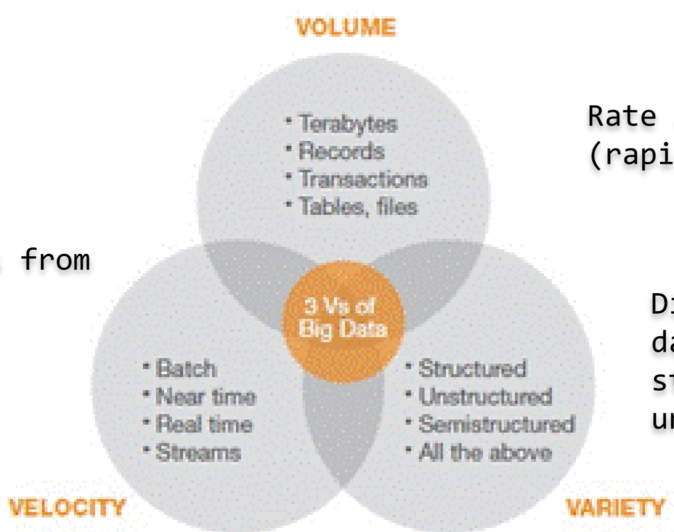
Databases
ER Model

COMP 1531
Aarthi Natarajan
Week 09

What is data?

- **Data** – facts that can be recorded and have implicit meaning (Elmasri & Navathe)
- Today data is being generated at an exponential rate
  - Financial market data, posts to social media sites, growing logs of web transactions, computation physics…BIG Data

Rate at which data flows from different sources



Rate at which data is generated (rapid, exponential growth)

Diversity in data types – structured data (financial market data), semi-structured (XML, web logs, email), unstructured (audio, video streams)

Why do we need a database ?

- Data by itself is not very useful.
- Give a context to data to transform data into information e.g., the numbers 45,55,67 do not mean much, but given a context such as these are the marks of students in COMP 1531, this is now information

  DATA -> INFORMATION -> DECISION

- This data needs to be:
  – Stored (in a structured format)
  – Manipulated (efficiently, usefully)
  – Shared (by very many users, concurrently)
  – Transmitted
- Red text handled by **databases**; green by networks.

Databases today…

- **Nearly every computer application uses a database**
    - Google, EBay, Amazon, iTunes Shop
    - Library catalogues, Train time tables, Airline bookings
    - Bank accounts, credit card, debit card
    - Medical records (Medicare), Tax Office
    - Facebook, Twitter, …
  - Every time you use a loyalty card, you're inputting information about your buying habits into the database of the company you are buying from
- **Challenges in building effective databases**
  - efficiency, security, scalability, maintainability, availability, integration, new media types (e.g. music), ...

What is a database ?

- A **database** represents a logically coherent collection of related data
- A **database management system (DBMS)** is a software application that allows users to:
  - create and maintain a database (DDL)
  - define queries to retrieve data from the database
  - perform transactions that cause some data to be written or deleted from the database (DML)
  - provides concurrency, integrity, security to the database
- A database and DBMS are collectively referred to as a **database system**

Data Models

A data model describes how the data is structured in the database

There are several types of data models

- Relational model
  - a data structure where data is stored as a set of records known as tables
  - each table consists of rows of information (also called a tuple)
  - each row contains fields known as columns

StudentId	FirstName	LastName
213899	Joe	Bloggs
321456	Sam	Hunt
456789	John	Smith

- Document model
  - data is stored in a hierarchical fashion e.g., XML
- Object-oriented model
  - a data structure where data is stored as a collection of objects
- Object-Relational model
  - a hybrid model that combines the relational and the object-oriented database models

More database terminology

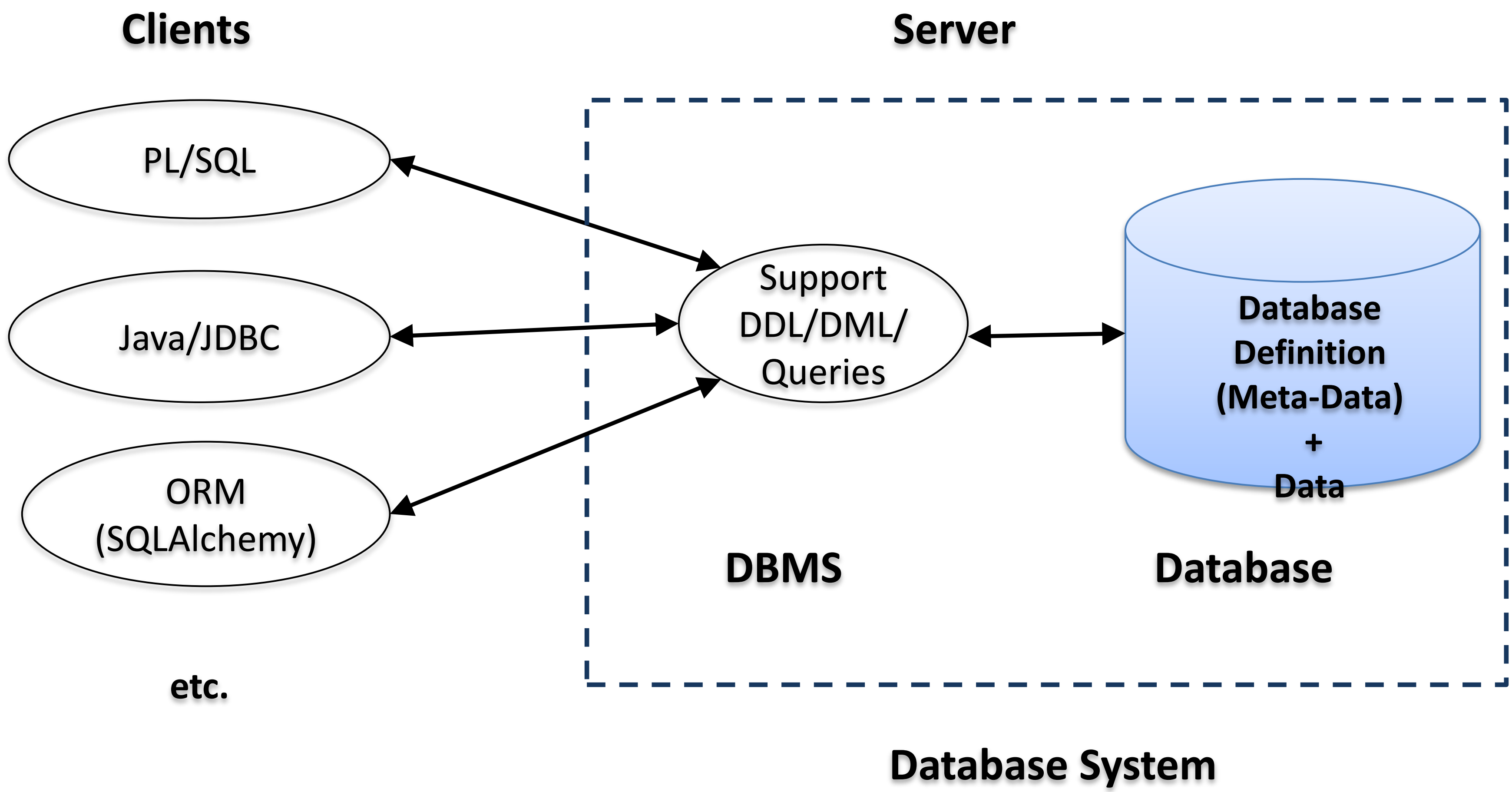
- A database schema adheres to a data model and provides a logical view of the database, defining how the data is organised and the relationships between them and is typically set up at the beginning
- A database schema instance is the state of the database at a particular instance of time

Relational Database System

- A Relational Database Management System (RDBMS) is a DBMS that:
  - is based on a relational data model i.e., stores data as *tuples* or *records* in *tables*
  - Allows the user to create *relationships* between tables
- Examples of relational database systems
  - Open Source
    - PostgreSQL, MySQL, SQLite
  - Commercial
    - Oracle, DB2 (IBM), MS SQL Server, Sybase

Database System Architecture

Typical environment for a modern database system



SQL Queries and results travel along the client <->server links

Data Modelling for Databases

Database design

Typical steps in a database design

1. requirements analysis (identify data and operations)
2. **data modelling** (high-level, abstract)

   an important early stage of database application development (aka "database engineering")
3. database schema design (detailed, relational model/tables)
4. database implementation (create instance of schema)
5. build operations/interface (SQL, stored procedures, GUI)
6. performance tuning (physical re-design)
7. schema evolution (logical schema re-design)

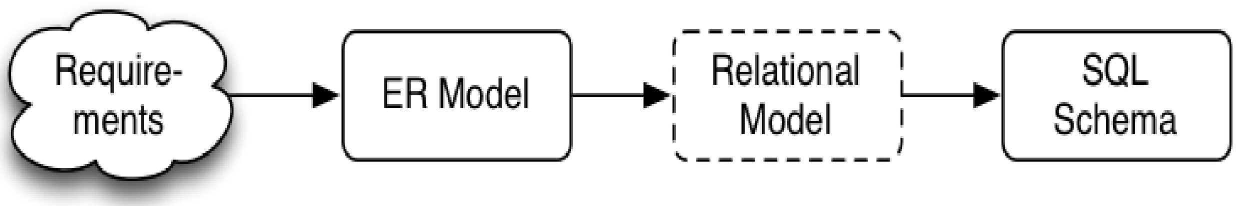
Data Modelling

Data modelling, in general consists of building:

- Logical models: abstract model e.g., ER Model, OO Model
- Physical models: record-based models e.g., relational model, classes which deal with the physical layout of data in storage

A data-modelling strategy for designing a database

- Design using abstract model (conceptual-level modelling)
  - i.e., perform initial conceptual modelling with entity relationship (ER) models
- Map to physical model (implementation-level modelling)
  - Transform conceptual ER design into relational model



Data Modelling for Databases

Aims of Data Modelling:

- describe what data is contained in the database
  (e.g. entities: students, courses, accounts, branches, patients, ...)
- describe relationships between data items
  (e.g. John is enrolled in COMP3311, Paul's account is held at Coogee)
- describe constraints on data
  (e.g. 7-digit IDs, students can enrol in no more than 30UC per semester)

Data modelling is a design process

- converts requirements into a data model

Some Design Ideas

Consider the following during design:

- start simple ... evolve design as problem better understood
- identify objects (and their properties), then relationships
- most designs involve kinds (classes) of people
- keywords in requirements suggest data/relationships (rule-of-thumb: nouns → data, verbs → relationships)
- don't confuse operations with relationships (operation: he **buys** a book; relationship: the book **is owned** by him)
- consider all possible data, not just what's available

Entity Relationship Diagrams

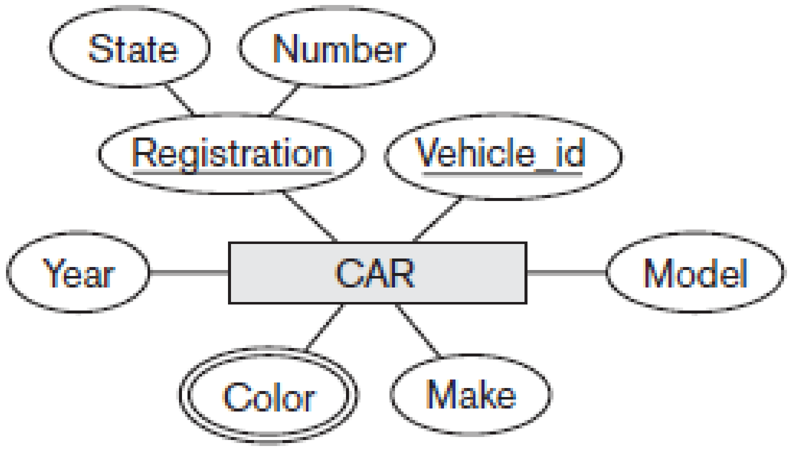
Entity-Relationship Conceptual Data Modelling

The world is viewed as a collection of inter-related entities.

ER modelling uses three major modelling constructs:

- **entity**:
  - a thing or object of interest in the real-world and is distinguishable from other objects
- **attribute**:
  - a data item or property of interest describing the entity *e.g., Joe (entity) described by name, address, age (attributes)*
- An **entity-set** (aka: entity-type) can be viewed as either:
  - a set of entities with the same set of attributes
  - an abstract description of a class of entities e.g., students, courses, accounts

e.g.,



An ER diagram showing an entity-set CAR with two key attributes (registration and vehicle_id), three single-valued attributes (year, model, make) and a multi-valued attribute (color)

CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR_1
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

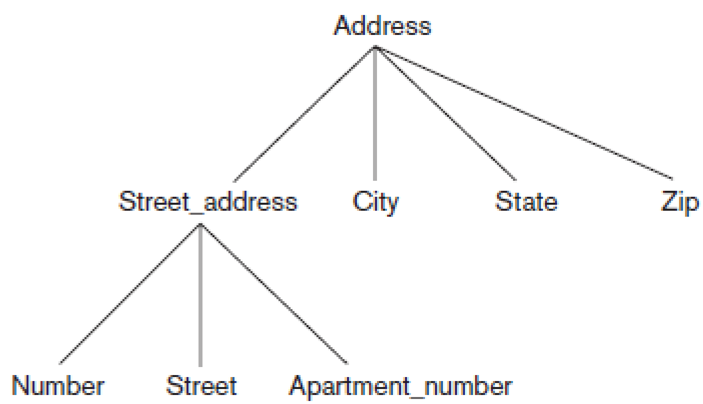
⋮

An entity set CAR with three entities

Attributes of an entity-set

In contrast to relational model, attributes in an ER model can be:

- Simple (attribute cannot be broken into smaller sub-parts)
  - e.g., `age` attribute for entity type Employee
- Composite (have a hierarchy of attributes)
  - e.g., entity type EMPLOYEE has a composite attribute `Address`



- Single-valued (have only one value for each entity)
  - e.g., an `vin_chassis` attribute for an entity type CAR
- Multi-valued (have a set of values for each entity)
  - e.g., a `Colors` attribute for CAR = (blue,black)

What if two entities have the same set of attribute values?

- They're regarded as the same entity.
- So, each entity must have a distinct set of attribute values.

  One approach:

  Define a key (super-key) : It is any set of attributes
  - whose set of values are distinct over entity set
  - natural (e.g. name + address + birthday) or artificial (e.g. SSN)
- Candidate key = any super-key such that **no subset** is also a superkey)

  e.g. (name + address) is a super-key, but not (name) or (address)
- Primary key = a candidate key designated by DB designer that uniquely identifies an entity e.g., SSN

Example (bank customer entities)

Customer = (custNo, name, address, taxFileNo)

- Definite super-keys:
  - any set of attributes involving custNo or taxFileNo
- Possible super-keys:
  - (name, address)
- Unlikely super-keys:
  - (name), (address)

Relationship Sets

Relationship: relates two or more entities, e.g.,

- Joe Smith (a STUDENT entity) ENROLLED IN (relationship) COMP1531 (a COURSE entity)
- Chris (an EMPLOYEE entity) WORKS FOR (relationship) ORACLE (a COMPANY entity)

Relationship Set (aka relationship type) : set of similar relationships, associating entities belonging to one entity-set to another

- **degree** = the number of entities involved in the relationship (in ER model, ≥ 2) e.g, the degree of WORKS FOR is 2
- **cardinality** = # associated entities on each side of relationship e.g., the cardinality of WORKS FOR is N:1

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)

Entity Relationship Diagrams

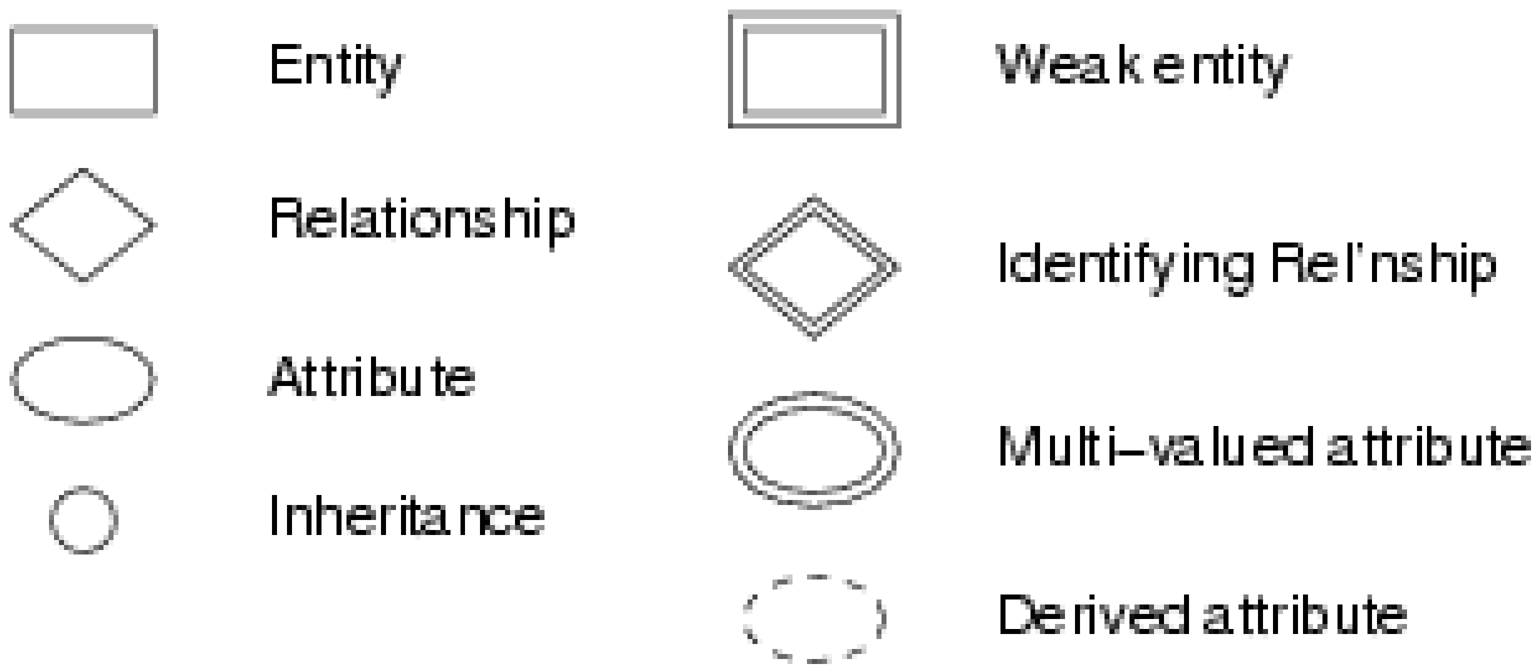
- ER diagrams are a graphical tool for data modelling
- An ER diagram consists of:
  - a collection of *entity set* definitions
  - a collection of *relationship set* definitions
  - *attributes* associated with entity and relationship sets
  - connections between entity and relationship sets

Warning: 99% of the time ...

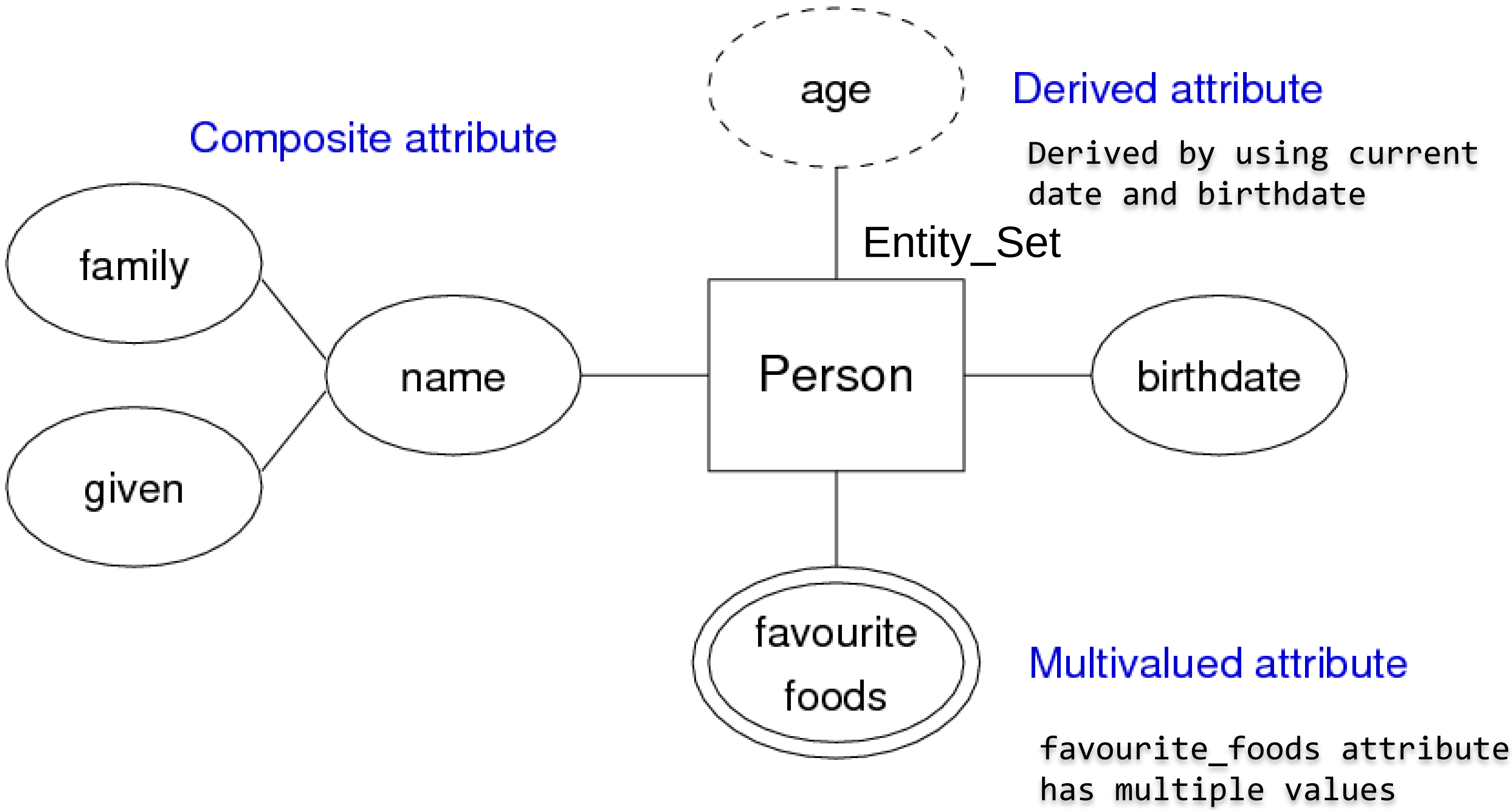
- we say "entity" when we mean "entity set"
- we say "relationship" when we mean "relationship set"
- If we want to refer to a specific entity, we generally say "entity instance"

Entity Relationship Diagrams

Specific visual symbols indicate different ER design elements:

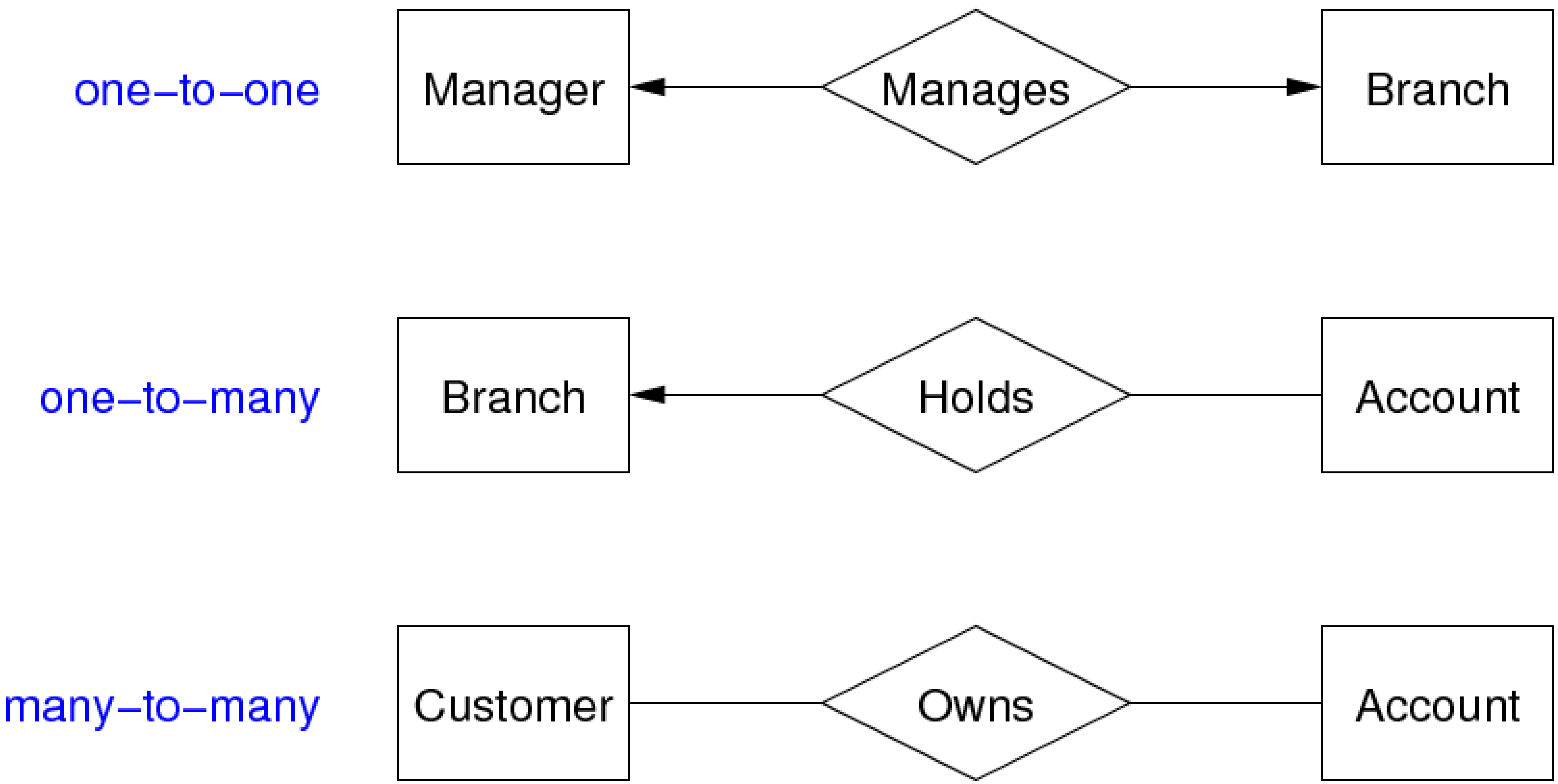


Example of attribute notations



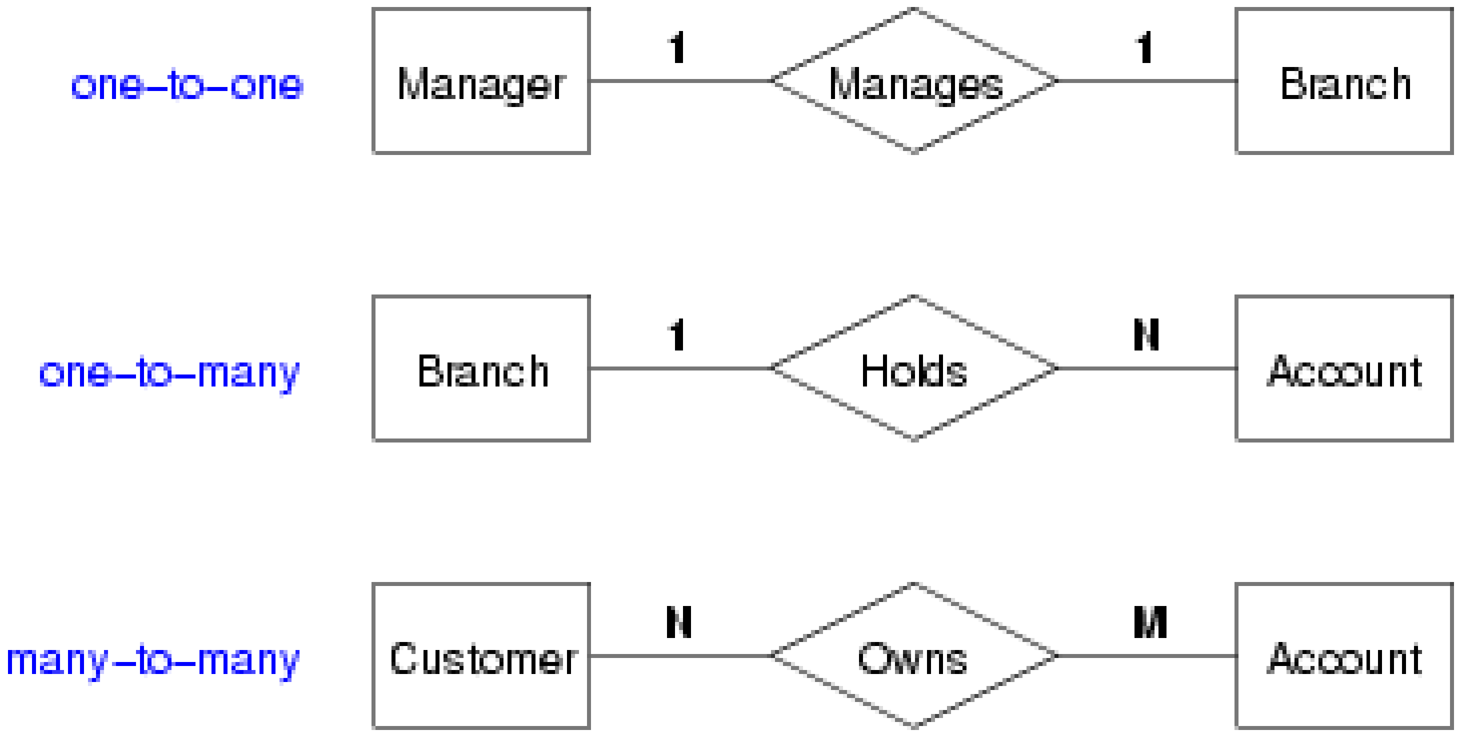
Cardinality in Relationship Sets

Examples:



An alternative explicit notation

Examples:



Relationship Sets in ER diagrams

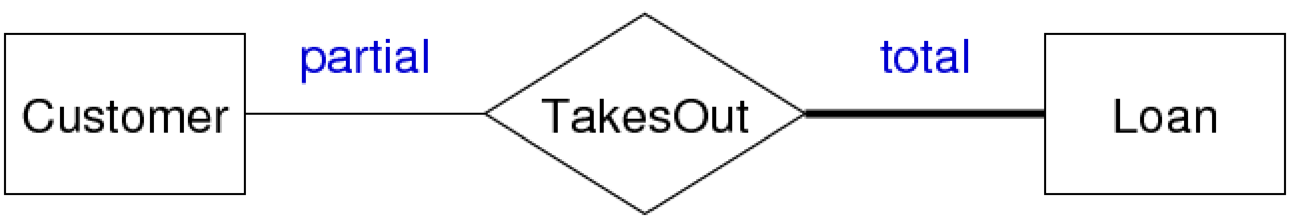
Level of participation constraint = a type of relationship constraint defined as:

Participation in relationship set *R* by entity set A may be:

- total - every $a \in A$ participates in ≥ 1 relationship in *R*
- partial - only some $a \in A$ participate in relationships in *R*

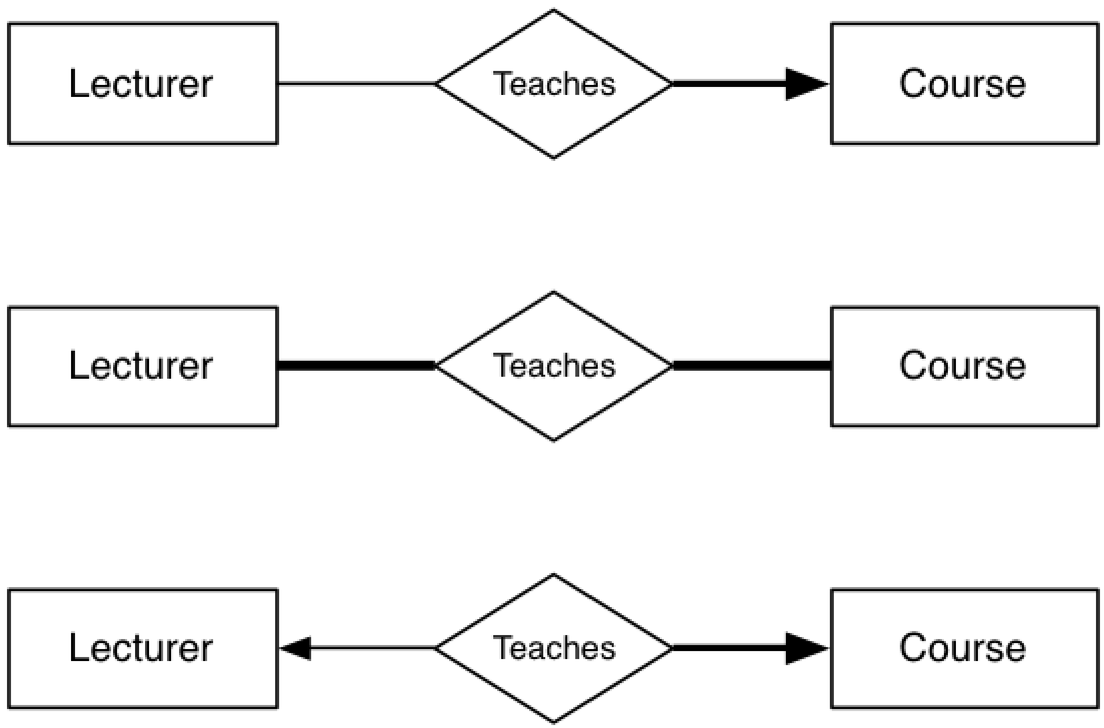
Example:

- every bank loan is associated with at least one customer
- not every customer in a bank has a loan



Exercise 1: Relationship Semantics

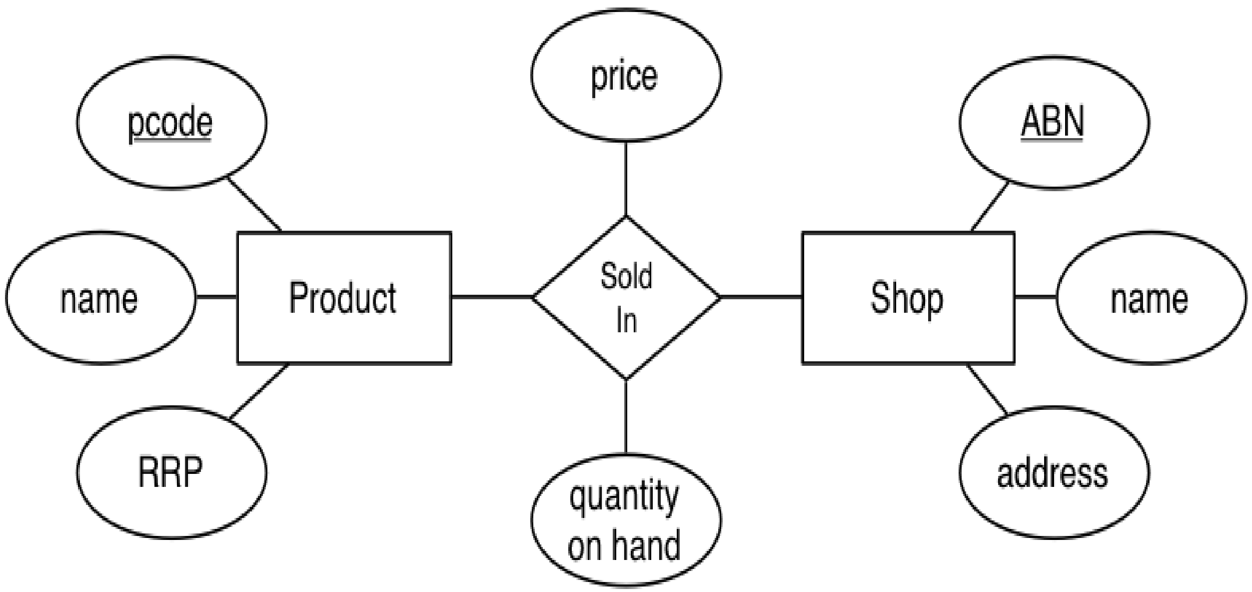
Describe precisely the scenarios implied by the following relationships:



Relationship Type with attributes

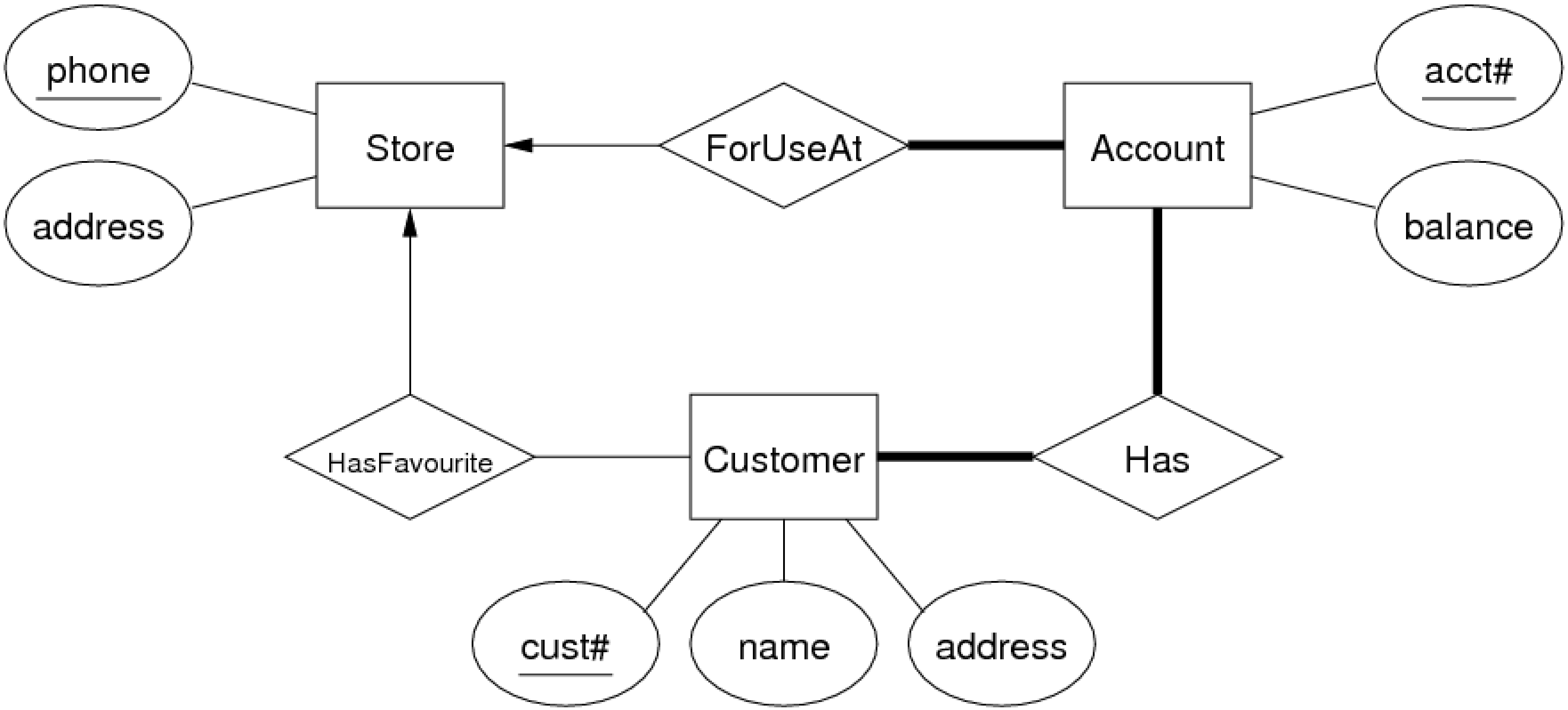
In some cases, a relationship needs associated attributes

Example:



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

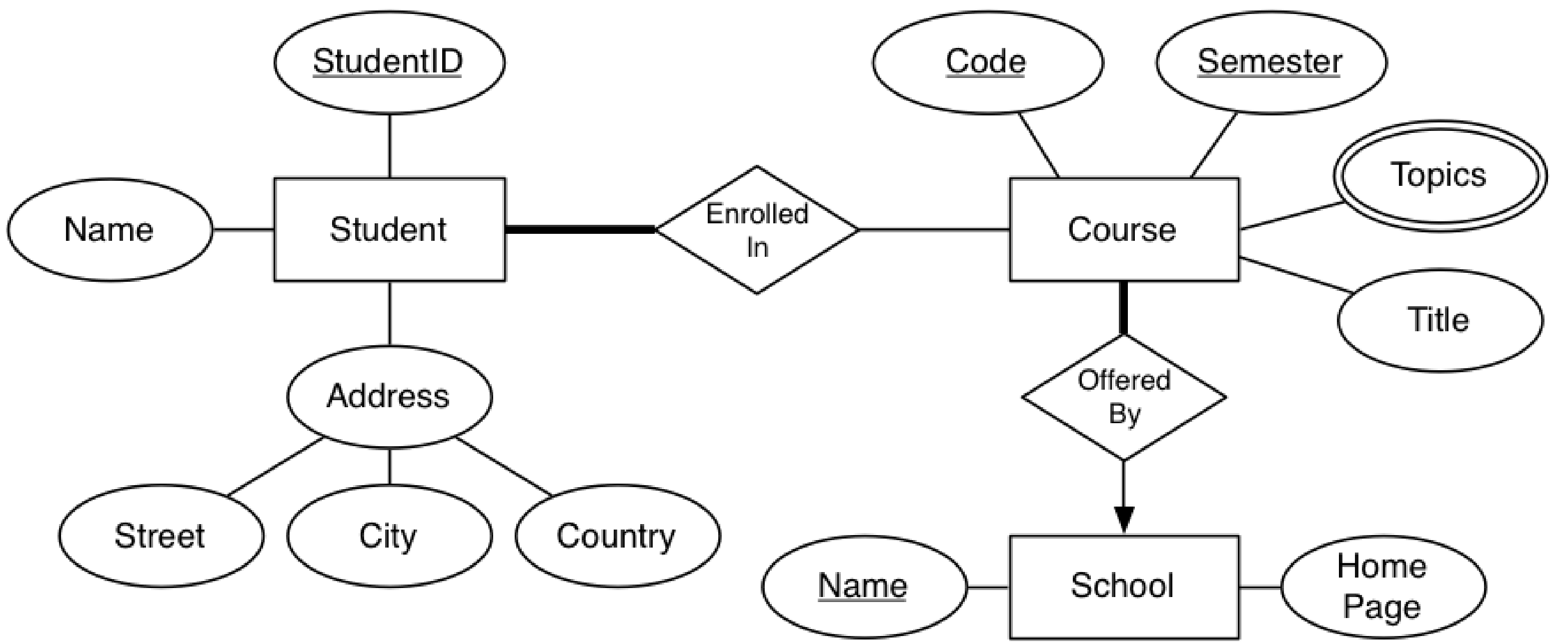
Putting it all together…
Example1: - a complete ER Diagram



primary key attributes are underlined e.g. cust#

Example 2:

Entities, relationships, attributes, keys, cardinality, participation, ...

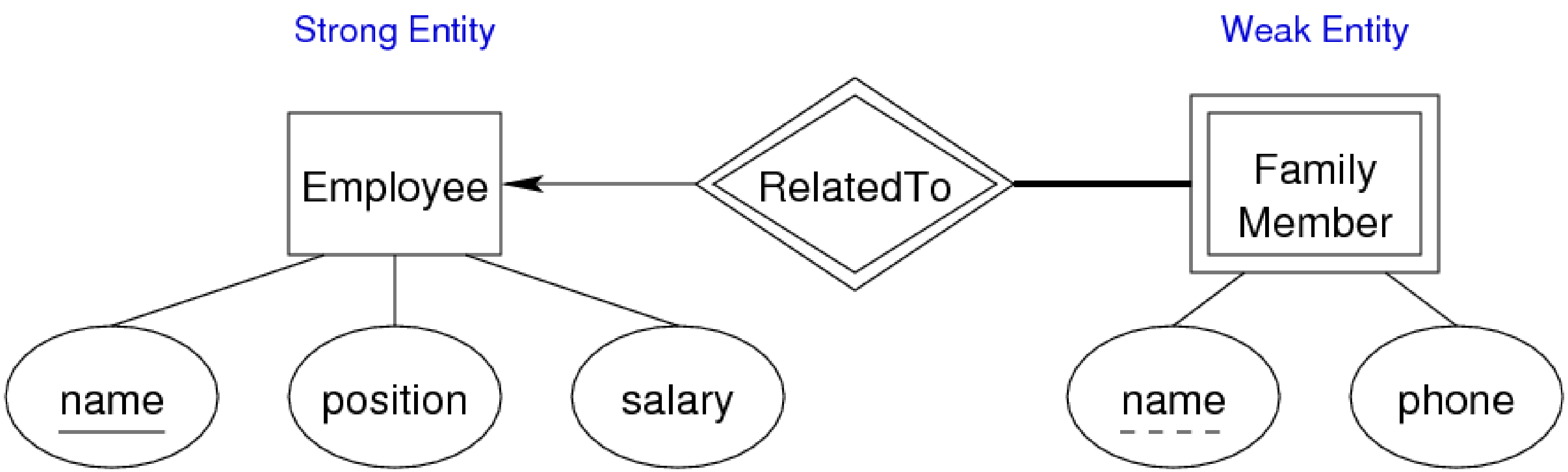


Weak Entity Set

A Weak entity set

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

Example:



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

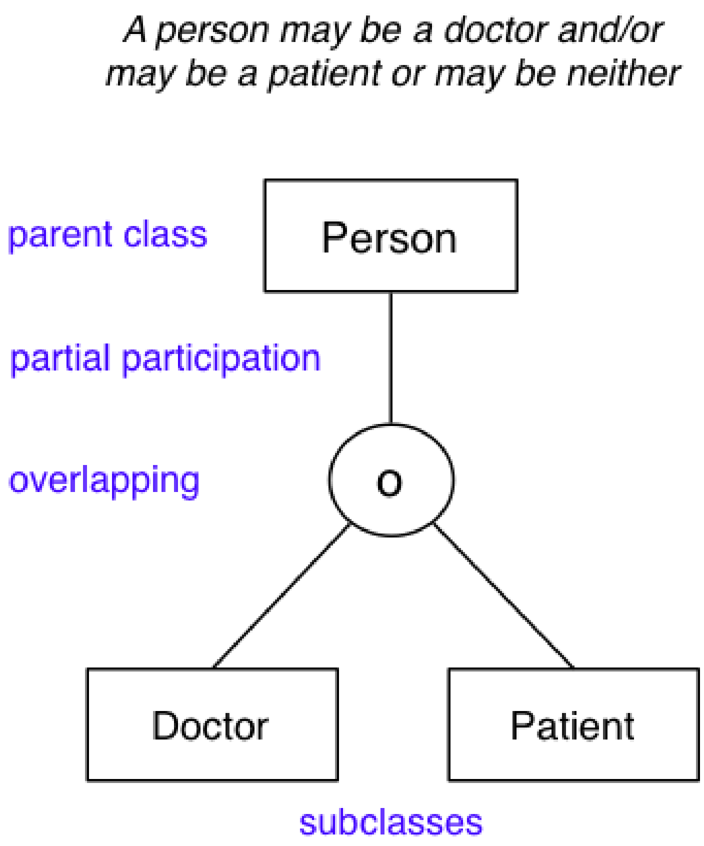
Properties of subclasses:

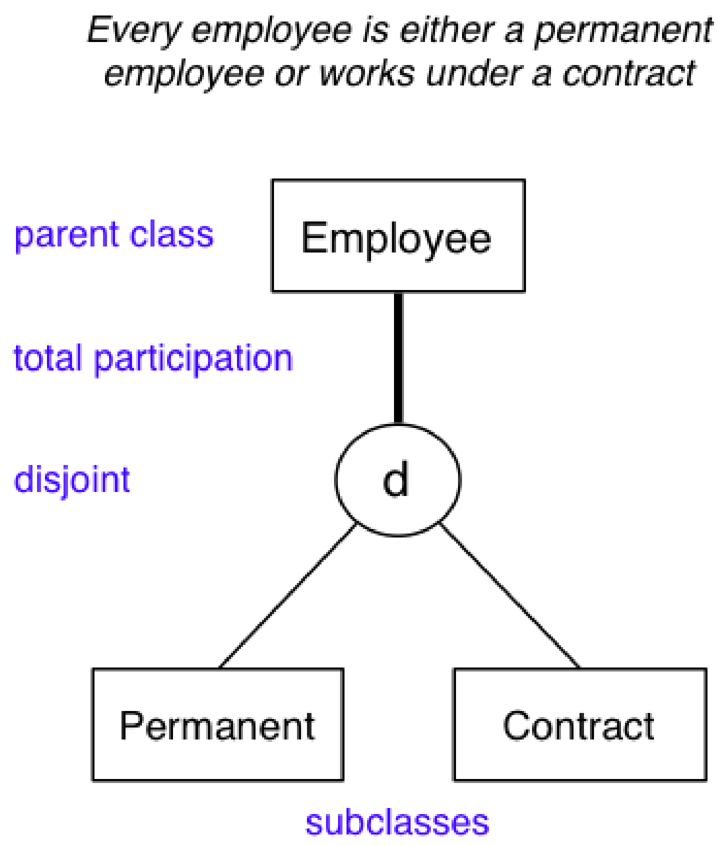
- overlapping or disjoint (can an entity be in multiple subclasses?)
- total or partial (does every entity have to also be in a subclass?)

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

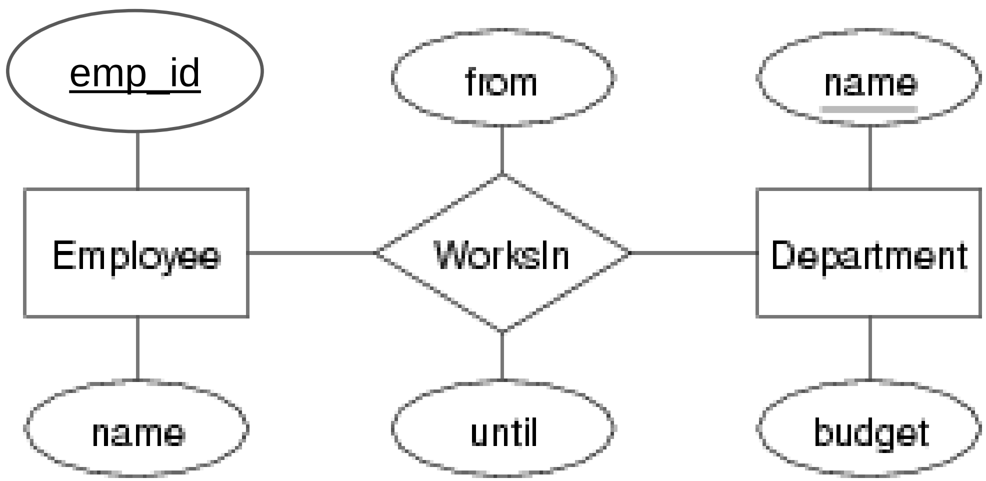
Develop an ER design for the following scenario:

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 …

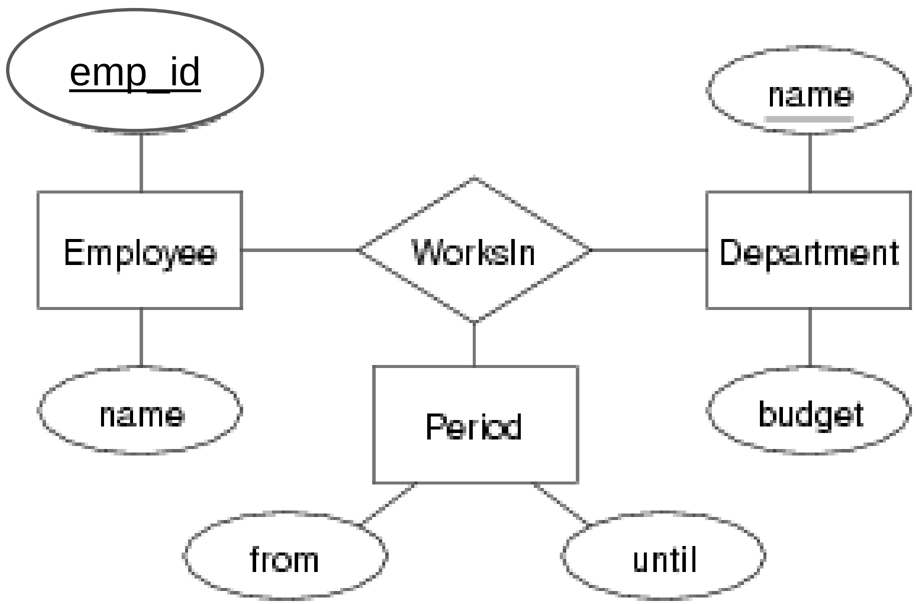
Attribute vs Entity Example (v1)



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

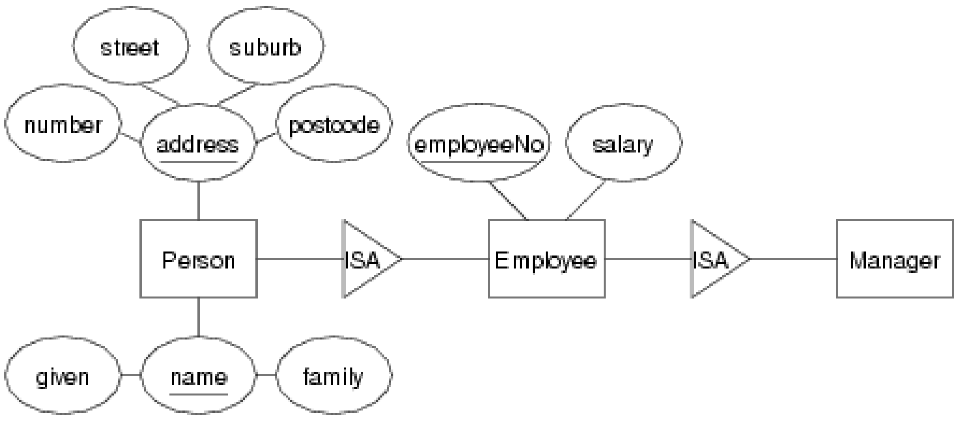
ER diagrams are typically too large to fit on a single screen. (or a single sheet of paper, if printing)

One commonly used strategy:

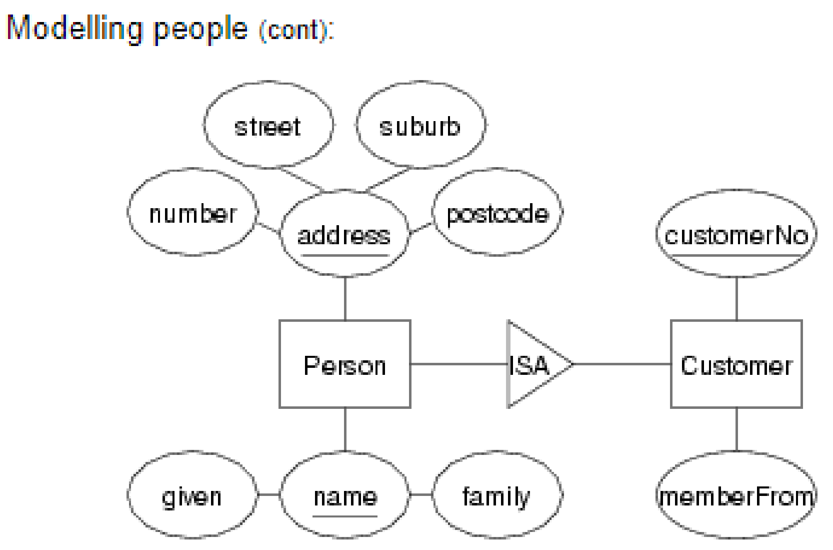
- define entity sets separately, showing attributes
- combine entities and relationships on a single diagram (but without showing entity attributes)
- if very large design, may use several linked diagrams as seen in the example in the next three set of slides

e.g. an ER model for a Bank

(1) Modelling people (employees)

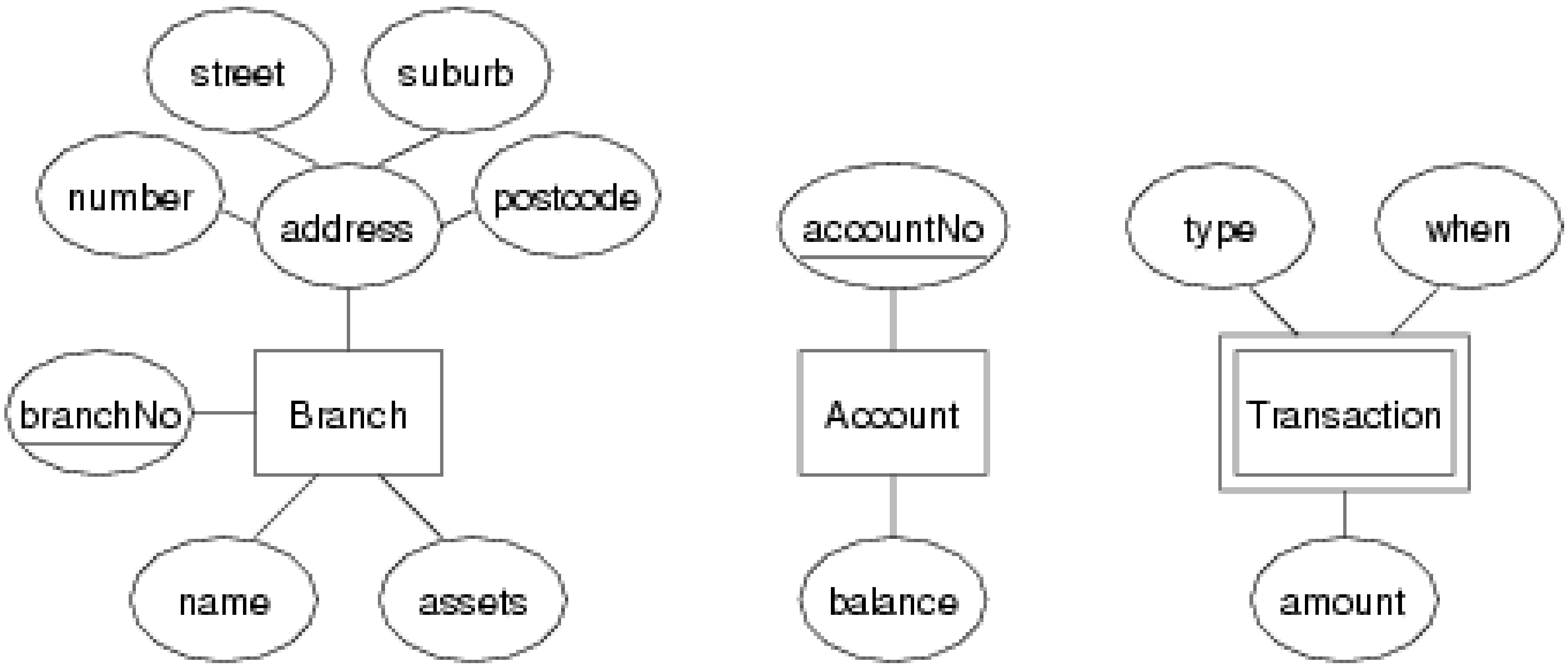


(2) Modelling people (customer)



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