

## The Relational Model

CMU SCS 15-415/615  
C. Faloutsos – A. Pavlo  
Lecture #3  
R & G, Chap. 3

## Roadmap



- **Introduction**
- **Integrity constraints (IC)**
- **Enforcing IC**
- **Querying Relational Data**
- **ER to tables**
- **Intro to Views**
- **Destroying/altering tables**

Faloutsos - Pavlo, 15-415/615

2

## Why Study the Relational Model?

- **Most widely used model.**
  - Vendors: IBM/Informix, Microsoft, Oracle, Sybase, etc.
- **“Legacy systems” in older models**
  - e.g., IBM’s IMS
- **Object-oriented concepts have merged in**
  - *object-relational model*
    - Informix->IBM DB2, Oracle

Faloutsos - Pavlo, 15-415/615

3

## Relational Database: Definitions

- *Relational database*: a set of *relations*
- (relation = table)
- specifically


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

Faloutsos - Pavlo, 15-415/615

4

## Relational Database: Definitions

- **Relation**: made up of 2 parts:
  - **Schema** : specifies name of relation, plus name and type of each column.
  - **Instance** : a **table**, with rows and columns.
    - #rows = *cardinality*
    - #fields = *degree / arity*



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

Faloutsos - Pavlo, 15-415/615

5

## Relational Database: Definitions

- relation: a **set** of rows or **tuples**.
  - all rows are distinct
  - no order among rows (why?)

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

Faloutsos - Pavlo, 15-415/615

6

## Ex: Instance of Students Relation

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

- Cardinality = 3, arity = 5 ,
- all rows distinct
- Q: do values in a column need to be distinct?

Faloutsos - Pavlo, 15-415/615

7

## SQL - A language for Relational DBs

- **SQL\*** (a.k.a. "Sequel"), **standard language**
- **Data Definition Language (DDL)**
  - create, modify, delete relations
  - specify constraints
  - administer users, security, etc.
  - E.g.:

```
create table student
(ssn fixed, name char(20));
```

\* **Structured Query Language**

Faloutsos - Pavlo, 15-415/615

8

## SQL - A language for Relational DBs

- **Data Manipulation Language (DML)**
  - Specify *queries* to find tuples that satisfy criteria
  - add, modify, remove tuples

```
select * from student ;
```

```
update takes set grade=4
  where name='smith'
  and cid = 'db';
```

## SQL Overview

- CREATE TABLE <name> ( <field> <domain>, ... )
- INSERT INTO <name> (<field names>) VALUES (<field values>)
- DELETE FROM <name> WHERE <condition>

## SQL Overview

- UPDATE <name>  
SET <field name> = <value>  
WHERE <condition>
- SELECT <fields> ←  
FROM <name>  
WHERE <condition>

## Creating Relations in SQL

- **Creates the Students relation.**

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

## Creating Relations in SQL

- **Creates the Students relation.**
  - Note: 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 FLOAT)
```

Faloutsos - Pavlo, 15-415/615

13

## Table Creation (continued)

- **Another example:**

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

Faloutsos - Pavlo, 15-415/615

14

## Adding and Deleting Tuples

- **Can insert a single tuple using:**

```
INSERT INTO Students
(sid, name, login, age, gpa)
VALUES
('53688', 'Smith', 'smith@cs',
18, 3.2)
```

Faloutsos - Pavlo, 15-415/615

15

## Adding and Deleting Tuples

- 'mass'-delete (**all** Smiths!):

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

Faloutsos - Pavlo, 15-415/615

16

## Roadmap



- Introduction
- **Integrity constraints (IC)**
- Enforcing IC
- Querying Relational Data
- ER to tables
- Intro to Views
- Destroying/altering tables

## Keys

- Keys help associate tuples in different relations
- Keys are one form of integrity constraint (IC)

Enrolled			Students				
sid	cid	grade	sid	name	login	age	gpa
53666	15-101	C	53666	Jones	jones@cs	18	3.4
53666	18-203	B	53688	Smith	smith@cs	18	3.2
53650	15-112	A	53650	Smith	smith@math	19	3.8
53666	15-105	B					

## (Motivation: )

- In flat files, how would you check for duplicate ssn, in a student file?
- (horror stories, if ssn is duplicate?)

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

## Keys

- Keys help associate tuples in different relations
- Keys are one form of integrity constraint (IC)

Enrolled			Students				
sid	cid	grade	sid	name	login	age	gpa
53666	15-101	C	53666	Jones	jones@cs	18	3.4
53666	18-203	B	53688	Smith	smith@cs	18	3.2
53650	15-112	A	53650	Smith	smith@math	19	3.8
53666	15-105	B					

**FOREIGN Key**      **PRIMARY Key**

## Primary Keys

- **A set of fields is a *superkey* if:**
  - No two distinct tuples can have same values in all key fields
- **A set of fields is a *key* for a relation if :**
  - minimal superkey

Student (ssn, name, address)

{ssn,name}: superkey  
 {ssn}: superkey, AND key  
 {name}: not superkey

## Primary Keys

- **what if >1 key for a relation?**

## Primary Keys

- **what if >1 key for a relation?**
  - one of the keys is chosen (by DBA) to be the **primary key**. Other keys are called **candidate** keys..
  - Q: example of >1 superkeys?

## Primary Keys

- **what if >1 key for a relation?**
  - one of the keys is chosen (by DBA) to be the **primary key**. Other keys are called **candidate** keys..
  - Q: example of >1 superkeys?
  - A1: student: {ssn}, {student-id#}, {driving license#, state}
  - A2: Employee: {ssn}, {phone#}, {room#}
  - A3: computer: {mac-address}, {serial#}

## Primary Keys

- **E.g.**

- *sid* is a key for Students.
- What about *name*?
- The set  $\{sid, gpa\}$  is a superkey.

## Syntax:

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

## Syntax:

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

PRIMARY KEY == UNIQUE, NOT NULL

## Drill:

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

### Drill:

```

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

Q: what does this mean?

### Primary and Candidate Keys in SQL

```

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

"Students can take only one course, and no two students in a course receive the same grade."

### Foreign Keys

Enrolled

sid	cid	grade
53666	15-101	C
53666	18-203	B
53650	15-112	A
53666	15-105	B

Students

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

### Foreign Keys, Referential Integrity

- **Foreign key**: Set of fields `referring` to a tuple in another relation.
  - Must correspond to the primary key of the other relation.
  - Like a `logical pointer`.
- **foreign key constraints enforce referential integrity** (i.e., no dangling references.)



## Foreign Keys in SQL

Example: Only existing students may enroll for courses.

- *sid* is a foreign key referring to **Students**:

Enrolled			Students				
sid	cid	grade	sid	name	login	age	gpa
53666	15-101	C	53666	Jones	jones@cs	18	3.4
53666	18-203	B	53688	Smith	smith@cs	18	3.2
53650	15-112	A	53650	Smith	smith@math	19	3.8
53666	15-105	B					

Faloutsos - Pavlo, 15-415/615

33

## Foreign Keys in SQL

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

Enrolled			Students				
sid	cid	grade	sid	name	login	age	gpa
53666	15-101	C	53666	Jones	jones@cs	18	3.4
53666	18-203	B	53688	Smith	smith@cs	18	3.2
53650	15-112	A	53650	Smith	smith@math	19	3.8
53666	15-105	B					

Faloutsos - Pavlo, 15-415/615

34

## Roadmap

- Introduction
- Integrity constraints (IC)
- **Enforcing IC**
- Querying Relational Data
- ER to tables
- Intro to Views
- Destroying/altering tables



Faloutsos - Pavlo, 15-415/615

35

## Enforcing Referential Integrity

- **Subtle issues:**
- **What should be done if an Enrolled tuple with a non-existent student id is inserted?**

Enrolled			Students				
sid	cid	grade	sid	name	login	age	gpa
53666	15-101	C	53666	Jones	jones@cs	18	3.4
53666	18-203	B	53688	Smith	smith@cs	18	3.2
53650	15-112	A	53650	Smith	smith@math	19	3.8
53666	15-105	B					

Faloutsos - Pavlo, 15-415/615

36

## Enforcing Referential Integrity

- **Subtle issues:**
- **What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)**

Enrolled			Students				
sid	cid	grade	sid	name	login	age	gpa
53666	15-101	C	53666	Jones	jones@cs	18	3.4
53666	18-203	B	53688	Smith	smith@cs	18	3.2
53650	15-112	A	53650	Smith	smith@math	19	3.8
53666	15-105	B					

Faloutsos - Pavlo, 15-415/615

37

## Enforcing Referential Integrity

- **Subtle issues, cont'd:**
- **What should be done if a Student's tuple is deleted?**

Enrolled			Students				
sid	cid	grade	sid	name	login	age	gpa
53666	15-101	C	53666	Jones	jones@cs	18	3.4
53666	18-203	B	53688	Smith	smith@cs	18	3.2
53650	15-112	A	53650	Smith	smith@math	19	3.8
53666	15-105	B					

Faloutsos - Pavlo, 15-415/615

38

## Enforcing Referential Integrity

- **Subtle issues, cont'd:**
- **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'*.)

Faloutsos - Pavlo, 15-415/615

39

## Enforcing Referential Integrity

- **Similar issues arise if primary key of Students tuple is updated.**

Faloutsos - Pavlo, 15-415/615

40

## 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.

## Integrity Constraints (ICs)

- A ***legal*** instance of a relation: **satisfies all specified ICs.**
  - DBMS should not allow illegal instances.
- **we prefer that ICs are enforced by DBMS (as opposed to ?)**
  - Blocks data entry errors, too!

## Where do ICs Come From?

## Where do ICs Come From?

- the application!

## Where do ICs Come From?

- Subtle point: 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!
  - Eg., *name* is not a key,
  - but the assertion that *sid* is a key is given to us.

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

Faloutsos - Pavlo, 15-415/615

45

## Where do ICs Come From?

- Key and foreign key ICs are the most common; more general ICs supported too.

Faloutsos - Pavlo, 15-415/615

46

## Roadmap

- Introduction
- Integrity constraints (IC)
- Enforcing IC
- Querying Relational Data
- **ER to tables**
- Intro to Views
- Destroying/altering tables



Faloutsos - Pavlo, 15-415/615

47

## ER to tables outline:

- **strong entities**
- **weak entities**
- **(binary) relationships**
  - 1-to-1, 1-to-many, etc
  - total/partial participation
- **ternary relationships**
- **ISA-hierarchies**
- **aggregation**



Faloutsos - Pavlo, 15-415/615

48

### Logical DB Design: ER to Relational

- (strong) entity sets to tables.

Diagram showing an entity **Employees** with attributes **ssn**, **name**, and **lot**.

Faloutsos - Pavlo, 15-415/615 49

### Logical DB Design: ER to Relational

- (strong) entity sets to tables.

Ssn	Name	Lot
123-22-6666	Attishoo	48
233-31-5363	Smiley	22
131-24-3650	Smethurst	35

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

Faloutsos - Pavlo, 15-415/615 50

### Relationship Sets to Tables

Many-to-many:

Diagram showing a many-to-many relationship **Works\_In** between **Employees** and **Departments**. **Employees** has attributes **ssn**, **name**, and **lot**. **Departments** has attributes **did**, **dname**, and **budget**. The relationship **Works\_In** has the attribute **since**.

Faloutsos - Pavlo, 15-415/615 51

### Relationship Sets to Tables

Many-to-many:

Ssn	Name	Lot
123-22-6666	Attishoo	48
233-31-5363	Smiley	22
131-24-3650	Smethurst	35

Ssn	did	since
123-22-6666	51	1/1/91
123-22-6666	56	3/3/93
233-31-5363	51	2/2/92

Faloutsos - Pavlo, 15-415/615 52

### Relationship Sets to Tables

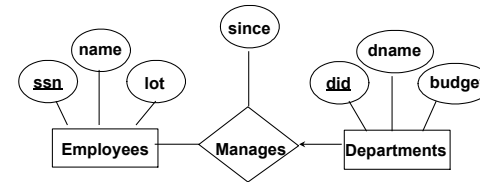
- key of **many-to-many** relationships:
  - Keys from participating entity sets (as foreign keys).

```
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)
```

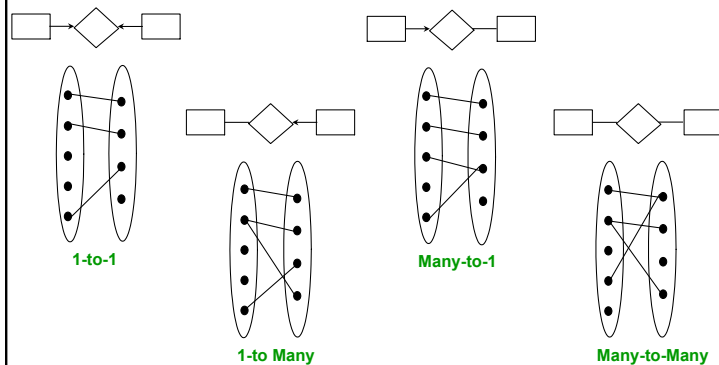
Ssn	did	since
123-22-6666	51	1/1/91
123-22-6666	56	3/3/93
233-31-5363	51	2/2/92

### Review: Key Constraints in ER

- **1-to-many:**



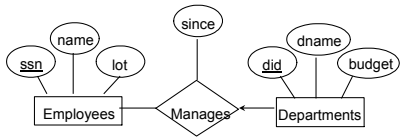
### Review: Key Constraints in ER



### ER to tables - summary of basics

- **strong entities:**
  - key -> primary key
- **(binary) relationships:**
  - get keys from all participating entities - pr. key:
  - 1-to-1 -> either key (other: 'cand. key')
  - 1-to-N -> the key of the 'N' part
  - M-to-N -> both keys

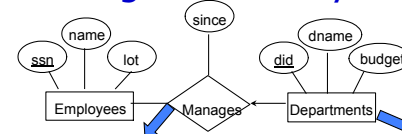
### A subtle point (1-to-many)



Faloutsos - Pavlo, 15-415/615

57

### Translating ER with Key Constraints



```
CREATE TABLE Manages(
  ssn CHAR(11),
  did INTEGER,
  since DATE,

  PRIMARY KEY (did),
  FOREIGN KEY (ssn)
  REFERENCES Employees,
  FOREIGN KEY (did)
  REFERENCES Departments)
```

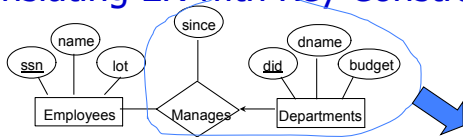
```
CREATE TABLE Departments(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  PRIMARY KEY (did),
)
```

Two-table-solution

Faloutsos - Pavlo, 15-415/615

58

### Translating ER with Key Constraints



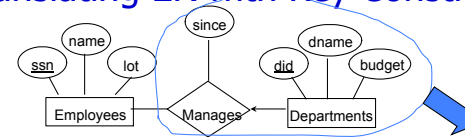
```
CREATE TABLE Dept_Mgr(
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  dname CHAR(20),
  budget REAL,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn)
  REFERENCES Employees)
```

Single-table-solution

Faloutsos - Pavlo, 15-415/615

59

### Translating ER with Key Constraints



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

Faloutsos - Pavlo, 15-415/615

60

## Pros and cons?

Faloutsos - Pavlo, 15-415/615

61

## Drill:

What if the toy department has no manager (yet) ?

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

Faloutsos - Pavlo, 15-415/615

62

## Drill:

What if the toy department has no manager (yet) ?

A: one-table solution can not handle that.

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

Faloutsos - Pavlo, 15-415/615

63

## ER to tables outline:



- ✓ **strong entities**
- **weak entities**
- **(binary) relationships**
  - ✓ 1-to-1, 1-to-many, etc
  - total/partial participation
- **ternary relationships**
- **ISA-hierarchies**
- **aggregation**

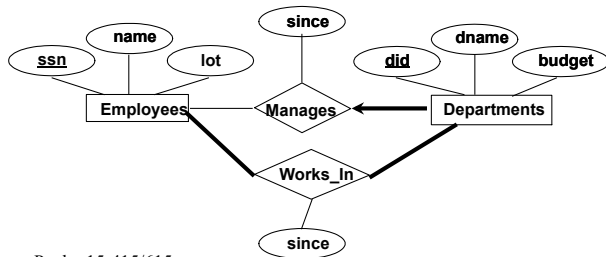
Faloutsos - Pavlo, 15-415/615

64



## 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!)



Faloutsos - Pavlo, 15-415/615

65

## 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)
```

Faloutsos - Pavlo, 15-415/615

66

## Participation Constraints in SQL

- **Total participation ('no action' -> do NOT do the delete)**
- **Ie, a department MUST have a manager**

```
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)
```

Faloutsos - Pavlo, 15-415/615

67

## Participation Constraints in SQL

- **Partial participation, ie, a department may be headless**

```
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 SET NULL)
```

Faloutsos - Pavlo, 15-415/615

68

## ER to tables outline:

- ✓ strong entities
- • weak entities
- (binary) relationships
  - ✓ 1-to-1, 1-to-many, etc
  - ✓ total/partial participation
- ternary relationships
- ISA-hierarchies
- aggregation

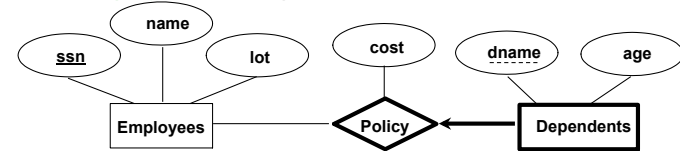


Faloutsos - Pavlo, 15-415/615

69

## Review: Weak Entities

- A **weak entity** can be identified uniquely only by considering the primary key of another (**owner**) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.

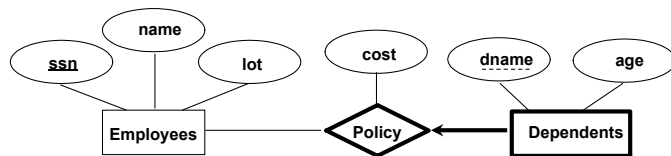


Faloutsos - Pavlo, 15-415/615

70

## Review: Weak Entities

How to turn 'Dependents' into a table?



Faloutsos - Pavlo, 15-415/615

71

## Translating Weak Entity Sets

- **Weak entity set and identifying relationship set are translated into a single table.**

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

Faloutsos - Pavlo, 15-415/615

72

## Translating Weak Entity Sets

- **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 (-> 'CASCADE')

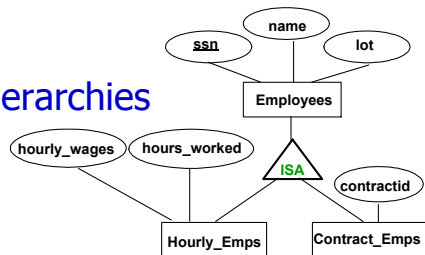
```
CREATE TABLE Dep_Policy (
  dname CHAR(20),
  age INTEGER,
  cost REAL,
  ssn CHAR(11) NOT NULL,
  PRIMARY KEY (dname, ssn),
  FOREIGN KEY (ssn) REFERENCES Employees,
  ON DELETE CASCADE)
```

## ER to tables outline:



- ✓ strong entities
- ✓ weak entities
- (binary) relationships
  - ✓ 1-to-1, 1-to-many, etc
  - ✓ total/partial participation
- ternary relationships
- ➔ ISA-hierarchies
- aggregation

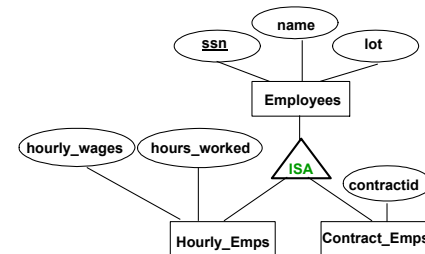
## Review: ISA Hierarchies



- **Overlap constraints:** Can Joe be an Hourly\_Emps as well as a Contract\_Emps entity? (*Allowed/disallowed*)
- **Covering constraints:** Does every Employees entity also have to be an Hourly\_Emps or a Contract\_Emps entity? (*Yes/no*)

## Drill:

- **What would you do?**



### Translating ISA Hierarchies to Relations

- General approach: 3 relations: Employees, Hourly\_Emps and Contract\_Emps.
  - how many times do we record an employee?
  - what to do on deletion?
  - how to retrieve **all** info about an employee?

EMP (ssn, name, lot)


H\_EMP(ssn, h\_wg, h\_wk)


CONTR(ssn, cid)

--	--

### Translating ISA Hierarchies to Relations

- Alternative: 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.

EMP (ssn, name, lot)


H\_EMP(ssn, h\_wg, h\_wk, name, lot)    CONTR(ssn, cid, name, lot)


--	--	--

Notice: 'black' is gone!

### ER to tables outline:

- ✓ strong entities
- ✓ weak entities
- ✓ (binary) relationships
  - ✓ 1-to-1, 1-to-many, etc
  - ✓ total/partial participation
- ➔ ternary relationships
- ✓ ISA-hierarchies
- aggregation



### Ternary relationships; aggregation

- rare
- keep keys of all participating entity sets

(or: avoid such situations:  
break into 2-way relationships or  
add an auto-generated key  
)

## Roadmap



- Introduction
- Integrity constraints (IC)
- Enforcing IC
- Querying Relational Data
- ER to tables
- **Intro to Views**
- Destroying/altering tables

## Views

- **Virtual tables**

```
CREATE VIEW YoungActiveStudents
(name,grade)
AS SELECT S.name, E.grade
FROM Students S, Enrolled E
WHERE S.sid=E.sid and S.age<21
```

- DROP VIEW

## Views and Security

- **DBA: grants authorization to a view for a user**
- **user can only see the view - nothing else**

## Roadmap



- Introduction
- Integrity constraints (IC)
- Enforcing IC
- Querying Relational Data
- ER to tables
- Intro to Views
- **Destroying/altering tables**

## Table changes

- DROP TABLE
- ALTER TABLE, e.g.  

```
ALTER TABLE students
  ADD COLUMN maiden-name CHAR(10)
```

## Relational Model: Summary

- A **tabular** representation of data.
- Simple and intuitive; widely used
- Integrity constraints can be specified by the DBA, based on customer specs. DBMS checks for violations.
  - Two important ICs: **primary** and **foreign** keys
  - also: not null, unique
  - In addition, we *always* have domain constraints.
- **Mapping** from ER to Relational is (fairly) straightforward:

## ER to tables - summary of basics

- **strong entities:**
  - key -> primary key
- **(binary) relationships:**
  - get keys from all participating entities - pr. key:
  - 1:1 -> either key
  - 1:N -> the key of the 'N' part
  - M:N -> both keys
- **weak entities:**
  - strong key + partial key -> primary key
  - ..... ON DELETE CASCADE

## ER to tables - summary of advanced

- **total/partial participation:**
  - NOT NULL; ON DELETE NO ACTION
- **ternary relationships:**
  - get keys from all; decide which one(s) -> prim. key
- **aggregation: like relationships**
- **ISA:**
  - 2 tables ('total coverage')
  - 3 tables (most general)