INFO20003 Database Systems

Dr Renata Borovica-Gajic

Lecture 04

Relational Model &

Translating ER diagrams

Week 2

Coverage

- Relational Model
- Keys & Integrity Constraints
- Translating ER to Logical and Physical Model

Readings: Chapter 3, Ramakrishnan & Gehrke, Database Systems

Relational Data Model

- **Data Model** allows us to translate real world things into structures that a computer can store
- Many models: <u>Relational</u>, ER, O-O, Network, Hierarchical, etc.

- **Relational Model:**
  - –Rows & Columns (Tuples/records and Attributes/fields)
  - –Keys & Foreign Keys to link Relations

Enrolled

sid	cid	grade
53666	Carnatic101	5
53666	Reggae203	5.5
53650	Topology112	6
53666	History105	5

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	5.4
53688	Smith	smith@eecs	18	4.2
53650	Smith	smith@math	19	4.8

Relational Database: Definitions

- ***Relational database***: a set of *relations*.
- ***Relation***: made up of 2 parts:
  - ***Schema*** : specifies name of relation, plus name and type of each column (attribute).

    Example: Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)
  - ***Instance*** : a ***table*, **with rows and columns.

    #rows = *cardinality*

    #fields = *degree (or arity)*
- You can think of a relation as a *set* of *rows* or *tuples*.
  - all rows are *distinct*, no order among rows

Example Instance of Students Relation

Students

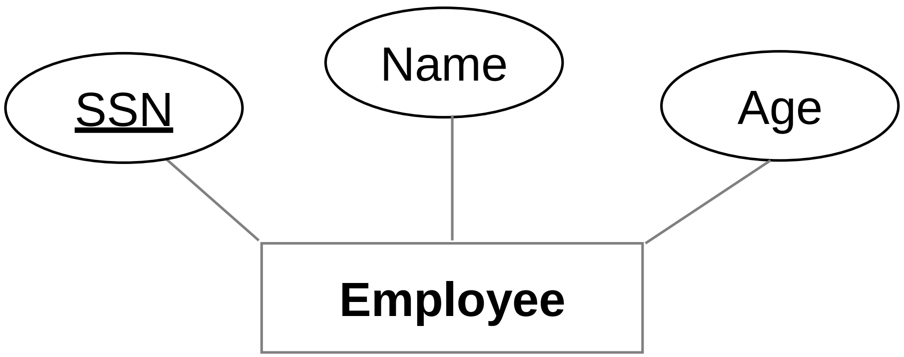
sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Cardinality = 3, degree (arity) = 5, all rows distinct

Logical Design: ER to Relational Model

In logical design **entity** set becomes a **relation.** Attributes become attributes of the relation.

Conceptual Design:



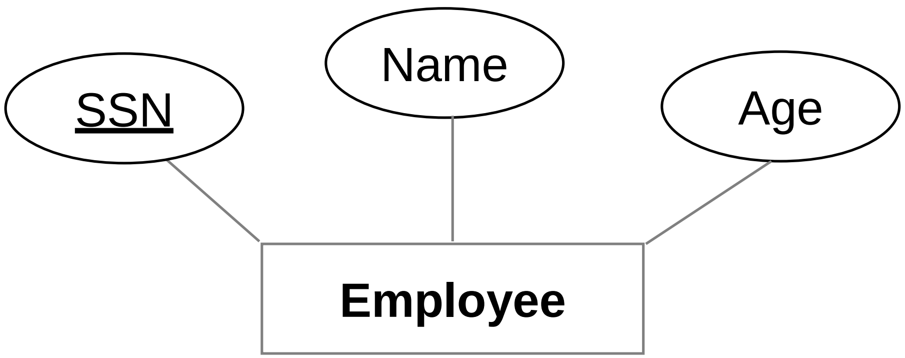
Logical Design:

Employee (ssn,
name,
age)

ER to Logical to Physical

In physical design we choose data types

1. Conceptual Design:



2. Logical Design:

Employee (ssn, name, age)

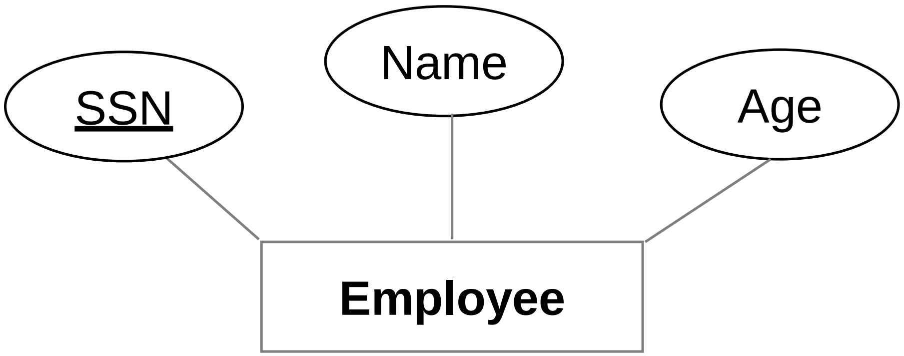
3. Physical Design:
Employee
(ssn CHAR(11),
name VARCHAR(20),
age INTEGER)

The Entire Cycle

1. Conceptual Design
2. Logical Design
3. Physical Design
4. Implementation
5. Create Instance

The Entire Cycle

1. Conceptual Design:



2. Logical Design:

Employee (<u>ssn</u>,
name,
age)

3. Physical Design:

Employee
(<u>ssn</u> CHAR(11),
name VARCHAR(20),
age INTEGER)

4. Implementation:

```
CREATE TABLE Employee
    (ssn CHAR(11),
     name VARCHAR(20),
     age INTEGER,
     PRIMARY KEY (ssn))
```

5. Instance:

EMPLOYEE

<u>ssn</u>	name	age
0983763423	John	30
9384392483	Jane	30
3743923483	Jill	20

Creating Relations in SQL

Example: Creating the Students relation.

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

The type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

Relational Model

- Relational Model & SQL overview
- **Keys & Integrity Constraints**
- Translating ER to Logical and Physical Model

Readings: Chapter 3, Ramakrishnan & Gehrke, Database Systems

Keys

- Keys are a way to associate tuples in different relations
- Keys are one form of **integrity constraint (IC)**
- **Example:** *Only students can be enrolled in subjects.*

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 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 it is a superkey and no subset of the fields is a superkey (minimal subset)
- Out of all keys *one* is chosen to be the ***primary key*** of the relation**.** Other keys are called ***candidate*** keys
- Each relation has a primary key

- **Your turn:**
  1. *Is sid* a key for Students?
  2. What about *name*?
  3. Is the set {*sid, gpa*} a superkey? Is the set {*sid, gpa*} a key?
  4. Find a primary key from this set {sid, login}

Primary and Candidate Keys in SQL

- There are possibly many *<u>candidate keys</u>* (specified using UNIQUE), one of which is chosen as the *primary key*. Keys must be chosen carefully.

Example:

For a given student and course, there is a single grade.

```
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 & Referential Integrity

- ***Foreign key*** : A set of fields in one relation that is used to ‘refer’ to a tuple in another relation. Foreign key must correspond to the primary key of the other relation.

- If all foreign key constraints are enforced in a DBMS, we say ***referential integrity*** is achieved.

Foreign Keys in SQL

Example: *Only students listed in the Students relation should be allowed to enroll in courses.*

- *sid* is a foreign key referring to Students

```
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	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

Enforcing Referential Integrity

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a Students tuple is deleted?
  - Also delete all Enrolled tuples that refer to it?
  - Disallow deletion of a Students tuple that is referred to?
  - Set sid in Enrolled tuples that refer to it to a *default sid*?
  - (In SQL, also: Set sid in Enrolled tuples that refer to it to a special value *null,* denoting `*unknown*' or `*inapplicable*'.)
- Note: Similar issues arise if primary key of Students tuple is updated.

Integrity Constraints (ICs)

- ***IC***: condition that must be true for *any* instance of the database; e.g., *domain constraints.*
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.

- A ***legal*** instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.

Relational Model

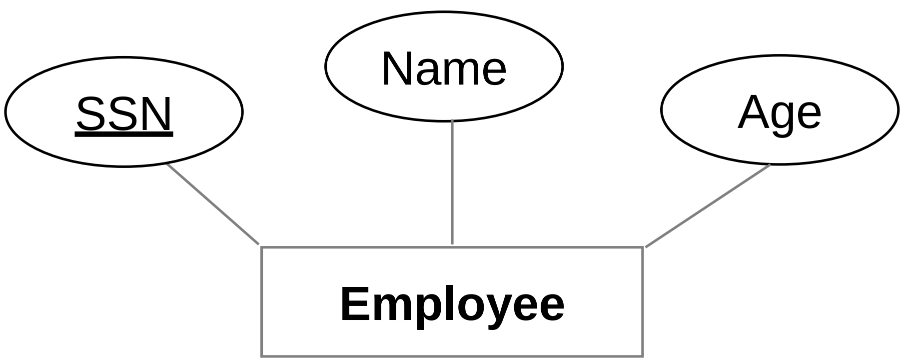
- Relational Model & SQL overview
- Keys & Integrity Constraints
- **Translating ER to Logical and Physical Model**

Readings: Chapter 3, Ramakrishnan & Gehrke, Database Systems

Logical Design: Recap

In logical design **entity** set becomes a **relation.** Attributes become attributes of the relation.

Conceptual Design:



Logical Design:

Employee (<u>ssn</u>,
name,
age)

THE UNIVERSITY OF MELBOURNE

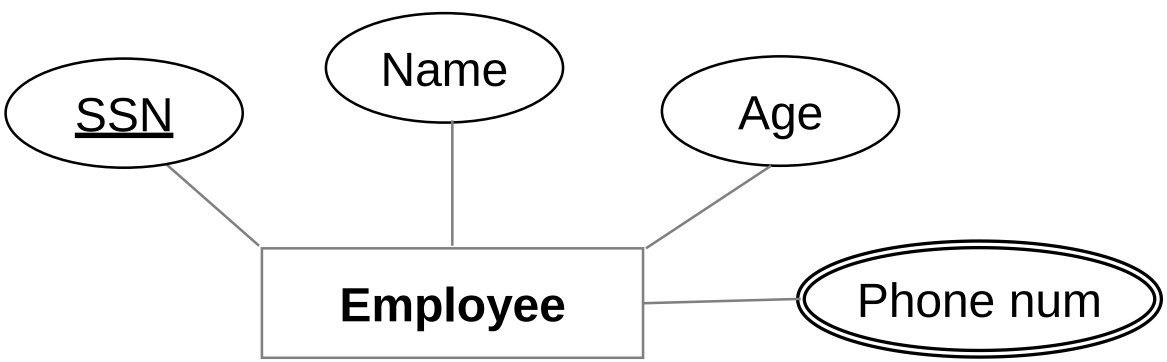
Multi-valued attributes in logical design

- Multi-valued attributes need to be unpacked (flattened) when converting to logical design. **There is an alternative of creating a lookup table discussed in the next lecture.*

Example:

For employees we need to capture their home phone number and work phone number.

Conceptual Design:



Logical Design:

Employee (<u>ssn</u>,
name,
age,
home_num,
work_num)

Multi-valued attribute

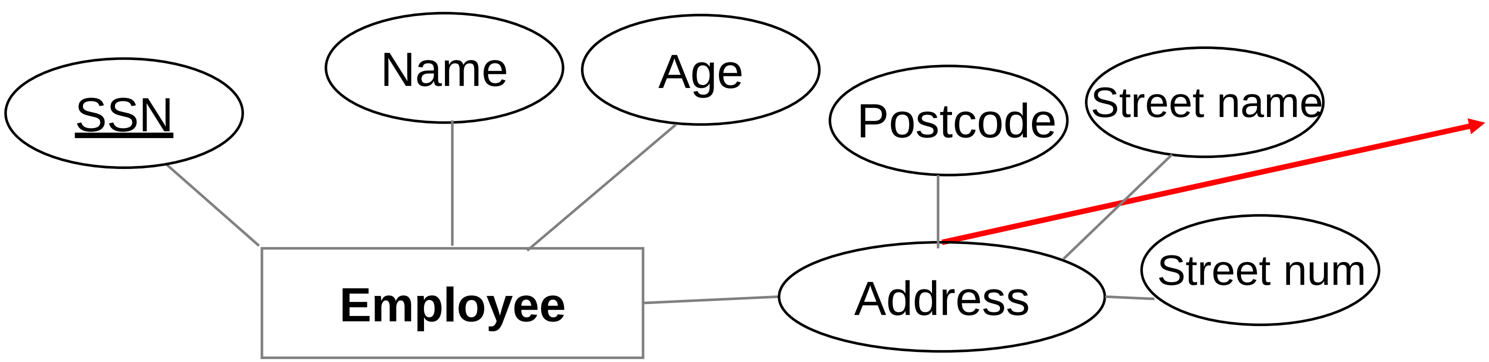
Composite attributes in logical design

- Composite attributes need to be unpacked (flattened) when converting to logical design.

Example:

For employees we need to capture an address consisting of a postcode, street name and number.

Conceptual Design:



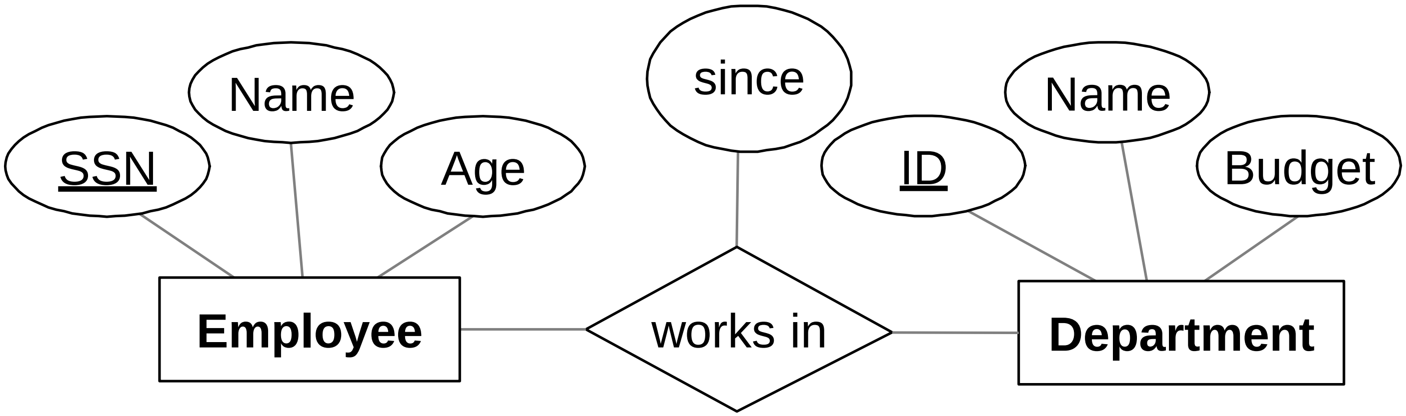
Composite attribute

Logical Design:

Employee (ssn,
name,
age,
postcode,
street_name,
street_num)

ER to Logical Design: Many to Many

Conceptual Design:



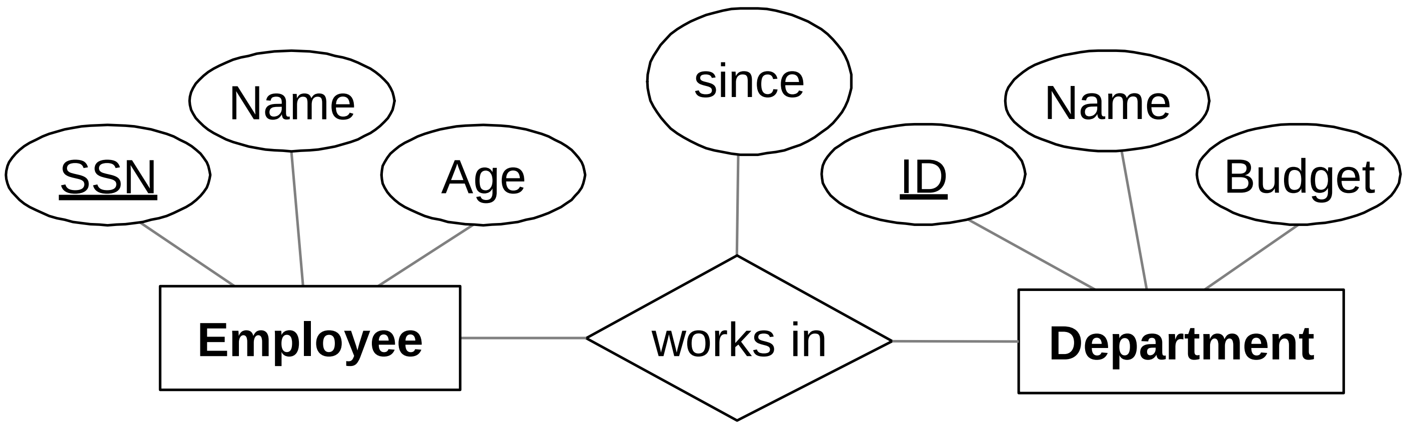
Logical Design:

In translating a **many-to-many** relationship set to a relation, attributes of a *new* relation must include:

1. Keys for each participating entity set (as foreign keys). This set of attributes forms a ***superkey*** of the relation.
2. All descriptive attributes.

ER to Logical Design: Many to Many

Conceptual Design:



Logical Design:

Employee (ssn,
name
age)

Department (did,
dname,
budget)

Works_In (**ssn**,
did,
since)

Keys from connecting entities become PFK

This is called an associative entity

Note: Underline = PK,
italic and underline = FK,
underline and bold = PFK

Logical to Physical Design

Logical Design:

Employee (ssn, name, age)

Department (did, dname, budget)

Works_In (**ssn**, **did**, since)

Note: Underline = PK,
italic and underline = FK,
underline and bold = PFK

Physical Design:

Employee
(ssn CHAR(11),
name VARCHAR(20),
age INTEGER)

Department
(did INTEGER,
dname VARCHAR(20),
budget FLOAT)

Works_In(
ssn CHAR(11),
did INTEGER,
since DATE)

Implementation (Create table)

Logical Design:

Employee (<u>ssn</u>, name, age)

Department (<u>did</u>, dname, budget)

Works_In (<u>**ssn**</u>, <u>**did**</u>, since)

Note: Underline = PK,
italic and underline = FK,
underline and bold = PFK

Implementation:

```
CREATE TABLE Employee
        (ssn CHAR(11),
         name VARCHAR(20),
         age INTEGER,
         PRIMARY KEY  (ssn))
```

```
CREATE TABLE Department
        (did INTEGER,
         dname VARCHAR(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 Employee,
     FOREIGN KEY (did) REFERENCES Department)
```

Example Instances

Employee

ssn	name	age
0983763423	John	30
9384392483	Jane	30
3743923483	Jill	20

Department

did	dname	budget
101	Sales	10K
105	Purchasing	20K
108	Databases	1000K

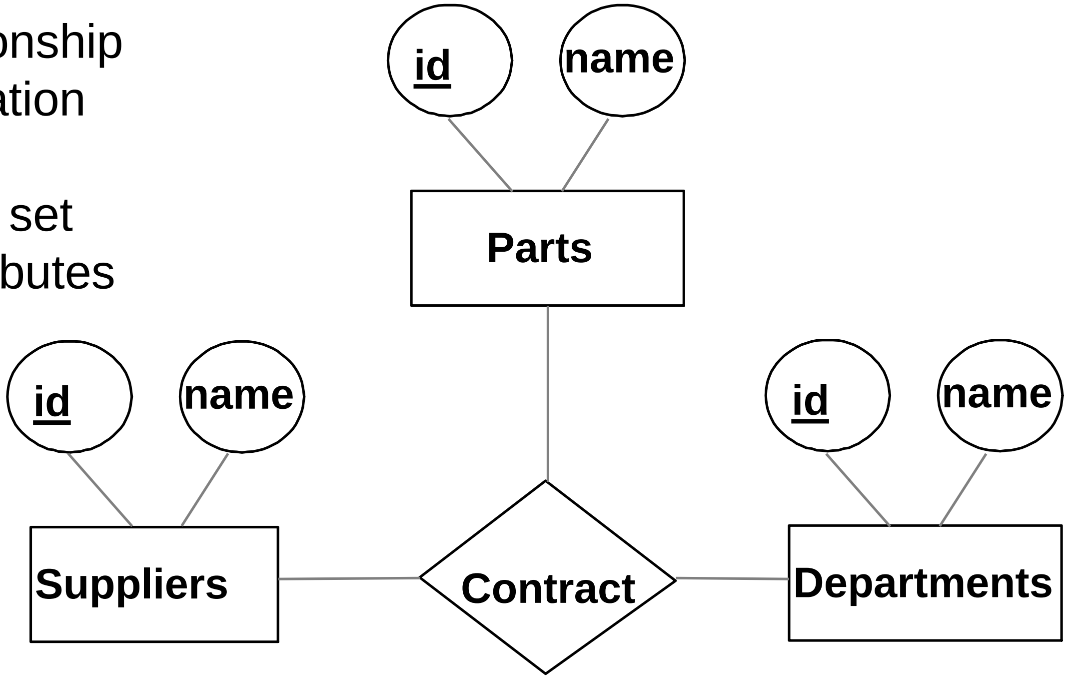
Works_In

ssn	did	since
0983763423	101	1 Jan 2003
0983763423	108	2 Jan 2003
9384392483	108	1 Jun 2002

ER to Logical Design: Ternary relationship

In translating a many-to-many relationship set to a relation, attributes of the relation must include:

- Keys for each participating entity set (as foreign keys). This set of attributes forms a *superkey* for the relation.
- All descriptive attributes.

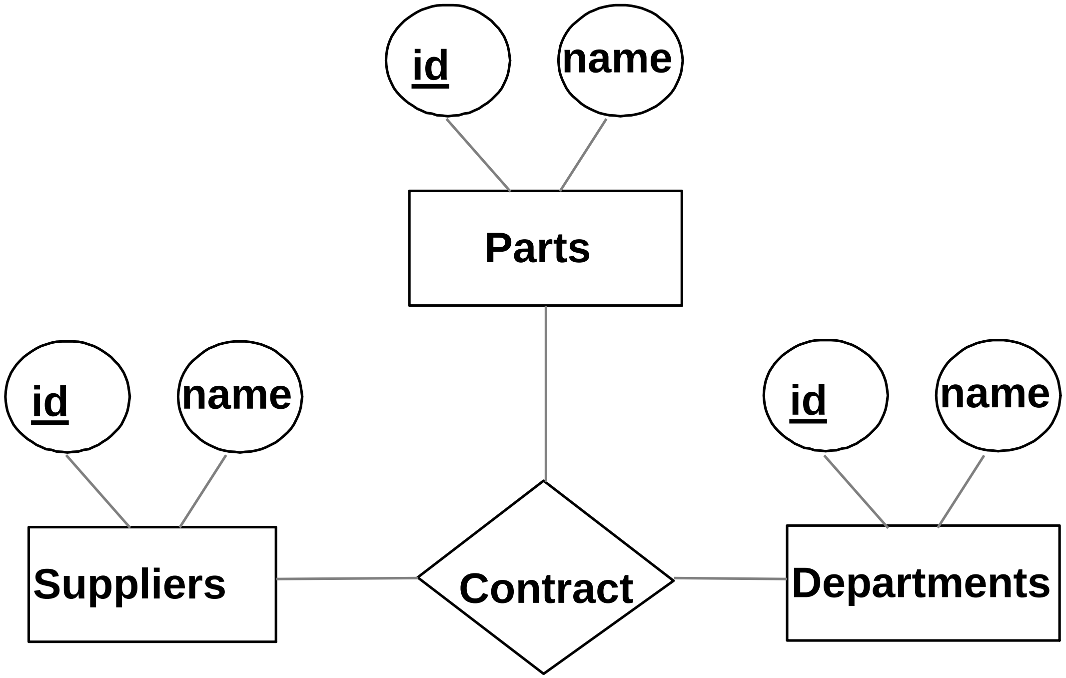


Logical Design:

Contracts (
supplier_id,
part_id,
department_id)

Note: Underline = PK, italic and underline = FK, underline and bold = PFK

ER to Logical to Implementation: Ternary relationship



Logical Design:

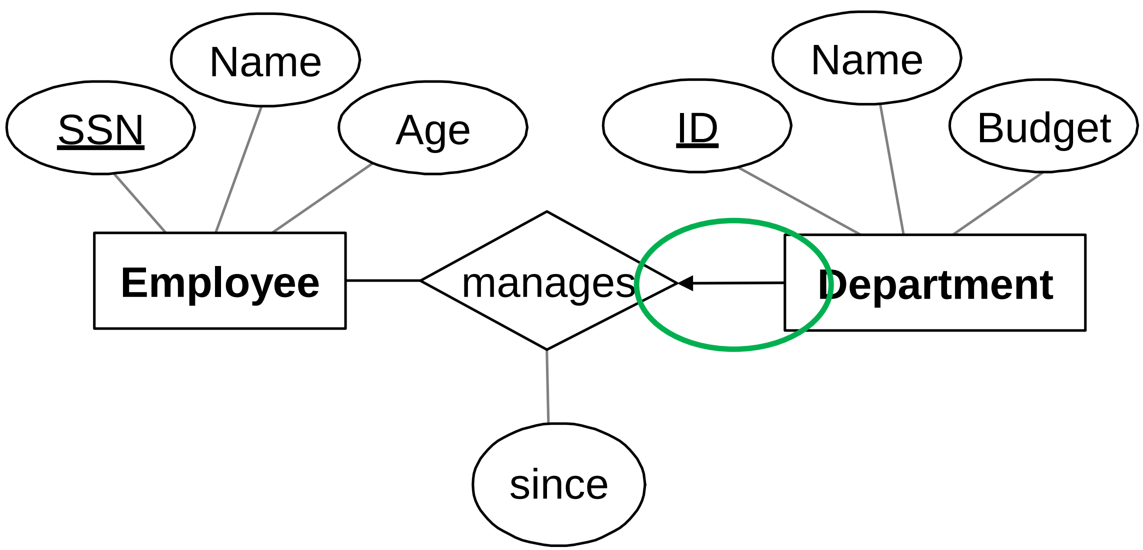
Contracts (
supplier_id,
part_id,
department_id)

Implementation:

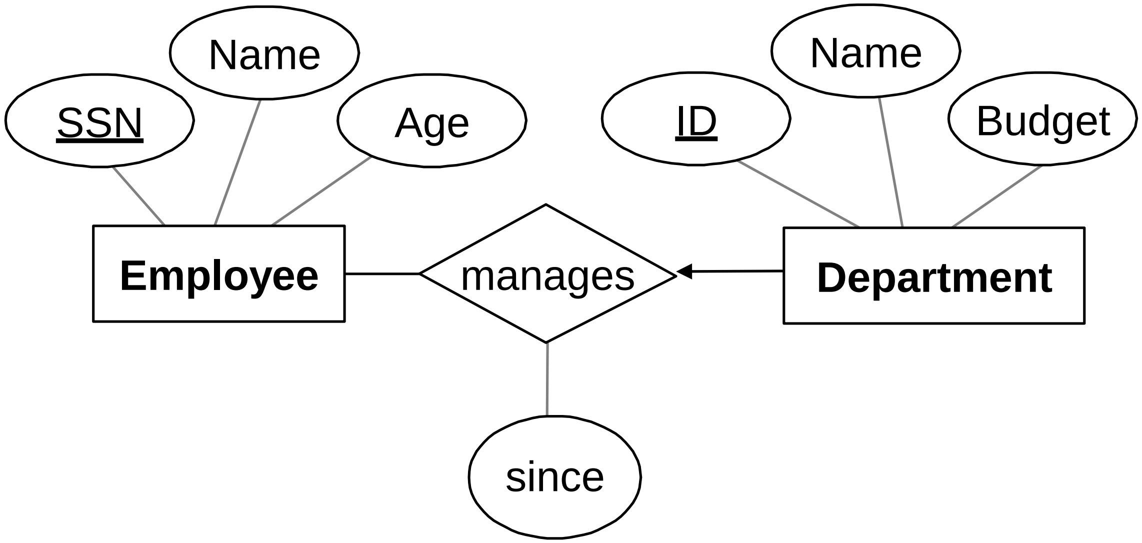
```
CREATE TABLE Contracts (
    supplier_id  INTEGER,
    part_id  INTEGER,
    department_id  INTEGER,
    PRIMARY KEY (supplier_id, part_id, department_id),
    FOREIGN KEY (supplier_id) REFERENCES Suppliers,
    FOREIGN KEY (part_id) REFERENCES Parts,
    FOREIGN KEY (department_id) REFERENCES Departments)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


Review: Key Constraints in ER

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



Logical design: Key Constraints



Logical Design:

Employee (<u>ssn</u>, name, age)

Department (<u>did</u>, dname, budget)

Manages (**<u>ssn</u>, <u>did</u>**, since)

vs.

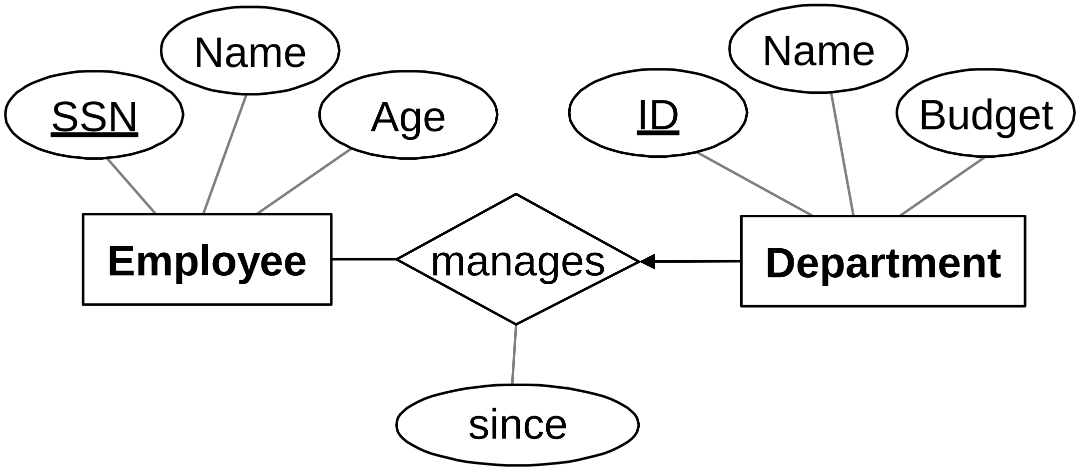
Employee (<u>ssn</u>, name, age)

Department (<u>did</u>, dname, budget, *<u>ssn</u>*, since)

Note: Underline = PK, italic and underline = FK, underline and bold = PFK

Key Constraints in SQL

Implementation:




```
CREATE TABLE Employee
    (ssn CHAR(11),
     name VARCHAR(20),
     age INTEGER,
     PRIMARY KEY  (ssn))
```

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

vs.

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

Which one is better?

Logical Design: Key Constraints Rule

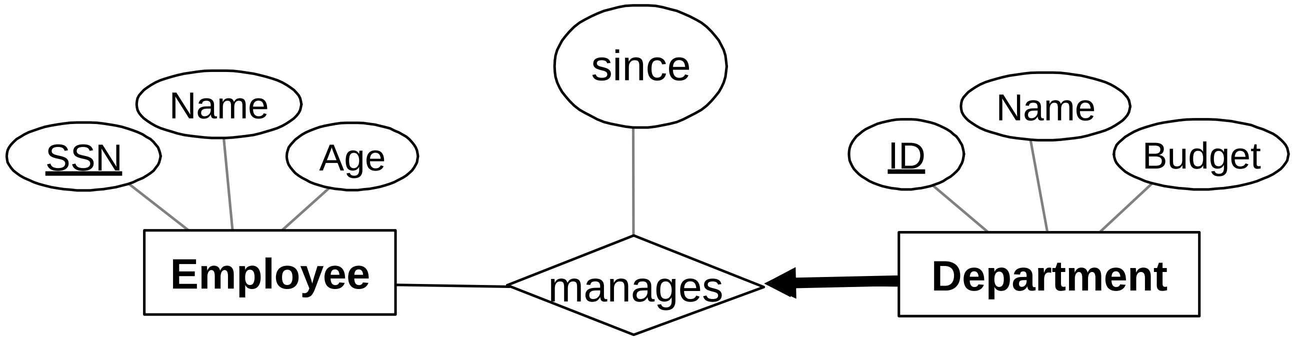
- RULE: *Primary key from the many side becomes a foreign key on the one side*
- This is the way to ensure that the key constraint holds

```
CREATE TABLE Department
        (did INTEGER,
         dname CHAR(20),
         budget FLOAT,
         ssn CHAR(11),
         since DATE,
         PRIMARY KEY  (did)
         FOREIGN KEY (ssn)
          REFERENCES Employee)
```

Each department will have a *single* manager

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*.



Participation Constraints in SQL

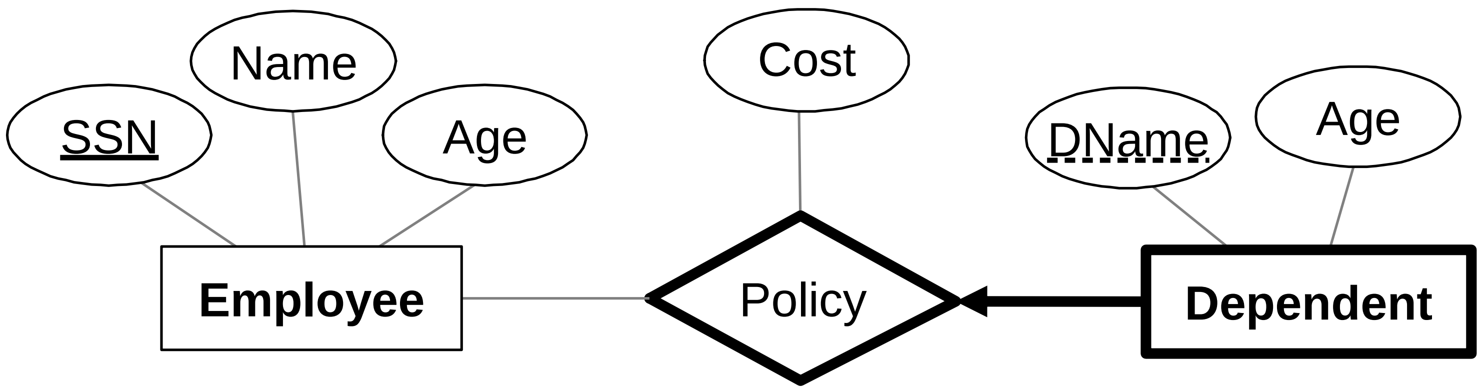
- We specify total participation with key words NOT NULL
  - **NOT NULL = this field cannot be empty**

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

NOTE: Every time we create a table or draw a physical design we should specify whether attributes are NULL or NOT NULL. We haven't done it in each slide of this lecture due to clarity and lack of space – but don't forget this in your design/implementation!

Review: Weak Entities

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



Translating Weak Entities

- 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.

Logical Design:

Dependent (<u>dname</u>, age, cost, **<u>ssn</u>**)

Note: Underline = PK, italic and underline = FK, underline and bold = PFK

Implementation:

```
CREATE TABLE  Dependent (
  dname  CHAR(20) NOT NULL,
  age  INTEGER NULL,
  cost  DECIMAL(7,2) NOT NULL,
  ssn  CHAR(11) NOT NULL,
  PRIMARY KEY  (dname, ssn),
  FOREIGN KEY  (ssn) REFERENCES Employees
    ON DELETE CASCADE)
```

Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used.
- Integrity constraints can be specified based on application semantics. DBMS checks for violations.
  - Two important ICs: primary and foreign keys
  - In addition, we *always* have domain constraints.
- Rules to translate ER to logical design (relational model)

What's examinable

- Translate conceptual (ER) into logical & physical design
- Understand integrity constraints
- Use DDL of SQL to create tables with constraints

Next Lecture

- ER Modelling Example with MySQL Workbench
  - You will need this for workshops/labs (and assessment)