DATE,
    cid      INTEGER NOT NULL,
PRIMARY KEY  (ssn),
FOREIGN KEY  (cid) REFERENCES PR_Cands
  ON DELETE NO ACTION);
```

must vote.

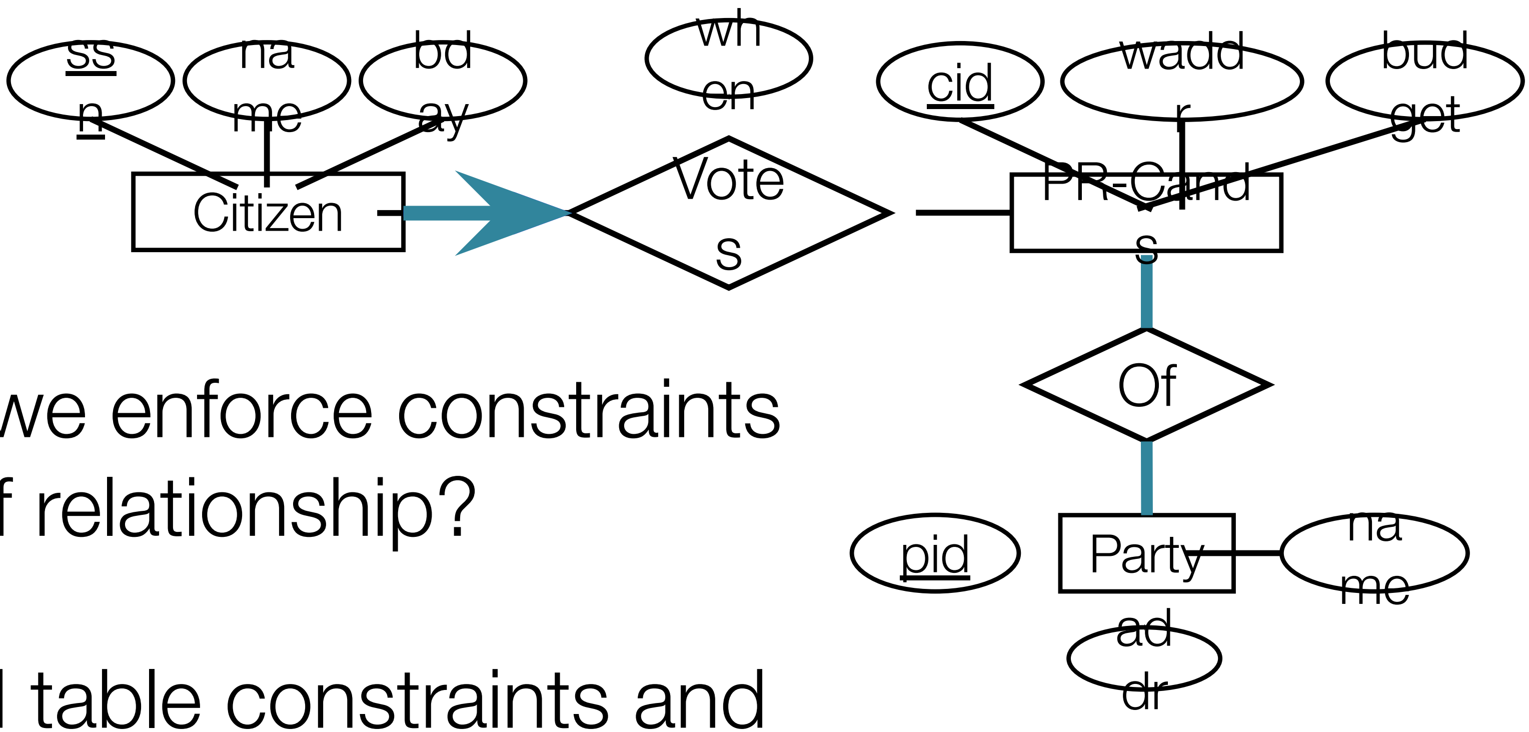
⇒ just disallow

(Don't allow cand withdraw)

Can we enforce the participation constraint using Approach 1 (three tables) ?

No !!!

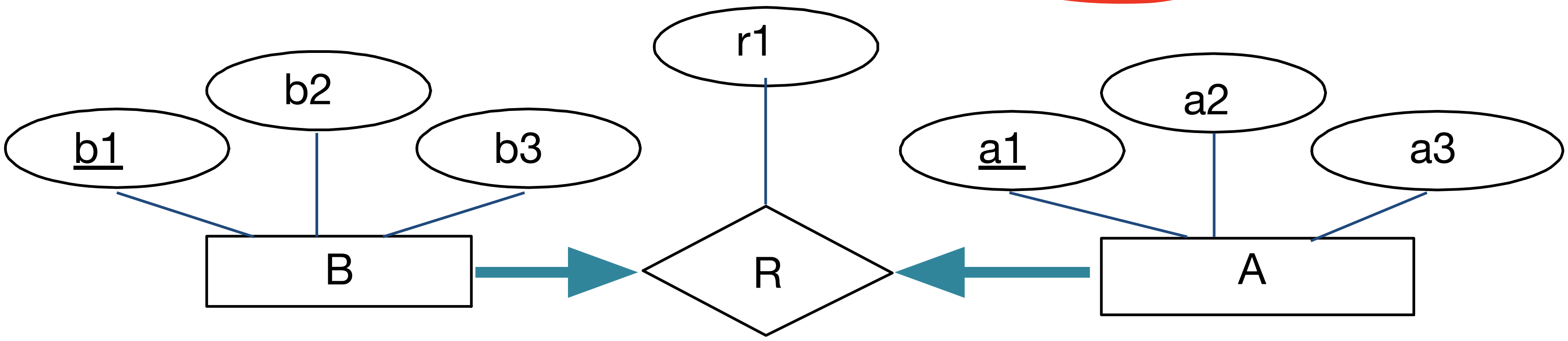
# Participation Constraints



Can we enforce constraints on Of relationship?

Need table constraints and assertions (later).

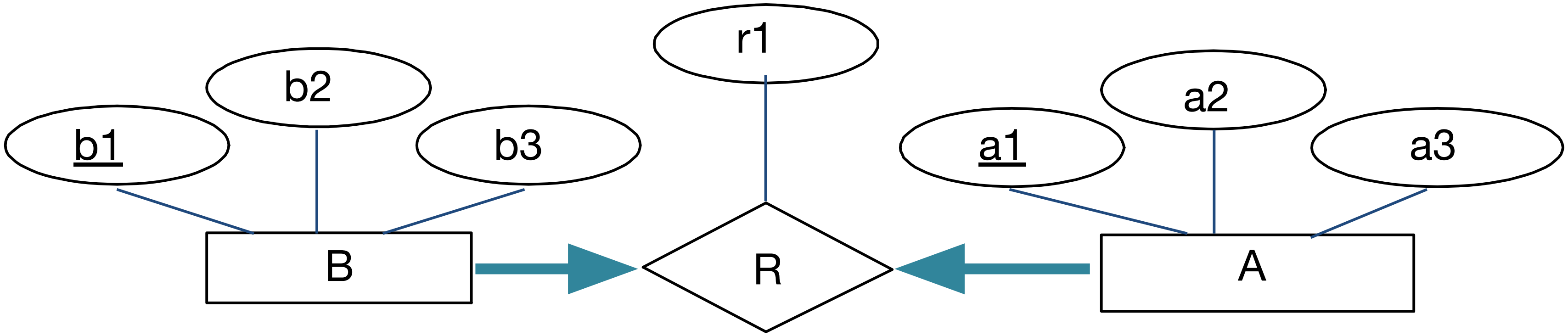
# Mapping Participation Constraints (1-to-1)



```
CREATE TABLE  RAB(
    r1 Integer,
    a1 Integer,
    a2 Integer,
    a3 Integer,
    b1 Integer,
    b2 Integer,
    b3 Integer …)
```

Key constraints?

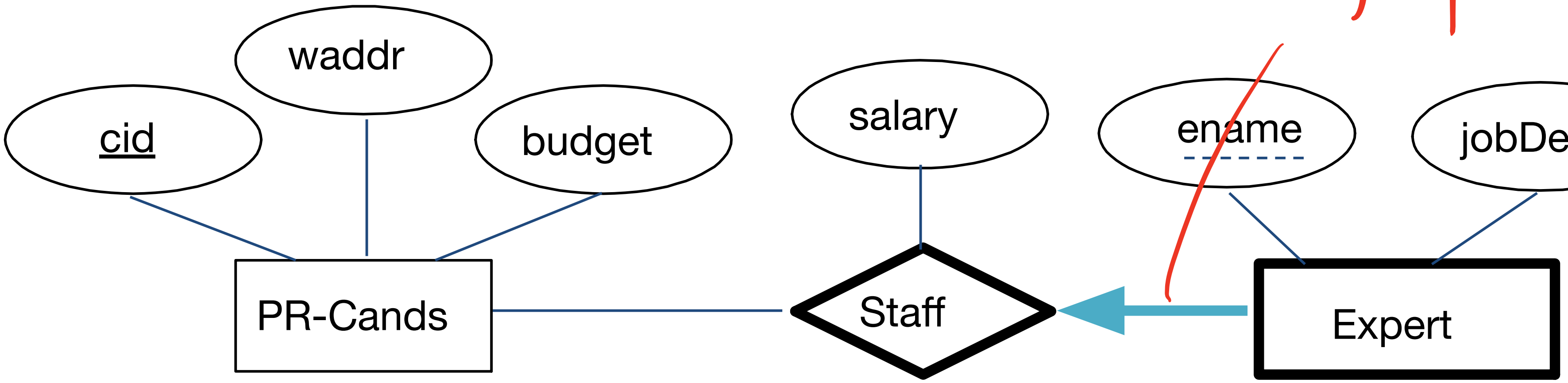
# Mapping Participation Constraints (1-to-1)



```
CREATE TABLE  RAB(
    r1 Integer,
    a1 Integer,
    a2 Integer,
    a3 Integer,
    b1 Integer NOT NULL,
    b2 Integer,
    b3 Integer,
       UNIQUE (b1), PRIMARY KEY (a1))
```

candidate key

# Weak Entities



key + par

- Approach 2: Combine weak entity and owning relationship into one relation
  - Delete all weak entities when an owner entity is deleted.

```
CREATE TABLE  Expert_Staff  (
    ename       CHAR(20),
    jobDesc     CHAR(40),
    salary      REAL,
    cid          INTEGER,
    PRIMARY KEY  (ename, cid),
    FOREIGN KEY  (cid) REFERENCES PR-Cands
        ON DELETE CASCADE)
```

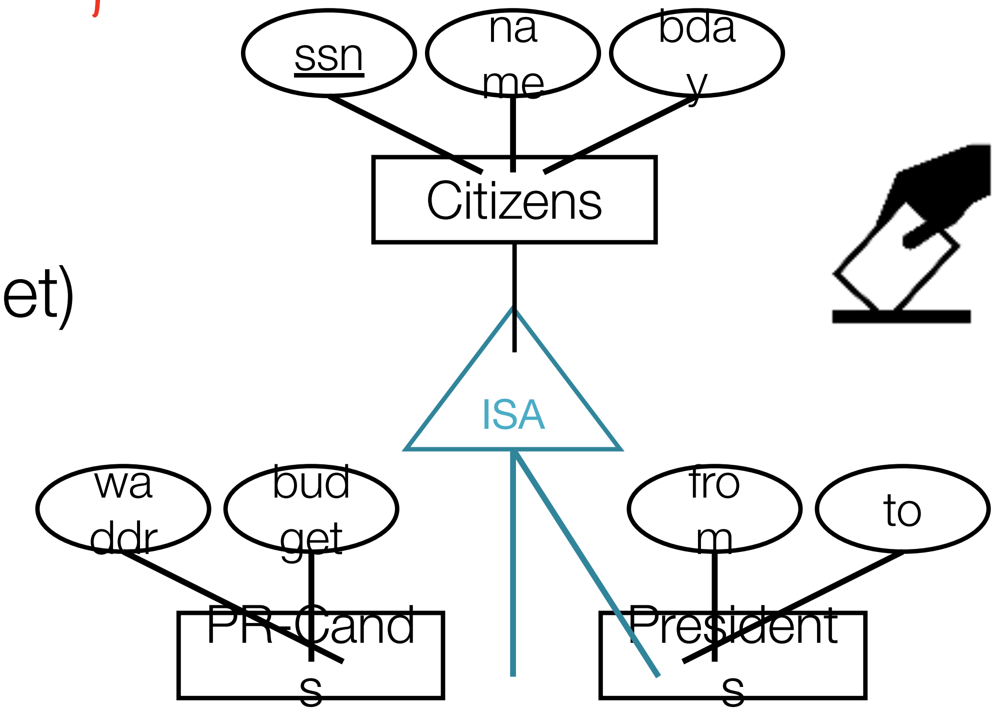
two table design

pair for weak entities

# ISA Hierarchies: General Approach

seperate table for parents and childs,

- Three relations:
  - Citizens (ssn, name, bday)
  - PR-Cands (ssn, waddr, budget)
  - Presidents (ssn, from, to)



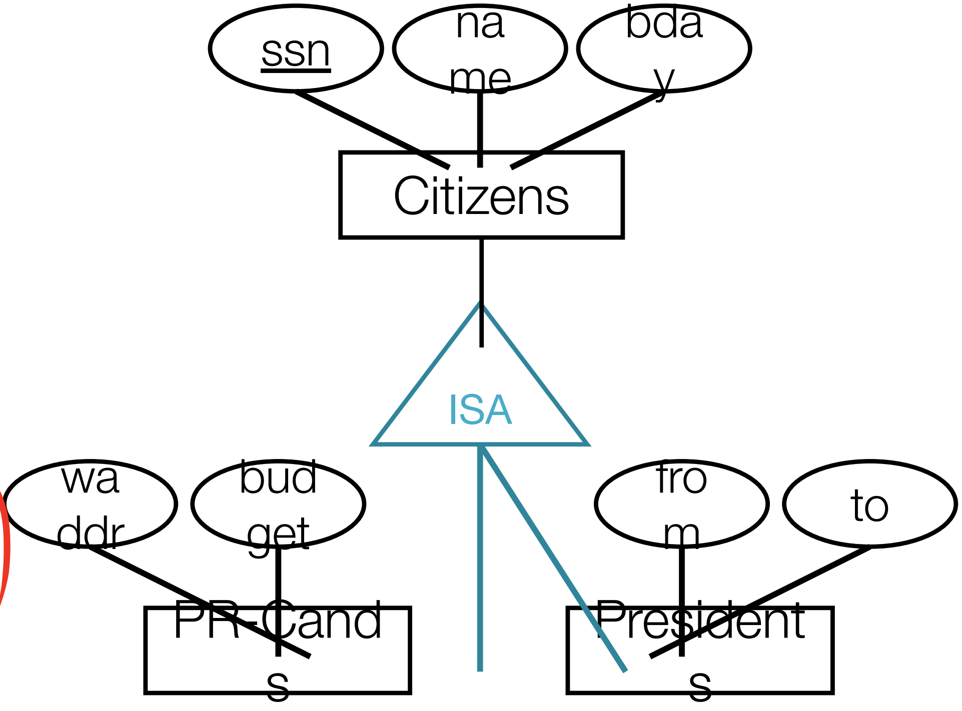
- Queries:
  - Involving all citizens => Easy
  - Involving just PR-Cands => need to join PR-cands with Citizens to get some attributes

from base class

# ISA Hierarchies: Alternative

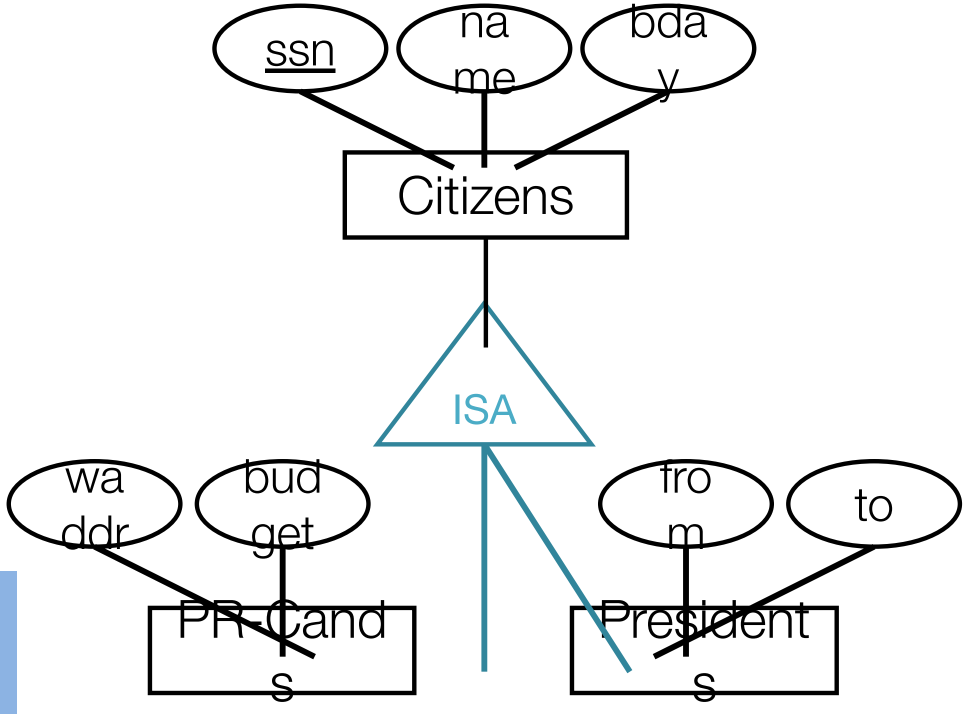
- Two relations:
  - PR-Cands (ssn, name, bday, waddr, budget)
  - Presidents (ssn, name, bday, from, to)

+ citizen ( not pre / pre-can)



# Question??

- Two relations:
  - PR-Cands (<u>ssn</u>, name, bday, waddr, budget)
  - Presidents (<u>ssn</u>, name, bday, from, to)

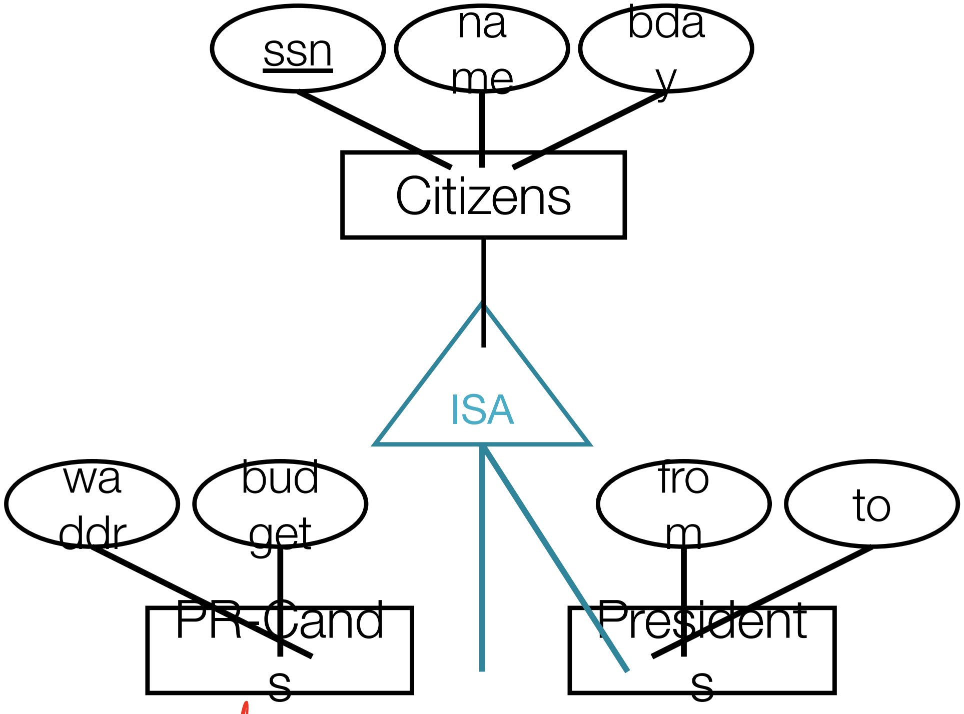


This design works best if we have:

A. Total Covering, Overlapping
B. Total Covering, Disjoint
C. Partial Covering, Overlapping
D. Partial Covering, Disjoint

# ISA Hierarchies: Alternative

- Two relations:
    - PR-Cands (ssn, name, bday, waddr, budget)
    - Presidents (ssn, name, bday, from, to)

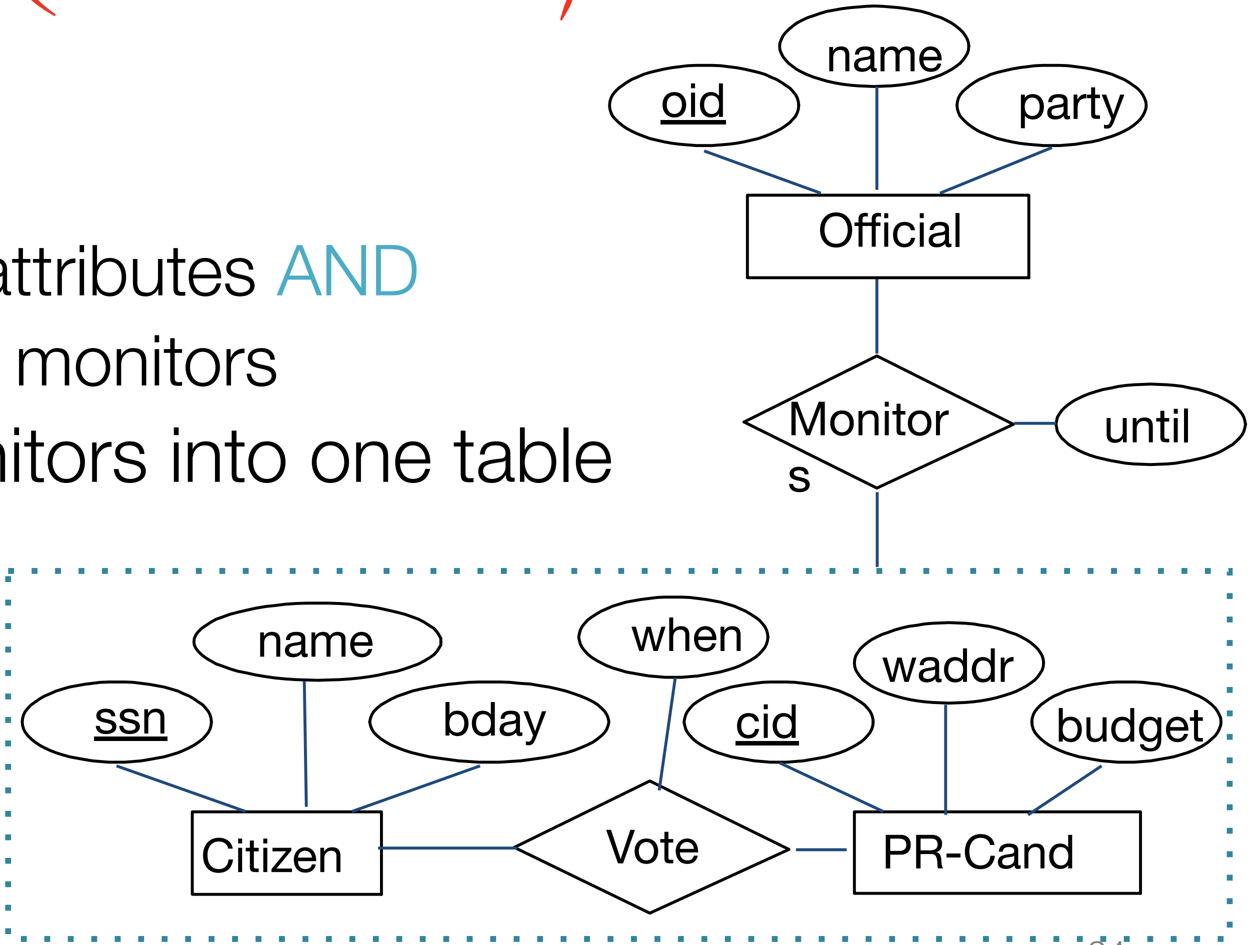


- Problems:
    - What if citizen is both?
        - Redundancy ( pre + pre-cand ).
    - What if citizen is neither?
        - Use General approach

# Aggregation

- Keep keys of all participating entity sets; decide primary
- Keys for Monitors (oid, (ssn, cid))
  - oid
  - (ssn, cid) aggre.

- Q: What if Vote:
  - has no descriptive attributes AND
  - total participation in monitors
- A: Fold Vote and Monitors into one table

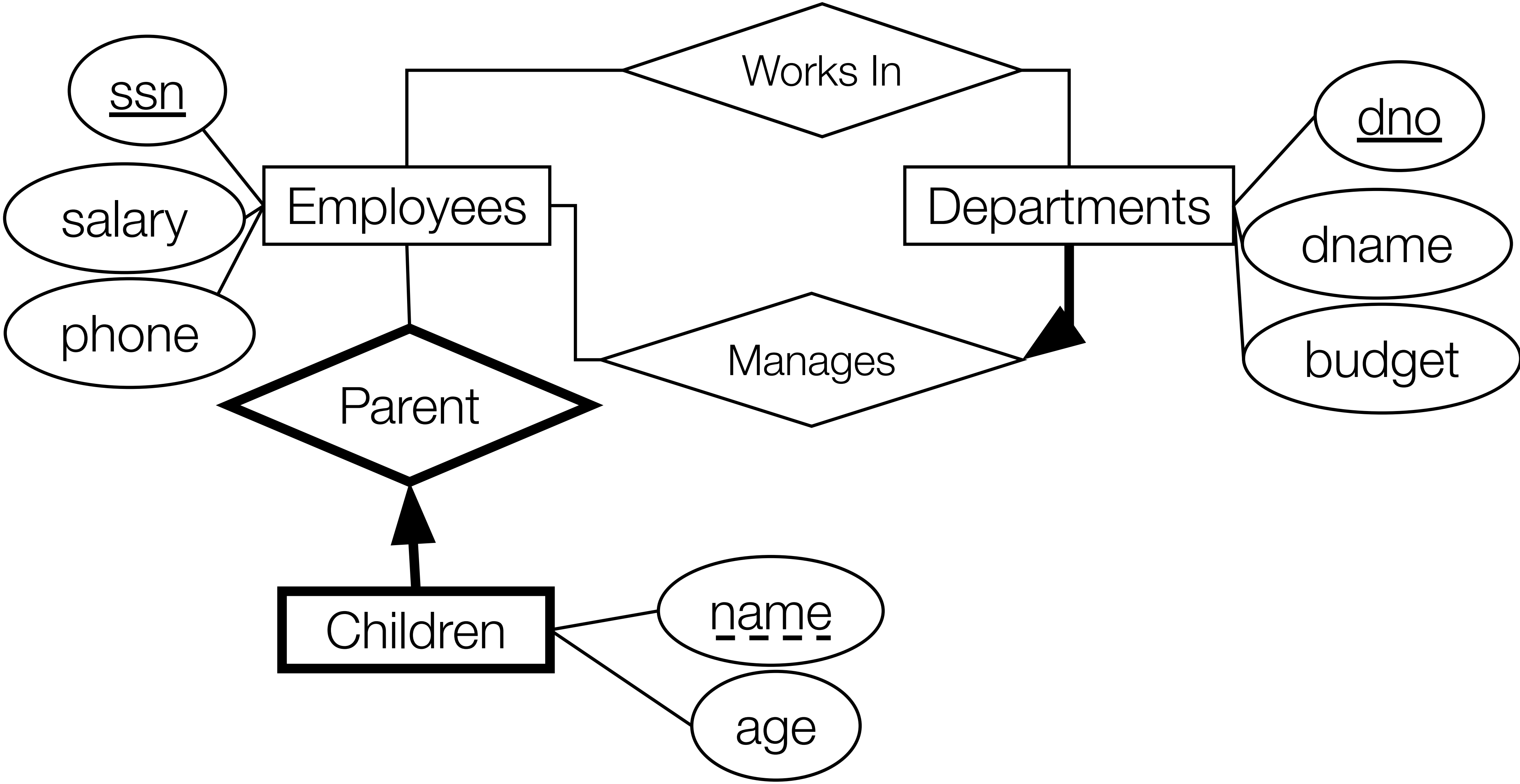


On your own

# Exercise – Part 1

- A **company database** needs to store information about
  - **employees** (identified by ssn, with salary and phone attributes),
  - **departments** (identified by dno, with dname and budget attributes), and
  - **children of employees** (with name and age attributes).
- Employees work in (zero or more) departments
- Each department is managed by exactly one employee
- A child must be identified uniquely by name when the parent (who is an employee; assume 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

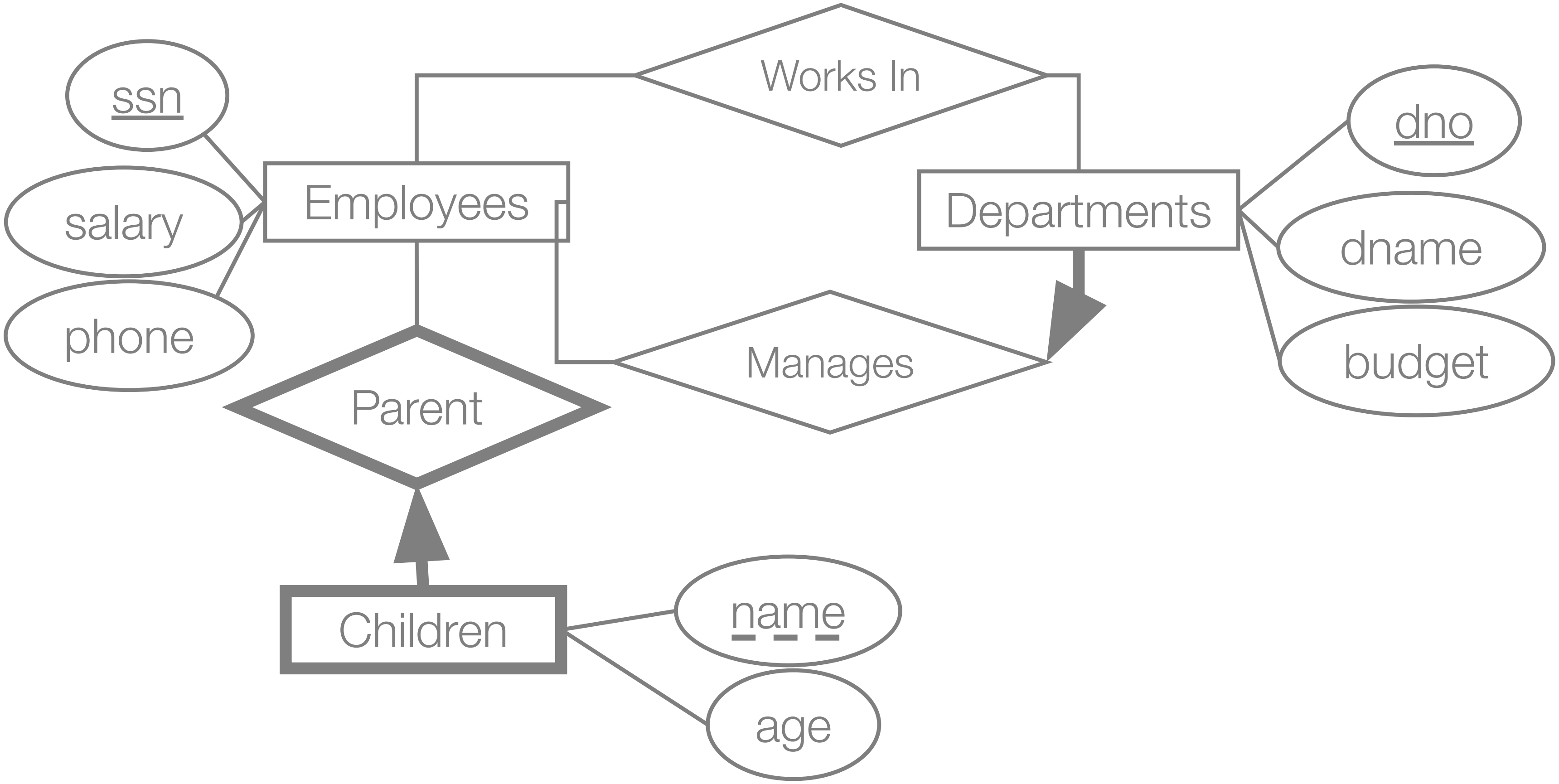
# ER Diagram (one solution)



On your own

# Exercise – Part 2

- Write SQL statements to create the corresponding relations, and to capture as many of the constraints as possible.



# SQL DDL – One solution

```
CREATE TABLE Employees (
ssn INTEGER,
salary REAL,
phone CHAR(10),
PRIMARY KEY(ssn))
```

```
CREATE TABLE Works (
ssn INTEGER,
dno INTEGER,
PRIMARY KEY (ssn, dno),
FOREIGN KEY (ssn)
   REFERENCES employees,
FOREIGN KEY (dno)
   REFERENCES departments)
```

```
CREATE TABLE Departments (
dno INTEGER,
dname CHAR(20),
budget real,
manager INTEGER NOT NULL,
PRIMARY KEY (dno),
FOREIGN KEY (manager)
    REFERENCES employees(ssn))
```

```
CREATE TABLE Children (
name CHAR(20),
age REAL,
parent INTEGER NOT NULL,
PRIMARY KEY(name, parent),
FOREIGN KEY(parent)
   REFERENCES employees
   ON DELETE CASCADE)
```

# Integrity Constraints

- Describe conditions that must be satisfied by every legal instance
- Types of integrity constraints
  - Domain constraints
  - Primary key constraints
  - Foreign key constraints
  - General constraints

# Table Constraints

- More general than key constraints
- Can use a query to express constraint
  - Constraints checked each time table updated
  - CHECK constraint always true for empty relation

```
CREATE TABLE Athlete
(aid INTEGER PRIMARY KEY,
 name CHAR(30),
 age INTEGER,
 country CHAR(20),
 sport CHAR(20)),
CHECK  ( age >= 18
        AND age <= 80 )
```

```
CREATE TABLE Compete
   (aid INTEGER,  oid INTEGER,
    PRIMARY KEY  (aid, oid),
    FOREIGN KEY (aid)
      REFERENCES Athlete
    CONSTRAINT  noRussia
      CHECK  (`Russia' <>
                ( SELECT  O.city
                FROM  Olympics O
                WHERE  O.oid=oid)))
```

cross-table constraint

# Try it out in sqlplus or sqlite

```
CREATE TABLE Sailors
    ( sid     INTEGER,
   sname   CHAR(10),
   rating  INTEGER,
   age     REAL,
   PRIMARY KEY  (sid),
      CHECK (rating >= 1 AND rating <= 10));

INSERT INTO Sailors VALUES (1, 's1', 11, 25);
```

Do you get a constraint violation error?

# Try it out in sqlplus or sqlite (Example in book 5.7)

```
CREATE TABLE Sailors
     ( sid      INTEGER,
    sname    CHAR(10),
    rating   INTEGER,
    age      REAL,
    PRIMARY KEY  (sid),
        CHECK (rating >= 1 AND rating <= 10));

INSERT INTO Sailors VALUES (1, 's1', 11, 25);
```

Do you get a constraint violation error?

```
> Error: CHECK constraint failed: Sailors
```

# More general CHECK

- Interlake boats cannot be reserved.
- Note: these are not supported in Oracle or SQLite

```
CREATE TABLE  Reserves
  ( sname  CHAR(10),
 bid     INTEGER,
 day     DATE,
 PRIMARY KEY (bid,day),
 CONSTRAINT  noInterlakeRes
 CHECK   ('Interlake' NOT IN
           ( SELECT  B.bname
         FROM  Boats B
         WHERE  B.bid=bid)))
```

# Constraints Over Multiple Relations

- For general constraint over multiple tables, use an assertion

*Number of boats plus number of sailors is < 100*

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

# Practical Considerations

- CHECK with subqueries and ASSERTIONS
  - Part of SQL standard (since 1992?)
  - But, they are not supported in many major databases
    - Because it's only been 30 years lolol
  - **Main concern:** Performance issues; CHECK or ASSERTION constraints over multiple tables are very slow
- Instead: Triggers
  - Most major database systems support them
  - Triggers are procedural

# Active Databases & Triggers (5.8 in the book)

**Trigger:** Procedure that starts automatically if specified changes occur to the DBMS

- Three parts:
  - Event (activates the trigger)
  - Condition (test that is run when the trigger is activated)
  - Action (what happens if the trigger runs)
    - Before and After Triggers
- Trigger Execution
  - Row-level Triggers: Once per modified row
  - Statement-level Triggers: Once per SQL statement

# Triggers

- When condition is checked, success/failure can be used to trigger arbitrary actions.
- Used for many things:
  - Often used to fill out fields in a form
  - Check complex actions (such as credit limit in a shopping application)
  - Check preferred customer status
  - Generate logs for auditing and security checks.
- Must specify when to check condition
  - Usually just before or just after an update

# Try it out

- Modify athlete_create.sql so that it has the UPDATE and DELETE constraints in COMPETE relation as in the previous slide.
  - Modified file available in athlete_modified.sql.
- Try the following and check COMPETE:

```
sqlite> DELETE FROM Athlete WHERE name='Michael
Phelps';
sqlite> UPDATE Athlete SET aid=5 WHERE aid=4;
```

Implementation Note: In SQLite, make sure you issued “`PRAGMA foreign keys = ON;`” command to enforce foreign key constraints. By default, SQLite ignores them for backward compatibility.

# Implementation Notes

- Oracle's sqlplus:
  - You cannot use NO ACTION constraints. They are the default and thus not needed.
  - String literals like 'USA' must use **single quotes**, not double quotes
- SQLite:
  - You need `PRAGMA foreign_keys = ON;` to enforce foreign key constraints. This is for backward compatibility.

# Oracle Trigger Example

- First trigger executed before the activating statement, second executes after the activating statement.
- In combination with:
  - `FOR EACH ROW` - execute once per modified record
  - (default) - execute once per activating statement
- Activating statements:
  - `INSERT`
  - `DELETE`
  - `UPDATE`

statement-level

```
CREATE TRIGGER init_count
BEFORE INSERT ON Student       /* Event */
   DECLARE
    count INTEGER; -> declare
   BEGIN                       /* Action */
    count := 0
   END;

CREATE TRIGGER incr_count
AFTER INSERT ON Student        /* Event */
   FOR EACH ROW  row-level-trigger
   WHEN (new.age >= 18)        /* Condition */
   BEGIN                   /* Action */
    count := count + 1;
   END;
```

# Recall CASCADE constraints

```
CREATE TABLE Athlete
(aid INTEGER PRIMARY KEY,
 name CHAR(30),
 country CHAR(20),
 sport CHAR(20));
```

```
CREATE TABLE Olympics
(oid INTEGER PRIMARY KEY,
 year INTEGER,
 city CHAR(20));
```

```
CREATE TABLE Compete
    (aid INTEGER,
     oid INTEGER,
     PRIMARY KEY  (aid, oid),
     FOREIGN KEY (aid) REFERENCES Athlete
        ON DELETE CASCADE
     FOREIGN KEY (oid) REFERENCES Olympics
     );
```

# CASCADE Using Triggers

```
CREATE TABLE Compete
    (aid INTEGER,
     oid INTEGER,
     PRIMARY KEY  (aid, oid),
     FOREIGN KEY (aid) REFERENCES Athlete
     FOREIGN KEY (oid) REFERENCES Olympics);

CREATE OR REPLACE TRIGGER cascade_on_delete
AFTER  DELETE ON Athlete
FOR EACH ROW
BEGIN
  DELETE FROM Compete
  WHERE Compete.aid = :OLD.aid;
END;
```

# Trying out triggers

- Try out the file athlete_trigger_cascade.sql in sqlplus
- Also try it out in sqlite to see if it supports triggers the same way
- Try removing a row from athlete. Does it cascade to Compete via the trigger?
- Try dropping the trigger:
  - `DROP TRIGGER` *`triggername;`*
- What happens now on deleting a row from Athlete?

# Triggers: Pitfalls and Pain

- Triggers can be recursive!
  - Chain of triggers can be hard to predict, which makes triggers difficult to understand and debug
- Errors with “mutating” table
  - A table that is currently being modified by an UPDATE, DELETE, or INSERT statement, or a table that might be updated by the effects of a DELETE CASCADE constraint
  - The session that issued the triggering statement cannot query or modify a mutating table
  - Used to prevent a trigger from seeing **inconsistent data**