# ER Model, Relational Model
# Mapping ER to Relational


COMP 1531

Aarthi Natarajan

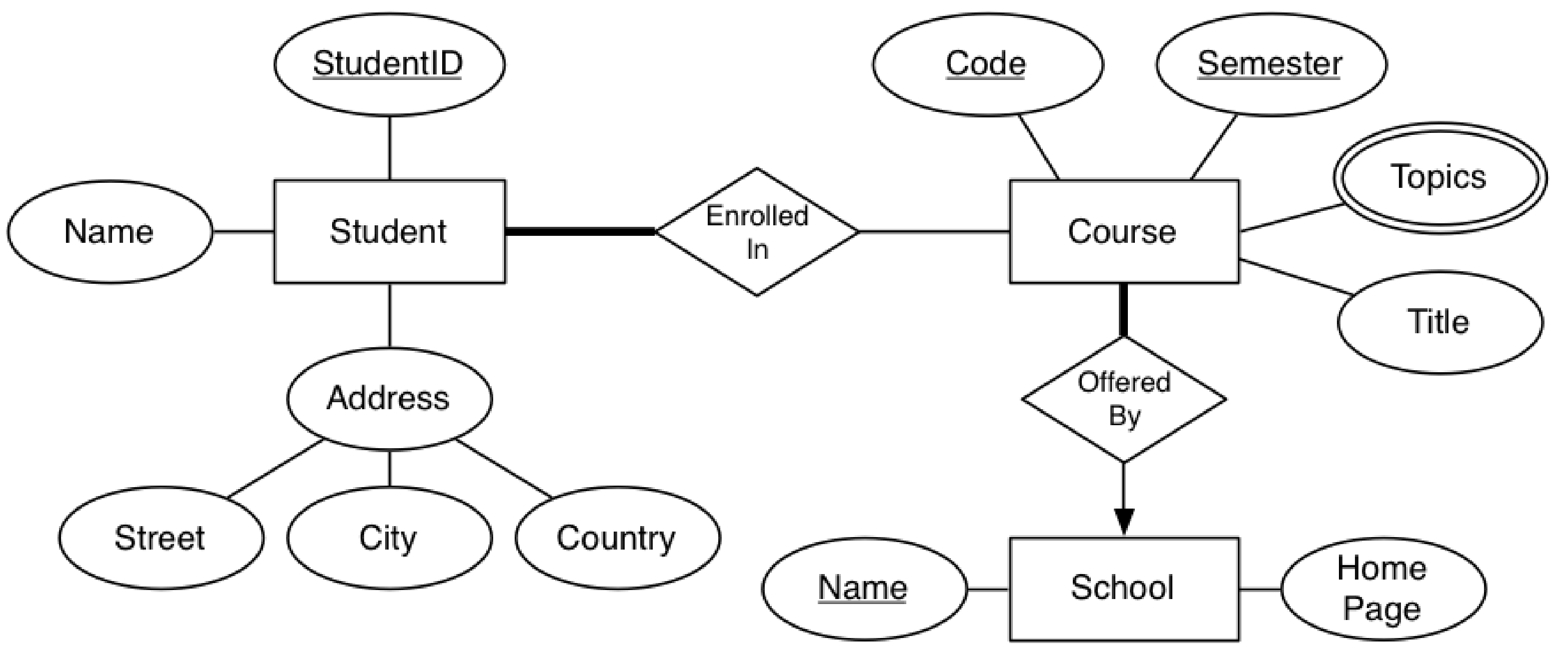

Week 09

# Entity Relationship Diagrams
# Story so far…

# ER Diagrams

- Last week, we looked at entity-sets and entity instances, attributes, keys and defining relationships (cardinality, degree, participation)
- Putting it all together…

  Example 1:

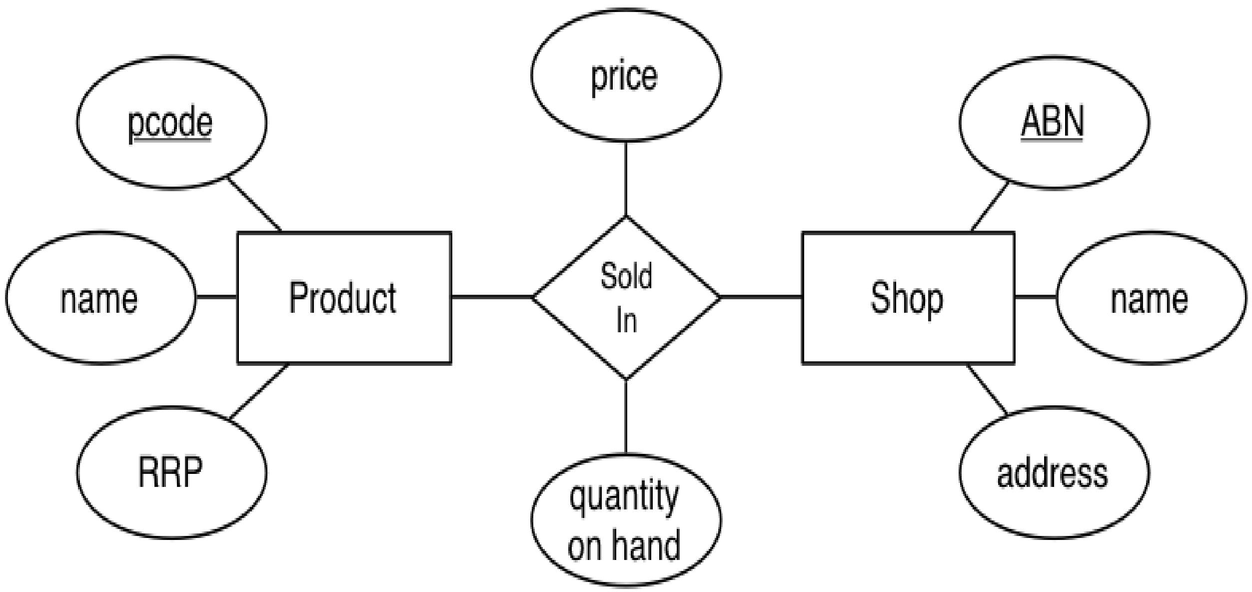



primary key attributes are underlined e.g. StudentID#

# Relationship Type with attributes

In some cases, a relationship needs associated attributes

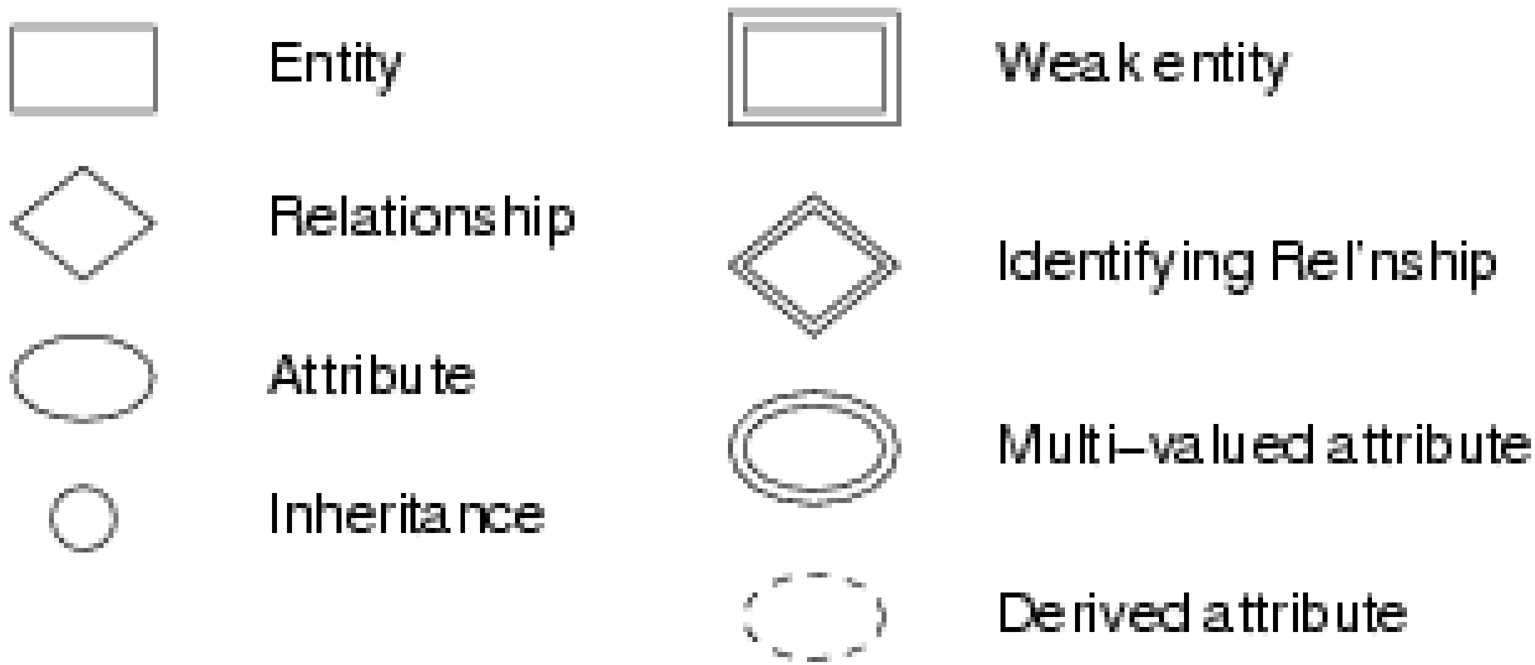
**Example**:



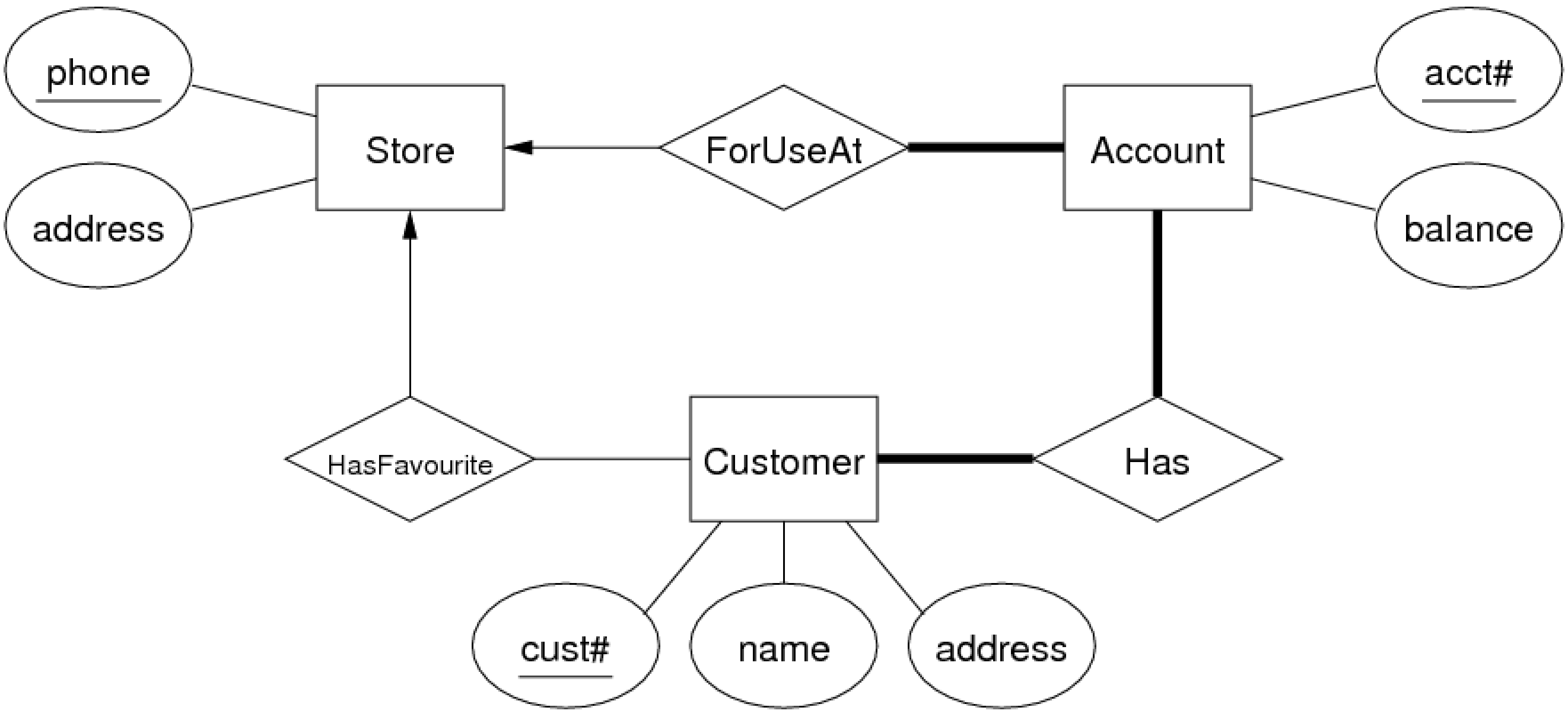

(price and quantity are related to products in a particular shop)

# Summarising notations in an Entity Relationship Diagram

Specific visual symbols indicate different ER design elements:



# Another Example:



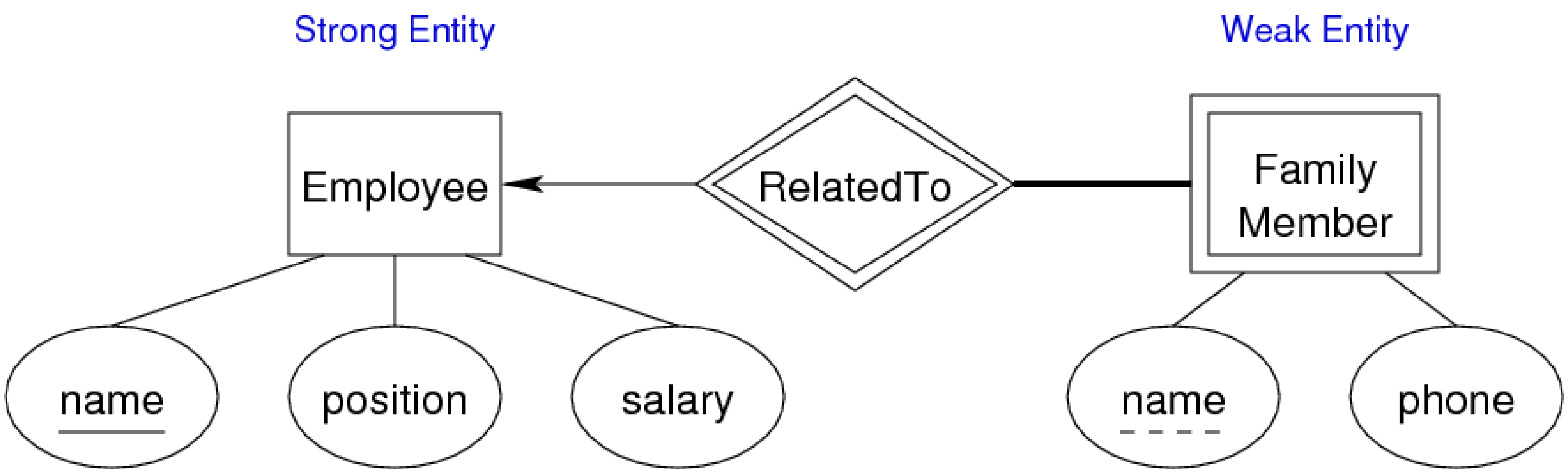

key attributes are <u>underlined</u> e.g. cust#

# Weak Entity Set

A Weak entity set

- has no key of its own;
- exist only because of association with strong entities

Example:



# ER model vs OO model

Analogy between ER and OO models:

- an entity is like an object instance
- an entity set is like a class

Differences between ER and OO models:

- ER modelling doesn't consider operations (methods)

# Subclasses and Inheritance

A subclass of an entity set *A* is a set of entities:

- with all attributes of *A*, plus (usually) it own attributes
- that is involved in all of *A*'s relationships, plus its own

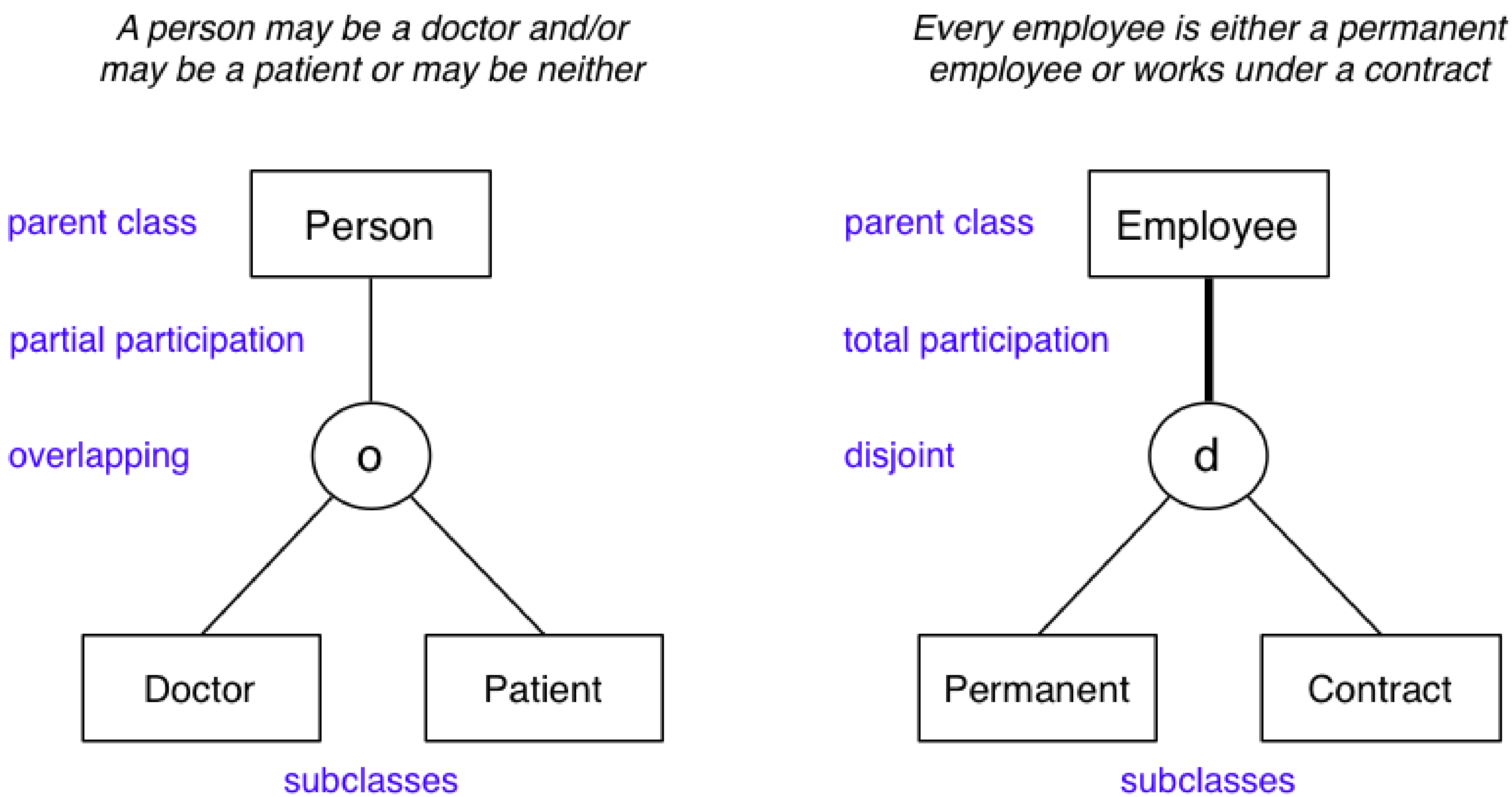
Properties of subclasses:

- overlapping or disjoint (can an entity be in multiple subclasses?)
- total or partial (does every entity have to also be in a subclass?)

Special case: entity has one subclass ("B is-a A" specialisation)

# Subclasses and Inheritance

## Example:



# Design considerations using the ER model

- should an "object" be represented by an attribute or entity?
- is a "concept" best expressed as an entity or relationship?
- should we use *n*-way rel$^{n}$ship or several 2-way rel$^{n}$ships?
- is an "object" a strong or weak entity? (usually strong)
- are there subclasses/superclasses within the entities?

Answers to above are worked out by *thinking* about the application domain.

# Exercise 1:

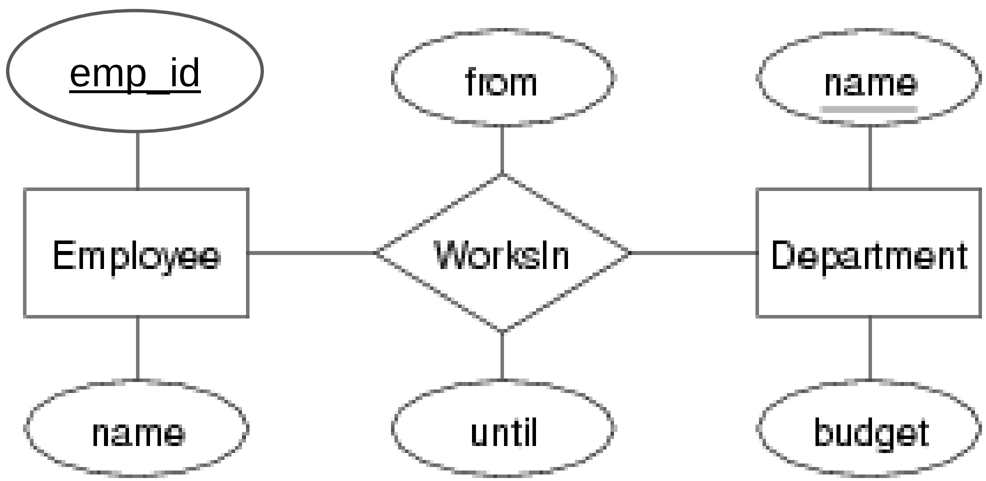
Develop an ER design for the following scenario:

A database records information about employees and the departments they work for:

- For each employee, the name and emp_id
- For each department, the name and allocated budget
- An employee may work for several departments for different periods of time
- A department may have several employees working for it

# Design considerations …

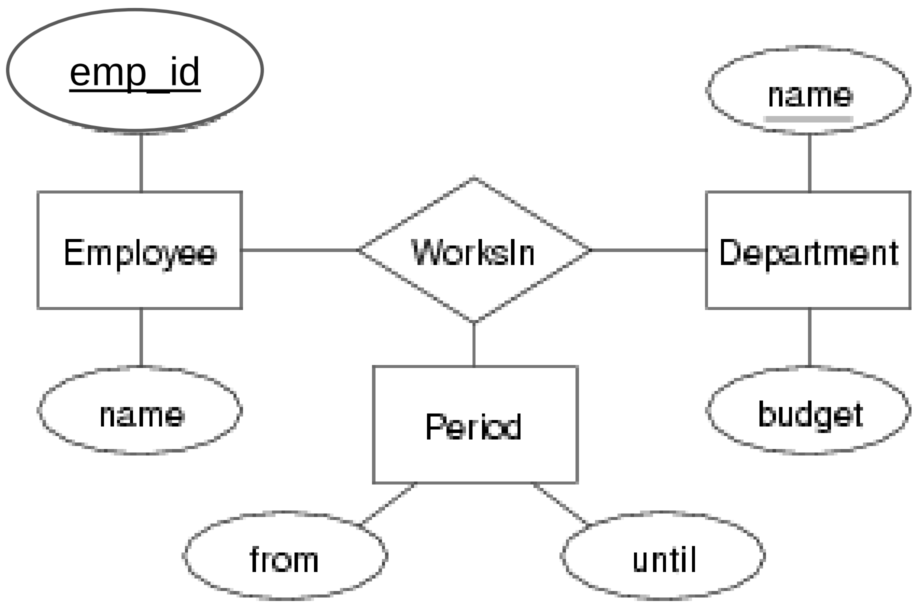
## Attribute vs Entity Example (v1)




Assumption: Employees can work for several departments, but cannot work for the same department over two different time periods.

# Design considerations …

## Attribute vs Entity Example (v2)




Assumption: Employees can work for the same department over two different time periods.

# Design using the ER model

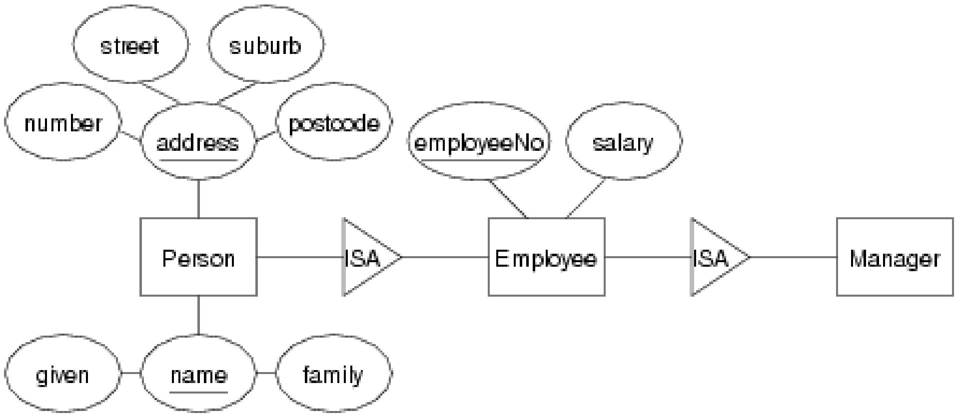
ER diagrams are typically too large to fit on a single screen. (or a single sheet of paper, if printing)

One commonly used strategy:

- define entity sets separately, showing attributes
- combine entities and relationships on a single diagram (but without showing entity attributes)
- if very large design, may use several linked diagrams as seen in the example in the next three set of slides

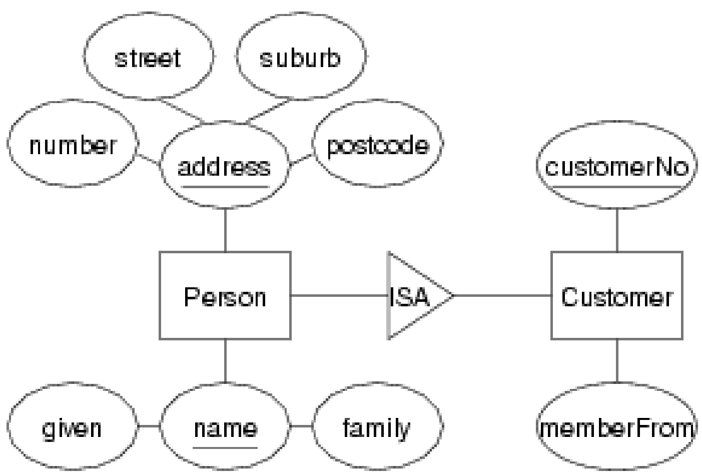
# e.g. an ER model for a Bank

## (1) Modelling people (employees)



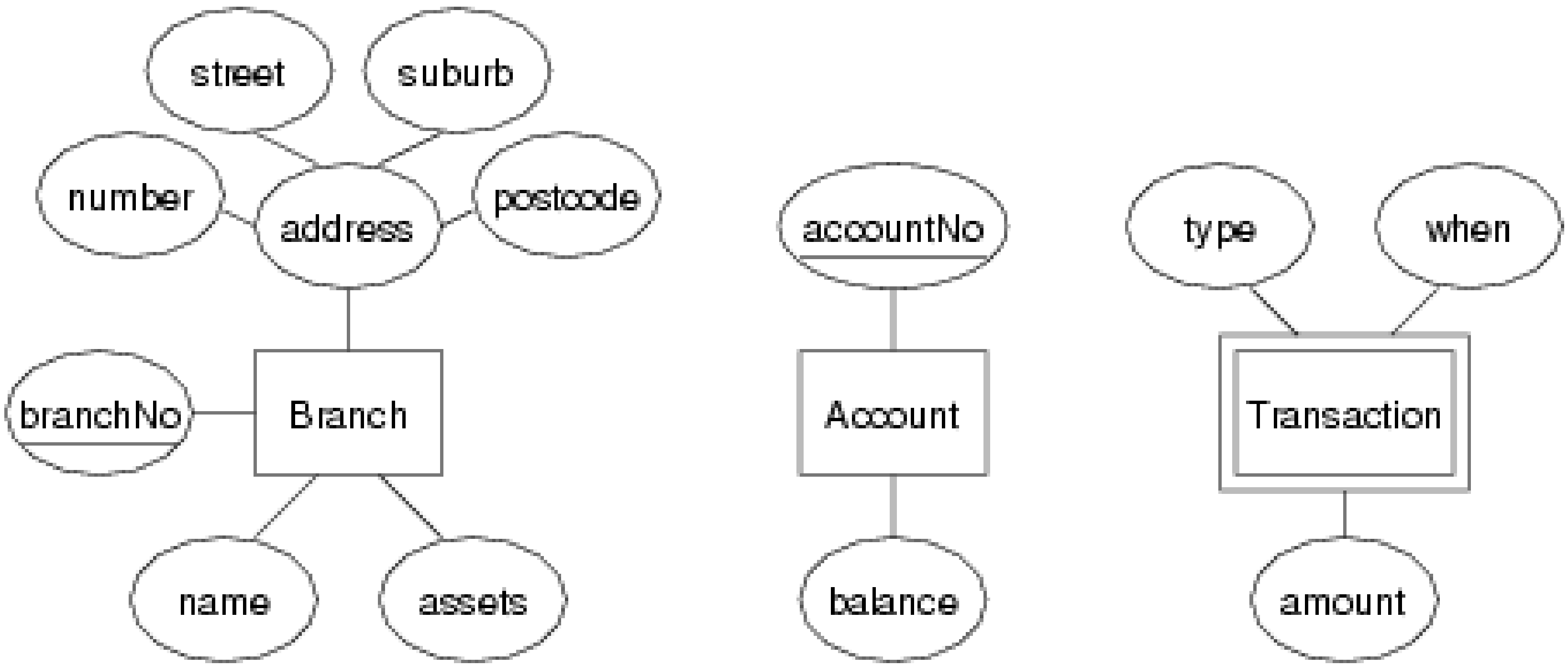

## (2) Modelling people (customer)

Modelling people (cont):



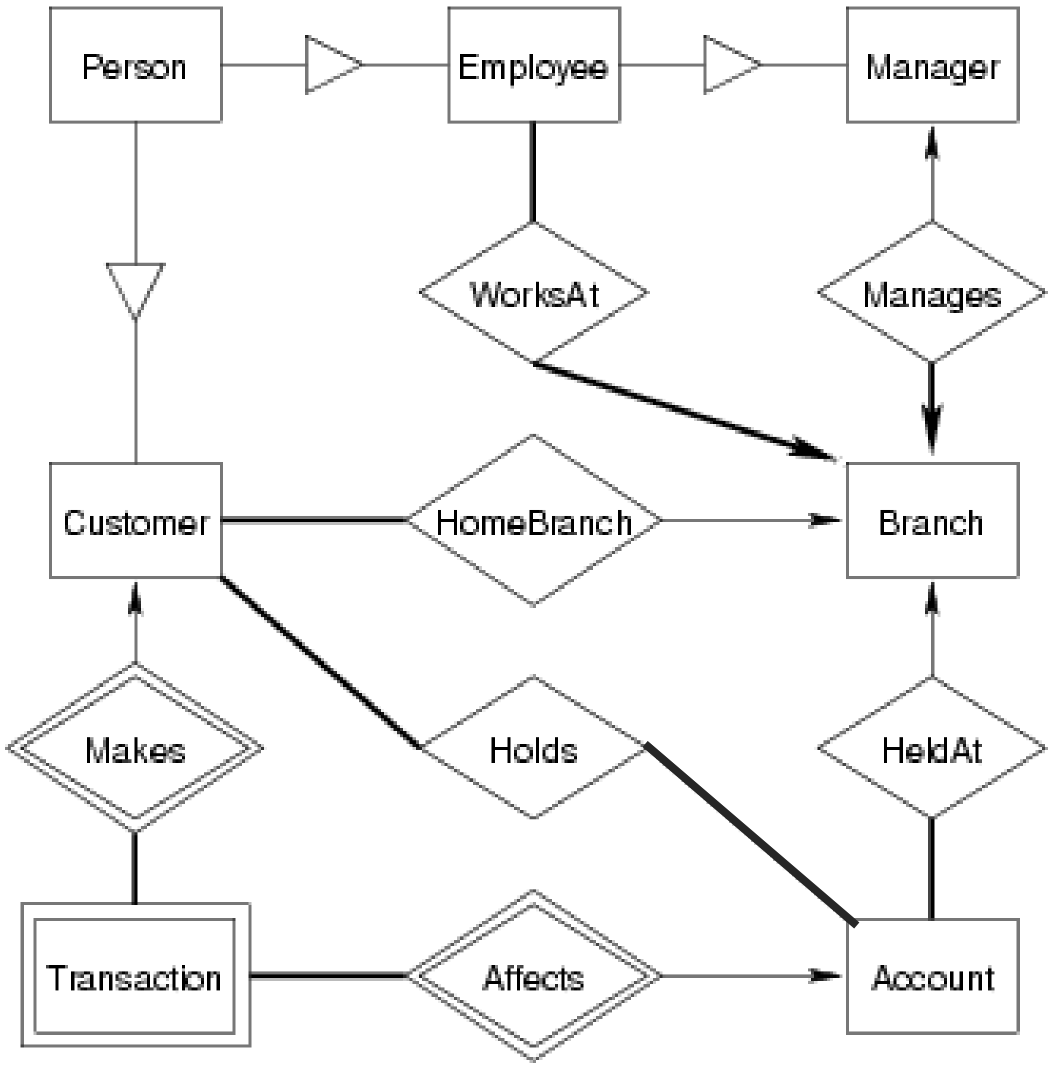
# e.g. an ER model for a Bank

## (3) Modelling branches, accounts, transactions



# e.g. an ER model for a Bank

## (4) Putting it all together with relationships

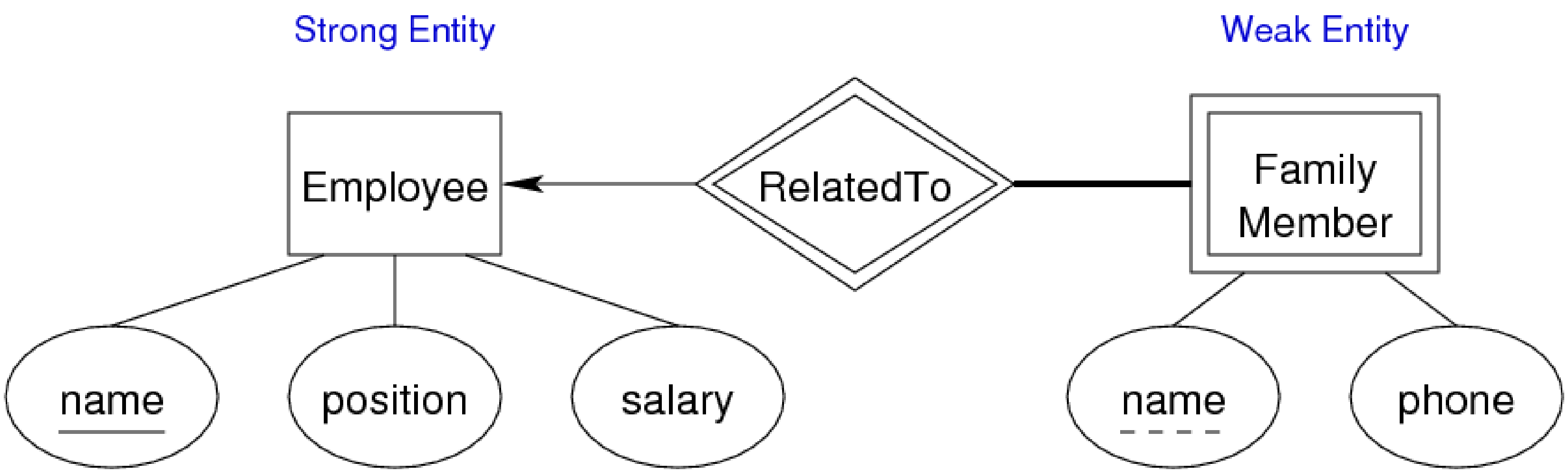


# Weak Entity Set

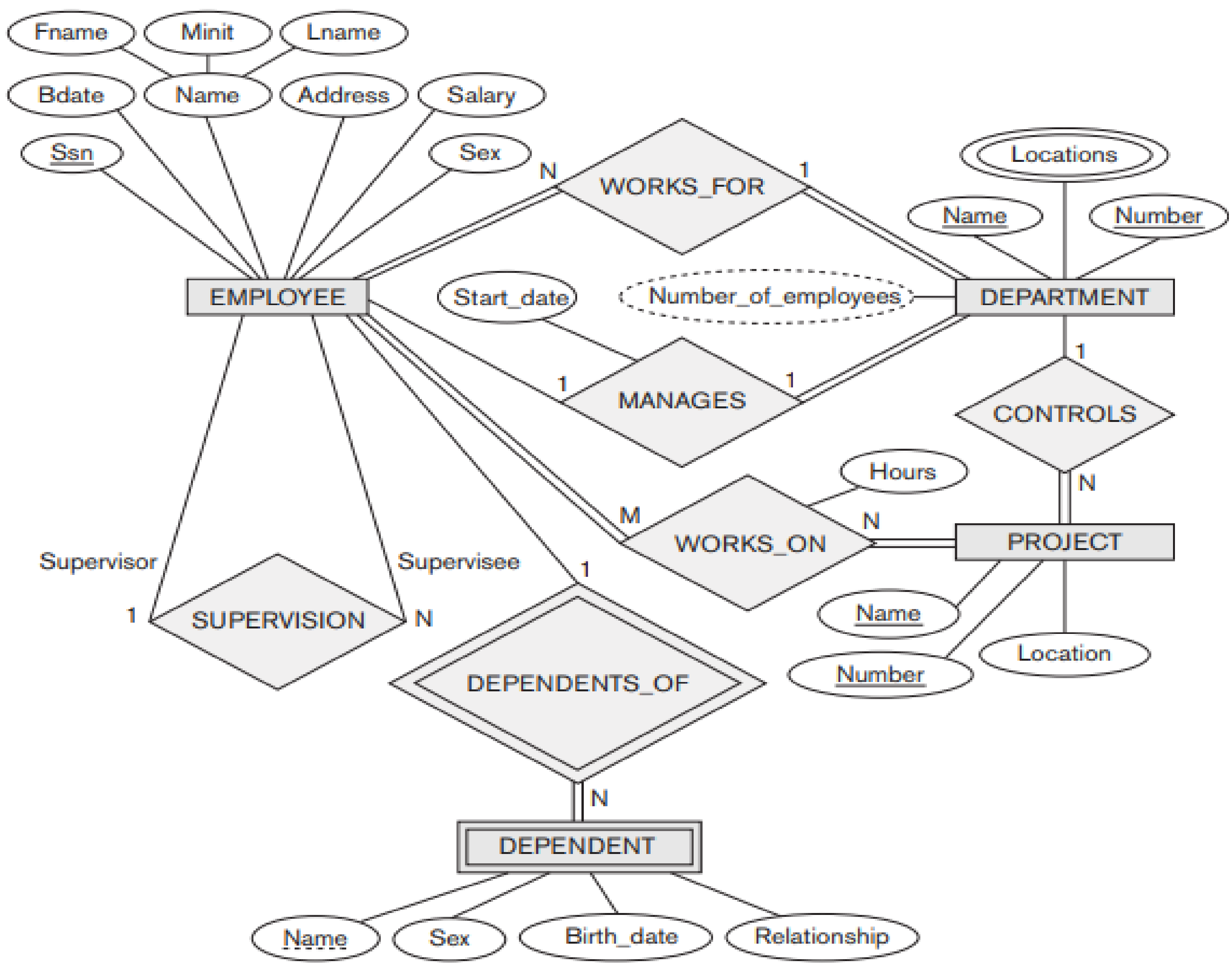
A Weak entity set

- has no key of its own;
- exists only because of association with strong entities

Example:

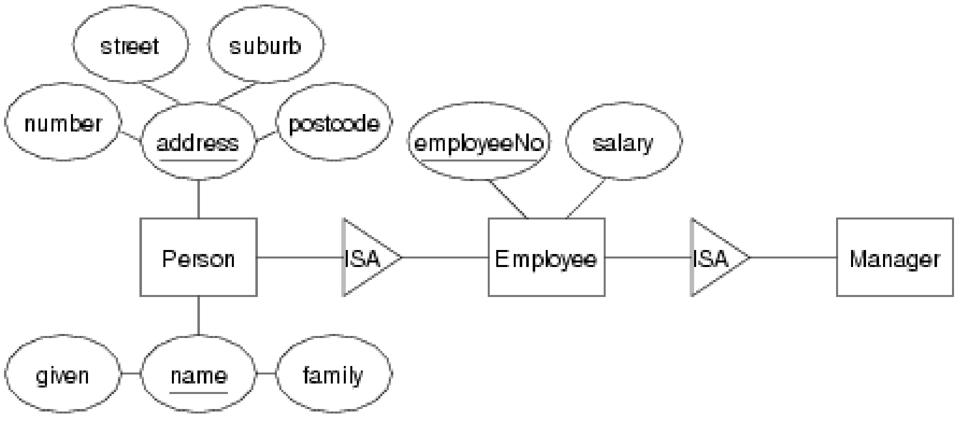


# A complex ER Model

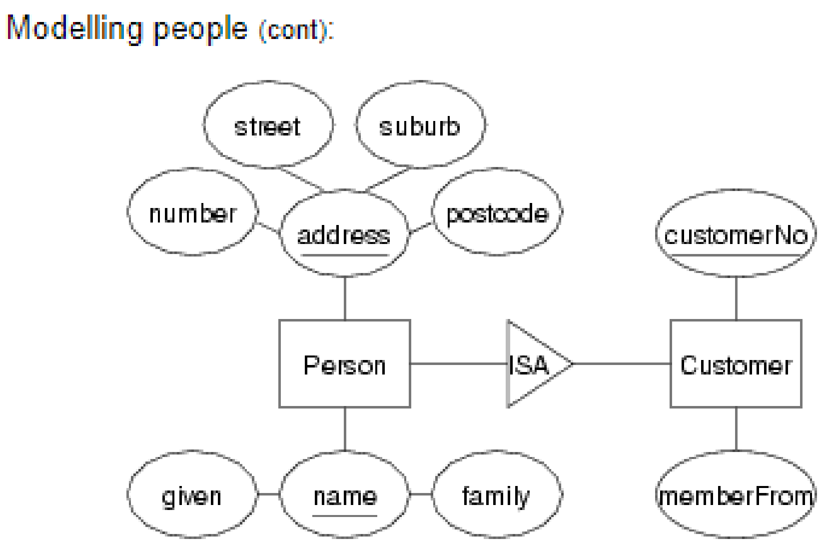


# e.g. an ER model for a Bank

## (1) Modelling people (employees)

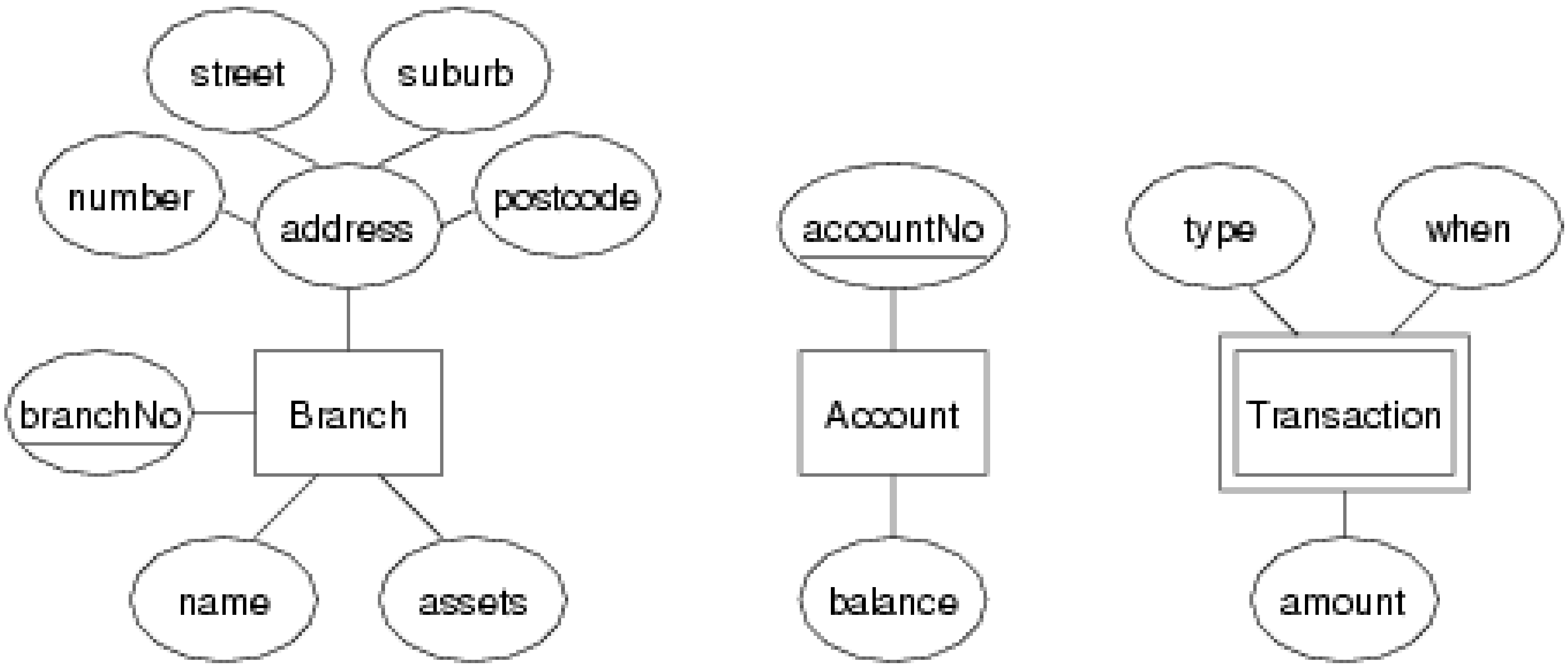



## (2) Modelling people (customer)



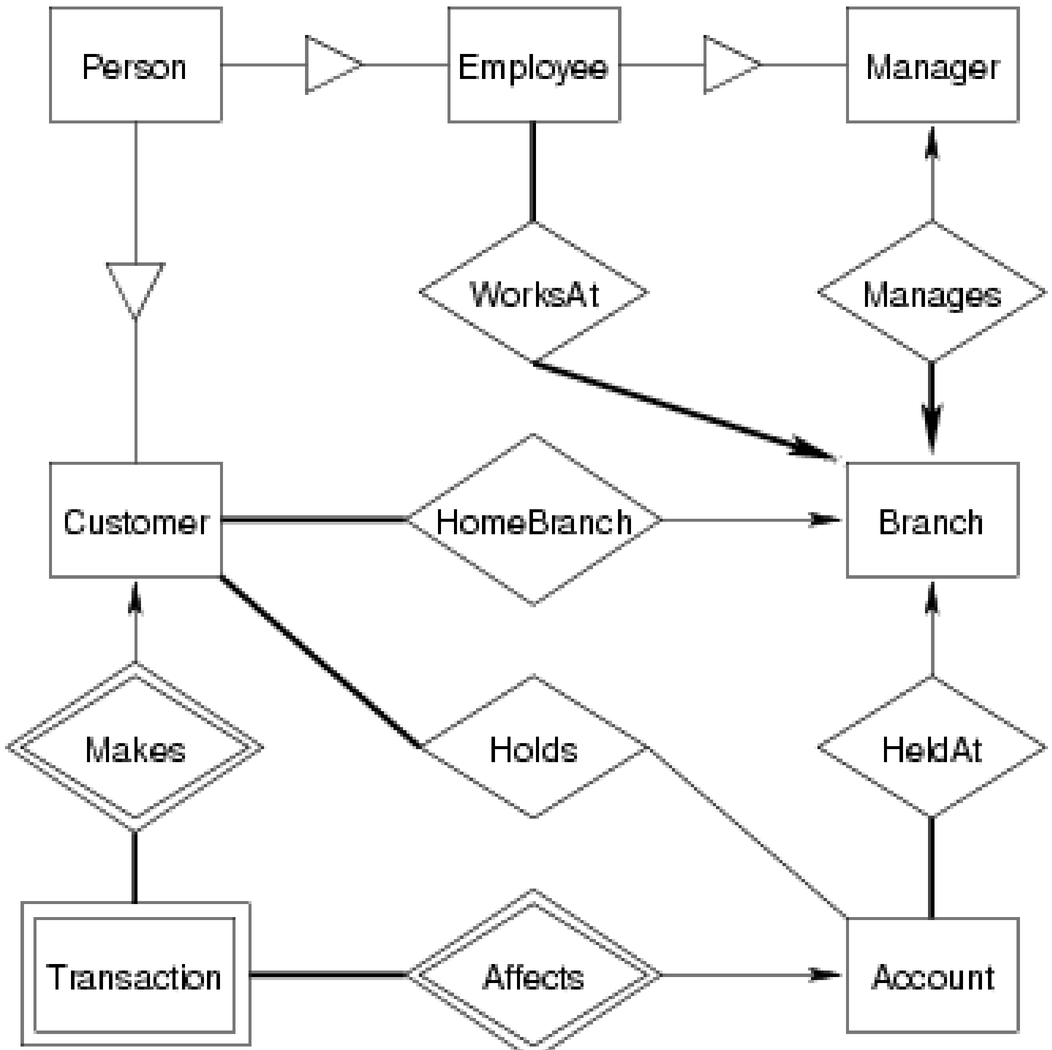
# e.g. an ER model for a Bank

## (3) Modelling branches, accounts, transactions



# e.g. an ER model for a Bank

## (4) Putting it all together with relationships



# Exercise 2:

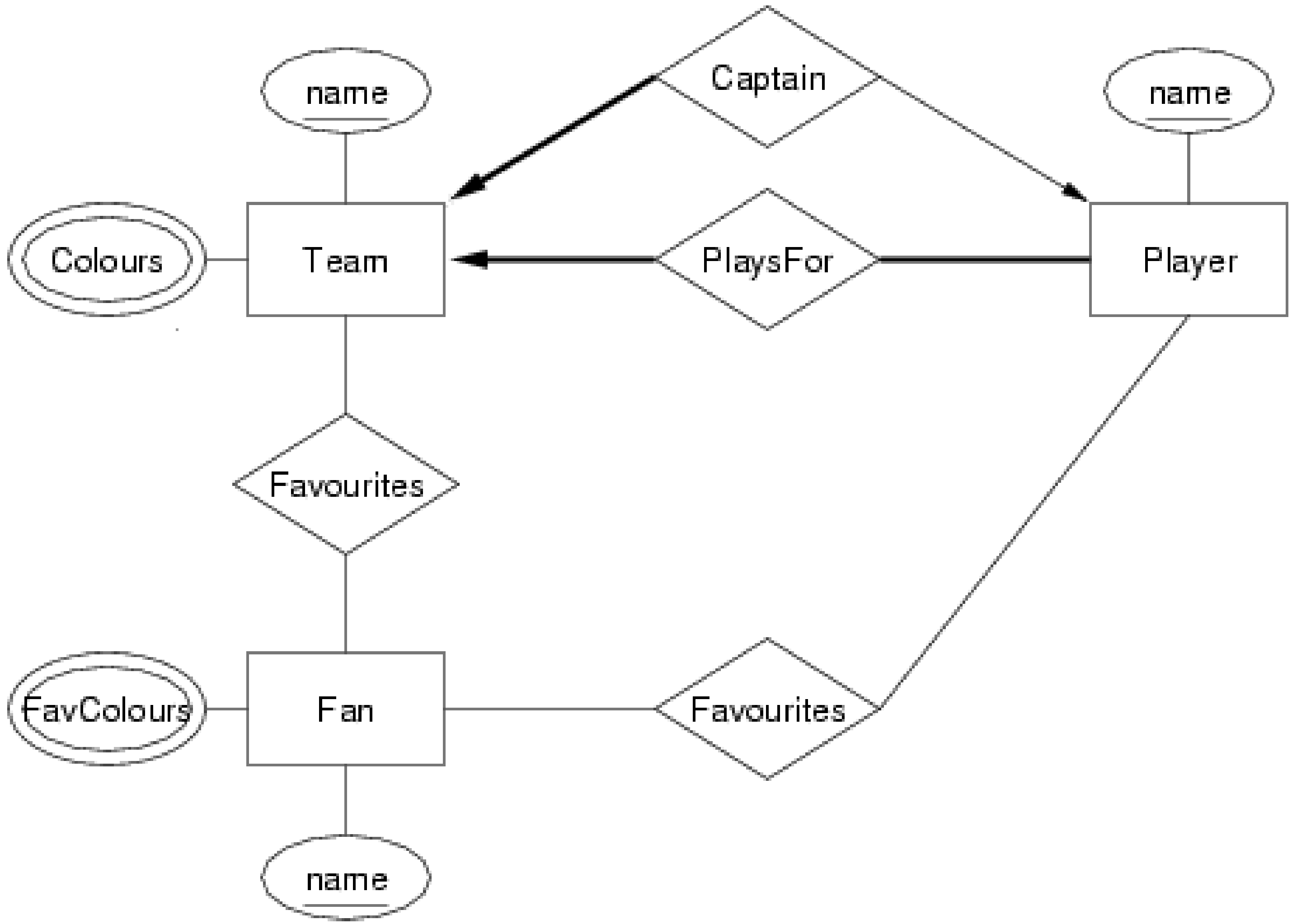
Develop an ER design for the following scenario:

A database records information about teams, players, and their fans, including:

- For each team, its name, its players, its captain (one of its players) and the colours of its uniform.
- For each player, their name and team.
- For each fan, their name, favourite teams, favourite players, and favourite colour.

State all assumptions made

# Exercise 2 (Solution): ER Design



# Exercise 3: Give an ER design to model the following scenario ...

- Patients are identified by an **SSN**, and their **names**, **addresses** and **ages** must be recorded.
- Doctors are identified by an SSN. For each doctor, the name, specialty and years of experience must be recorded.
- Each pharmacy has a name, address and phone number. A pharmacy **must** have a manager.
- A pharmacist is identified by an SSN, he/she can only work for one pharmacy. For each pharmacist, the name, qualification must be recorded.
- For each drug, the trade name and formula must be recorded.
- Every patient has a primary physician. Every doctor has at least one patient.
- Each pharmacy **sells** several drugs, and has a price for each. A drug could be sold at several pharmacies, and the price could vary between pharmacies.
- Doctors prescribe drugs for patients. A doctor could prescribe one or more drugs for several patients, and a patient could obtain prescriptions from several doctors. Each prescription has a date and quantity associated with it.

# What kind of data, relationships and constraints exist?

## Possible data:

- people: doctors, patients, pharmacists
- pharmacies, drugs
- person's SSN, name, address(?)
- doctor: as for person + specialty
- pharmacist: as for person + qualification etc.

## Possible relationships:

- doctors **treat** patients, patients **have primary** physician
- pharmacists **work in** pharmacies
- drugs **are sold in** pharmacies
- doctors **prescribe** drugs **for** patients etc.

## Possible constraints:

- every person has **exactly one, unique** SSN
- pharamacist works in ≤ 1 pharmacy
- patient has **exactly one** primary physician
- doctor treats ≥ 1 patient etc.

# Summary of ER

- ER model is popular for doing conceptual design
  - high-level, models relatively easy to understand
  - good expressive power, can capture many details
- Basic constructs:
  - entities,   relationships,   attributes
  - relationship constraints:   total / partial,   n:m / 1:n / 1:1
- Other constructs:
  - inheritance hierarchies,   weak entities
- Many notational variants of ER exist
  (especially in the expression of constraints on relationships)

# Relational Data Model

# Relational Data Model

The relational data model describes the world as a collection of inter-connected relations (or tables)

Goal of relational model:

- a simple, general data modelling formalism
- maps easily to file structures (i.e. implementable)

Relational model has two styles of terminology:

| | | | |
|---|---|---|---|
| mathematical | relation | tuple | attribute |
| data-oriented | table | record (row) | field (column) |

**STUDENT**

| Name | Student_number | Class | Major |
|---|---|---|---|
| Smith | 17 | 1 | CS |
| Brown | 8 | 2 | CS |

# Relational Data Model

The relational model has one structuring mechanism ...

- a relation corresponds to a mathematical "relation"
- a relation can also be viewed as a "table"

Each relation (table) (denoted $R,S,T,\ldots$) has:

- a name (unique within a given database)
- a set of attributes (or column headings)

Each attribute (denoted $A,B,\ldots$ or $a_1,a_2,\ldots$) has:

- a name (unique within a given relation)
- an associated domain (set of allowed values)

DB definitions also make extensive use of constraints

# Relational Data Model

Example of a relation (table): Bank Account

*Relation, Table*

*Attributes, Columns, Fields*

**Account**

| **branchName** | **accountNo** | **balance** |
|---|---|---|
| Downtown | A–101 | 500 |
| Mianus | A–215 | 700 |
| Perryridge | A–102 | 400 |
| Round Hill | A–305 | 350 |
| Brighton | A–201 | 900 |
| Redwood | A–222 | 700 |
| Brighton | A–217 | 750 |

*Tuples, Rows, Records*

# Relational Data Model

A ***tuple (row)*** is a **set** of **values (attribute or column values)**

Attribute values:

- Are atomic (no composite or multi-valued attributes).
- Belong to a domain; a domain has a **name**, **data type** and **format**
- A distinguished value **NULL** belongs to all domains.
- NULL has several interpretations: none, don't know, irrelevant

| **Column Header** | **Domain Name** | **Domain Data Type, Format and Constrain** |
|---|---|---|
| phone_number | local_phone_numbers - (set of phone numbers valid in australia) | character string of the format *(dd) dddddddd*, where each *d* is a numeric (decimal) digit and the first two digits form a<br>valid telephone area code. |
| age | employee_age (set of possible ages for employees in the company) | An integer value between 15 and 80 |

# Relational Data Model

- A ***relation(table)*** is a **set** of ***tuples***.
- Since a relation is a set, there is **no ordering** on rows.
- Normally, **we define a standard ordering** on components of a tuple.
- The following are different presentations of the same relation:

| branchName | accountNo | balance |
|---|---|---|
| Downtown | A–101 | 500 |
| Mianus | A–215 | 700 |
| Perryridge | A–102 | 400 |
| Round Hill | A–305 | 350 |
| Redwood | A–222 | 700 |

| accountNo | branchName | balance |
|---|---|---|
| A–305 | Round Hill | 350 |
| A–222 | Redwood | 700 |
| A–215 | Mianus | 700 |
| A–102 | Perryridge | 400 |
| A–101 | Downtown | 500 |

- Each relation generally has a **primary key** (subset of attributes, unique over relation)

# Relational Data Model

A ***database*** is a **set** of ***relations(tables)***

**Account**

| branchName | accountNo | balance |
|---|---|---|
| Downtown | A–101 | 500 |
| Mianus | A–215 | 700 |
| Perryridge | A–102 | 400 |
| Round Hill | A–305 | 350 |
| Brighton | A–201 | 900 |
| Redwood | A–222 | 700 |

**Branch**

| branchName | address | assets |
|---|---|---|
| Downtown | Brooklyn | 9000000 |
| Redwood | Palo Alto | 2100000 |
| Perryridge | Horseneck | 1700000 |
| Mianus | Horseneck | 400000 |
| Round Hill | Horseneck | 8000000 |
| North Town | Rye | 3700000 |
| Brighton | Brooklyn | 7100000 |

**Customer**

| name | address | customerNo | homeBranch |
|---|---|---|---|
| Smith | Rye | 1234567 | Mianus |
| Jones | Palo Alto | 9876543 | Redwood |
| Smith | Brooklyn | 1313131 | Downtown |
| Curry | Rye | 1111111 | Mianus |

**Depositor**

| account | customer |
|---|---|
| A–101 | 1313131 |
| A–215 | 1111111 |
| A–102 | 1313131 |
| A–305 | 1234567 |
| A–201 | 9876543 |
| A–222 | 1111111 |
| A–102 | 1234567 |

# Expressing Relational Data Model Mathematically

Given a relation (table) $R$ which has:

- $n$ attributes $a_1, a_2, \ldots a_n$
- with corresponding domains $D_1, D_2, \ldots D_n$

We define:

- Relation Schema of $R$ as: $\boldsymbol{R}(a_1{:}D_1, a_2{:}D_2, \ldots a_n{:}D_n)$
- Tuple of $R$ as: an element of $D_1 \times D_2 \times \ldots \times D_n$ (i.e. list of values)
- Instance of $R$ as: subset of $D_1 \times D_2 \times \ldots \times D_n$ (i.e. set of tuples)
- Database schema : a collection of relation schemas.
- Database (instance) : a collection of relation instances.

# Example of a Relation Schema

Given a relation or table, Account , which has:

- 3 attributes branchName, accountNo, balance
- with corresponding domains string, string, int ,

then we can define the schema of Account as:

**Account** (branchName:string, accountNo:string, balance: int) OR simply as

**Account** (branchName, accountNo, balance)

and a tuple Account (i.e., row of Account )can be specified as:

(Downtown, A-101, 500)

and an instance of relation Account (a set of tuples or rows ) as:

*** No duplicates**

{ (Downtown, A-101, 500), (Mianus, A-215, 700),
(Perryridge, A-102, 400), (Round Hill, A-305, 350),
(Brighton, A-201, 900), (Redwood, A-222, 700)}

# Relation Schema

- The **degree** (or **arity**) of a relation is the number of attributes *n* of its relation schema, so a relation schema *R* of degree *n* is denoted by $R(a_1, a_2, \ldots a_n)$.

e.g. A relation of degree seven, which stores information about university students, would contain seven attributes describing each student. as follows:

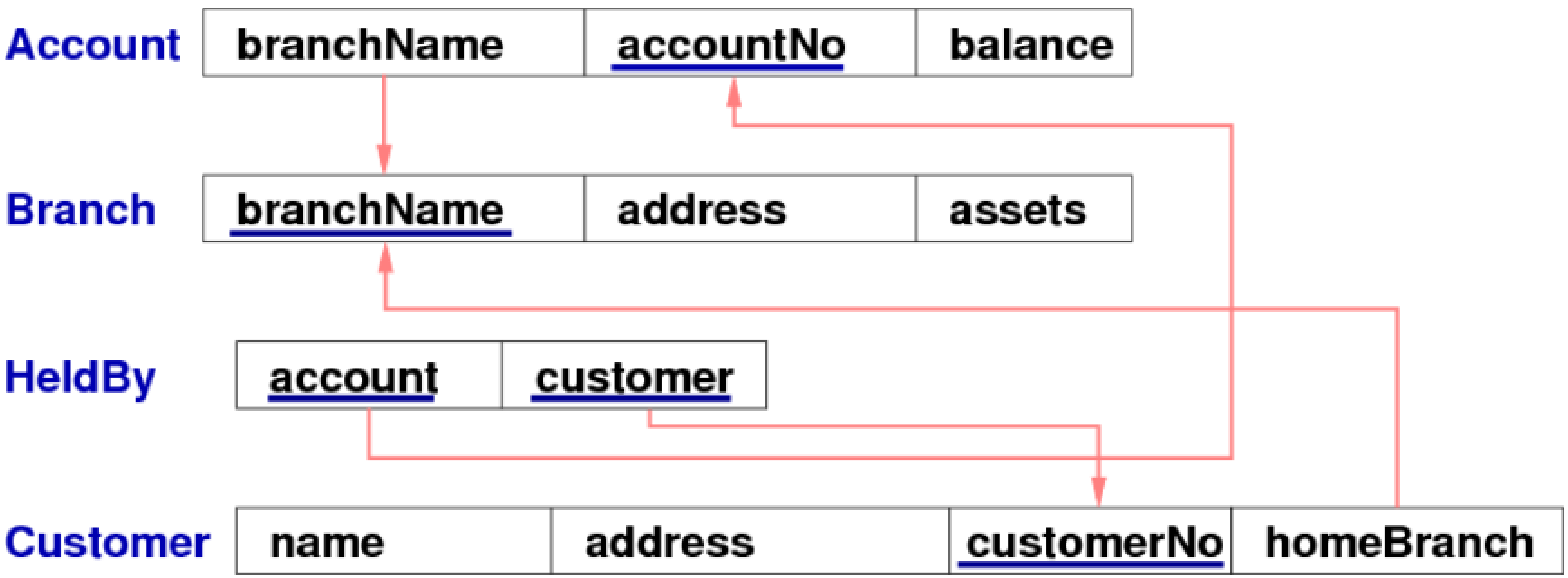
```
Student (name, ssn, home_phone, address,
office_phone, age, gpa)
```

OR as

**Student** (name: string, ssn: string, home_phone: string, address: string, office_phone: string, age: integer, gpa: int)

# Database Schema – a collection of relation schemas

Example of a **database schema** with 4 relation schemas:



# Database Instance – a collection of relation instances

Example of a **database instance** with 4 relation instances:

**Account**

| branchName | accountNo | balance |
|---|---|---|
| Downtown | A–101 | 500 |
| Mianus | A–215 | 700 |
| Perryridge | A–102 | 400 |
| Round Hill | A–305 | 350 |
| Brighton | A–201 | 900 |
| Redwood | A–222 | 700 |

**Branch**

| branchName | address | assets |
|---|---|---|
| Downtown | Brooklyn | 9000000 |
| Redwood | Palo Alto | 2100000 |
| Perryridge | Horseneck | 1700000 |
| Mianus | Horseneck | 400000 |
| Round Hill | Horseneck | 8000000 |
| North Town | Rye | 3700000 |
| Brighton | Brooklyn | 7100000 |

**Customer**

| name | address | customerNo | homeBranch |
|---|---|---|---|
| Smith | Rye | 1234567 | Mianus |
| Jones | Palo Alto | 9876543 | Redwood |
| Smith | Brooklyn | 1313131 | Downtown |
| Curry | Rye | 1111111 | Mianus |

**HeldBy**

| account | customer |
|---|---|
| A–101 | 1313131 |
| A–215 | 1111111 |
| A–102 | 1313131 |
| A–305 | 1234567 |
| A–201 | 9876543 |
| A–222 | 1111111 |
| A–102 | 1234567 |

# Constraints

- Relations are used to represent real-world *entities* and *relationships* between these *entities*
- To represent real-world problems, need to describe
  - what values are/are not allowed
  - what combinations of values are/are not allowed
- Constraints are logical statements that do this:
  - Domain constraint
  - Key constraint
  - Referential integrity
- A DBMS should provide capabilities to enforce these constraints

# Key (Uniqueness) Constraint

- A key attribute is one whose value can be used to uniquely identify each tuple (row) in the relation
- A relation can have more than one key, so each key is a candidate key e.g., registration and VIN_Chassis_no

| Registration | VIN_Chassis_No | Make | Model |
|---|---|---|---|
| NSW XNCNC – 121 | A695323 | Mercedes | GLC Coupe |
| VIC ABC145 – 908 | BX98765 | Mercedes | C Class |

Example of key constraints:

- Student( id, ...) is guaranteed unique
- Class(..., day, time, location,...) is unique

# Entity Integrity Constraint

- One of the candidate keys is designated as the primary key of the relation
- An entity integrity constraint states that no primary key value can be NULL

# Domain Constraints

Domain constraints specify that within each tuple, the value of each attribute $(a_1, a_2, \dots a_n)$ must be an atomic value from the corresponding domain $D_1, D_2, \dots D_n$

e.g.,

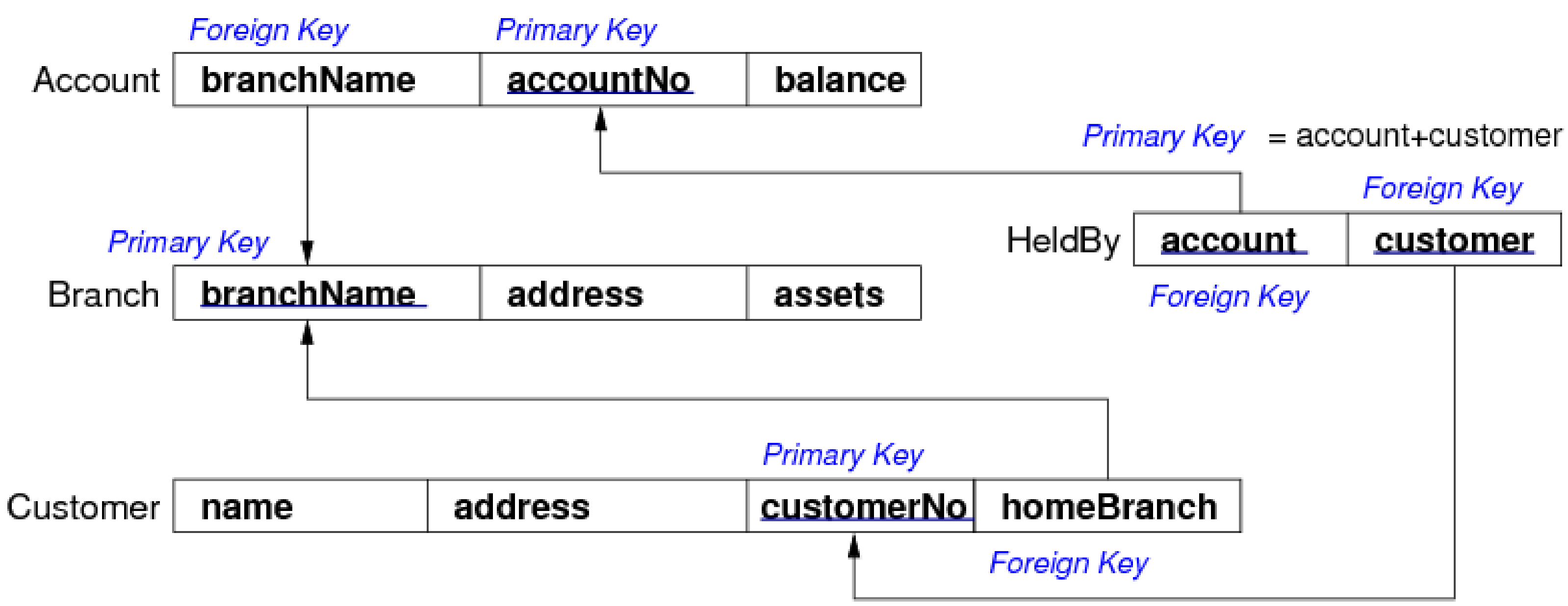
- Employee.age attribute is typically defined as integer
- better modelled by adding extra constraint (15<age<66)

Note: NULL satisfies all domain constraints (except (NOT NULL))

# Referential Integrity Constraint

- describe references between relations (tables)
- are related to the notion of a foreign key (FK), where the primary key of the parent table is linked to a foreign key in the child table

  e.g., the Account relation (child) needs to take note of the branch where each account is held
- the notion that the branchName (in Account) ***must refer*** to a valid branchName (in Branch) is a **referential integrity constraint**



# Referential Integrity Constraints

- Related to the notion of a foreign key
- A set of attributes *F* in relation $R_1$ is a foreign key if:
  - the attributes in *F* correspond to the attributes in the primary key of another relation $R_2$
  - the value for *F* in each tuple of $R_1$
    - either occurs as a primary key in $R_2$
    - or is entirely NULL
- Foreign keys are critical in relational DBs;
  - they provide ...the "glue" that links individual relations (tables)
  - the way to assemble query answers from multiple tables

# Putting all together…

A relational database schema is viewed as:

- a set of relation schemas $\{ R_1, \ R_2, \ \ldots \ R_n \}$, and
- a set of integrity constraints

A relational database instance is

- a set of relation instances $\{ r_1(R_1), \ r_2(R_2), \ \ldots \ r_n(R_n) \}$
- where all of the integrity constraints are satisfied

One of the important functions of a relational DBMS:

- ensure that all data in the database satisfies constraints

# Relational Model vs DBMS

The relational model is a mathematical construct

- giving a representation for data structures
- with constraints on relations/tuples
- and an *algebra* for manipulating relations/tuples (union, intersect…)

Relational Database Management Systems (RDBMS)

- provide an implementation of the relational model
- Uses SQL (Structured Query Language) as language for data definition, query, updates

# Describing Relational Schemas

- SQL (Structured Query Language) provides the formalism to express relational schemas

- SQL provides a Data Definition Language (DDL) for creating relations

```
CREATE TABLE TableName (
attrName1 domain1 constraints1 ,
attrName2 domain2 constraints2 , ...
PRIMARY KEY (attri,attrj,...)
FOREIGN KEY (attrx,attry,...)
REFERENCES OtherTable (attrm,attrn,...)
);
```

# DBMS Terminology

To remember:

- DBMS-level ... database names must be unique
- database-level ... schema names must be unique
- schema-level ... table names must be unique
- table-level ... attribute names must be unique

Sometimes it is convenient to use same name in several tables

We distinguish which attribute we mean using qualified names

e.g. **Account.branchName** vs **Branch.branchName**

# Features of RDBMS

- Support large-scale data-intensive applications
- Provide efficient storage and retrieval (disk/memory management)
- Support multiple simultaneous users (privilege, protection)
- Support multiple simultaneous operations (transactions, concurrency)
- Maintain reliable access to the stored data (backup, recovery)
- Use SQL as language for:
  - data definition (creating, deleting relations i.e. tables)
  - relation query (selecting tuples)
  - relation update (changing relations)

# Mapping ER to Relational Data Model
# Tomorrow …

# Mapping ER Designs to Relational Model

# ER to Relational Mapping

By examining semantic correspondences, a formal mapping between the ER and relational models has been developed

| ER Model | Relational Model |
|---|---|
| ER attribute | attribute (atomic) |
| ER entity-instance<br>ER relationship-instance | tuple (row) |
| ER entity-set<br>ER relationship | relation (table) |
| ER key | primary key of relation |

# Relational Model vs Entity Model

There are also differences between relational and ER models
Compared to ER, the relational model:

- Uses *relations* to model *entities* and *relationships*
- Has **no** *composite* or *multi-valued* attributes (only atomic)
- Has **no** *object-oriented notions* (e.g. subclasses, inheritance)

# (1) Mapping Strong Entities

An *entity* consists of:

- a collection of attributes;
  attributes are simple, composite and multi-valued

A *relation schema* consists of:

- a collection of attributes;
  all attributes have atomic data values

So, even the mapping from entity to relation schema is not simple

# (1) Mapping Strong Entities

An obvious mapping

- an entity set *E* with atomic attributes $a_1, a_2, \ldots A_n$

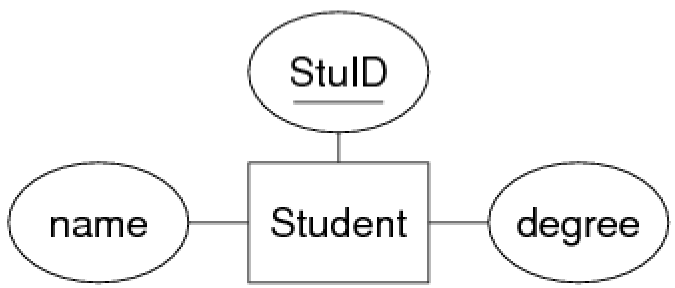
maps to

- a relation (table) *R* with attributes (columns) $a_1, a_2, \ldots A_n$

Each row in relation *R* corresponds to an entity in *E*

Example:

*ER Model*




*Relational Version*

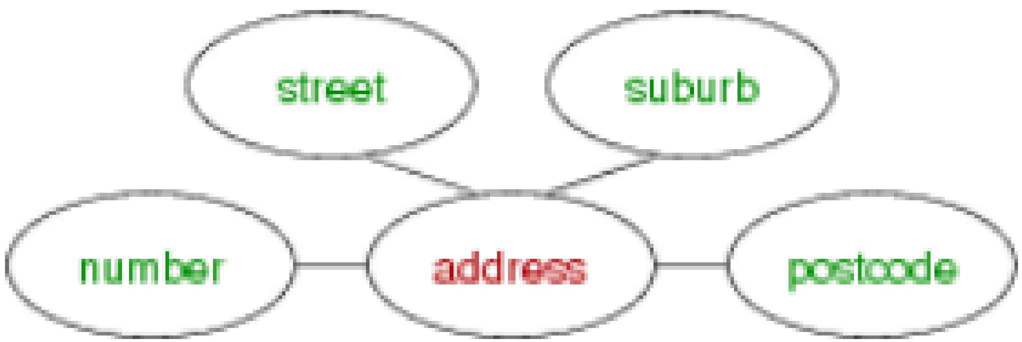
Student

| <u>**StuID**</u> | **name** | **degree** |
|---|---|---|

(Note: the key is preserved in the mapping)

# (2) Mapping Composite Attributes

- ER supports composite (hierarchical) attributes.
- The relational model supports only atomic attributes.
- Composite attributes consist of
  - structuring attributes (non-leaf attributes)
  - data attributes (containing atomic values)



# (2) Mapping Composite Attributes

- One approach : remove structuring attributes
  - map atomic components to a set of atomic attributes (possibly with renaming)
  - E.g. Addr {number, street, suburb, pcode} maps to (AddrNumber, AddrStreet, AddrSuburb, AddrPcode)

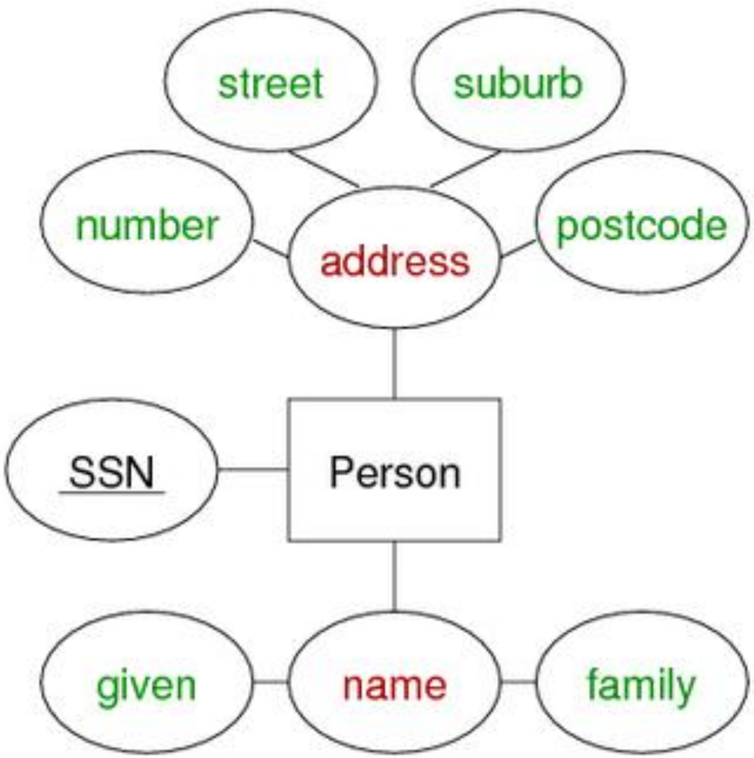
- Alternative approach: concatenate atomic attribute values into a string
  - e.g., name {"John","Smith"} → "John Smith“
  - However, this approach:
  - requires extra extraction effort if components *are* required
  - cannot exploit efficient query capabilities on components

# (2) Mapping composite attributes

Mapped by concatenation or flattening

Example:

*ER Model*




*Relational Version #1*

Person

| <u>SSN</u> | name | address |
|---|---|---|

*Relational Version #2*

Person

| <u>SSN</u> | givenName | familyName | ... |
|---|---|---|---|

| ... | number | street | suburb | postcode |
|---|---|---|---|---|

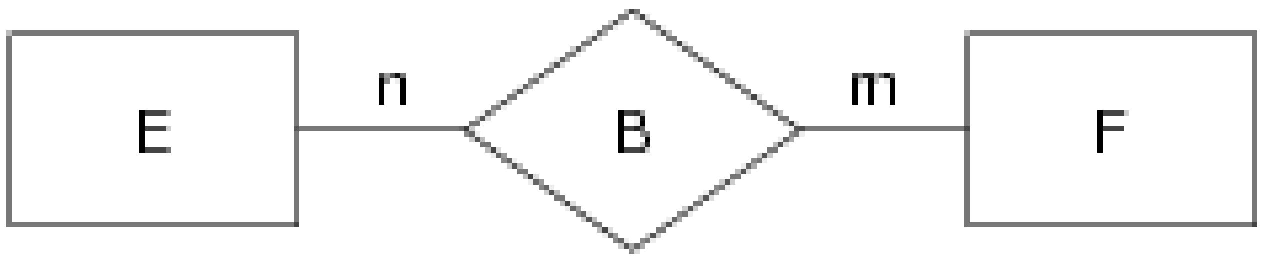
# (3) Mapping Relationships

ER **relationship** → relational **table** (relation)

- Identify one entity as "parent" and other entity as "child"
- as general rule,
  - **PK of parent is added to child as FK**
- Any attributes of the relationship
  - are added to ***child*** relation

# (3a) Mapping N:M Relationships

A <u>binary relationship set *B*</u> between entity sets *E* and *F* gives associations between pairs of entities in *E* and *F*




We can represent

- entity set *E* by relation *S* (using attribute mappings as above)
- entity set *F* by relation *T* (using attribute mappings as above)

But how to represent *B*?

# One possibility –

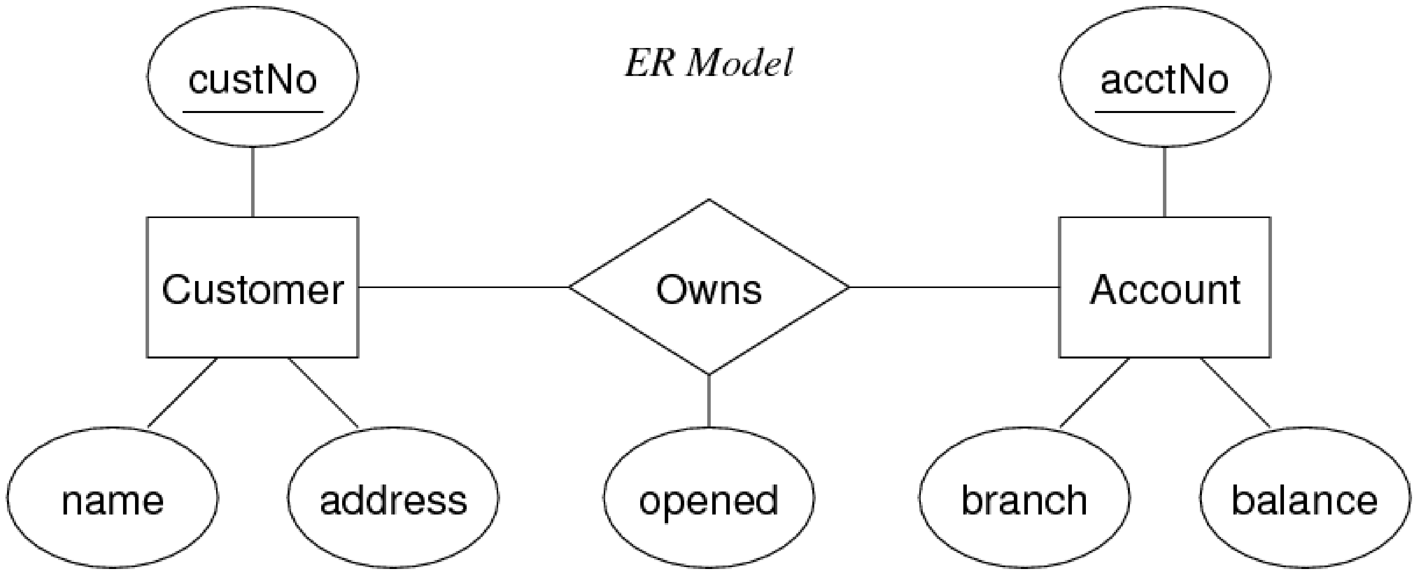
- Represent the relationship set *B* explicitly by a relation *R* containing:
  - all attributes from the primary keys of *S* and *T*
  - all attributes associated with the relationship set *B*

- where *S* and *T* are relations representing entity sets *E* and *F*
- and the *key* for *R* is the union of the key attributes for *S* and *T*

And this approach works generally for:

- relationship degree ≥ *2*
- relationship multiplicity 1:1, 1:N, N:M
- associated attributes are simply included in *R*

# Example - Mapping N:M Relationship




*Relational Version*

Customer

| <u>custNo</u> | name | address |
|---|---|---|

Account

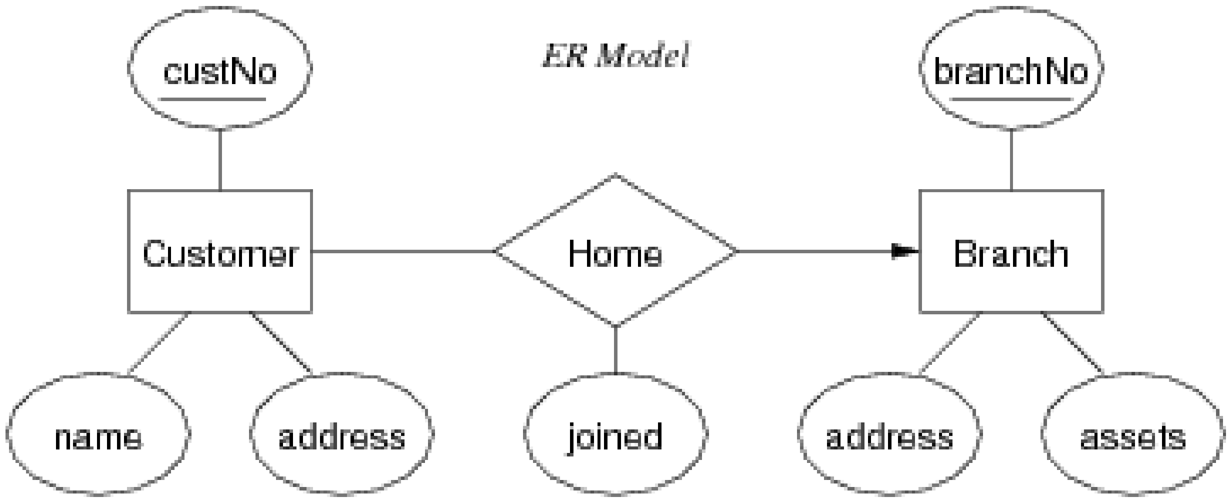
| <u>acctNo</u> | branch | balance |
|---|---|---|

Owns

| <u>custNo</u> | <u>acctNo</u> | opened |
|---|---|---|

# (3b) Mapping 1:M Relationships

## Example:




**Relational Version**

**Generic Mapping**

Customer

| custNo | name | address |
|---|---|---|

Branch

| branchNo | address | assets |
|---|---|---|

Home

| custNo | branchNo | joined |
|---|---|---|

**Optimised Mapping**

Customer

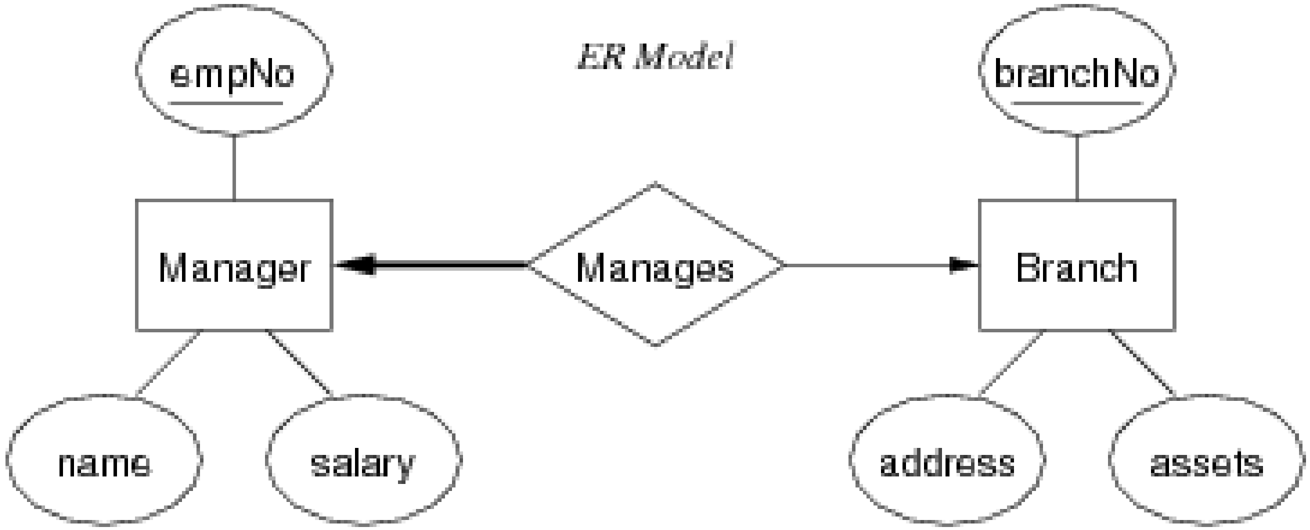
| custNo | name | address | branchNo | joined |
|---|---|---|---|---|

Branch

| branchNo | address | assets |
|---|---|---|

# (3c) Mapping 1:1 Relationships

## Example:

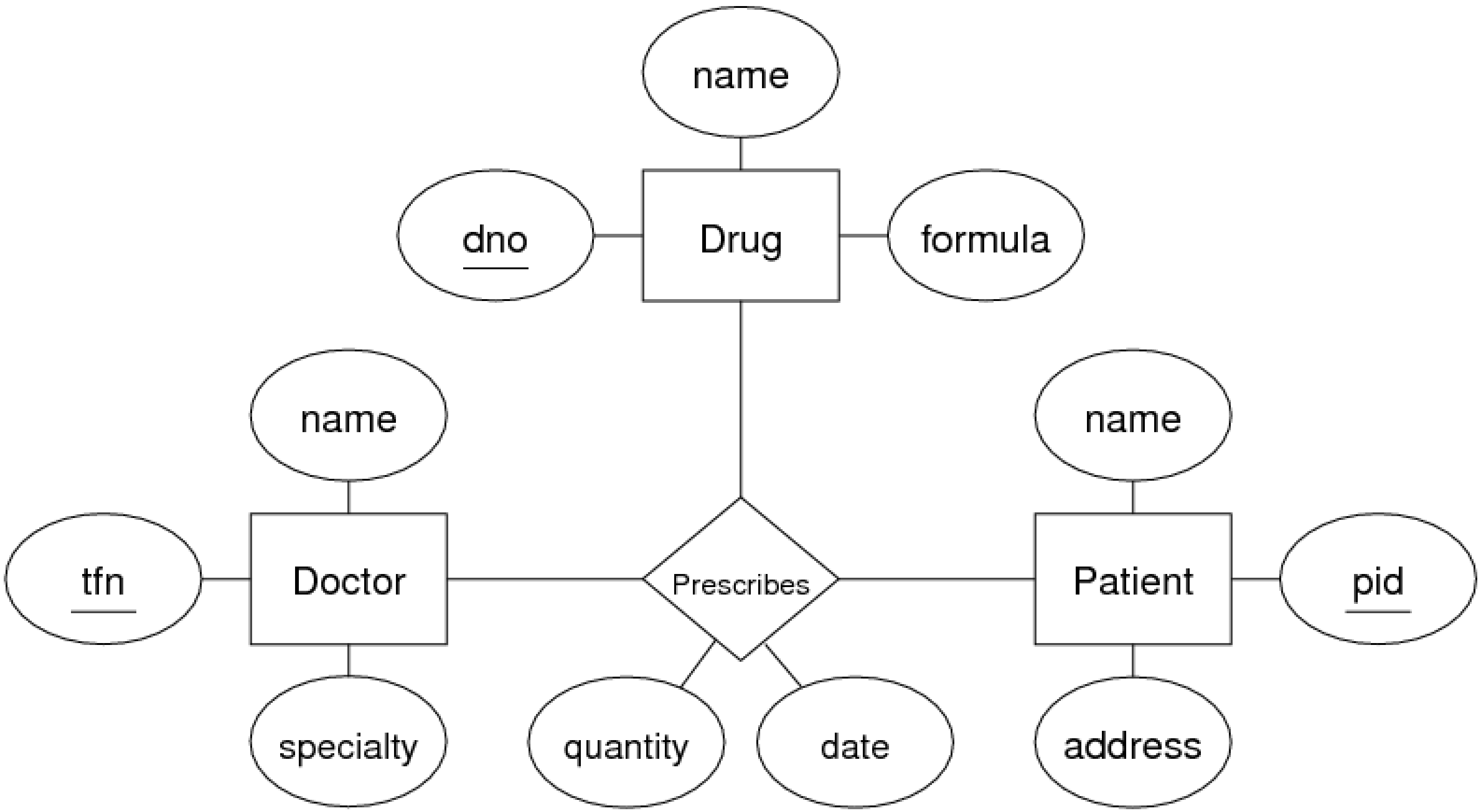



- Handled similarly to 1:N relationships
- For a 1:1 relationship between entity sets *E* and *F* (*S* and *T*):
  - choose one of *S* and *T* (e.g. *S*) (*Note : Choose the entity set that participates totally, if only one of them does*)
  - add the attributes of *T*'s primary key to *S* as foreign key
  - add the relationship attributes as attributes of *S*

# (3d) Mapping n-way relationships

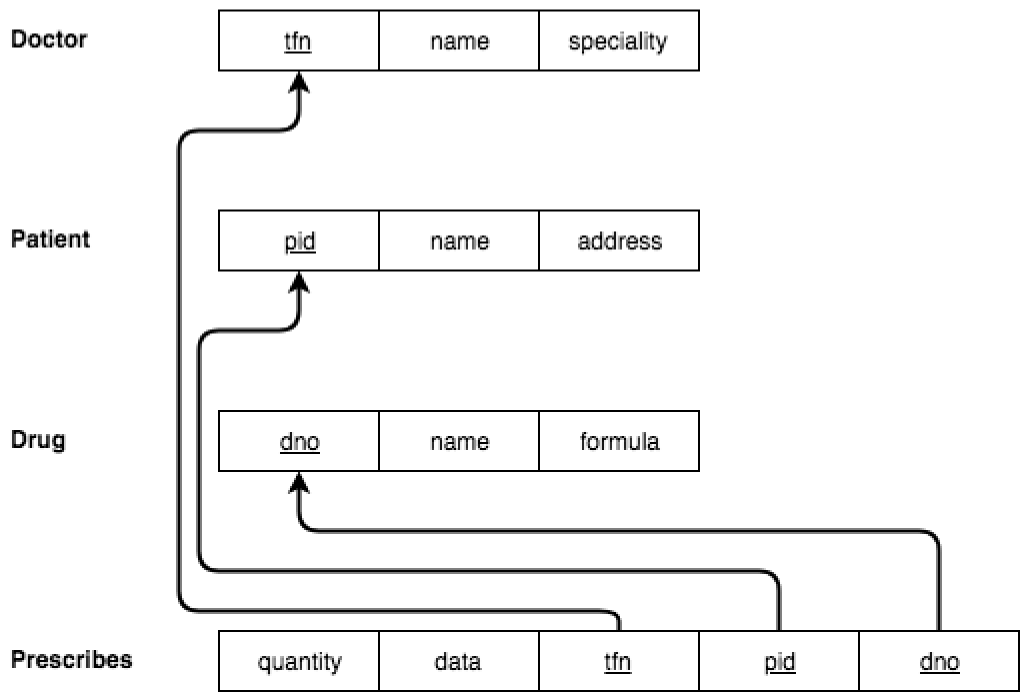
**Exercise:**

Convert the following ER design into a relational data model



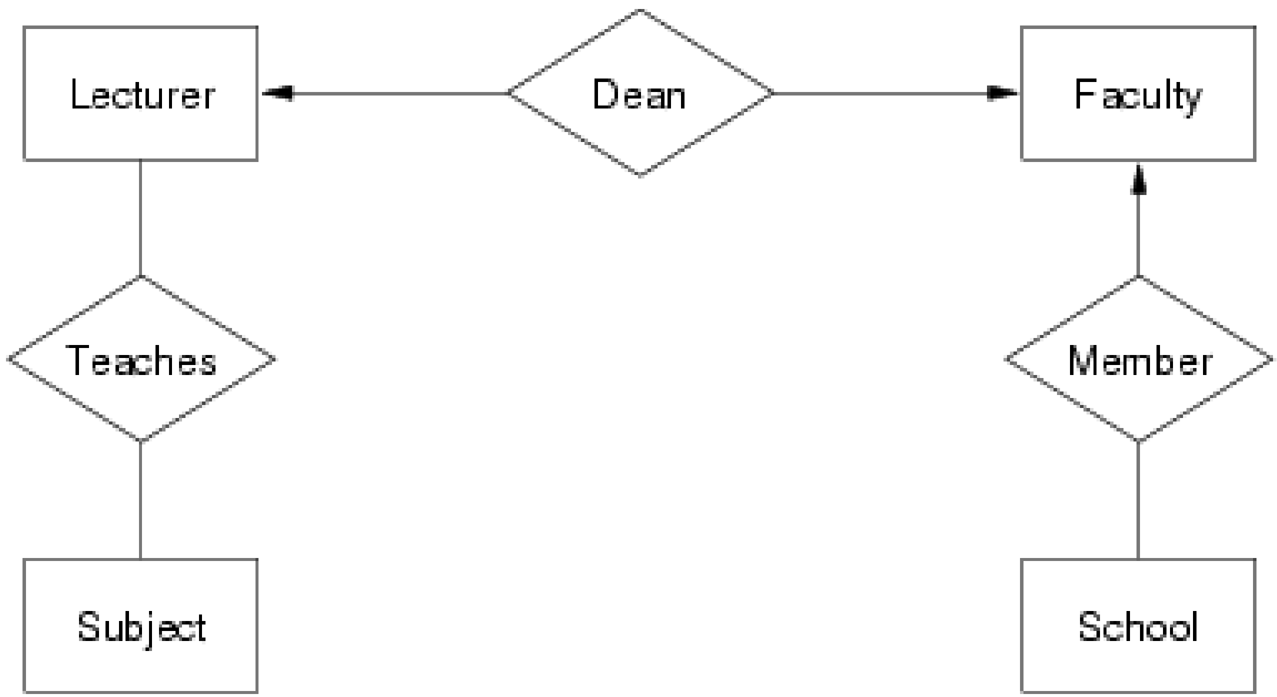
# (3d) Mapping n-way relationships

## Solution:



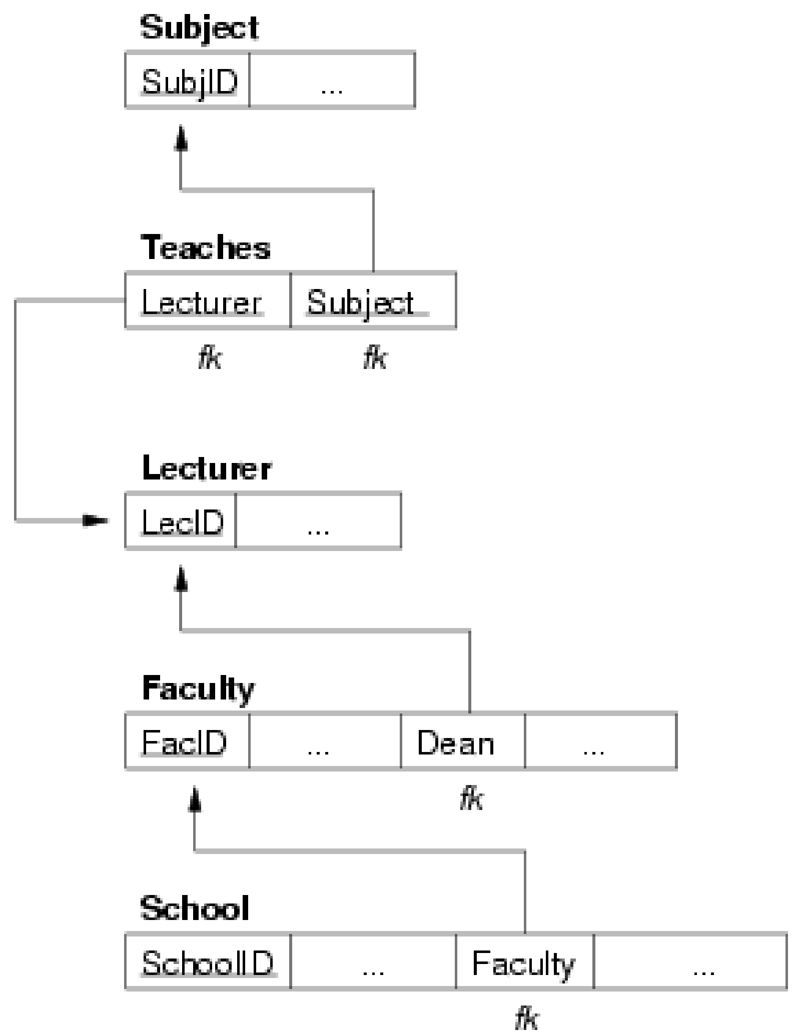
# **Exercise:**

## Convert the following ER design into a relational data model




*You can assume that each attribute contains (at least) a suitably-named attribute containing a unique identifying number (e.g. the Lecturer entity contains a LecID attribute).*

# Solution: Relational Model




Relational model for a very small University ER model

# So far …

## Summarising ER → Relational Mapping

### Mapping entities and attributes

- ER attribute → relational attribute
- ER entity → relational tuple
- ER entity-set → relational table (relation)
- ER key → relational primary key

### Mapping Relationships

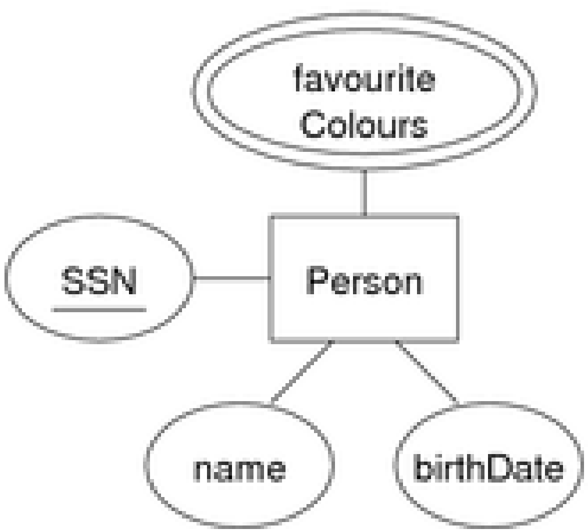
- ER relationship → create a new relational table (relation)
  - N:M relationship → add FK for each participating entity plus relationship attributes)
  - 1:N relationship → FK plus relationship attributes
  - 1:1 relationship → FK plus relationship attributes

# (4) Mapping multi-valued attributes

- treat like an N:M relationship between entities and values
- create a new relation where each tuple contains:
  - the primary key attributes from the entity
  - one value for the multi-valued attribute from the corresponding entity

Example:

*ER Model*




*Relational Version*

Person

| <u>SSN</u> | name | birthdate |
|---|---|---|

FavColour

| <u>SSN</u> | <u>colour</u> |
|---|---|

# (4) Mapping multi-valued attributes contd…

Example: the two entities

```
Person(12345, John, 12-feb-1990, [red,green,blue])
Person(54321, Jane, 25-dec-1990, [green,purple])
```

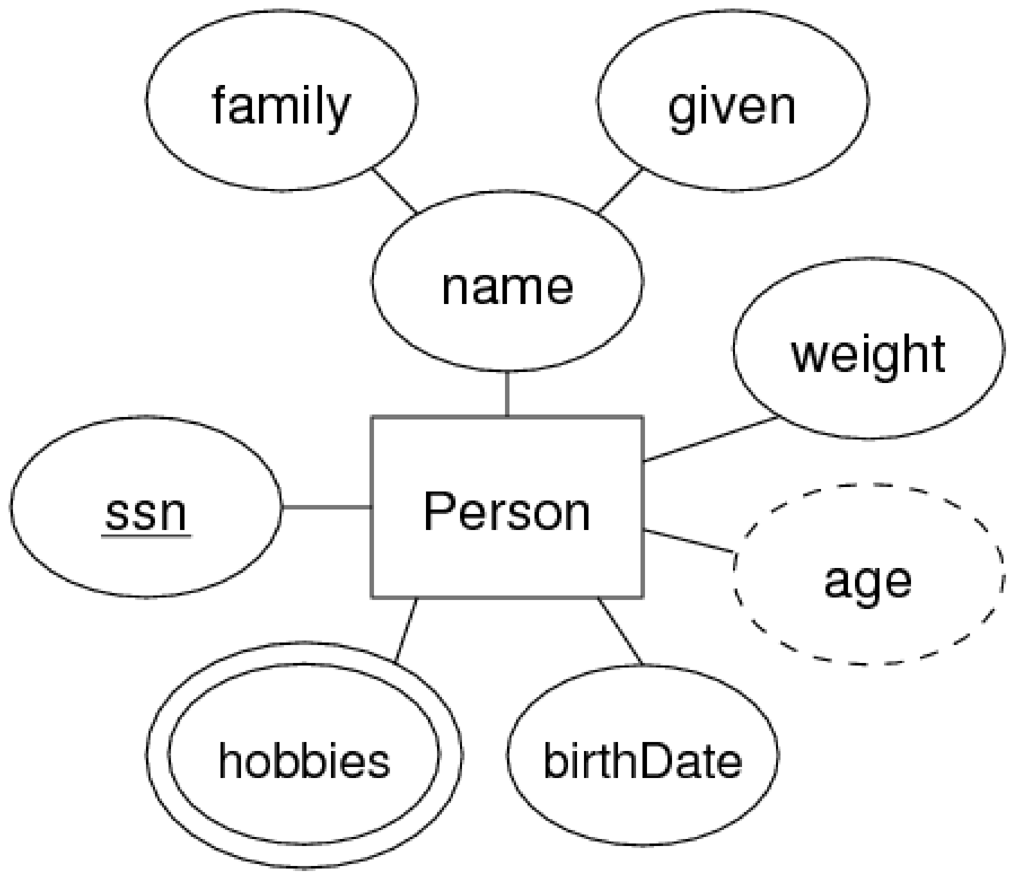
would be represented as:

```
Person(12345, John, 12-feb-1990)
Person(54321, Jane, 25-dec-1990)

FavColour (12345, red)
FavColour(12345, green)
FavColour(12345, blue)
FavColour(54321, green)
FavColour(54321, purple)
```

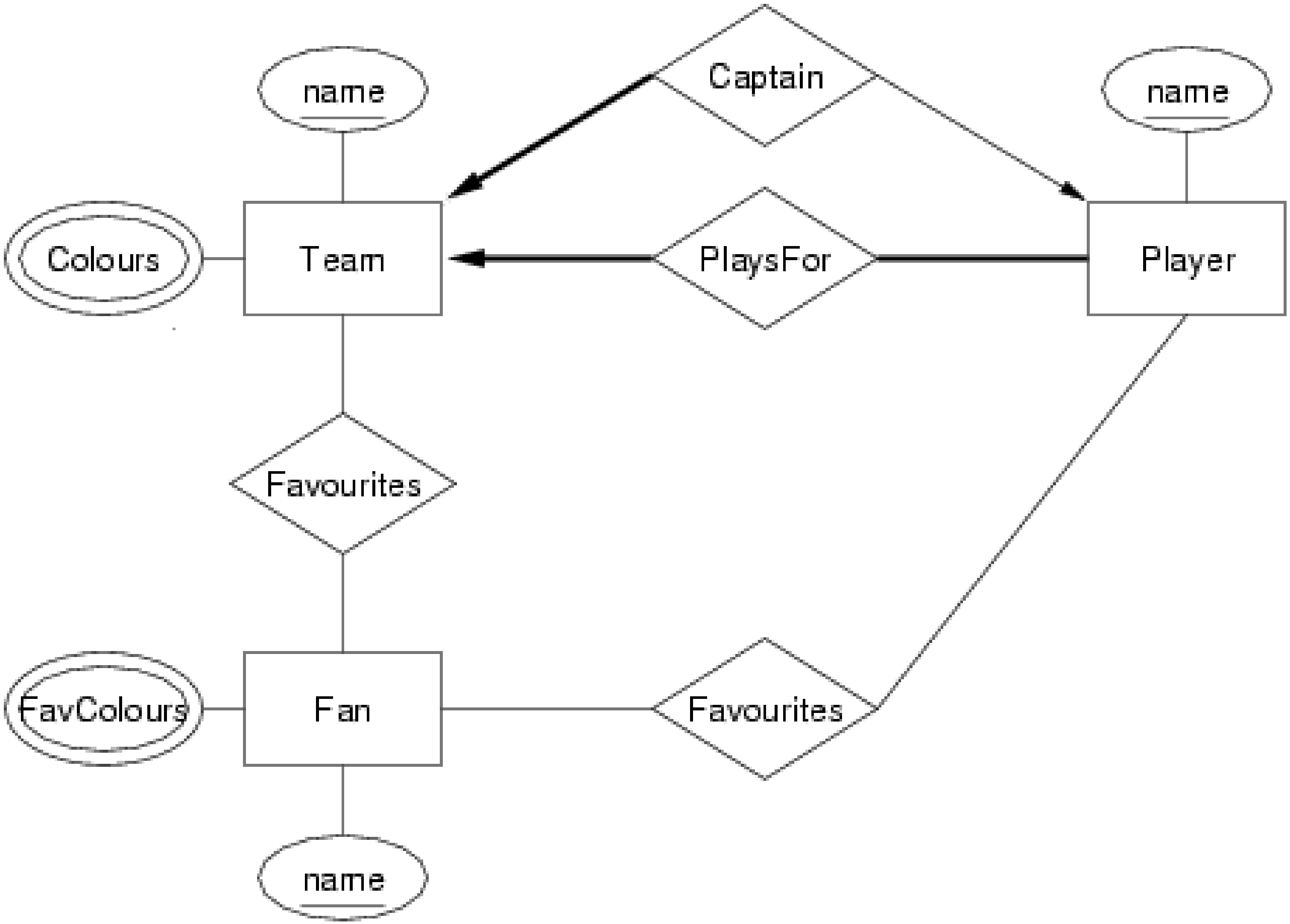
# Exercise:

## Convert the following ER design to relational form



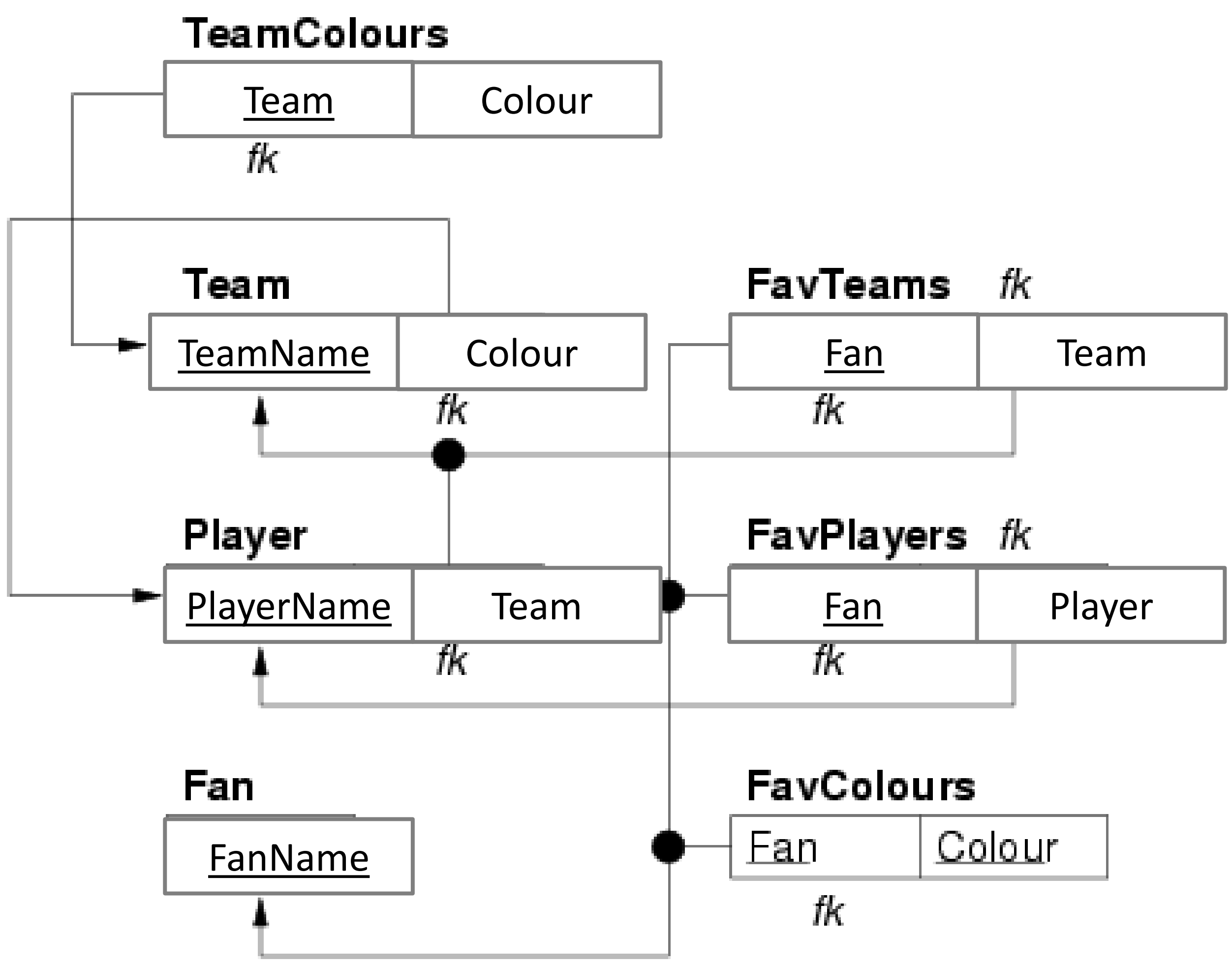
# Exercise:

1. Convert the following ER design into a relational data model based on the box schema notation
2. Which elements of the ER design do not appear in the relational model?



# Solution:

Convert the following ER design into a relational data model based on the box schema notation



# (5) Mapping sub classes

Three different approaches to mapping subclasses to tables:

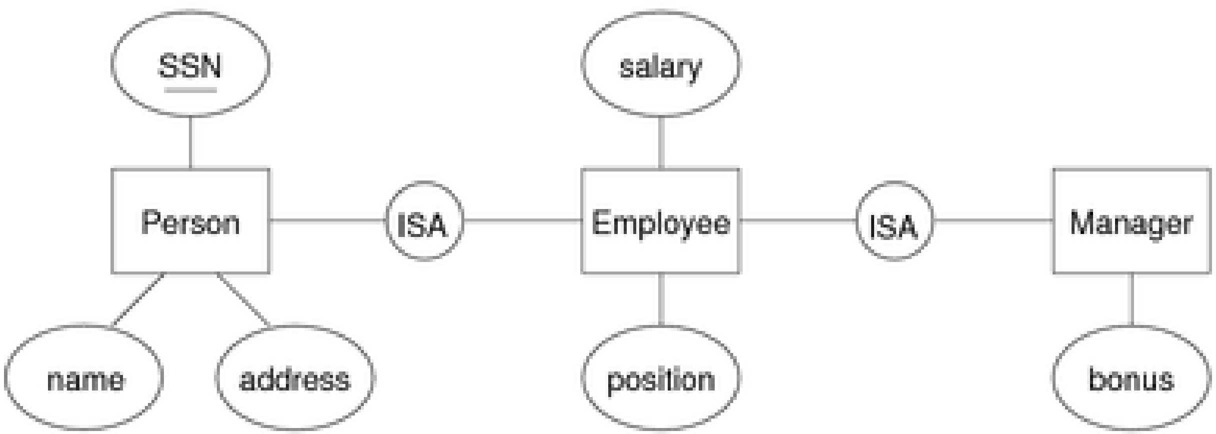
- ER style
  - each entity becomes a separate table,
  - containing attributes of subclass + FK to superclass table
- object-oriented
  - each entity becomes a separate table,
  - inheriting all attributes from all superclasses
- single table with nulls
  - whole class hierarchy becomes one table,
  - containing all attributes of all subclasses (null, if unused)

*Which mapping is best depends on how data is to be used*

# (5) Mapping sub classes in ER style

The subclass relation contains:

- all of the subclass-specific attributes
- uses the superclass primary key to capture the association




*Relational Model*

Person

| | | |
|---|---|---|
| SSN | name | address |

Employee

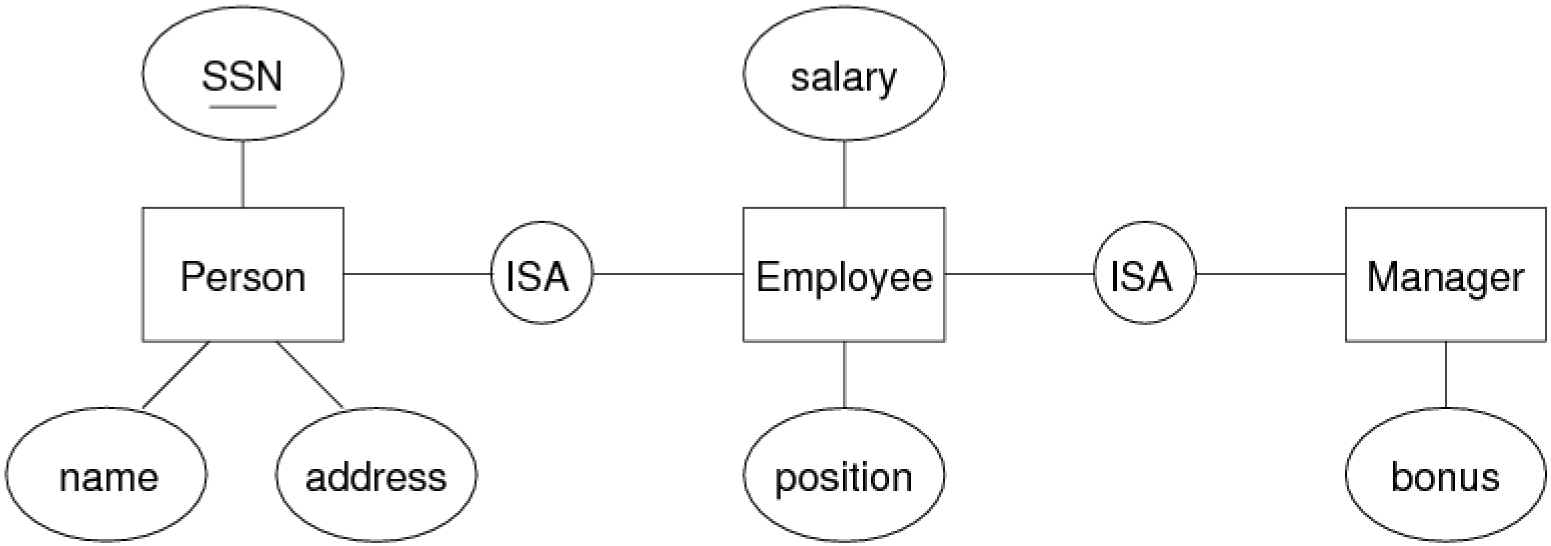
| | | |
|---|---|---|
| SSN | salary | position |

Manager

| | |
|---|---|
| SSN | bonus |

# (5) OO Mapping of sub classes

*Entity–Relationship Model*




*Relational Model*

Person

| | | |
|---|---|---|
| **<u>SSN</u>** | **name** | **address** |

Employee

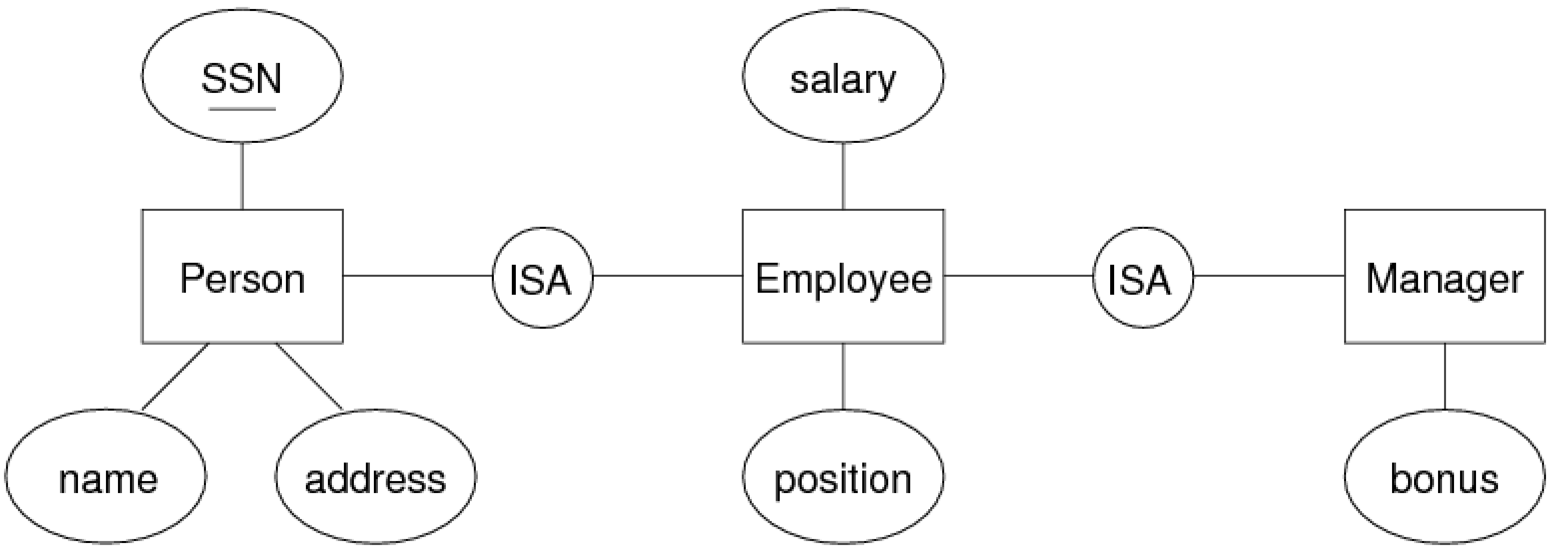
| | | | | |
|---|---|---|---|---|
| **<u>SSN</u>** | **name** | **address** | **salary** | **position** |

Manager

| | | | | | |
|---|---|---|---|---|---|
| **<u>SSN</u>** | **name** | **address** | **salary** | **position** | **bonus** |

# (5) Single table with nulls mapping

*Entity–Relationship Model*




*Relational Model*

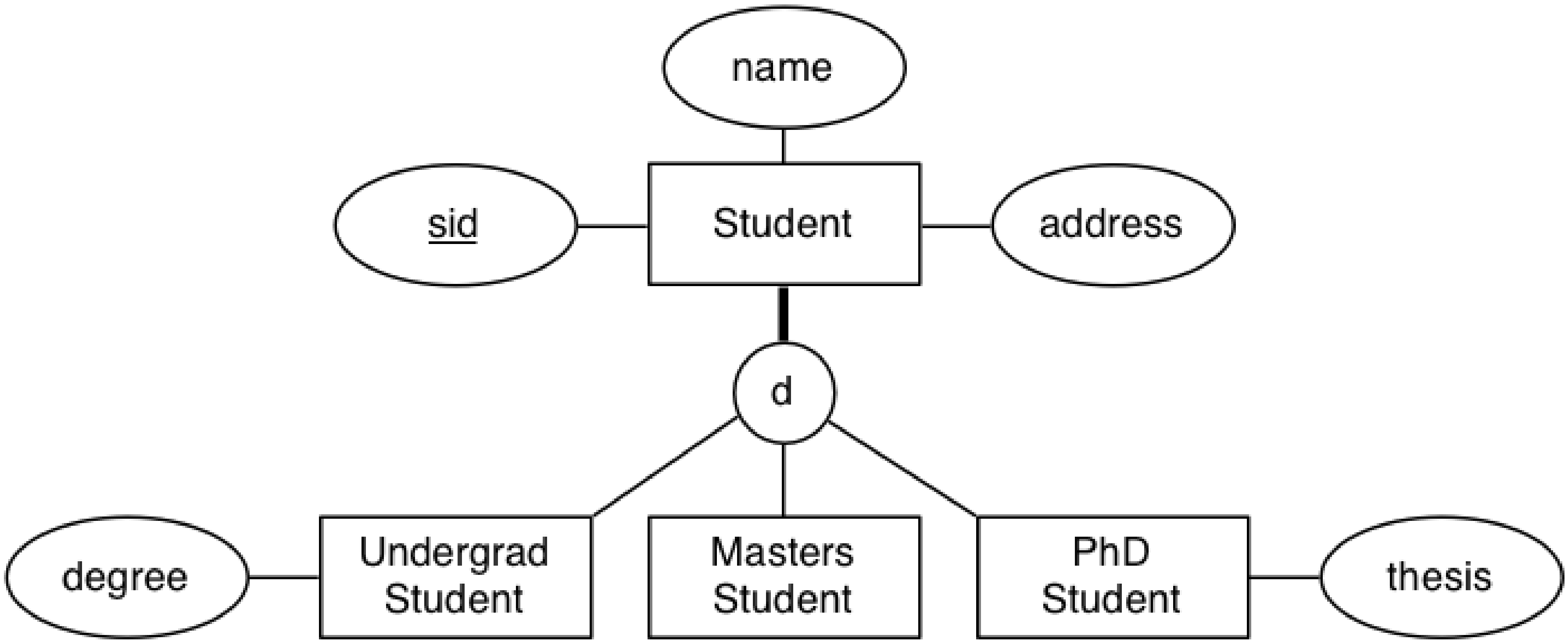
Person

| **SSN** | **name** | **address** | **salary** | **position** | **bonus** |
|---|---|---|---|---|---|

*NULL for Person who is not Employee*

*NULL for Employee who is not Manager*

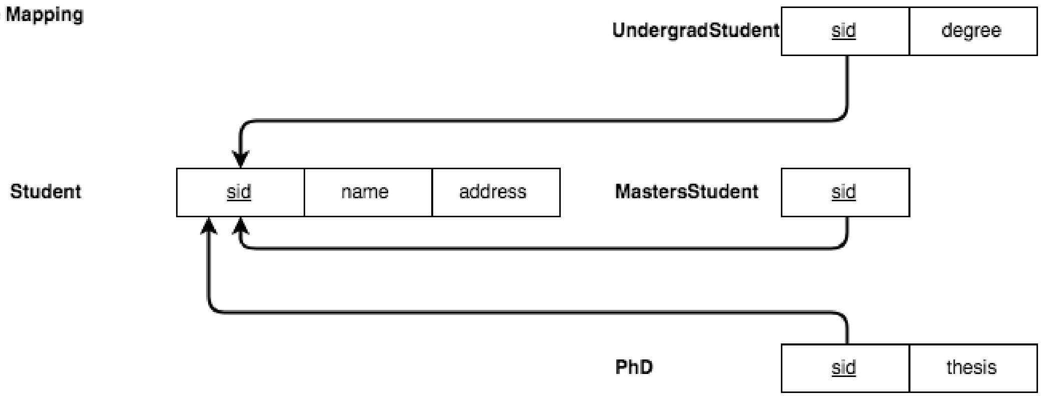
# (5) Exercise: Disjoint subclasses



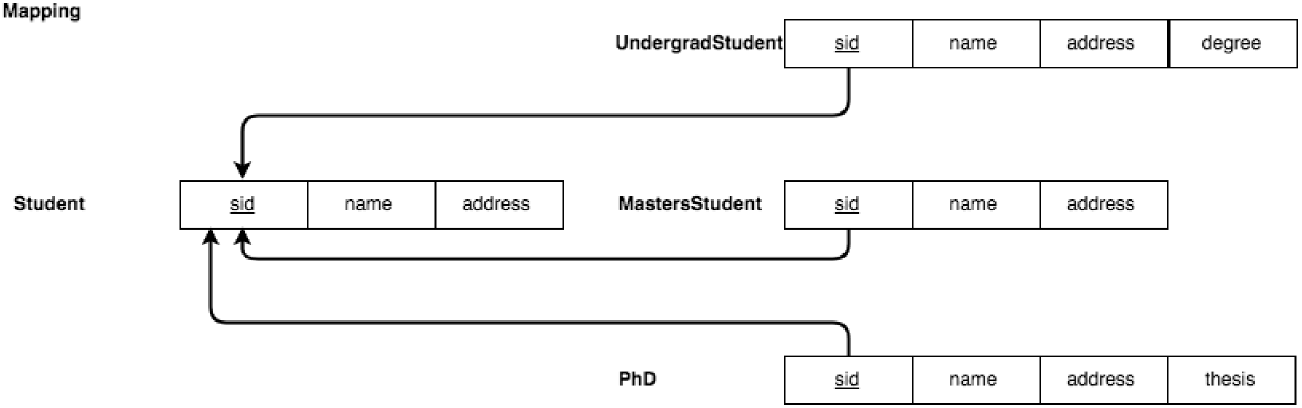

Use (a) ER-mapping, (b) OO-mapping, (c) 1-table-mapping

# (5) Solution: Disjoint subclasses

ER Style Mapping




OO Style Mapping

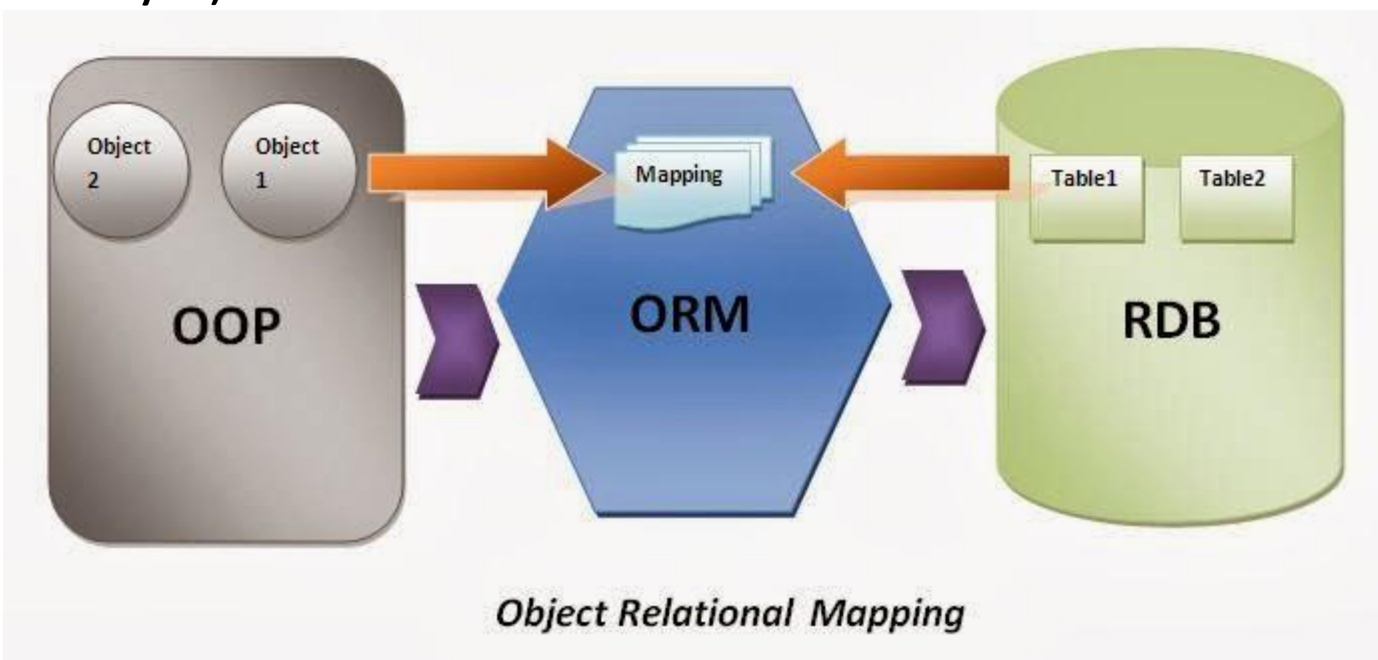



Single Table Style Mapping

| | | | | | | |
|---|---|---|---|---|---|---|
| **Student** | sid | name | address | degree | thesis | subclass |

# Introduction to Object Relational Mapper (ORMs)

- A high-level abstraction framework that maps a relational database system to objects
- Automates all the CRUD (create/retrieve/update/delete) operations
- ORM is agnostic to which relational database is used (well at least theoretically…)




- Many different ORM frameworks around e.g., Hibernate, TopLink, SQLAlchemy

# Why are ORMs useful?

- Shields the developer from having to write complex SQL statements and focus on the application logic using their choice of programming language
- Harmonisation of data types between the OO language and the SQL database
- Automates transfer of data stored in relational database tables into objects