Relational Model

Overview

- **ER Model** (last lecture)
  - Conceptual design (High-level database design)
  - *No direct relationship with database technologies*

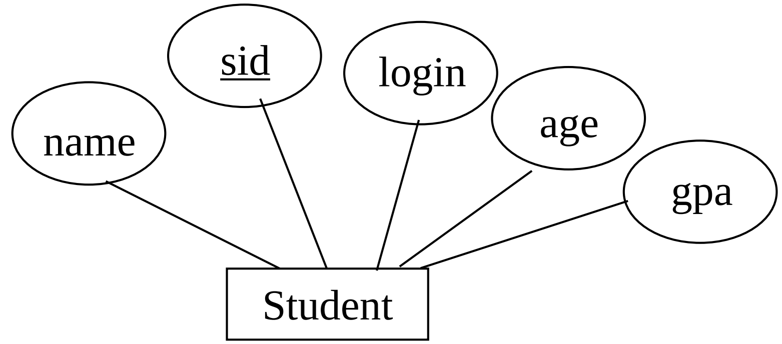
- **Relational Data Model** (this chapter)
  - Logical design
  - Maps the *conceptual requirements* into the *data model* associated with a specific database management system.

What is a Relation?

- A **relation** is a more concrete construction of the ER diagram

- A relation is (just!) a table!

- We will use **table** and **relation** interchangeably



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

What is a Relational Database?

- **Relational database:** a set of **relations**

- **Relation:** a named data table consisting of two parts:
  - **Schema**
    - A list of column/attribute names with their **data types**

Students(*sid*: char(20), *name*: char(20), *login*: char(10), *age*: integer, *gpa*: real)

  - **Instance**
    - Made up of zero or more tuples (or called *records*, *rows*)

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

What is a Relational Database?

- **Relation:** concept in relational data model:
  - **Schema**
    - specifies name of relation
    - specifies name and type of each attribute
  - **Instance**
    - a table of tuples (or called *records*, *rows*)
    - attributes (or called *fields*, *columns*).
    - Number of tuples = *cardinality*
    - Number of attributes = *degree*

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

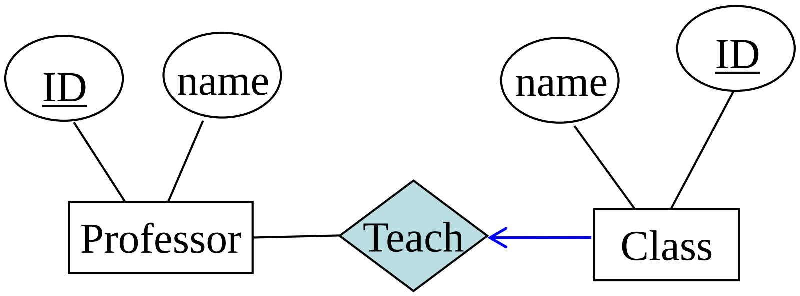
Cardinality = 3. and Degree = 5.

- **No two tuples are completely identical in a relation!**

Relation vs. Relationship

Relationship:

- concept in ER model
- Describes relationship between entities



Relation:

- concept in relational data model
- Essentially a table (a set of tuples)

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

The SQL Query Language

- A major strength of the relational model, which supports simple and powerful *querying* of data.

- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.

- SQL was developed by IBM in the 1970s.

Create Relations in SQL

- Creates the **Students** relation.
  - Observe that the type of each attribute is specified.
  - When each tuple is added/modified, the DBMS must ensure that the tuple follows the type for each attribute.

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

Insert and Delete Tuples in SQL

- Can insert a single tuple using:

```
INSERT
INTO  Students (sid, name, login, age, gpa)
VALUES ('53698', 'Bob', 'bob@comp', 18, 3.2)
```

  - The INTO clause can be omitted if each value in VALUES corresponds to the correct attribute.
  - Require single quotes for strings, e.g., 'Bob'.

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

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

Data Types in SQL

- Different RDMS may support different data types
  - **MySQL**: CREATE TABLE my_table(date_col DATE)
    - E.g., '2024-01-25' (YYYY-MM-DD)
  - **PostgreSQL**: CREATE TABLE my_table(date_col TIMESTAMP)
    - E.g., '2024-01-25 10:23:54' (both date and time)

- MySQL as an example
  - **String data types**: CHAR(size), fixed length string
  - **Numeric data types**: INT(size), SMALLINT(size), FLOAT(size,d), FLOAT(p)
  - **Date and time data types**: DATE, YEAR

Creating Relations in SQL

- **char(n)**. Fixed length character string, with user-specified length n.
- **varchar(n)**. Variable length character strings, with user-specified maximum length n.
- **Integer(int)**. Integer (a finite subset of the integers that is machine-dependent).
- **smallint**. Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d)**. Fixed point number, with user-specified precision of p digits, with d digits to the right of decimal point. (ex., numeric(3,1), allows 44.5 to be stores exactly, but not 444.5 or 0.32)
- **real, double precision**. Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n)**. Floating point number, with user-specified precision of at least n digits.

Question 1

- Create the following relation using SQL.
  - RK: integer
  - GP: games played
  - MPG: minutes per game

Points Per Game Leaders - Qualified

RK	PLAYER	TEAM	GP	MPG
1	Stephen Curry, PG	GS	79	34.2
2	James Harden, SG	HOU	82	38.1
3	Kevin Durant, SF	OKC	72	35.8
4	DeMarcus Cousins, C	SAC	65	34.6
5	LeBron James, SF	CLE	76	35.6
6	Damian Lillard, PG	POR	75	35.7
7	Anthony Davis, PF	NO	61	35.5
8	Russell Westbrook, PG	OKC	80	34.4

Question 2

- Create the following Stocks relation using SQL, where Symbol, Name, Last, Change, and % Chg are represented by 20 characters, 100 characters, real number, real number, and real number, respectively.

Figure 3.31 Stock Quotations

Symbol	Name	Last	Change	% Chg
$COMPX	Nasdaq Combined Composite Index	1,400.74 ▼	-4.87	-0.35%
$INDU	Dow Jones Industrial Average Index	9,255.10 ▼	-19.80	-0.21%
$INX	S&P 500 INDEX	971.14 ▼	-5.84	-0.60%
ALTR	Altera Corporation	13.45 ▼	-0.450	-3.24%
AMZN	Amazon.com, Inc.	15.62 ▲	+0.680	+4.55%
CSCO	Cisco Systems, Inc.	13.39 ▼	-0.280	-2.05%
DELL	Dell Computer Corporation	24.58 ▼	-0.170	-0.69%
ENGCX	Enterprise Growth C	14.60 ▼	-0.210	-1.42%
INTC	Intel Corporation	18.12 ▼	-0.380	-2.05%
JNJ	Johnson & Johnson	53.29 ▼	-0.290	-0.54%
KO	Coca-Cola Company	56.70 ▼	-0.580	-1.01%
MSFT	Microsoft Corporation	53.96 ▲	+1.040	+1.97%
NKE	NIKE, Inc.	57.34 ▲	+0.580	+1.02%

- Perform the following actions in SQL:
  a) Add a new record (‘Google’, ‘Google Inc.’, 450.00, 10%, 200.00).
  b) Delete the records with *Last* < 15.

Integrity Constraints

- **Integrity Constraints (ICs):** conditions that must be true for *any* instance of the database; e.g., data type.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.

- ICs are based upon the semantics of the application that is being described in the database relations.

- A **legal** instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
  - Avoids data entry errors, too!

Some Representative Examples of Integrity Constraints

- Primary key constraint
- Unique constraint
- Not NULL constraint
- Referential integrity constraint

Primary Key Constraint

- A set of attribute(s) is a **key** for a relation if :
  1. No two tuples can have same value(s) in all the attribute(s) of the key.
  2. This is not true for any subset of the key (*minimal* requirement).

- Example:

  Students(*sid*: char(20), *name*: char(20), *login*: char(10), *age*: integer, *gpa*: real).

- *sid* is a key. (What about *name*?)

- {*sid*, *gpa*} is not a key (Condition 2 is violated).

Unique Constraint

- If there is more than one key for a relation (i.e., **candidate keys**), one is chosen as the **primary key**.

- **Candidate key** is unique.

- Example:

  Students(*sid*: char(20), *name*: char(20), *login*: char(10), *age*: integer, *gpa*: real).

- *login* is a candidate key, which must be unique.

Primary and Unique Constraints in SQL

- Primary key is specified using PRIMARY KEY.
- Candidate key(s) is (are) specified using UNIQUE.

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

Question 3

- Suppose that the table "Students" is created by the following SQL

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

- What's the result of executing the following statements?
  - INSERT INTO Students VALUES ('00001', 'Bob', 'bob@comp', 18, 3.2)
  - INSERT INTO Students VALUES ('00001', 'Tom', 'tom@comp', 18, 3.2)
  - INSERT INTO Students VALUES ('00002', 'Bob', 'bob@comp', 18, 3.2)

Not NULL constraint

- If a column of a table is NOT NULL, you must give a value when you insert a tuple to the table.

- SQL Example:

```
CREATE TABLE Students
          (sid  CHAR(20),
           name  CHAR(20) NOT NULL,
           login   CHAR(10),
           age   INTEGER,
           gpa  REAL)
```

- Remark: All attributes of the primary key MUST NOT be NULL.

Referential Integrity Constraint

- Example: The Students and Enrolled Relations

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

Enrolled

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

- Assume that we want to insert a tuple (‘50000’, ‘CS160’, ‘A’) into Enrolled.
- Before we do so, we may want to make sure there exists a student in Students with *sid* = ‘50000’.
- This is called as referential integrity. (How to achieve this?)

Foreign Key

- **Foreign key**: Set of attribute(s) in one relation that is used to \`refer' to a tuple in another relation (can be itself).
  - Must correspond to primary key of the referenced relation. Like a \`logical pointer'.
  - E.g., *sid* of Enrolled is a foreign key referring to Students.

Students(sid: char(20), name: char(20), login: char(10), age: integer, gpa: real)
Enrolled(sid: char(20), cid: char(20), grade: char(2))

- Can achieve referential integrity: make sure every sid in Enrolled exists in Students.

Foreign Key in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.
- But some tuples in *Students* may not be referenced.

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

When a tuple is inserted in table *Enrolled*, what constraints are being checked?

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 Key in SQL

- Attribute names can be different.

Students(sid: char(20), name: char(20), login: char(10), age: integer, gpa: real)
Enrolled2(stuid: char(20), cid: char(20), grade: char(2))

```
CREATE TABLE Enrolled2
    (stuid CHAR(20),  cid CHAR(20),  grade CHAR(2),
     PRIMARY KEY  (stuid, cid),
     FOREIGN KEY (stuid) REFERENCES Students (sid) )
```

Foreign Key in SQL

- Foreign keys can refer to the same relation.

- Example:

Students2(<u>sid</u>:char(20), name:char(20), login:char(10), age:integer, gpa:real, partner:char(20))

```
CREATE TABLE Students2
    (sid CHAR(20), name CHAR(20), login CHAR(10),
     age INTEGER, gpa REAL, partner CHAR(20),
     PRIMARY KEY (sid),
     FOREIGN KEY (partner) REFERENCES Students2 (sid) )
```

- If a student has no partner, this attribute can be NULL (a special keyword in SQL denoting *`unknown'* or *`inapplicable'*).
- NULL is allowed in non-primary keys, including foreign keys.

Enforcing Referential Integrity via Foreign Key

- Example:
  - Students and Enrolled Relations
  - 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?
  - Disallow deletion/update of a Students tuple that is referred to. (NO ACTION)
  - Also delete/update all Enrolled tuples that refer to it. (CASCADE)
- Similar if primary key of Students tuple is updated.

Enforcing Referential Integrity in SQL

- NO ACTION (*delete/update is rejected*).
- CASCADE (also delete/update all tuples that refer to deleted tuple).
- The DBMS adopts the default referential integrity if there is no specification.

```
CREATE TABLE Enrolled
  (sid CHAR(20) DEFAULT '53688',
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid, cid),
   FOREIGN KEY (sid)
     REFERENCES Students (sid)
           ON DELETE CASCADE
           ON UPDATE NO ACTION)
```

```
CREATE TABLE Enrolled
  (sid CHAR(20) DEFAULT '53688',
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid, cid),
   FOREIGN KEY (sid)
     REFERENCES Students (sid)
           ON DELETE CASCADE
           ON UPDATE CASCADE)
```

Question 4

Consider the instances of Students and Enrolled relations on Slide 23. Suppose that the relation “Enrolled” is created based on the following SQL queries.

```
CREATE TABLE Enrolled
  (sid CHAR(20) DEFAULT '53688',
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid, cid),
   FOREIGN KEY (sid)
     REFERENCES Students (sid)
          ON DELETE NO ACTION
          ON UPDATE NO ACTION)
```

What happens if we perform the following operations for the instance “Students”:

a) Delete the tuple with sid = 53666
b) Insert a tuple with sid = 53600, name = ‘Edison’, login = ‘edison@cs’, age = 31, and gpa = 4.0.
c) Update the tuple with sid = 53650 to 53700

Question 5

- Repeat Question 3 if the relation “Enrolled” is created based on the following SQL queries.

```
CREATE TABLE Enrolled
  (sid CHAR(20) DEFAULT '53688',
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid, cid),
   FOREIGN KEY (sid)
     REFERENCES Students (sid)
           ON DELETE CASCADE
           ON UPDATE CASCADE)
```

Summary

- Mostly commonly used table constraints
  - Domain (data type) constraint
  - Primary Key constraint
  - Unique constraint
  - Not NULL constraint
  - Referential Integrity constraint

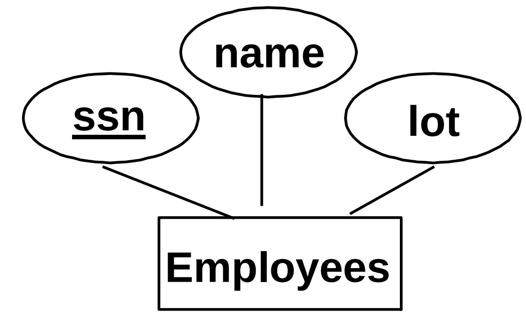
- More general constraints can also be supported

ER Model to Relation?

- How can we convert the high-level ER model to relational tables?

- Next we will explain the conversion for each component in an ER diagram.

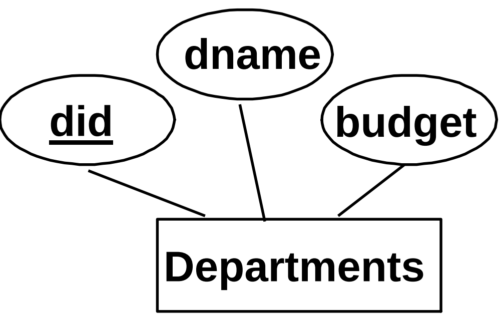
Entity Set to Relation




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

Employees

ssn	name	lot
R123	Alice	32
P625	Bob	87
A252	Candy	16

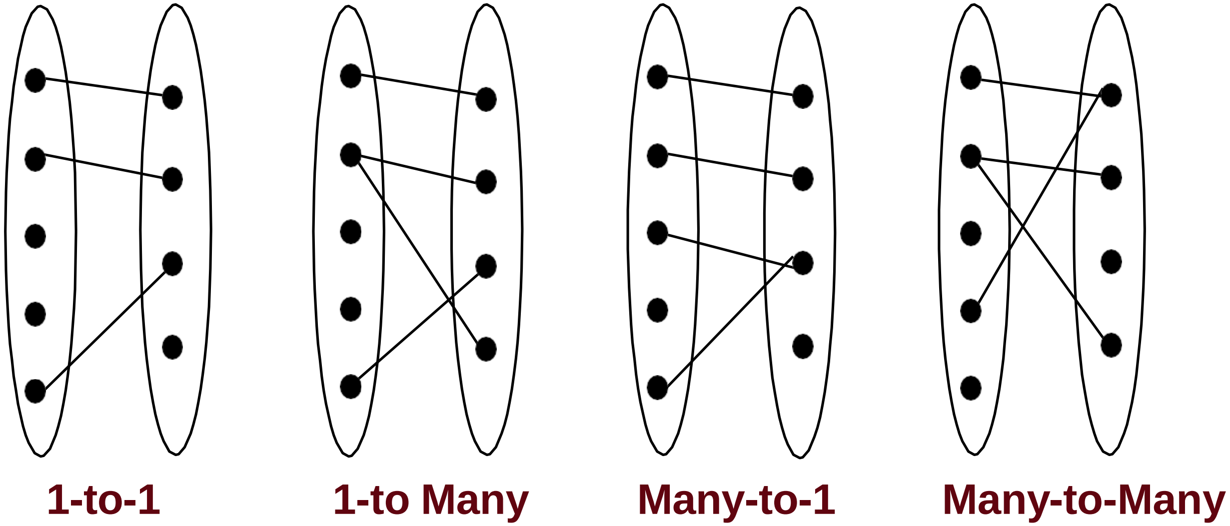



```
CREATE TABLE Departments
    (did INTEGER,
    dname CHAR(20),
    budget  REAL,
    PRIMARY KEY  (did))
```

Departments

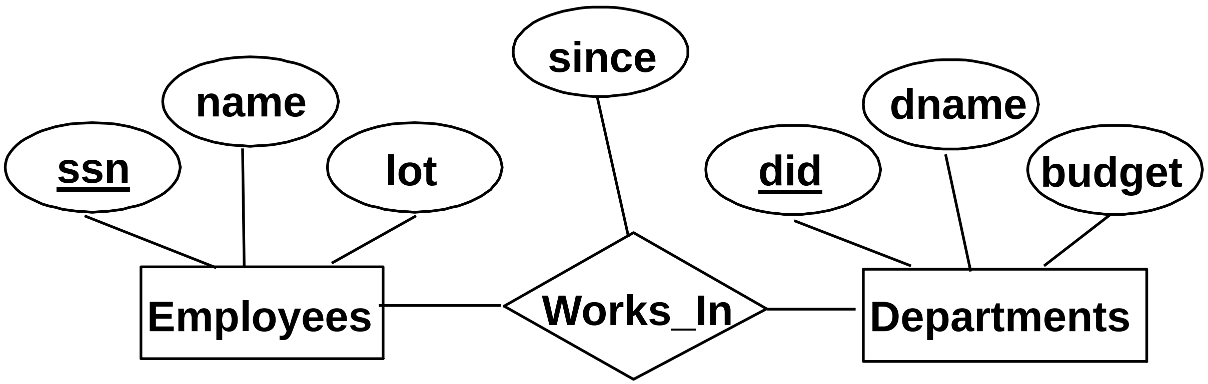
did	dname	budget
1	COMP	15000
2	MATH	15000
3	PHYS	12000

Review: Cardinality Constraints



Translation to relational model?

m-m Relationship Set to Relation



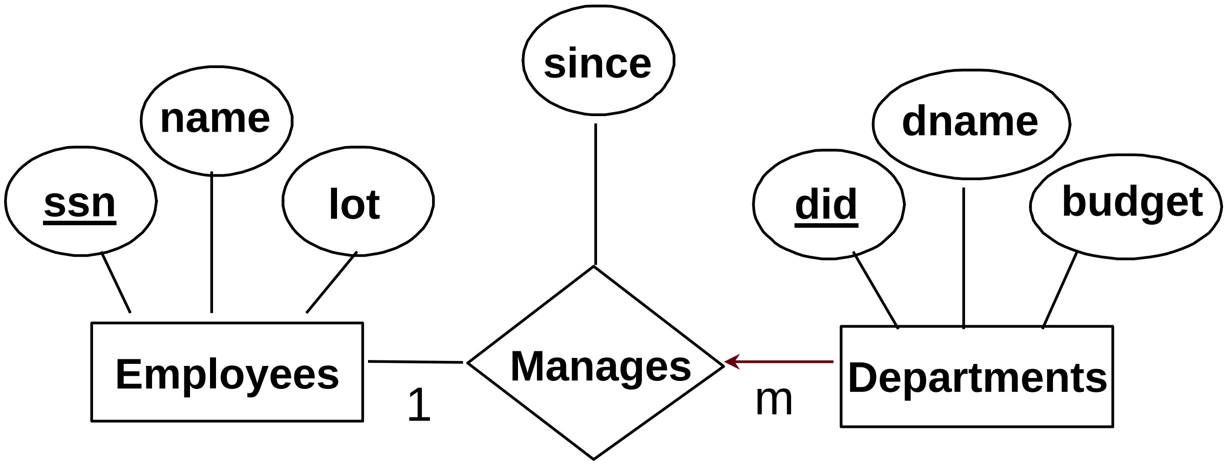
Works_In

ssn	did	since
R123	1	2011
R123	2	2016
P625	1	2018

- To convert a relationship set to a relation, it must include:
  - keys for each participating entity set (as foreign keys)
  - all descriptive attributes of the relationship
- What is the primary key of this relation?
- Which relations do we have for this ER diagram?

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

1-m Relationship Set to Relation



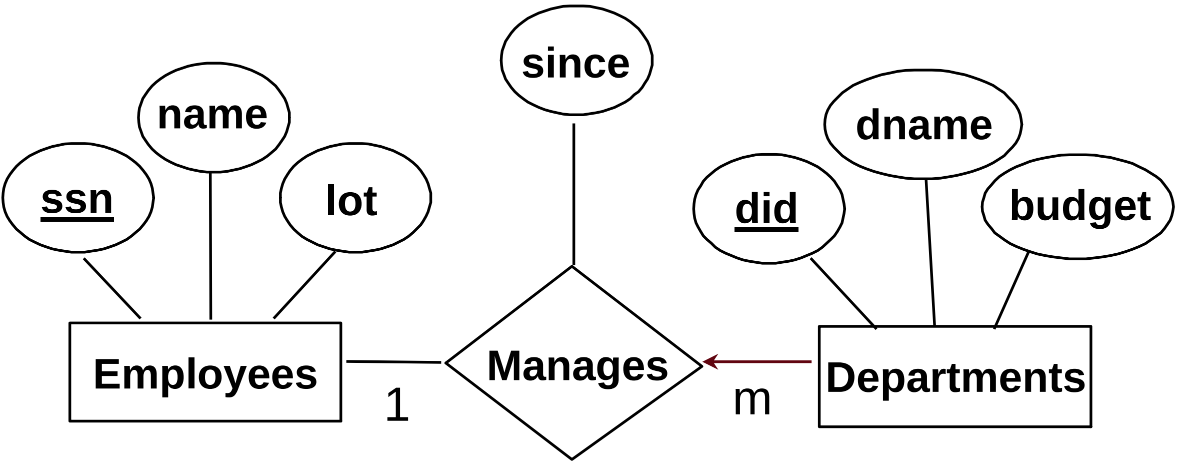
Manages

ssn	did	since
R123	1	2011
R123	2	2016
~~P625~~	~~1~~	~~2018~~

- Map the relationship "Manages" to a relation.
- What is the primary key of this relation?
- Which relations do we have for this ER diagram?

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

1-m Relationship Set to Relation



Dept_Mgr

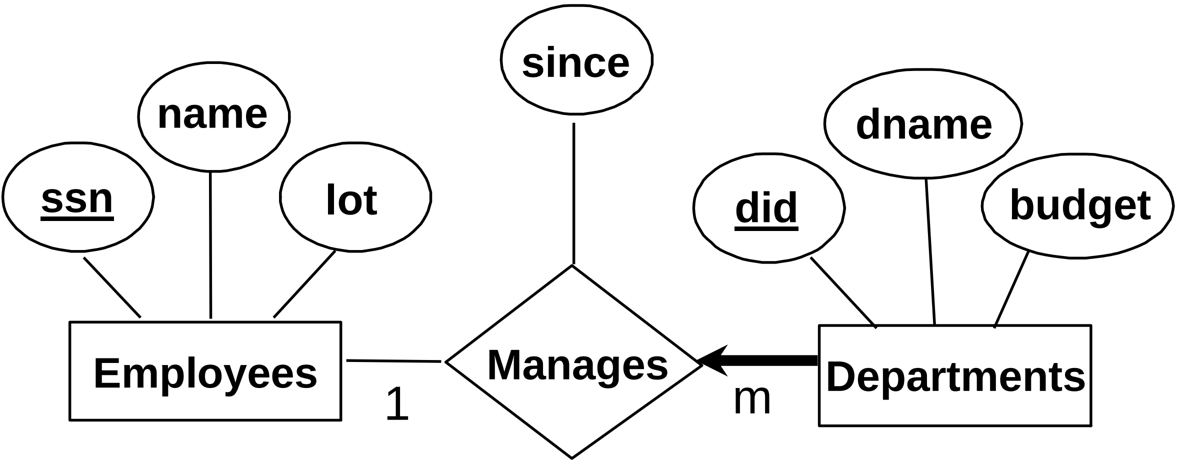
did	dname	budget	ssn	since
1	COMP	15000	R123	2011
2	MATH	15000	R123	2016

- Since each department has a unique manager, we can combine Manages and Departments together into one relation, called Dept_Mgr.

- There are two relations, which are Employees and Dept_Mgr.

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

1-m Relationship Set (with Participation Constraint) to Relation



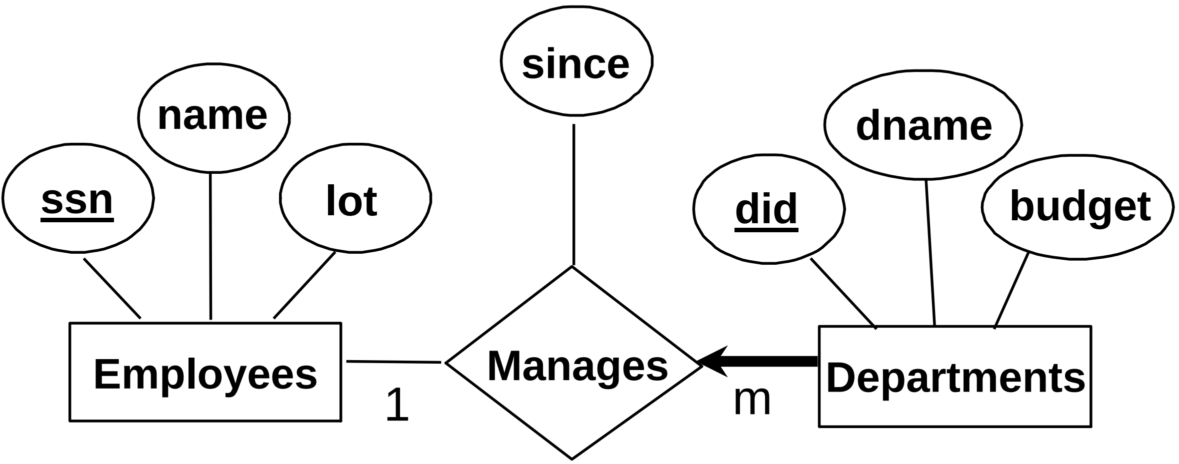
Dept_Mgr

did	dname	budget	ssn	since
1	COMP	15000	R123	2011
2	MATH	15000	R123	2016

- Setting ssn to be NOT NULL makes sure every did (or department) in the Dept_Mgr must have the corresponding ssn (or employee) to manage it. Therefore, every department must participate in this relationship.

```
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11) NOT NULL,
  since INTEGER,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn)
          REFERENCES Employees)
```

1-m Relationship Set (with Participation Constraint) to Relation



Manages

ssn	did	since
R123	1	2011
R123	2	2016

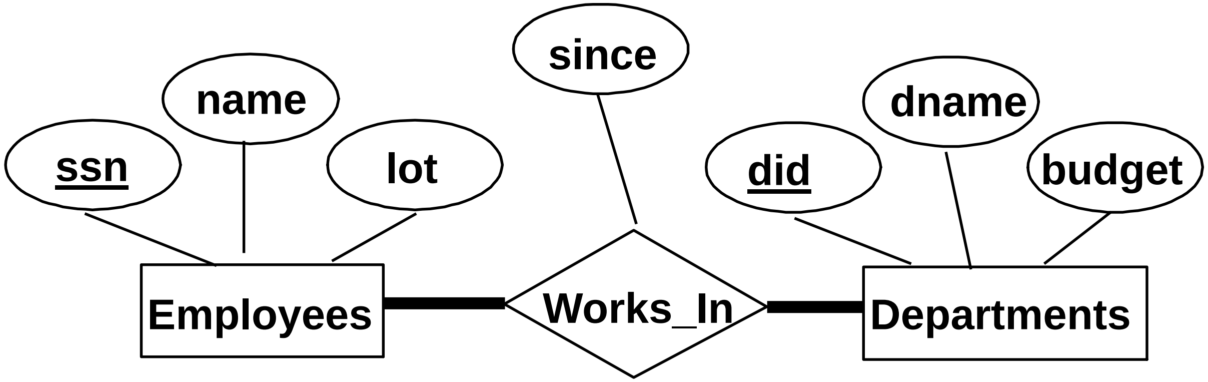
- Cannot capture participation constraints by adding the NOT NULL constraint to ssn.
- Because we can add many new tuples into the Departments relation, i.e., it cannot guarantee every did of Department appears in Manages.

```
CREATE TABLE  Manages(
  ssn  CHAR(11) NOT NULL,
  did  INTEGER,
  since  INTEGER,
  PRIMARY KEY  (did),
  FOREIGN KEY (ssn)
          REFERENCES Employees,
  FOREIGN KEY (did)
          REFERENCES Departments)
```

(doesn't work)

Hard to Capture All Participation Constraints!

- Example:

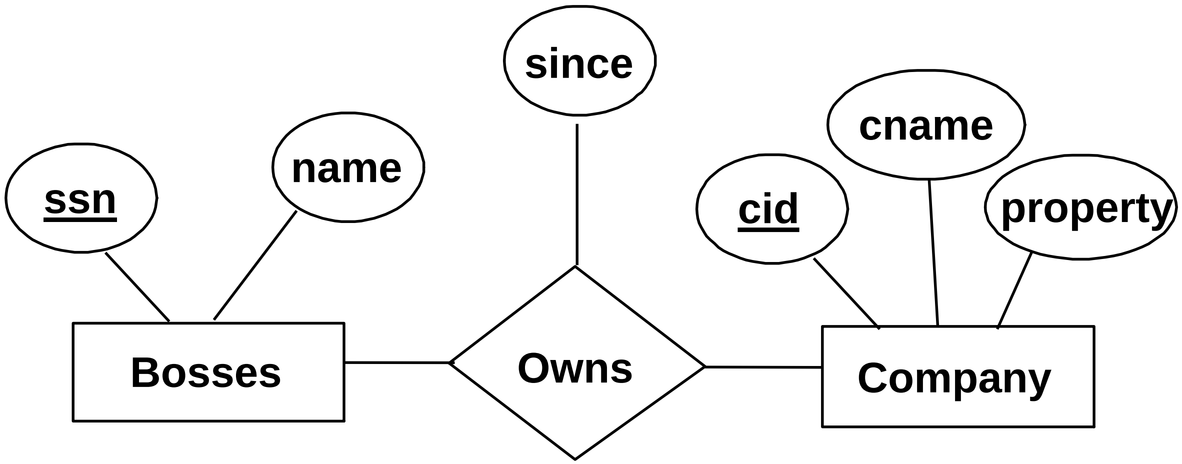


- It is insufficient to use only PRIMARY KEY, UNIQUENESS, NOT NULL, and FOREIGN KEY to implement all participation constraints.

- More powerful features of SQL are needed to implement the relationship *Works_In*.

Question 6

Consider the following ER diagram.



Write the SQL query that can capture the cardinality constraint of the above ER diagram.

Solution to Question 1

- Create the following relation using SQL.
  - CREATE TABLE **Points Per Game Leader - Qualified**

    (RK INTEGER,
    PLAYER VARCHAR(20),
    TEAM CHAR(10),
    GP INTEGER,
    MPG REAL)

Points Per Game Leaders - Qualified

RK	PLAYER	TEAM	GP	MPG
1	Stephen Curry, PG	GS	79	34.2
2	James Harden, SG	HOU	82	38.1
3	Kevin Durant, SF	OKC	72	35.8
4	DeMarcus Cousins, C	SAC	65	34.6
5	LeBron James, SF	CLE	76	35.6
6	Damian Lillard, PG	POR	75	35.7
7	Anthony Davis, PF	NO	61	35.5
8	Russell Westbrook, PG	OKC	80	34.4

Solution to Question 2

- CREATE TABLE **Stocks**

  (Symbol CHAR(20),
  Name CHAR(100),
  Last REAL,
  Change REAL,
  %Chg REAL)

- INSERT INTO Stocks(Symbol, Name, Last, Change, %Chg) VALUES(‘Google’, ‘Google Inc.’, 450.00, 10%, 200.00)

- DELETE FROM Stocks WHERE Last<15

Solution to Question 3

- The first record is successfully inserted into the table

sid	name	login	age	gpa
00001	Bob	bob@comp	18	3.2

- The DBMS rejects the 2nd statement since its sid=00001, which is the primary key and a record with sid=00001 already exists in the table.

- The DBMS rejects the 3rd statement since its login='bob@comp', which should be unique and a record with login='bob@comp' already exists in the table.

Solution to Question 4

- The DBMS rejects this operation as there are three records with the foreign key sid=53666 in the Enrolled relation. The ‘NO ACTION’ referential integrity has been imposed to this foreign key.

- The DBMS executes this operation successfully.

- The DBMS rejects this operation as there is one record with foreign key sid=53650 in the Enrolled relation. The ‘NO ACTION’ referential integrity has been imposed to this foreign key.

Solution to Question 5

- The DBMS executes this operation successfully.

Students

sid	name	login	age	gpa
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- The DBMS executes this operation successfully.

sid	name	login	age	gpa
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8
53600	Edison	edison@cs	31	4.0

- The DBMS executes this operation successfully.

sid	name	login	age	gpa
53688	Smith	smith@eecs	18	3.2
53700	Smith	smith@math	19	3.8
53600	Edison	edison@cs	31	4.0

Solution to Question 6

```
CREATE TABLE Bosses
        (ssn      INTEGER,
         name     CHAR(20),
         PRIMARY KEY(ssn)
        );

CREATE TABLE Company
        (cid       INTEGER,
         cname     CHAR(20),
         property REAL,
         PRIMARY KEY(cid)
        );
```

```
CREATE TABLE Owns
      (ssn          INTEGER,
       cid          INTEGER,
       since        INTEGER,
       PRIMARY KEY (ssn, cid),
       FOREIGN KEY (ssn) REFERENCES
                          Bosses(ssn),
       FOREIGN KEY (cid) REFERENCES
                          Company(cid)
      );
```

Conclusion

- Relational database
  - Schema
  - Instance
- Integrity constraints
  - Primary key constraint
  - Unique constraint
  - Not NULL constraint
  - Referential integrity constraint (foreign key)
- Mapping ER-diagram to Relational Models
  - Handle the cardinality constraints in the ER-diagram.
  - Handle the participation constraints in the ER-diagram.
- Basic SQL