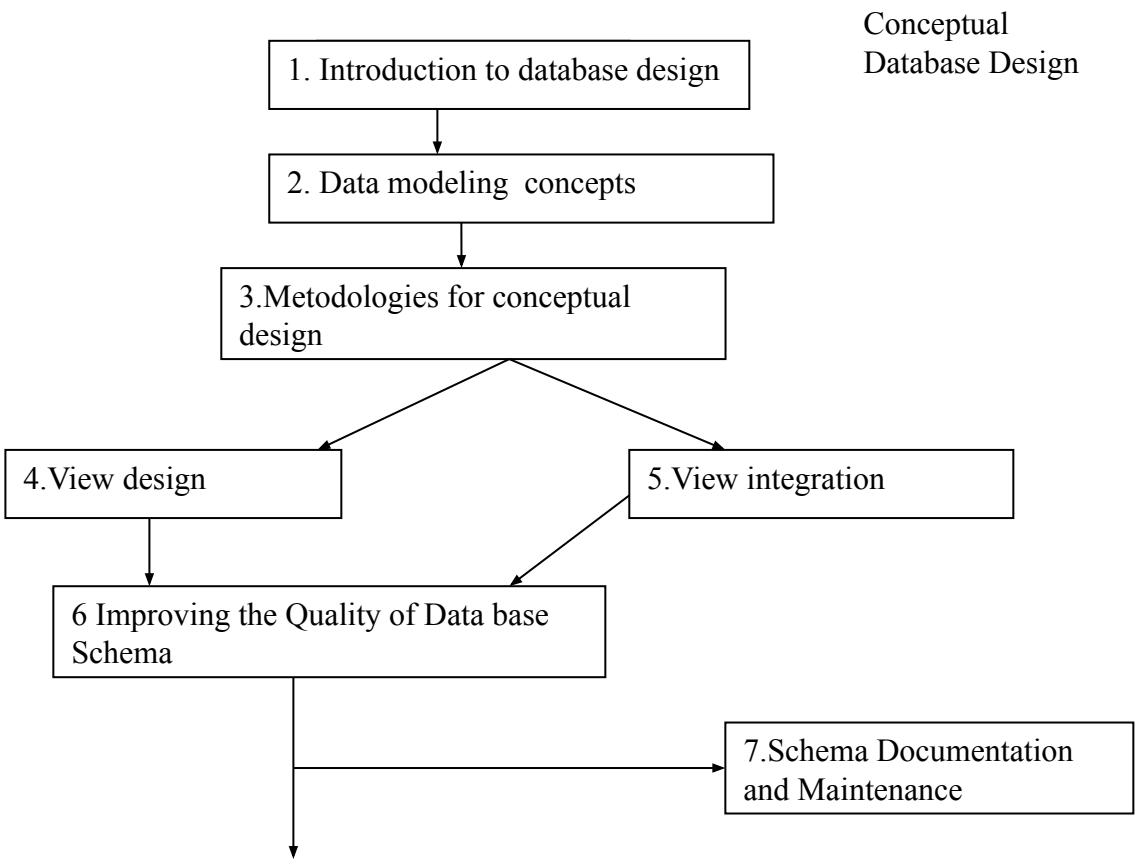
Information Systems Design

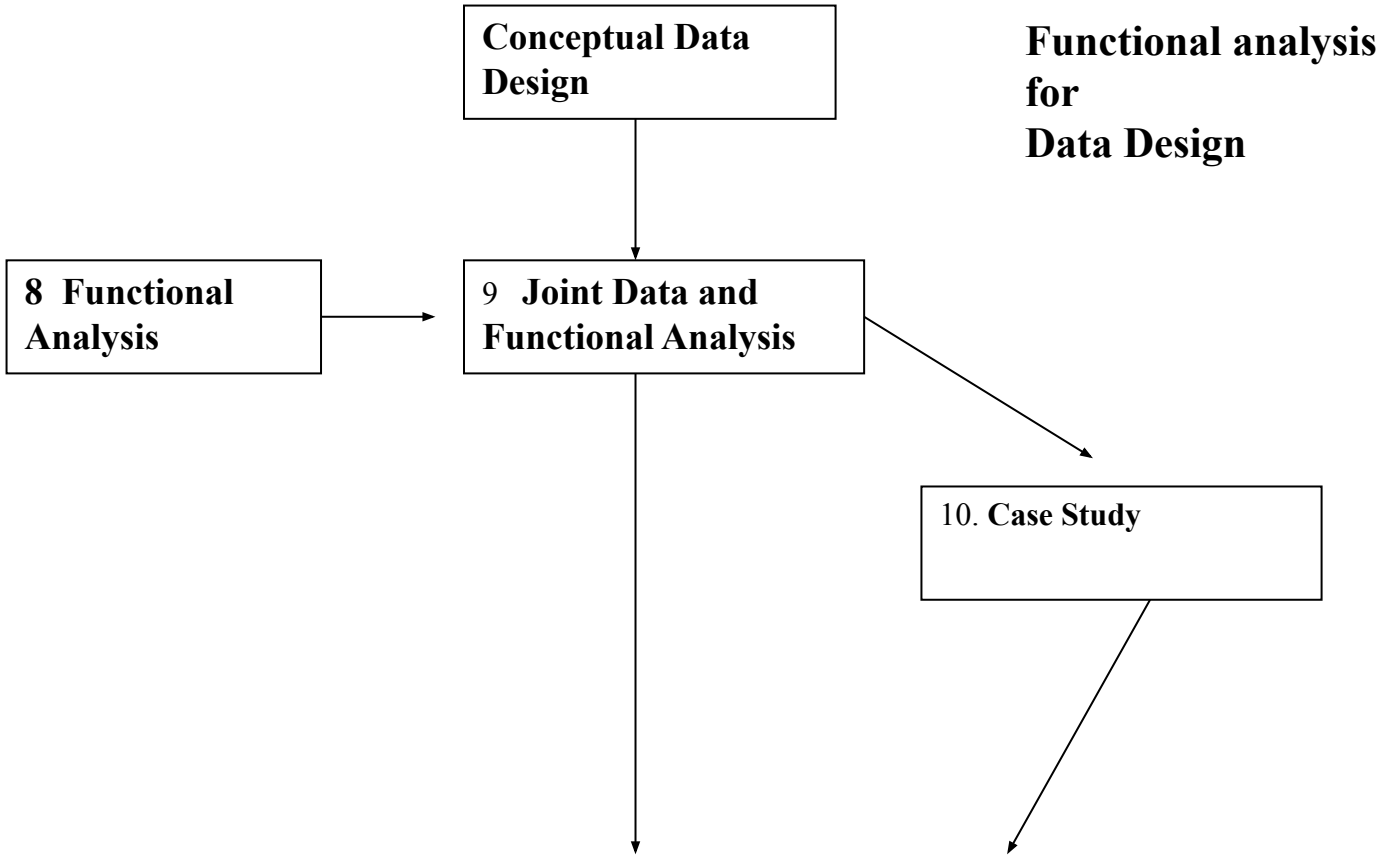
Goals of this course

- To provide f thorough and systematic treatment of conceptual and logical design
- To base this treatment on the Entity-Relation model
- To advocate that conceptual design and function anslysis be conducted together
- To address completely the translation of conceptual design in ER model in the three popular data models- relational, network, hierarhical, and vice versa
- To illustrate the concepts via realistic large case study
- To provide a survey of state of art of design tools
- To provide enough support for students in terms of exercises and bibliographic notes

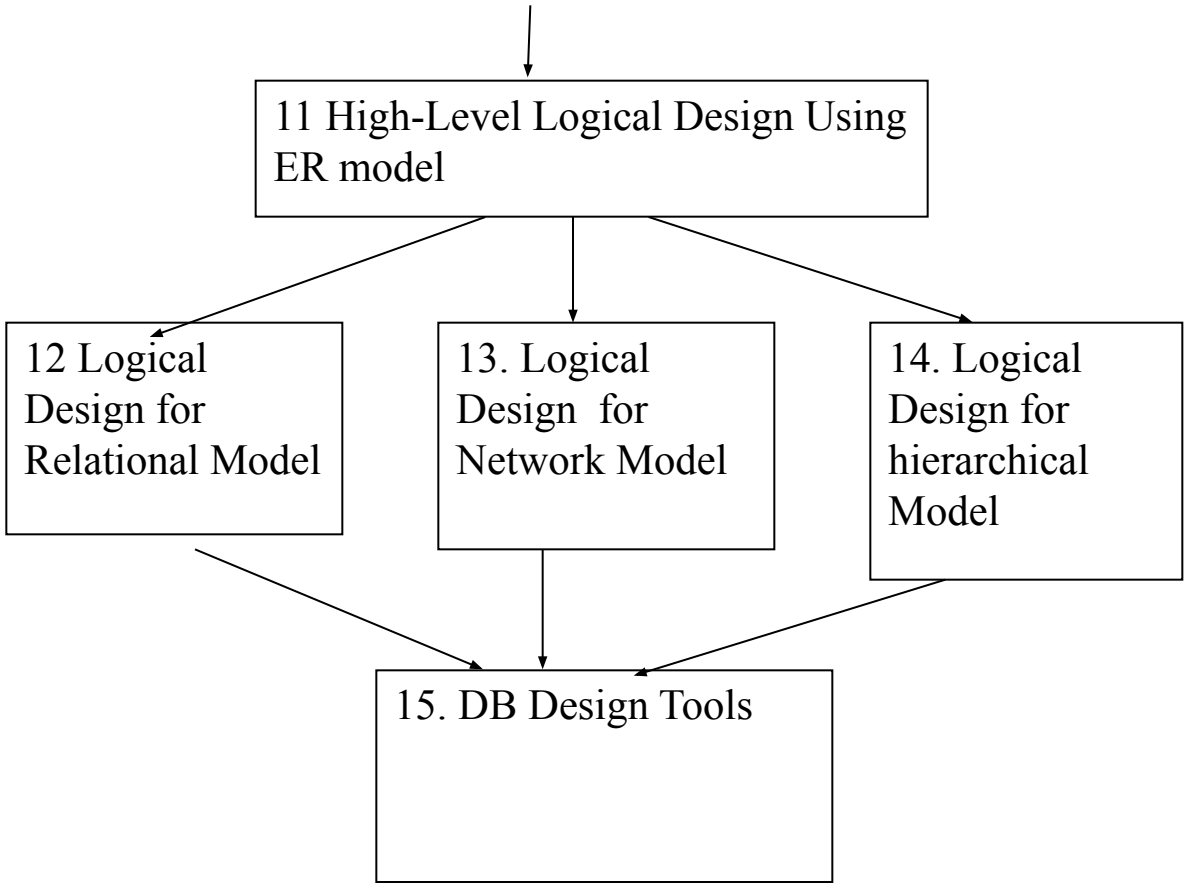
Conceptual DB Design



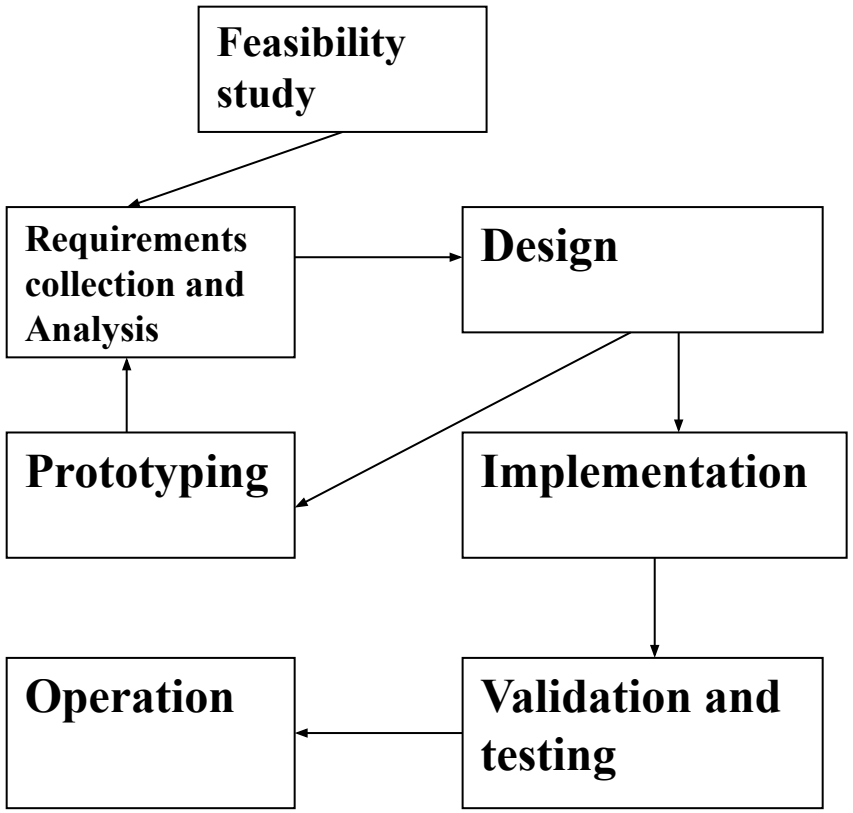
Functional Analysis for DB Design



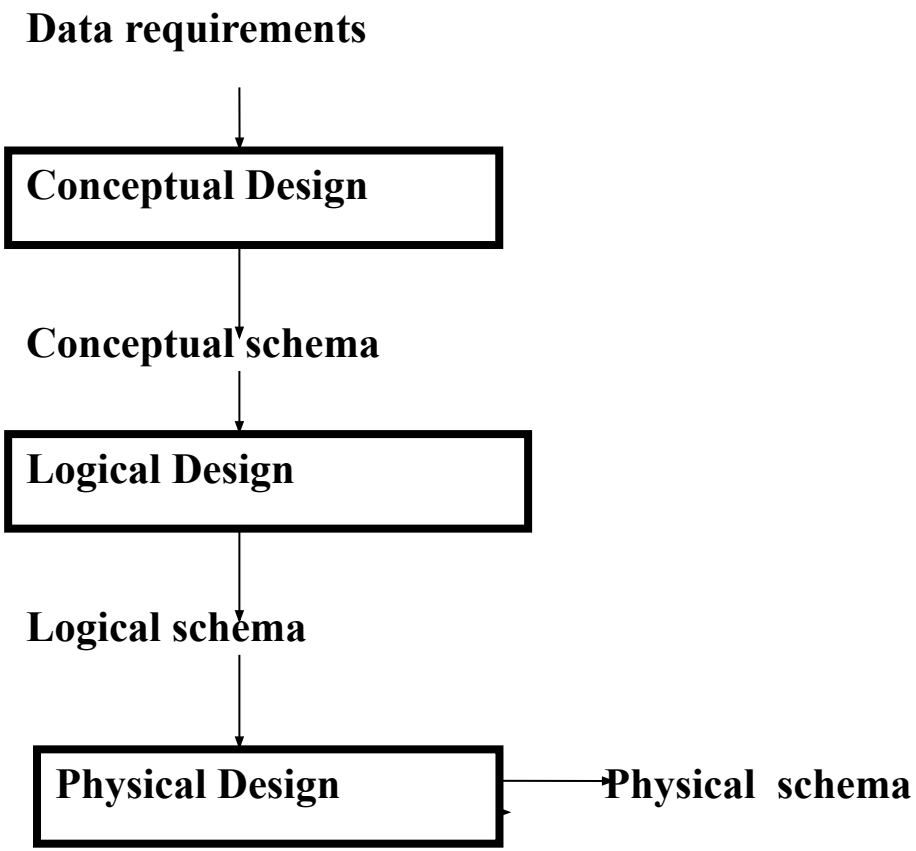
Logical Design and Data Tools

- 

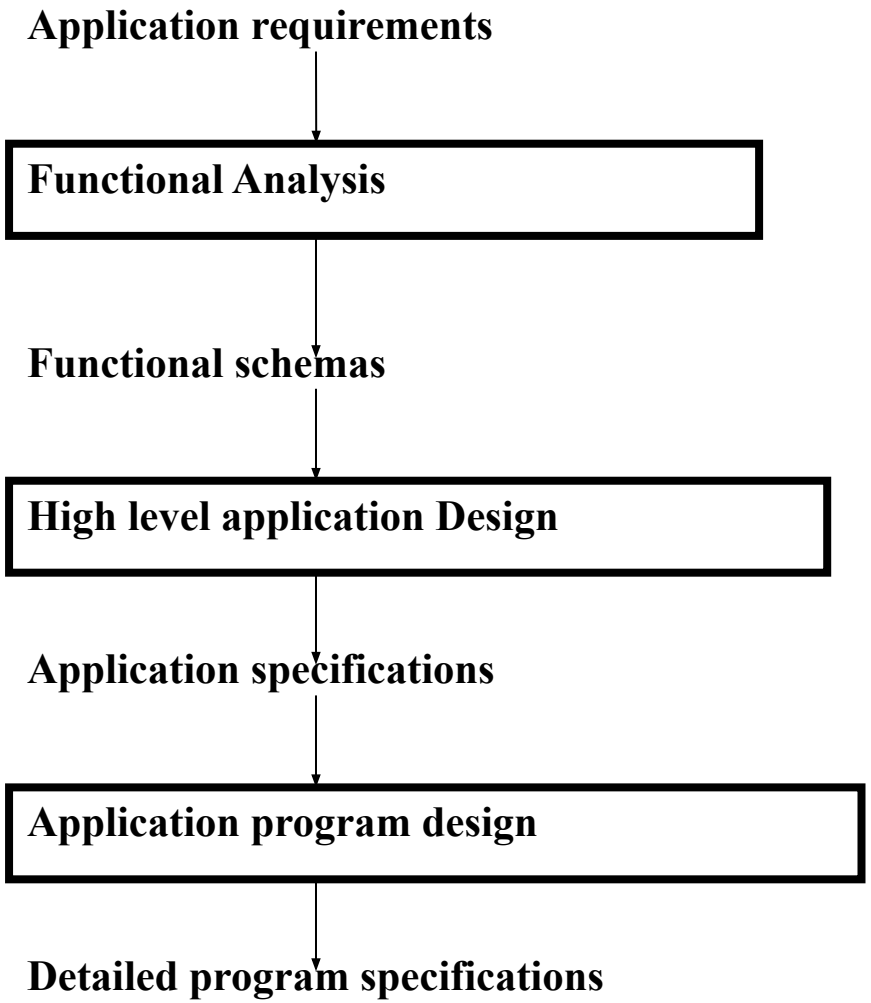
DB design in the Information Systems Life Cycle



Phases of DB Design



Function driven approach to information system design

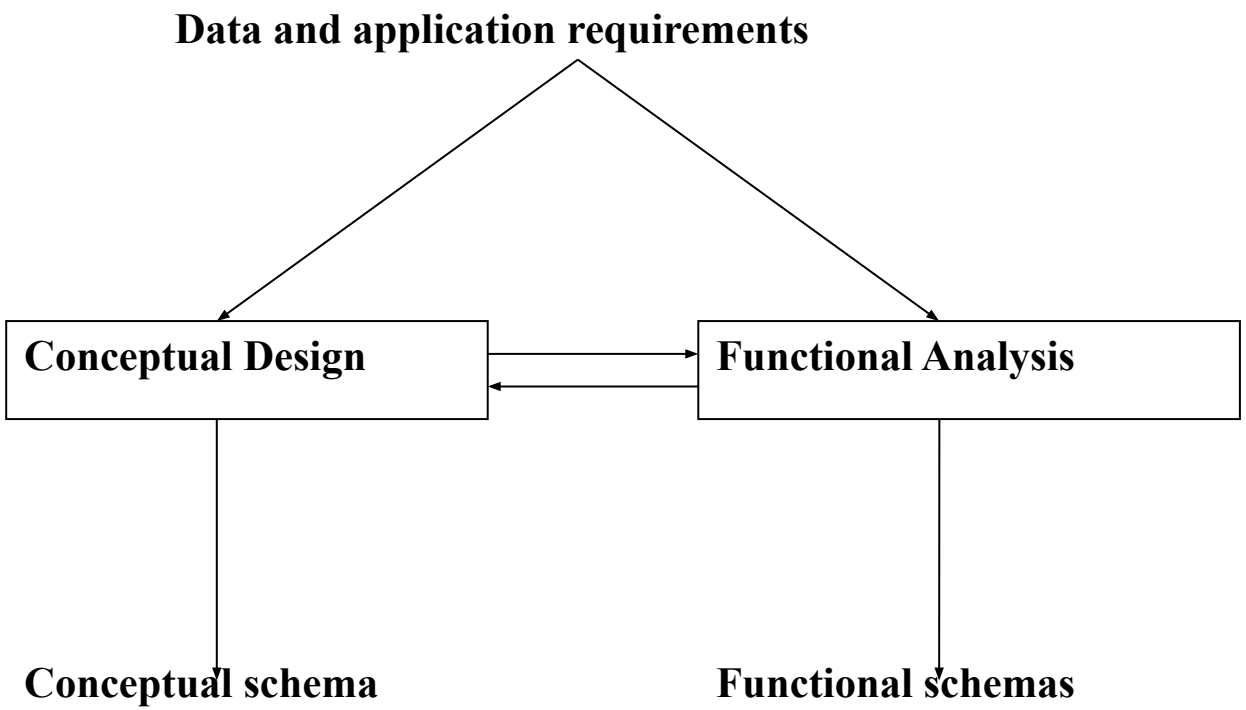


Dependence of DB design phases on the class of DBMS

-

Dependence on	**DBMS Class**	**Specific DBMS**
Conceptual design	No	NO
Logical design	Yes	No
Physical design	Yes	Yes

Joint data- and function- driven approach to information systems design



Bibliography

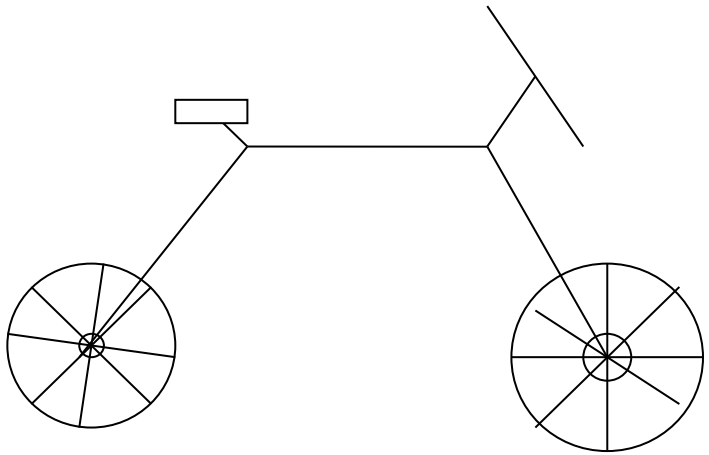
- 1. W.Davis System Analysis and Design : A structured Approach . Addison-Wesley 1983
- 2. R.Farley Software engineering Concepts. VcGraw Hills 1985
- 3. A.Cardenas Data Base Management System. 2 ed. Allyn and Bacon 1985

- **Data Modeling Concepts**

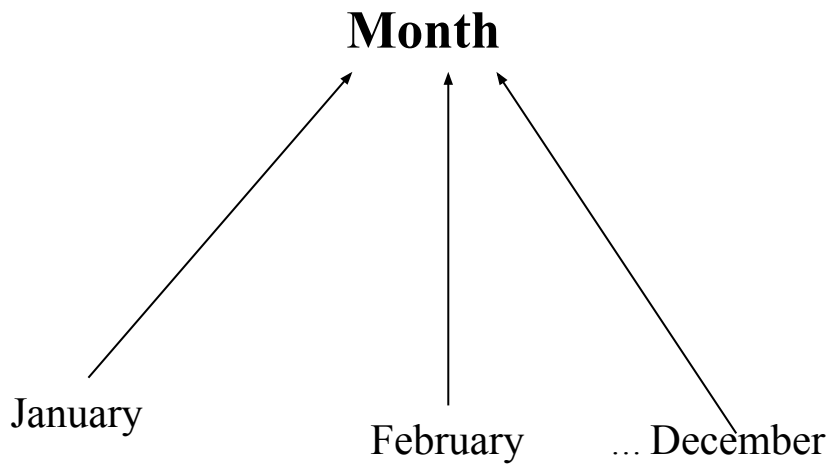
Structure of the lecture

- Section 1- Abstractions
- Section 2- Properties of mapping
- Section 3- Data models, Schemas, Instances of DB
- Section 4- ER Model
- Section 5- How to read an ER-schema

Abstractions in Conceptual Data Design

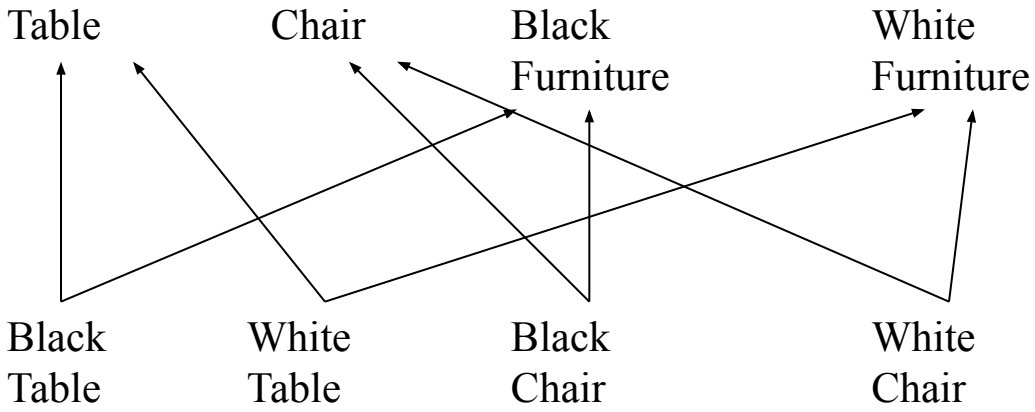
- 

Classification Abstraction

- 

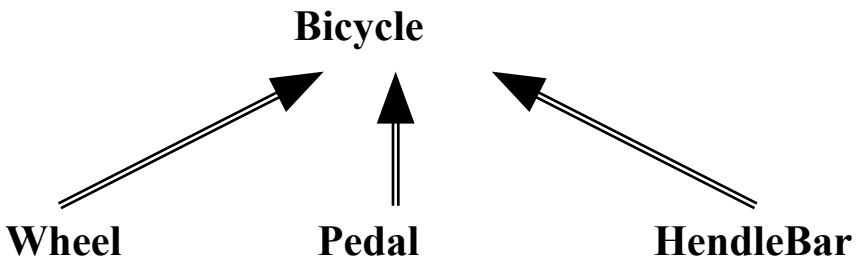
- The Classification Abstraction is used for defining as class of real world objects characterized by common properties

- One level tree having as its root class
- A Node leaf is a member of root class
- {black chair, Black table, White chair, White table}



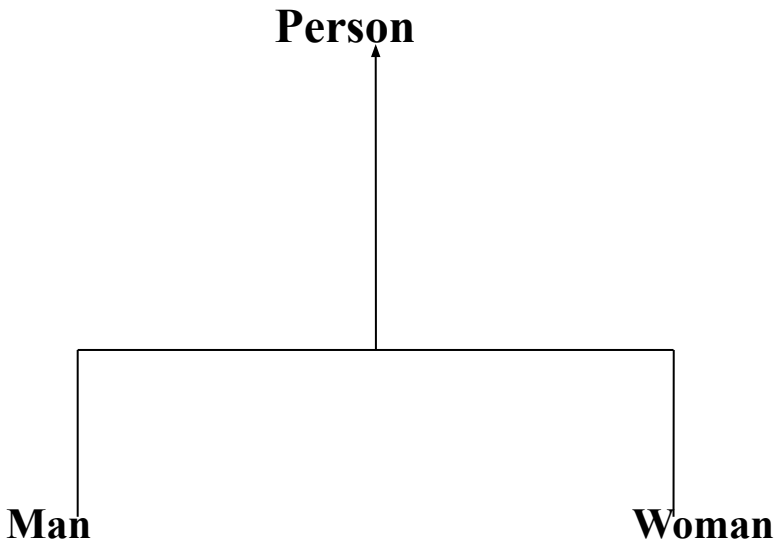
Aggregation Abstraction

- Aggregation abstraction defines a new class from set of (other) classes that representits component parts
- Leaf IS PART OF root class (is A)



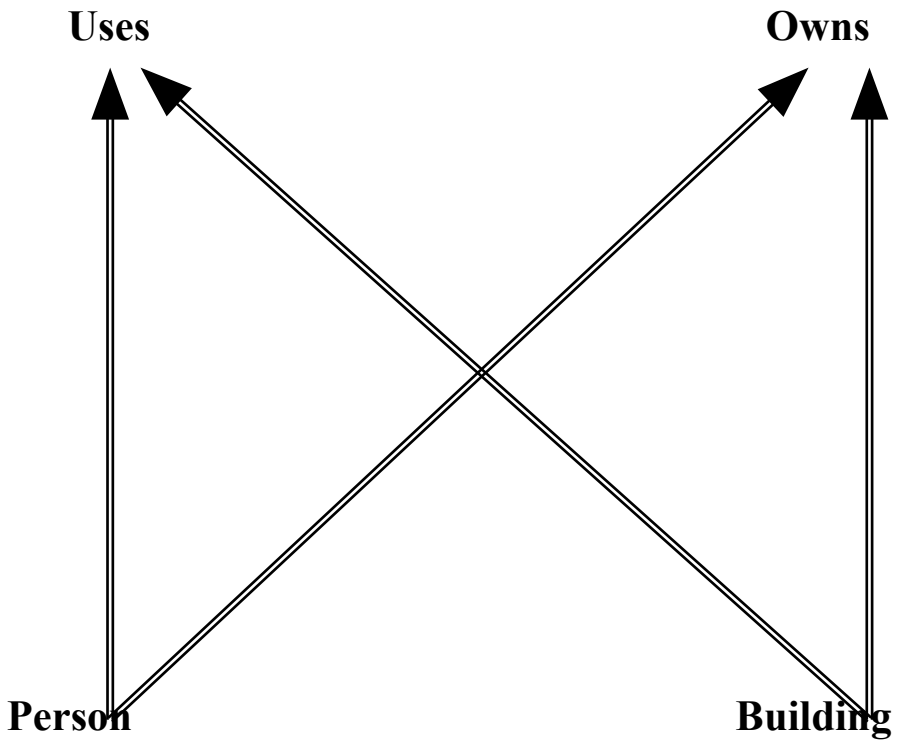
Generalization Abstraction

- A generalization abstraction defines a subset relationship between the elements of two or more classes In generalization all the abstractions defined for the generic class are inherited by all the subset classes

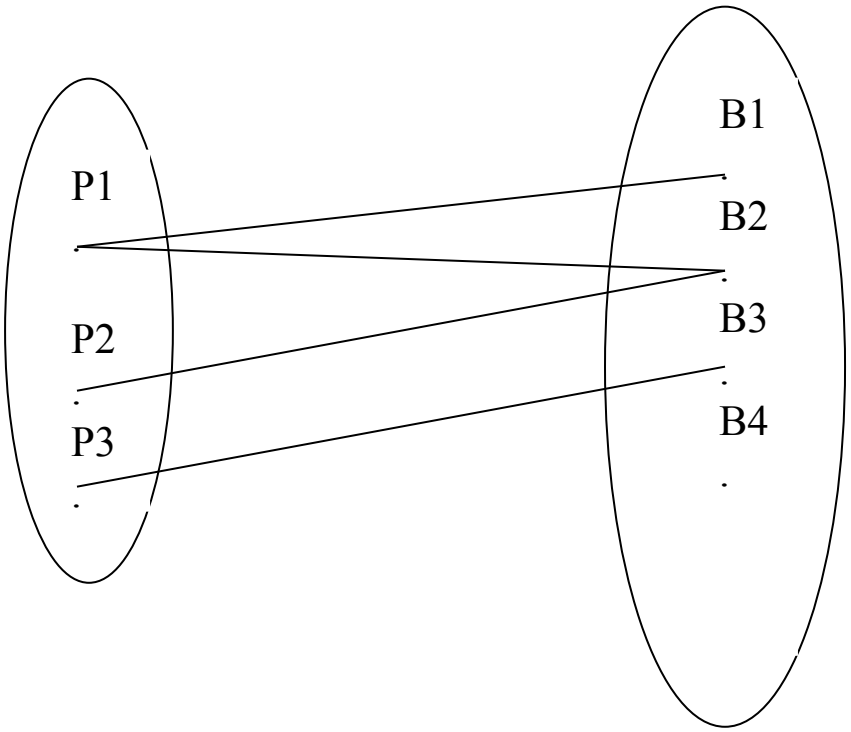


Properties of Mapping

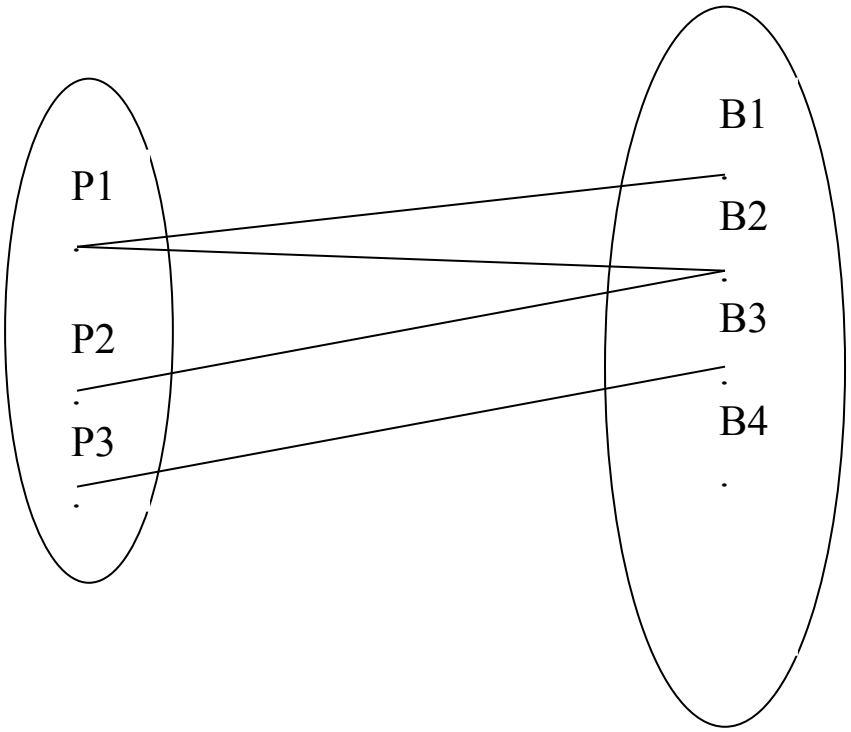
- A Binary aggregation is a mapping established between two classes.



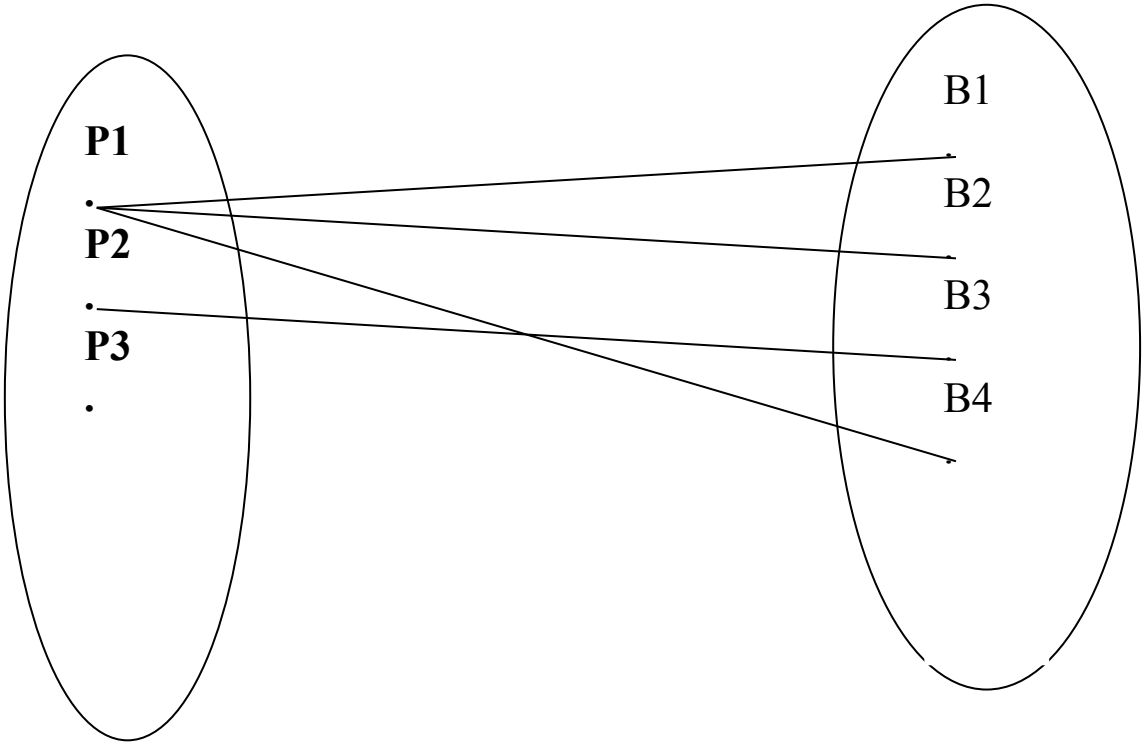
Binary aggregation USES

- 

Binary aggregation OWNS



Binary aggregation OWNS



Minimal cardinality (min-card)

- Let us consider aggregation A between classes C1 and C2

The minimal cardinality or min-card of C1 in A denoted min-card(C1,A), is the minimum number of mappings in which every element of C1 can participate.

Similarly, the min-card of C2 in A, denoted min-card(C2,A), is the minimum number of mappings in which each element of C2 can participate

Maximal Cardinality

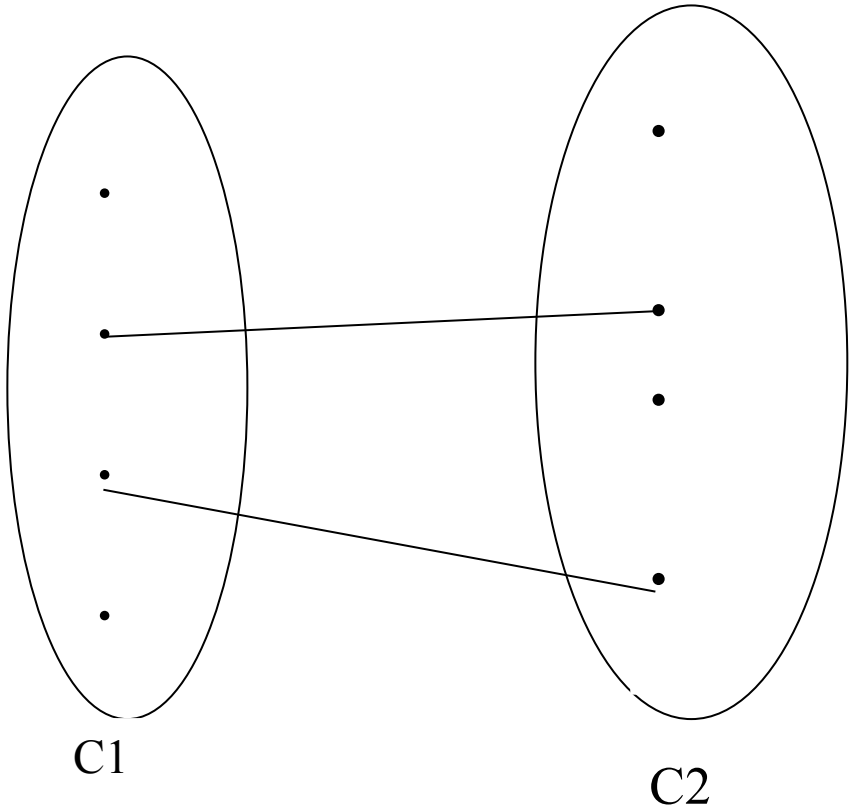
- Let us consider aggregation A between classes C1 and C2

The maximal cardinality or max-card of C1 in A denoted max-card(C1,A), is the maximum number of mappings in which every element of C1 can participate.

Similarly, the max-card of C2 in A, denoted max-card(C2,A), is the maximum number of mappings in which each element of C2 can participate

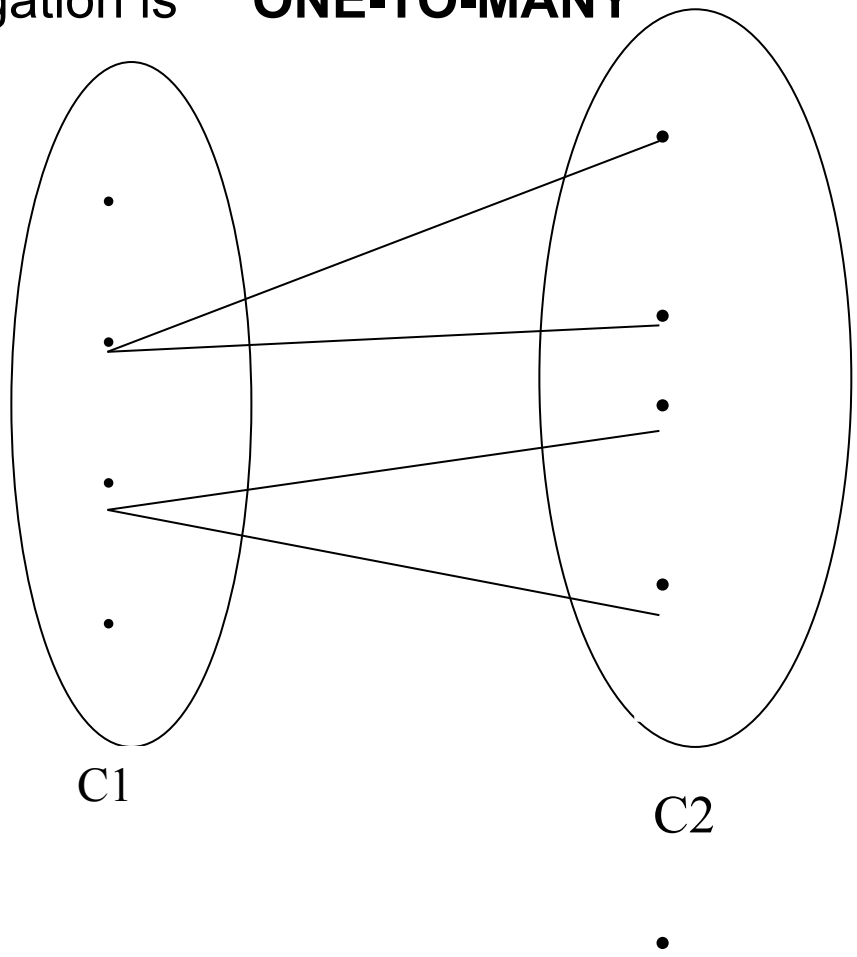
One –to-one mapping

- If max-card(C1,A)=1 and max-card(C2,A)=1 then we say that the aggregation is **ONE-TO-ONE**



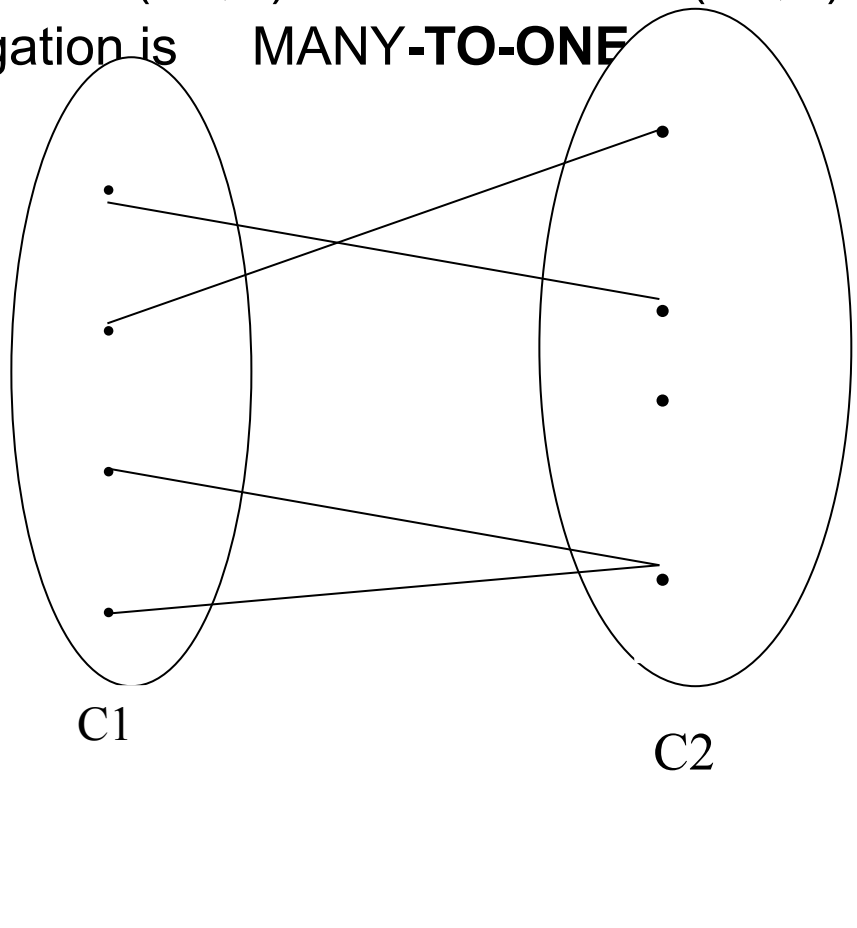
One to many mapping

- If max-card(C1,A)=n and max-card(C2,A)=1 then we say that the aggregation is **ONE-TO-MANY**



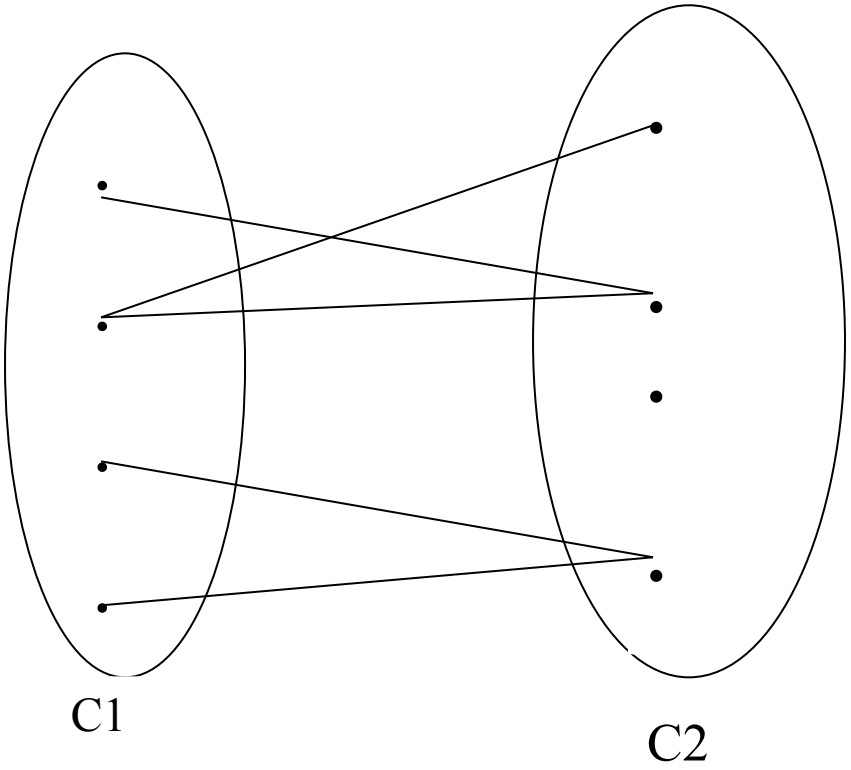
Many –to –one mapping

- If max-card(C1,A)=1 and max-card(C2,A)=n then we say that the aggregation is MANY**-TO-ONE**



Many –to-many mapping

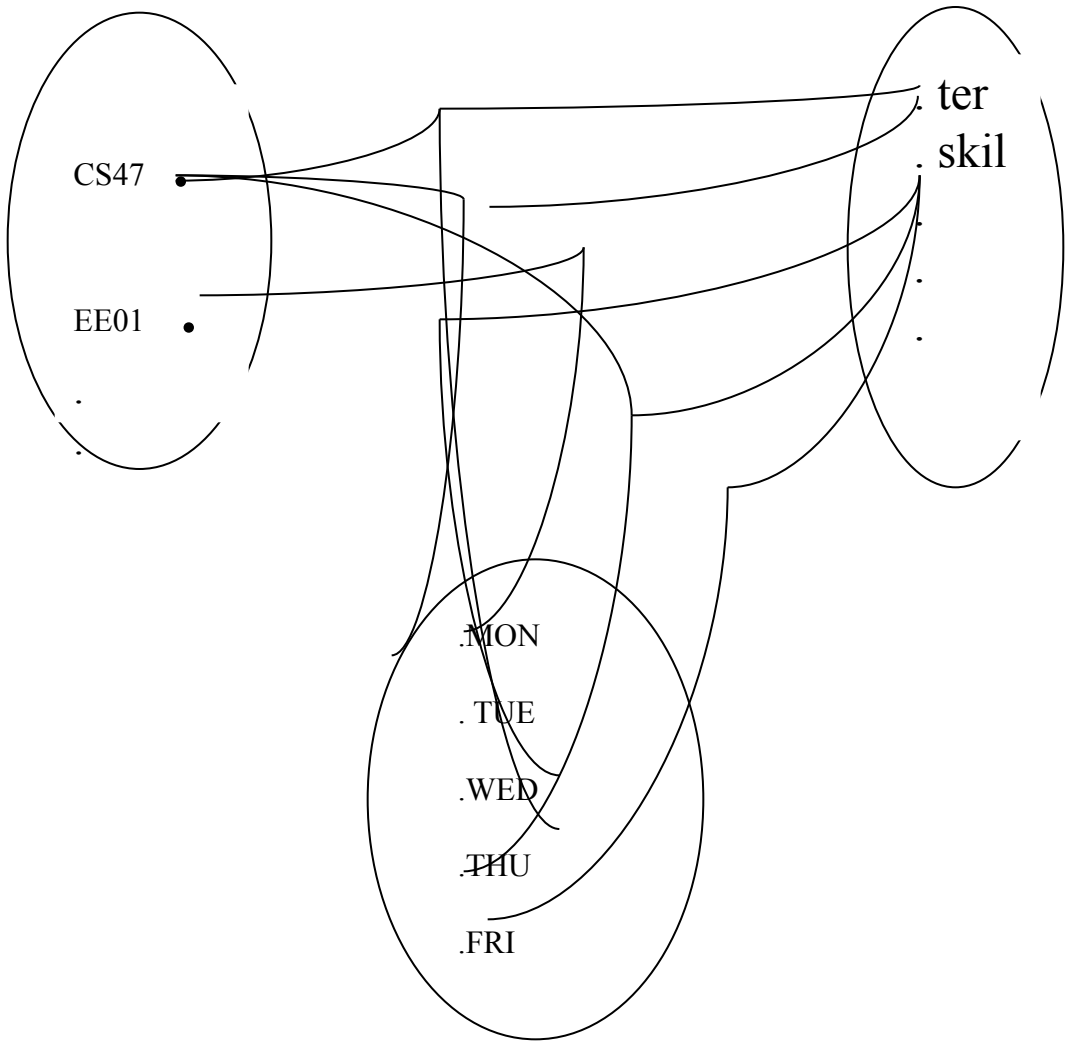
- If max-card(C1,A)=m and max-card(C2,A)=n (m, n>1), then we say that the aggregation is MANY**-TO-MANY**



N-ary aggregation

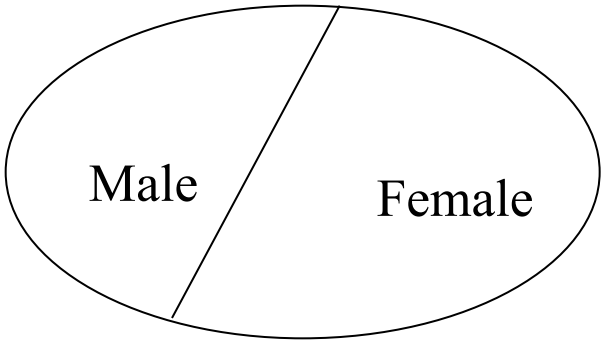
- **An n-ary aggregation is a mapping established among three or more classes**
- **Minimal Cardinality** (min-card) Let us consider the aggregation A between classes C1,C2,…,Cn The min-card of Ci in A is minimal number of mappings in which each element of Ci can participate
- **Maximal Cardinality** (max-card) Let us consider the aggregation A between classes C1,C2,…,Cn. The max-card of Ci in A is maximum number of mappings, in which each element of Ci can participate
- The two values of minimal and maximal cardinality completly characterize each participation of one class in aggregation

Representation of ternary aggregation Meets



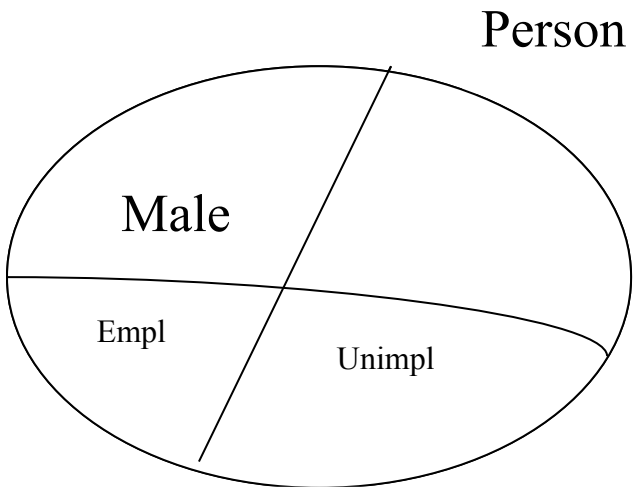
Generalization

- **A Generalization Abstraction** Person es the mapping from generic class to the subset class



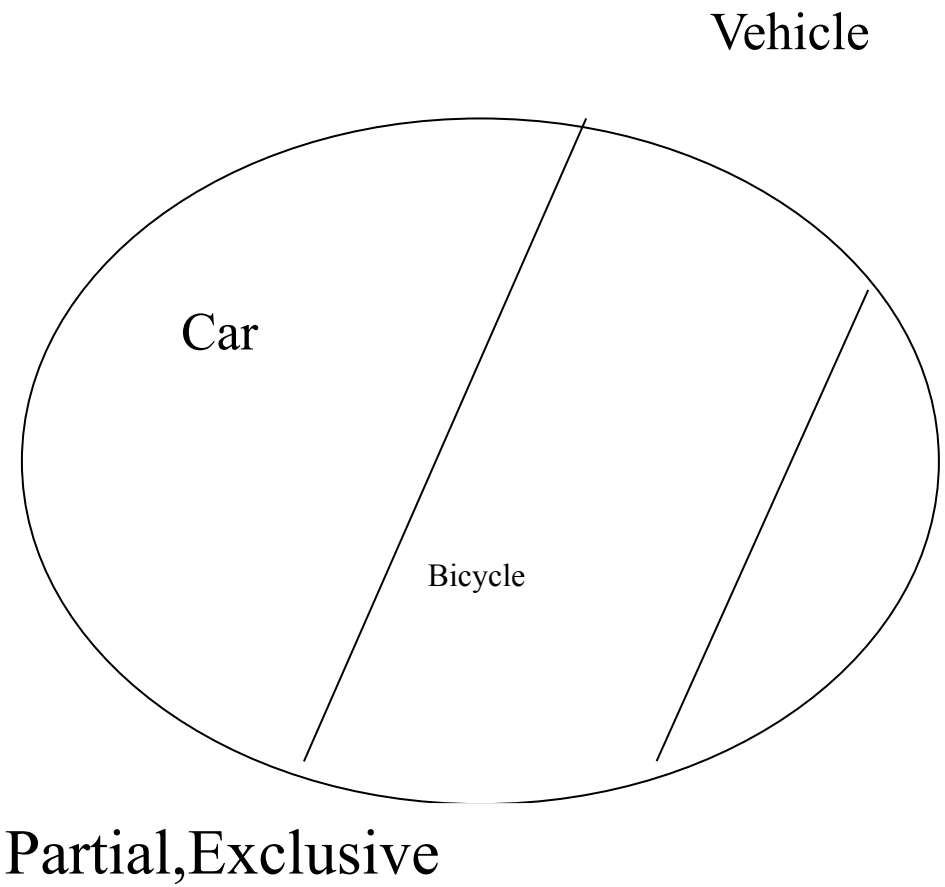
Total, Exclusive

Partial, overlapping generalization



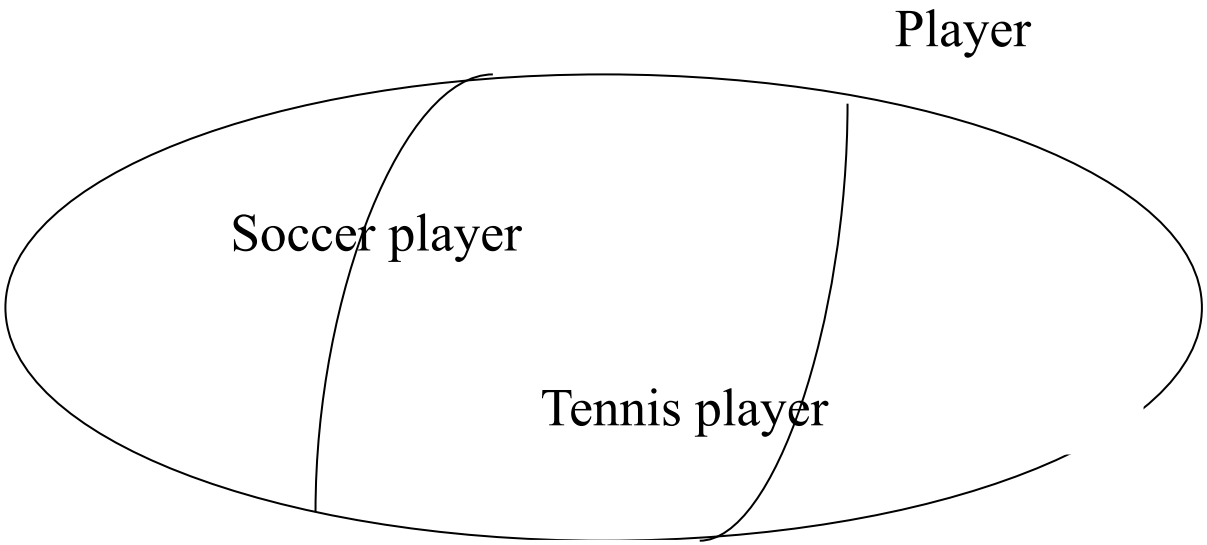
Partial,Overlapping

Partial, Exclusive Generalization

- 

Partial,Exclusive

Total, overlapping generalization

- 

Total, Overlapping

Data models

- **A Data model** is a collection of concepts that can be used to describe a set of data and operations to manipulate the data.

- When a data model describes a set of concepts from a given reality,

It is called **a conceptual data model**

The Concepts in data model are typically by using abstraction mechanisms and are described through linguistic and graphic representations . Syntax can be defined fnd a graphical notation can be developed as parts of data model.

Conceptual model is tool for representing reality at high level of abstraction

Logical models support data descriptions that can be processed by computer. They include *hierarchical, network, relational models*

These models to phisical structure of database

Schema

- Schema is a representation of a specific portion of reality, built using a particular data model
- Schema is static, time – invariant collection of linguistic or graphic representations that describe the structure of data of interest such as what within one organization

Person

NAME	SEX	ADDRESS	SOCIAL SECURITY NUMBER

Car

PLATE	MAKE	COLOUR

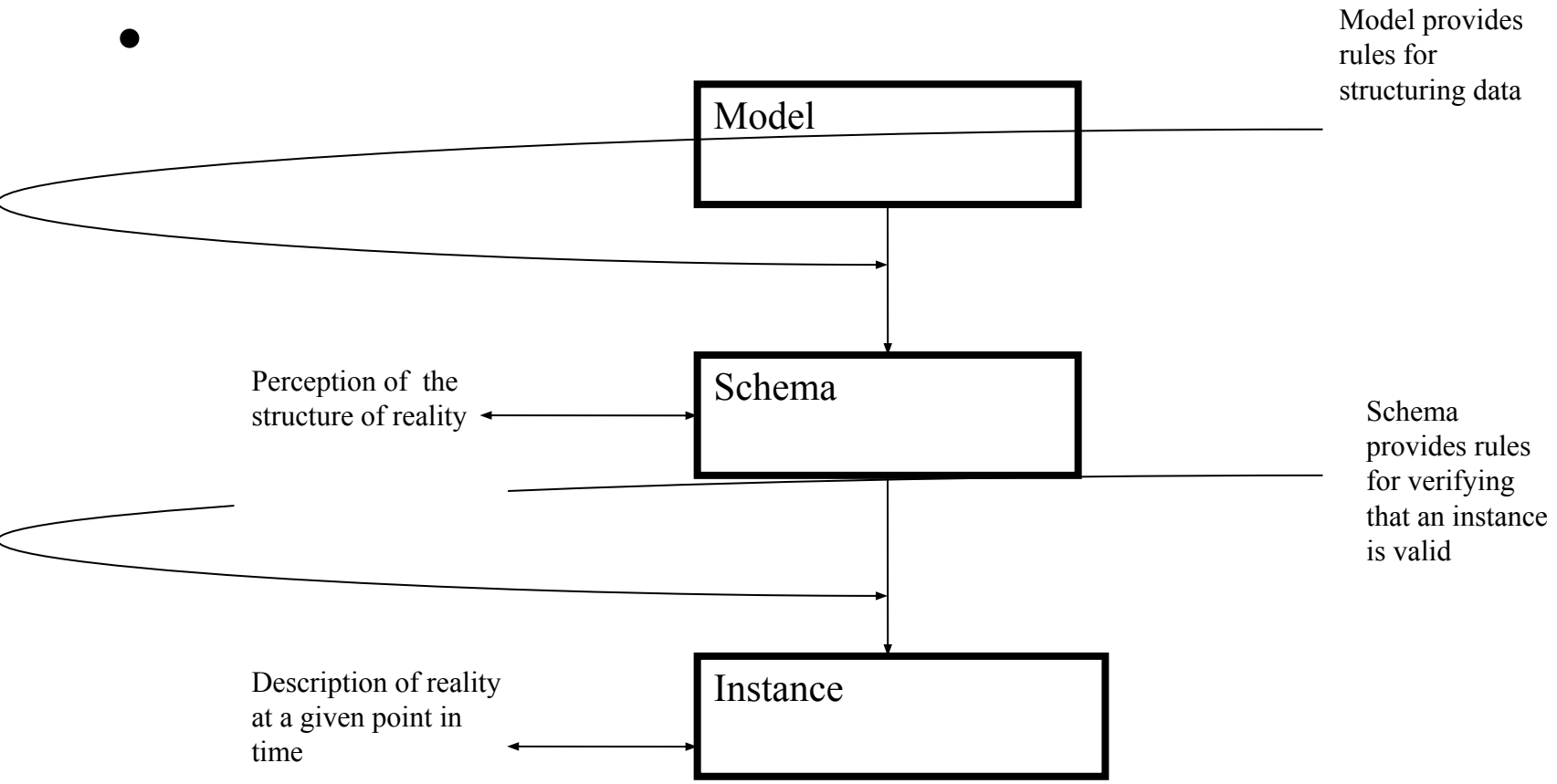
SOCIAL SECURITY NUMBER	PLATE

SAMPLE SCHEMA

Instances

- **An instance of schema is a dynamic, time variant collection of data that conforms to the structure of data defined by the schema**
- **Sample instance**
- **PERSON**
- JOHN SMITHM 11WEST12.,FT.LAUDERDALE 387-6713-362
- MARY SMITHF 11WEST12.,FT.LAUDERDALE 389-4816-381
- JOHN DOLEM 11RAMONA ST. PAOLO ALTO 391-3873-132
- **CAR**
- CA13718 MASERATI WHITE
- FL18MIAI PORSCHE BLUE
- CA CATA17 DATSUN WHITE
- FL 171899 FORD RED
- **OWNS**
- 387-6713-362 FL 18MIAI
- 387-6713-362 FL171899
- 391-3873-132 CA13718
- 391-3873-132 CA CATA17
- **Sample instance after insertion**
- **CAR**
- CA13718 MASERATI WHITE
- FL18MIAI PORSCHE BLUE
- CA CATA17 DATSUN WHITE
- FL171899 FORD RED
- NY BABYBLUE FERRARI RED
- **OWNS**
- **387-6713-362 FL18MIAI**
- **387-6713-362 FL171899**
- **391-23873-132 CA13718**
- **391-32873-132 CA CATA17**
- **389-4816-381 NY BABYBLUE**

Relationships between model, schema, instance



Qualities of Conceptual Models

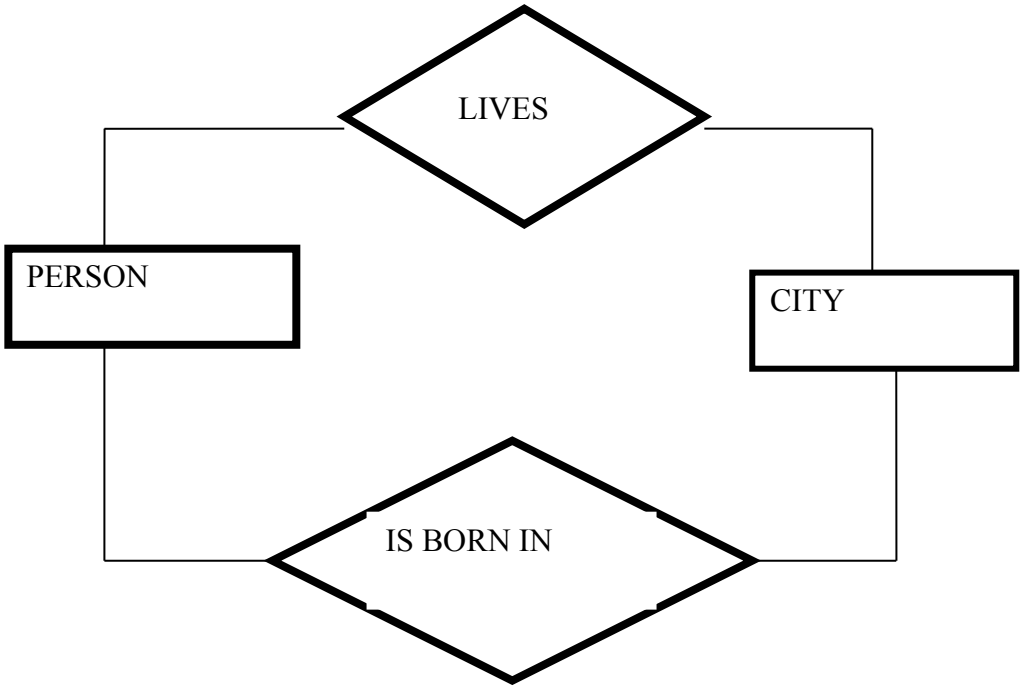
- 1. Expressiveness
- 2. Simplicity
- 3. Minimality
- 4. Formality
- **PROPERTIES OF GRAPHIC REPRESENTATIONS**
- 1.Graphic Completeness
- 2. Ease of Reading

The Entity –Relationship Model

- **Basic elements of the ER Model**
- **Entities.** Entities represent classes of real world objects
- **Relationships.** Relationships aggregation of two or more entities
- Binary and n-ary relationships
- **Rings –** are binary relationships connecting an entity to itself

(recursive relationships)

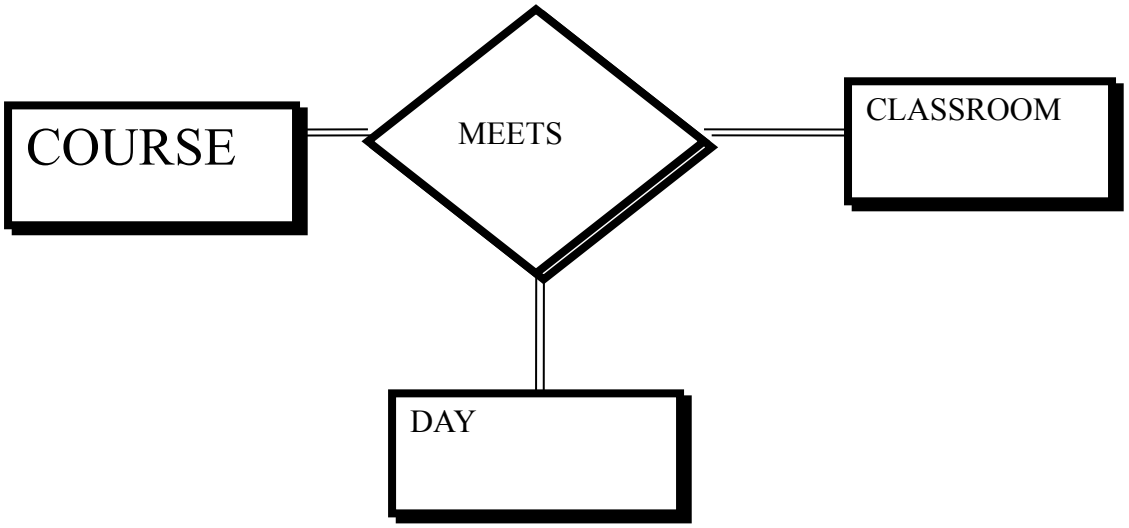
- Portion of ER-schema representing entities PERSON, CITY and relationships IS BORN IN and LIVES IN



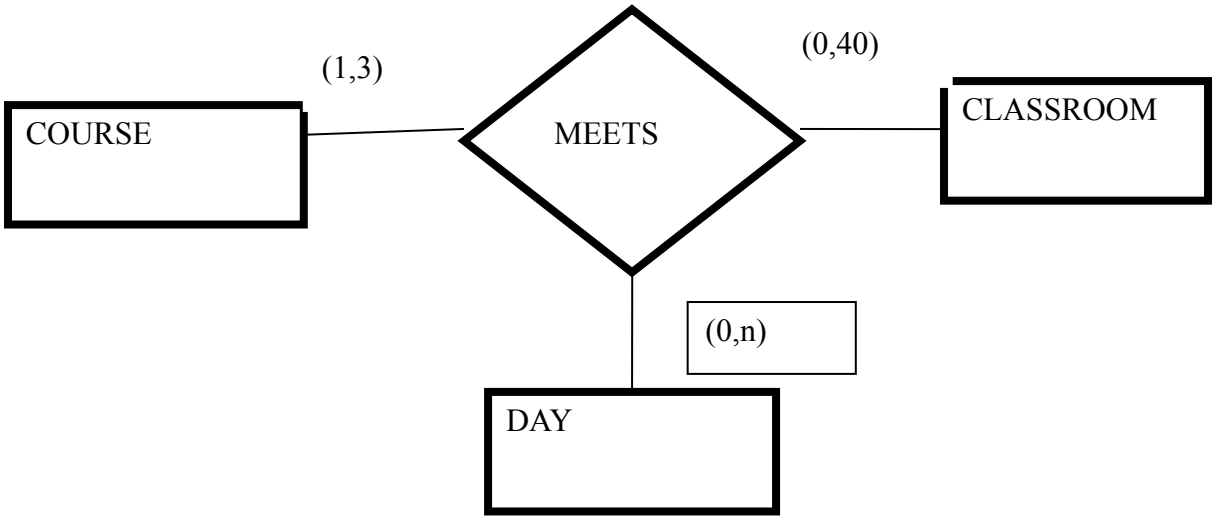
Instance for previous schema

- PERSON={p1,p2,p3}
- CITY= {c1,c2,c3}
- LIVES IN= { <p1,c1>,<p2,c3>,<p3,c3>}
- IS BORN IN= {<p1,c1>,<p3,c1>}

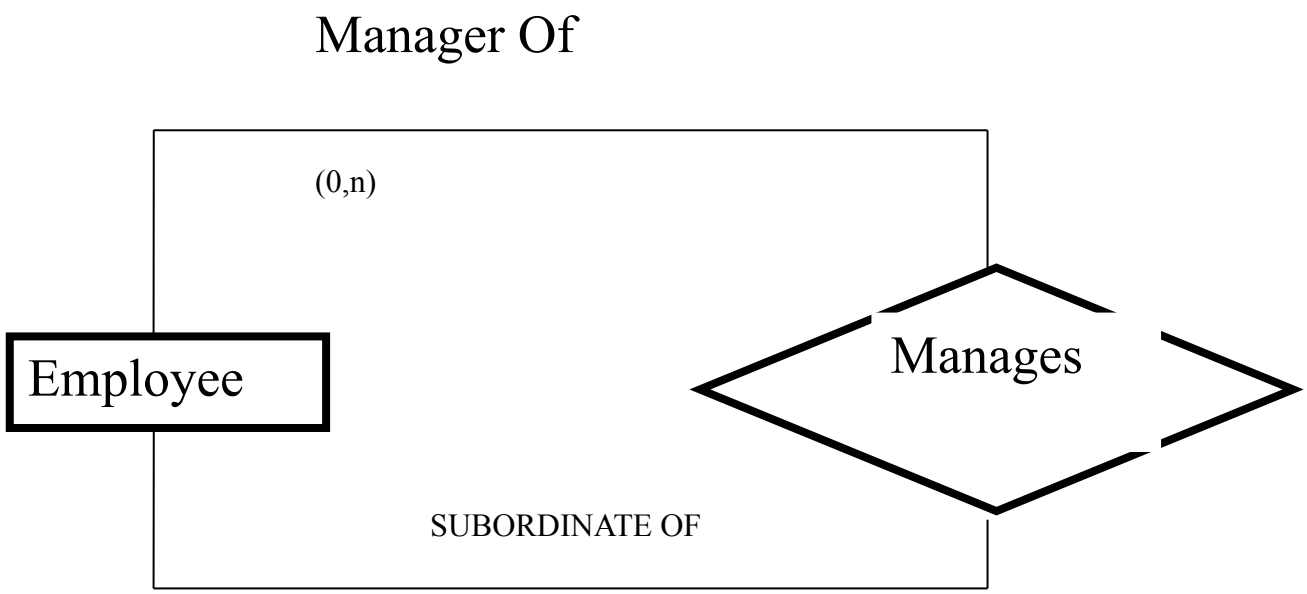
N-ary relationship MEETS



Relationship MEETS

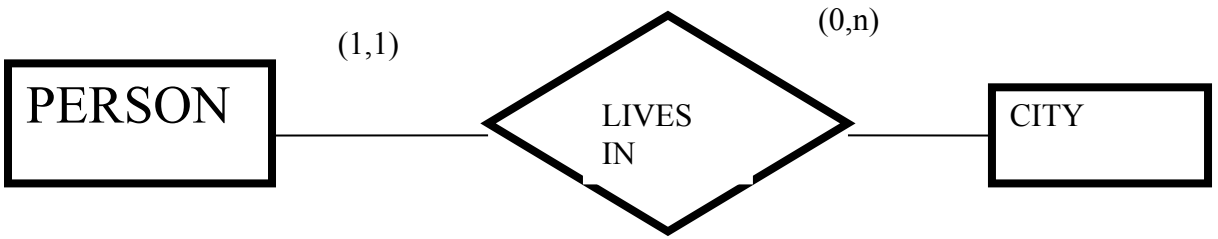


Relationship MANAGES

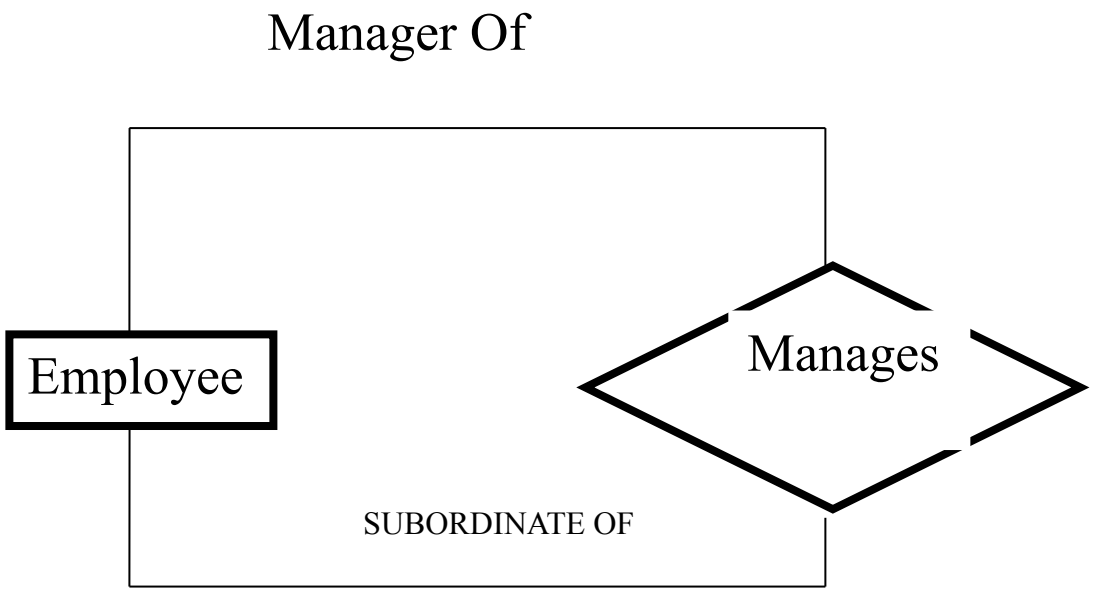


- min-card(PERSON,LIVES IN)=1
- max-card(PERSON,LIVES IN)=1
- min-card(CITY,LIVES IN)=0
- max-card(CITY,LIVES IN)=n

Relationship LIVES IN

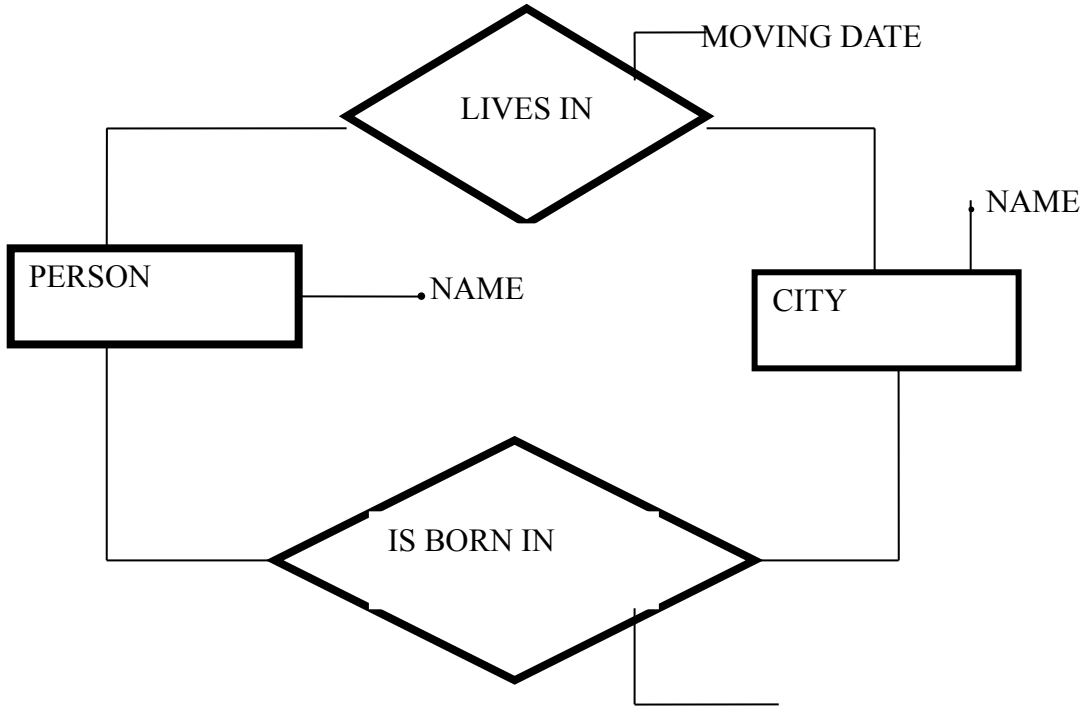


Ring relationship MANAGES

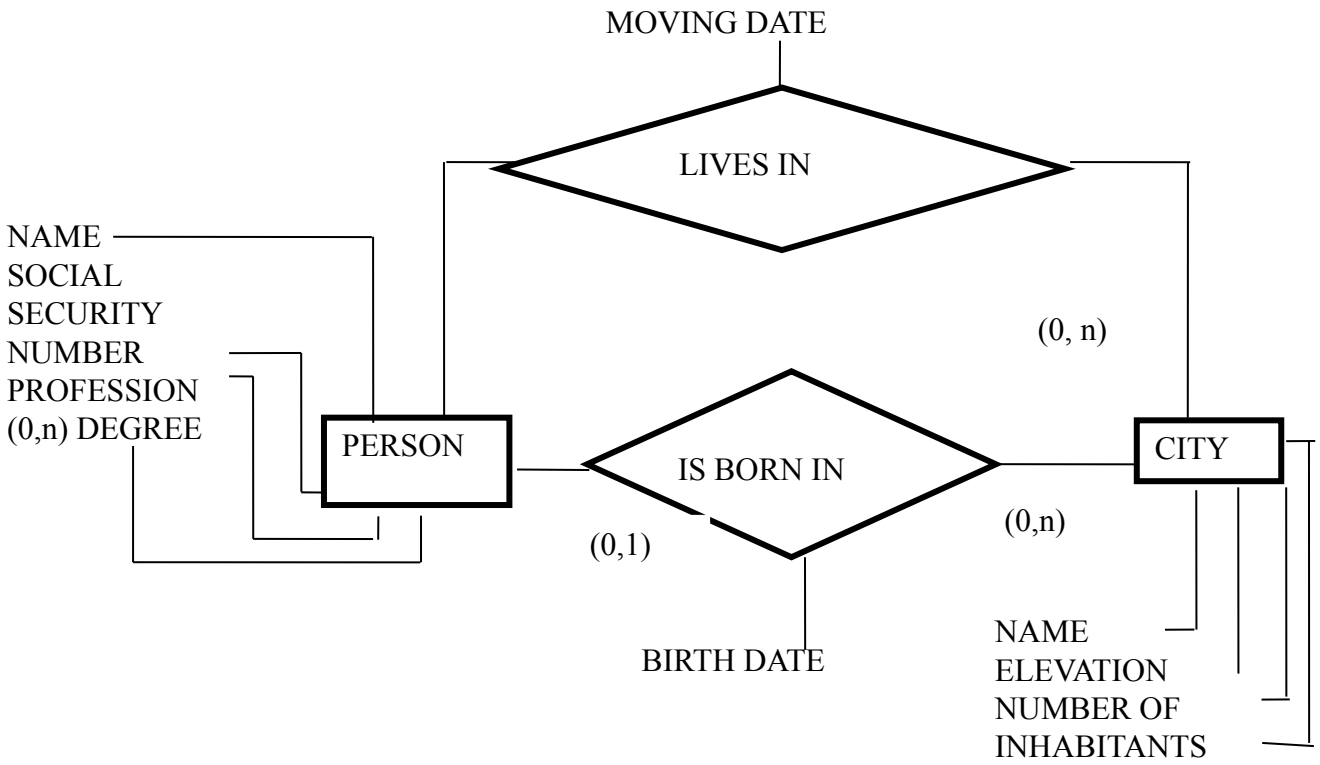


Attributes

- Attributes represent elementary properties of entities or relations



An ER schema with entities,relationships,attributs



An example of instance of DB schema

PERSON={p1:<JOHN,345-8798-564,STUDENT,()>,
p2:<SUE,675-6756-343, MGR,(M.S.,Eng,Ph.D.)>
P3:<MARTIN,676-453-8482,FARMER,(HS)>}
CITY={c1:<ROME,100,3000000>,
c2:<NEW-YORK,0,9000000>,
c3:<ATLANTA,100,2000000>}
LIVES IN={<p1,c1:<1-02-80>>,
<p2,c3:<7-23-83>>
<p3,c3:<6-04-81>>}
IS BORN IN={<p1,c1:<1-05-55>>
<p3,c1:<6-14-35>>}

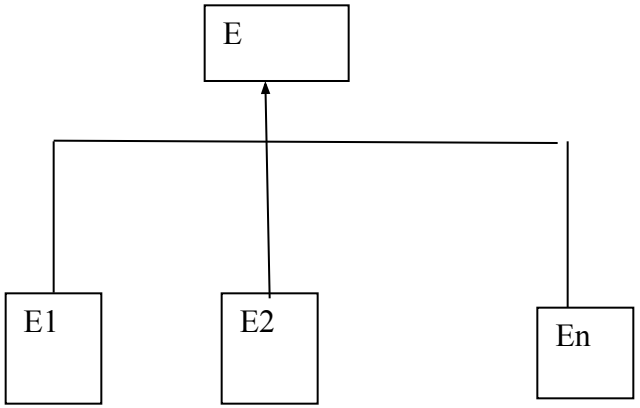
Schema PERSONNEL

- Schema : PERSONNEL
- Entity: PERSON
- Attributes: NAME: text (50)
- SOCIAL SECURITY NUMBER: text (12)
- PROFESSION: text (20)
- (0,n) DEGREE: text (20)
- Entity: CITY
- Attributes: NAME: text (30)
- ELEVATION: integer
- NUMBER OF INHABITANTS: integer
- Relationship:LIVES IN
- Connected entities: (0,n) CITY
- (1,1) PERSON
- Attributes: MOVING DATE: date

- Relationship: IS BORN IN
- Connected entities: (0,n) CITY
- (0,1) PERSON
- Attributes: BIRTH DATE: date

Generalization Hierarchies

- In the ER model it is possible to establish generalization hierarchies between entities
- An entity E is generalization of a group of entities E1,E2,…,En if each object of classes E1,E2,…,En is also object of class

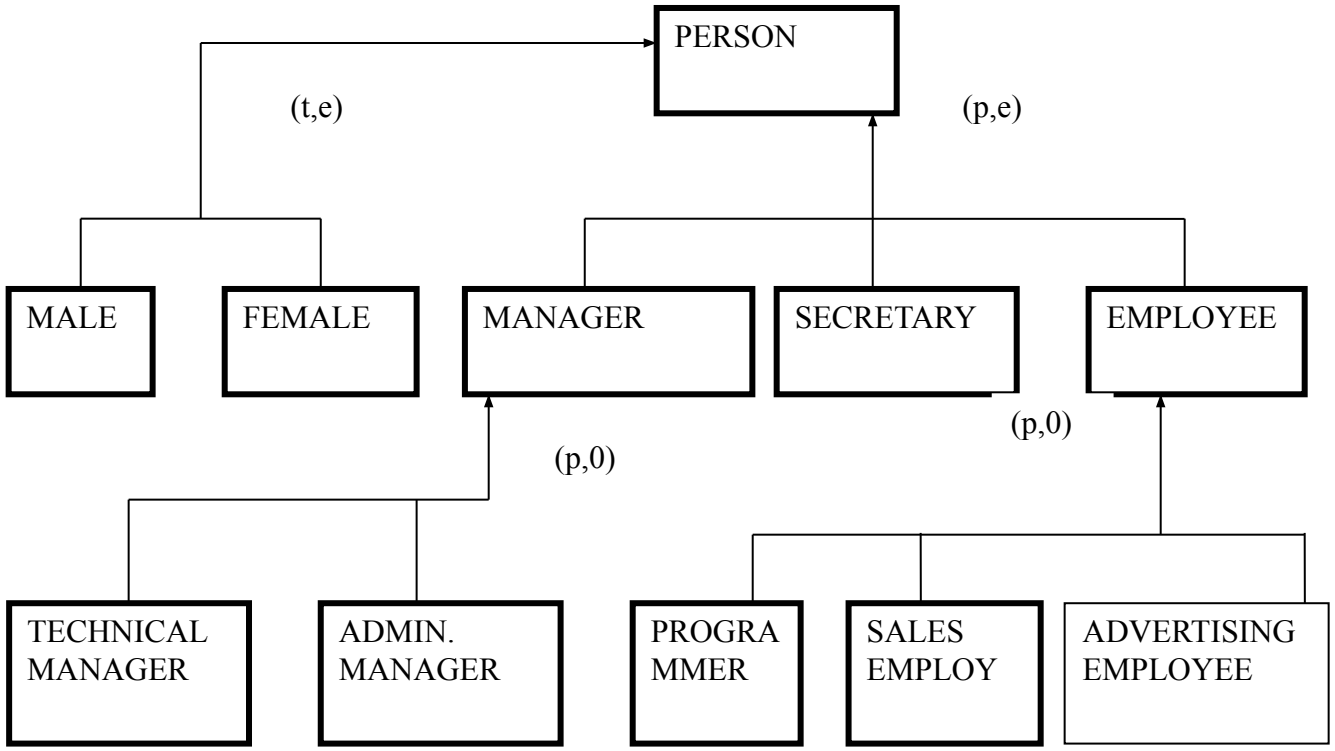


COVERAGE:

- Total generalization (t)
- Partial generalization (p)
- Exclusive (e)
- Overlapping (o)

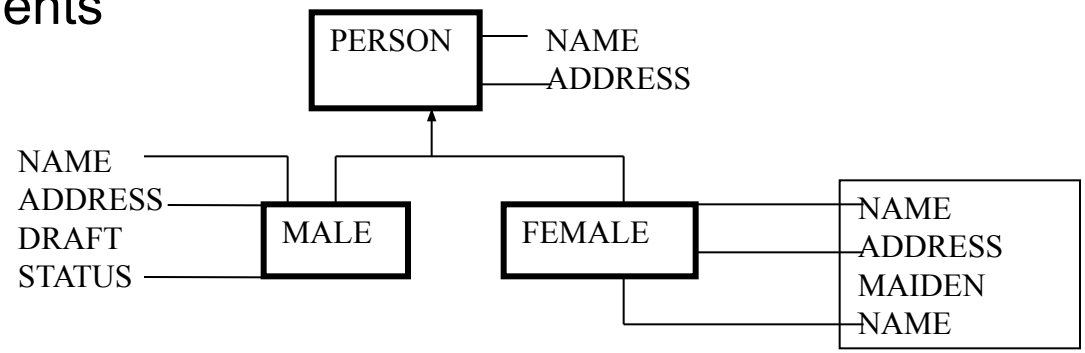
Pair: (t,e) the most frequently used

Generalization hierarchy for entity PERSON

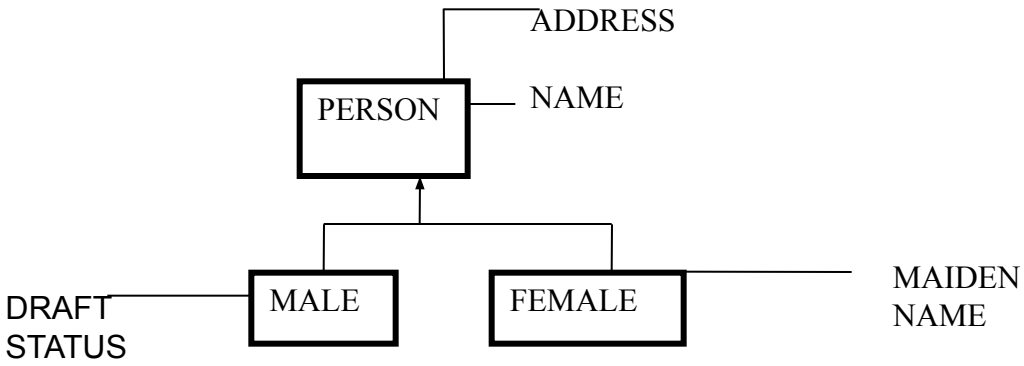


Inheritance

- All the properties of the generic entity are inherited by the subset elements



Incorrect representation

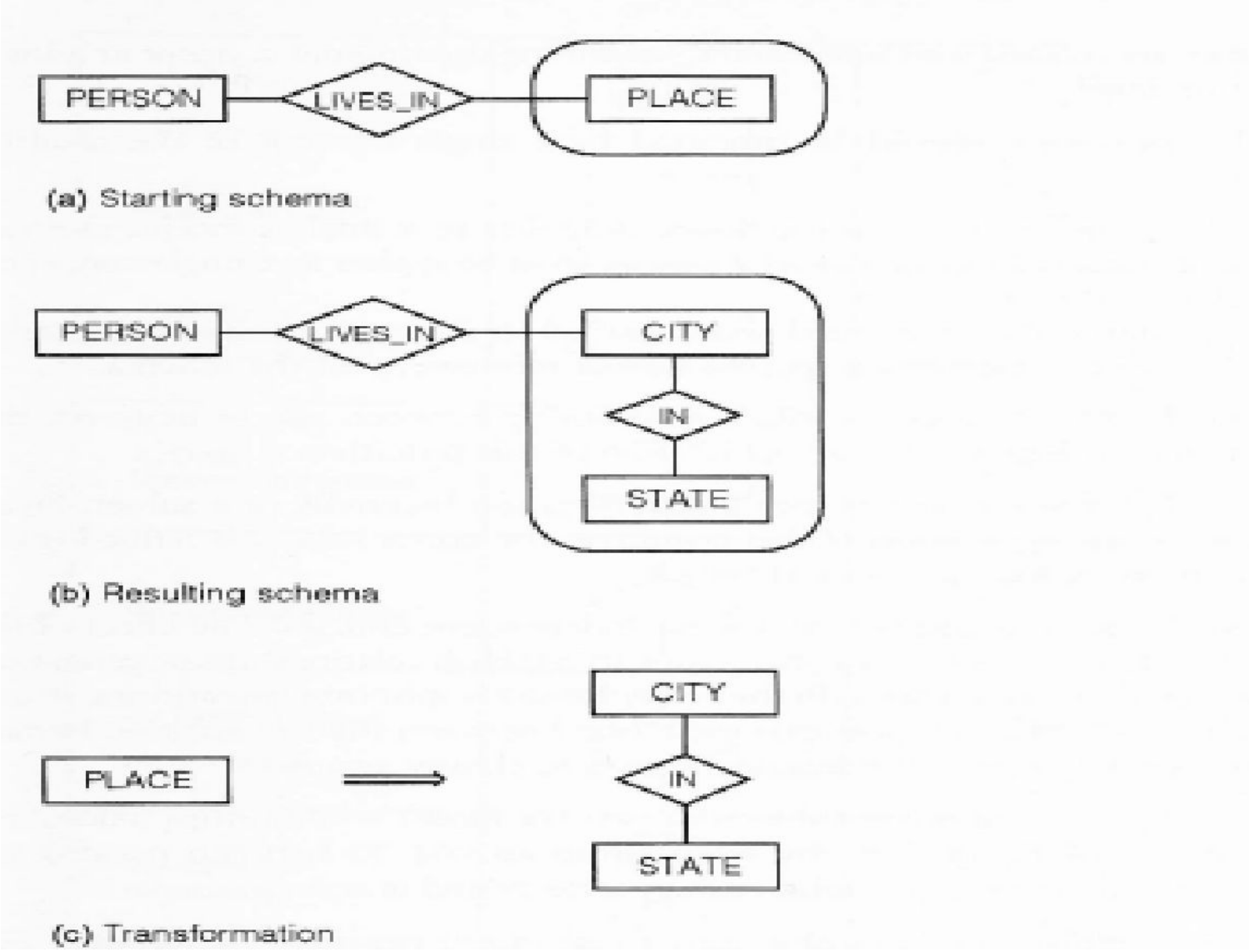


Correct representation

Formal definition of Inheritance

- Let E be an entity. Let A1, A2,…,An be single valued, mandatory attributes of E. Let E1, E2,…,En be other entities connected to E by mandatory, one –to-one or many-to-one binary relationships R1,R2,…,Rn (i.e.
- min-card(E,Ri)=1)) Consider as a possible identifier the set I= {A1,…,An,E1,…,Em}, 0 ≤n, 0 ≤m, 1 ≤n+m
- The value of the identifier for a particular entity instance is defined as the collection of all values of attributes A1,..,An and all instances of entities Ej,
- j=1,…,m connected to e with i ≤n, j <m
- Because of the assumption of considering mandatory single- valued attributes or mandatory relationships with max-card set to each instance of E is mapped either to one value of attributes Ai or to one instance of entity Ej, i ≤n, j ≤m

Exammpe of schema transformation

- 

- Each schema TRANSFORMATION has **a starting schema** and a **resulting schema**
- Each SCHEMA TRANSFORMATION maps names of concepts in starting schema to names of concepts in resulting schema.
- Concepts in the resulting schema must inherit all logical connections in the starting schema

Properties of top –down primitives

- They have a simple structure: the starting schema is a single concept, the resulting schema consists of small set of concepts.
- All names are refined into new names describing the original concept in the lower abstraction level
- Logical connections should be inherited by the single concept of the resulting schema

Top –Down Primitives

- 1. Primitive T_1 refines an entity into a relationship between two or more entities. The example in Figure 3.1 is an application of this primitive.

  2. Primitive T_2 refines an entity into a generalization hierarchy or a subset. Figure 3.3a presents an application of this primitive: the entity PERSON is refined into a generalization including MALE and FEMALE.

  3. Primitive T_3 splits an entity into a set of independent entities. The effect of this primitive is to introduce new entities, not to establish relationships or generalizations among them. In Figure 3.3b the entity AWARD is split into two entities, NOBEL_PRIZE and OSCAR. No relationship is established between the two entities, because they are two different and independent ways to classify awards.

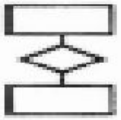
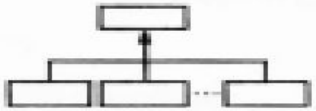
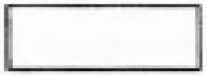
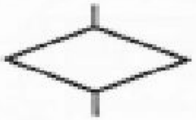
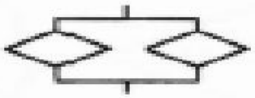
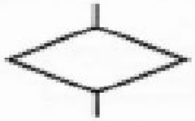
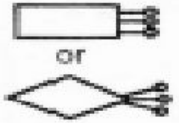
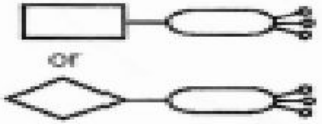
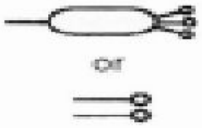
  4. Primitive T_4 refines a relationship into two (or more) relationships among the same entities. In Figure 3.3c the relationship RELATED_TO between persons and cities is refined into the two relationships LIVES_IN and IS_BORN_IN.

5. Primitive T_5 refines a relationship into a *path* of entities and relationships. Applying this primitive corresponds to recognizing that a relationship between two concepts should be expressed via a third concept, which was hidden in the previous representation. In Figure 3.3d the relationship WORKS_IN between EMPLOYEE and DEPARTMENT is refined into a more complex aggregation that includes the entity MANAGER and two new relationships.

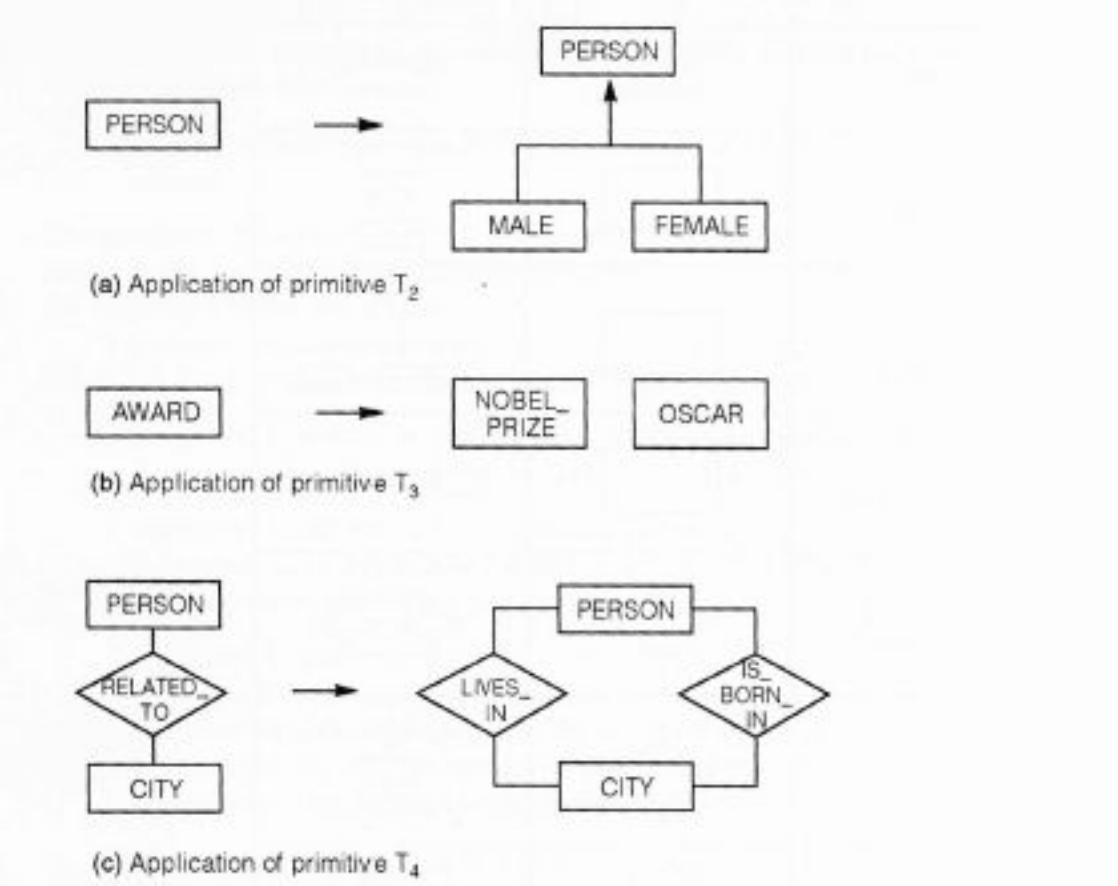
6. Primitive T_6 refines an entity (or a relationship) by introducing its attributes. In Figure 3.3e the attributes NAME, SEX, and AGE are generated for the entity PERSON.

7. Primitive T_7 refines an entity (or a relationship) by introducing a composite attribute. In Figure 3.3f the composite attribute ADDRESS is generated for the entity PERSON.

8. Primitive T_8 refines a simple attribute either into a composite attribute or into a group of attributes. In Figure 3.3g, the attribute DATE is refined into a composite attribute with three attributes: DAY, MONTH, and YEAR. The attribute HEALTH_DATE is refined in terms of the attributes HEALTH_STATE and DATE_LAST_VAOCINATION.

Primitive	Starting Schema	Resulting Schema
T_1: Entity → Related entities		
T_2: Entity → Generalization (Entity → Subset)		
T_3: Entity → Uncorrelated entities		
T_4: Relationship → Parallel relationships		
T_5: Relationship → Entity with relationships		
T_6: Attribute development		
T_7: Composite attribute development		
T_8: Attribute refinement		

Application of top-down primitives



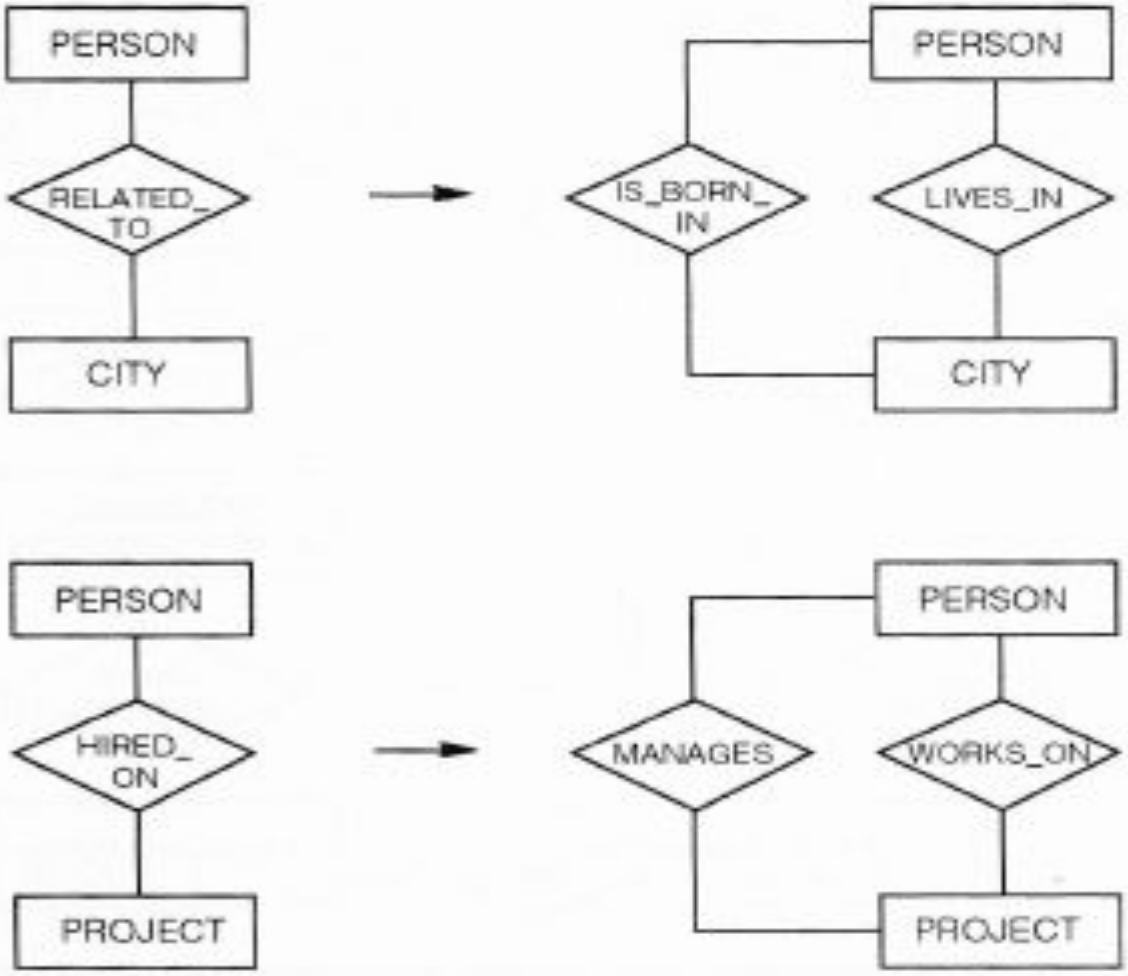
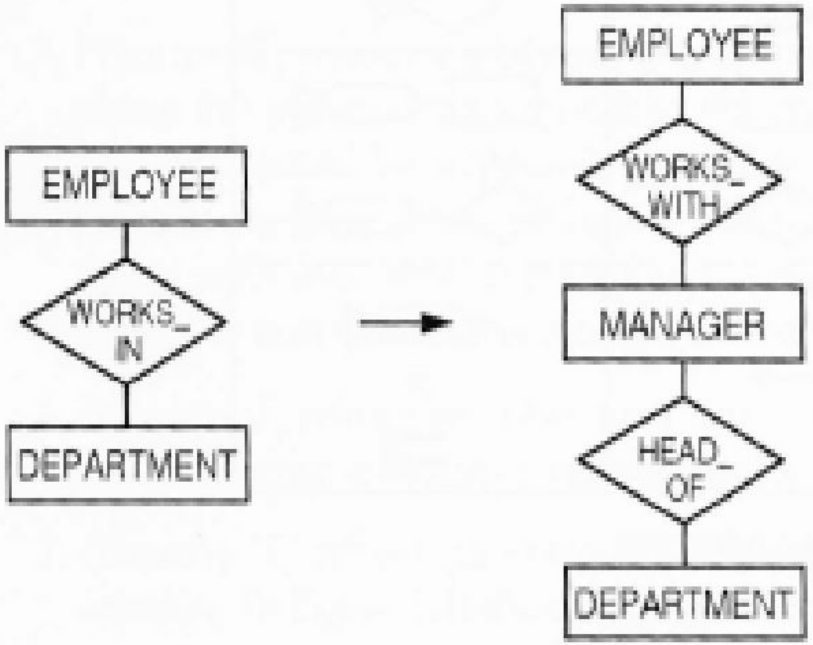
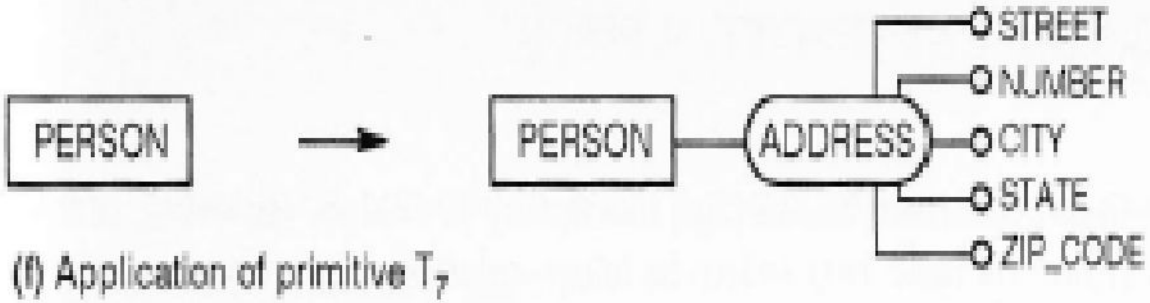


Figure 3.5 Two applications of primitive T_4

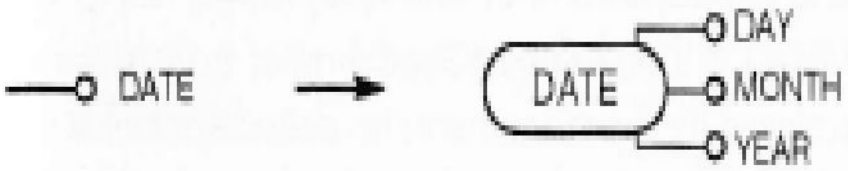


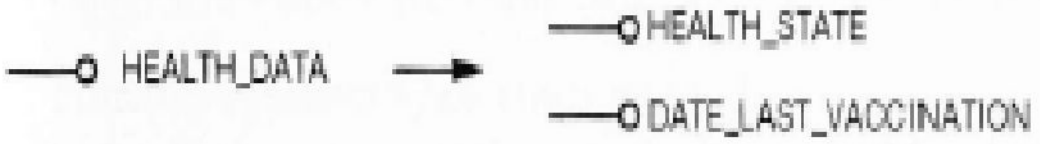
(d) Application of primitive T_5

(e) Application of primitive T_6



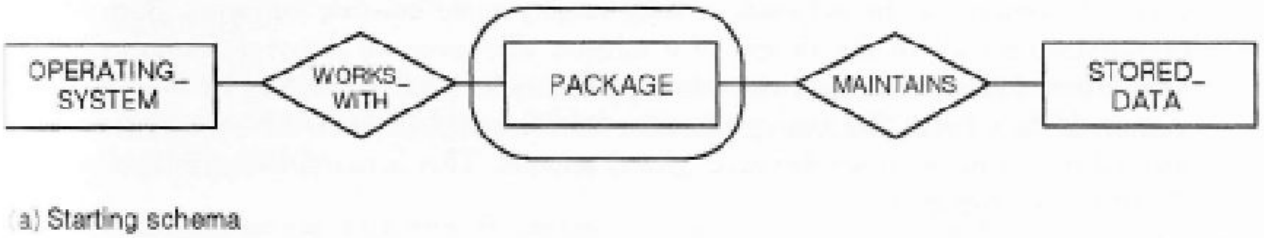
(f) Application of primitive T_7



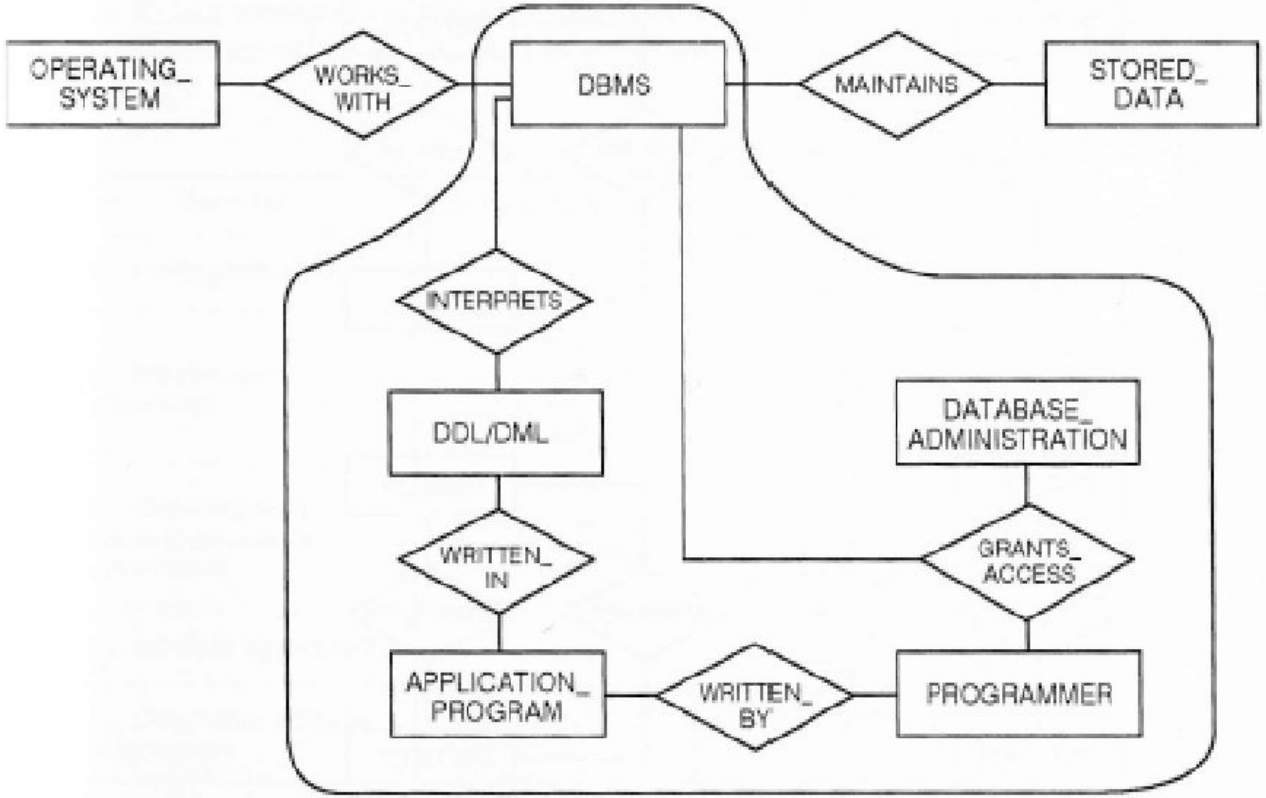


(g) Applications of primitive T_8

Applying of complex top –down schema transformation



(a) Starting schema



(b) Resulting schema

Bottom-up primitives

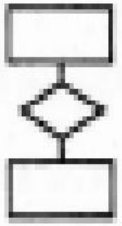
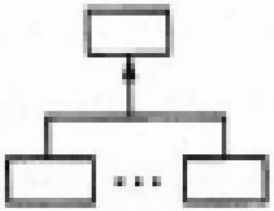
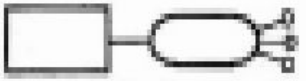
- 1. Primitive B_1 generates a new entity. This primitive is used when the designer discovers a new concept with specific properties that did not appear in the previous schema.

  2. Primitive B_2 generates a new relationship between previously defined entities. In Figure 3.7a a new relationship LIVES_IN is established between PERSON and PLACE.

  3. Primitive B_3 creates a new entity that is elected as a generalization (either a subset or a generalization hierarchy) among previously defined entities. In Figure 3.7b a generalization hierarchy is generated, creating the new entity PERSON.

  4. Primitive B_4 generates a new attribute and connects it to a previously defined entity or relationship. In Figure 3.7c NAME, SEX, and AGE attributes are added to the entity PERSON.

  5. Primitive B_5 creates a composite attribute and connects it to a previously defined entity or relationship. In Figure 3.7d the composite attribute ADDRESS is added to the entity PERSON.

Primitive	Starting Schema	Resulting Schema
B_1: Entity generation		
B_2: Relationship generation		
B_3: Generalization generation (subset generation)		
B_4: Attribute aggregation		
B_5: Composite attribute aggregation		

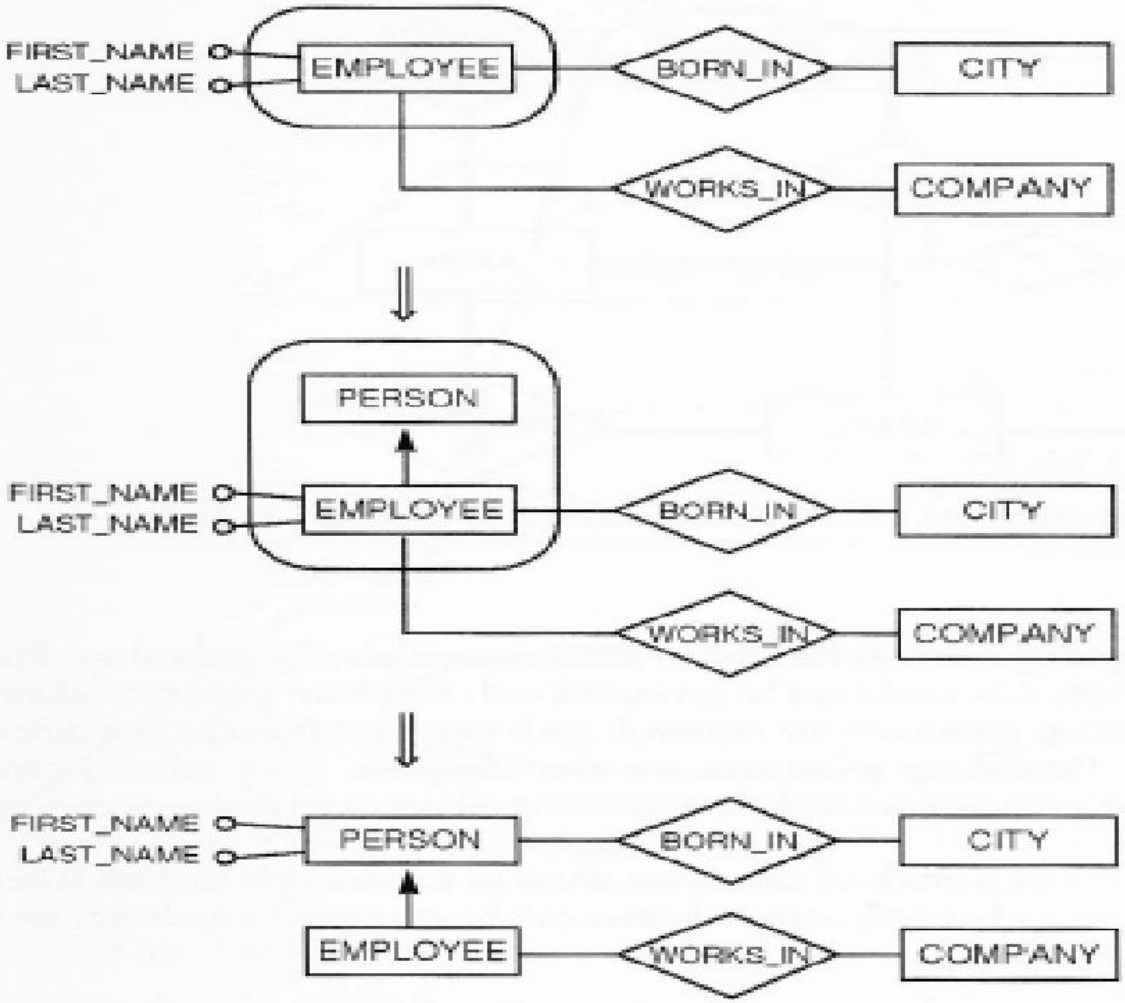


Figure 3.8 Migration of properties after applying a bottom-up primitive

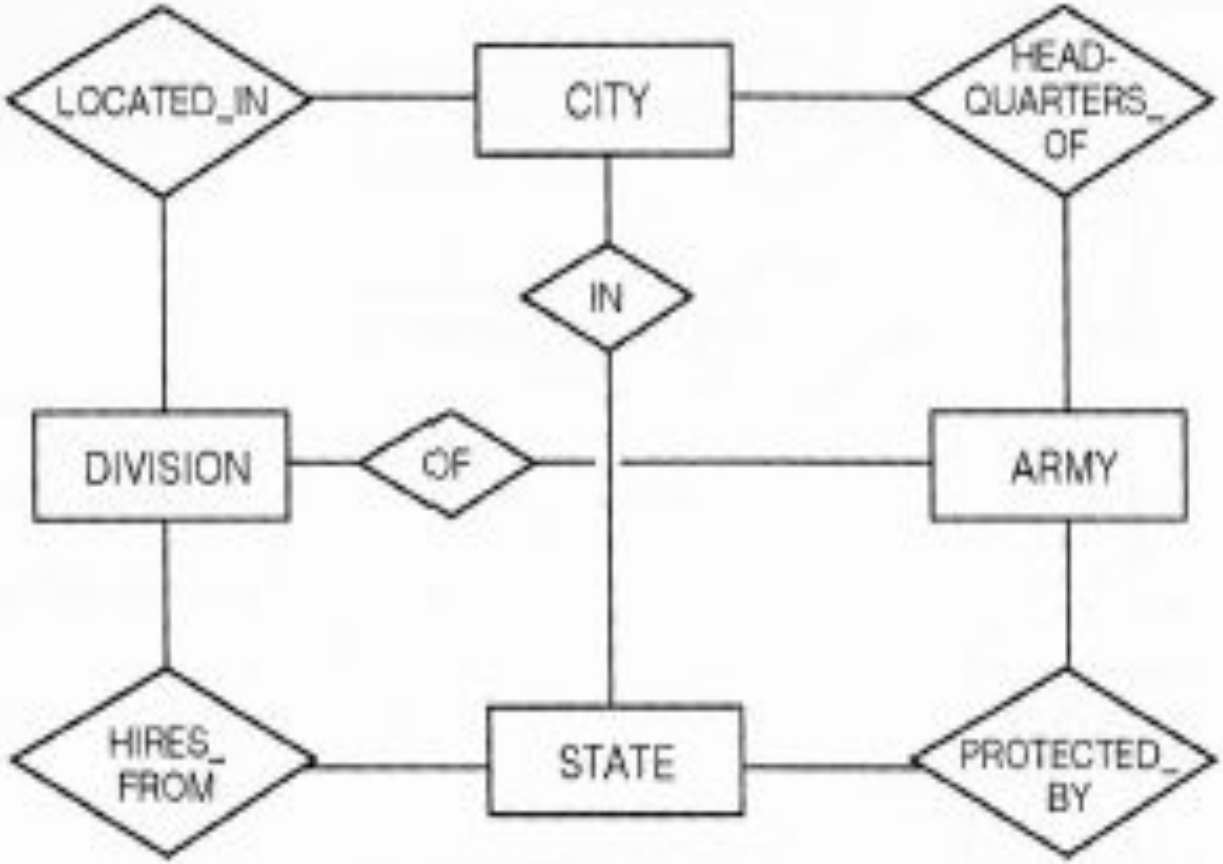


Figure 3.9 Example of schema that cannot be generated by applying top-down primitives

- Strategies for Schema Design

Top-down strategy

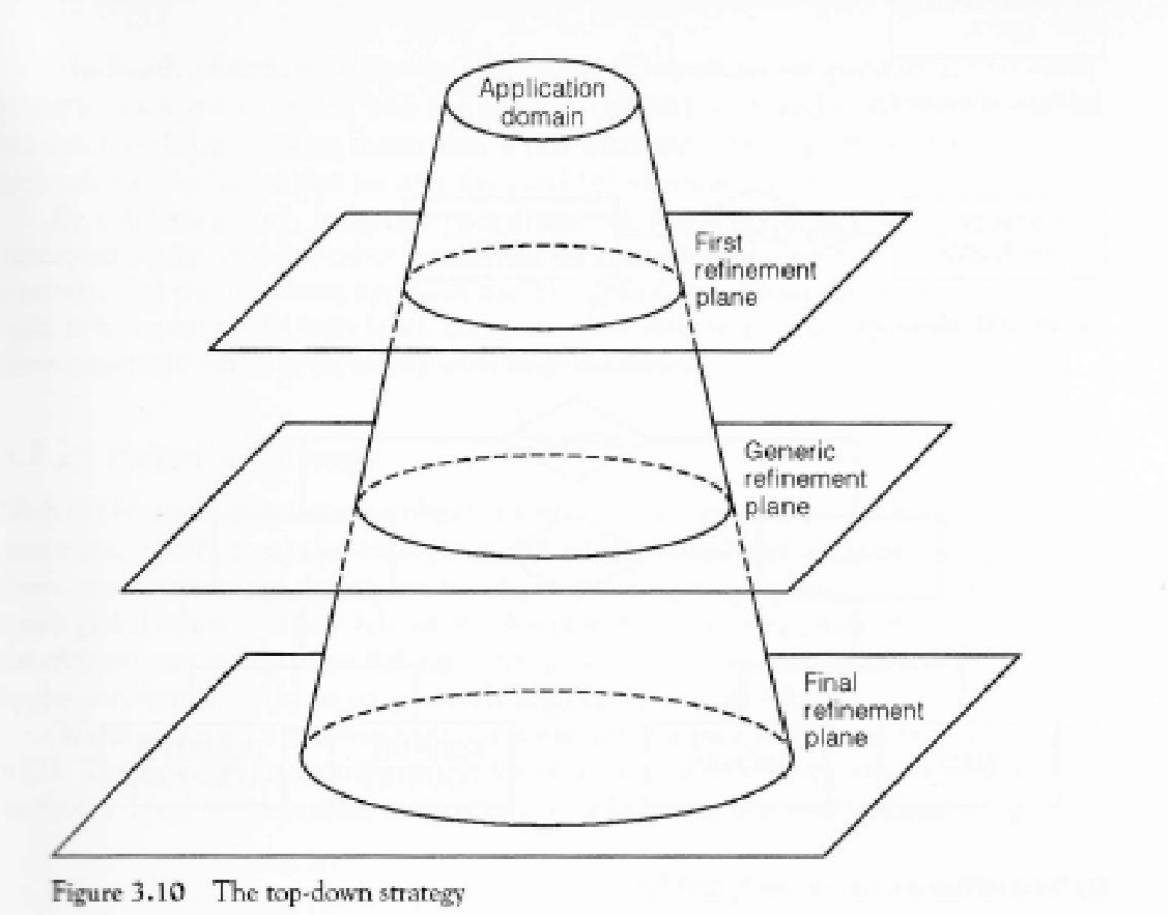
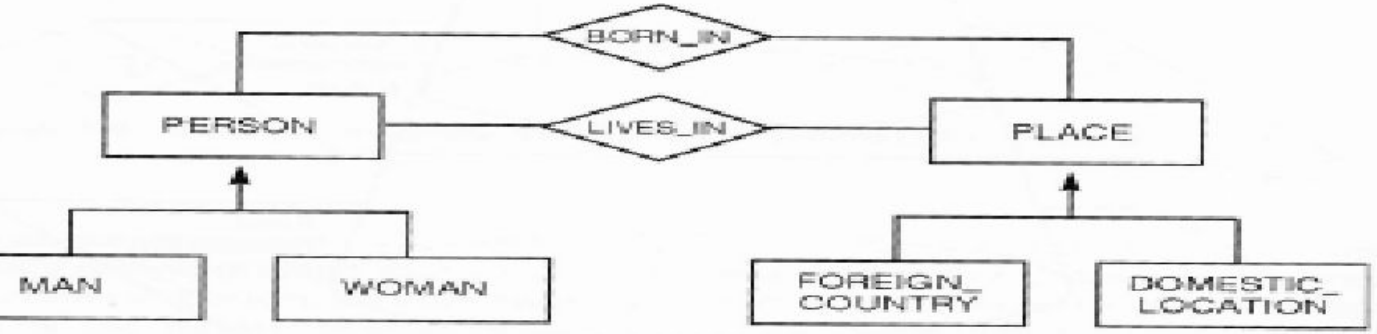
- 

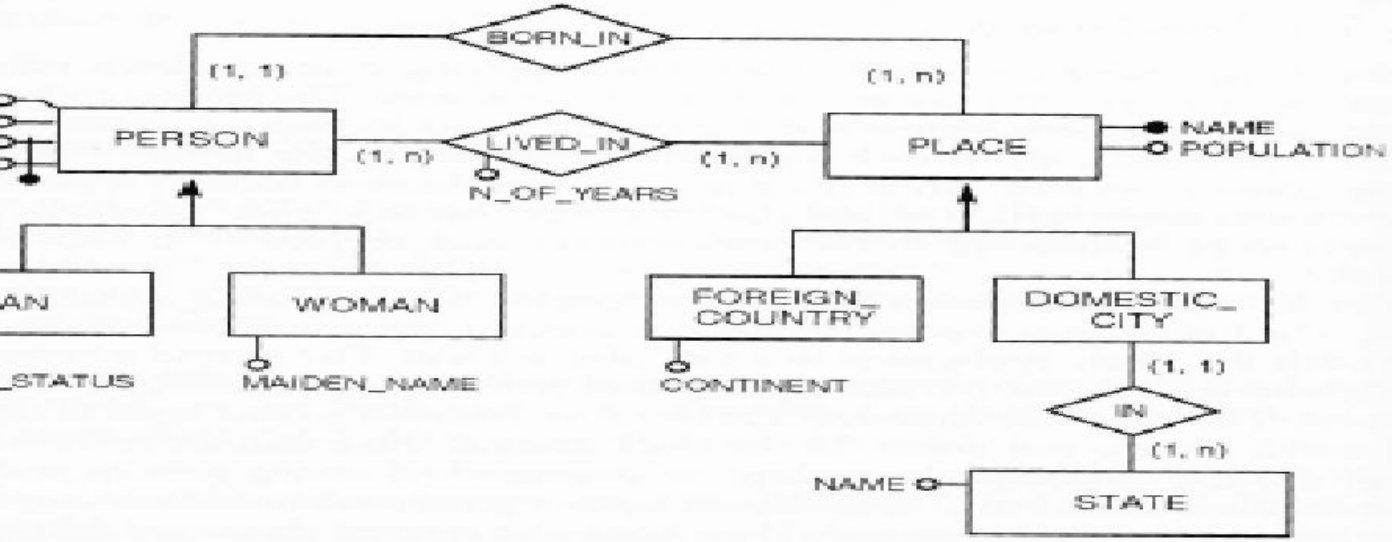
Figure 3.10 The top-down strategy

- In the top-down strategy schema is obtained applying pure top-down refinement primitives

ond refinement



refinement (primitives T_2 and T_4)



al schema (primitives T_1 and T_6)

Bottom-Up strategy

- In the bottom –up strategy we apply pure bottom –up primitives

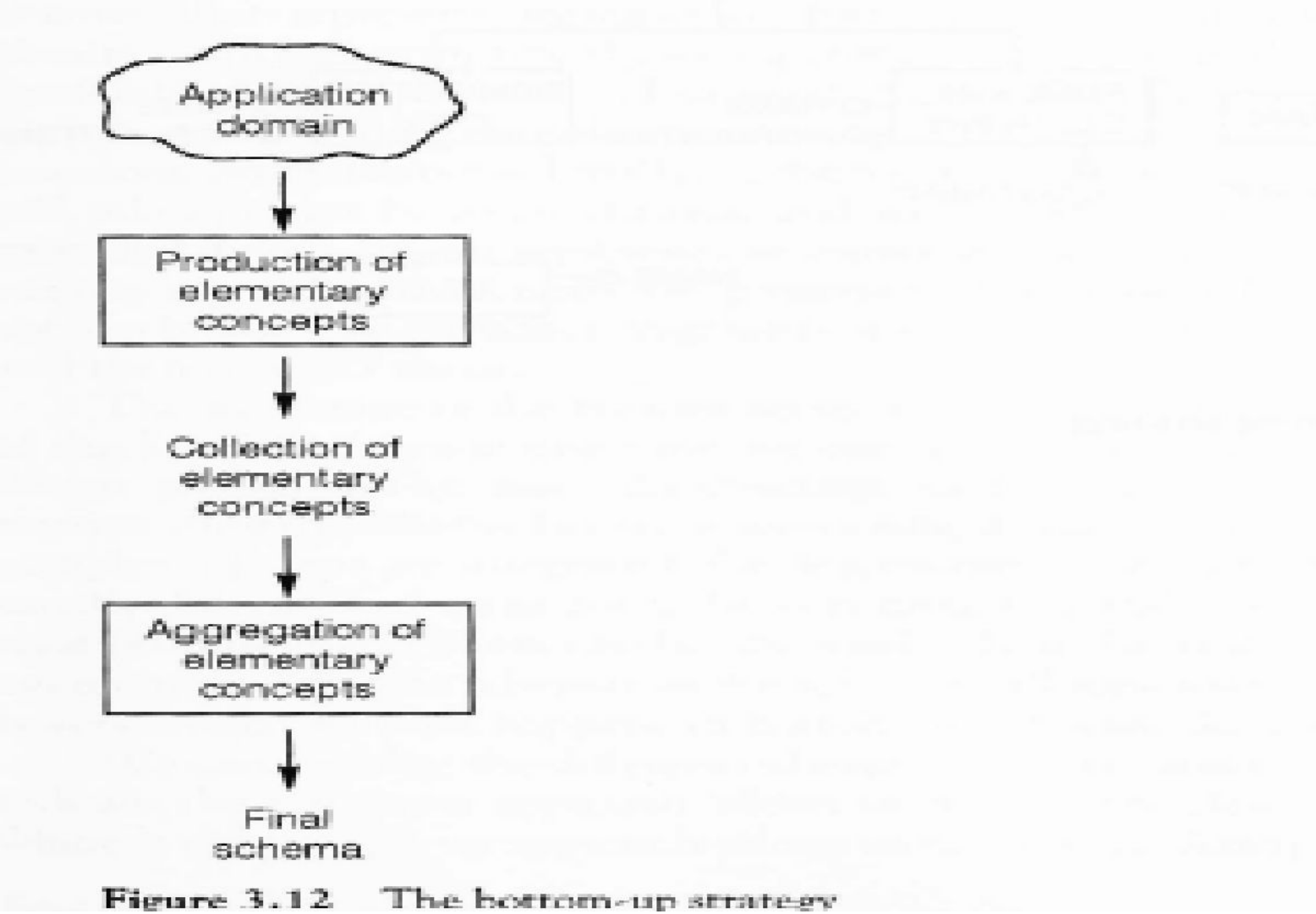
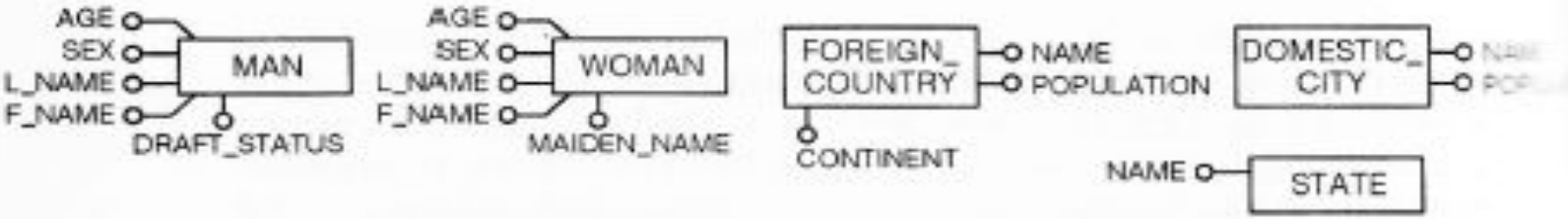
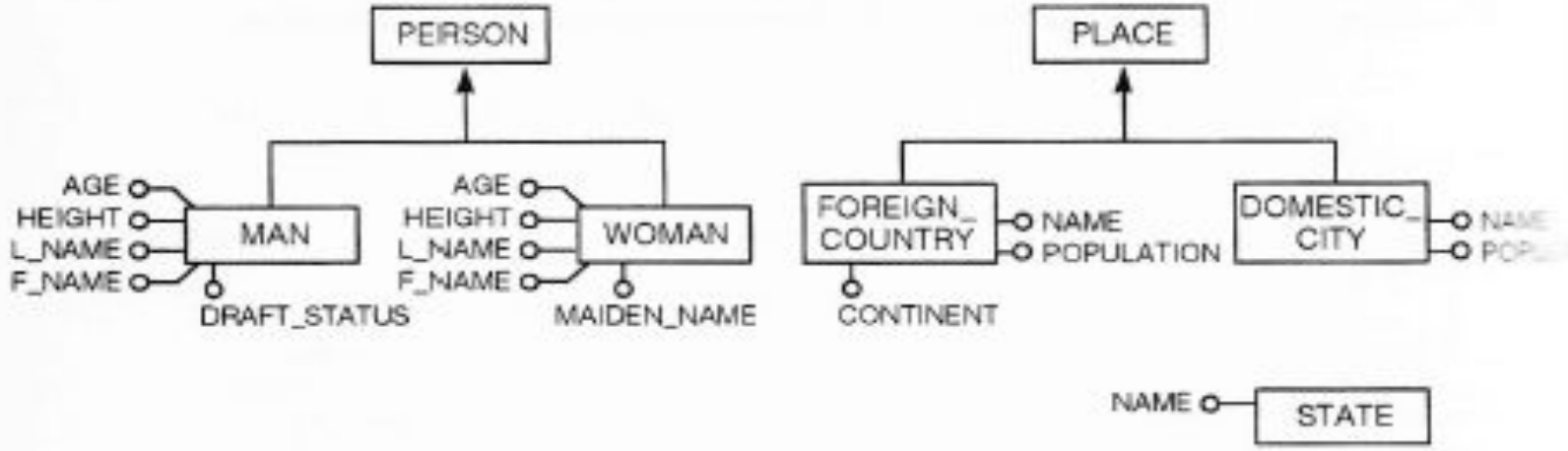


Figure 3.12 The bottom-up strategy

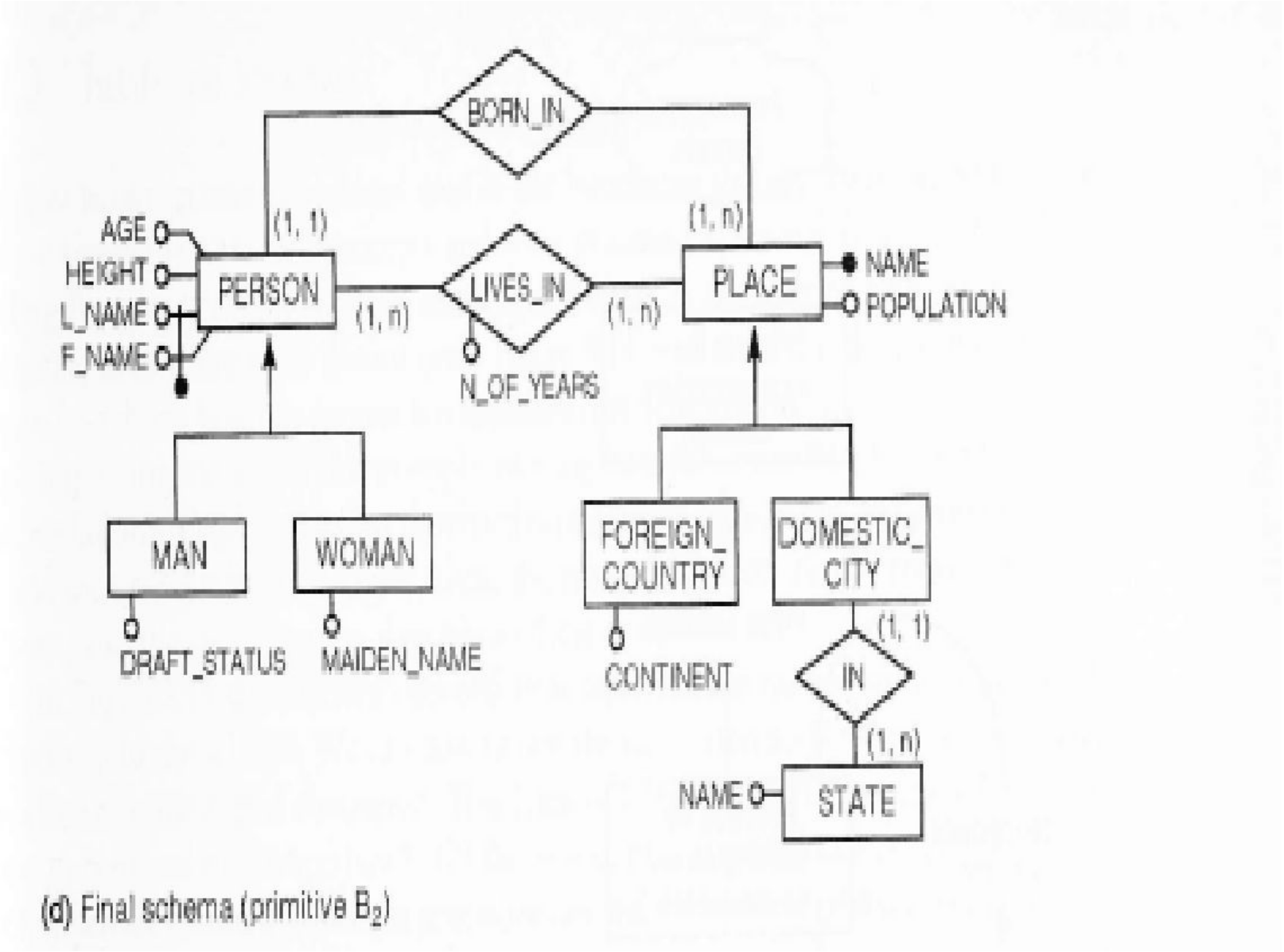
(a) First schema



(b) Second schema



(c) Third schema



(d) Final schema (primitive B_2)

Inside-out strategy

- This strategy is a special type of bottom –up strategy

Here we fix the most important or evident concepts and then proceed by moving as oil stain does finding first the concepts that are conceptually close to starting concepts and then navigate to more distant ones

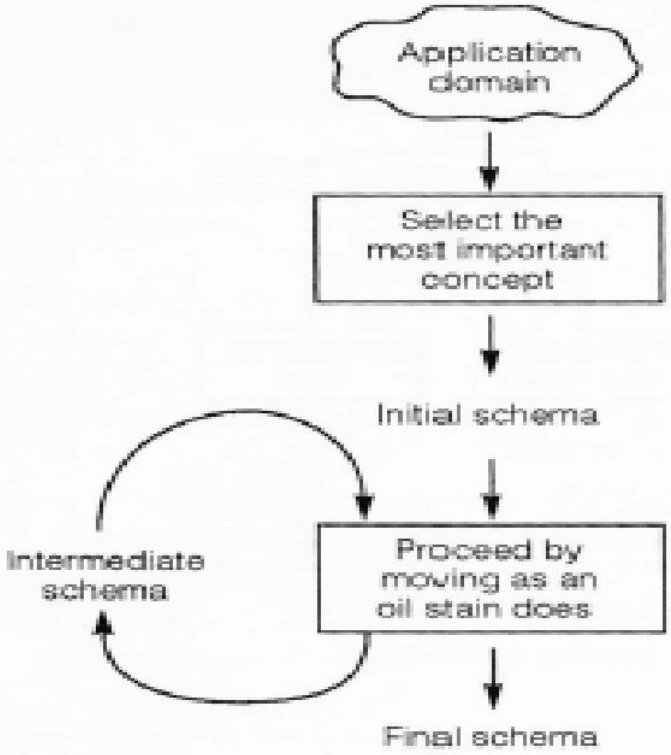


Figure 3.14 The inside–out strategy

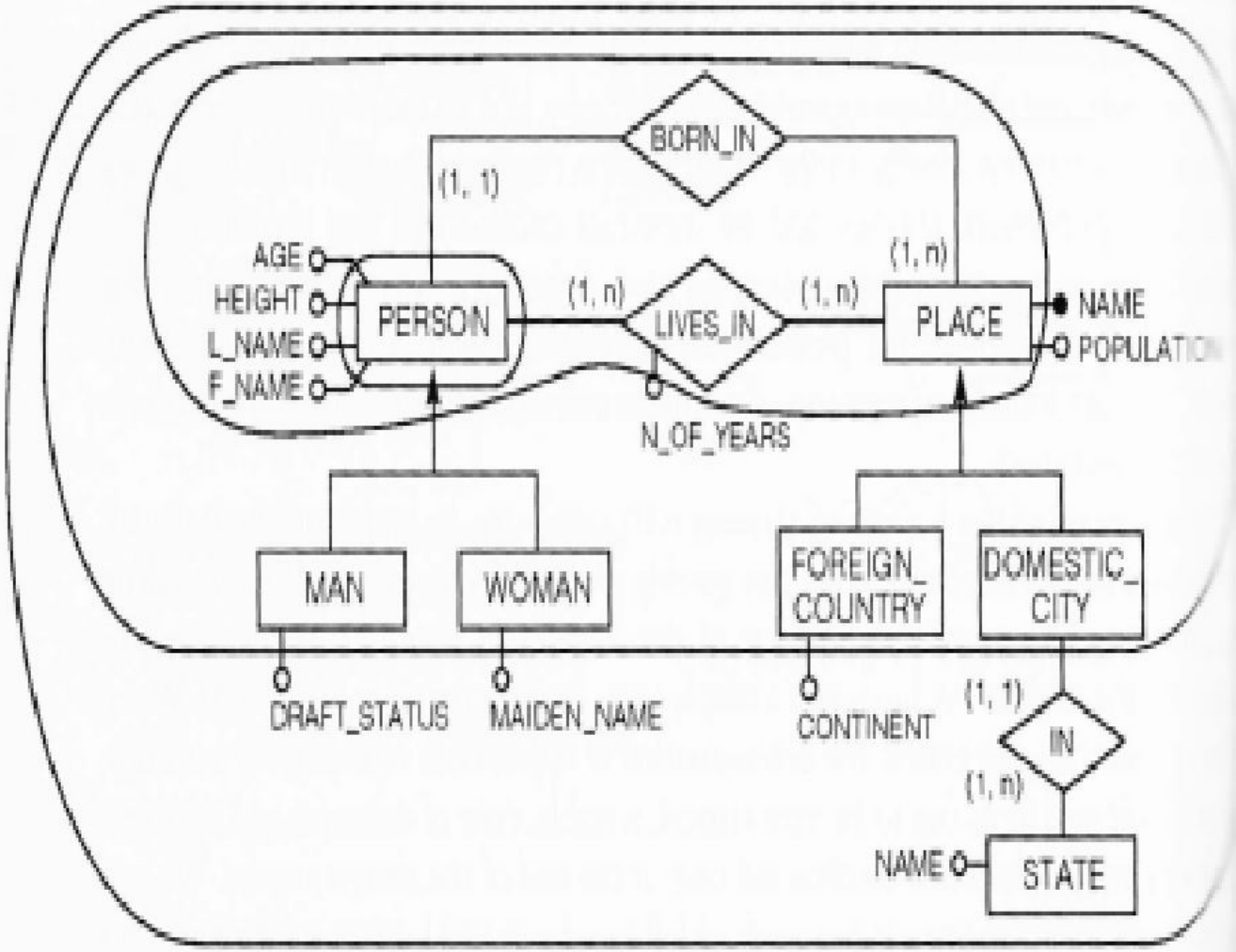


Figure 3.15 A design session using the inside-out strategy

Mixed strategy

The mixed strategy takes advantage of both top-down and bottom-up strategies, by allowing a *controlled* partitioning of requirements. The main idea of this approach is shown in Figure 3.16; when the application domain is very complex, the designer partitions the requirements into subsets, which are later separately considered. At the same time, the designer produces a *skeleton schema*, which acts as a frame for the most important concepts of the application domain and embeds the links between partitions. The overhead introduced by this step is rewarded, since the presence of the skeleton schema allows an easier bottom-up integration of the different schemas produced.

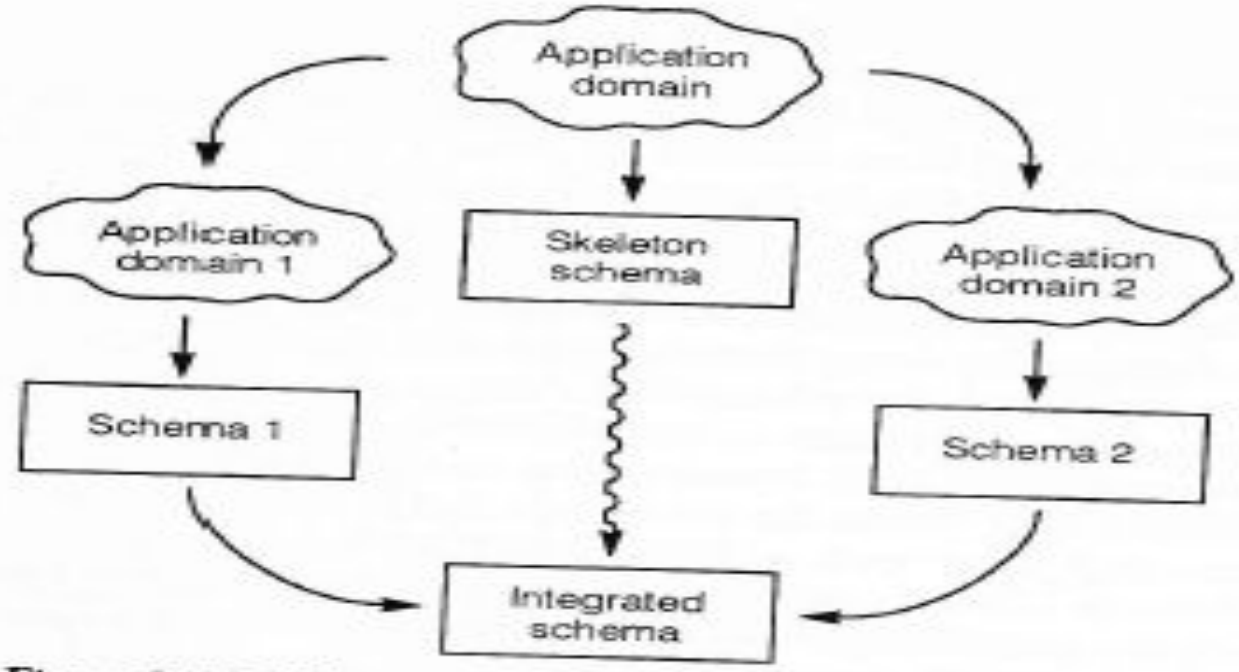
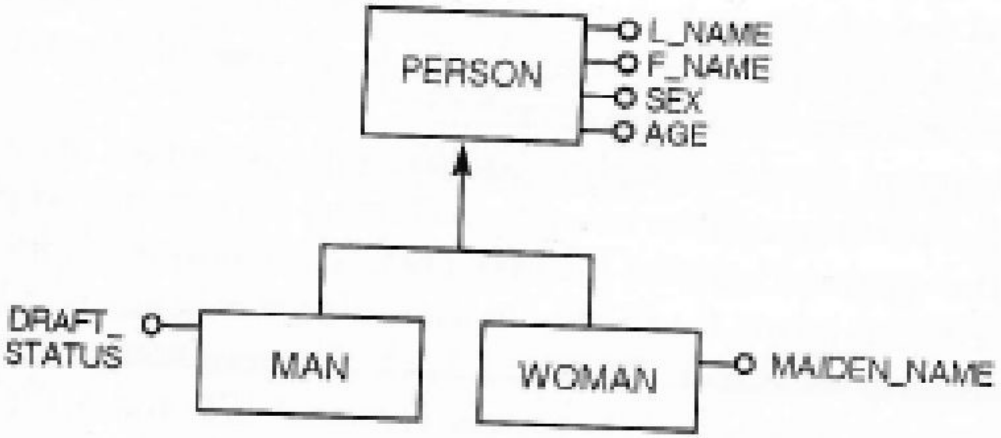


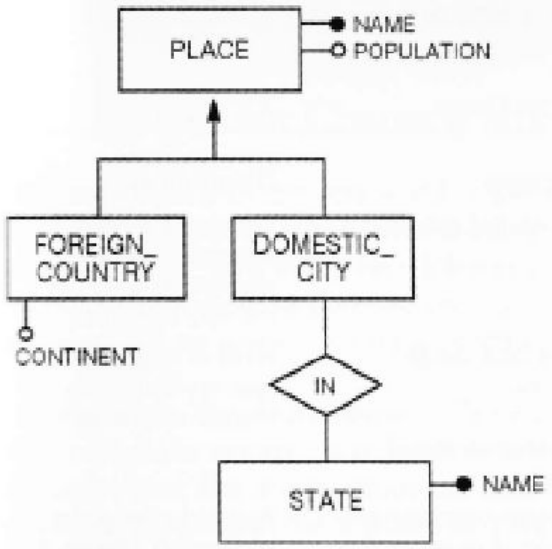
Figure 3.16 The mixed strategy

(a) Skeleton schema

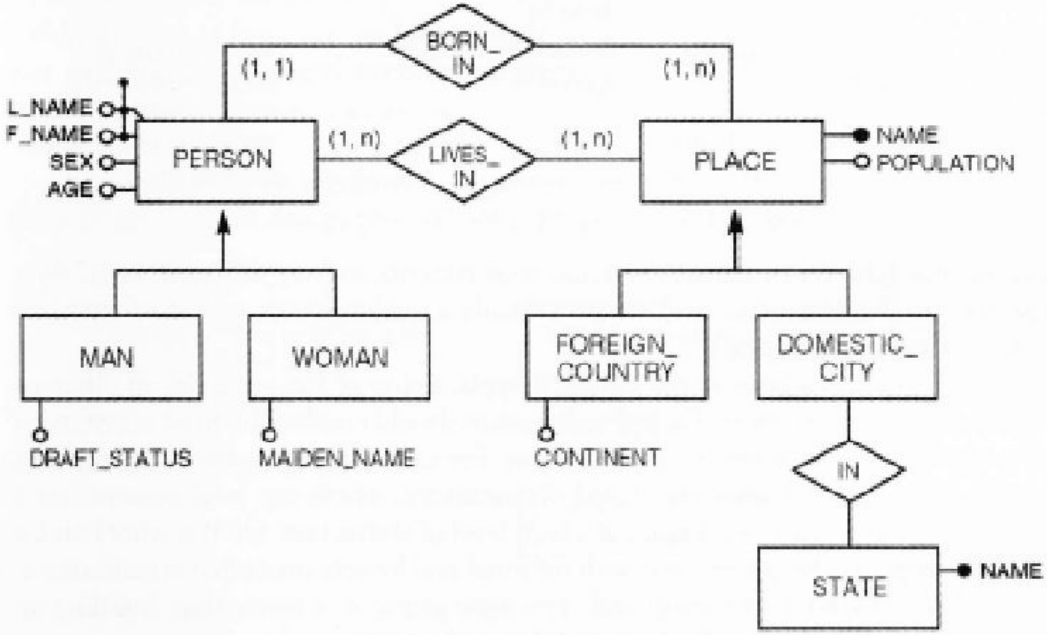


(b) PERSON schema

Figure 3.17 A design session using the mixed strategy



(c) PLACE schema



(d) Integrated schema

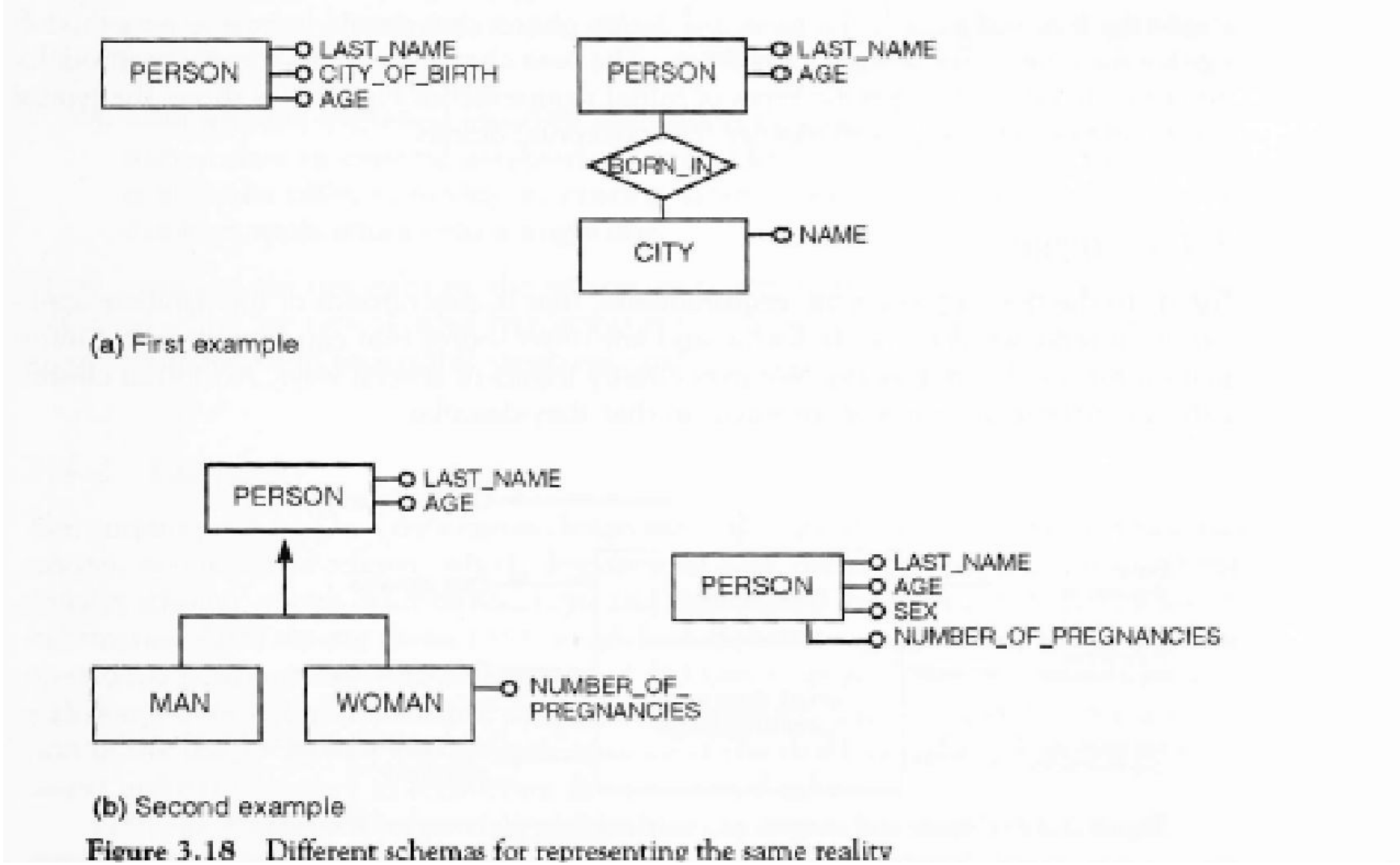
Figure 3.17 (cont'd) A design session using the mixed strategy

Table 3.1 Comparison of the Strategies for Schema Design

Strategy	Description	Advantages	Disadvantages
Top-down	Concepts are progressively refined	No undesired side effects	Requires a capable designer with high abstraction ability at the very beginning
Bottom-up	Concepts are built from elementary components	Ease of local design decisions No burden on initial designer	Need of restructuring after applying each bottom-up primitive
Inside-out	Concepts are built with an oil-stain approach	Ease of discovering new concepts close to previous ones No burden on the initial designer	A global view of the application domain built only at the end
Mixed	Top-down partitioning of requirements; bottom-up integration using a skeleton schema	Divide-and-conquer approach	Requires critical decisions about the skeleton schema at the beginning of the process

Criteria for Choosing among Concepts

-

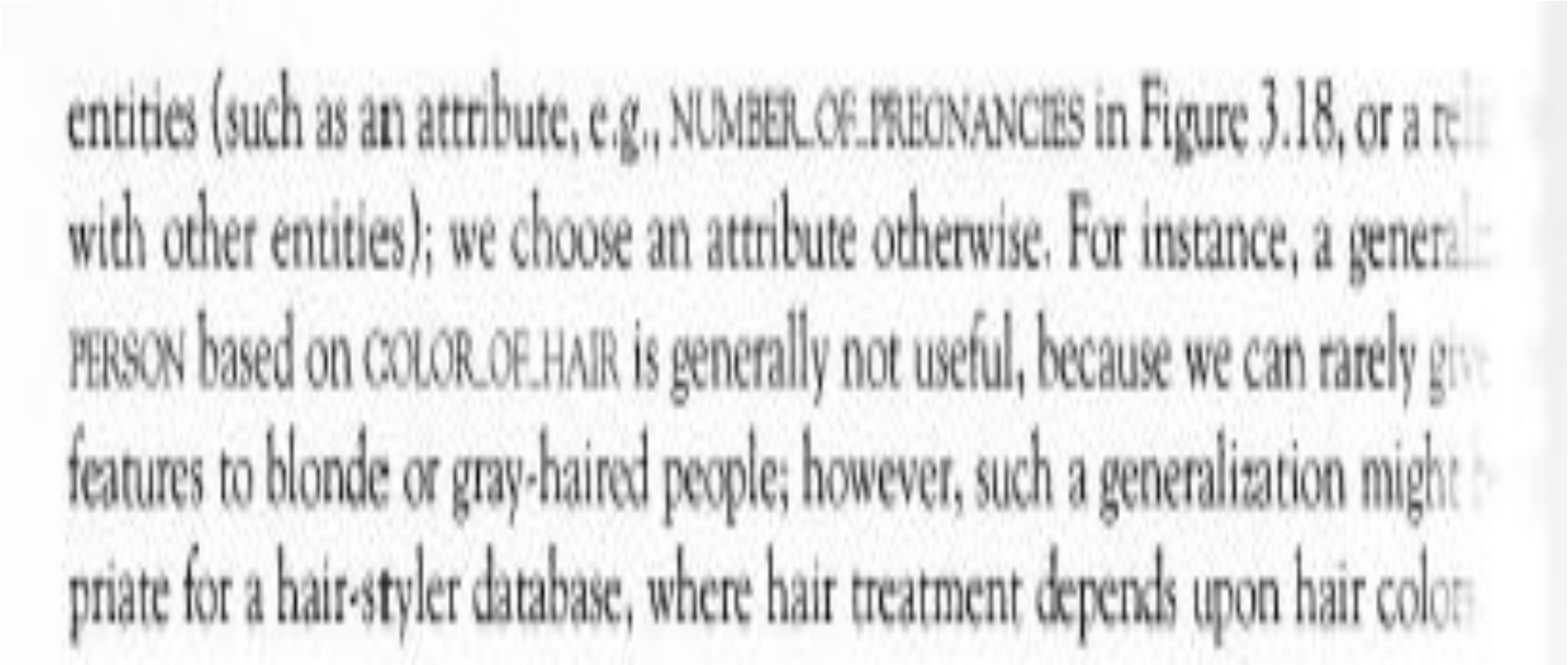


Entity vs. Simple attribute

Entity vs. Simple Attribute. This problem involves choosing whether a real-world object should be modeled as an entity or as an attribute. We should choose an entity when we understand that several properties (attributes, relationships, generalizations, subsets) can be associated with the object to be modeled, either now or later in the design process. We should choose an attribute when the object has a simple atomic structure and no property of interest seems applicable to it. For instance, the concept of color is typically an attribute (say, for the entity CAR); however, if our application is concerned with building pieces of furniture, and particularly with their coloring as a process, then COLOR may well become an entity, with the attributes NAME, COLOR_CODE, REQUIRED_NUMBER_OF_PAINTS, RUST_PROTECTION, and so on.

Generalization vs. attribute

- Generalization will be used when we expect that some property will be associated to the lower level

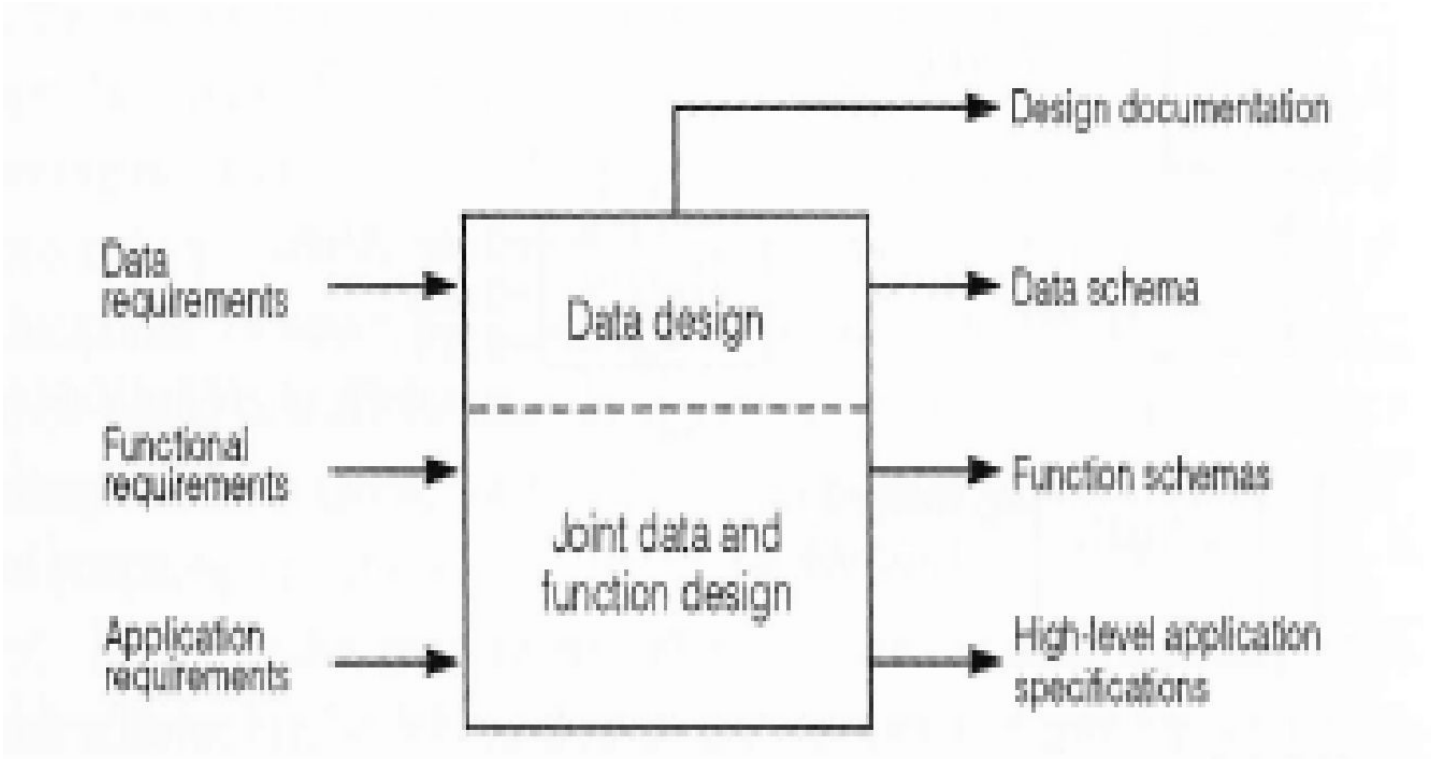


entities (such as an attribute, e.g., NUMBER_OF_PREGNANCIES in Figure 3.18, or a re
with other entities); we choose an attribute otherwise. For instance, a generali
PERSON based on COLOR_OF_HAIR is generally not useful, because we can rarely giv
features to blonde or gray-haired people; however, such a generalization might b
priate for a hair-styler database, where hair treatment depends upon hair color

Composite attribute vs set of simple attributes

Composite Attribute vs. a Set of Simple Attributes. We choose a com
tribute when it is natural to assign a name to it; we choose a set of simple attrib
they represent independent properties. For instance, ADDRESS is a good abstract
attributes STREET, CITY, STATE, ZIP_CODE.

Inputs, outputs and activities of conceptual design



1. **Data requirements** describe the structure of data that should be stored within the database (e.g., employees have a name, a salary, a Social Security number).

2. **Functional requirements** describe the dynamic structure of the information system, by identifying several functions of activities within the system (e.g., order processing, shipment management) and the flows of information between them (e.g., orders, shipping notes). These terms will be more extensively defined in

3. **Application requirements** describe operations on data (e.g., insertions, updates, queries: "insert a new employee", "change the salary of an employee", "retrieve the salaries of all employees").

A second classification of inputs is in terms of the linguistic representation used in their description. We apply this classification to data requirements, but a similar classification might be done for dataflow and application requirements as well. Requirements are expressed with a variety of "languages":

1. *Natural language*, used in interviews and several types of documents.
2. *Forms*, which are paper modules used to collect data and to exchange data among users.
3. *Record formats*, COBOL *data divisions*, and so on, which describe the structure of data in traditional file systems; they should be considered (together with *screen layouts*, e.g., screens presented to users) when we develop a database starting from a traditional file system.
4. *Data schemas*, expressed in a data description language, which describe the structure of data in existing databases; they should be considered when we want to change the DBMS, to modify an existing database application, or to merge several database applications into a single one.

Outputs

- The outputs produced by a conceptual design methodology include (1) the *conceptual data schema*, or *conceptual schema*, which describes all data present in requirements; (2) the *function schemas*, which describe functions and activities of the information system and of information flows among them; (3) the *high-level application specifications*, which describe operations performed on the database; and (4) other *design documents*, which provide additional information on each of the preceding outputs. They are useful as documentation of the design activity and for maintenance of the database, when changing requirements make it necessary to restructure the conceptual schema.

Activities

Requirements Analysis. During requirements analysis, the designer sho the requirements in detail and work slowly and carefully to start producing a c schema. The fundamental goal of this phase is to give the requirements a struc makes the subsequent modeling activity easier (we deal with this problem in Ch The designer should eliminate ambiguities in the requirements (imprecise or descriptions of reality), aiming to produce requirements descriptions that will b input for conceptual design.

Initial Conceptualization. The goal of this activity is to make a first concepts to be represented in the conceptual schema. This activity is typical in mixed, and to some extent in inside-out strategies, but it is omitted in a pure b strategy. At this stage, the designer creates a preliminary set of abstractions that candidates to be represented as entities, relationships, and generalizations. Th produced is largely incomplete, since it represents only some aspects of requir

Incremental Conceptualization. This is the central activity of conceptu using the general strategies described in Section 3.2, a draft schema is progressive into the final conceptual schema.

Integration. This activity is typical in bottom-up or mixed strategies; it merging several schemas and producing a new global representation of all of the integration, we determine the common elements of different schemas and disc *flicts*, (i.e., different representations of the same concepts) and *interschema* proper constraints among different schemas).

Restructuring. As in software design, it is sometimes worthwhile to sus design process during conceptual design and give some attention to measuring and i the quality of the product obtained. However, what is a good schema? Wha relevant qualities in conceptual design? We will answer this question in Chapte

The preceding activities are typical of any design process. The relevance o them depends greatly on the specific design situation. For instance, if require expressed in natural language with a lot of ambiguities and omissions, then it i nient to avoid a deep analysis of requirements and proceed to initial conceptuali requirements are expressed using forms, however, it is worthwhile to perform an a analysis of their structure. This enables a straightforward translation of requireme the conceptual schema.