SDSC5003

The Relational Model

Overview

- The Relational Model
- Creating Relations in SQL

Relational Model

- **RG Chapter 3**
- GUW Chapter 2

Relational Database: Definitions

- *Relational database:* a set of *relations*
- *Relation = Instance + Schema*
  - *Instance* : a *table,* with rows and columns.
    #Rows = *cardinality,* #fields = *degree or arity.*
  - *Schema* : specifies name of relation, plus name and type of each column.
    - e.g., Students(<u>*sid*: string</u>, *name*: string, *login*: string, *age*: integer, *gpa*: real).
- Can think of a relation as a *set* of rows or *tuples* (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?

Quick Question

How many distinct tuples are in a relation instance with cardinality 22?

SQL

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)
  - SQL-99 (major extensions)
  - ...SQL-2011

The SQL Query Language: Preview

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

```
SELECT  *
FROM  Students S
WHERE  S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

- To find just names and logins, replace the first line

```
SELECT  S.name, S.login
```

Exercise

1. Modify this query so that only the login column is included in the answer.
2. If the clause WHERE S.gpa >= 3.3 is added to the original query, what is the set of tuples in the answer?
3. What if the clause WHERE S.gpa > Jones is added?

```
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

Querying Multiple Relations

- What does the following query compute?

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade=“A”
```

Given the following instance of Enrolled:

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53688	Topology112	A
53666	History105	B

we get:

S.name	E.cid
Smith	Topology112

Creating Tables in SQL

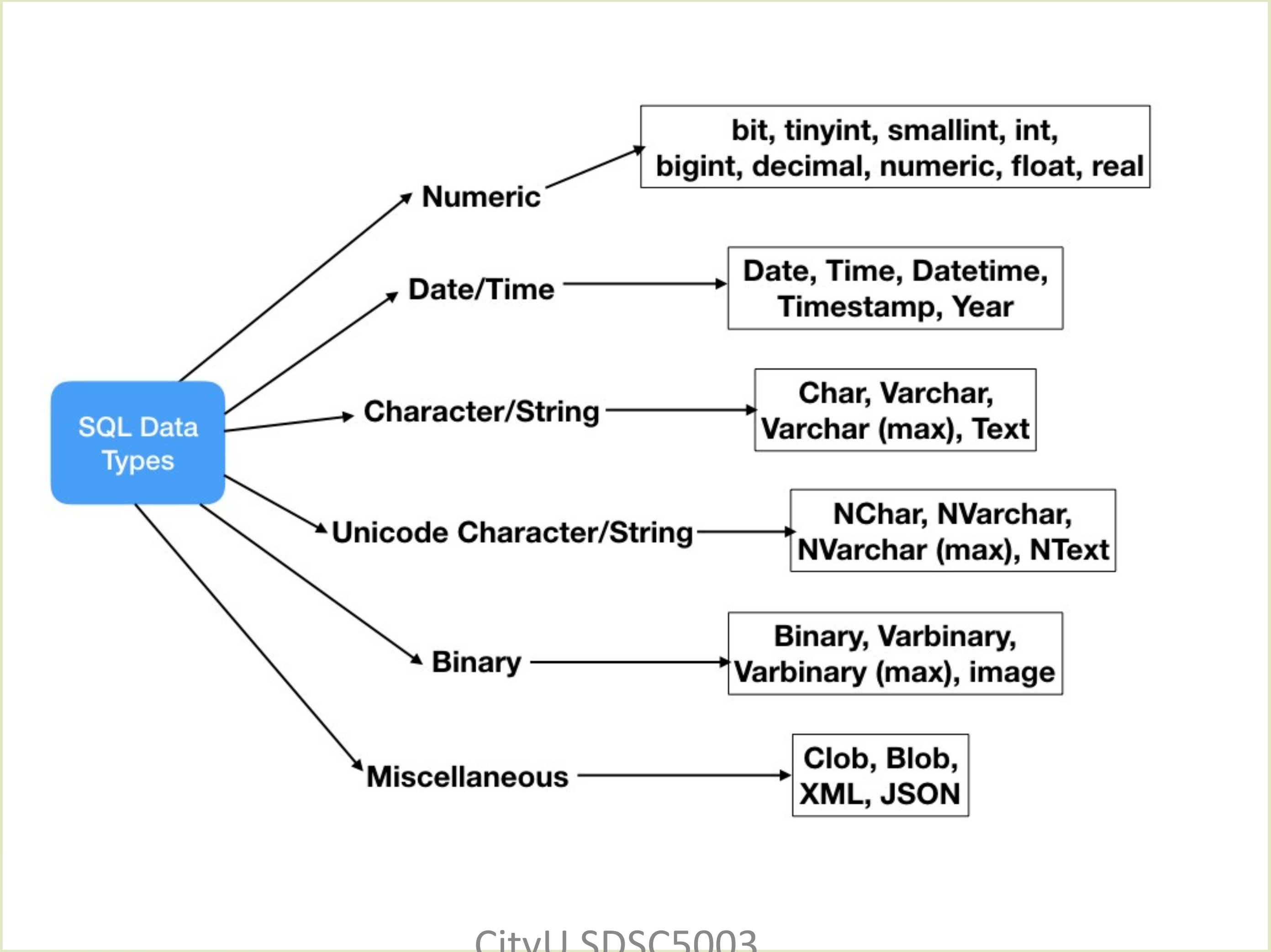
Creating Tables

- Creates the Students relation.
  - Specify the table name and field names
  - The type (domain) of each field is specified.

- The Enrolled table holds information about courses that students take.

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

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

SQL Data Types


SQL Data Types
Numeric
bit, tinyint, smallint, int, bigint, decimal, numeric, float, real
Date/Time
Date, Time, Datetime, Timestamp, Year
Character/String
Char, Varchar, Varchar (max), Text
Unicode Character/String
NChar, NVarchar, NVarchar (max), NText
Binary
Binary, Varbinary, Varbinary (max), image
Miscellaneous
Clob, Blob, XML, JSON

Adding and Deleting Tuples

- Can insert a single tuple using:

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

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

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

- Delete all records

```
DELETE
FROM Students
```

Destroying and Altering Relations

```
DROP TABLE Students
```

- Destroys the relation Students.
- Delete not only the records, but also the schema information.

```
ALTER TABLE Students
ADD year INTEGER
```

- Add a column

```
ALTER TABLE Students
Drop year
```

- Delete a column

Modifying Records

- Note that the **WHERE** statement is evaluated *before* the **SET** statement
- An **UPDATE** statement may affect more than one record

```
UPDATE Customer
SET age = 37
WHERE sin = '111'
```

Specifying Constraints in SQL

Integrity Constraints (ICs)

- An integrity constraint restricts the data that can be stored in a DB
  - To prevent invalid data being added to the DB
    - e.g. two people with the same SIN or
    - someone with a negative age
- When a DB schema is defined, the associated integrity constraints should also be specified
- A DBMS checks every update to ensure that it does not violate any integrity constraints
  - A *legal* instance of a relation is one that satisfies all specified ICs.

Types of Integrity Constraints

- Domain Constraints
  - Specified when tables are created by selecting the type of the data
    - e.g. **`age INTEGER`**
- Key Constraints
  - Identifies primary keys and other candidate keys
  - Many to one or one to many
- Foreign Key Constraints
  - References primary keys of other tables
- General constraints

Primary Key Constraints

- A set of fields is a (candidate) *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 is >1 key for a relation, one of the keys is chosen to be the *primary key*.
- Examples.
  - *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."
- What does the Unique constraint do? Is this a good idea?

```
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) )
```

Exercise

- Assume that a patient can be uniquely identified by either SIN or MSP number
  - SIN is chosen as the primary key

Exercise

- Assume that a patient can be uniquely identified by either SIN or MSP number
  - SIN is chosen as the primary key

```
CREATE TABLE Patient (
    sin       CHAR(11),
    msp       CHAR(15),
    fName     CHAR(20),
    lName     CHAR(20),
    age       INTEGER,
    UNIQUE (msp)
    PRIMARY KEY (sin) )
```

Exercise (cont.)

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

1. Give an example of an attribute (or set of attributes) that you can deduce is *not* a candidate key, if this instance is legal.
2. Is there any example of an attribute (or set of attributes) that you can deduce *is* a candidate key?
3. Does every relational schema have *some* candidate key?

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 in another relation.
- Like a pointer.
- E.g. *sid* is a foreign key referring to Students:
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.

Foreign Keys in SQL

- 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 )
```

Enrolled

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

Students

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

Enforcing Referential Integrity

- *sid* in Enrolled is a foreign key that references Students.

1. What should be done if an Enrolled tuple with a non-existent student id is inserted?
2. What should be done if a Students tuple is deleted, e.g. sid = 53666?
   - A. Delete all Enrolled tuples that refer to 53666.
   - B. Disallow deletion 53666.
   - C. Set sid in Enrolled tuples that refer to 53666 to a *default sid*.
   - D. Set sid in Enrolled tuples that refer to it to a special value *null*, denoting `*unknown*’ or `*inapplicable*’.

Foreign Keys – Insertions in the Referencing Table

accnum	balance	type	sin
761	904.33	CHQ	111
856	1011.45	CHQ	333
903	12.05	CHQ	222
1042	10000.00	SAV	333

Inserting {409, 0, CHQ, 555} into *Account* violates the foreign key on *sin* as there is no *sin* of 555 in *Customer*

sin	fname	lname	age	salary
111	Buffy	Summers	23	43000.00
222	Xander	Harris	22	6764.87
333	Rupert	Giles	47	71098.65
444	Dawn	Summers	17	4033.32

Foreign Keys – Insertions in the Referencing Table

accnum	balance	type	sin
761	904.33	CHQ	111
856	1011.45	CHQ	333
903	12.05	CHQ	222
1042	10000.00	SAV	333

Inserting {409, 0, CHQ, 555} into *Account* violates the foreign key on *sin* as there is no *sin* of 555 in *Customer*

The insertion is rejected; before it is processed a *Customer* with a *sin* of 555 must be inserted into the *Customer* table

sin	fname	lname	age	salary
111	Buffy	Summers	23	43000.00
222	Xander	Harris	22	6764.87
333	Rupert	Giles	47	71098.65
444	Dawn	Summers	17	4033.32

Foreign Keys – Updates to the Referencing Table

accnum	balance	type	sin
761	904.33	CHQ	111
856	1011.45	CHQ	333
903	12.05	CHQ	222
1042	10000.00	SAV	333

Changing this record's *sin* to 555 also violates the foreign key, again leading to the transaction being rejected

sin	fname	lname	age	salary
111	Buffy	Summers	23	43000.00
222	Xander	Harris	22	6764.87
333	Rupert	Giles	47	71098.65
444	Dawn	Summers	17	4033.32

Foreign Keys – Deletions in the Referenced Table

accnum	balance	type	sin
761	904.33	CHQ	111
856	1011.45	CHQ	333
903	12.05	CHQ	222
1042	10000.00	SAV	333

Deleting this record will violate the foreign key, because a record with that *sin* exists in the *Account* table

sin	fname	lname	age	salary
111	Buffy	Summers	23	43000.00
222	Xander	Harris	22	6764.87
333	Rupert	Giles	47	71098.65
444	Dawn	Summers	17	4033.32

Foreign Keys – Updates to the Referenced Table

accnum	balance	type	sin
761	904.33	CHQ	111
856	1011.45	CHQ	333
903	12.05	CHQ	222
1042	10000.00	SAV	333

Updating this record so that the *sin* = 666 will violate the foreign key, because a record with the original *sin* exists in the *Account* table

sin	fname	lname	age	salary
111	Buffy	Summers	23	43000.00
222	Xander	Harris	22	6764.87
333	Rupert	Giles	47	71098.65
444	Dawn	Summers	17	4033.32

Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
  - Default is NO ACTION (*delete/update is rejected*)
  - CASCADE (also delete all tuples that refer to 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 )
```

General Constraints

- A DB may require constraints other than primary keys, foreign keys and domain constraints
  - Limiting domain values to subsets of the domain
    - e.g. limit age to positive values less than 150
  - Or other constraints involving multiple attributes
- SQL supports two kinds of general constraint
  - *Table constraints* associated with a single table
  - *Assertions* which may involve several tables and are checked when any of these tables are modified
- These will be covered later in the course

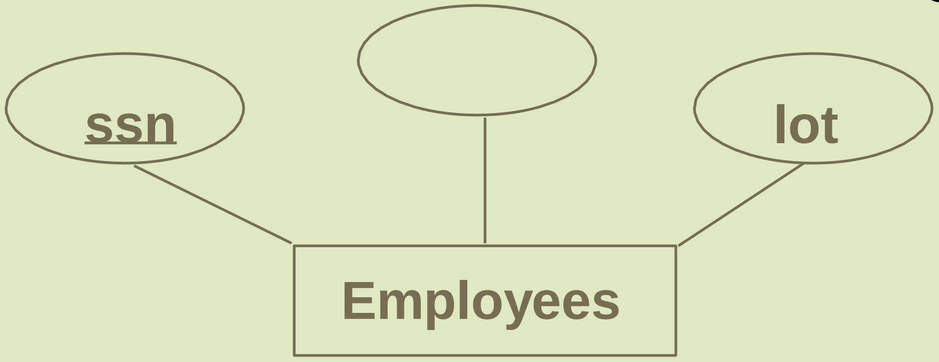
Translate ER Diagrams to SQL

Problem Solving Steps

- Understand the business rules/requirements
- Draw the ER diagram
- Draw the Relational Model
- Write the SQL and create the database

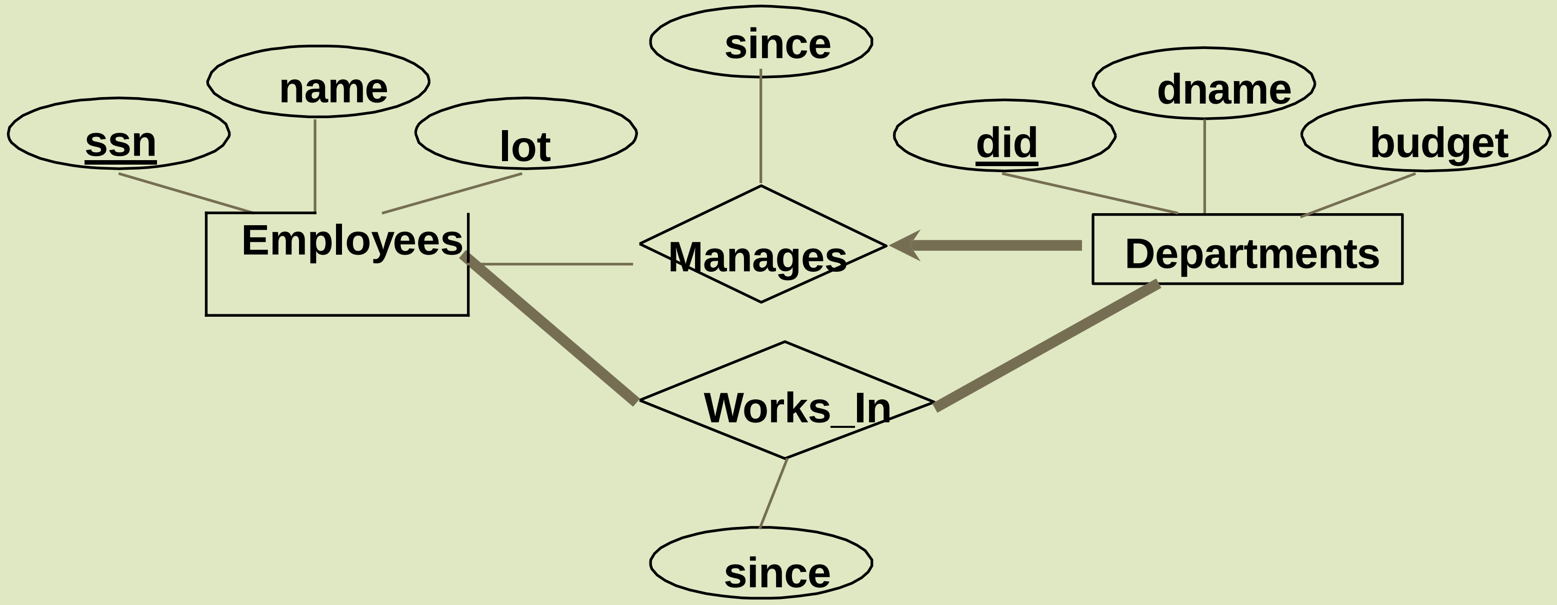
Logical DB Design: ER to Relational

- Entity sets to tables:



```
CREATE TABLE Employees

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

Review: The Works_In Relation



Exercise:

1. Write a create statement for the Departments entity set.
2. Write a create statement for the Works_In relation. Do not consider the participation constraint now.

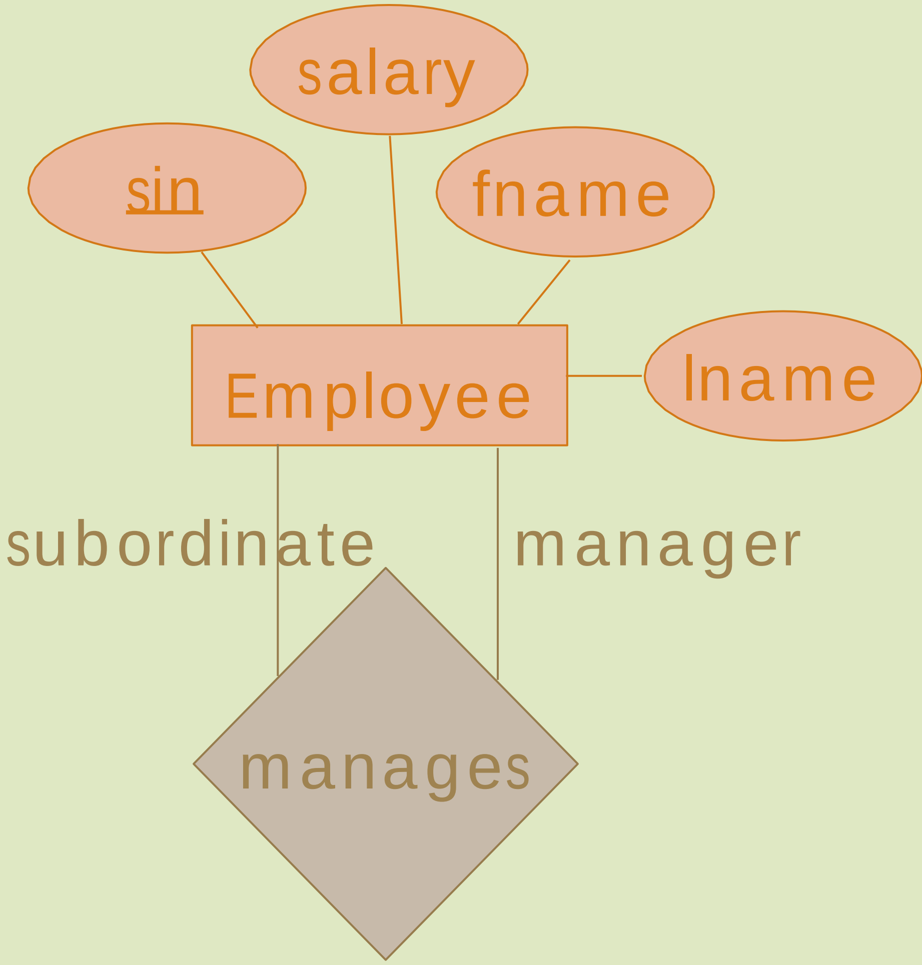
Relationship Sets to Tables

- 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 *key* for the relation. (Superkey?)
  - All descriptive attributes.

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

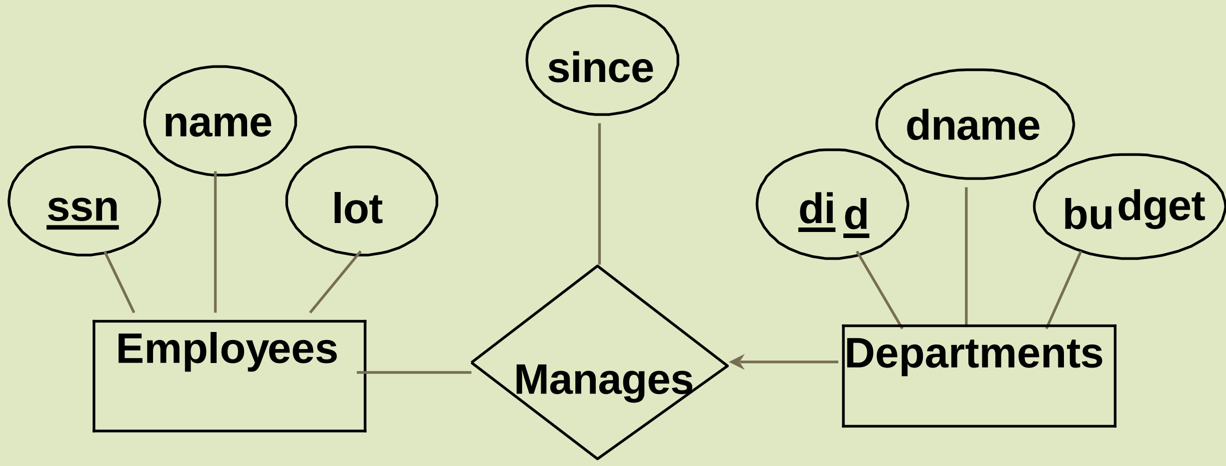
Relationship Set to Table ...

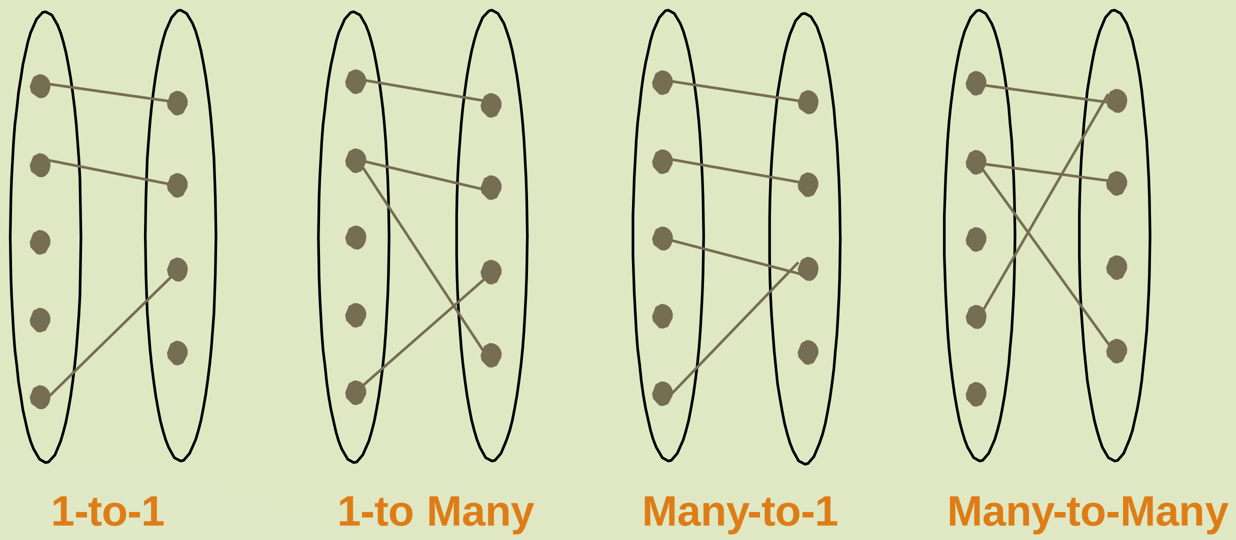
```
CREATE TABLE Manages  (
    manSIN CHAR(11),
    subSIN  CHAR(11),
    FOREIGN KEY (manSIN)
    REFERENCES Employee,
    FOREIGN KEY (subSIN)
    REFERENCES Employee,
    PRIMARY KEY (manSIN,
    subSIN) )
```



Key Constraints

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





Translation to relational model?

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.

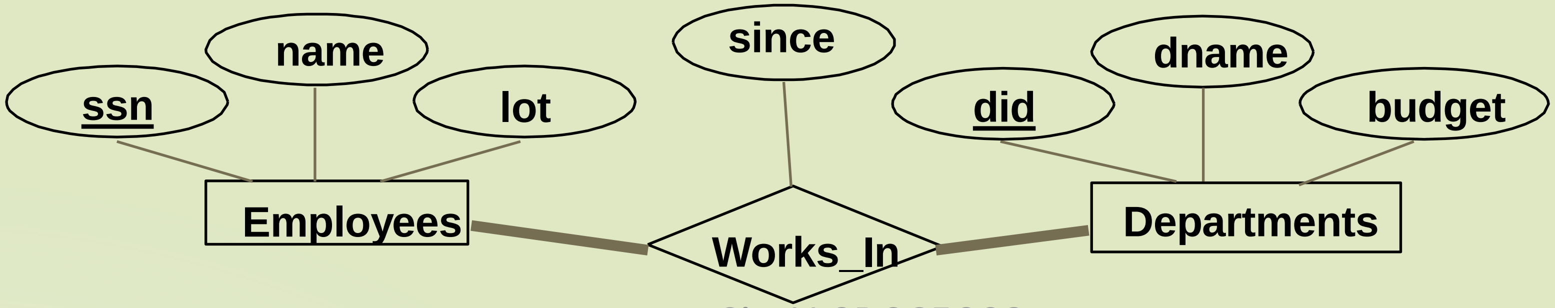
```
CREATE TABLE  Manages(
  ssn  CHAR(11),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY  (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments)
```

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

Participation Constraints

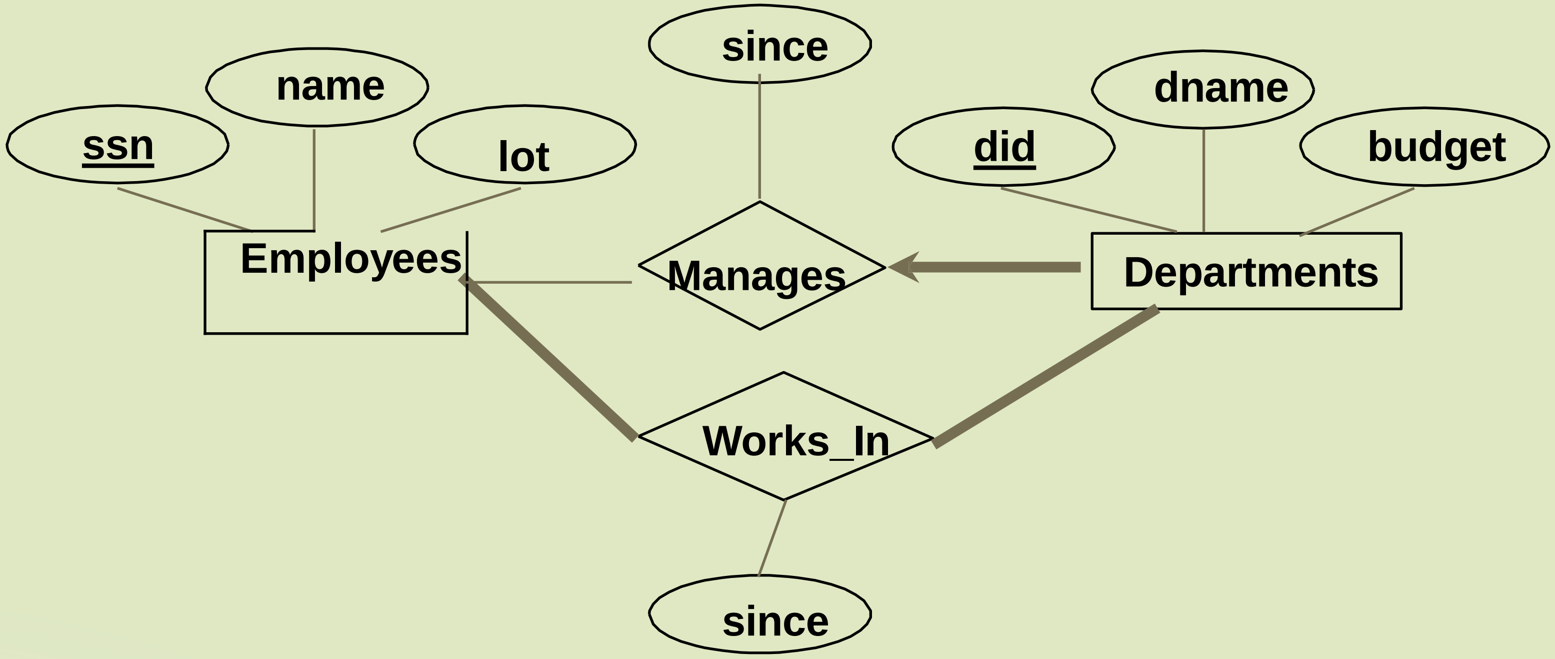
- Every department must have some employee.
- Each employee must work in some department.
- Can we capture these constraints?

```
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)
```



Participation Constraint + Key Constraint

- Every department must have a manager.
- Can we capture this constraint?

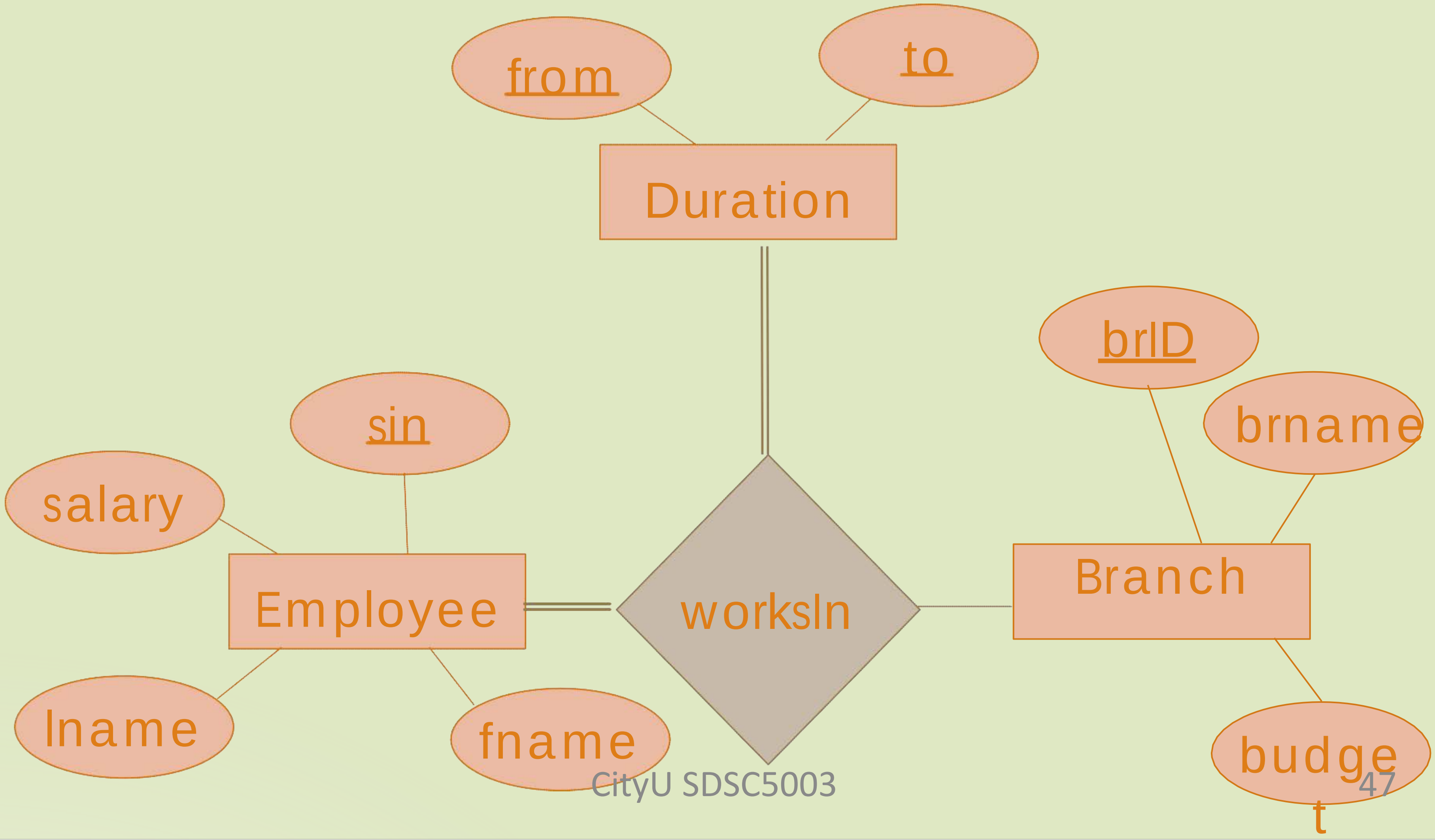


Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship.
- But little else (with what we have so far).

```
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)
```

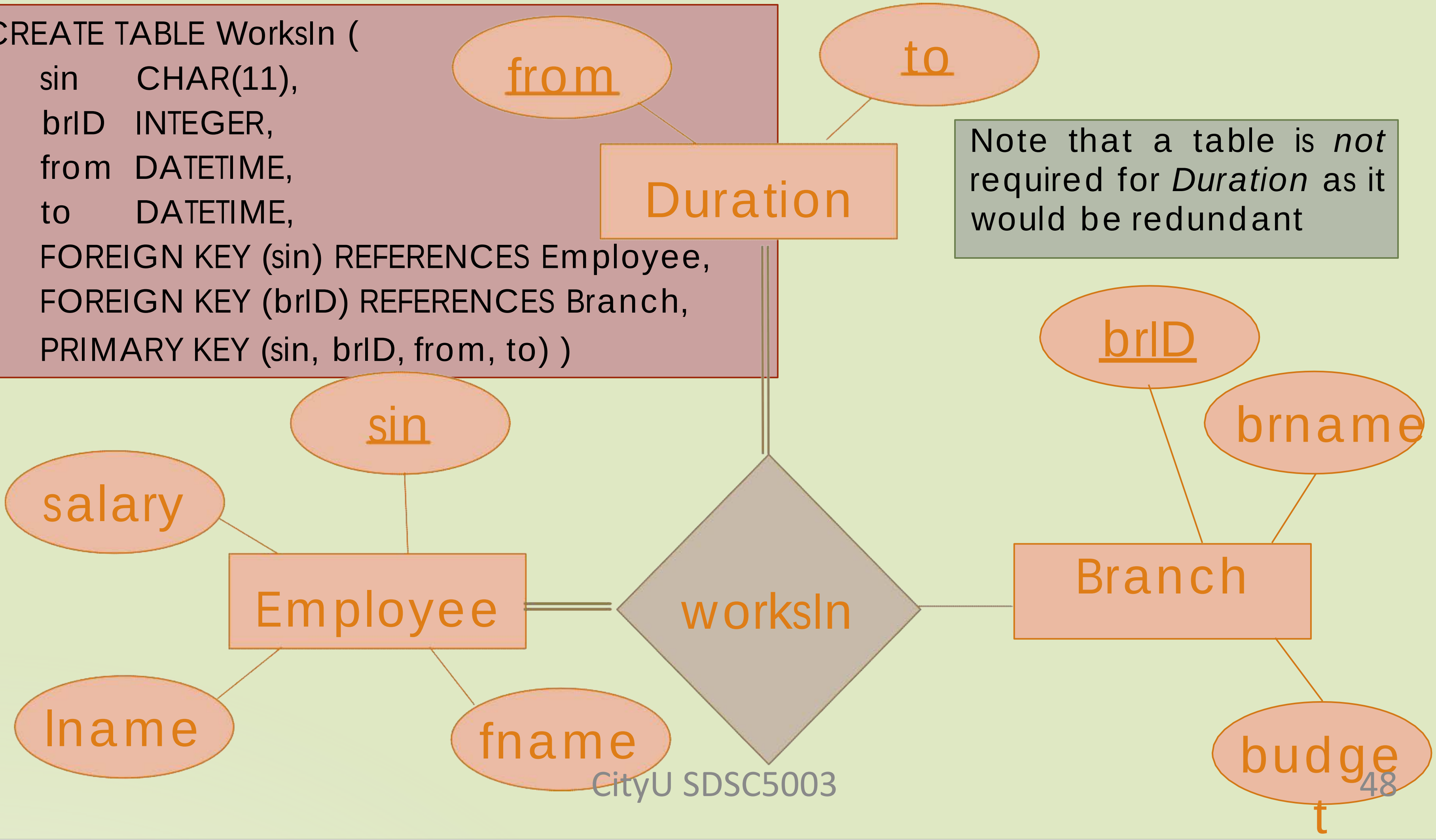
More Relationship Sets ...



More Relationship Sets ...

```
CREATE TABLE WorksIn (
    sin    CHAR(11),
    brID   INTEGER,
    from   DATETIME,
    to     DATETIME,
    FOREIGN KEY (sin) REFERENCES Employee,
    FOREIGN KEY (brID) REFERENCES Branch,
    PRIMARY KEY (sin, brID, from, to) )
```

Note that a table is *not* required for *Duration* as it would be redundant

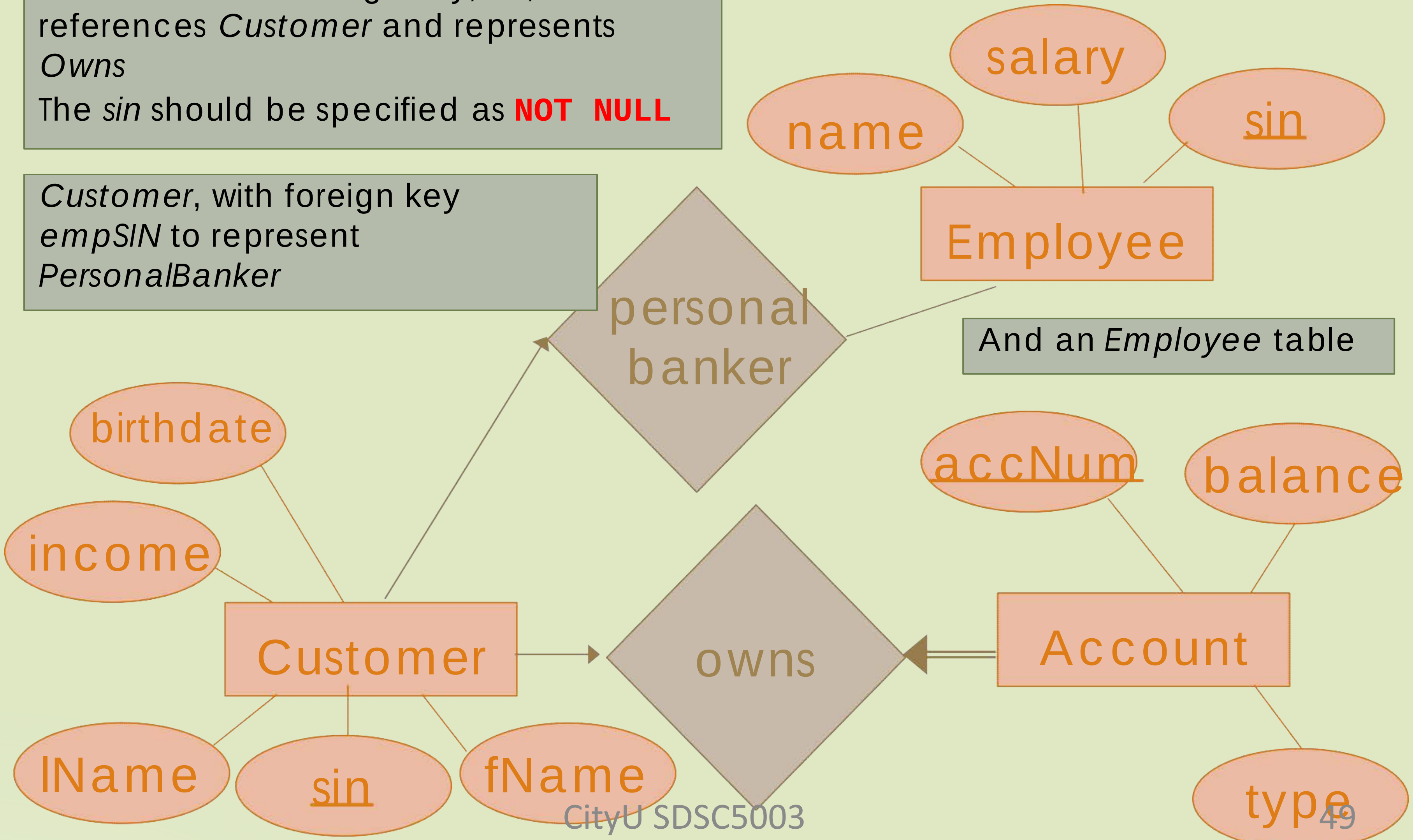


More Relationship Sets ...

Account with a foreign key, *sin*, references *Customer* and represents *Owns*
The *sin* should be specified as **NOT NULL**

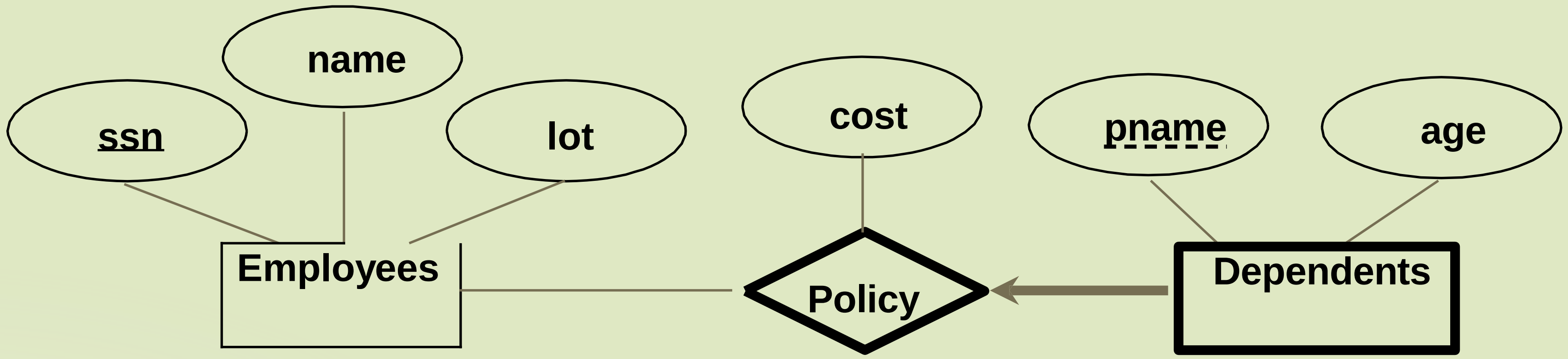
Customer, with foreign key *empSIN* to represent *PersonalBanker*

And an *Employee* table



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.

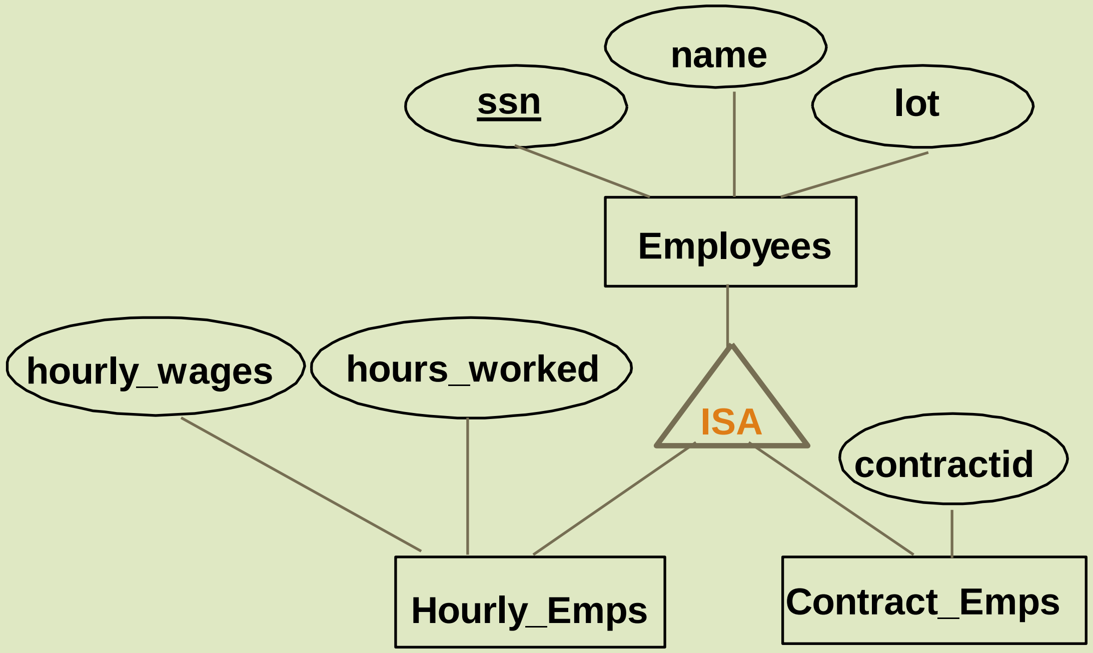


Translating Weak Entity Sets

- 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.
  - What guarantees existence of owner?

```
CREATE TABLE Dep_Policy (
  pname  CHAR(20),
  age  INTEGER,
  cost  REAL,
  owner  CHAR(11),
  PRIMARY KEY  (pname, owner),
  FOREIGN KEY  (owner) REFERENCES Employees(ssn),
     ON DELETE CASCADE)
```

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)*

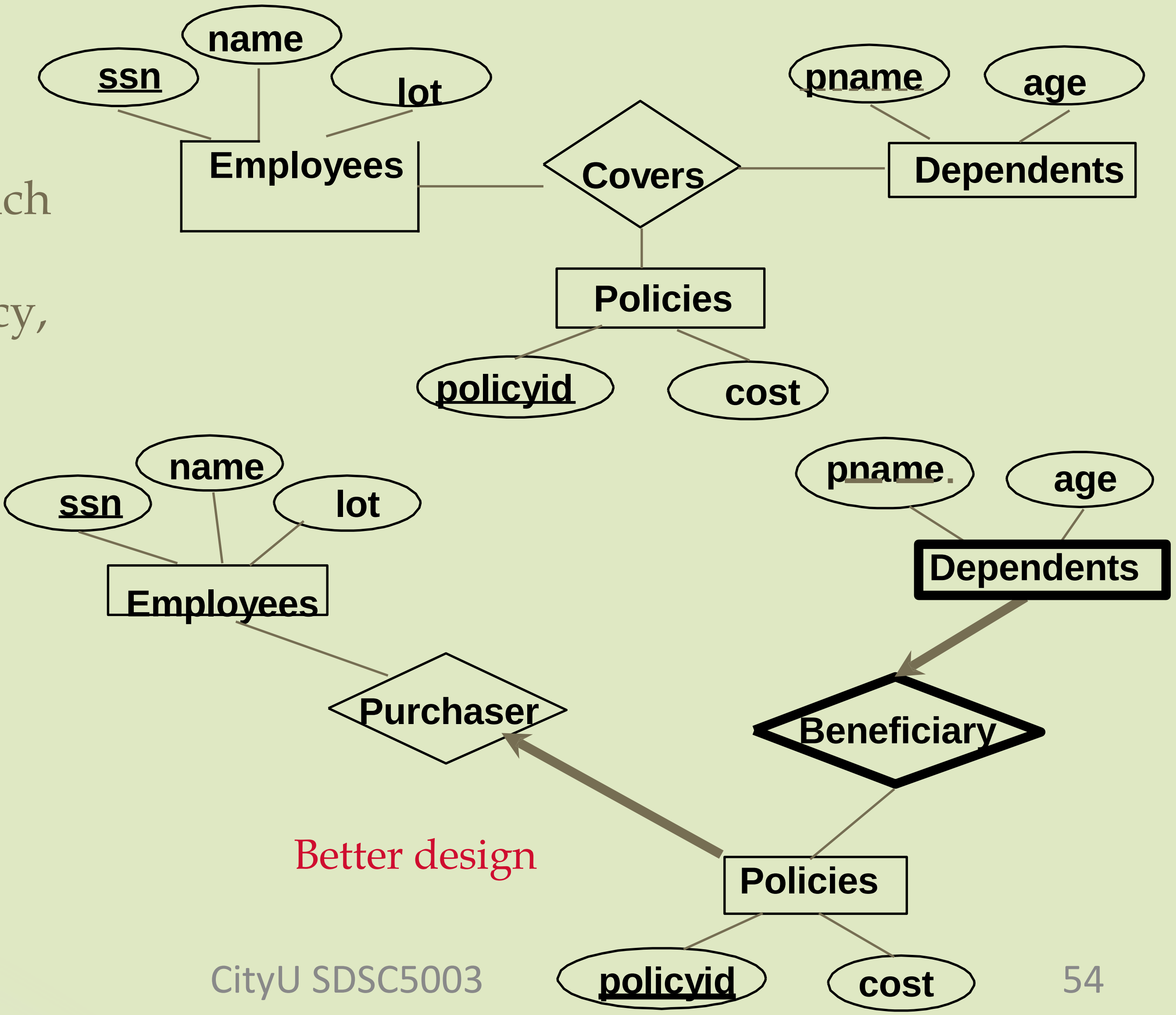
Translating ISA Hierarchies to Relations

- 3 relations: Employees, Hourly_Emps and Contract_Emps.
  - Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages*, *hours_worked*, <u>*ssn*</u>)
- 2 relations: Just Hourly_Emps and Contract_Emps.
  - *Hourly_Emps*: <u>*ssn*</u>, *name*, *lot*, *hourly_wages*, *hours_worked*.
  - Each employee must be in one of these two subclasses.
- 1 relation: Employees.
  - *Emps*: <u>*ssn*</u>, *name*, *lot*, *hourly_wages*, *hours_worked*, *contractid*.
  - Requires null values.

Review: Binary vs. Ternary Relationships

- If each policy is owned by just 1 employee, and each

  the covering policy, first diagram is inaccurate.
- What are the additional constraints in the 2nd diagram?



Better design

Binary vs. Ternary Relationships SQL

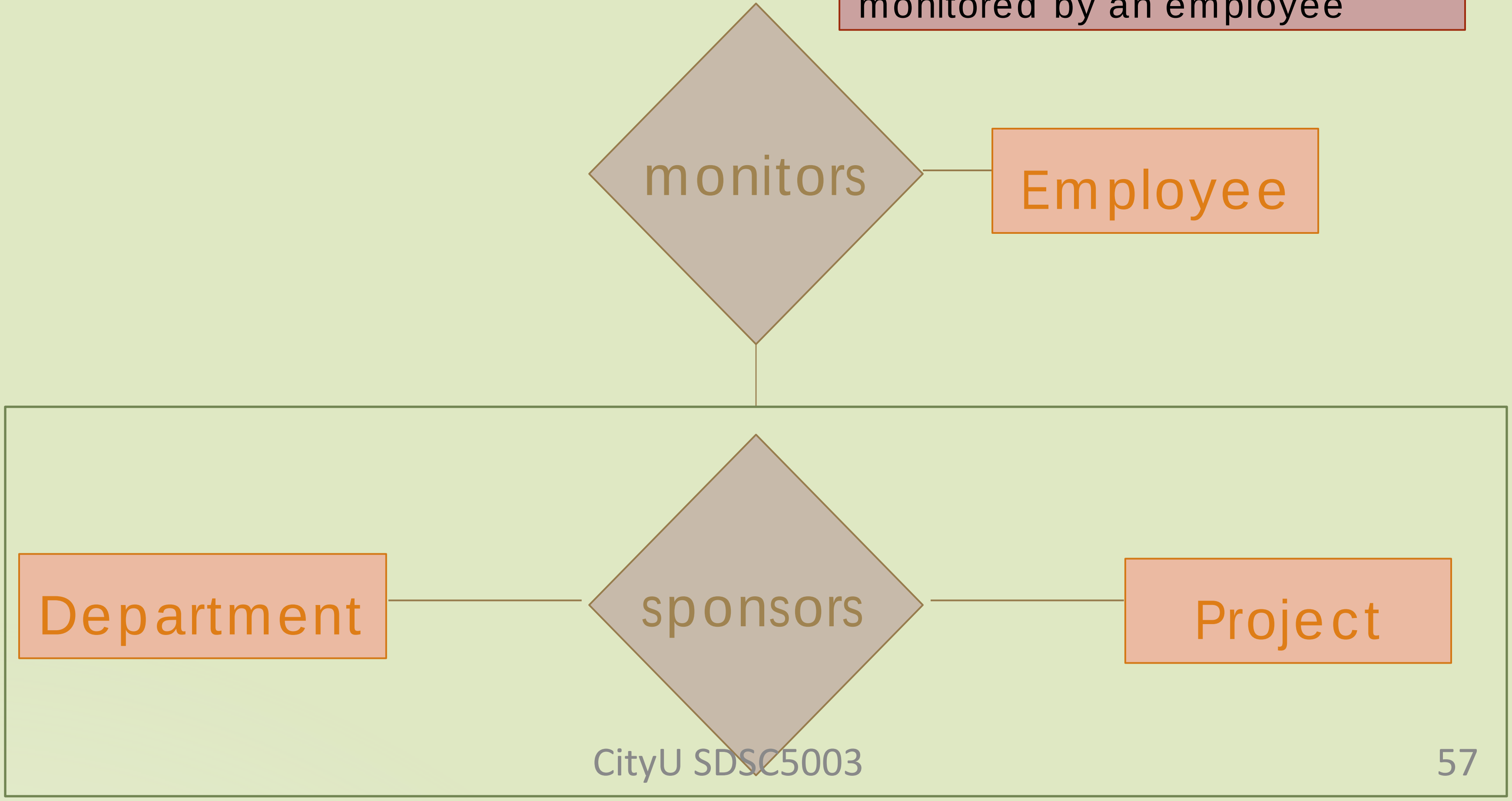
```
CREATE TABLE Policies (
  policyid INTEGER,
  cost REAL,
  ssn CHAR(11) NOT NULL,
  PRIMARY KEY (policyid).
  FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE CASCADE)

CREATE TABLE Dependents (
  pname CHAR(20),
  age INTEGER,
  policyid INTEGER,
  PRIMARY KEY (pname, policyid).
  FOREIGN KEY (policyid) REFERENCES Policies,
    ON DELETE CASCADE)
```

Aggregation

- An aggregate entity is represented by the table defining the relationship set in the aggregation
- The relationship *between* the aggregate entity and the other entity has the following attributes:
  - The primary key of the participating entity set, and
  - The primary key of the relationship set that defines the aggregate entity, and
  - Its own descriptive attributes, if any
- The normal rules for determining primary keys and omitting tables apply

Aggregation Example

In this example, each *department*, *project* pair is monitored by an employee



Aggregation Example

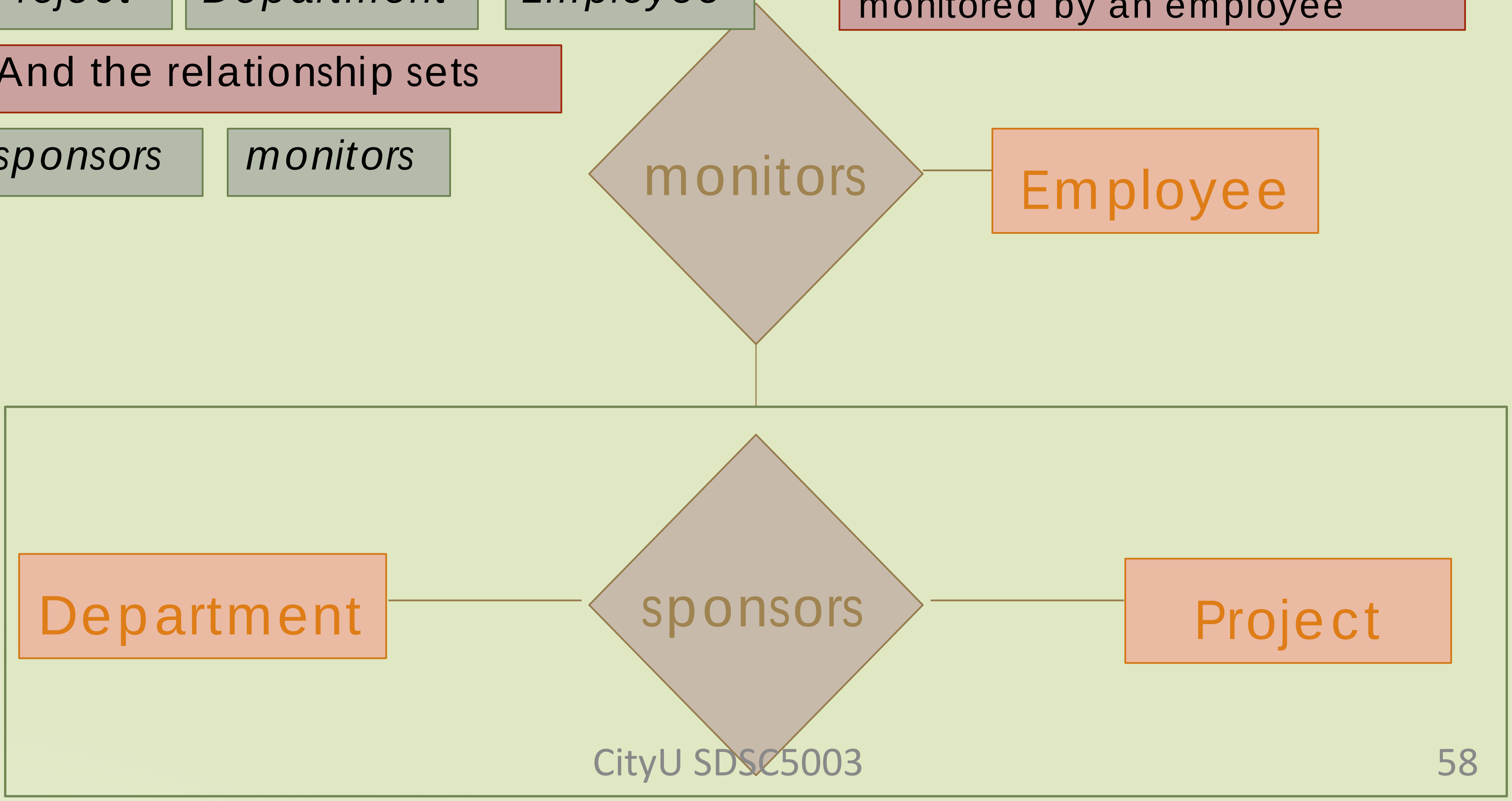
Create tables for the entity sets

Project *Department* *Employee*

And the relationship sets

sponsors *monitors*

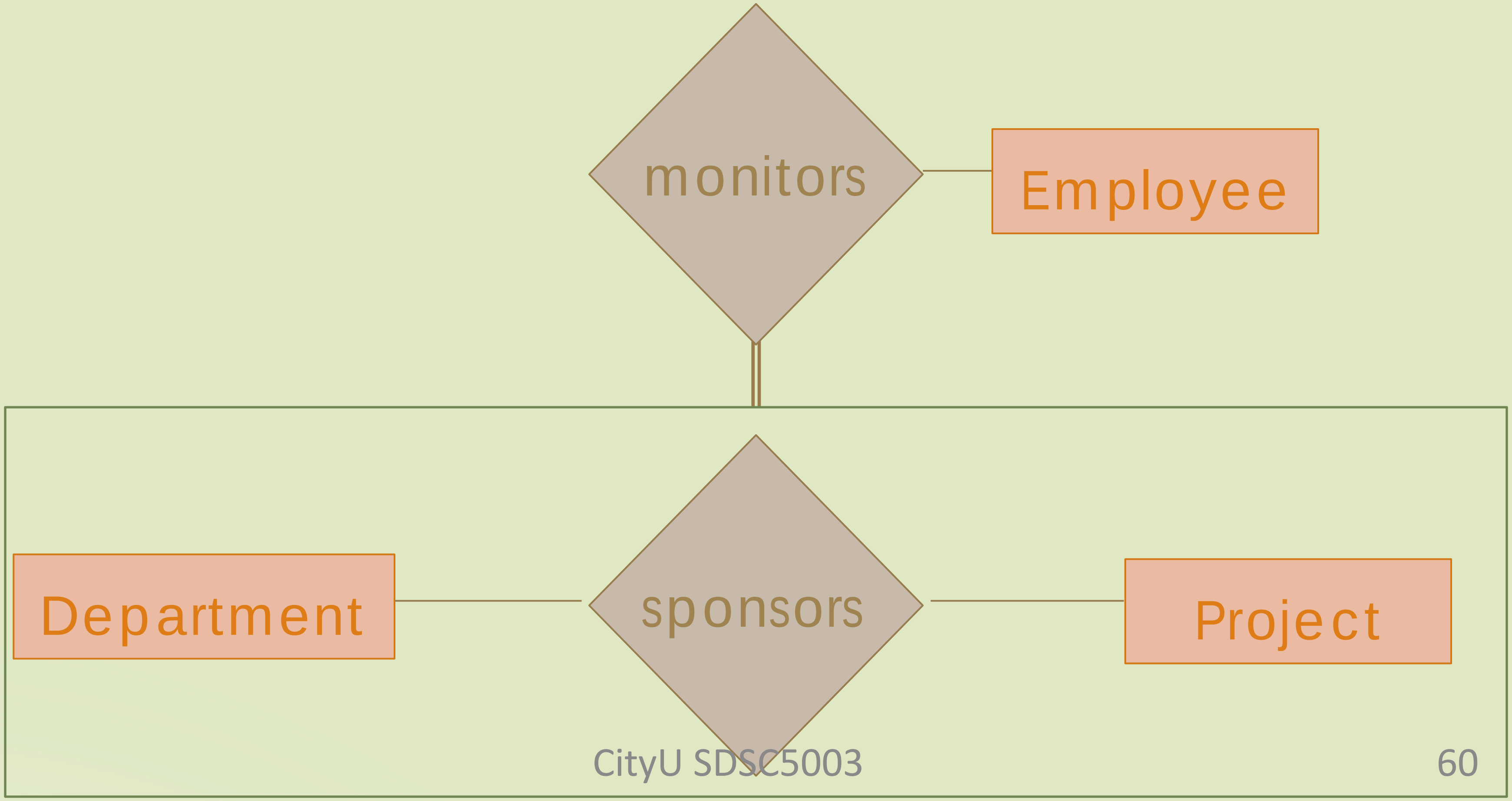
In this example, each *department*, *project* pair is monitored by an employee



Aggregation – Special Case

- There is a case where no table is required for an aggregate entity
  - Even where there are no cardinality constraints
  - If there is *total participation* between the aggregate entity and its relationship, and
  - If the aggregate entity does not have any descriptive attributes
  - Insert the attributes of the aggregate entity into the table representing the relationship with that entity

Aggregation Example



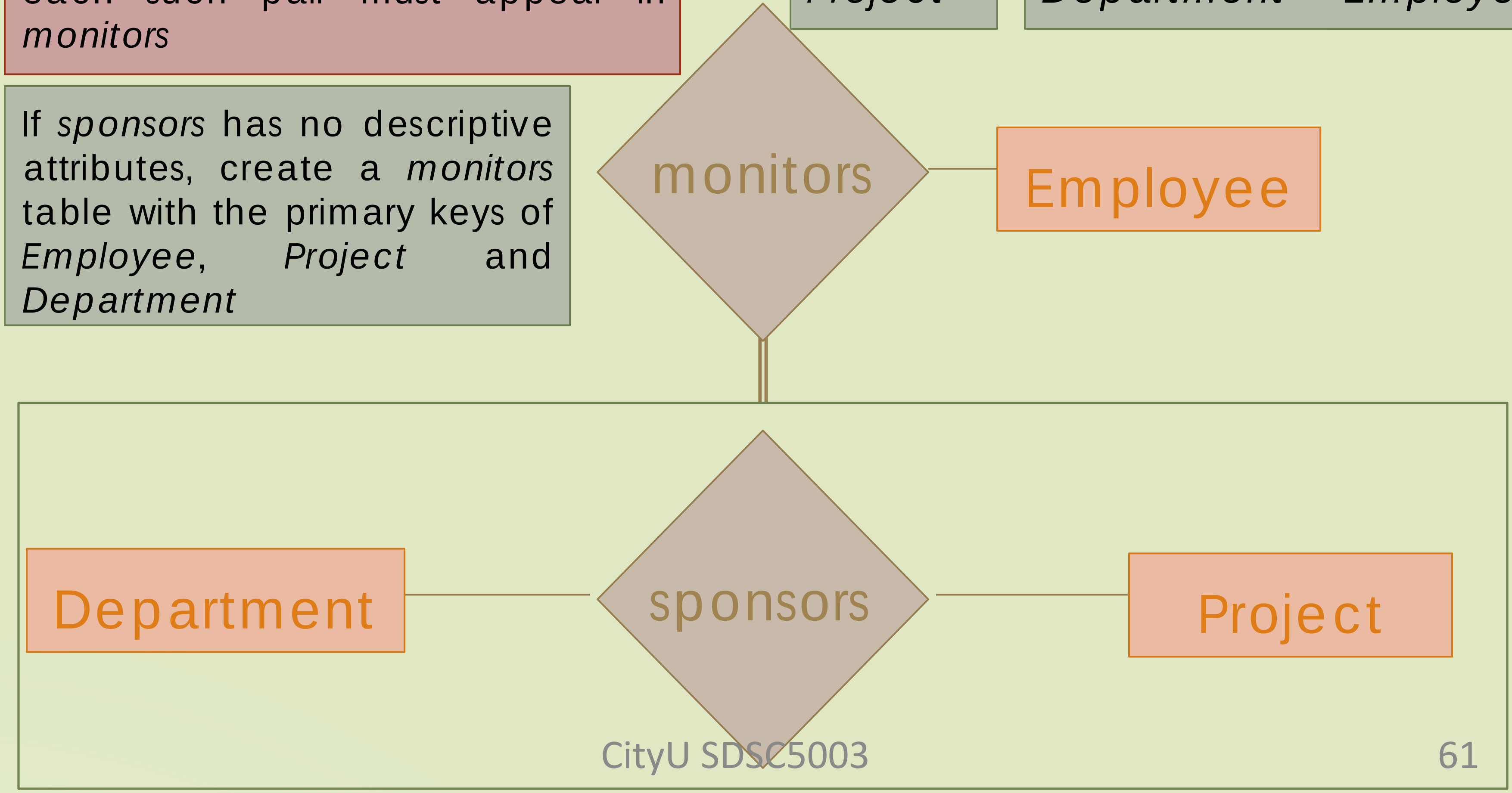
Aggregation Example

A project, department pair must be monitored by an employee, so each such pair must appear in *monitors*

If *sponsors* has no descriptive attributes, create a *monitors* table with the primary keys of *Employee*, *Project* and *Department*

Create tables for the entity sets

Project *Department* *Employee*



Summary: From ER to SQL

- Basic construction: each entity set becomes a table.
- Each relationship becomes a table with primary keys that are also foreign keys referencing the entities involved.
- Key constraints in ER give option of merging entity table with relationship table (e.g. Dept_Mgr).
  - Use not-null to enforce participation.

View

Introduction to Views

- Views are not explicitly stored in a DB but are created as required from a view definition
  - At least conceptually, if not necessarily in practice
- Once defined, views may be referred to in the same way as a tables
- View are useful for
  - Convenience – users can access the data they require without referring to many tables
  - Security – users can only access appropriate data
  - Independence – views can help mask changes in the conceptual schema

Views

- A *view* is just a relation, but we store a *definition*, rather than a set of tuples.

```
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
```

- Views can be dropped using the DROP VIEW command.

Exercise 3.19

Consider the following **schema**.

Emp(<u>eid: integer</u>, ename: string, age: integer, salary: real)

Works(<u>eid: integer, did: integer</u>, pct_time: integer)

Dept(<u>did: integer</u>, budget: real, managerid: integer)

And the **view**

```
CREATE VIEW SeniorEmp(sname, sage, salary)
AS SELECT E.ename, E.age, E.salary
FROM Emp E
WHERE E.age >50
```

Exercise ctd.

How will the system process the query:

```
SELECT S.sname
FROM SeniorEmp S
WHERE S.salary > 100,000
```