Database Design & Modeling : Entity / Relationship

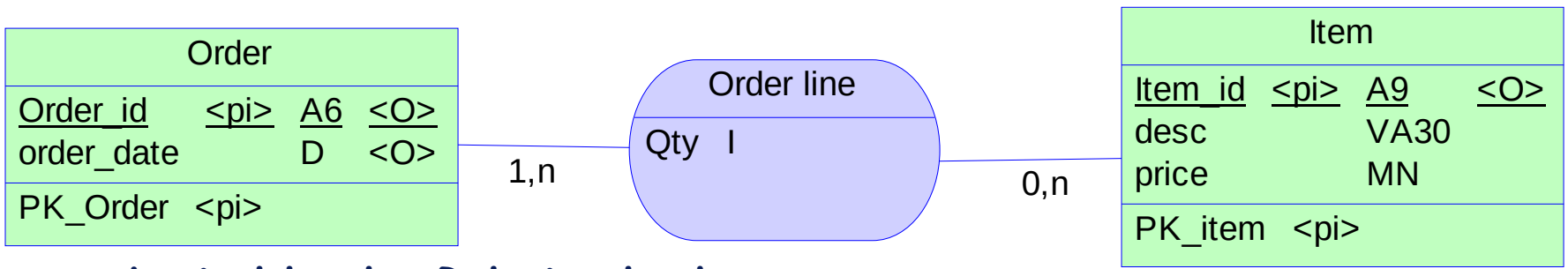
Yann Thierry-Mieg
ECE 2005-2006

Merise and Entity-Relationship

- E/R schema ISO standard
  - Based on Chen'76
  - Many notations exist : Merise, Axial, Sched, UML profile …
- Goal :
  - Graphical description of a database schema
  - Independent from actual database realization (network, RDBMS…)
  - Concise and readable description of a database
- Advantages :
  - Automatic translation to a physical relational data model
  - Good tool support (i.e. Sybase AMC suite)

Description levels

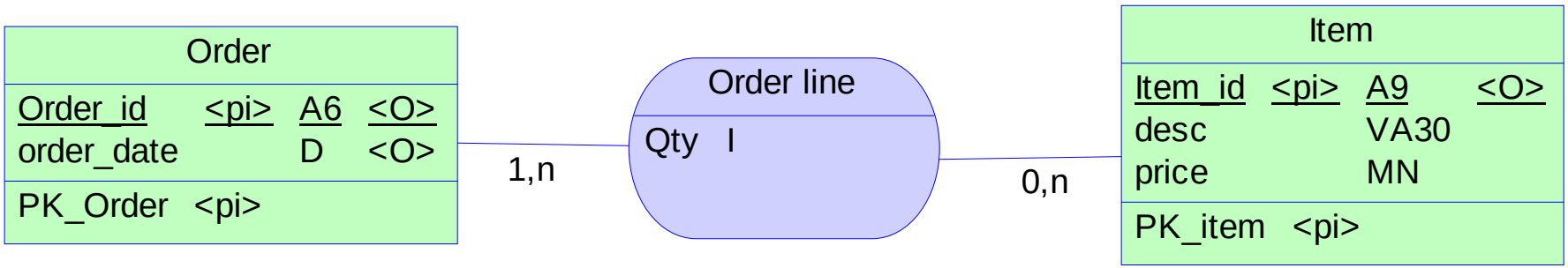
- **Three description levels are distinguished**
  - **Conceptual level = Entity/relationship schema :**



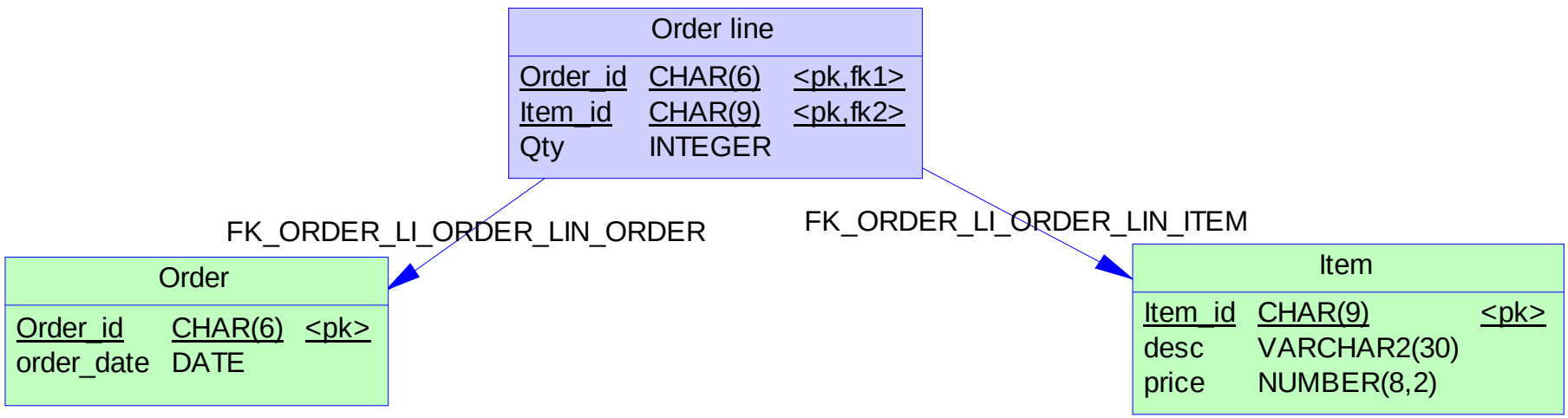
  - **Logical level = Relational schema :**
    - *Order(Order_id,Order_date)*
    - *Item(Item_id,Desc,price)*
    - *Order_line (**Order_id*,Item_id***,Qty)*
  - **Physical level**
    - *Primary and foreign keys*
    - *Indexes, tablespace …*
    - *SQL commands*

Description levels (2)

- **Conceptual level**



- **Physical Level**



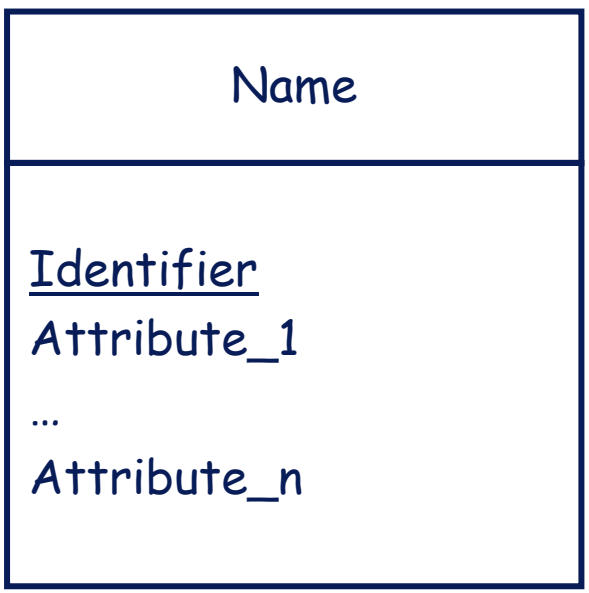
Entity : Definition

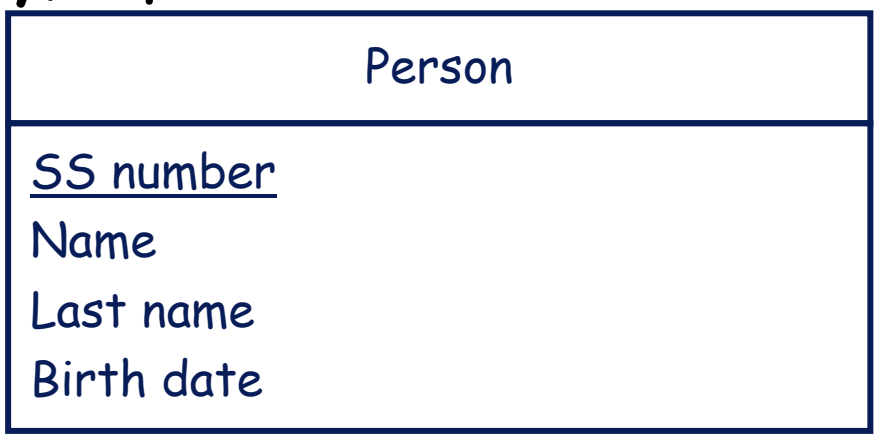
- An entity is a *standalone* object which is significant for the business model, and is described by a set of *properties (attributes)*

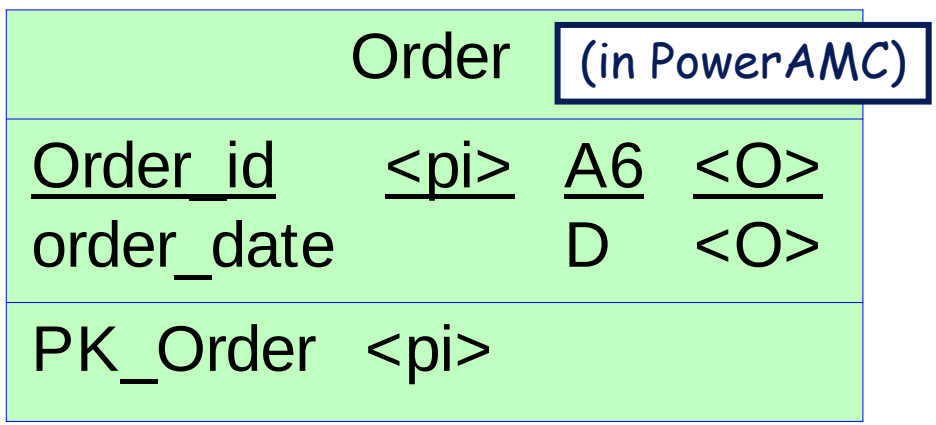
- Standalone :
  - Can be defined independently of the rest of the data
- In practice :
  - Equivalent to a row of a database
- Examples
  - Some persons, of properties ss_nb, name, last name, birth date :
    - *1580975002013 John Smith 17-SEP-56*
    - *2690345002017 Jane Jones 12-MAR-69*
  - An order, defined by it's number and date
    - *0012 14-MAR-98*

Entity type

- Concept similar to OO class (entity type) with respect to an instance (row).
- An entity type (or simply **entity**) represents a set of entities of same nature (same properties)
- Graphically :





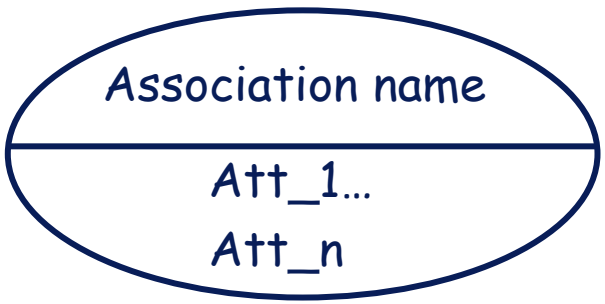


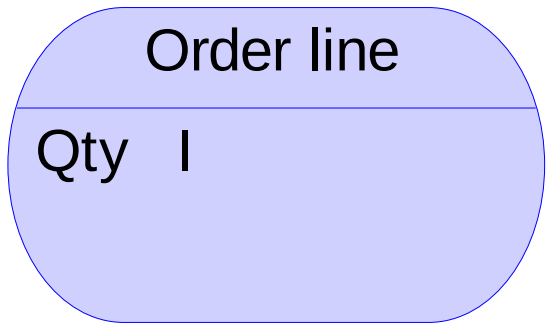
Entity Identifier

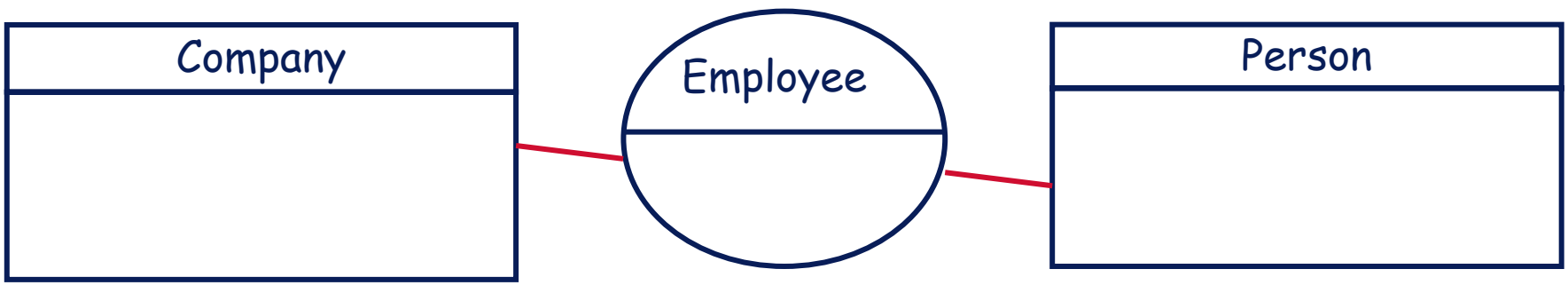
- An entity identifier is a set of it's attribute which allow to uniquely identify a member of it's population.
  - Will serve to create a primary key
- Graphically:
  - Underlined in the entity definition
- Examples
  - Soc. Sec. number of a person
  - Order id of an order …
- Can be composed of more than one attribute
  - Order id + date :if order id is reset every day.

Association or Relation

- An association or relation represents a relationship which is significant for the business model, and links together two or more entities. A relation may additionally bear a set of attributes.
  - Links entities in an E/R schema
  - Not standalone : defined with respect to existing entities
- Graphically :

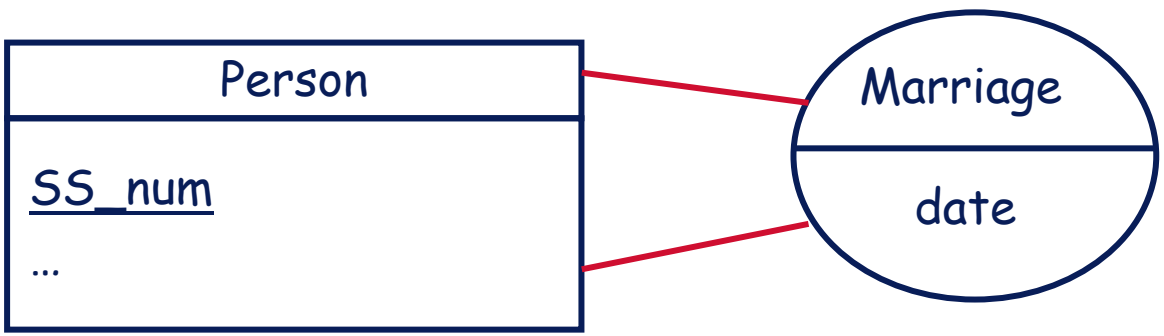






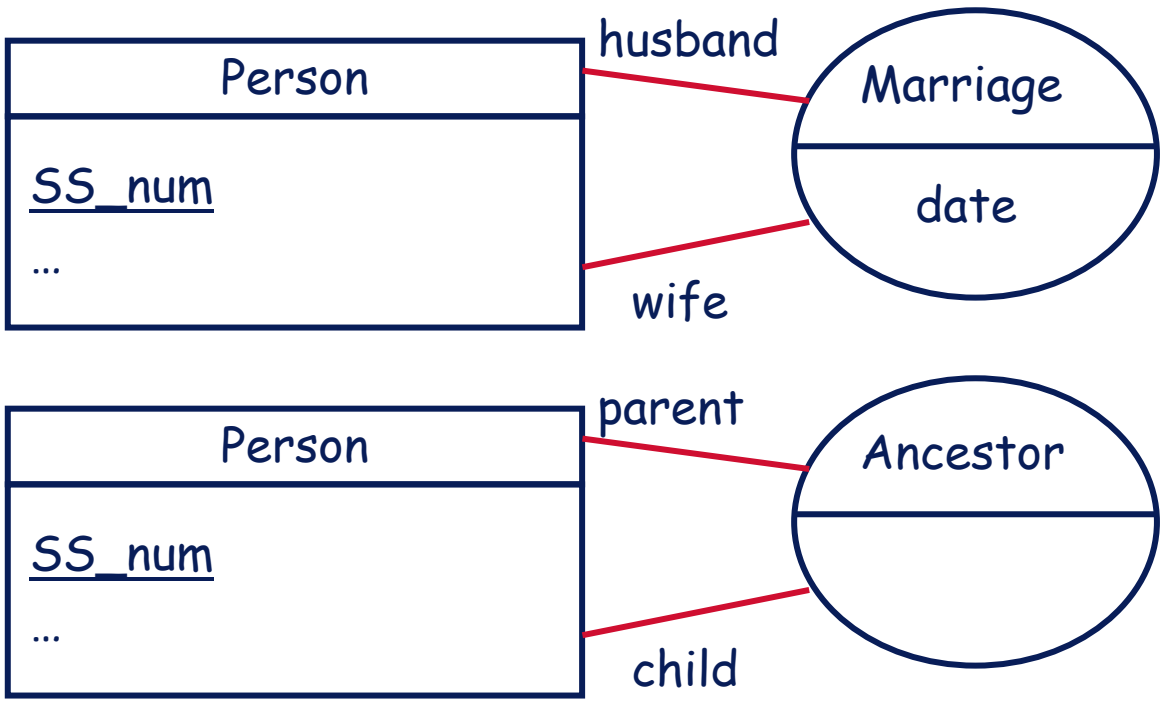
Dimension of a Relation

- The dimension of a relation is defined as the number of entities it connects together
  - Any relation is at least binary
  - Dimension = number of arcs from the relation to entities
  - We use the term binary (2), ternary (3) or n-ary relation
- An association may be reflexive
  - Here, dimension is still 2



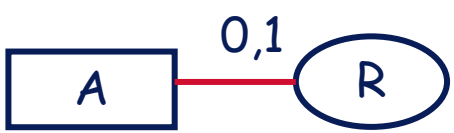
Role

- A role allows to distinguish the occurrences of an entity linked to a reflexive association
  - Graphically added to association edge
- Examples :

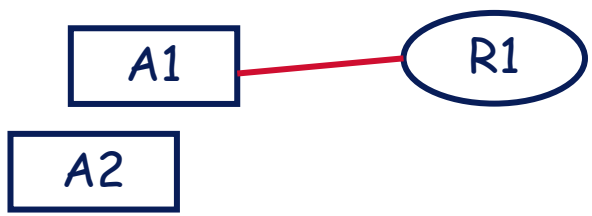


Cardinality

- Cardinalities express for a given entity occurrence, how many relationship occurrences can refer to it
  - Composed of two values min,max
  - ALWAYS READ from SQUARE (entity) to CIRCLE (RELATION) : Opposite of UML.
  - In fact no ambiguity, a relation occurrence always connects EXACTLY ONE OCCURRENCE of each entity it touches
- Although any cardinality specification is possible, usually use
  - 0 or 1 for minimum
  - 1 or N for maximum
  - i.e. : 0,1 1,1 0,N 1,N
- Examples :



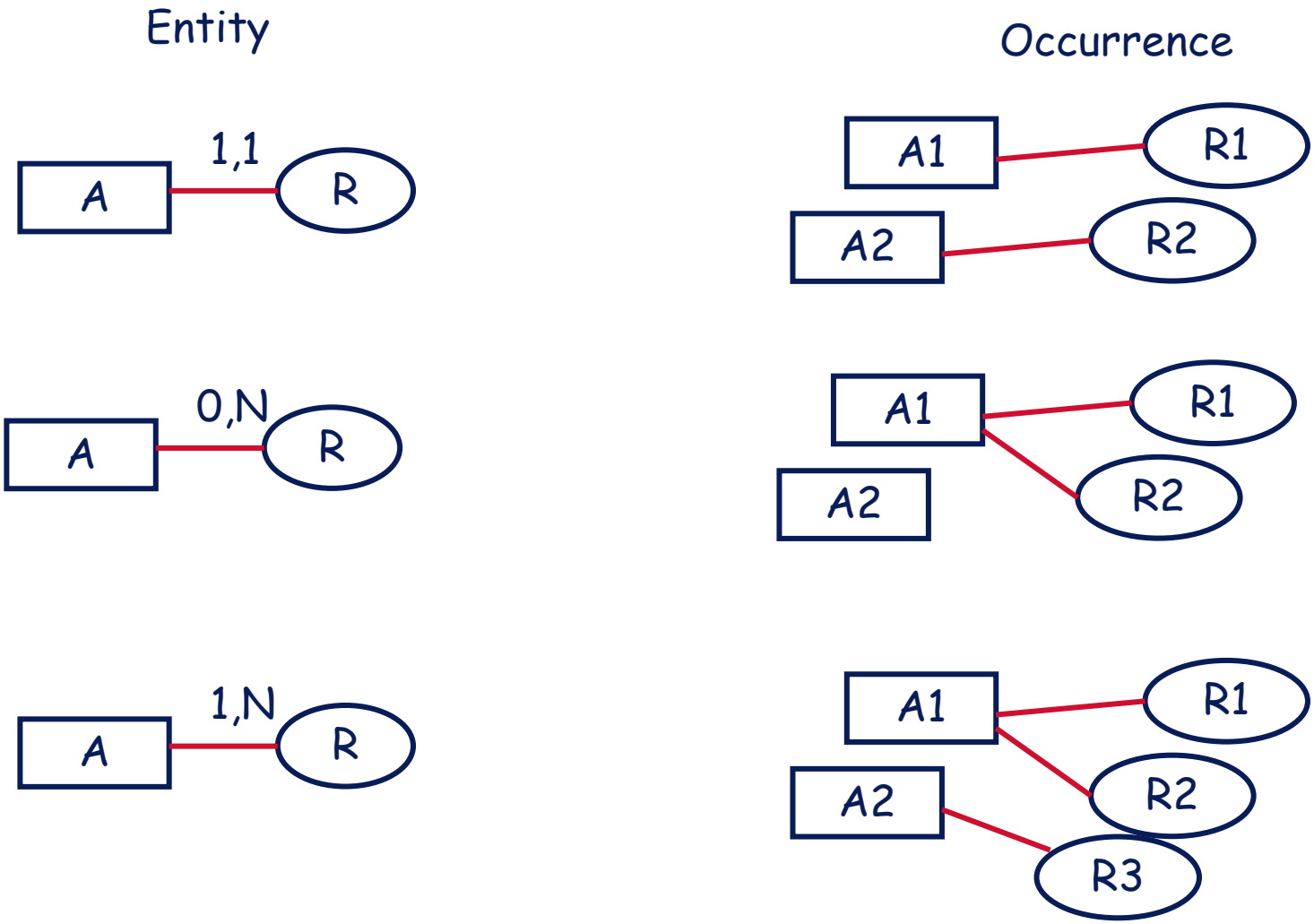
Entity



Occurrence

Cardinality (2)

- **Examples :**

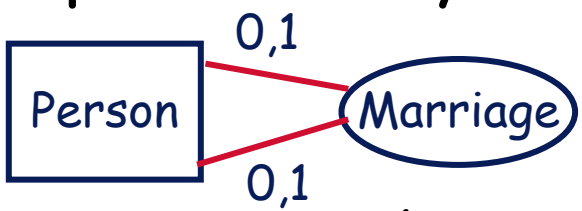


More cardinality examples

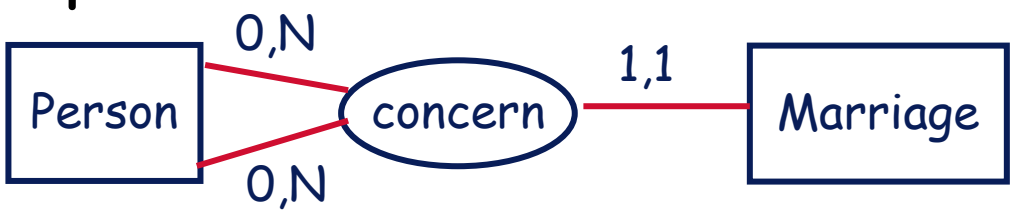
- An order is passed by a single customer

- An order may concern at least one and possibly several items

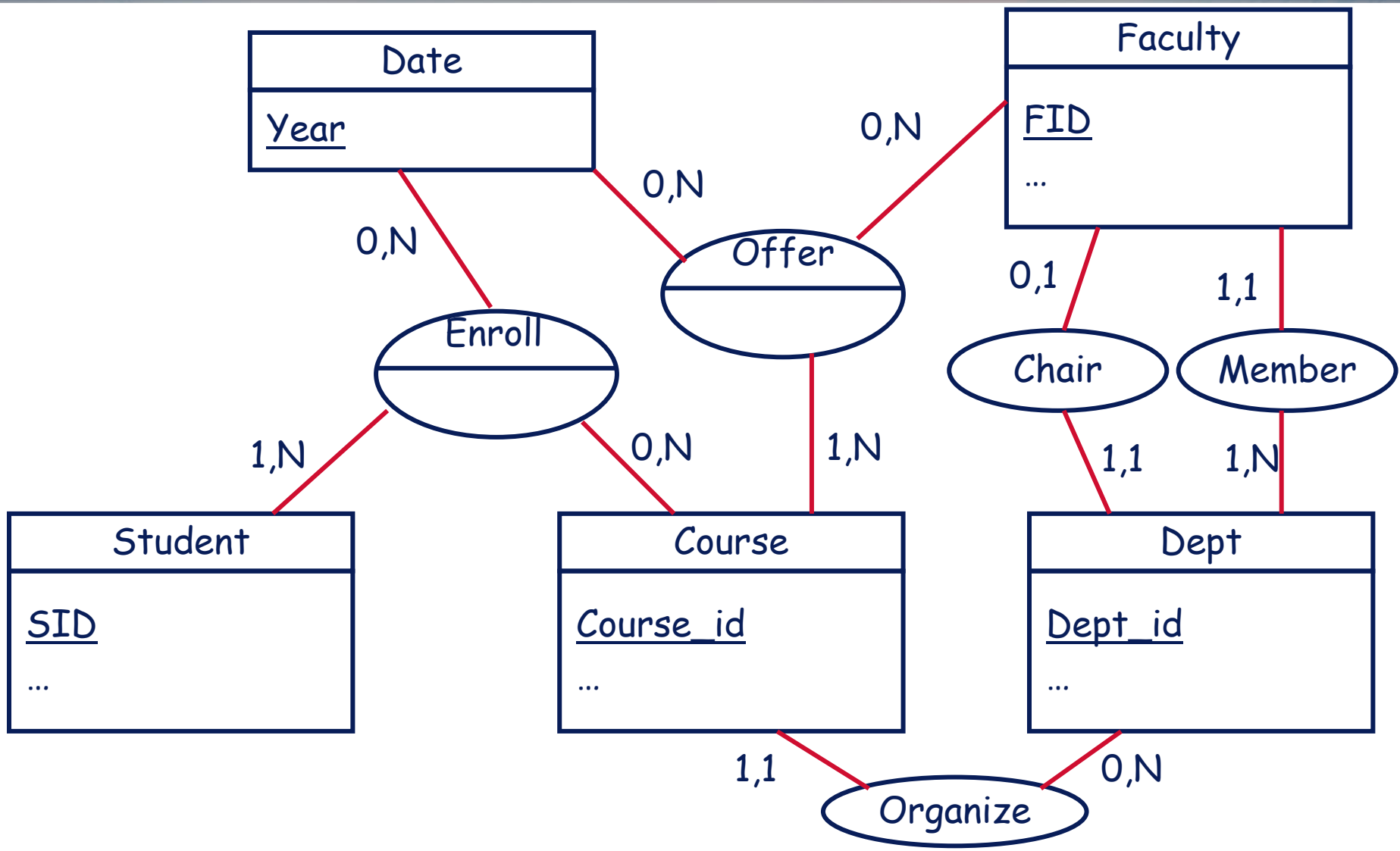
- A person can only be married once



- A person can be married several times



University conceptual schema



From CDM to LDM
Conceptual -> Logical Data Model

- **Automatic rules are given to translate a CDM (Entity/Association) to an LDM (Relational) :**
  1. **Any entity A yields a relation RA, of primary key composed of the identifier properties of the entity**
  2. **For associations, two cases are possible :**
     1. *The association is BINARY and is linked by at least one arc of cardinality 0,1 or 1,1 to an entity A.*

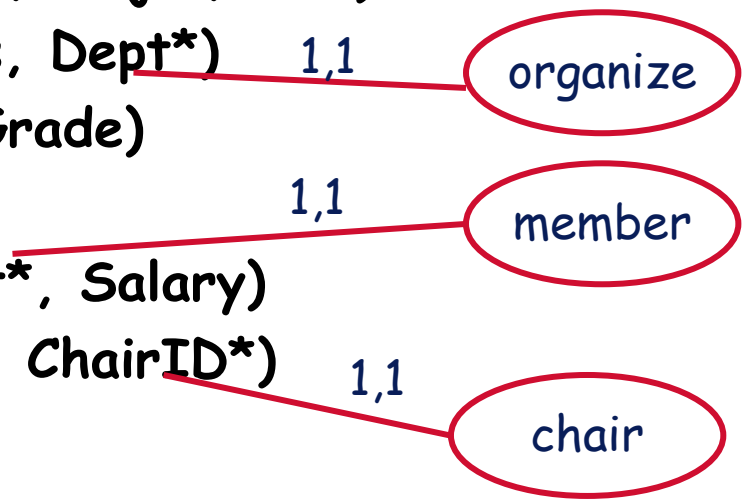
        *The association will be **absorbed** by the entity A, and will appear as a foreign key in the relation RA.*
     2. *The association is n-ary (n>2) OR only has 0,N or 1,N arcs.*

        *The association is translated as a new relation, of primary key composed of **the union of the keys of the entities it connects.***

All properties of entities or associations are translated as attributes of the relation they correspond to.

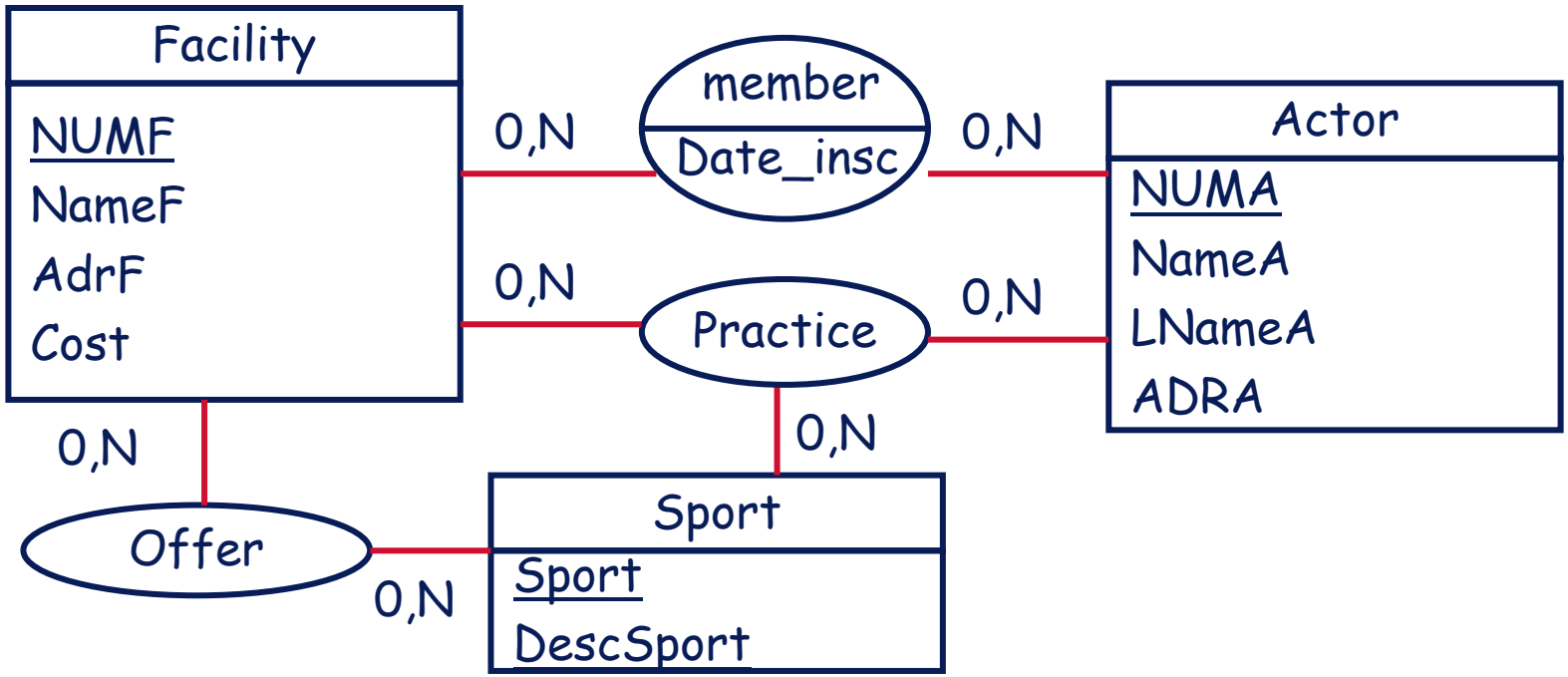
University : Resulting Logical schema

- Students(SID, Name, Age, Sex, Major, GPA)
- Courses(Course_id, Title, Hours, Dept*)
- Enrollment(SID*, Cno*, Year, Grade)
- Offers(Cno, Year, FID)
- Faculty(FID, Name, Rank, Dept*, Salary)
- Departments(Dept_id, Location, ChairID*)



- Date(year)
  - Should be created by basic application of translation rules. In practice, optimized away.
- Optimization rule :
  - Any entity that has a single attribute may be removed from the relational schema (i.e. no table)
  - Frequent example : Date
  - However, such a table is sometimes kept, so the data is not lost if it not used anymore. (i.e. a sport not practiced or offered)

SPORACT database



FACILITY(NUMF,NAMEF,ADRF,COST)
ACTOR(NUMA,NAMEA,LNAMEA,ADRA)
MEMBER(NUMA,NUMF,DATEINSC)
OFFERS(NUMF,SPORT)
PRACTICE(NUMA,NUMF,SPORT)
SPORT(SPORT, DESCSPORT)

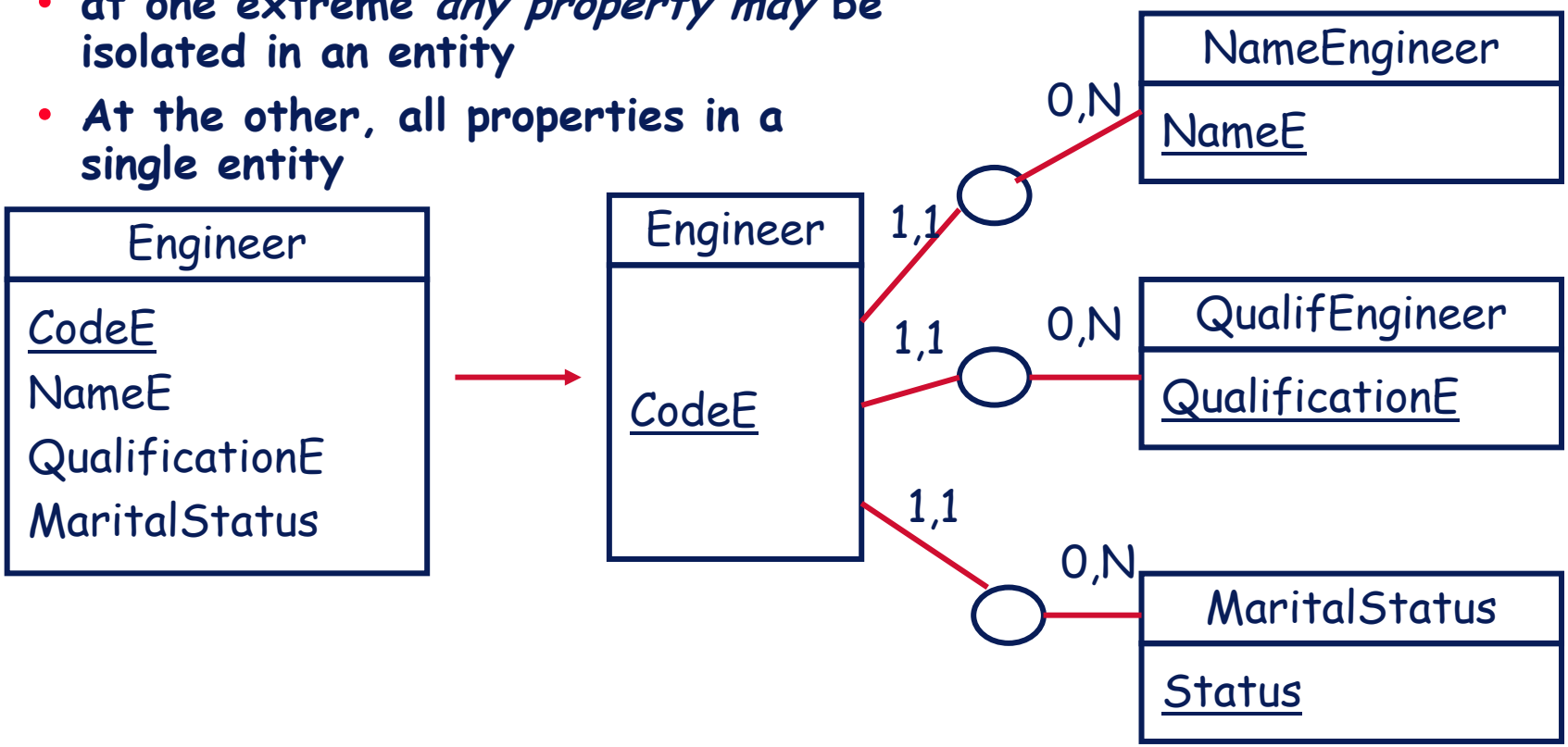
Designing a database schema

Design of a database schema

- Two main approaches to construct a first entity/association schema :
  - Bottom-up : make a list of all basic data that should be stored in the information system. Group them to form entities and/or associations.
  - Top-down : make a list of likely candidate entities and associations. Then add properties to these candidates.
- Bottom up is particularly appropriate when creating a database for an existing paper based system.
- Both approaches require definition of a *data dictionary,* that gives the *business definition* of each property of each entity, i.e.
  - NameA : Name of the actor
  - Cost : cost of a member card in the facility, per annum.
- Property names should be unique over the model :
  - e.g. NameActor not just Name

Decomposition of entities

- **Basic problem :**
  - **at one extreme *any property may* be isolated in an entity**
  - **At the other, all properties in a single entity**



When should we decompose ?

Decomposition : Motivation

- Consider the following schema :

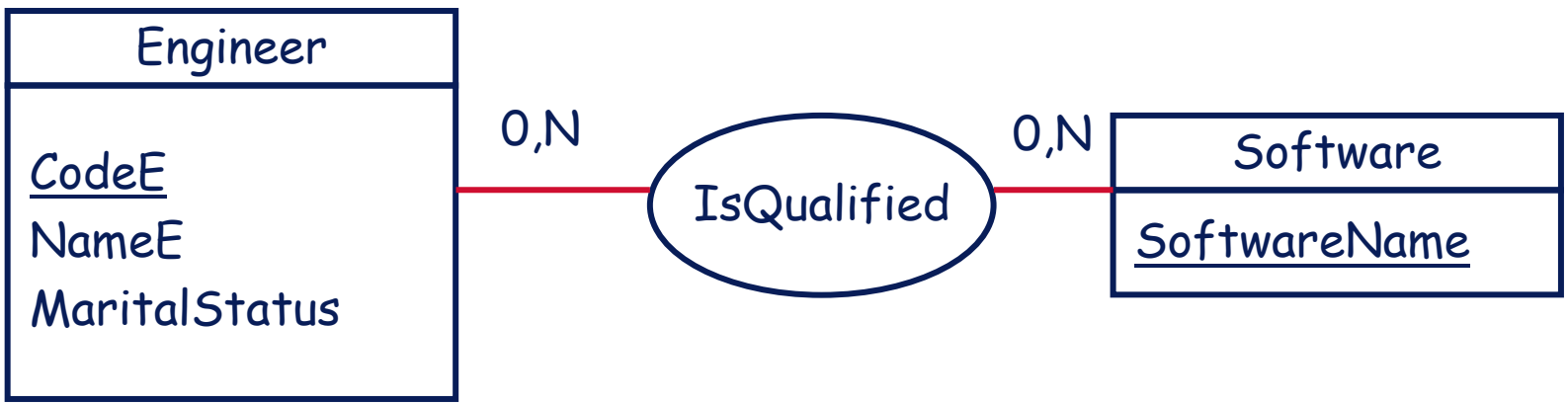
ID_I	ID_D	desc	qty	adr	price	cap
I1	D1	"a"	3	A1	21	200
I2	D1	"b"	5	A1	6,50	200
I1	D2	"a"	12	A2	21	150

Stock
CodeItem
CodeDepot
DescItem
QtyInStock
AdressDepot
ItemPrice
DepotCapacity

- Importance of decomposition to avoid :
  - Data redundancy : the same data is stored twice, leading to update problems (an adress change must be done for each item in stock)
  - Data loss = update anomaly : when a depot is empty we lose it's adress !! when an item is out of stock we lose it's description !!

Decomposition criterion (1)

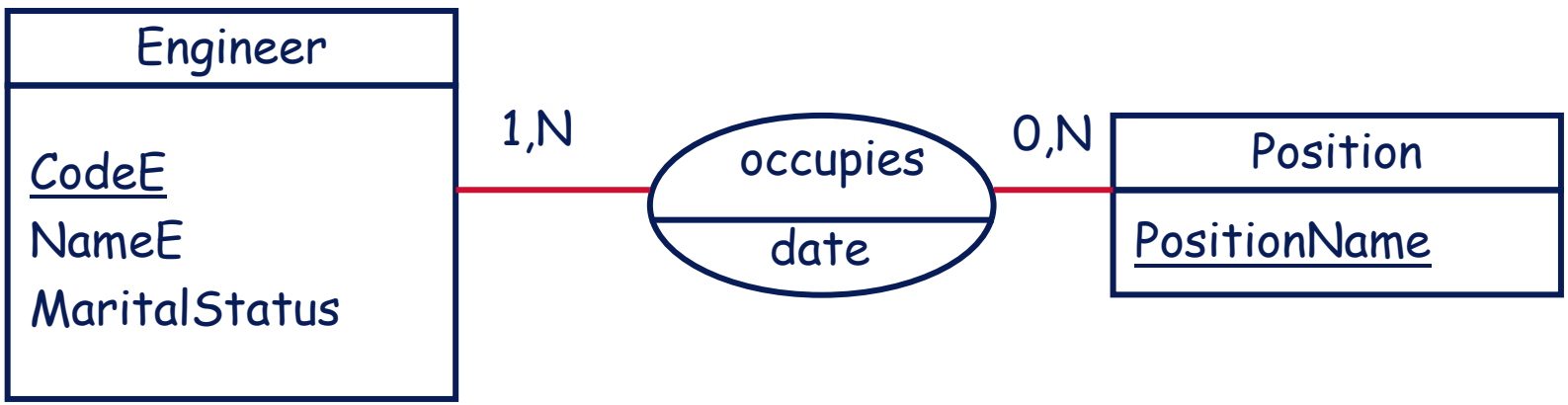
- **Case 1 : Mandatory**
  - **A property that may take several values for an instance of the entity**
  - **E.g. children of a person, qualifications of an engineer**



- **Corresponds to first normal form (1NF) :**
  - any entity must have an identifier, and
  - for a value of this identifier, all properties should take a **single** value

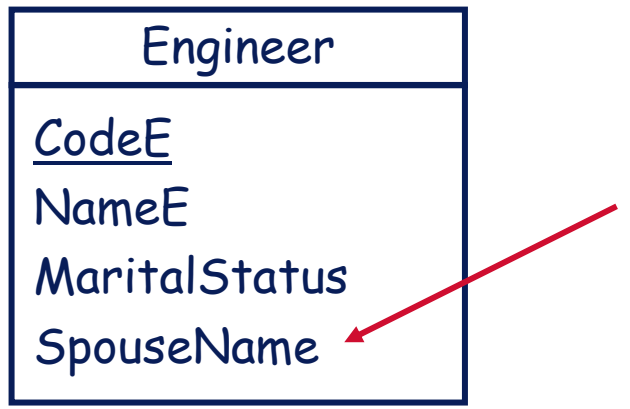
Decomposition criterion (2)

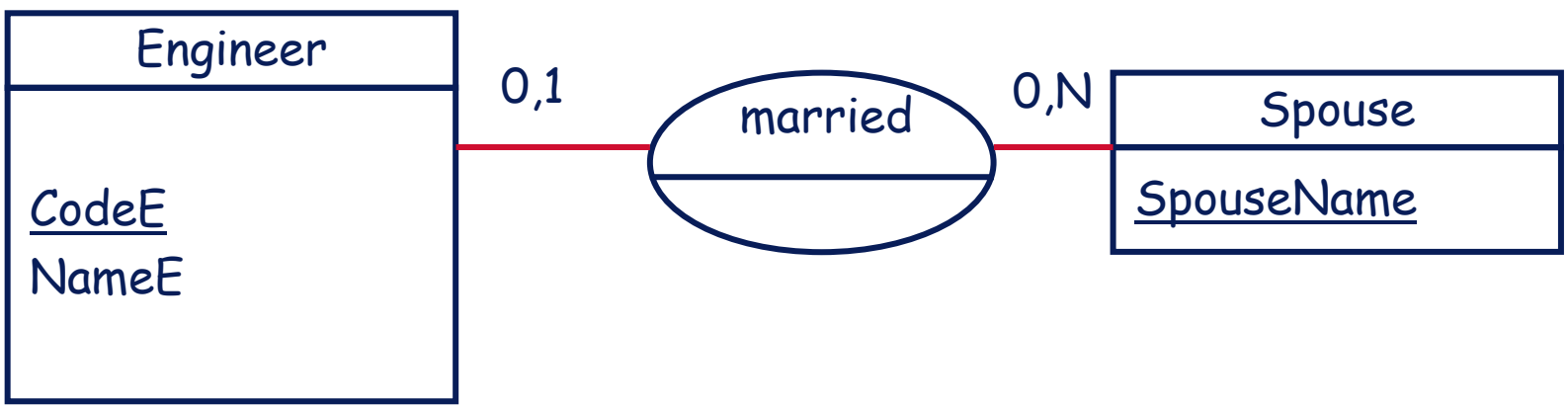
- **Case 2 : Mandatory**
  - A property that has a single value at time t but which may evolve and for which we wish to maintain a history
  - E.g. marital name of a person, position of an employee



Decomposition criterion (3)

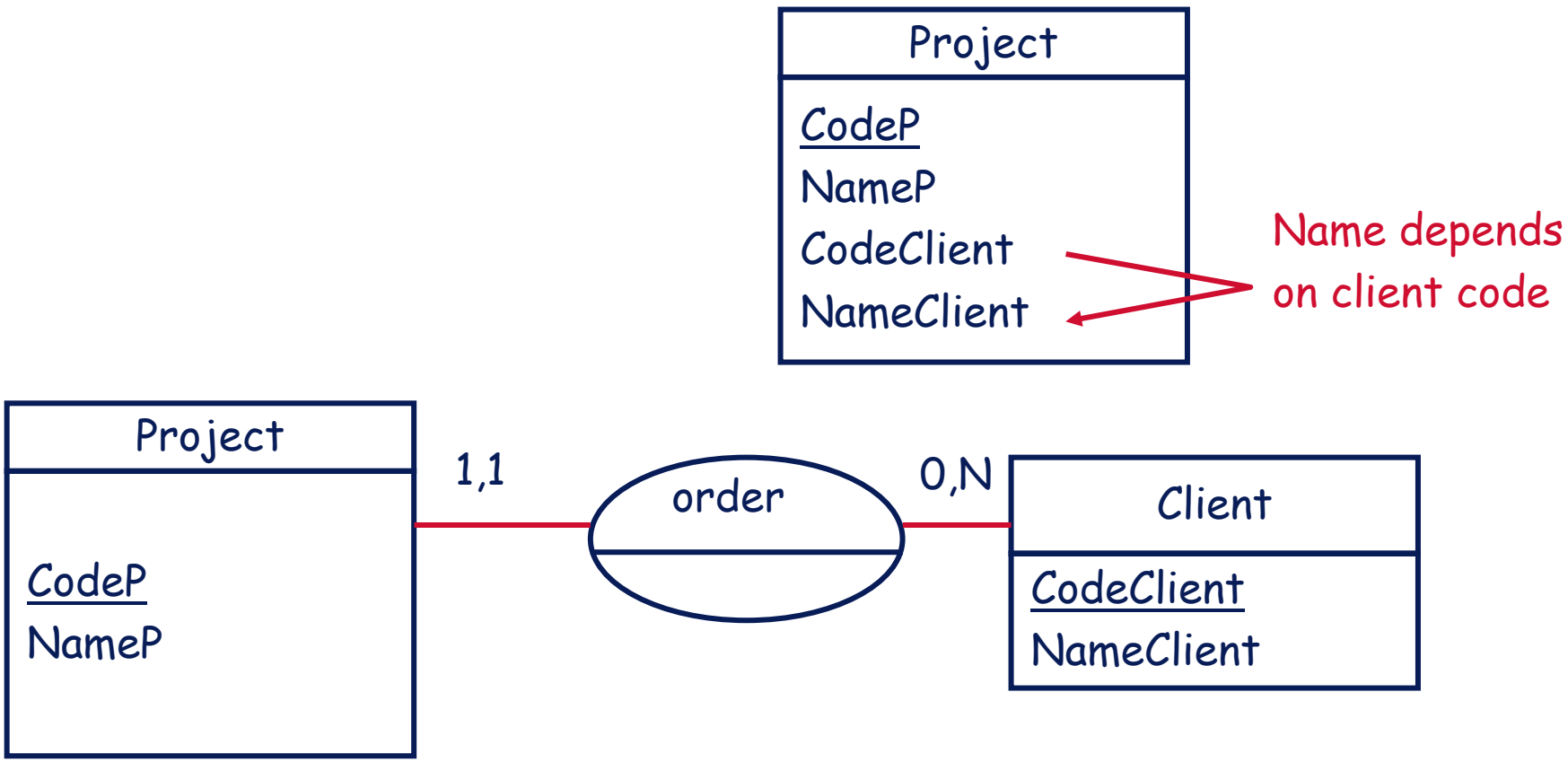
- Case 3 : Strongly recommended, at least in conceptual elaboration
  - A property that may not have a value for certain occurrences of the entity
  - E.g. marital name only valid for married women, date of discharge only valid for fired employees





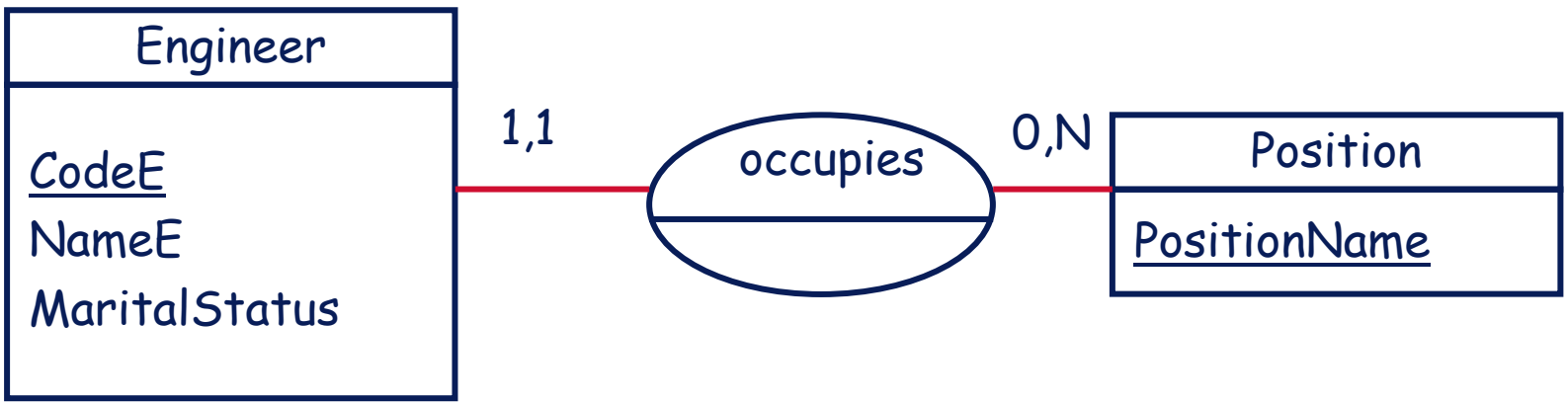
Decomposition criterion (4)

- Case 4 :Mandatory, at least in conceptual elaboration
  - A property that depends on the value of another property
  - E.g. transitive dependency



Decomposition criterion (5)

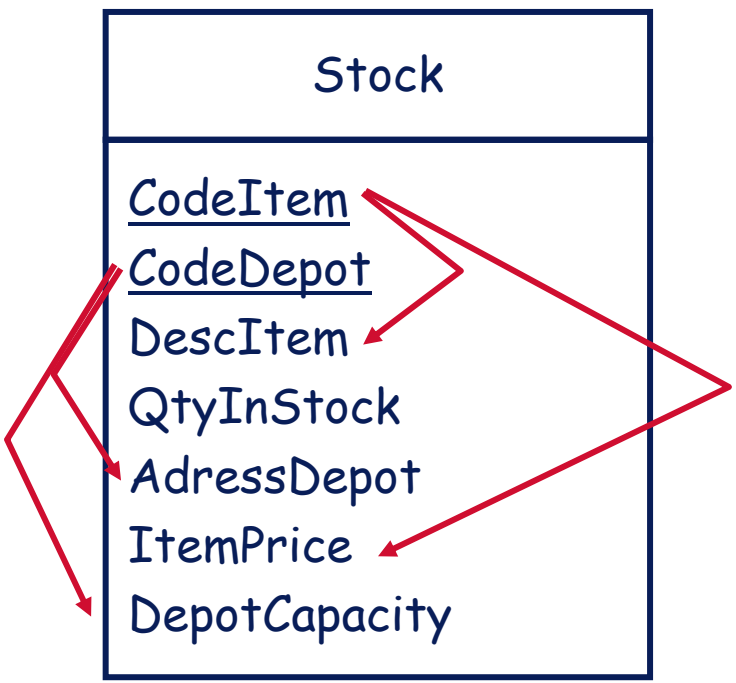
- Case 5 : may be indicated
  - A business operation rule concerns the property
  - E.g. The salary bonus of engineers depends on their position



Decomposition : Conclusion

- Normal forms and the study of *functional dependencies* → formalizes these notions
- Automatic analysis of functional dependencies is possible

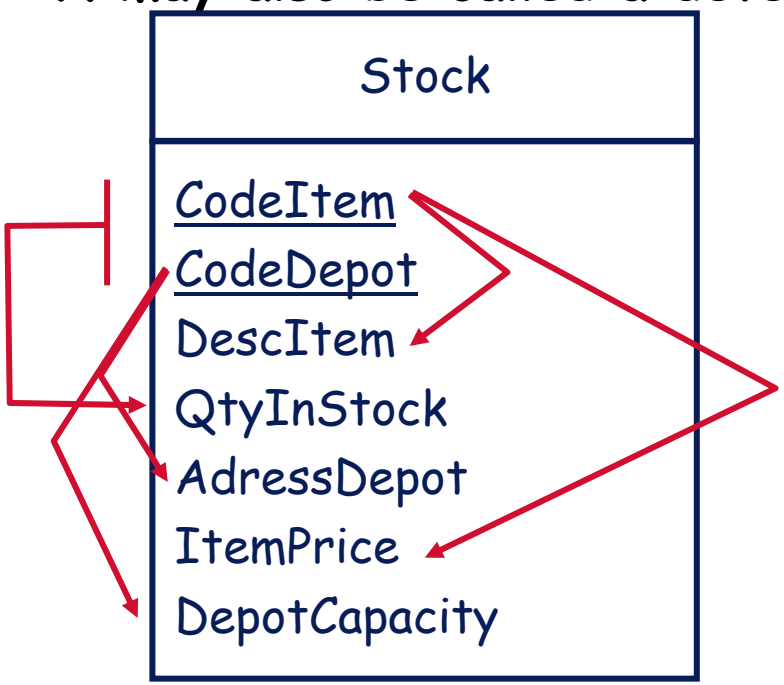
Not in normal form



In 3rd normal form !!

Functional dependency : Definition

- Let R(A1, ..., An) be a relation schema. Let X and Y be two subsets of {A1, ..., An}, i.e., X, Y $\subseteq$ R. R satisfies a *functional dependency*, denoted by X $\rightarrow$ Y, if in every legal instance r(R), for any pair of tuples t1 and t2 in r(R), if t1[X] = t2[X], then t1[Y] = t2[Y].
- If X $\rightarrow$ Y, we say that "X (functionally) determines Y".
- X may also be called a determinant



Example:

CodeItem -> DescItem
CodeItem -> ItemPrice
CodeDepot -> AddressDepot
CodeDepot -> DepotCapacity
CodeItem,CodeDepot -> QtyInStock

Which FD is satisfied?

- Which FD does R(A, B, C, D) satisfy, if the following instance is the only instance of R?

R

A	B	C	D
A1	B1	C1	D1
A1	B2	C1	D2
A2	B2	C2	D2
A2	B3	C2	D3
A3	B3	C2	D4

$A \rightarrow B$,
$A \rightarrow C$,
$C \rightarrow A$,
$A \rightarrow D$,
$B \rightarrow D$,
$AB \rightarrow D$

Inference Rules for FDs

Let F be a set of FDs satisfied by R, and X, Y, Z $\subset$ R.

- Armstrong's Axioms (1974) for deriving new FDs

(IR1) *Reflexivity*: If X $\supseteq$ Y, then X $\rightarrow$ Y is satisfied by R.

(IR2) *Augmentation*: If X $\rightarrow$ Y is satisfied by R, then XZ $\rightarrow$ YZ is also satisfied by R.

(IR3) *Transitivity*: If X $\rightarrow$ Y and Y $\rightarrow$ Z are satisfied by R, then so is X $\rightarrow$ Z.

Additionally one may use :

- Decomposition : if X->Y,Z then X->Y and X->Z
- Union : if X->Y and X->Z then X->Y,Z
- Pseudo-transitivity : if X->Y and Y,T->Z then X,T->Z

Normal Forms

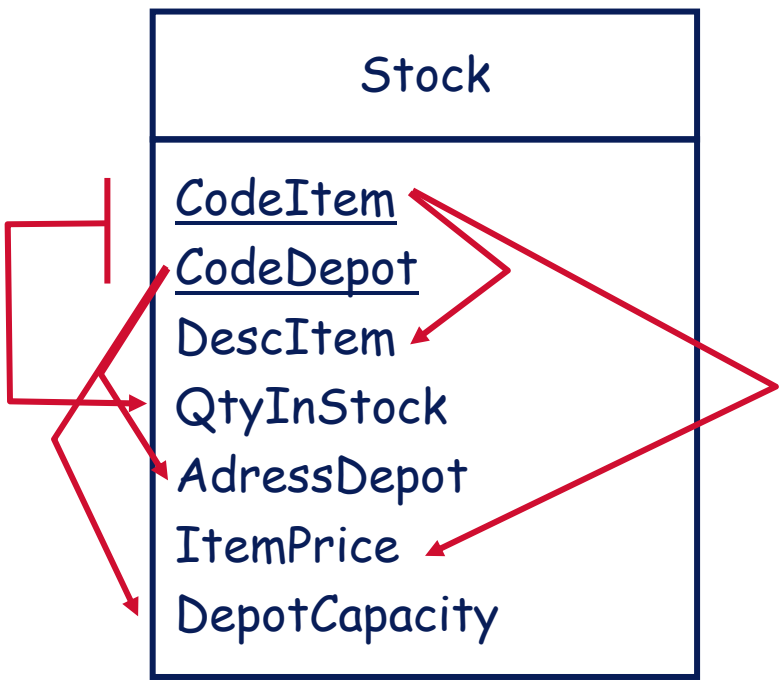
- Normal form definitions are based on the notion of DF
- 1NF,2NF,3NF,BCNF provide rising degrees of protection against data redundancy and anomalies

- A relation is in first normal form 1NF if it has a key : each attribute is determined by the key and non repetitive.

- Example :
- Offer(NumF,List of sports) : i.e. (101,{FOOT,BASKET,KART}) is not 1NF
- Offer(NumF,Sport) is 1NF : (101,FOOT),(101,BASKET),(101,KART)

Second Normal Form 2NF

- A DF X,Y->Z is *elementary* iff. X->Z and Y->Z are false.
- A relation is 2NF, if it is 1NF and
  - the key is composed of a single attribute,
  - or the attributes of the key have an elementary DF over every other attribute



Example:

CodeItem,CodeDepot -> DescItem

CodeItem,CodeDepot -> DepotCapacity

Are decomposable :

CodeItem -> DescItem

CodeDepot -> DepotCapacity

Not 2NF !!

Third Normal Form 3NF

- A DF X->Y is direct if there does not exist Z different from X and Y such that X->Z and Z->Y. We further suppose that Z->X is false.
- A relation is 3NF, if it is 2NF and the dependencies between the key and the other attributes are *elementary and direct*.

- Example:
- Actor(NumA,NameA,BirthA,BirthTownA,BirthCountryA)
- These DF are *direct :*
  - NumA -> NameA
  - NumA -> BirthA
  - NumA ->BirthTownA
  - BirthTownA -> BirthCountryA
- NumA->BirthCountryA is not because:
  - BirthTownA -> BirthCountryA

Boyce Codd NF

- 3NF does not preclude existence of DF from attributes not part of the key to an attribute that is part of the key. Therefore anomalies may remain. BCNF gives better protection, however :
  - 3NF is really mandatory !! You should always check that your logical schema is 3NF. Maybe later optimization choices will degrade it to 2NF, but a good design should produce 3NF relations.
  - Problem of decomposing : more joins in requests => response time is bad (complexity of join is quadratic !).
  - Extreme solution : single table, a lot of columns containing null values… (shudder !) NOT recommended, still exists in practice : all operations become selections (linear complexity, less with well chosen indexes).
- A relation is BCNF if it is 3NF and it does not contain any DF except K->A, where K is the (whole) key, and A an arbitrary attribute not in the key.
- Example : Adress(Town,Street,Zip) : is in 3NF but not BCNF if Zip->Town
- Decompose into : ZipT(Zip,Town) and Adress(Zip,Street)