Database design

Normalization

Normalization

- Introduction
- Normal form of Boyce Codd
- Decomposition in normal form
- Properties of decompositions
- Lossless decomposition
- Conservation of dependencies

Normalization

Introduction

Normalization

- Normalization is a process which, starting from a non-normalized relational schema, obtains a normalized relational schema.
- Normalization is not a design methodology, but a verification tool.

Normalization and ER model

- The design methodology based on ER schemas normally produces normalized relational schemas.
- Normalization checks can also be applied to ER schemas.

Example

Exam Passed

StudentID	Residence	CodCourse	CourseName	Vote
s94539	Milan	04FLYCY	Electronic calculators	30
s94540	Turin	01FLTCY	Database design	26
s94540	Turin	01KPNCY	Computer network	28
s94541	Pescara	01KPNCY	Computer network	29
s94542	Lecce	04FLYCY	Electronic calculators	25

Example: constraints

- The primary key is the pair StudentID, CodCourse
- The residence of each student is unique and is an attribute of the student alone, regardless of the exams he or she has passed
- The name of the course is unique and is a function of the course only, regardless of which students pass the corresponding exam

Redundancy and Anomalies

- In all rows where a student appears, his or her residence is repeated
  - redundancy

Redundancy and Anomalies

- In all rows where a student appears, his or her residence is repeated
  - redundancy
- If a student's residence changes, all the rows in which it appears must be modified at the same time
  - update anomaly

Redundancy and Anomalies

- If a new student enrols at university, he or she cannot be entered in the database until he or she passes the first exam
  - insertion anomaly

Redundancy and Anomalies

- If a new student enrols at university, he or she cannot be entered in the database until he or she passes the first exam
  - insertion anomaly
- If a student withdraws from studies, it is not possible to keep track of his residence
  - deletion anomaly

Redundancy

- A single relation is used to represent heterogeneous information
  - some data are repeated in different tuples without adding new information
    - redundant data

Anomalies

- Redundant information must be updated atomically (all at the same time)

Anomalies

- Redundant information must be updated atomically (all at the same time)
- The deletion of a tuple implies the deletion of all concepts represented in it
  - including those that might still be valid

Anomalies

- Redundant information must be updated atomically (all at the same time)
- The deletion of a tuple implies the deletion of all concepts represented in it
  - including those that might still be valid
- The insertion of a new tuple is only possible if at least the complete information about the primary key exists
  - it is not possible to insert the part of the tuple relating to only one concept

Normalization

Boyce-Codd normal form

Functional dependence

- It is a special type of integrity constraint
- It describes functional links between the attributes of a relation

Functional dependence

- It is a special type of integrity constraint
- It describes functional links between the attributes of a relation
- Esxample: the residence is unique for each student
    - each time the same student appears, the value is repeated
    - the value of StudentID determines the value of Residence

Functional dependence

- A relation r satisfies the functional dependence $X \rightarrow Y$ if, for each pair t_1, t_2 of tuples of r, having the same values for attributes in X, t_1 and t_2 also have the same values for attributes in Y
  - X determines Y (in r)

Functional dependence

- A relation r satisfies the functional dependence $X \rightarrow Y$ if, for each pair t_1, t_2 of tuples of r, having the same values for attributes in X, t_1 and t_2 also have the same values for attributes in Y
  - X determines Y (in r)
- Examples

StudentID → Residence

StudentID CodCourse → NameCourse

Non-trivial dependence

- The dependence

  StudentID CodCourse → CodCourse

  The dependency is trivial because CodCourse is part of both sides
- A functional dependence X → Y is non-trivial if no attribute in X appears among the attributes in Y

Functional dependencies and keys

- Given a key K of a relation r

$$K \rightarrow \text{any other attribute of } r$$
$$\text{(or set of attributes)}$$

- Examples
  - StudentID CodCourse → Residence
  - StudentID CodCourse → NameCourse
  - StudentID CodCourse → Vote

Functional dependencies and anomalies

- Anomalies are caused by attribute properties involved in functional dependencies
  - Examples
    - StudentID → Residence
    - CodCourse → NameCourse

Functional dependencies and anomalies

- Anomalies are caused by attribute properties involved in functional dependencies
  - Examples
    - StudentID → Residence
    - CodCourse → NameCourse
- Functional dependencies on keys do not give rise to anomalies
  - Example
    - StudentID CodCourse → Vote

Functional dependencies and anomalies

- The anomalies are caused by
  - the inclusion of mutually independent concepts in the same relation

Functional dependencies and anomalies

- The anomalies are caused by
  - the inclusion of mutually independent concepts in the same relation
  - functional dependencies $X \rightarrow Y$ allowing for multiple tuples with the same value of X
    - X doesn't contain a key

Boyce Codd normal form (BCNF)

- BCNF = Boyce Codd Normal Form
- A relation r is in BCNF if, for every (non-trivial) functional dependency $X \rightarrow Y$ defined on it, X contains a key of r (X is superkey of r)
- Anomalies and redundancies are not present in BCNF reports because independent concepts are separated in different reports

Normalization

Normal form decomposition

BCNF decomposition

- Normalization
  - process of replacing a non-normalised relation by two or more relations in BCNF

BCNF decomposition

- Normalization
  - process of replacing a non-normalised relation by two or more relations in BCNF
- Criteria
  - a relation representing several independent concepts is decomposed into smaller relations, one for each concept, by means of functional dependencies

BCNF decomposition

- The new relationships are obtained by projections onto the sets of attributes corresponding to the functional dependencies
- The keys of the new relations are the left parts of the functional dependencies
  - the new relations are in BCNF

Example

- Functional dependencies in the example
  - StudentID → Residence
  - CodCourse → NameCourse
  - StudentID CodCourse → Vote

Example

- By

  R (StudentID, Residence, CodCourse NameCourse, Vote)

- The relations in BCNF are

  R_1 (StudentID, Residence) = $\pi_{StudentID,\ Residence} R$

  R_2 (CodCourse, NameCourse) = $\pi_{CodCourse,\ NameCourse}\ R$

  R_3 (StudentID, CodCourse, Vote) =

  $\pi_{StudentID,\ CodCourse,\ Vote}\ R$

Example

R_1

StudentID	Residence
s94539	Milan
s94540	Turin
s94541	Pescara
s94542	Lecce

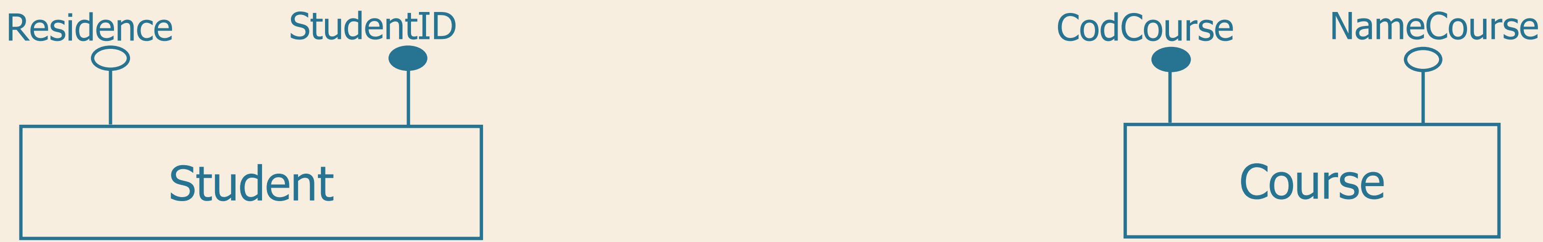
R_2

CodCourse	NameCourse
04FLYCY	Electronic calculators
01FLTCY	Database design
01KPNCY	Computer network

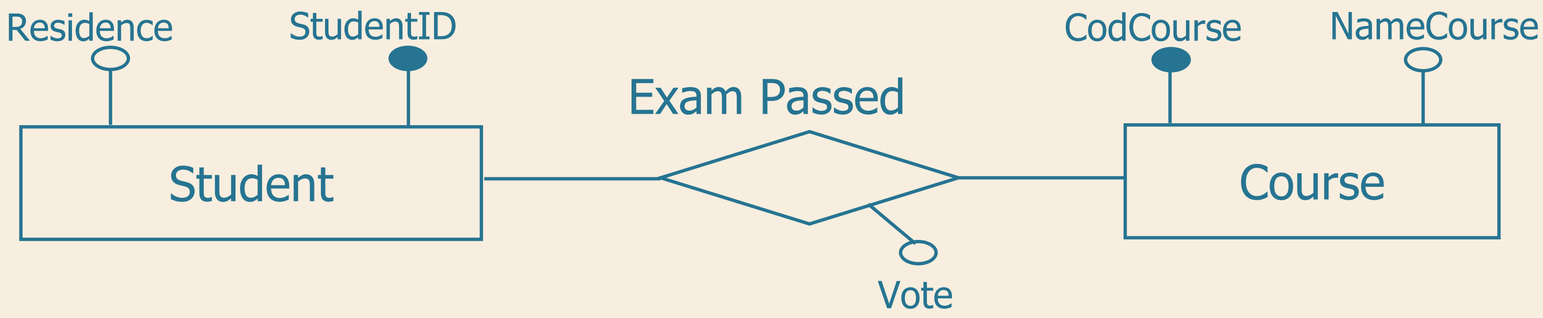
R_3

StudentID	CodCourse	Vote
s94539	04FLYCY	30
s94540	01FLTCY	26
s94540	01KPNCY	28
s94541	01KPNCY	29
s94542	04FLYCY	25

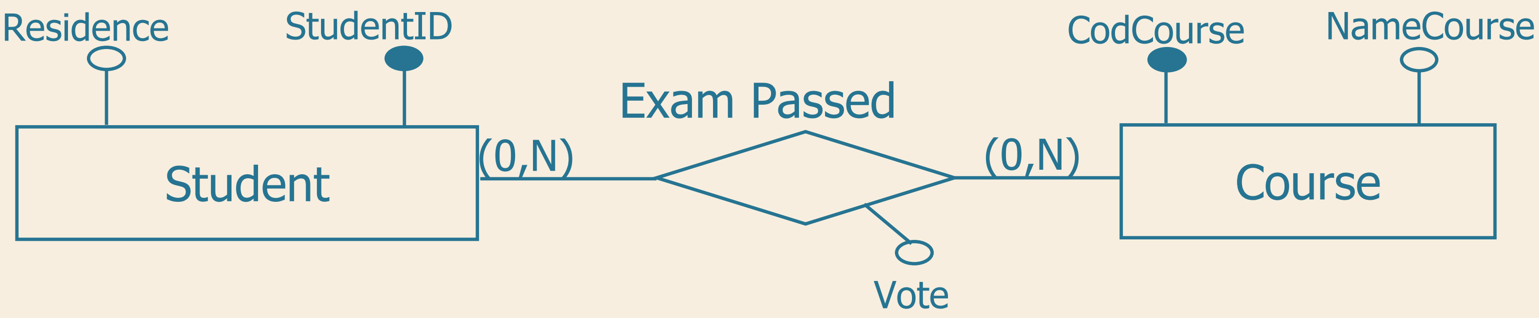
Example: corresponding ER scheme



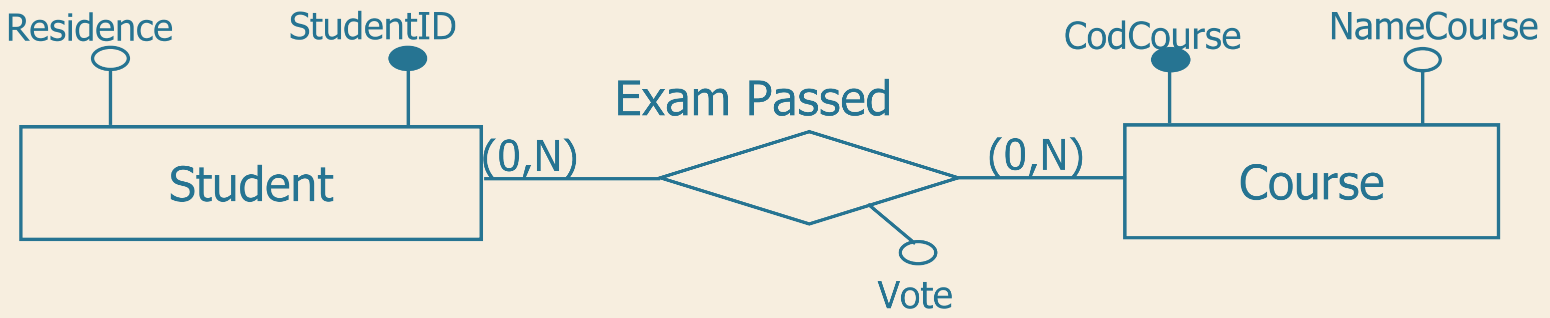
Example: corresponding ER scheme



Example: corresponding ER scheme



Example: relational logic scheme



Student (StudentID, Residence)

Course (<u>CodCourse</u>, NameCourse)

Exam Passed (<u>StudentID</u>, <u>CodCourse</u>, Vote)

Normalization

Decomposition properties

Decomposition properties

- Are all decompositions acceptable?
  - essential properties for “good” decomposition
- Problems
  - information loss
  - loss of dependencies

Example

Employee	Category	Salary
Rossi	2	1800
Verdi	3	1800
Bianchi	4	2500
Neri	5	2500
Bruni	6	3500

R (Employee, Category, Salary)

Example

Employee	Category	Salary
Rossi	2	1800
Verdi	3	1800
Bianchi	4	2500
Neri	5	2500
Bruni	6	3500

R (<u>Employee</u>, Category, Salary)

Employee $\rightarrow$ Category

Example

Employee	Category	Salary
Rossi	2	1800
Verdi	3	1800
Bianchi	4	2500
Neri	5	2500
Bruni	6	3500

R (Employee, Category, Salary)

Employee → Category

Employee → Salary

Example

Employee	Category	Salary
Rossi	2	1800
Verdi	3	1800
Bianchi	4	2500
Neri	5	2500
Bruni	6	3500

R (Employee, Category, Stipendio)

Employee → Category

Employee → Salary

Category → Salary

Normalization

Lossless Decomposition

Example: decomposition (n.1)

R (Employee, Category, Salary)

- Decomposition based on functional dependencies

Employee → Salary

Category → Salary

Example: decomposition (n.1)

R (Employee, Category, Salary)

- Decomposing

  R_1 (Employee, Salary) =

  $\pi_{Employee,\ Salary}\ R$

Example: decomposition (n.1)

R (Employee, Category, Salary)

- Decomposing

  R_1 (Employee, Salary) =

  $\pi_{Employee,\ Salary}\ R$

Employee	Salary
Rossi	1800
Verdi	1800
Bianchi	2500
Neri	2500
Bruni	3500

Example: decomposition (n.1)

R (Employee, Category, Salary)

- Decomposing

R_1 (Employee, Salary) = $\pi_{\text{Employee, Salary}}$ R

Employee	Salary
Rossi	1800
Verdi	1800
Bianchi	2500
Neri	2500
Bruni	3500

R_2 (Category, Salary) = $\pi_{\text{Category, Salary}}$ R

Category	Salary
2	1800
3	1800
4	2500
5	2500
6	3500

Example: recomposition (n.1)

- Recomposing

$$R_1 \bowtie R_2$$

Example: recomposition (n.1)

- Recomposing

$$R_1 \bowtie R_2$$

Employee	Category	Salary
Rossi	2	1800

Example: recomposition (n.1)

- Recomposing

$$R_1 \bowtie R_2$$

Employee	Category	Salary
Rossi	2	1800
Rossi	3	1800

Example: recomposition (n.1)

- Recomposing

$$R_1 \bowtie R_2$$

Employee	Category	Salary
Rossi	2	1800
Rossi	3	1800
Verdi	2	1800

Example: recomposition (n.1)

- Recomposing

$$R_1 \bowtie R_2$$

Employee	Category	Salary
Rossi	2	1800
Rossi	3	1800
Verdi	2	1800
Verdi	3	1800

Example: recomposition (n.1)

- Recomposing

$$R_1 \bowtie R_2$$

Employee	Category	Salary
Rossi	2	1800
Rossi	3	1800
Verdi	2	1800
Verdi	3	1800
Bianchi	4	2500
…	…	…

Example: recomposition (n.1)

- Recomposing

$$R_1 \bowtie R_2$$

Employee	Category	Salary
Rossi	2	1800
Rossi	*3*	*1800*
Verdi	*2*	*1800*
Verdi	3	1800
Bianchi	4	2500
…	…	…

"spurious" tuples

Example: recomposition (n.1)

- Recomposing

$$R_1 \bowtie R_2$$

Employee	Category	Salary
Rossi	2	1800
Rossi	*3*	*1800*
Verdi	*2*	*1800*
Verdi	3	1800
Bianchi	4	2500
…	…	…

"spurious" tuples

- Reconstruction with loss of information

Decomposition without loss

- The decomposition of a relation r into two sets of attributes X_1 and X_2 is lossless if the join of the projections of r into X_1 and X_2 is equal to r itself (no "spurious" tuples)
- A decomposition performed to normalize must be lossless

Decomposition without loss

- Given the relation $r(X)$ and sets of attributes X_1 and X_2 such that

  $$X = X_1 \cup X_2$$

  $$X_0 = X_1 \cap X_2$$

  if r satisfies the functional dependence

  $$X_0 \rightarrow X_1 \text{ o } X_0 \rightarrow X_2$$

  the decomposition of r on X1 and X2 is lossless
- Common attributes form a key to at least one of the decomposed relations

Example: loss of information

R_1 (<u>Employee</u>, Salary) R_2 (<u>Category</u>, Salary)

- Verification of condition for lossless decomposition

X_1 = Employee, Salary
X_2 = Category, Salary

Example: loss of information

R_1 (Employee, Salary) R_2 (Category, Salary)

- Verification of condition for lossless decomposition

X_1 = Employee, Salary
X_2 = Category, Salary
X_0 = Salary

Example: loss of information

R_1 (Employee, Salary) R_2 (Category, Salary)

- Verification of condition for lossless decomposition

X_1 = Employee, Salary
X_2 = Category, Salary
X_0 = Salary

- The attribute Salary does not satisfy the condition for lossless decomposition

Example: decomposition (n.2)

R (Employee, Category, Salary)

- Decomposition based on functional dependencies

Employee → Category

Employee → Salary

Example: decomposition (n.2)

R (Employee, Category, Salary)

- Decomposing

  R_1 (Employee, Category) =

  $\pi_{Employee,\ Salary}\ R$

Example: decomposition (n.2)

R (Employee, Category, Salary)

- Decomposing

  R_1 (Employee, Category) =

  $\pi_{\text{Employee, Salary}}$ R

Employee	Category
Rossi	2
Verdi	3
Bianchi	4
Neri	4
Bruni	5

Example: decomposition (n.2)

R (<u>Employee</u>, Category, Salary)

- Decomposing

R_1 (<u>Employee</u>, Category) = $\pi_{Employee,\ Salary}$ R

<u>Employee</u>	Category
Rossi	2
Verdi	3
Bianchi	4
Neri	4
Bruni	5

R_2 (<u>Employee</u>, Salary) = $\pi_{Category,\ Salary}$ R

<u>Employee</u>	Salary
Rossi	1800
Verdi	1800
Bianchi	2500
Neri	2500
Bruni	3500

Example: decomposition without loss?

R_1 (<u>Employee</u>, Category) R_2 (<u>Employee</u>, Salary)

$$R_1 \bowtie R_2$$

- Is decomposition without loss?

Example: decomposition without loss

R_1 (Employee, Category) R_2 (Employee, Salary)

$R_1 \bowtie R_2$

- Verifying the condition for lossless decomposition

X_1 = Employee, Category
X_2 = Employee, Salary

Example: decomposition without loss

R_1 (<u>Employee</u>, Category) R_2 (<u>Employee</u>, Salary)

$$R_1 \bowtie R_2$$

- Verifying the condition for lossless decomposition

X_1 = Employee, Category
X_2 = Employee, Salary
X_0 = Employee

Example: decomposition without loss

R_1 (Employee, Category) R_2 (Employee, Salary)

$R_1 \bowtie R_2$

- Verifying the condition for lossless decomposition

X_1 = Employee, Category
X_2 = Employee, Salary
X_0 = Employee

- The attribute Employee satisfies the condition for lossless decomposition

Normalization

Conservation of dependencies

Example: inserting a new tuple

R_1 (Employee, Category) R_2 (Employee, Salary)

- Inserting the tuple

  Employee: Gialli – Category: 3 – Salary: 3500

Example: inserting a new tuple

R_1 (Employee, Category) R_2 (Employee, Salary)

- Inserting the tuple

  Employee: Gialli – Category: 3 – Salary: 3500

Employee	Category
Rossi	2
Verdi	3
Bianchi	4
Neri	4
Bruni	5

Example: inserting a new tuple

R_1 (Employee, Category) R_2 (Employee, Salary)

- Inserting the tuple

  Employee: Gialli – Category: 3 – Salary: 3500

Employee	Category
Rossi	2
Verdi	3
Bianchi	4
Neri	4
Bruni	5

Gialli	3

Example: inserting a new tuple

R_1 (Employee, Category) R_2 (Employee, Salary)

- Inserting the tuple

  Employee: Gialli – Category: 3 – Salary: 3500

Employee	Category
Rossi	2
Verdi	3
Bianchi	4
Neri	4
Bruni	5

Gialli	3

Employee	Salary
Rossi	1800
Verdi	1800
Bianchi	2500
Neri	2500
Bruni	3500

Example: inserting a new tuple

R_1 (Employee, Category) R_2 (Employee, Salary)

- Inserting the tuple

  Employee: Gialli – Category: 3 – Salary: 3500

Employee	Category
Rossi	2
Verdi	3
Bianchi	4
Neri	4
Bruni	5

Gialli	3

Employee	Salary
Rossi	1800
Verdi	1800
Bianchi	2500
Neri	2500
Bruni	3500

Gialli	3500

Example: inserting a new tuple

- What happens if I insert the tuple (Gialli,3500) in R_2?
  - in the original report insertion is prohibited because it causes the violation of the dependency Category → Salary
  - in the decomposition it is no longer possible to recognise any violation, since the attributes Category and Salary are in separate relationships
- The dependence between Category and Salary has been lost

Conservation of dependencies

- A decomposition preserves dependencies if each of the functional dependencies of the original schema is present in one of the decomposed relations
- Dependencies should be retained to ensure that the same constraints are satisfied in the decomposed schema as in the original schema

Example: decomposition (n.3)

R (Employee, Category, Salary)

- Decomposition based on functional dependencies

Employee → Category

Category → Salary

Example: decomposition (n.3)

R (<u>Employee</u>, Category, Salary)

- Decomposing

  R_1 (<u>Employee</u>, Category) =

  $\pi_{\text{Employee, Category}}\ R$

<u>Employee</u>	Category
Rossi	2
Verdi	3
Bianchi	4
Neri	5
Bruni	6

Example: decomposition (n.3)

R (Employee, Category, Salary)

- Decomposing

R_1 (Employee, Category) = $\pi_{Employee,\ Category}\ R$

Employee	Category
Rossi	2
Verdi	3
Bianchi	4
Neri	5
Bruni	6

R_2 (Category, Salary) = $\pi_{Category,\ Salary}\ R$

Category	Salary
2	1800
3	1800
4	2500
5	2500
6	3500

Example

- Recomposing

$$R_1 \bowtie R_2$$

Example: decomposition without loss

- Recomposing

$$R_1 \bowtie R_2$$

- Condition check for lossless decomposition

X_1 = Employee, Category

X_2 = Category, Salary

Example: decomposition without loss

- Recomposing

$$R_1 \bowtie R_2$$

- Condition check for lossless decomposition

  X_1 = Employee, Category
  X_2 = Category, Salary
  X_0 = Category

Example: decomposition without loss

- Recomposing

$$R_1 \bowtie R_2$$

- Condition check for lossless decomposition

  X_1 = Employee, Category

  X_2 = Category, Salary

  X_0 = Category

- The attribute Category satisfies the condition for lossless decomposition

Example: Conservation of functional dependencies

- Recomposing

$$R_1 \bowtie R_2$$

- Conserved functional dependencies

$$\text{Employee} \rightarrow \text{Category}$$

$$\text{Category} \rightarrow \text{Salary}$$

Example: Conservation of functional dependencies

- Recomposing

$$R_1 \bowtie R_2$$

- Conserved functional dependencies

$$\text{Employee} \rightarrow \text{Category}$$
$$\text{Category} \rightarrow \text{Salary}$$

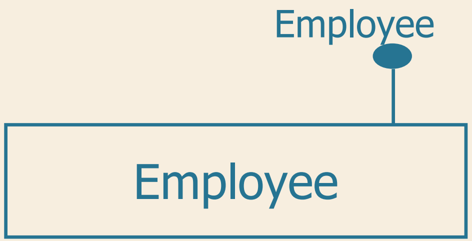
- Functional dependency

$$\text{Employee} \rightarrow \text{Salary}$$

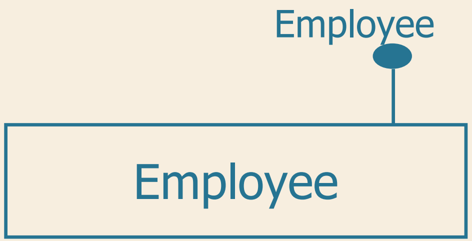
can be reconstructed from

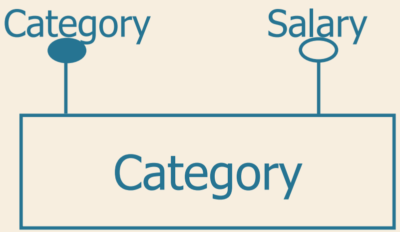
$$\text{Employee} \rightarrow \text{Category}$$
$$\text{Category} \rightarrow \text{Salary}$$

Example: corresponding ER scheme

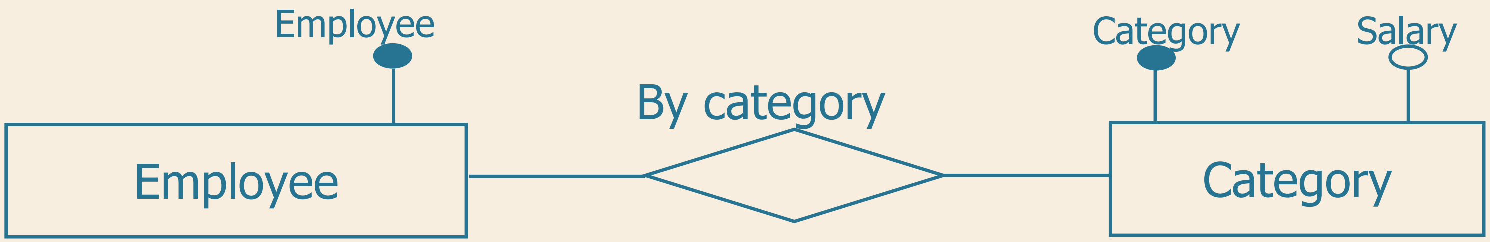


Example: corresponding ER scheme

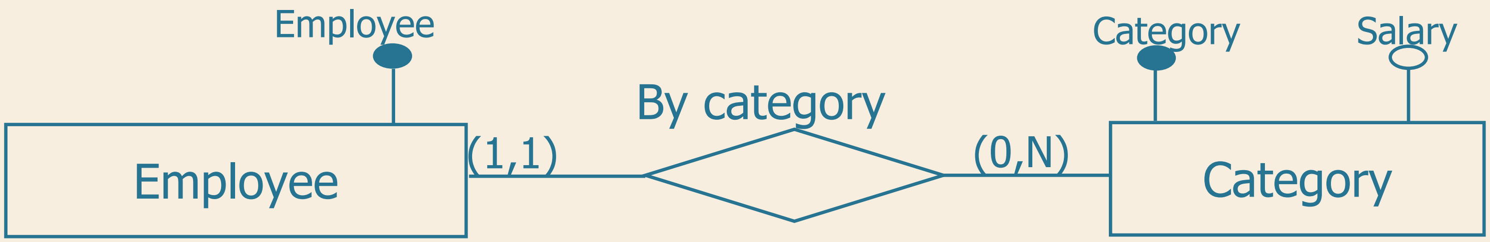




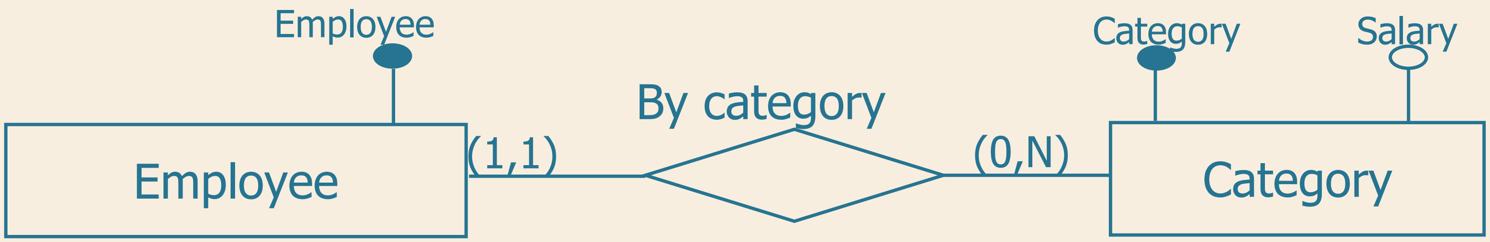
Example: corresponding ER scheme



Example: corresponding ER scheme



Example: ER schema



Employee (<u>Employee</u>, Category)
Category (<u>Category</u>, Salary)

Quality of a decomposition

- Decompositions must always satisfy the properties
  - decomposition without loss
    - ensures that the information in the original relation is accurately reconstructed (without spurious tuples) from that in the decomposed relations
  - conservation of dependencies
    - ensures that the decomposed relations have the same capacity as the original relation to represent the integrity constraints