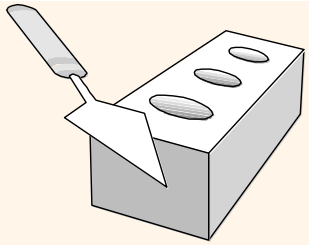
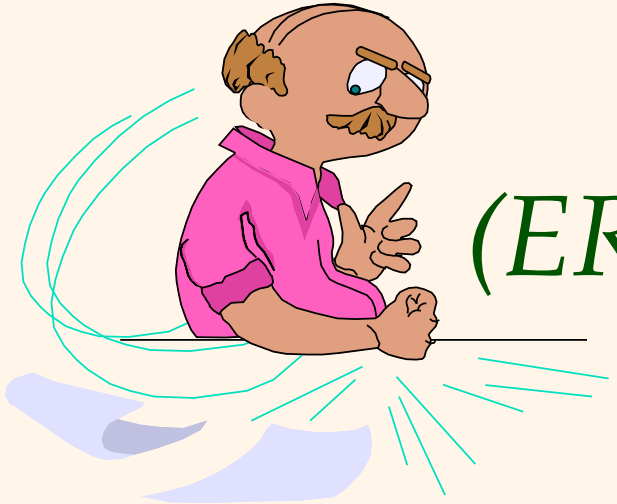


Introduction to Data Management

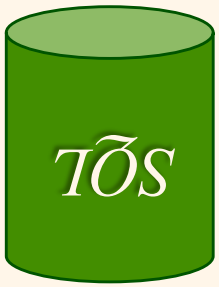
**** The "Flipped" Edition ****



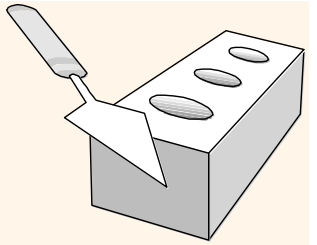
Lecture #4

(ER Based DB Design)

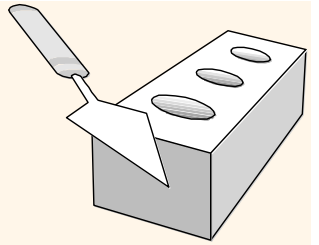
Instructor: Mike Carey
mjcarey@ics.uci.edu



Today's Notices

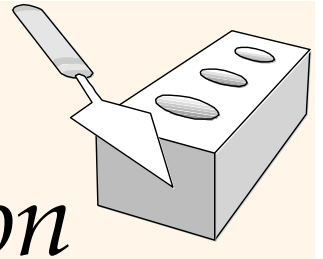


- Continue watching the wiki page:
  - http://www.ics.uci.edu/~cs122a/
- And the Piazza page:
  - piazza.com/uci/fall2021/cs122aeecs116/home
- Partners: He/she is your "brainstorming buddy"!
  - Individual HW submissions (***not*** team submissions)
  - See the partner-related part of the first HW assignment
- HW#1 is out! (Post questions on Piazza if needed)
  - ***SWOOSH.com*** (watch out, Zoom & Piazza!)
  - Please don't discuss ***solutions*** on Piazza!
- Quiz timing (for Quiz > 0)
  - Available from Wed 3PM – Fri 3PM each week (***NEW PLAN!)***



Relational Database: Definitions

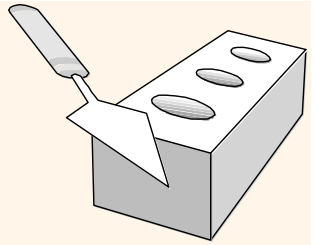
- Relational database: a set of *relations*
- Relation: consists of 2 parts:
  - ***Instance***: a table, with rows and columns. #rows = *cardinality*, #fields = *degree* or *arity*.
  - ***Schema***: specifies name of relation, plus name and type of each column.
    - E.g, Students (*sid*: string, *name*: 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) in the pure relational model (*vs.* reality of SQL ☺)



Example Instance of Students Relation

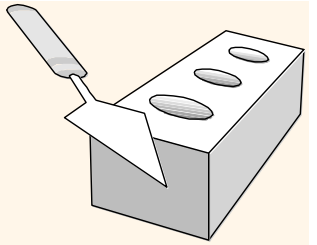
sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- Cardinality = 3, degree = 5, all rows distinct



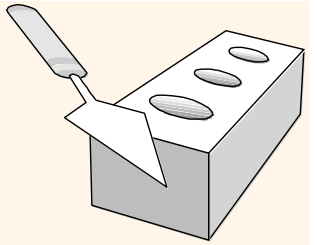
Relational Query Languages

- 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 (and set-based) semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.



SQL Query Language (Preview)

- Developed by IBM (System R) in the 1970s
- Need for a standard, since it is used by many vendors (Oracle, IBM, Microsoft, …)
- ANSI/ISO Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, very widely supported)
  - SQL-99 (major extensions, current standard)



SQL Query Language (Preview)

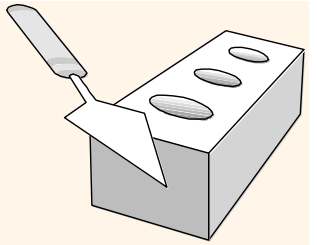
- To find all 18-year-old students, we can write:

```
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

- 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"
```

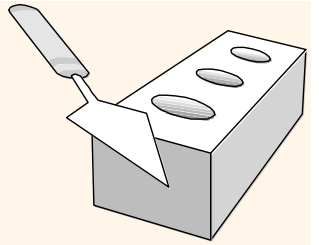
Given the following instances of Students and Enrolled:

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

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

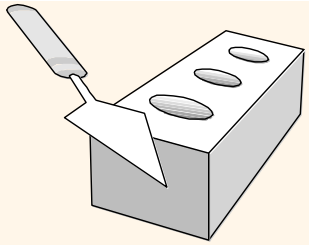
We will get:

S.name	E.cid
Smith	Topology112



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 (centrally), too!



Primary Key Constraints

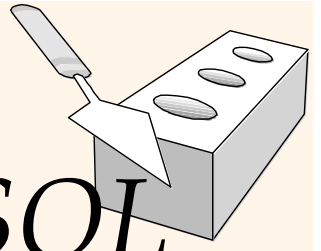
- A set of fields is a *key* for a relation if :
  1. No two distinct tuples can have the same values in all key fields, and
  2. This is not true for any subset of the key.

     Part 2 false? In that case, this is a "*superkey*".

     If there's > 1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.

     The others are referred to as *candidate keys*.

- E.g., *sid* is a key for Students. (What about *name*?) The set {*sid, gpa*} is a superkey.



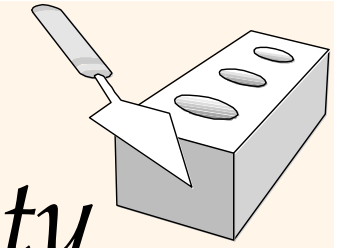
Primary and Candidate Keys in SQL

- Possibly several *candidate keys* (specified using UNIQUE), but one is chosen as the *primary key*.

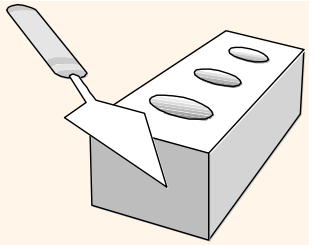
- Used carelessly, an IC can prevent the storage of database instances that arise in practice!
- "For a given student + course, there is a single grade." vs. "Students can take only one course and receive a single grade for that course; further, no two students in a course may ever receive the same grade."

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

Foreign Keys, Referential Integrity



- *Foreign key*: Set of fields in one relation used to "refer" to a tuple in another relation. (Must refer to the primary key of the other 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.



Foreign Keys in SQL

- Ex: Only students listed in the Students relation should be allowed to enroll for courses.

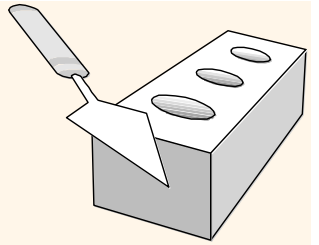
```
CREATE TABLE Enrolled
  (sid VARCHAR(20), cid VARCHAR(20), grade CHAR(2),
   PRIMARY KEY (sid, cid),
   FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

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

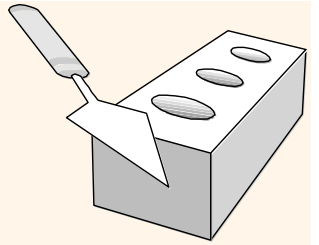
Students

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

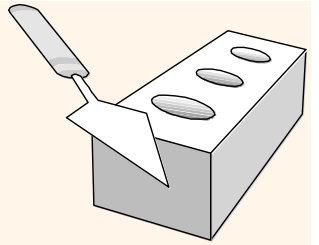
- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a Students tuple is deleted?
  - Also delete all Enrolled tuples that refer to it. Or...
  - Disallow deletion of a Students tuple if it 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 if primary key of Students tuple is updated.



Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
  - Default is NO ACTION (*delete/update is rejected*)
  - CASCADE (also delete all tuples that refer to the being-deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of the refering tuples)

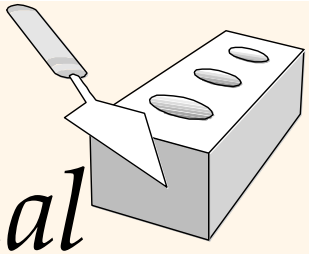
```
CREATE TABLE Enrolled
  (sid VARCHAR(20),
   cid VARCHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid),
   FOREIGN KEY (sid)
     REFERENCES Students
        ON DELETE CASCADE
        ON UPDATE SET DEFAULT )
```



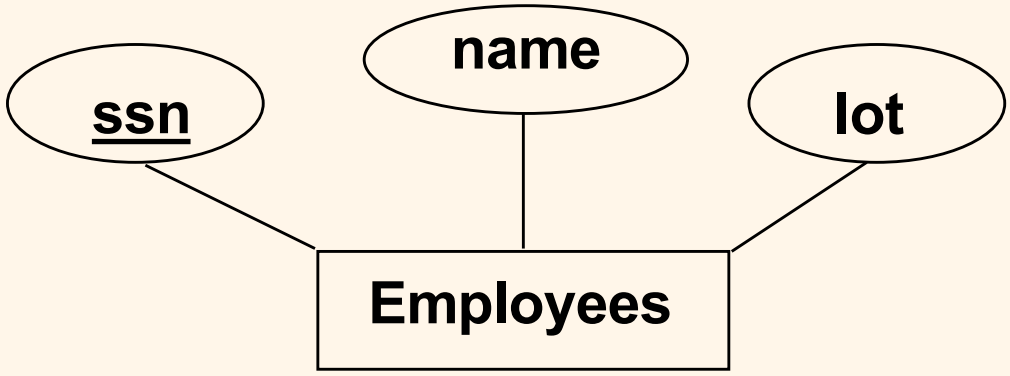
Where Do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations (perhaps via an E-R schema)
- 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!
  - For 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.

Logical DB Design: ER to Relational



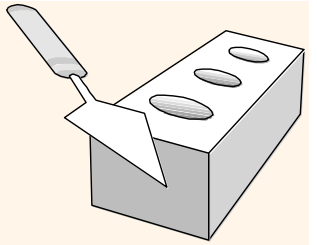
- Entity sets to tables:




```
CREATE TABLE Employees
    (ssn CHAR(11),
    name VARCHAR(20),
    lot  INTEGER,
    PRIMARY KEY  (ssn))
```

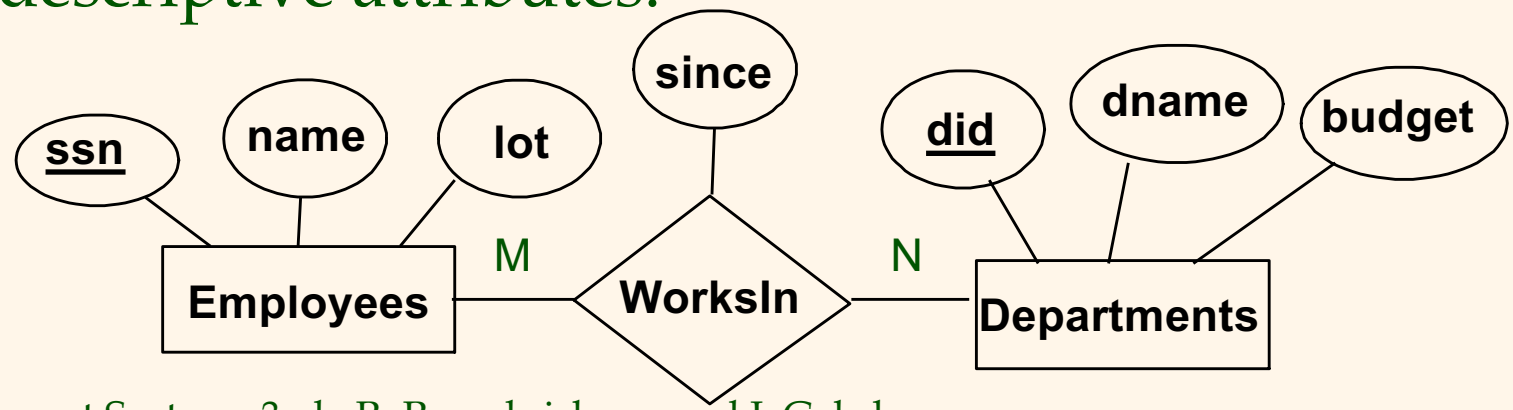
- Notes:
  - PRIMARY KEYs are NOT NULL by default
  - All other fields are NULL(-able) by default
  - One can say NOT NULL

Relationship Sets to Tables

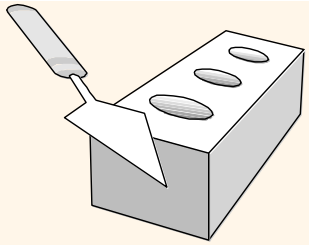


- In translating a relationship set to a relation, attributes of the relation should include:
  - Keys for each participating entity set (as foreign keys).
    - This set of attributes forms a *superkey* for the relation.
  - All descriptive attributes.

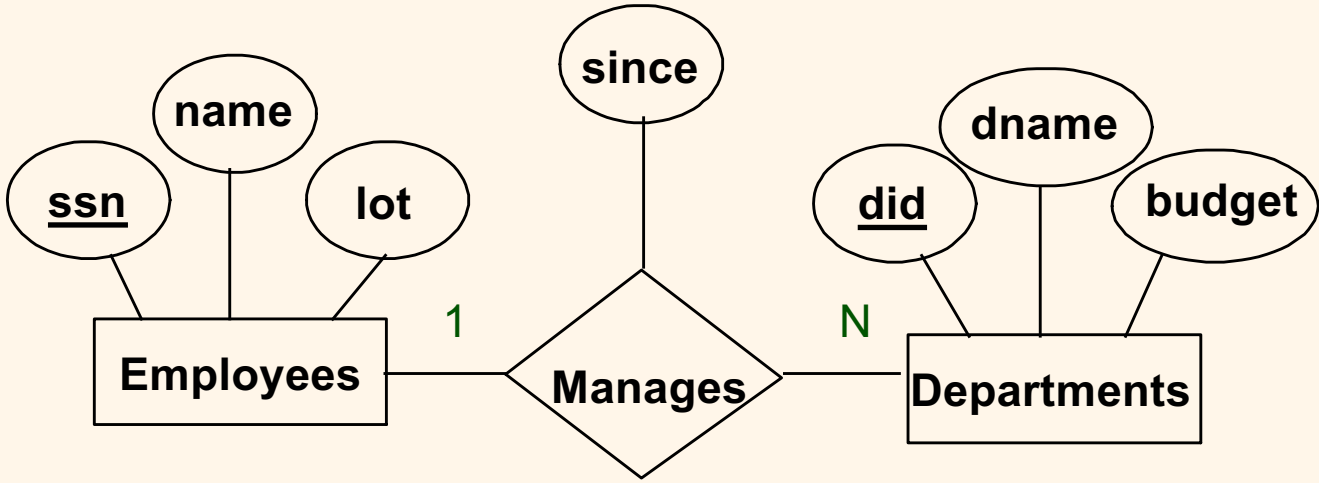
```
CREATE TABLE Works_In(
  ssn  CHAR(11),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
     REFERENCES Employees,
  FOREIGN KEY (did)
     REFERENCES Departments)
```

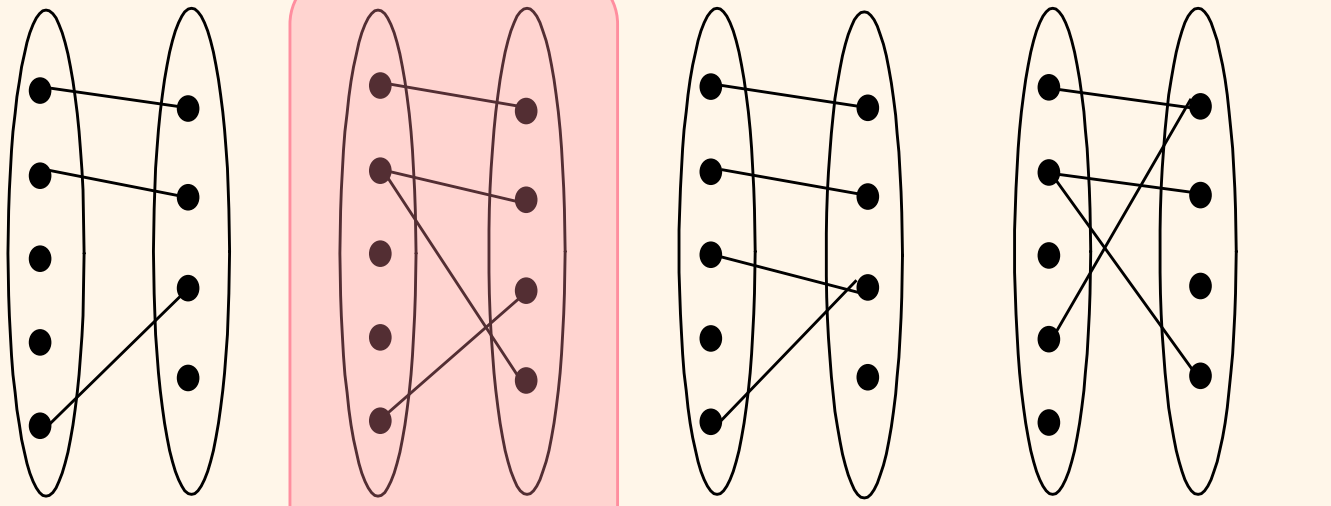


Key Constraints (Review)



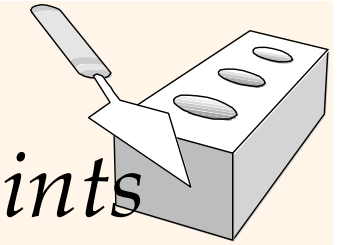
- Each dept has at most one manager, according to the *key constraint* on Manages.





Translation to relational model?

Translating ER Diagrams with Key Constraints



- Map the relationship to a table (Manages):
  - Note that did (alone) is the key!
  - Still separate tables for Employees and Departments.
- But, since *each* department has a *unique* manager, we could choose to fold Manages right into Departments.

(Q: Why do this...?)

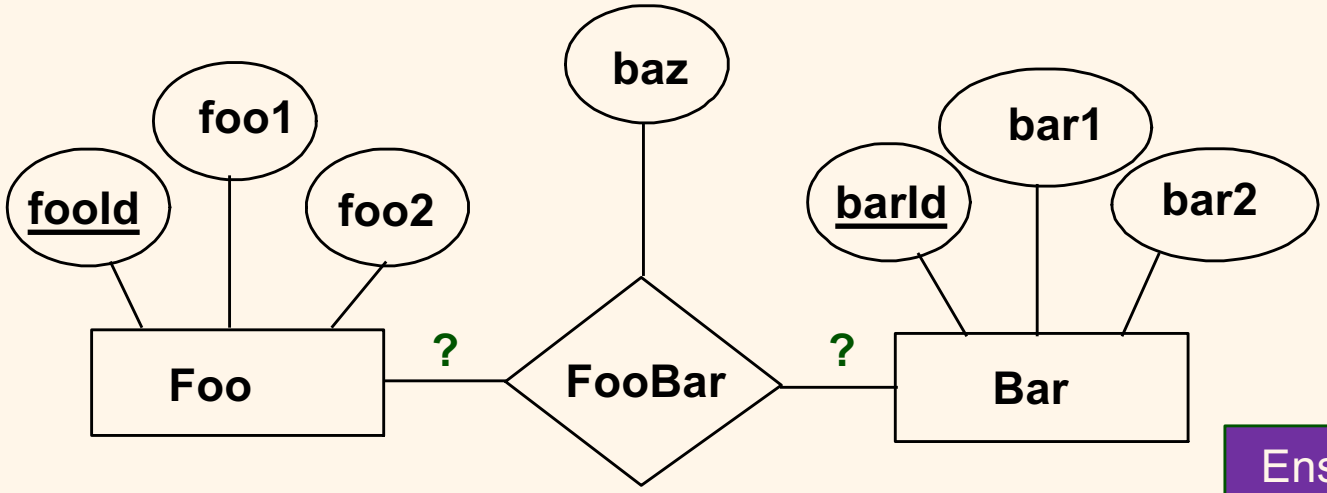
```
CREATE TABLE Manages (
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments)
```

vs.

```
CREATE TABLE Departments2 (
  did INTEGER,
  dname VARCHAR(20),
  budget REAL,
  mgr_ssn CHAR(11),
  mgr_since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (mgr_ssn) REFERENCES Employees)
```

***Note**: The relationship info has been pushed to the **N-side**'s entity table!*

Properly Reflecting Key Constraints



- So what are the translated relationship table's keys (*etc.*) when…

  FooBar is M:N? → FooBar(fooId, barId, baz)

  FooBar is N:1? → FooBar(fooId, barId, baz)

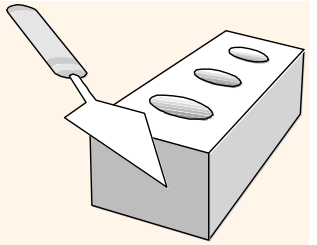
  Foobar is 1:N? → FooBar(fooId, barId, baz)

  Foobar is 1:1? → FooBar(fooId, barId, baz) *(Note: unique)*

Ensures unique Foo/Bar pairs

Ensures one Bar per Foo entity

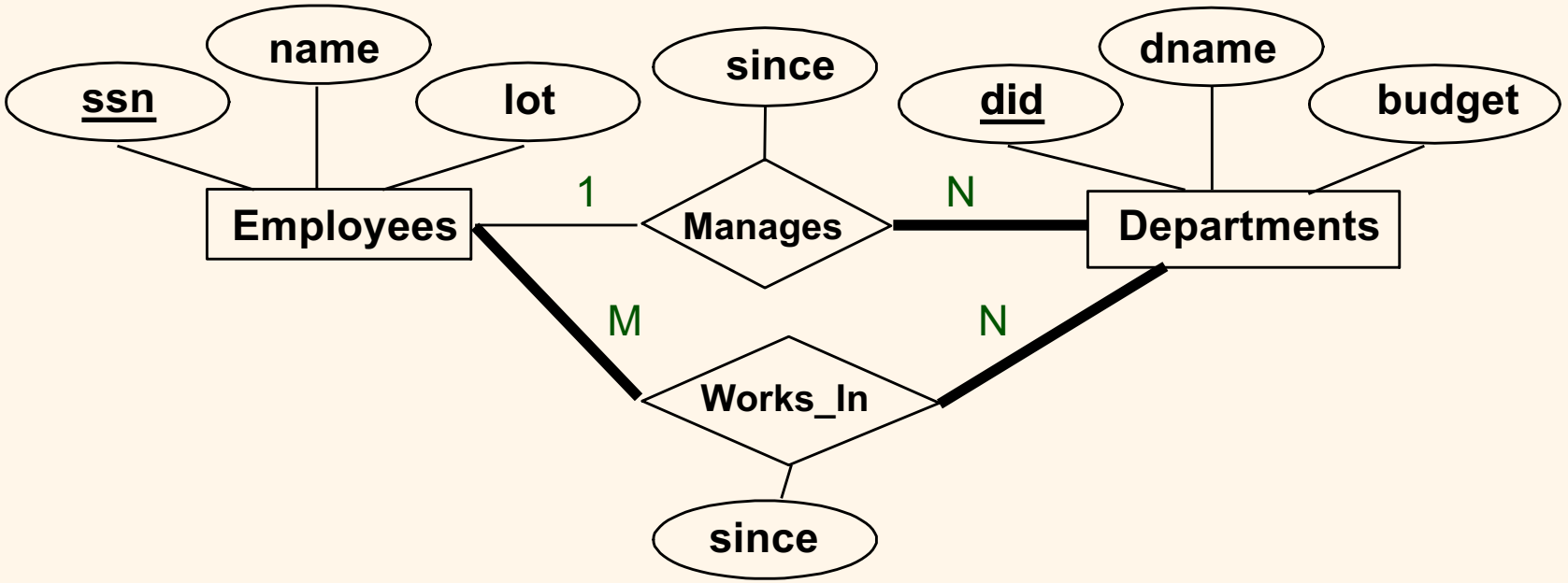
Review: Participation Constraints

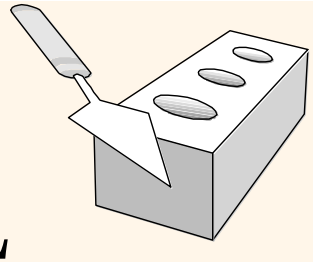


- Does every department have a manager?

  If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total* (vs. *partial*).

  - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!!)

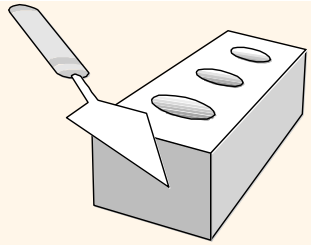




Participation Constraints in SQL

- We can capture participation constraints involving the ***N-side*** entity set in a binary relationship, but little else (without resorting to the use of *triggers*).

```
CREATE TABLE Department2 (
   did  INTEGER,
   dname  VARCHAR(20),
   budget  REAL,
   mgr_ssn CHAR(11) NOT NULL,
   mgr_since  DATE,
   PRIMARY KEY  (did),
   FOREIGN KEY  (mgr_ssn) REFERENCES Employees,
     ON DELETE NO ACTION*)          (*or: RESTRICT)
```



To Be Continued...