

Databases and Information Systems

Relational Model
Relational Calculus

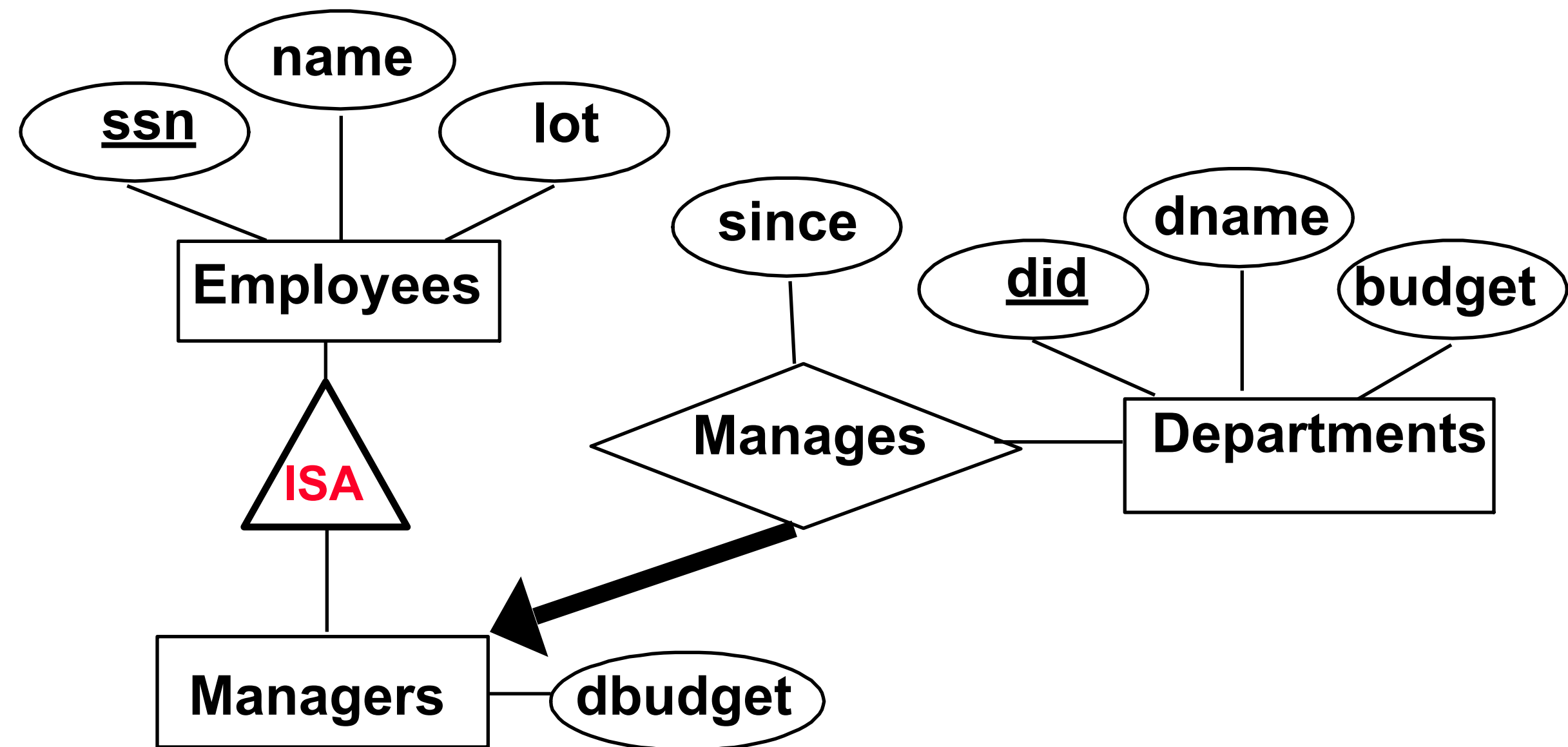
Dmitriy Traytel

slides also by Marcos Vaz Salles



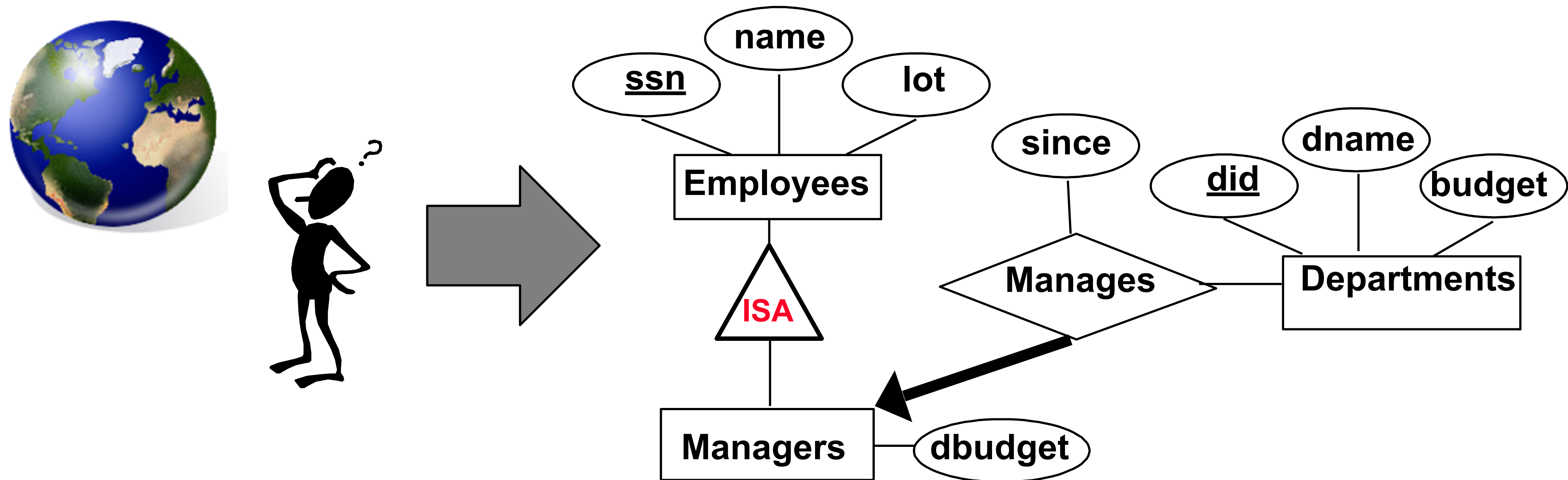
UNIVERSITY OF
COPENHAGEN

Do-It-Yourself Recap: Conceptual Models



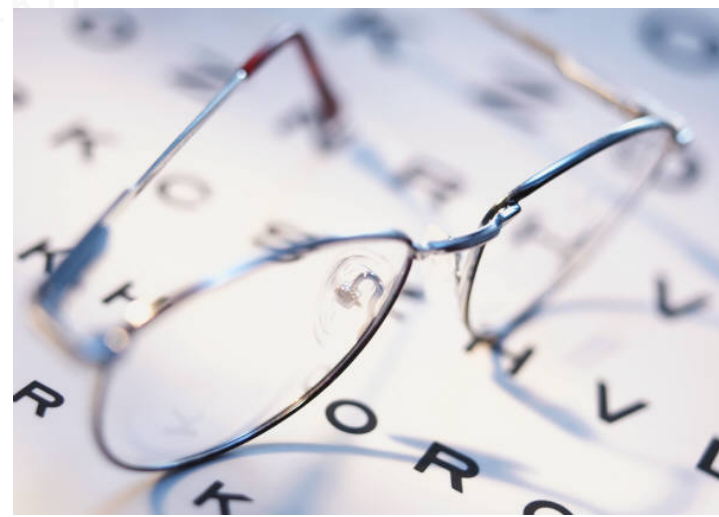
- Can you describe what entity sets and relationship sets are encoded by this model?

Do-It-Yourself Recap: Conceptual Models



- Can you describe what entity sets and relationship sets are encoded by this model?

What should we learn today?



- Define and explain the relational data model
- Define and explain the main classes of integrity constraints in the relational model, in particular key and foreign key constraints
- Model relational schemas and express them in SQL
- Explain what a query language is
- Identify the main relational database query languages
- Formulate queries in the relational calculus

Relational Database: Definitions

- **Relational database:** a set of *relations*
- **Relation:** made up of two parts:
 - **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)
 - **Instance:** a *table*, with rows and columns.
#Rows = *cardinality*, #fields = *degree / arity*.
- Can think of a relation as a **set** of rows or **tuples** (i.e., all rows are distinct).

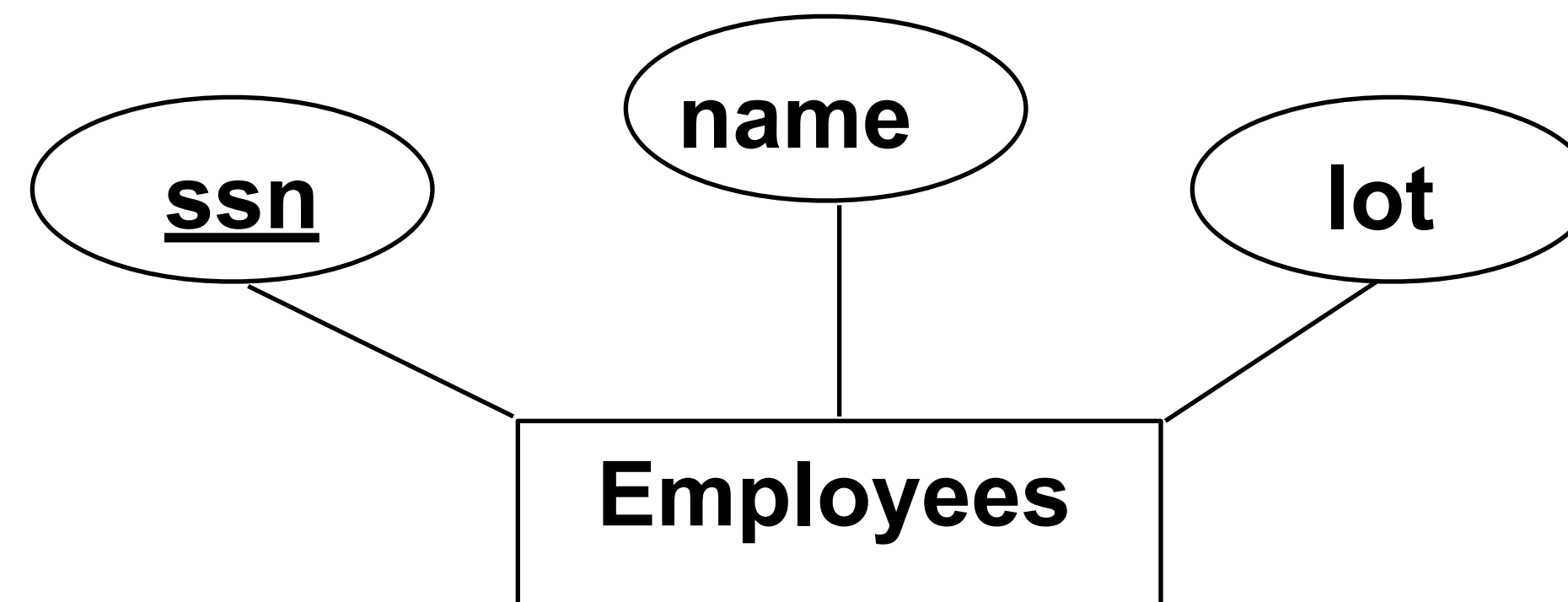
Example Instance of Students Relation

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

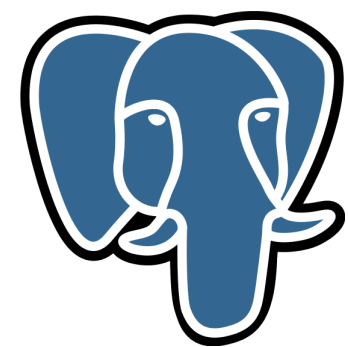
- Cardinality = 3, degree = 5, all rows distinct
- Do all columns in a relation instance have to be distinct?

Logical Design: E/R to Relational

- Entity sets to tables.



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



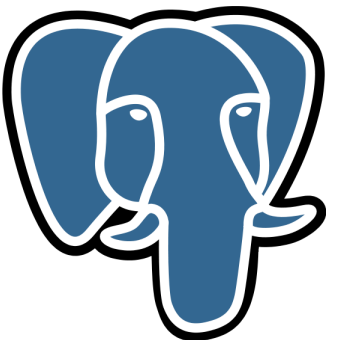
```
Employees
(ssn : string,
 name : string,
 lot : int)
https://traytel.bitbucket.io/rc-eval/
```


Example Instance

Employees

<u>ssn</u>	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20

```
INSERT INTO Employees(ssn,name,lot)
VALUES (0983763423, 'John', 10), (9384392483, 'Jane', 10), (3743923483, 'Jill', 20);
```



```
Employees
("0983763423", "John", 10) ("9384392483", "Jane", 10) ("3743923483", "Jill", 20)
```

<https://traytel.bitbucket.io/rc-eval/>

Integrity Constraints (ICs)

- **IC:** condition that must be true for *any* instance of the database
 - Domain constraints
 - Key constraints
 - Foreign key constraints (later)
- A *legal* instance of a relation is one that satisfies all specified ICs.
 - DBMS should not allow illegal instances
 - Avoids data entry errors too!

Primary Key Constraints

- A set of fields is a superkey for a relation if:
 - No two distinct tuples can have same values in all fields belonging to the superkey
- A set of fields is a key if:
 - The set of fields is a superkey
 - No proper subset of the set of fields is a superkey
- If there is >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.
- E.g., {ssn} is a key for Employees. (What about {name}?) The set {ssn, name} is a superkey.

What does this mean?

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

Candidate Keys

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

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

- Each student is enrolled in at most one course
- No two students in a course get the same grade

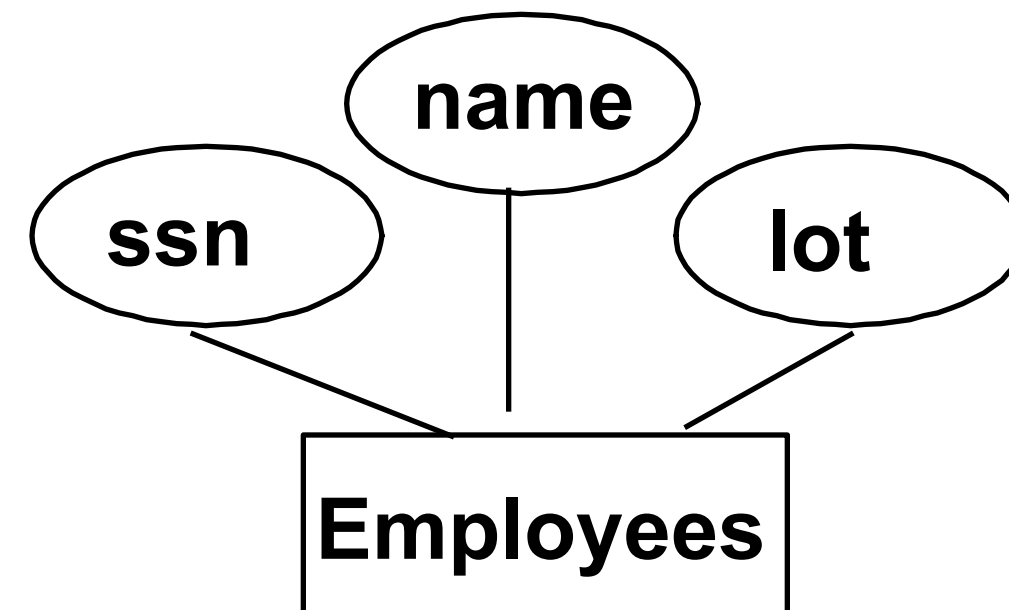
Where do ICs Come From?

- 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.

E/R to Relational (contd.)

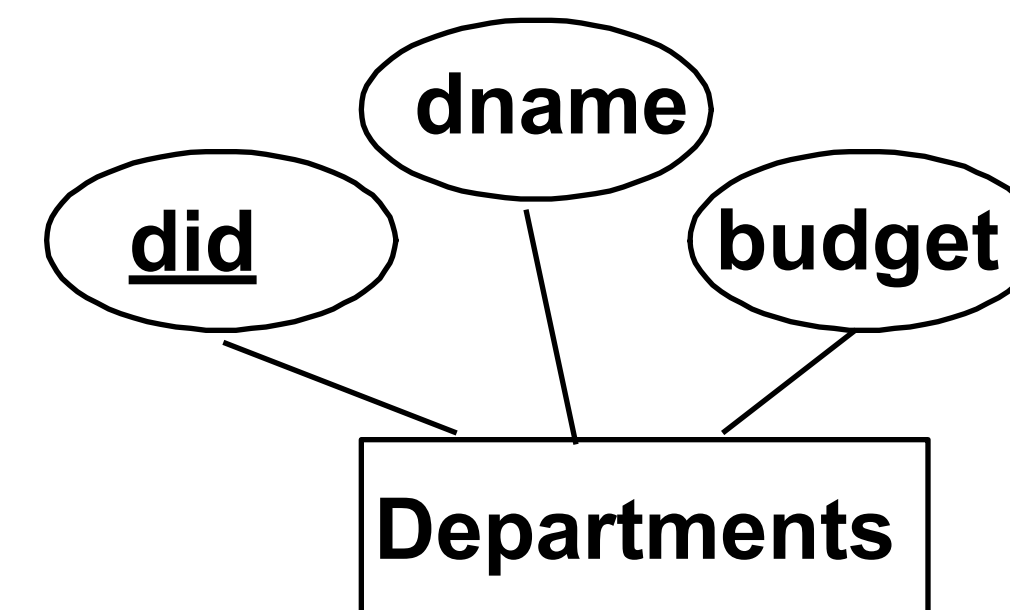
CREATE TABLE Employees

(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))



CREATE TABLE Departments

(did INTEGER,
dname CHAR(20),
budget FLOAT,
PRIMARY KEY (did))



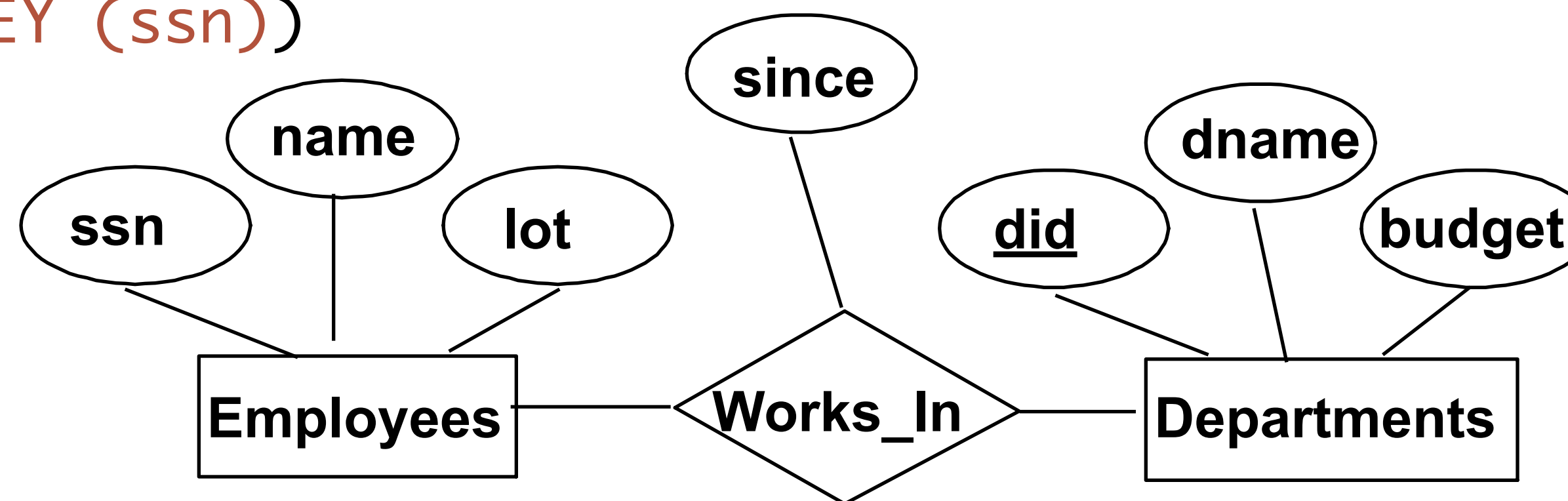
E/R to Relational (contd.)

CREATE TABLE Employees

(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))

CREATE TABLE Departments

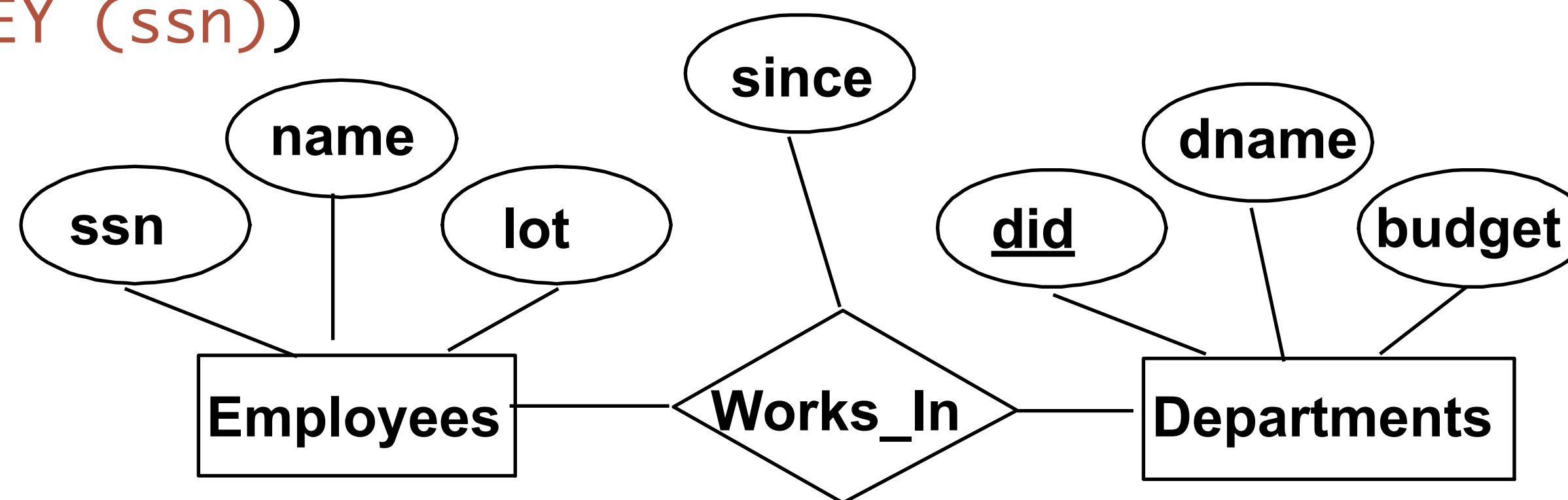
(did INTEGER,
dname CHAR(20),
budget FLOAT,
PRIMARY KEY (did))



E/R to Relational (contd.)

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

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



```
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)
```

Example Instance

Employees

<u>ssn</u>	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20

Departments

<u>did</u>	dname	budget
101	Sales	10K
105	Purchasing	20K
108	Databases	1000K

Works_In

<u>ssn</u>	<u>did</u>	since
0983763423	101	1 Jan 2003
0983763423	108	2 Jan 2003
9384392483	108	1 Jun 2002

Foreign Keys, Referential Integrity

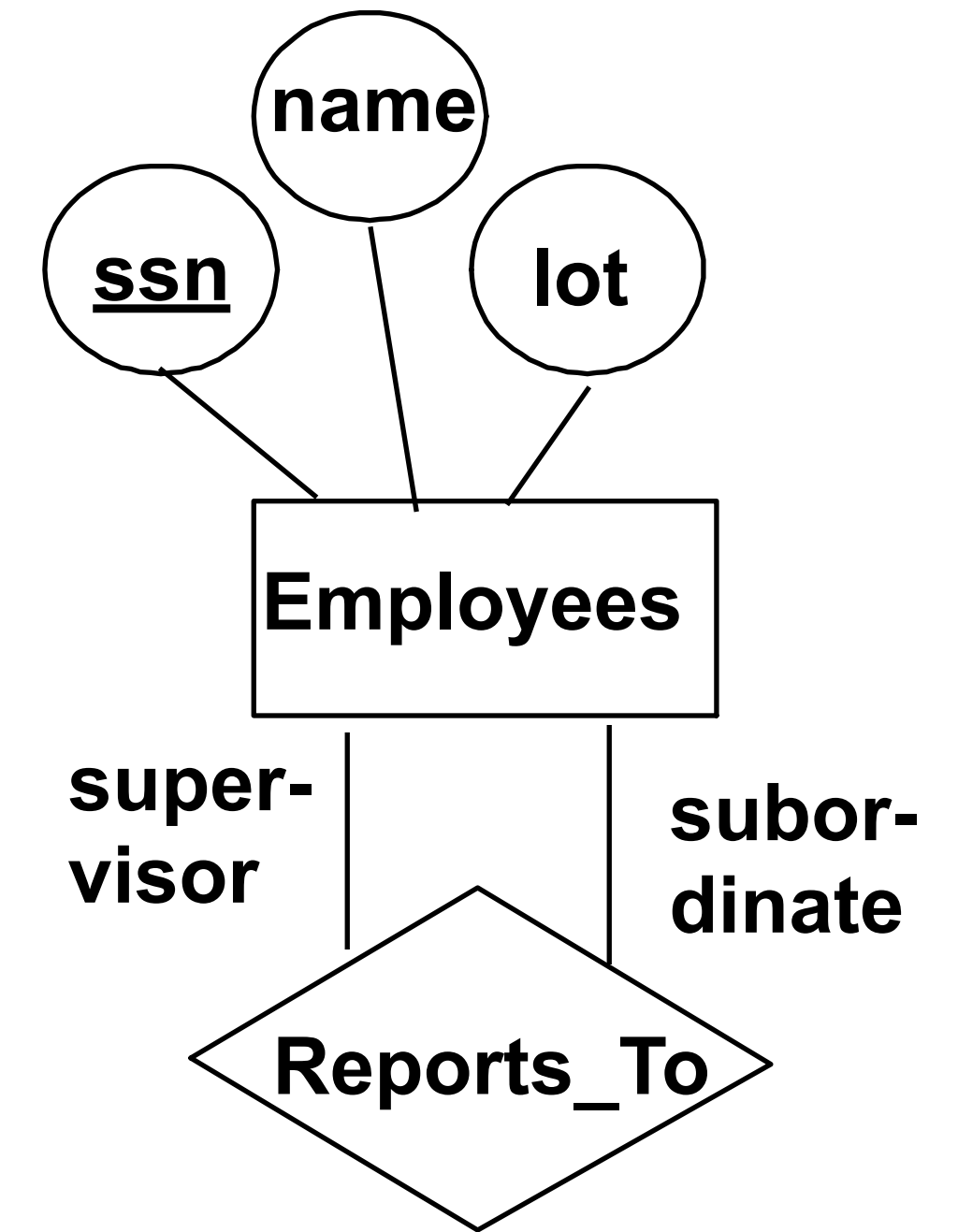
- Foreign key: Set of fields in one relation that is used to 'refer' to a tuple in another relation
 - Must correspond to primary key of the second relation
 - Like a 'logical pointer'.
- If all foreign key constraints enforced, referential integrity is achieved, i.e., no dangling references.
 - Not like HTML links!

Enforcing Referential Integrity

- What if a new “Works_In” tuple is added that references a non-existent employee?
 - Reject it!
- What if an Employee tuple is deleted?
 - Also delete all Works_In tuples that refer to it.
 - Disallow deletion of Employee tuple that is referred to.
 - Set *ssn* to some default value
 - Set *ssn* in Works_In to *null*, denoting ‘unknown’
- Similar if primary key of Employee tuple is updated

E/R to Relational (contd.)

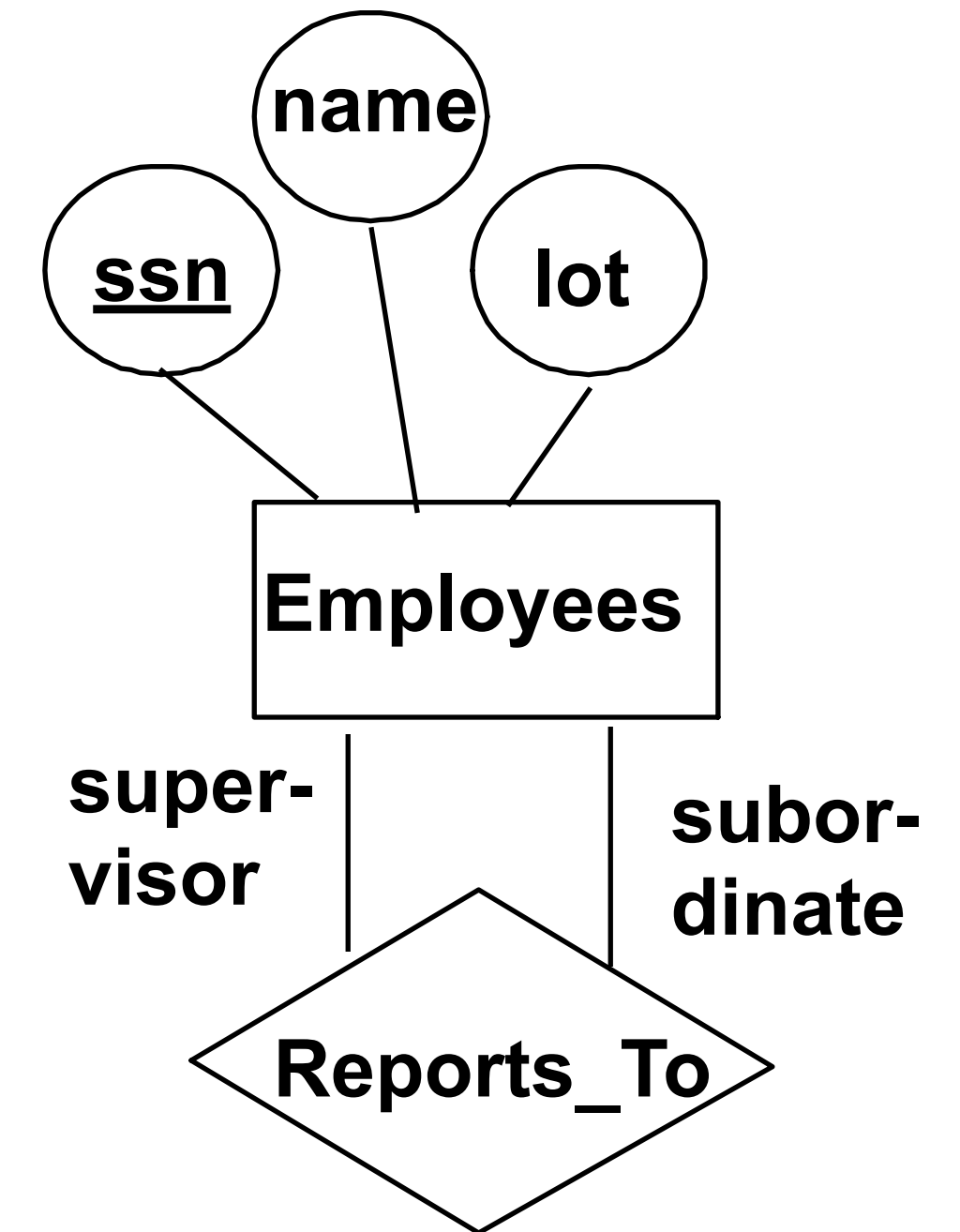
```
CREATE TABLE Employees
  (ssn CHAR(11),
   name CHAR(20),
   lot  INTEGER,
   PRIMARY KEY (ssn))
```



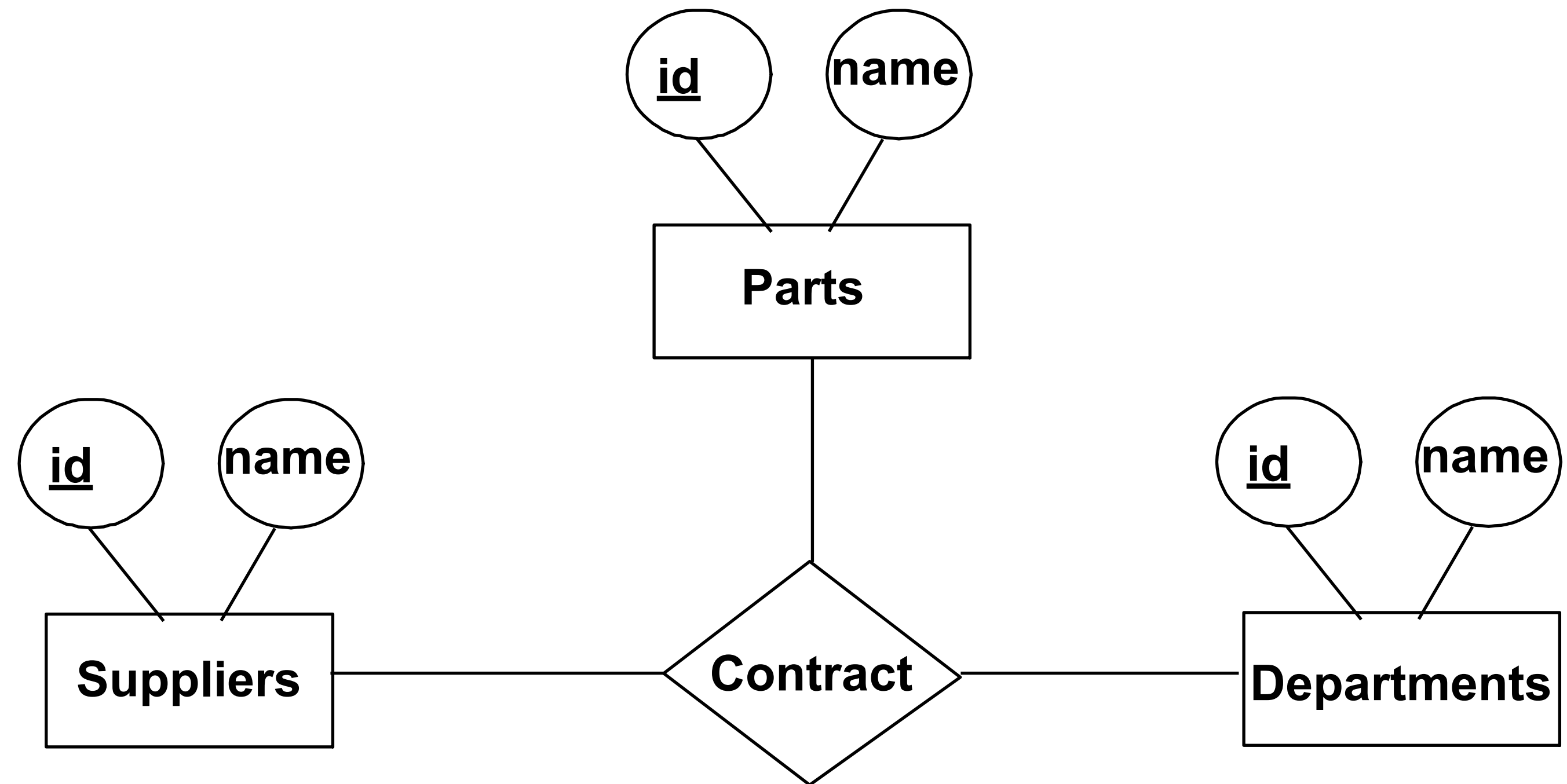
E/R to Relational (contd.)

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

```
CREATE TABLE Reports_To
    (supervisor_ssn CHAR(11),
     subordinate_ssn CHAR(11),
     PRIMARY KEY (supervisor_ssn, subordinate_ssn),
     FOREIGN KEY (supervisor_ssn) REFERENCES Employees,
     FOREIGN KEY (subordinate_ssn) REFERENCES Employees)
```

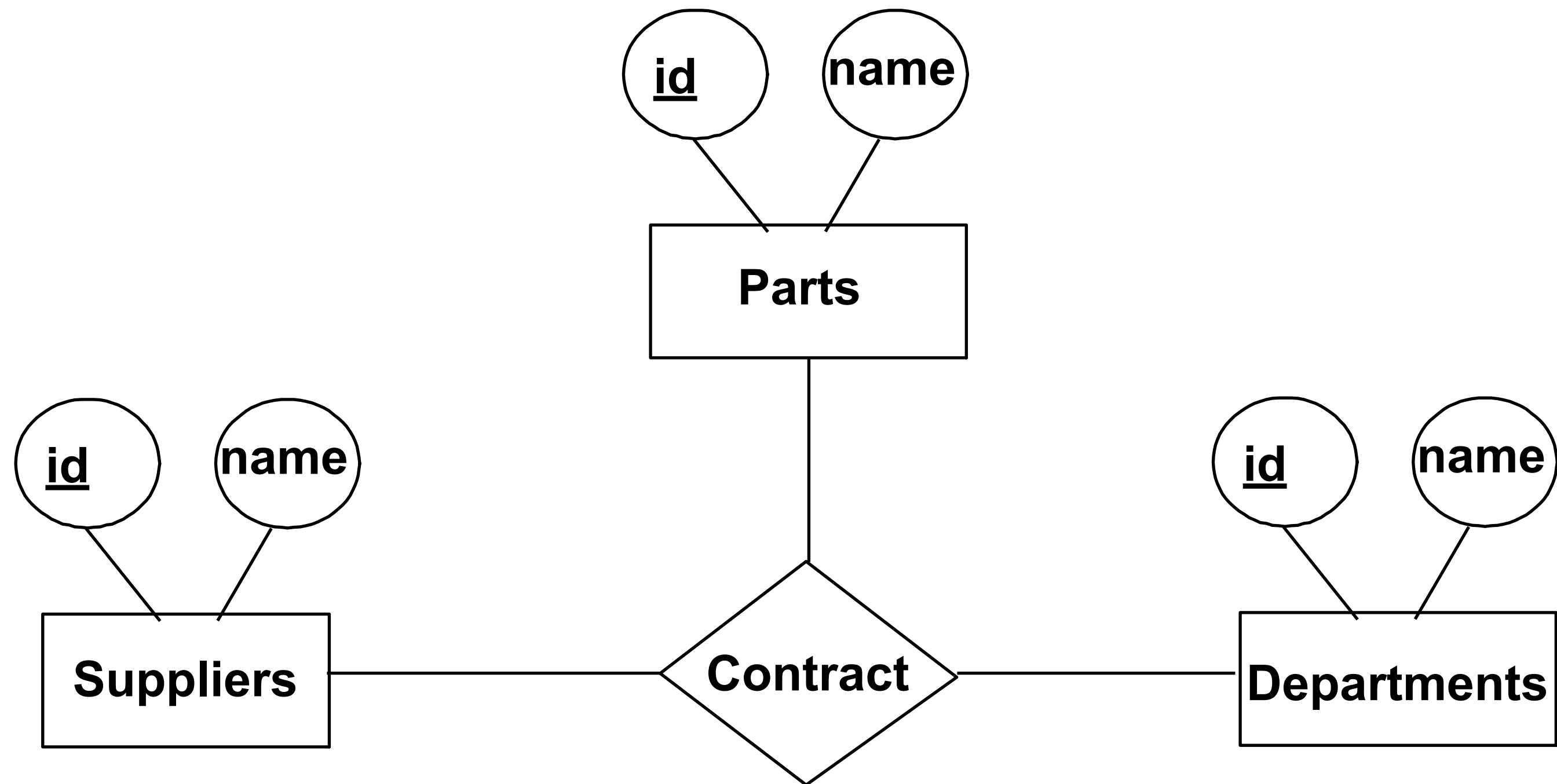


E/R to Relational (contd.)



E/R to Relational (contd.)

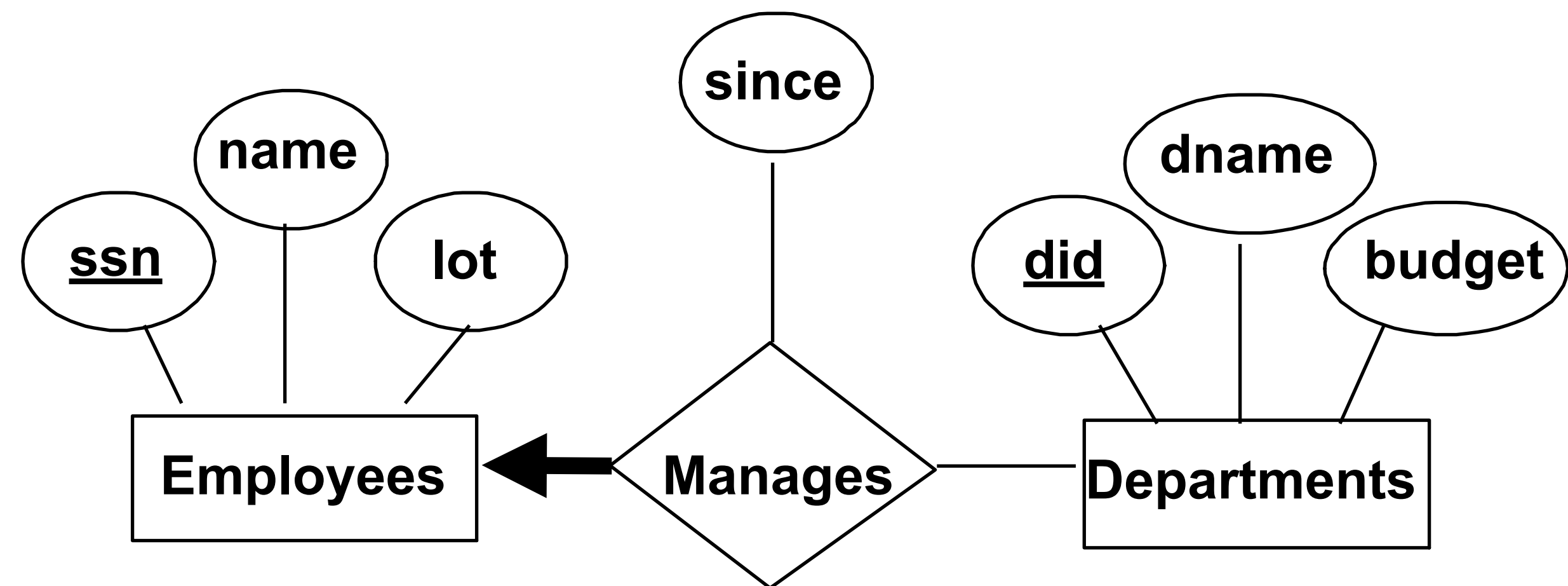
```
CREATE TABLE Contracts
(supplier_id INTEGER,
 part_id INTEGER,
 department_id INTEGER,
 PRIMARY KEY (supplier_id, part_id, department_id),
 FOREIGN KEY (supplier_id) REFERENCES Suppliers,
 FOREIGN KEY (part_id) REFERENCES Parts,
 FOREIGN KEY (department_id) REFERENCES Departments)
```



E/R to Relational (contd.)

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

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

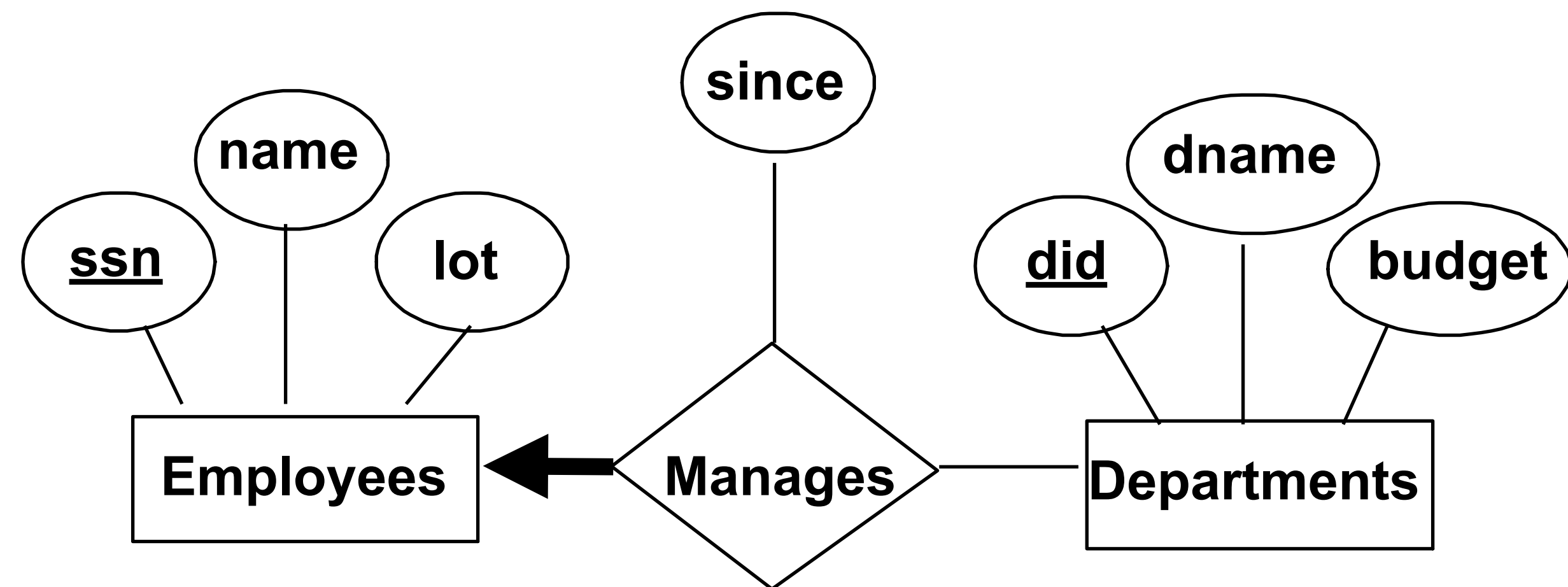


E/R to Relational (contd.)

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

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

```
CREATE TABLE Departments
(did INTEGER,
 dname CHAR(20),
 budget FLOAT,
 mgr_ssn CHAR(11),
 since DATE,
PRIMARY KEY (did),
FOREIGN KEY (mgr_ssn) REFERENCES Employees)
```

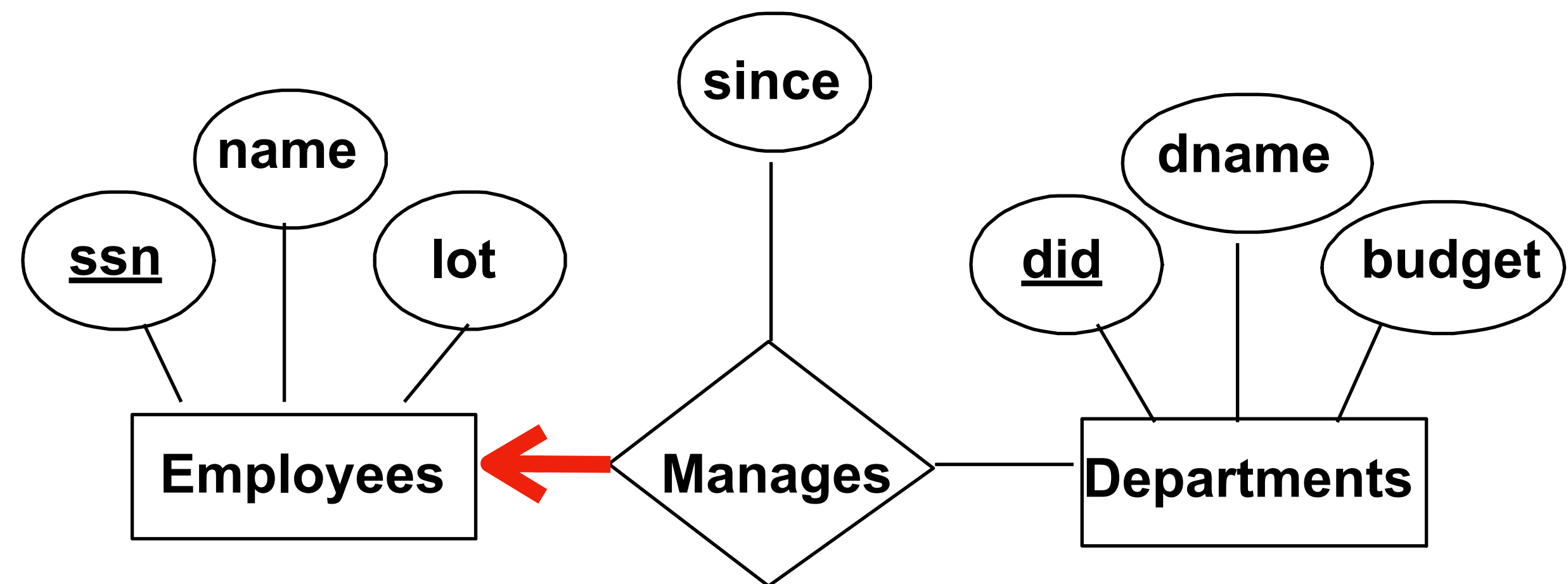


E/R to Relational (contd.)

- Each department has exactly one manager, according to the **referential integrity constraint** on Manages.

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

```
CREATE TABLE Departments
(did INTEGER,
 dname CHAR(20),
 budget FLOAT,
 mgr_ssn CHAR(11),
 since DATE,
PRIMARY KEY (did),
FOREIGN KEY (mgr_ssn) REFERENCES Employees)
```

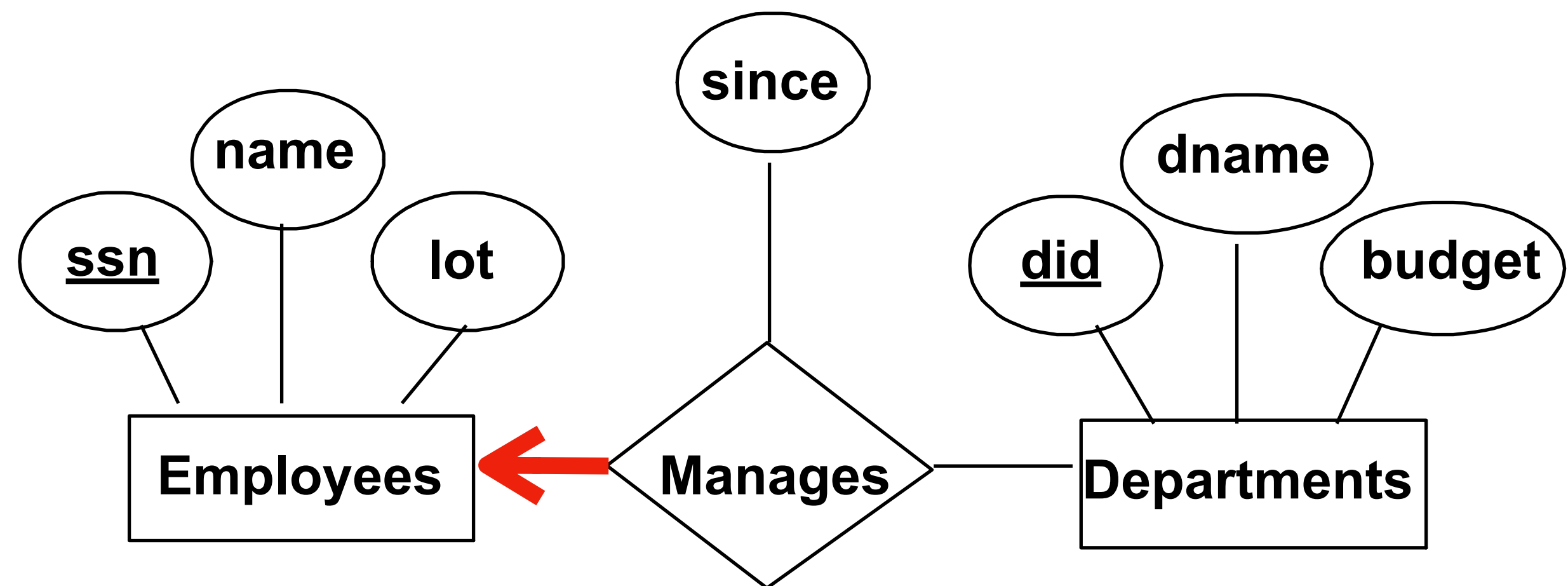


E/R to Relational (contd.)

- Each department has exactly one manager, according to the **referential integrity constraint** on Manages.

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

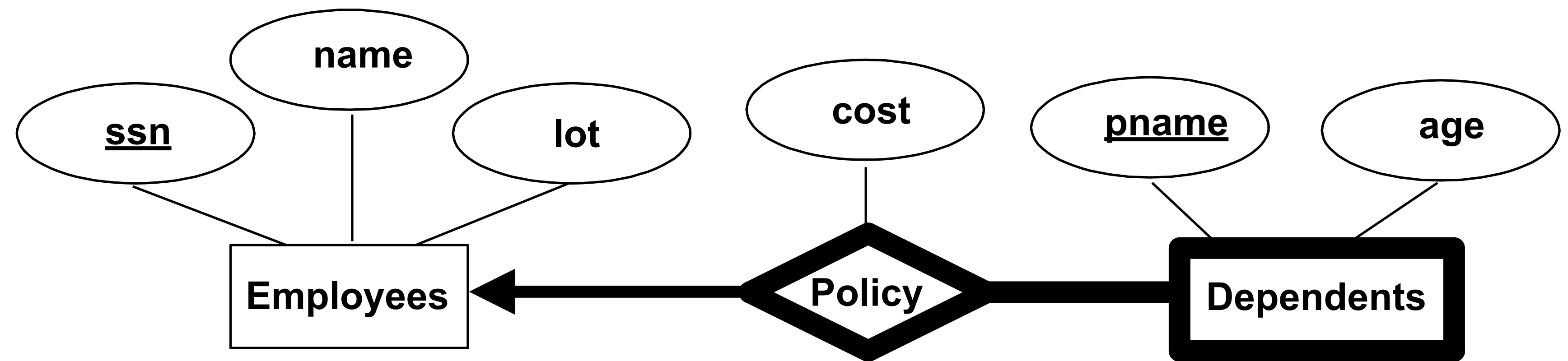
```
CREATE TABLE Departments
(did INTEGER,
 dname CHAR(20),
 budget FLOAT,
 mgr_ssn CHAR(11) NOT NULL,
 since DATE,
PRIMARY KEY (did),
FOREIGN KEY (mgr_ssn) REFERENCES Employees
ON DELETE NO ACTION)
```



E/R to Relational (contd.)

- A **weak entity** can be identified uniquely only by considering the primary key of another (owner) entity.

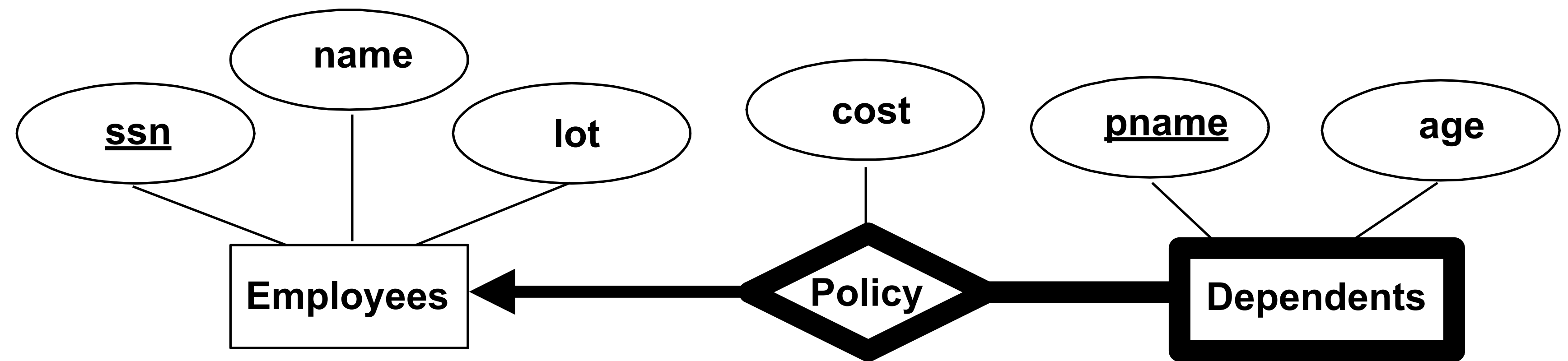
```
CREATE TABLE Employees  
(ssn CHAR(11),  
  name CHAR(20),  
  lot INTEGER,  
  PRIMARY KEY (ssn))
```



E/R to Relational (contd.)

- A **weak entity** can be identified uniquely only by considering the primary key of another (owner) entity.

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

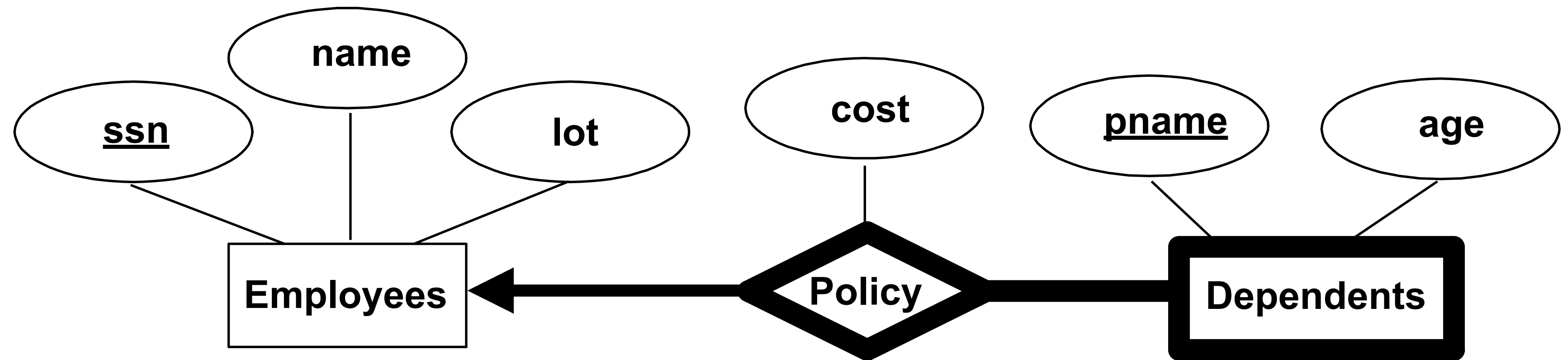


- Weak entity set and identifying relationship set are translated into a single table.
 - When the owner entity is deleted, all owned weak entities must also be deleted.

E/R to Relational (contd.)

- A **weak entity** can be identified uniquely only by considering the primary key of another (owner) entity.

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



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

- Weak entity set and identifying relationship set are translated into a single table.
 - When the owner entity is deleted, all owned weak entities must also be deleted.

Referential Integrity in SQL

- SQL supports various options on deletes and updates.
 - **NO ACTION** (delete/update is rejected)
 - **CASCADE** (delete/update all tuples that refer to deleted/updates tuple)
 - **SET NULL**
 - **SET DEFAULT**

```
CREATE TABLE Works_In
(ssn CHAR(11),
 did INTEGER,
 since DATE,
 PRIMARY KEY (ssn, did),
 FOREIGN KEY (ssn) REFERENCES Employees
      ON DELETE CASCADE ON UPDATE SET DEFAULT,
 FOREIGN KEY (did) REFERENCES Departments
      ON DELETE SET NULL ON UPDATE CASCADE)
```

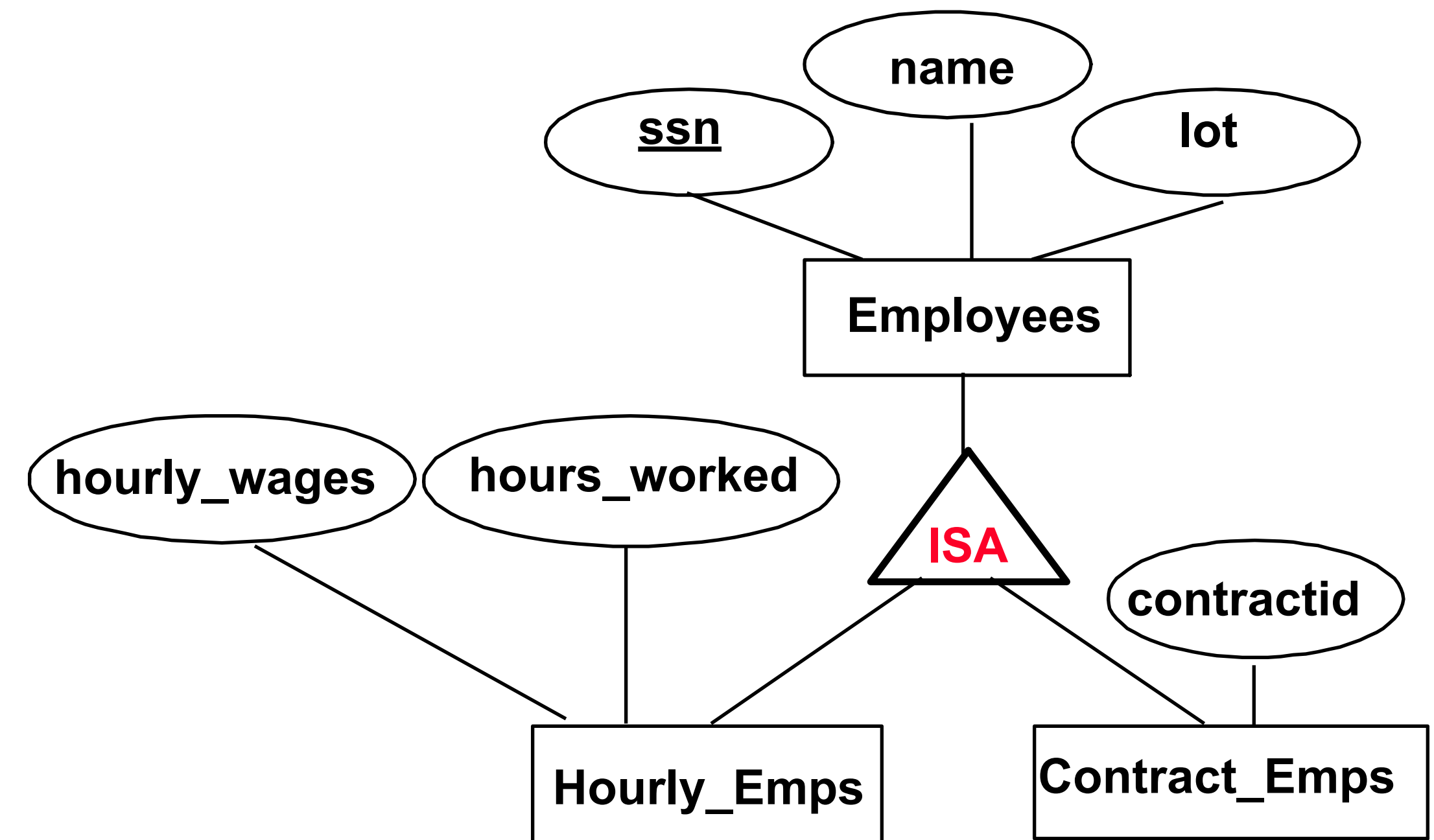
E/R to Relational (contd.)

- 3 relations: Employees, Hourly_Emps and Contract_Emps

- Every hourly employee is recorded in Employees (foreign key).
- For hourly employees, extra information recorded in Hourly_Emps (hourly_wages, hours_worked, ssn)
- Must delete Hourly_Emps tuple if referenced Employees tuple is deleted
- Queries involving all employees easy, those involving just Hourly_Emps must potentially assemble attributes from different tables.

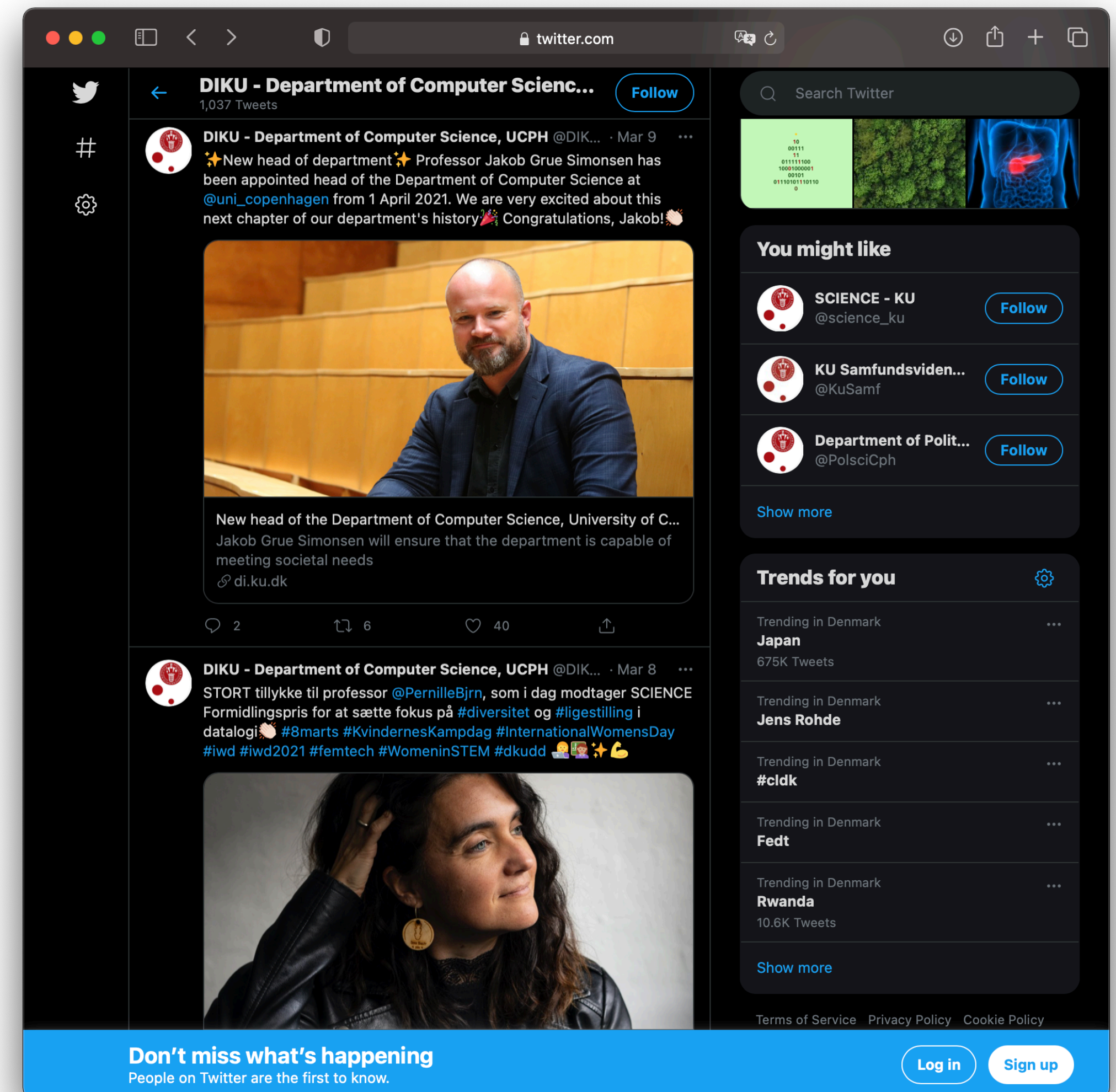
- 2 relations: Just Hourly_Emps and Contract_Emps.

- Hourly_Emps: ssn, name, lot, hourly_wages, hours_worked.
- Each employee must be in one of these two subclasses



Discussion: Relational Twitter

- Let us say you would like to model two tables from Twitter:
 - Users
 - Tweets
- Discuss
 - Which attributes should each table have?
 - What are the keys in these tables? Would you also need foreign keys? If so, which?
 - Do not model Followers for now! But you may think about it. 😊
 - If you have time, try to write the SQL



Steps in creating and using a (relational) database

1. Design schema; create using DDL
2. “Bulk load” initial data
3. Repeat: execute queries and modifications



Steps in creating and using a (relational) database

1. Design schema; create using DDL ✓
2. “Bulk load” initial data
3. Repeat: execute queries and modifications



Steps in creating and using a (relational) database

1. Design schema; create using DDL ✓
2. “Bulk load” initial data
3. Repeat: execute queries and modifications ←



Querying Relational Databases

- **Ad-hoc natural language queries**
 - All students with GPA > 3.7 applying to Stanford and MIT only
 - All engineering departments in CA with < 500 applicants
 - College with highest average accept rate over last 5 years
- Some easy to formulate; some harder
- Some easy for DBMS to execute efficiently; some harder
- “Query language” also used to modify data

Query Languages

- Queries return relations (“compositional”, “closed”)
- Relational model supports simple, powerful query languages:
 - Strong formal foundation based on logic.
 - Allows for much optimization.
- Relational Calculus, Relational Algebra, SQL, Datalog
- Query Languages **!=** programming languages!
 - QLs not expected to be “Turing complete”.
 - QLs not intended to be used for complex calculations.
 - QLs support easy, efficient access to large data sets.

Query Languages

- Relational Calculus
- Relational Algebra
- SQL

Query Languages

- Relational Calculus ➡ **declarative**: describe the result, not how to compute it
- Relational Algebra
- SQL


Query Languages

- Relational Calculus ➡ **declarative**: describe the result, not how to compute it
- Relational Algebra ➡ **operational**: describe the computation as sequence of table transformations
- SQL

Query Languages

- Relational Calculus ➡ **declarative**: describe the result, not how to compute it
- Relational Algebra ➡ **operational**: describe the computation as sequence of table transformations
- SQL ➡ start with RA and move to the “**real**” world

Query Languages

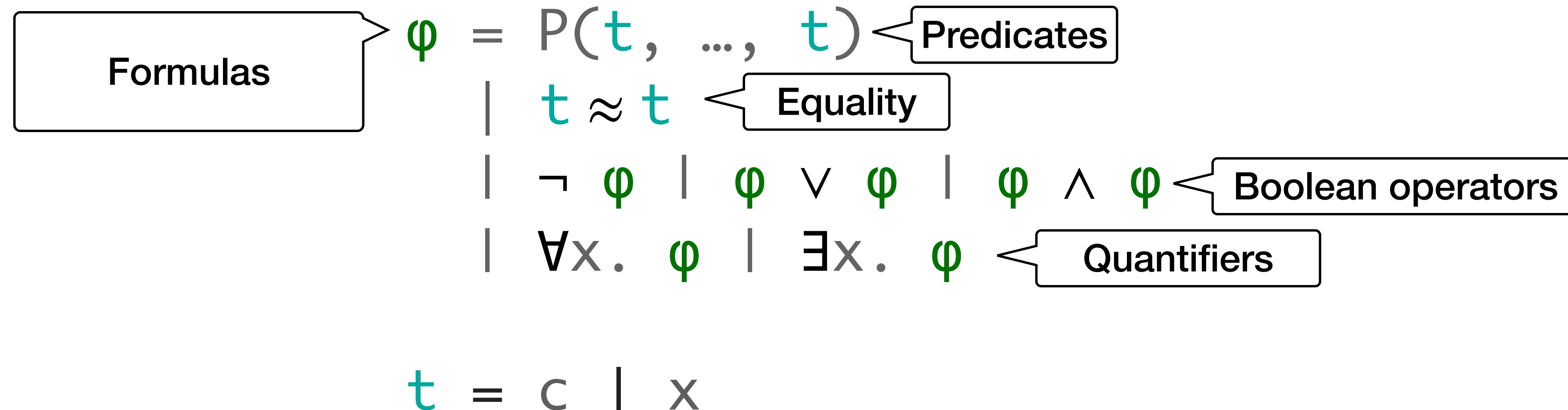
- 
- Relational Calculus ➡ **declarative**: describe the result, not how to compute it
 - Relational Algebra ➡ **operational**: describe the computation as sequence of table transformations
 - SQL ➡ start with RA and move to the “**real**” world

(Domain) Relational Calculus

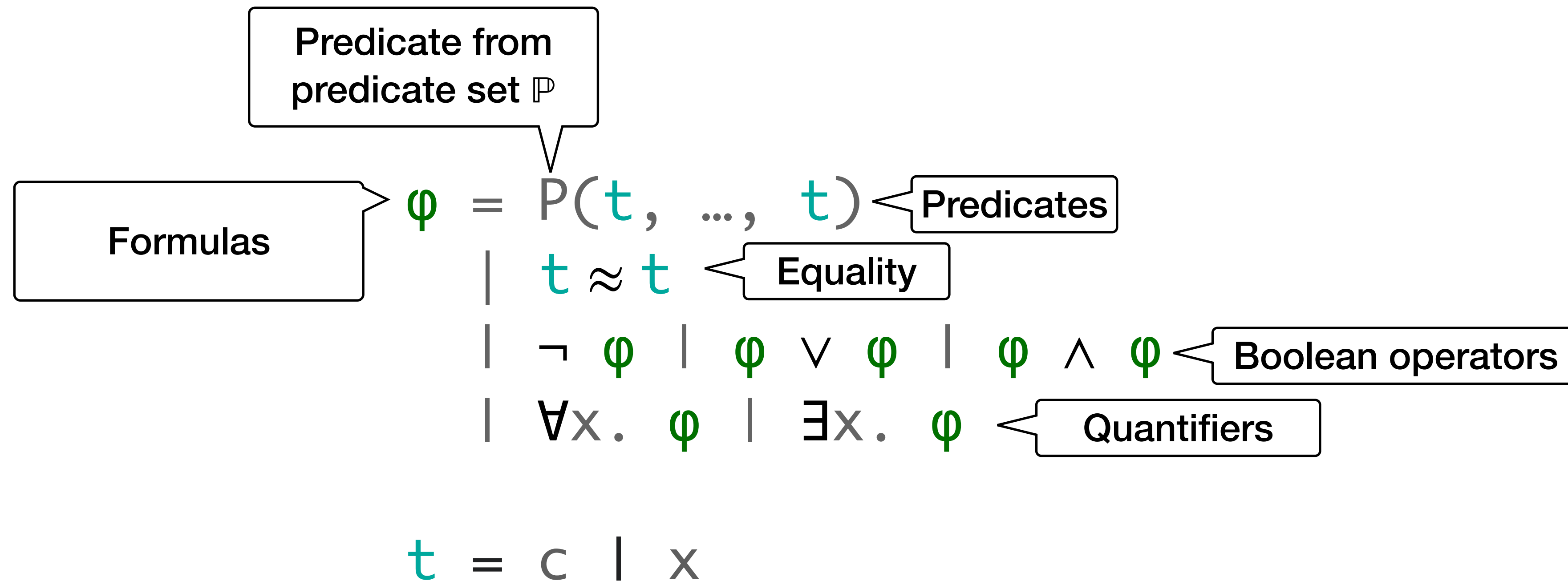
$$\begin{aligned}
 \varphi &= P(t, \dots, t) \\
 &| t \approx t \\
 &| \neg \varphi \quad | \quad \varphi \vee \varphi \quad | \quad \varphi \wedge \varphi \\
 &| \forall x. \varphi \quad | \quad \exists x. \varphi
 \end{aligned}$$

$$t = c \quad | \quad x$$

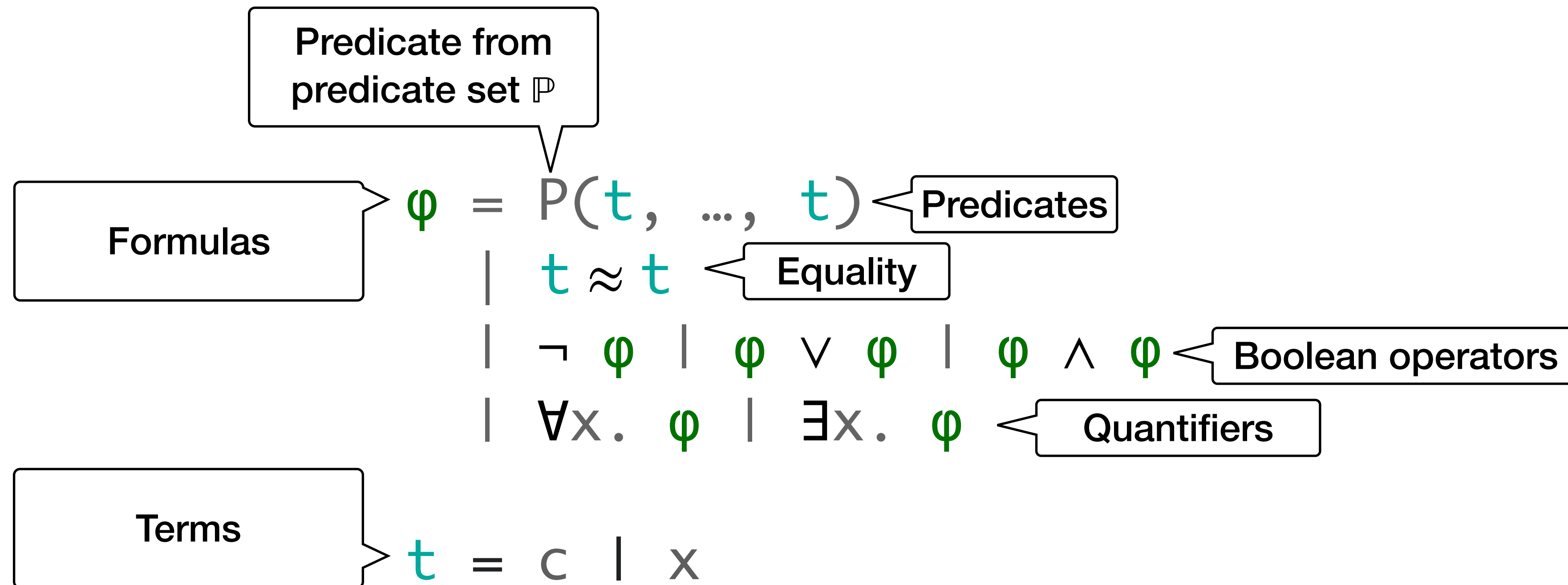
(Domain) Relational Calculus



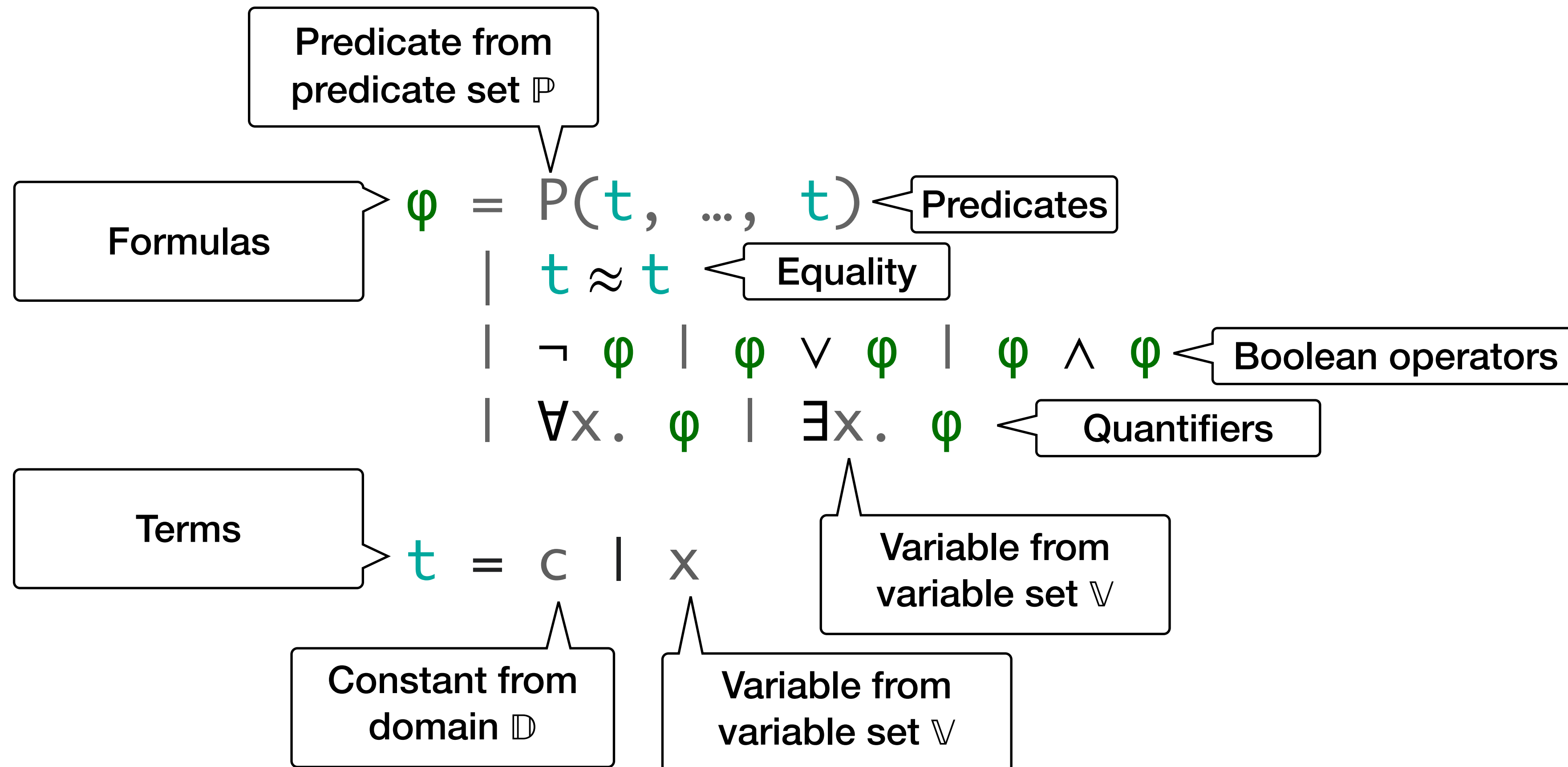
(Domain) Relational Calculus



(Domain) Relational Calculus



(Domain) Relational Calculus



A First Example

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



Employees named "John"

ssn	name	lot
0983763423	John	10

A First Example

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



Employees named “John”

ssn	name	lot
0983763423	John	10

$\text{Employees}(\text{ssn}, \text{name}, \text{lot}) \wedge \text{name} \approx \text{“John”}$

A Different Example?

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



Employees(ssn, “John”, lot)

A Different Example?

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



Employees named “John”

ssn	lot
0983763423	10

Employees(ssn, “John”, lot)

Drop the Lot

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



Employees named "John"

ssn
0983763423

Drop the Lot

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



Employees named “John”

ssn
0983763423

```
∃ lot. Employees(ssn, “John”, lot)
```

A Different Example

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



The names of employees who park at lot 10

name
John
Jane

A Different Example

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



The names of employees who park at lot 10

name
John
Jane

$\exists \text{ssn}, \text{lot}. \text{Employees}(\text{ssn}, \text{name}, \text{lot}) \wedge \text{lot} \approx 10$

A Different Example

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



The names of employees who park at lot 10

name
John
Jane

$\exists \text{ssn}, \text{lot}. \text{Employees}(\text{ssn}, \text{name}, \text{lot}) \wedge \text{lot} \approx 10$

Alternative: $\exists \text{ssn}. \text{Employees}(\text{ssn}, \text{name}, 10)$

A Really Different Example

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



The names of pairs of employees who
park at the same lot

A Really Different Example

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



The names of pairs of employees who park at the same lot

name1	name2
Jane	John
John	Jane
Jane	Jane
John	John
Jill	Jill

A Really Different Example

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



The names of pairs of employees who park at the same lot

name1	name2
Jane	John
John	Jane
Jane	Jane
John	John
Jill	Jill

$\text{Employees}(\text{ssn1}, \text{name1}, \text{lot1}) \wedge \text{Employees}(\text{ssn2}, \text{name2}, \text{lot2})$

A Really Different Example

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



The names of pairs of employees who park at the same lot

name1	name2
Jane	John
John	Jane
Jane	Jane
John	John
Jill	Jill

$\text{Employees}(\text{ssn1}, \text{name1}, \text{lot1}) \wedge \text{Employees}(\text{ssn2}, \text{name2}, \text{lot2})$
 $\wedge \text{lot1} \approx \text{lot2}$

A Really Different Example

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



The names of pairs of employees who park at the same lot

name1	name2
Jane	John
John	Jane
Jane	Jane
John	John
Jill	Jill

$\exists \text{ssn1, ssn2, lot1, lot2.}$

$\text{Employees}(\text{ssn1}, \text{name1}, \text{lot1}) \wedge \text{Employees}(\text{ssn2}, \text{name2}, \text{lot2})$
 $\wedge \text{lot1} \approx \text{lot2}$

A Really Different Example

Employees

ssn	name	lot
0983763423	John	10
9384392483	Jane	10
3743923483	Jill	20



The names of pairs of **different** employees who park at the same lot

name1	name2
Jane	John
John	Jane

$\exists \text{ssn1, ssn2, lot1, lot2.}$

$\text{Employees}(\text{ssn1}, \text{name1}, \text{lot1}) \wedge \text{Employees}(\text{ssn2}, \text{name2}, \text{lot2})$
 $\wedge \text{lot1} \approx \text{lot2} \wedge \neg \text{ssn1} \approx \text{ssn2}$

Try Using RC-eval

Employees(ssn:string, name:string, lot:int)

-> schema

Employees("0983763423", "John", 10)

Employees("9384392483", "Jane", 10)

Employees("3743923483", "Jill", 20)

-> instance

EXISTS lot. Employees(ssn,"John",lot)

-> RC query

Try Using RC-eval

Employees(ssn:string, name:string, lot:int)

-> schema

Employees("0983763423", "John", 10)

Employees("9384392483", "Jane", 10)

Employees("3743923483", "Jill", 20)

-> instance

EXISTS lot. Employees(ssn,"John",lot)

-> RC query



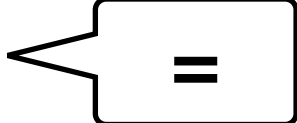


("0983763423")

RC-eval's Relational Calculus

$$\begin{aligned} \varphi = & P(t, \dots, t) \\ & | \quad t \approx t \\ & | \quad \neg \varphi \quad | \quad \varphi \vee \varphi \quad | \quad \varphi \wedge \varphi \\ & | \quad \forall x. \varphi \quad | \quad \exists x. \varphi \end{aligned}$$

$$t = c \quad | \quad x$$

RC-eval's Relational Calculus

$\varphi = P(t, \dots, t)$
| $t \approx t$ 
| $\neg \varphi$ | $\varphi \vee \varphi$ | $\varphi \wedge \varphi$ 
| $\forall x. \varphi$ | $\exists x. \varphi$ 

$t = c \mid x$

Relational Calculus Semantics

Fix a database (mapping of table names to relations) DB

$v \models P(t_1, \dots, t_n)$	$\iff (v(t_1), \dots, v(t_n)) \in DB(P)$
$v \models t_1 \approx t_2$	$\iff v(t_1) = v(t_2)$
$v \models \neg \varphi$	$\iff v \not\models \varphi$
$v \models \varphi \vee \psi$	$\iff v \models \varphi \text{ or } v \models \psi$
$v \models \varphi \wedge \psi$	$\iff v \models \varphi \text{ and } v \models \psi$
$v \models \forall x. \varphi$	$\iff v(x \mapsto c) \models \varphi \text{ for all } c \in \mathbb{D}$
$v \models \exists x. \varphi$	$\iff v(x \mapsto c) \models \varphi \text{ for some } c \in \mathbb{D}$

Relational Calculus Semantics

Fix a database (mapping of table names to relations) DB

Maps variables to domain elements

$$v \models P(t_1, \dots, t_n) \iff (v(t_1), \dots, v(t_n)) \in DB(P)$$

$$v \models t_1 \approx t_2 \iff v(t_1) = v(t_2)$$

$$v \models \neg \varphi \iff v \not\models \varphi$$

$$v \models \varphi \vee \psi \iff v \models \varphi \text{ or } v \models \psi$$

$$v \models \varphi \wedge \psi \iff v \models \varphi \text{ and } v \models \psi$$

$$v \models \forall x. \varphi \iff v(x \mapsto c) \models \varphi \text{ for all } c \in \mathbb{D}$$

$$v \models \exists x. \varphi \iff v(x \mapsto c) \models \varphi \text{ for some } c \in \mathbb{D}$$

Free Variables

$$\begin{aligned} \text{fv}(P(t_1, \dots, t_n)) &= \text{fv}(t_1) \cup \dots \cup \text{fv}(t_n) \\ \text{fv}(t_1 \approx t_2) &= \text{fv}(t_1) \cup \text{fv}(t_2) \\ \text{fv}(\neg \varphi) &= \text{fv}(\varphi) \\ \text{fv}(\varphi \vee \psi) &= \text{fv}(\varphi) \cup \text{fv}(\psi) \\ \text{fv}(\varphi \wedge \psi) &= \text{fv}(\varphi) \cup \text{fv}(\psi) \\ \text{fv}(\forall x. \varphi) &= \text{fv}(\varphi) - \{x\} \\ \text{fv}(\exists x. \varphi) &= \text{fv}(\varphi) - \{x\} \end{aligned}$$

We also write $\text{fv}(\varphi)$ for the **ordered tuple** of the free variables of a formula

Query Result

$$\{(\mathbf{v}(x_1), \dots, \mathbf{v}(x_n)) \mid \mathbf{v} \models \varphi \text{ and } \mathbf{fv}(\varphi) = (x_1, \dots, x_n)\}$$

Competition Task: RC Golf

Give an RC query φ that evaluates to a finite result

GOOD

x

(21)
(45)
(78)
(120)
(171)
(231)
(300)
(351)
(465)
(561)
(666)
(780)
(903)
(1035)
(1176)
(1326)
(1485)
(1653)
(1830)
(2016)

Competition Task: RC Golf

Give an RC query ϕ that evaluates to a finite result
containing all 20 tuples from the left (**GOOD**) set and

GOOD

x

(21)
(45)
(78)
(120)
(171)
(231)
(300)
(351)
(465)
(561)
(666)
(780)
(903)
(1035)
(1176)
(1326)
(1485)
(1653)
(1830)
(2016)

Competition Task: RC Golf

Give an RC query ϕ that evaluates to a finite result
containing all 20 tuples from the left (**GOOD**) set and
not containing any of the tuples from the right (**BAD**) set

BAD

x

(150)
(162)
(269)
(415)
(464)
(599)
(764)
(876)
(962)
(1095)
(1099)
(1151)
(1197)
(1422)
(1537)
(1552)
(1677)
(1792)
(1847)
(1918)

GOOD

x

(21)
(45)
(78)
(120)
(171)
(231)
(300)
(351)
(465)
(561)
(666)
(780)
(903)
(1035)
(1176)
(1326)
(1485)
(1653)
(1830)
(2016)

Competition Task: RC Golf

Give an RC query ϕ that evaluates to a finite result
containing all 20 tuples from the left (**GOOD**) set and
not containing any of the tuples from the right (**BAD**) set
over the following database:

schema
P(int)

Instance
P(0)(1)

BAD

x

(150)
(162)
(269)
(415)
(464)
(599)
(764)
(876)
(962)
(1095)
(1099)
(1151)
(1197)
(1422)
(1537)
(1552)
(1677)
(1792)
(1847)
(1918)

GOOD

x

(21)
(45)
(78)
(120)
(171)
(231)
(300)
(351)
(465)
(561)
(666)
(780)
(903)
(1035)
(1176)
(1326)
(1485)
(1653)
(1830)
(2016)

Competition Task: RC Golf

BAD

x

(150)
(162)
(269)
(415)
(464)
(599)
(764)
(876)
(962)
(1095)
(1099)
(1151)
(1197)
(1422)
(1537)
(1552)
(1677)
(1792)
(1847)
(1918)

Give an RC query ϕ that evaluates to a finite result
containing all 20 tuples from the left (**GOOD**) set and
not containing any of the tuples from the right (**BAD**) set
over the following database:

schema
P(int)

Instance
P(0)(1)

Shortest query (# characters) wins!

What should we learn today?

- Define and explain the relational data model
- Define and explain the main classes of integrity constraints in the relational model, in particular key and foreign key constraints
- Model relational schemas and express them in SQL
- Explain what a query language is
- Identify the main relational database query languages
- Formulate queries in the relational calculus

