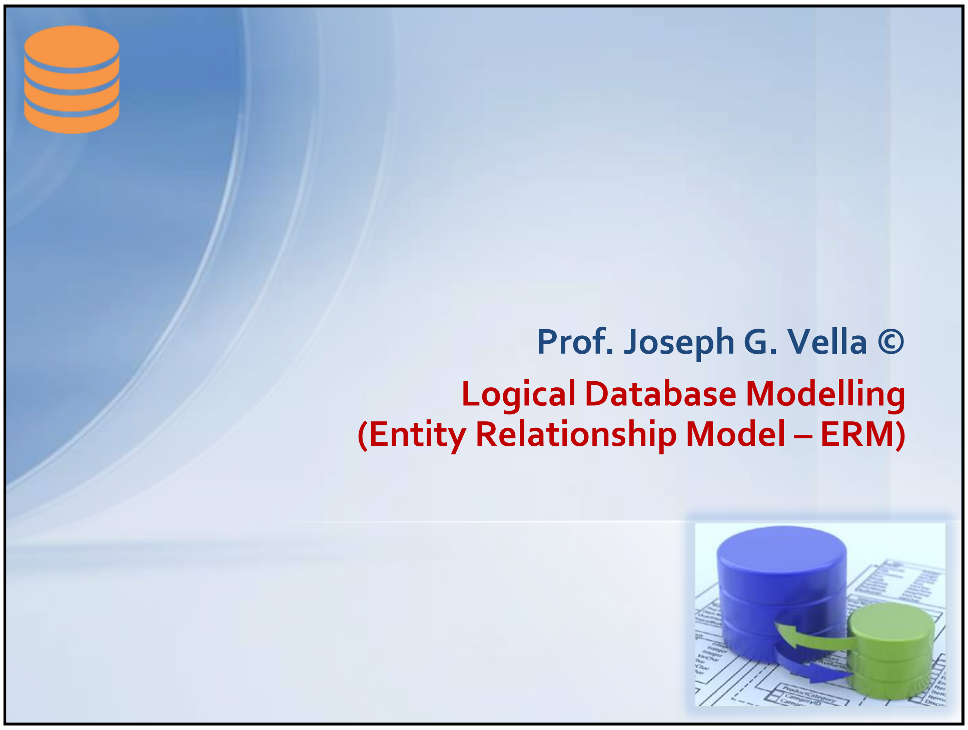
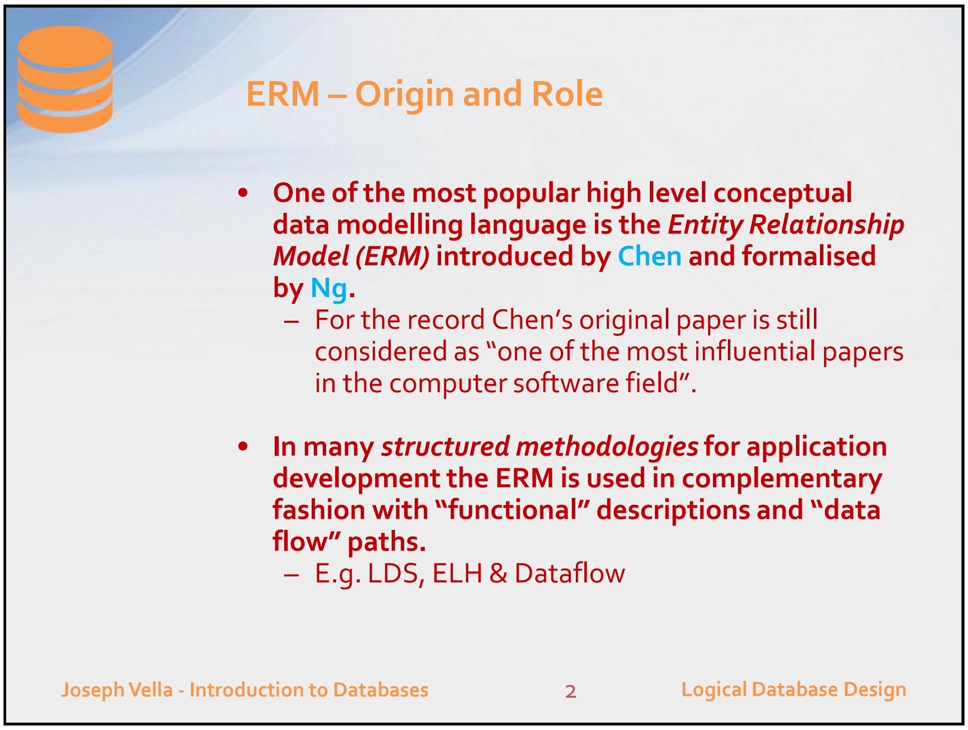


Prof. Joseph G. Vella ©

Logical Database Modelling (Entity Relationship Model – ERM)

1

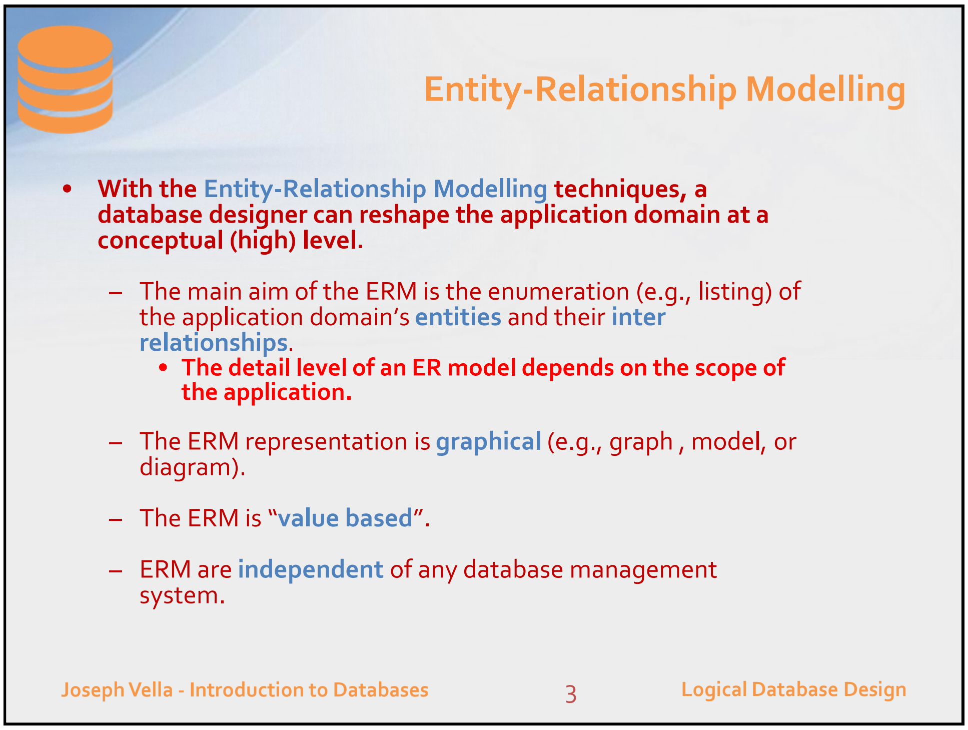


ERM – Origin and Role

- **One of the most popular high level conceptual data modelling language is the *Entity Relationship Model (ERM)* introduced by Chen and formalised by Ng.**
  - For the record Chen's original paper is still considered as "one of the most influential papers in the computer software field".
- **In many *structured methodologies* for application development the ERM is used in complementary fashion with "functional" descriptions and "data flow" paths.**
  - E.g. LDS, ELH & Dataflow

Joseph Vella - Introduction to Databases 2 Logical Database Design

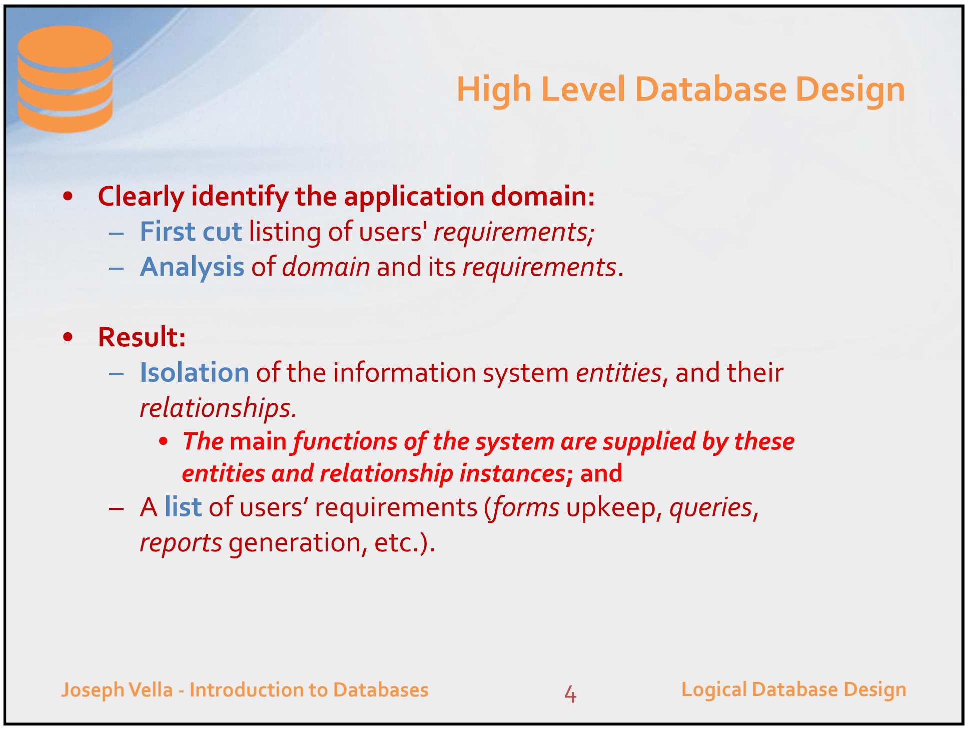
2



Entity-Relationship Modelling

- **With the Entity-Relationship Modelling techniques, a database designer can reshape the application domain at a conceptual (high) level.**
  - The main aim of the ERM is the enumeration (e.g., listing) of the application domain’s **entities** and their **inter relationships**.
    - **The detail level of an ER model depends on the scope of the application.**
  - The ERM representation is **graphical** (e.g., graph , model, or diagram).
  - The ERM is “**value based**”.
  - ERM are **independent** of any database management system.

3

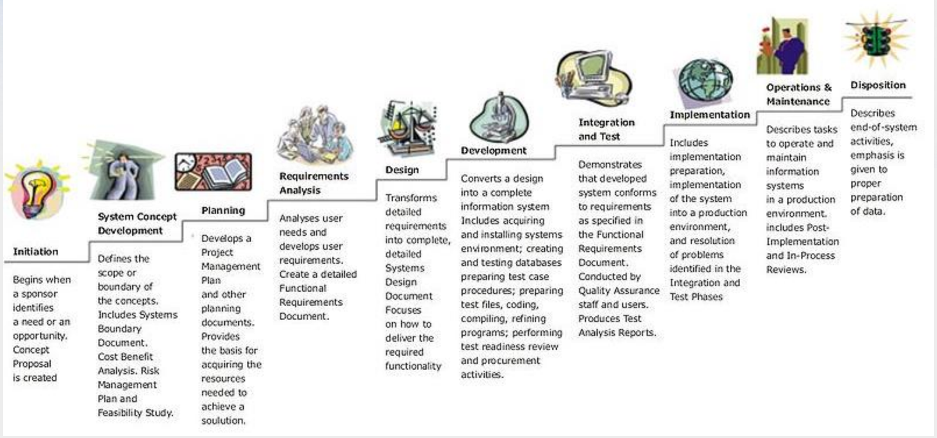


High Level Database Design

- **Clearly identify the application domain:**
  - First cut listing of users' *requirements;*
  - Analysis of *domain* and its *requirements*.
- **Result:**
  - Isolation of the information system *entities*, and their *relationships.*
    - ***The main functions of the system are supplied by these entities and relationship instances*; and**
  - A list of users’ requirements (*forms* upkeep, *queries*, *reports* generation, etc.).

4

Systems development life cycle (SDLC):
of a computer information system



Initiation: Begins when a sponsor identifies a need or an opportunity. Concept Proposal is created

System Concept Development: Defines the scope or boundary of the concepts. Includes Systems Boundary Document. Cost Benefit Analysis. Risk Management Plan and Feasibility Study.

Planning: Develops a Project Management Plan and other planning documents. Provides the basis for acquiring the resources needed to achieve a soulution.

Requirements Analysis: Analyses user needs and develops user requirements. Create a detailed Functional Requirements Document.

Design: Transforms detailed requirements into complete, detailed Systems Design Document Focuses on how to deliver the required functionality

Development: Converts a design into a complete information system Includes acquiring and installing systems environment; creating and testing databases preparing test case procedures; preparing test files, coding, compiling, refining programs; performing test readiness review and procurement activities.

Integration and Test: Demonstrates that developed system conforms to requirements as specified in the Functional Requirements Document. Conducted by Quality Assurance staff and users. Produces Test Analysis Reports.

Implementation: Includes implementation preparation, implementation of the system into a production environment, and resolution of problems identified in the Integration and Test Phases

Operations & Maintenance: Describes tasks to operate and maintain information systems in a production environment. includes Post-Implementation and In-Process Reviews.

Disposition: Describes end-of-system activities, emphasis is given to proper preparation of data.

Joseph Vella - Introduction to Databases 5 Logical Database Design

5

SDLC and DBLC overlap

	Initiation	System Concept Development	Planning	Requirements Analysis	Design	Development	Integration and Test	Implementation	Operations and Maintenance	Disposition
Information Requirements				+					+	
Logical Design					+				+	+
Physical Design					+	+	+		+	+
Implementation						+	+	+	+	

Joseph Vella - Introduction to Databases 6 Logical Database Design

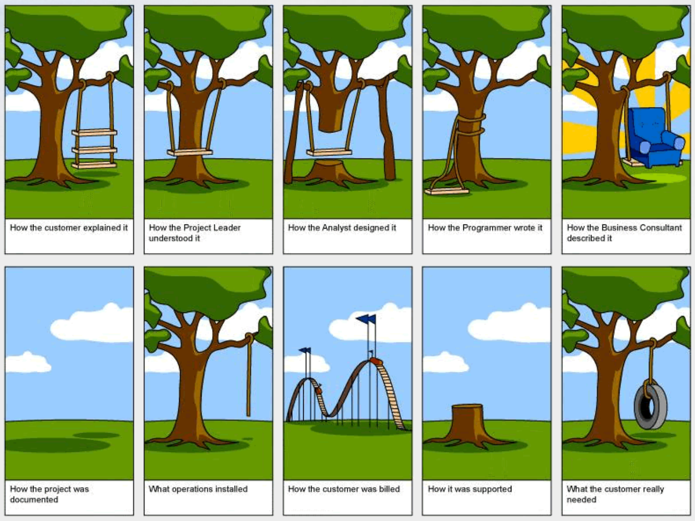
6

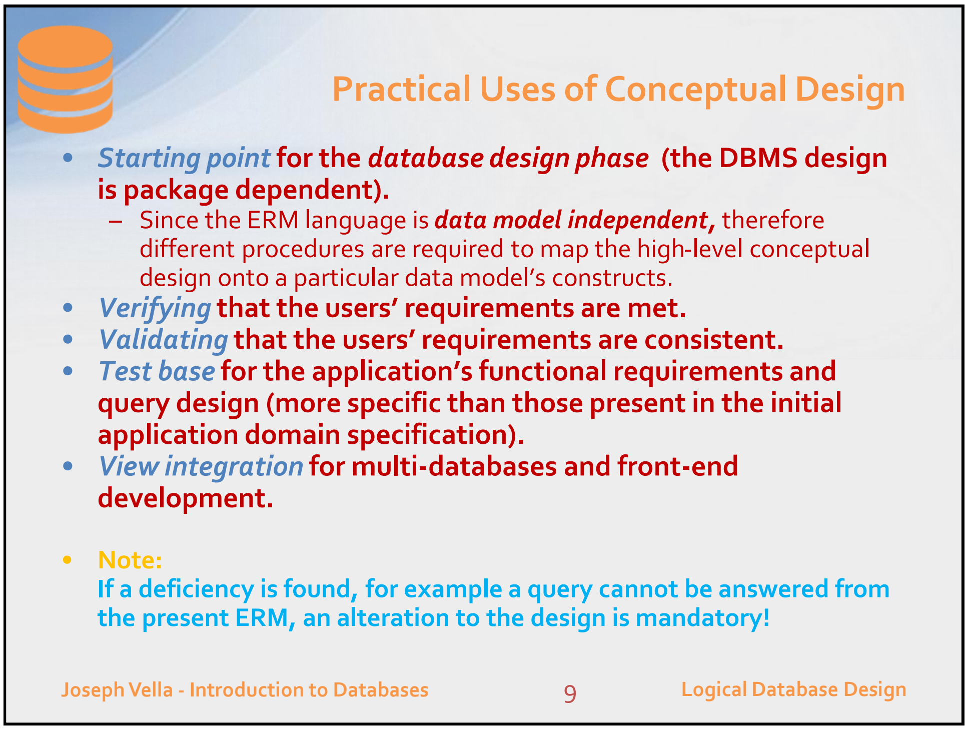
Systems Development Risks

- Better not to underestimate the initial stages (e.g., requirements and data requirements);
- Too formal against too informal specification dilemma;
- Technological advancement not properly gauged during design and development;
- Changing requirements. For example, in
  - Legal;
  - Business environment.

Systems Development Risks

a case study -cartoon


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

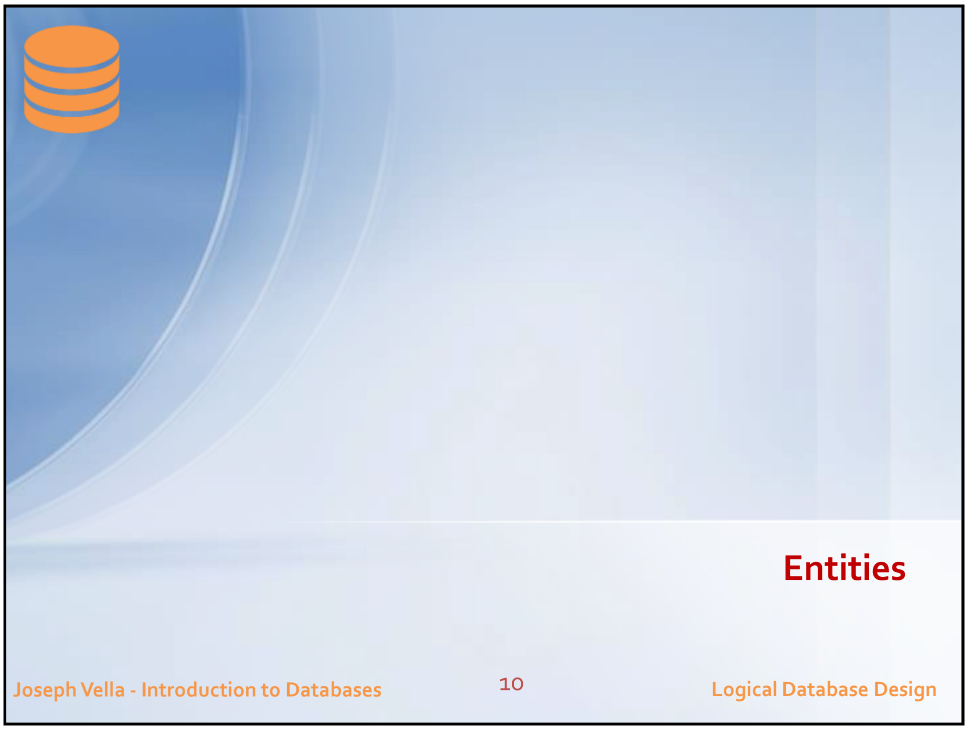


Practical Uses of Conceptual Design

- *Starting point* for the *database design phase* (the DBMS design is package dependent).
  - Since the ERM language is ***data model independent***, therefore different procedures are required to map the high-level conceptual design onto a particular data model's constructs.
- *Verifying* that the users' requirements are met.
- *Validating* that the users' requirements are consistent.
- *Test base* for the application's functional requirements and query design (more specific than those present in the initial application domain specification).
- *View integration* for multi-databases and front-end development.

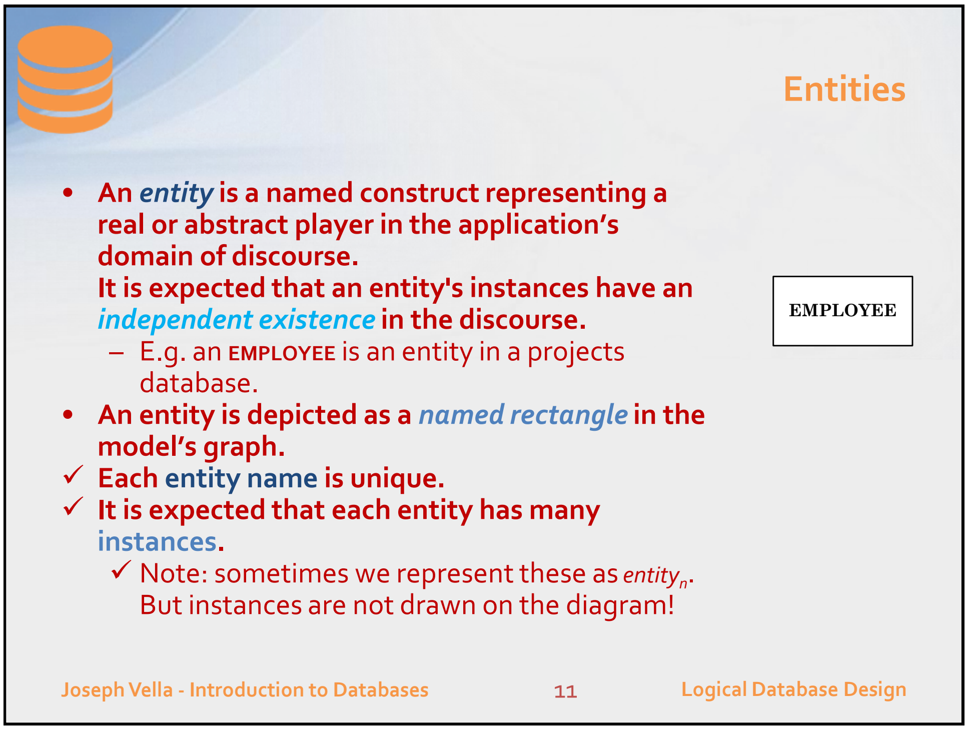
- Note:
  If a deficiency is found, for example a query cannot be answered from the present ERM, an alteration to the design is mandatory!

9



Entities

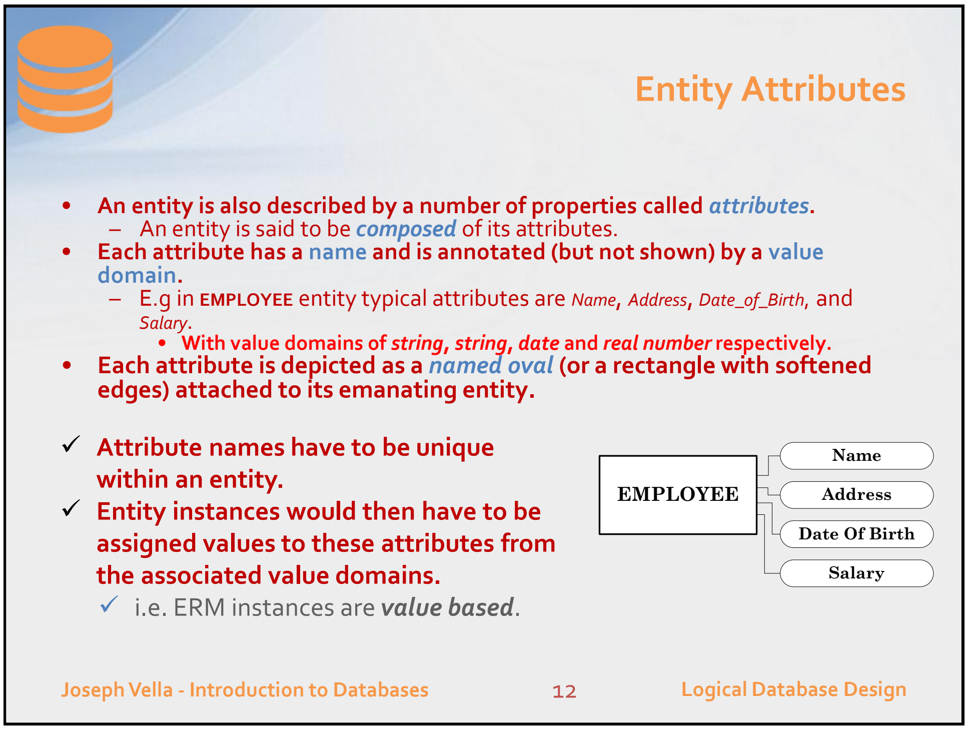
10



Entities

- An *entity* is a named construct representing a real or abstract player in the application's domain of discourse.
  It is expected that an entity's instances have an *independent existence* in the discourse.
  - E.g. an **EMPLOYEE** is an entity in a projects database.
- An entity is depicted as a *named rectangle* in the model's graph.
- ✓ Each entity name is unique.
- ✓ It is expected that each entity has many instances.
  - ✓ Note: sometimes we represent these as *entity$_n$*. But instances are not drawn on the diagram!

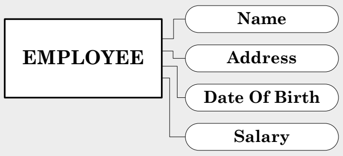
11



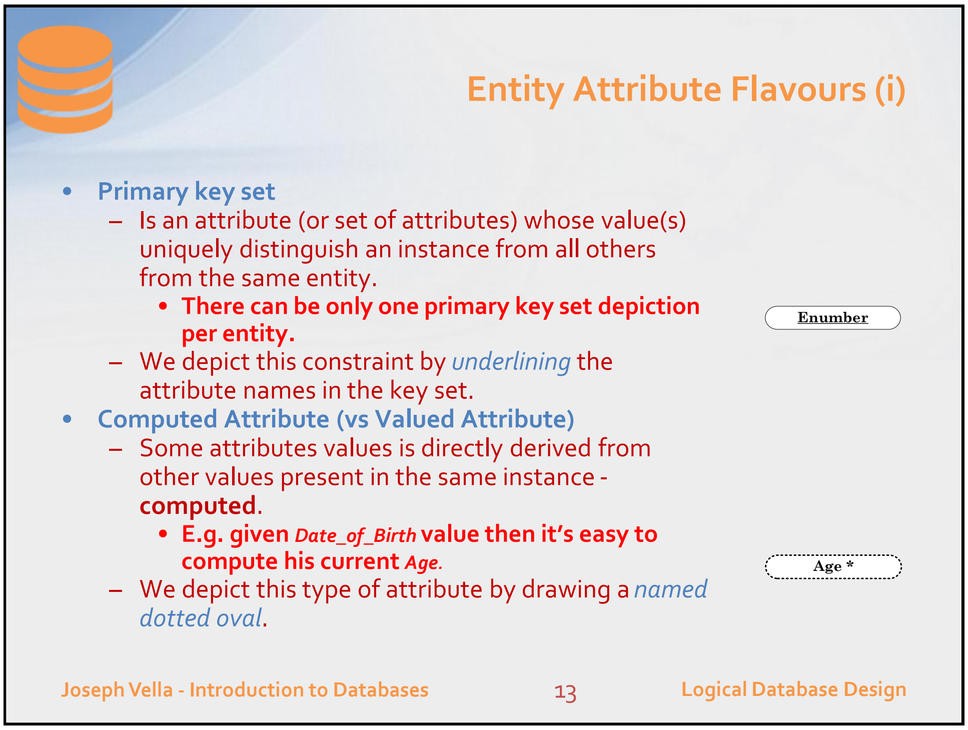
Entity Attributes

- An entity is also described by a number of properties called *attributes*.
  - An entity is said to be *composed* of its attributes.
- Each attribute has a name and is annotated (but not shown) by a value domain.
  - E.g in **EMPLOYEE** entity typical attributes are *Name*, *Address*, *Date_of_Birth*, and *Salary*.
    - With value domains of *string*, *string*, *date* and *real number* respectively.
- Each attribute is depicted as a *named oval* (or a rectangle with softened edges) attached to its emanating entity.

- ✓ Attribute names have to be unique within an entity.
- ✓ Entity instances would then have to be assigned values to these attributes from the associated value domains.
  - ✓ i.e. ERM instances are ***value based***.



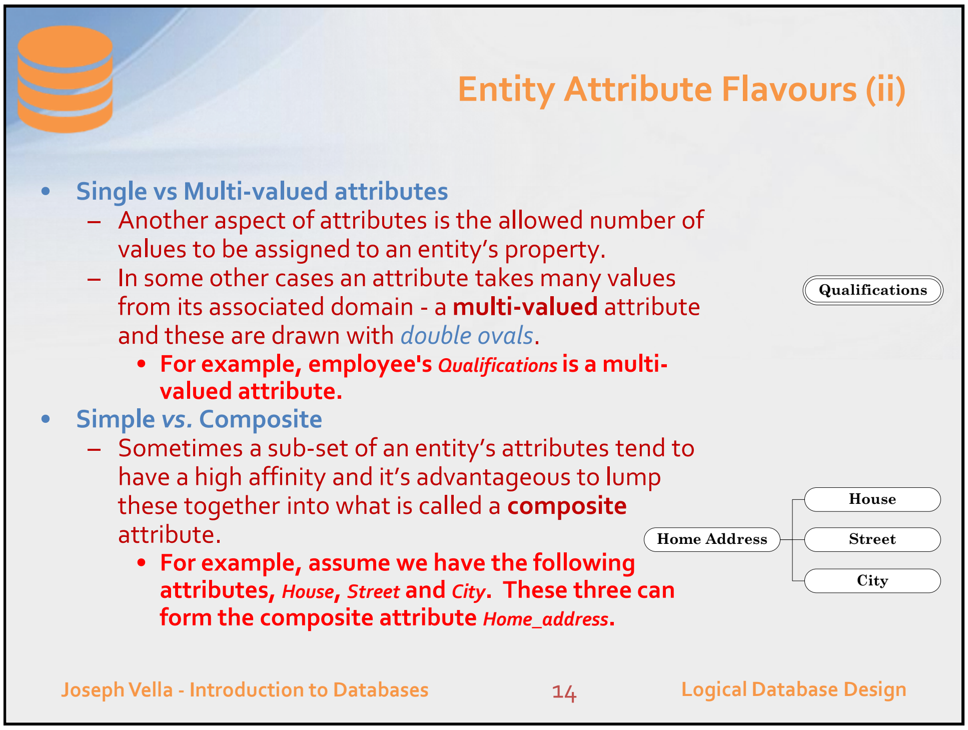
12



Entity Attribute Flavours (i)

- Primary key set
  - Is an attribute (or set of attributes) whose value(s) uniquely distinguish an instance from all others from the same entity.
    - **There can be only one primary key set depiction per entity.**
  - We depict this constraint by *underlining* the attribute names in the key set.
- Computed Attribute (vs Valued Attribute)
  - Some attributes values is directly derived from other values present in the same instance - **computed**.
    - **E.g. given *Date_of_Birth* value then it's easy to compute his current *Age*.**
  - We depict this type of attribute by drawing a *named dotted oval*.

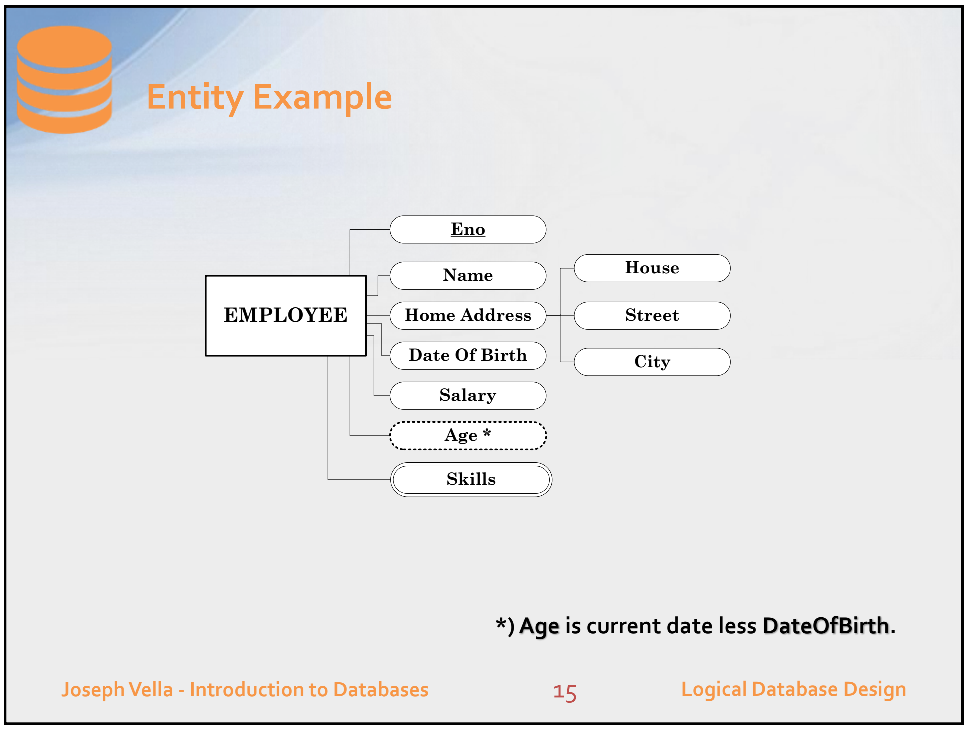
13



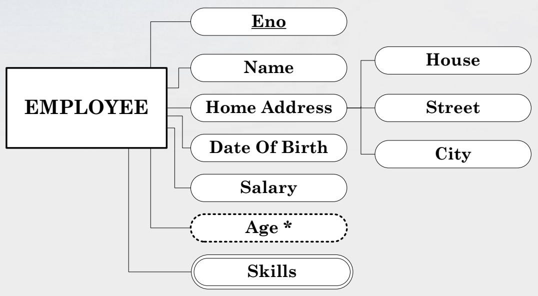
Entity Attribute Flavours (ii)

- Single vs Multi-valued attributes
  - Another aspect of attributes is the allowed number of values to be assigned to an entity's property.
  - In some other cases an attribute takes many values from its associated domain - a **multi-valued** attribute and these are drawn with *double ovals*.
    - **For example, employee's *Qualifications* is a multi-valued attribute.**
- Simple *vs.* Composite
  - Sometimes a sub-set of an entity's attributes tend to have a high affinity and it's advantageous to lump these together into what is called a **composite** attribute.
    - **For example, assume we have the following attributes, *House*, *Street* and *City*. These three can form the composite attribute *Home_address*.**

14



Entity Example



*) **Age** is current date less **DateOfBirth**.

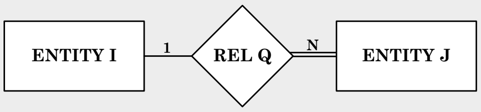
15

Relationships

16

Relationships

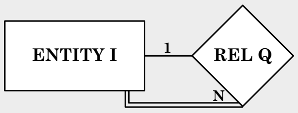
- **A relationship is a named association between a number of entities that is governed by certain rules and sometimes qualified with its own attributes.**
  - A relationship instance is a particular case of relationship and qualified by the respective entity instances it's relating.
  - A relationship name is unique between its participating entities.
  - A relationship is depicted with:
    - ***a diamond that inscribes its name;***
    - **a line connecting the relationship (diamond) with each participating entity (rectangle). This *line* carries two modeling indications:**
      - **Cardinality** (number – e.g., 1 and N);
      - **Participation** (single or double line).



17

Relationships – Degree constraints (i)

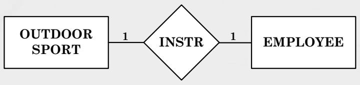
- **This constraint deals with enumerating the entities in a relationship.**
  - Any relationship instance must have an entity instance from each respective participating entities.
- **The degree of a relationship is the number of entities that define the relationship.**
  - If there are two, then the relationship is *binary* and if there are three then it's called *ternary* etc.
    - **If more than three we call them *n-ary*.**
    - **In practice the most common type is the binary relationship.**
- **If an entity is repeated in a relationship definition then the relationship is said to be recursive.**

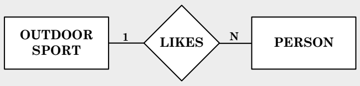


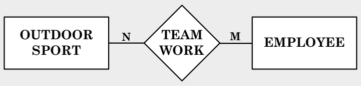
18

Relationships – Cardinality constraints (ii)

- How many times can an entity's instances appear in separate in relationship instances?
  - The most common cardinalities are the "*one*" (for once) to "*many*".
- For a binary relationship the number of these possibilities are four namely;
  - one to one (*1-1*),
  - one to many (*1-N*),
  - many to one (*N-1*), and
  - many to many (*N-M*).



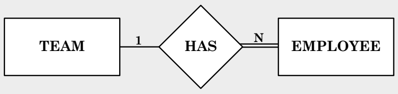




- In the modelling diagram the cardinalities are depicted by including the symbols "*1*" or "*N*" (or "*M*", "*P*") on the opposite edges of the relationship diamond.

Relationships – Participation constraints (iii)

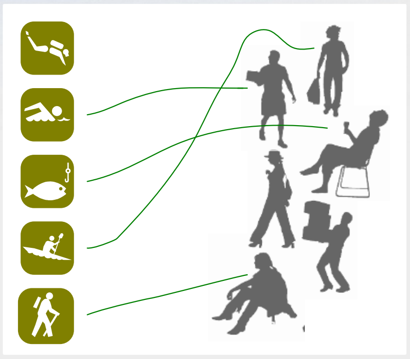
- Another constraint is the level of participation of an entity's instances in a relationship.
- The options are *total* (or *mandatory*) and *partial* (or *optional*) for each of the participating entity.
  - Total participation implies that all instances must participate in at least one relationship instance.
  - Partial participation implies that an entity instance may not be present in any relationship instance.
- In binary relationship there are four possible constraints;
  - *total-total,*
  - *total-partial,*
  - *partial-total, and*
  - *partial-partial.*

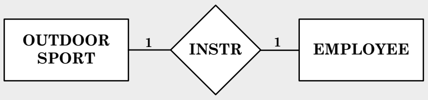


- For a total constraint the edge leading from a relationship to an entity is drawn as a *double line*.
  - E.g., Every employee instance is part of a team instance.

Relationships – 1-1 Participation

- Instructor relationship between outdoor sports and employees entities.

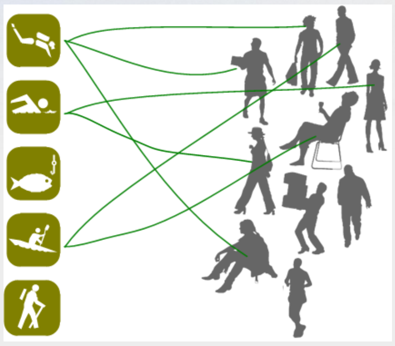


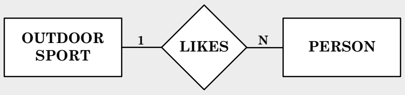


If a) a sport instance has to have one and only one instructor instance and b) each instructor instance can coach one and only one sport instance, then the line relationship participation constraints between **Sport** and **Instructor** each need to be total (double line).

Relationships – 1-N Participation

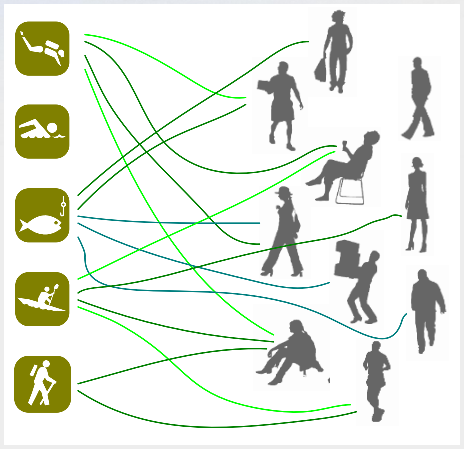
- Likes relationship between outdoor sports and persons entities.

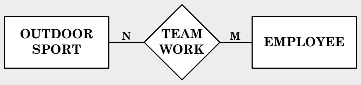




Relationships – N-M Participation

- Team work building relationship between outdoor sports and employees entities.

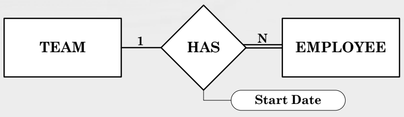




23

Relationship Attributes

- Relationships might have attributes that describe them.
  - This is denoted in an ERM diagram by connecting these attribute to the respective diamond.
    - A typical example of a relationship attribute is the case of the *Starting_date* for the *Employs* relationship between the entities TEAM and EMPLOYEE.

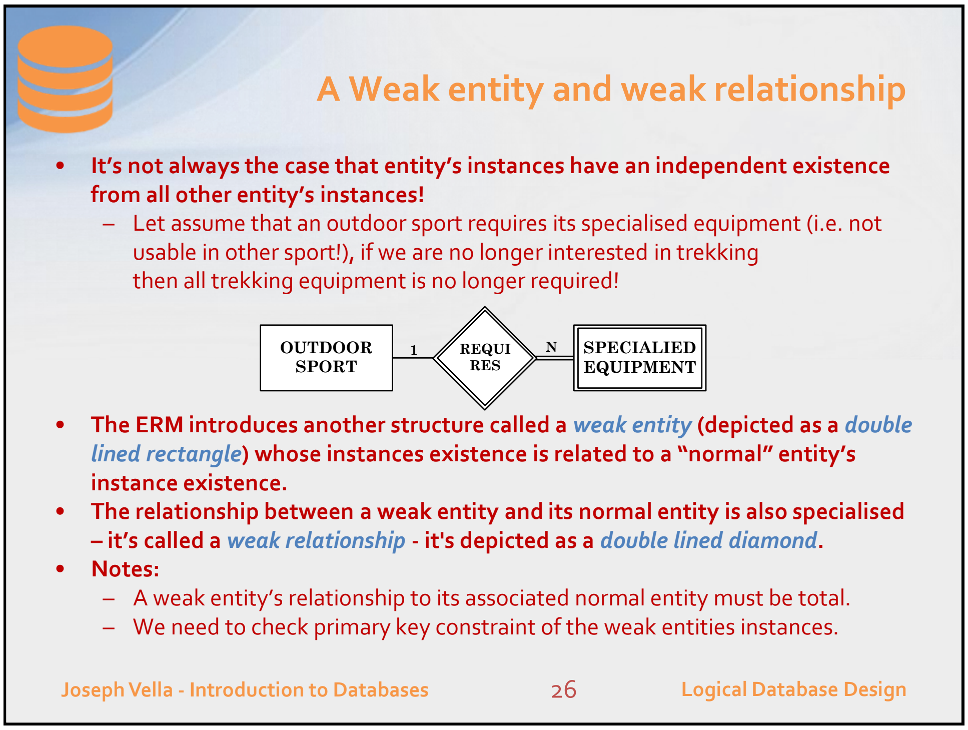


  - It is not uncommon, in a binary 1-N, that a relationship attribute is put with the other attributes of the many cardinality side entity (e.g. EMPLOYEE).
    - This is of course a "legal" arrangement but of questionable style even if the semantics of the requirements are not exactly compromised.

24

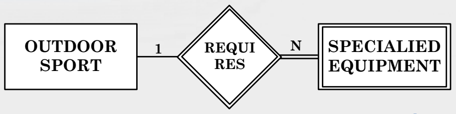
Weak Entities

25



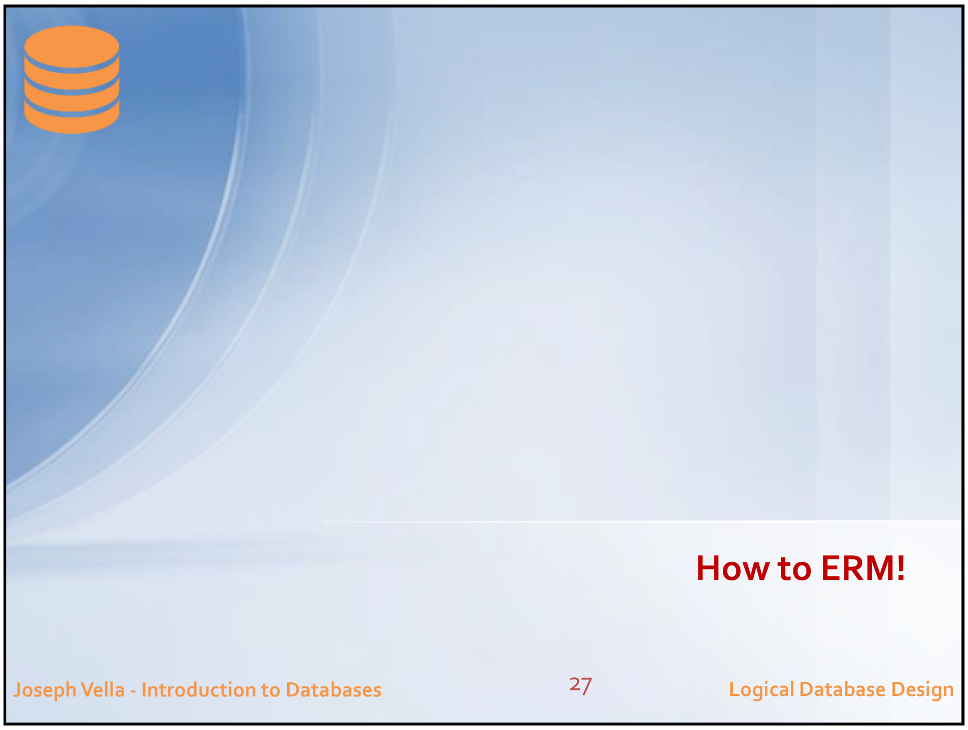
A Weak entity and weak relationship

- **It's not always the case that entity's instances have an independent existence from all other entity's instances!**
  - Let assume that an outdoor sport requires its specialised equipment (i.e. not usable in other sport!), if we are no longer interested in trekking then all trekking equipment is no longer required!



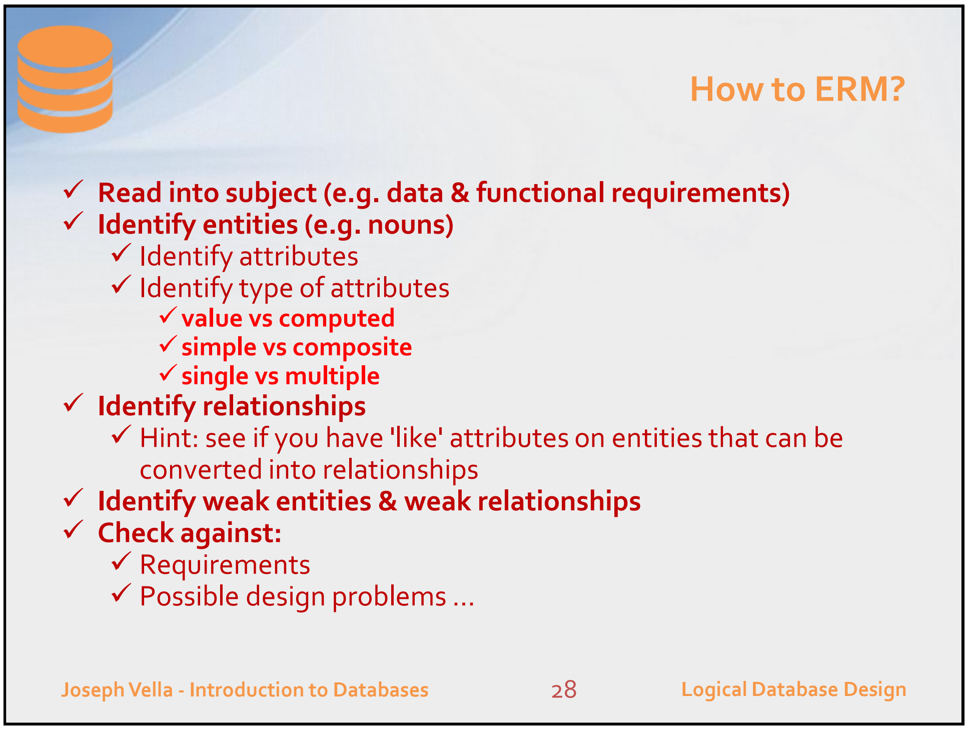
- **The ERM introduces another structure called a *weak entity* (depicted as a *double lined rectangle*) whose instances existence is related to a "normal" entity's instance existence.**
- **The relationship between a weak entity and its normal entity is also specialised – it's called a *weak relationship* - it's depicted as a *double lined diamond*.**
- **Notes:**
  - A weak entity's relationship to its associated normal entity must be total.
  - We need to check primary key constraint of the weak entities instances.

26



How to ERM!

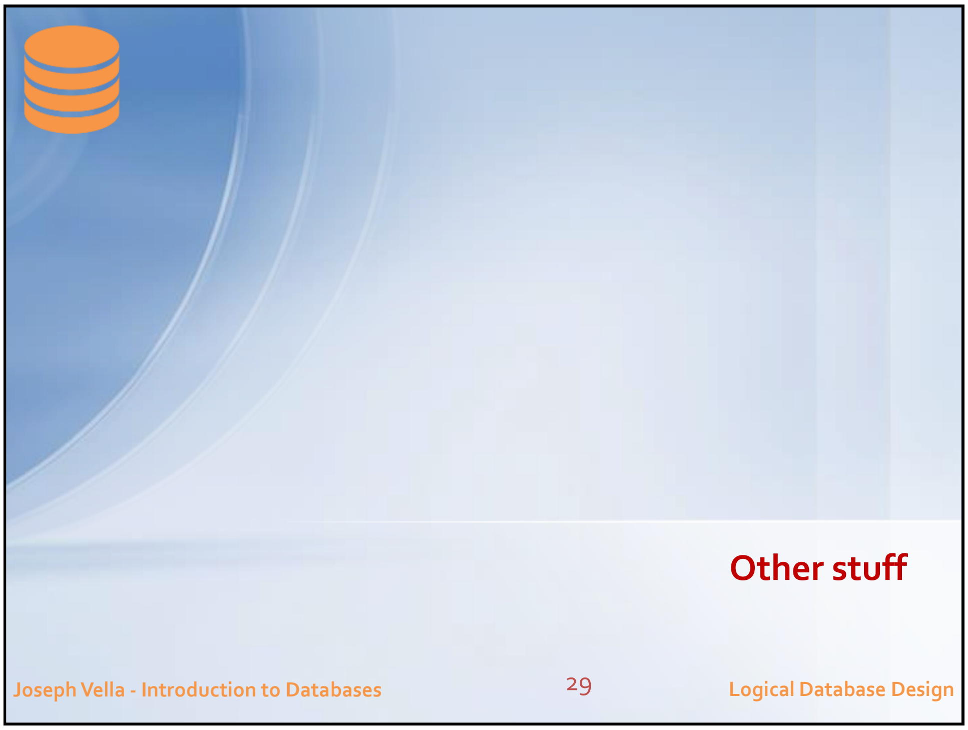
27



How to ERM?

- ✓ **Read into subject (e.g. data & functional requirements)**
- ✓ **Identify entities (e.g. nouns)**
  - ✓ Identify attributes
  - ✓ Identify type of attributes
    - ✓ **value vs computed**
    - ✓ **simple vs composite**
    - ✓ **single vs multiple**
- ✓ **Identify relationships**
  - ✓ Hint: see if you have 'like' attributes on entities that can be converted into relationships
- ✓ **Identify weak entities & weak relationships**
- ✓ **Check against:**
  - ✓ Requirements
  - ✓ Possible design problems …

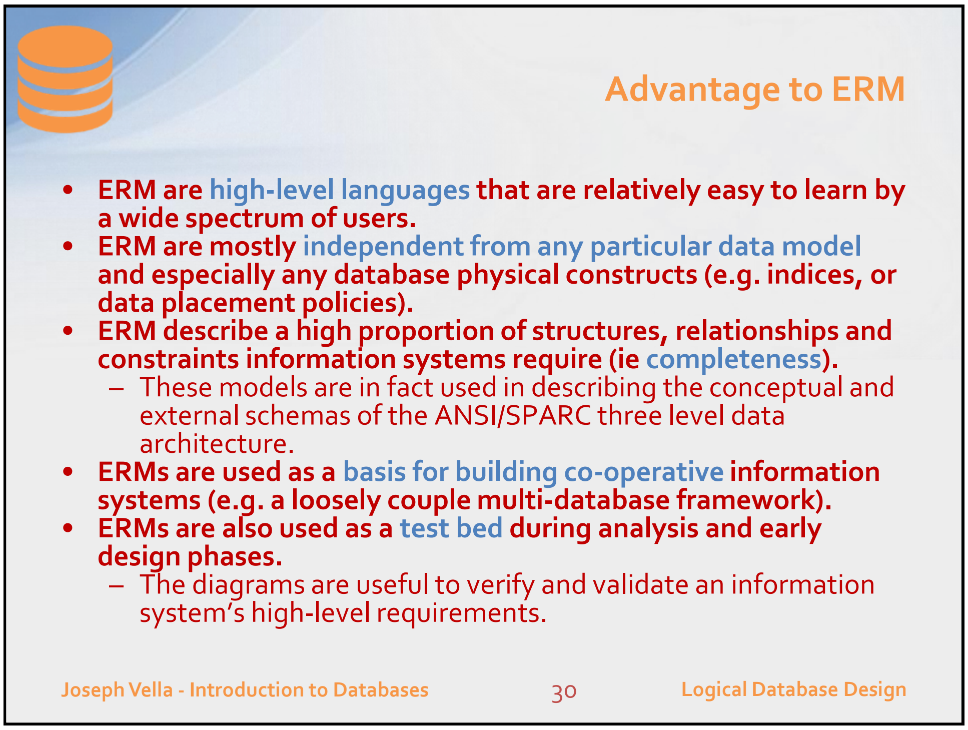
28



Other stuff

Joseph Vella - Introduction to Databases 29 Logical Database Design

29

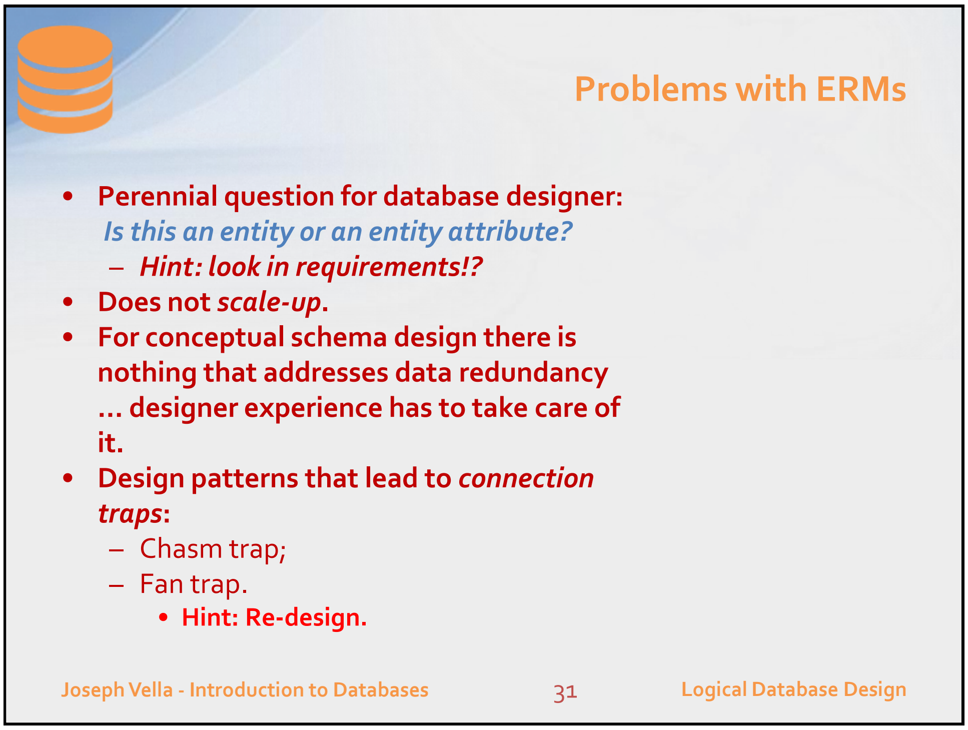


Advantage to ERM

- ERM are high-level languages that are relatively easy to learn by a wide spectrum of users.
- ERM are mostly independent from any particular data model and especially any database physical constructs (e.g. indices, or data placement policies).
- ERM describe a high proportion of structures, relationships and constraints information systems require (ie completeness).
  - These models are in fact used in describing the conceptual and external schemas of the ANSI/SPARC three level data architecture.
- ERMs are used as a basis for building co-operative information systems (e.g. a loosely couple multi-database framework).
- ERMs are also used as a test bed during analysis and early design phases.
  - The diagrams are useful to verify and validate an information system's high-level requirements.

Joseph Vella - Introduction to Databases 30 Logical Database Design

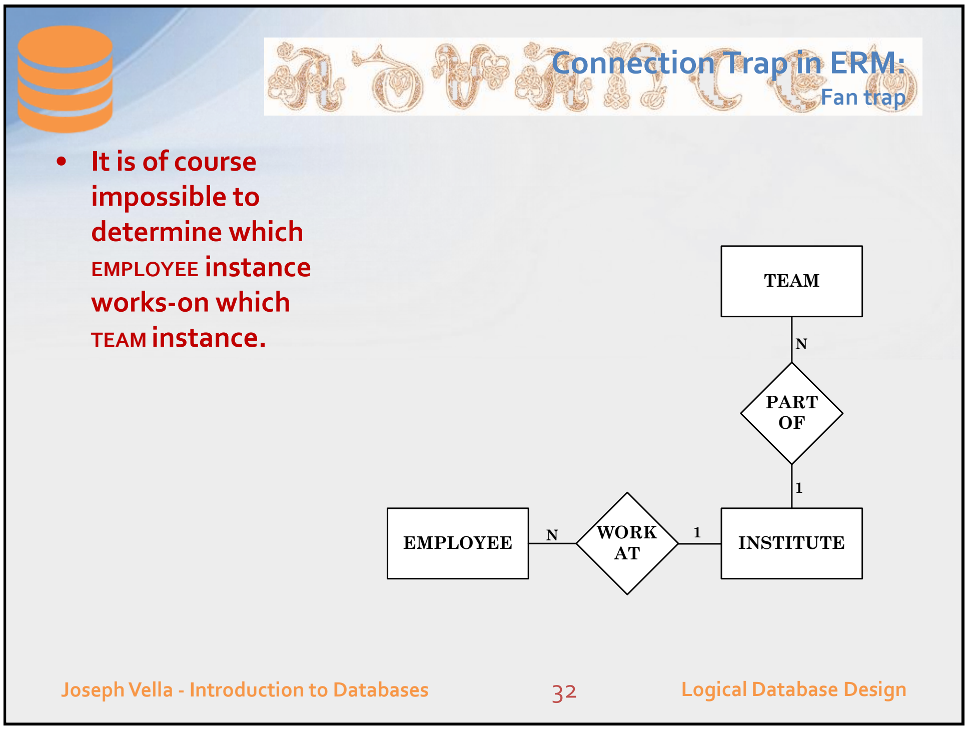
30



Problems with ERMs

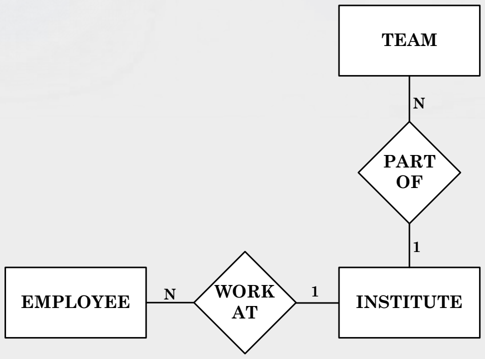
- Perennial question for database designer:
  Is this an entity or an entity attribute?
  - *Hint: look in requirements!?*
- Does not *scale-up*.
- For conceptual schema design there is nothing that addresses data redundancy … designer experience has to take care of it.
- Design patterns that lead to *connection traps*:
  - Chasm trap;
  - Fan trap.
    - Hint: Re-design.

31



Connection Trap in ERM: Fan trap

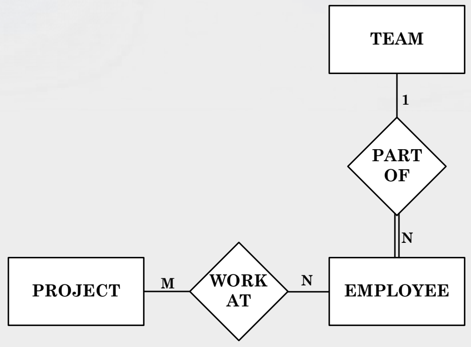
- It is of course impossible to determine which EMPLOYEE instance works-on which TEAM instance.



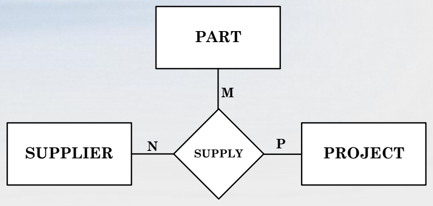
32

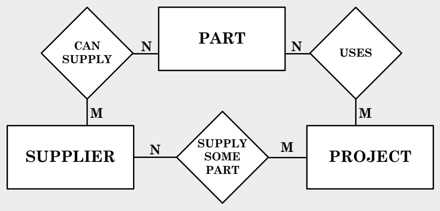
Connection Trap in ERM: Chasm trap

- Some instances of PROJECT may not connect to any TEAM instances through EMPLOYEE instances!

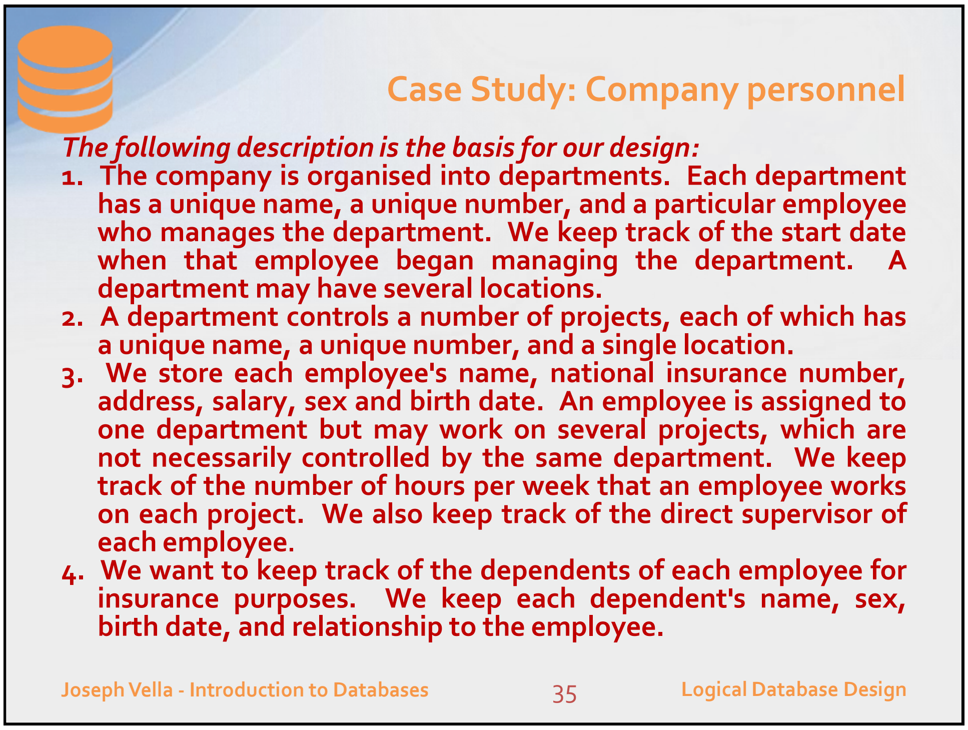


A Design Pattern Issue: Are two binary relationships equal to a ternary?





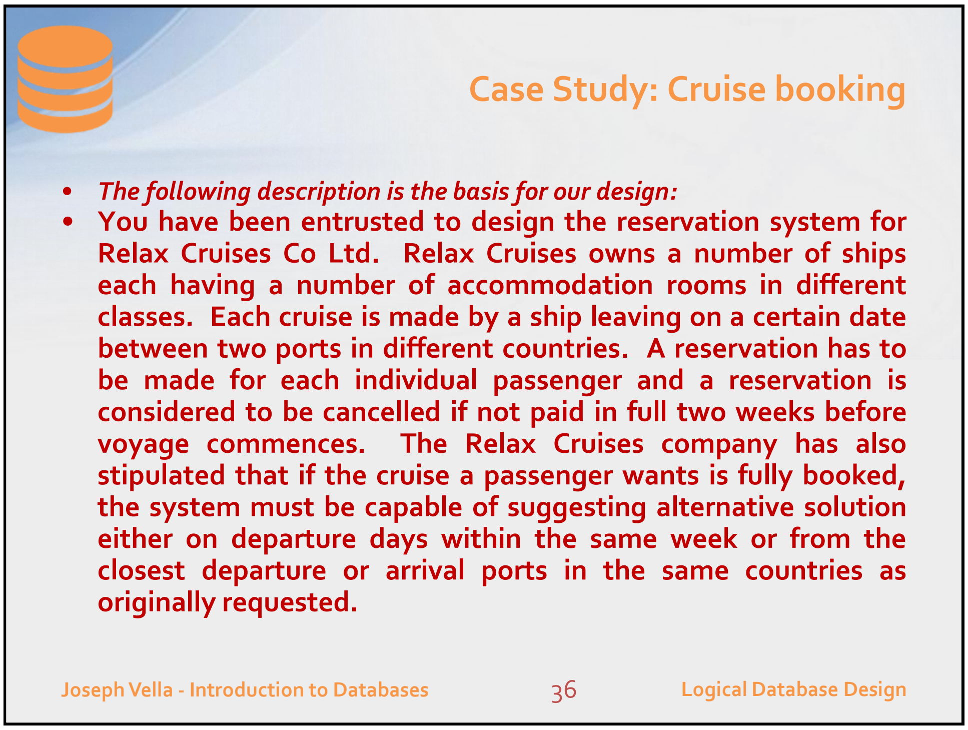
- No!
  As a ternary relationship can contain more relationship instances. In the lower design we cannot determine whether if a SUPPLIER instance has actually supplied a 'specific' PART to a 'specific' PROJECT.



Case Study: Company personnel

The following description is the basis for our design:

1. The company is organised into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
2. A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
3. We store each employee's name, national insurance number, address, salary, sex and birth date. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same department. We keep track of the number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee.
4. We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's name, sex, birth date, and relationship to the employee.



Case Study: Cruise booking

- *The following description is the basis for our design:*
- You have been entrusted to design the reservation system for Relax Cruises Co Ltd. Relax Cruises owns a number of ships each having a number of accommodation rooms in different classes. Each cruise is made by a ship leaving on a certain date between two ports in different countries. A reservation has to be made for each individual passenger and a reservation is considered to be cancelled if not paid in full two weeks before voyage commences. The Relax Cruises company has also stipulated that if the cruise a passenger wants is fully booked, the system must be capable of suggesting alternative solution either on departure days within the same week or from the closest departure or arrival ports in the same countries as originally requested.

Case Study: Cruise Workplace projects

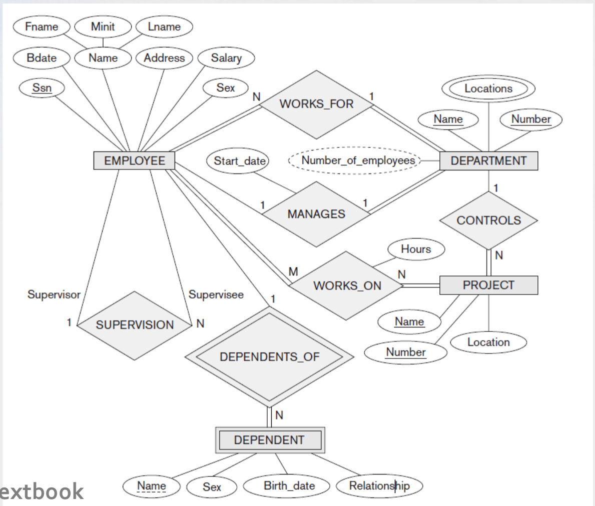
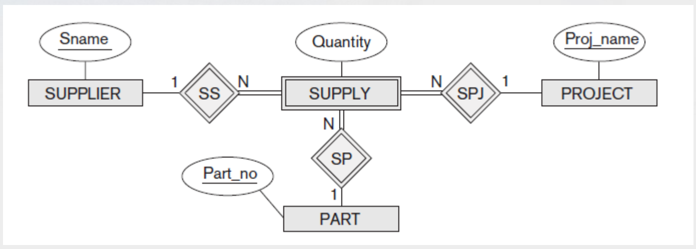


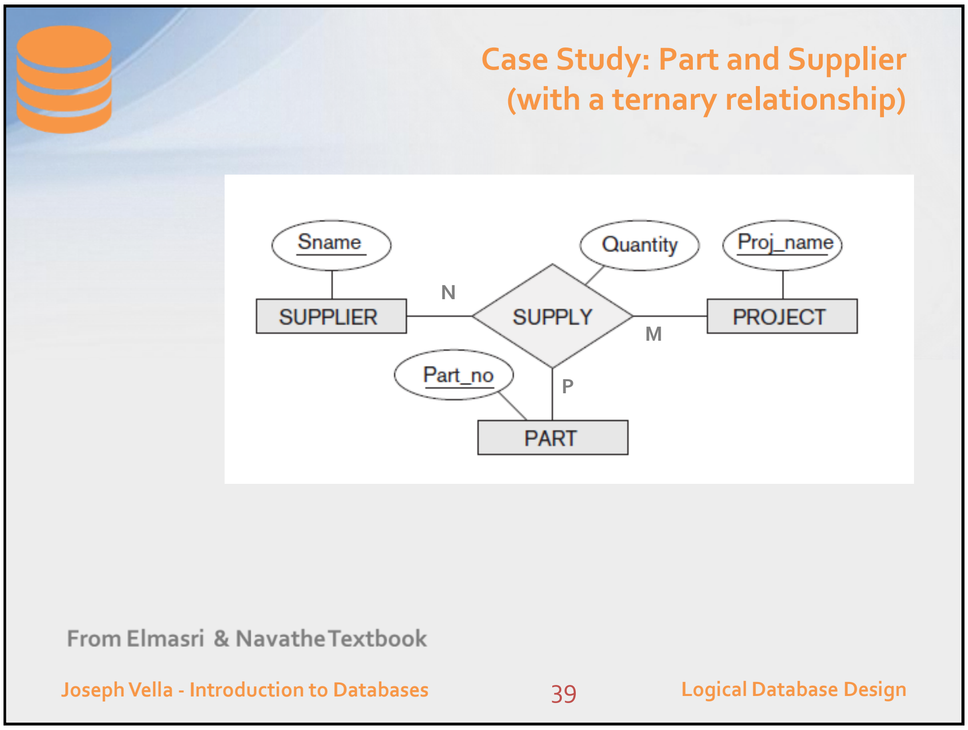
Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 3.14.

From Elmasri & Navathe Textbook

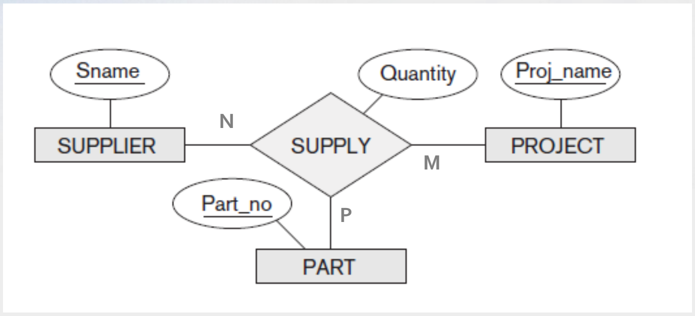
Case Study: Part and Supplier (with a weak relationship)



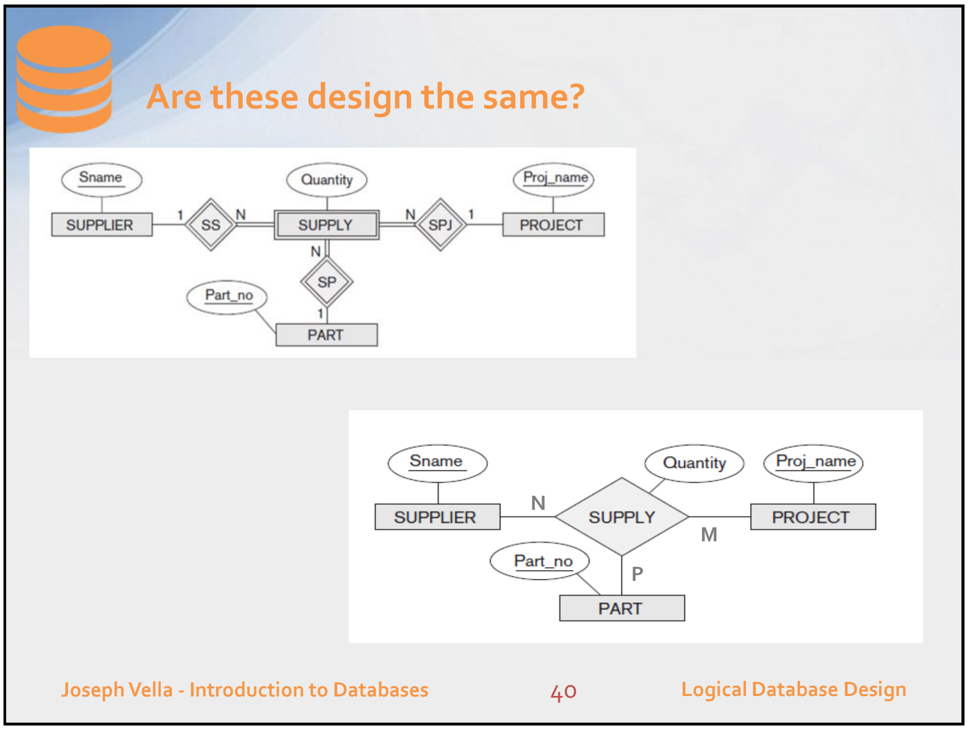
From Elmasri & Navathe Textbook



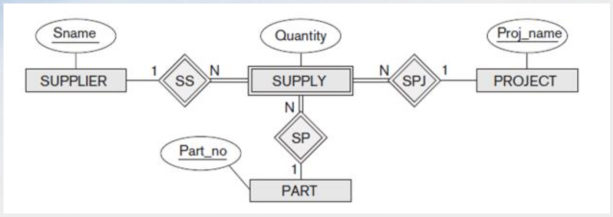
Case Study: Part and Supplier (with a ternary relationship)

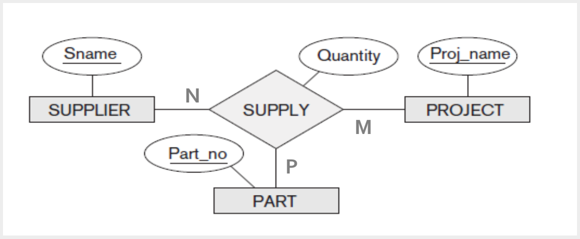


From Elmasri & Navathe Textbook



Are these design the same?


Sname
Quantity
Proj_name
SUPPLIER
1
SS
N
SUPPLY
N
SPJ
1
PROJECT
N
SP
1
Part_no
PART


Sname
Quantity
Proj_name
SUPPLIER
N
SUPPLY
M
PROJECT
Part_no
P
PART