Database Management Systems

Ehsan Noei
e.noei@utoronto.ca

UNIVERSITY OF TORONTO

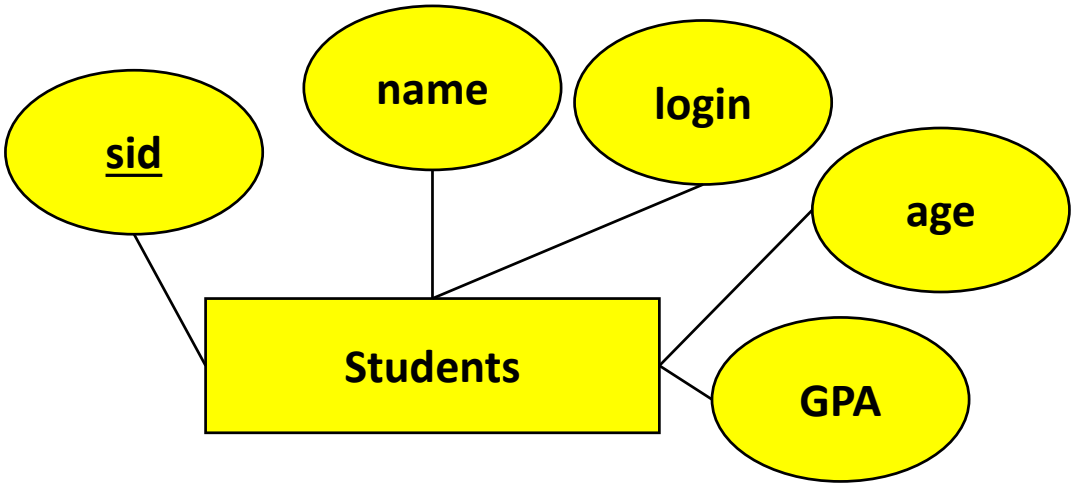
Relational Model

- The main construct for representing data in the relational model is a relation.
- Relational database: a set of relations.

Relational Database: Definitions

- *Relation:* consists 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)

Students (sid: string, *name:* string, *login:* string, *age:* integer, *gpa:* real)



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
- Do all columns in a relation instance have to be distinct?

Degree and Cardinality

- Degree: Degree of a relation is the number of fields.
- Cardinality: Cardinality of a relation instance is the number of tuples in it.

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.

```
SELECT  S.name, E.cid, E.grade
FROM  Students S, Enrolled E
WHERE  S.sid=E.sid AND S.age >18 AND E.cid=‘ITCS61  60'
```

The SQL Query Language

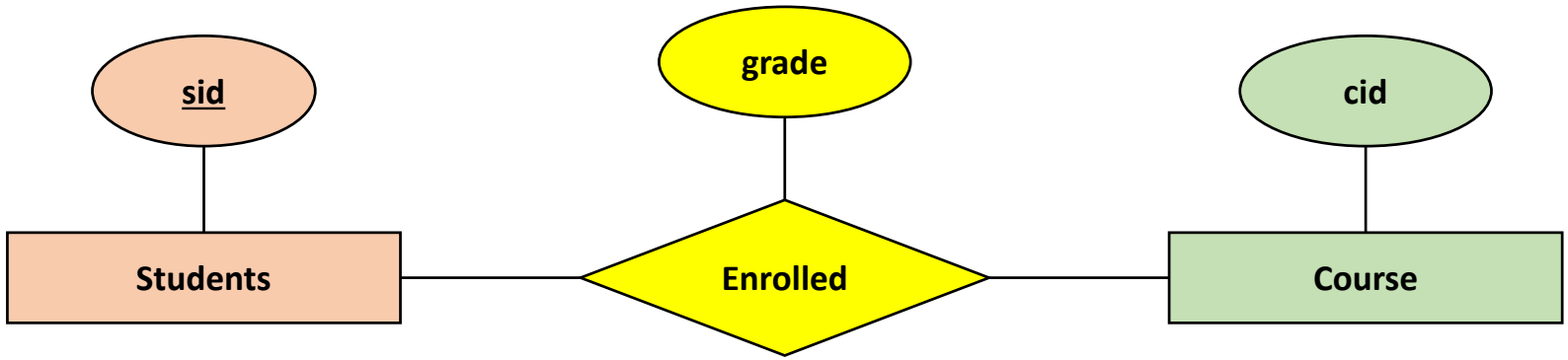
- Developed by IBM (system R) in the 1970s
- Need for a standard since it is used by many vendors
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, current standard)
  - SQL-99 (major extensions)

- Exact Numerics:
  - INTEGER
  - SMALLINT
  - BIGINT
  - NUMERIC
  - DECIMAL
- Approximate Numerics:
  - REAL
  - DOUBLE PRECISION
  - FLOAT
  - DECFLOAT
- Binary Strings:
  - BINARY
  - BINARY VARYING
  - BINARY LARGE OBJECT
- Boolean:
  - BOOLEAN
- Character Strings:
  - CHARACTERCHARACTER VARYING (VARCHAR)
  - CHARACTER LARGE OBJECT
  - NATIONAL CHARACTER
  - NATIONAL CHARACTER VARYING
  - NATIONAL CHARACTER LARGE OBJECT
- Datetimes:
  - DATE
  - TIME WITHOUT TIMEZONE
  - TIMESTAMP WITHOUT TIMEZONE
  - TIME WITH TIMEZONE
  - TIMESTAMP WITH TIMEZONE
- Intervals:
  - INTERVAL DAY
  - INTERVAL YEAR
- Collection Types:
  - ARRAY
  - MULTISET
- Other Types:
  - ROW
  - XML

Creating Relations in SQL

- Creates the Students relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

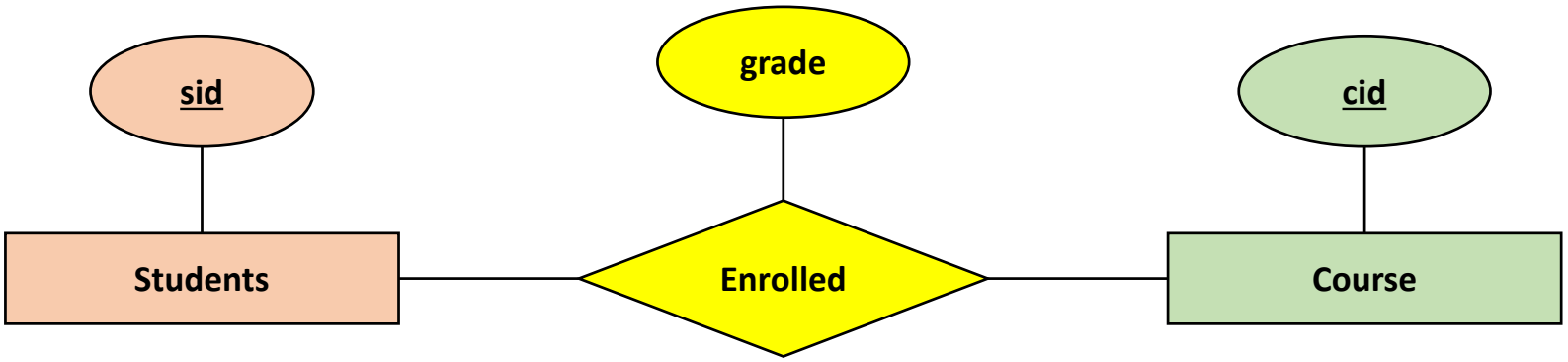
```
CREATE TABLE Students
    (sid CHAR(20),
     name CHAR(20),
     login CHAR(10),
     age INTEGER,
     gpa REAL)
```



Creating Relations in SQL

- As another example, the Enrolled table holds information about courses that students take.

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2))
```



Destroying and Altering Relations

DROP TABLE Students

- Destroys the relation Students. The schema information *and* the tuples are deleted.

ALTER TABLE Students
ADD COLUMN firstYear: integer

- The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53699, 'Green ', 'green@ee', 18, 3.5)
```

- More inserts:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53666, 'Jones', 'jones@cs', 18, 3.4)
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith ', 'smith@eecs', 18, 3.2)
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53650, 'Smith ', 'smith@math', 19, 3.8)
```

Adding and Deleting Tuples (continued)

- Students relation after inserts:

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
53600	Green	green@ee	18	3.5

Adding and Deleting Tuples (continued)

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

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

- Students instance after delete:

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53600	Green	green@ee	18	3.5

The SQL Query Language

- 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 FROM Students S
```

Adding and Deleting Tuples (continued)

- Insert tuples into the Enrolled instance:

```
INSERT INTO Enrolled (sid, cid, grade)
VALUES ('53831', 'Carnatic 101', 'C')
INSERT INTO Enrolled (sid, cid, grade)
VALUES ('53831', 'Reggae 203', 'B')
INSERT INTO Enrolled (sid, cid, grade)
VALUES ('53650', 'Topology 112', 'A')
INSERT INTO Enrolled (sid, cid, grade)
VALUES ('53666', 'History 105', 'B')
```

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='B'
```

Given the following instance of Enrolled (is this possible if the DBMS ensures referential integrity?):

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

we get:

S.name	E.cid
Jones	History 105

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!

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.

Primary and Candidate Keys in SQL

- Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.

- "For a given student and 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 receive the same grade."
- Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
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) )
```

Primary and Candidate Keys in SQL (continued)

- For Students relation with SID as the primary key

```
CREATE TABLE Students
    (sid CHAR(20),  name  CHAR(20),
     login CHAR(10), age INTEGER,
     gpa REAL, PRIMARY KEY  (sid) )
```

- Are there any separate fields or combinations of fields which also are candidates for primary key?
  - How about login?
  - How about age?
  - How about age & gpa?

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 in SQL

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

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

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

Foreign Keys in SQL (continued)

- Creates the customer information relation

```
CREATE TABLE Customer_Info
   (name CHAR(20),  addr CHAR(40),
    phone CHAR(10), email char (40),
     PRIMARY KEY  (name, addr))
```

- Now create the bank account relation with a foreign key

```
CREATE TABLE Bank_Acct
      (acct CHAR (4), name CHAR (20),
       address char (40), balance REAL,
       PRIMARY KEY  (acct) ,
       Foreign Key (name, address) references Customer_Info)
```

Foreign Key

- Can a foreign key refer to the same relation?
- Example
  - Each student may have a partner who must be also a student.
  - How about a student who does not have partner?
  - NULL introduced here.

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.
  - Disallow deletion of a Students tuple that 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/92

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

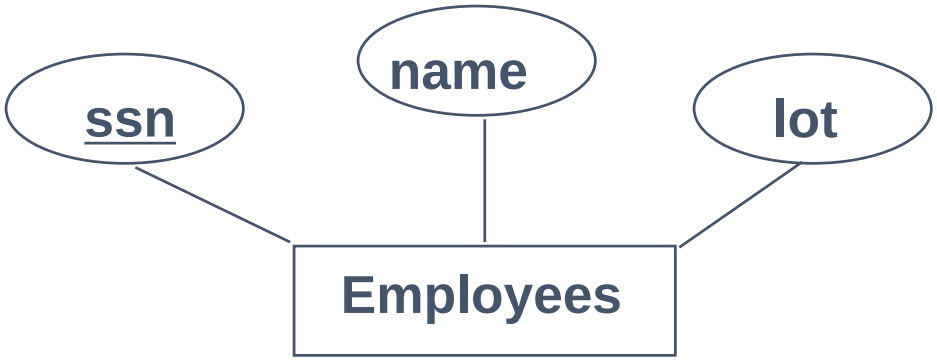
```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid)
     REFERENCES Students
     ON DELETE CASCADE
     ON UPDATE SET DEFAULT )
```

Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- 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!
  - 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.

Logical DB Design: ER to Relational

- Entity sets to tables.



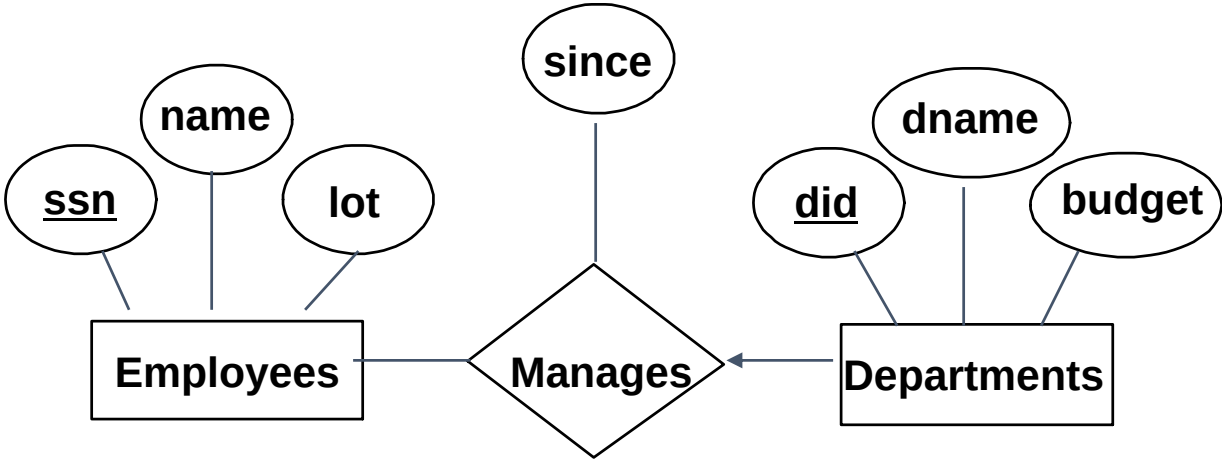

```
CREATE TABLE Employees
    (ssn CHAR(11),
    name CHAR(20),
    lot  INTEGER,
    PRIMARY KEY  (ssn))
```

Relationship Sets to Tables

- In translating a relationship set to a relation, attributes of the relation must 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(1),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
     REFERENCES Employees,
  FOREIGN KEY (did)
     REFERENCES Departments)
```

Review: Key Constraints



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

Translating ER Diagrams with Key Constraints

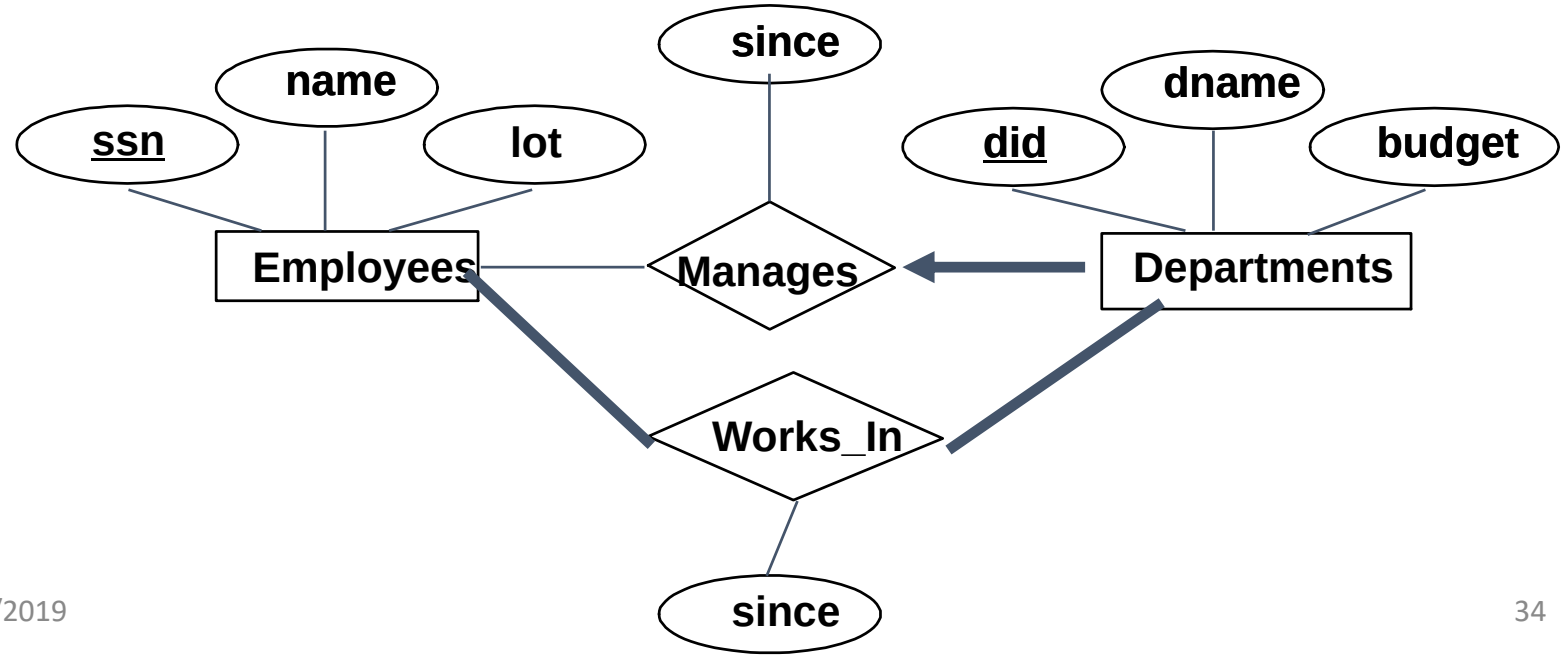
- Map relationship to a table:
  - Note that did is the key now!
  - Separate tables for Employees and Departments.
- Since each department has a unique manager, we could instead combine Manages and Departments.

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

```
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees)
```

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 one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```
CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
      ON DELETE NO ACTION)
```