Database design

Normalization

1

Normalization

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

2

2

Normalization

Introduction

3

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.

4

4

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.

5

5

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

6

6

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

7

7

Redundancy and Anomalies

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

8

8

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

9

9

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

10

10

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

11

11

Redundancy

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

12

12

Anomalies

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

13

13

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

14

14

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

15

15

Normalization

Boyce-Codd normal form

16

Functional dependence

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

17

17

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

18

18

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)

19

19

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

20

20

Non-trivial dependence

- The dependence

  StudentID CodCourse → CodCourse

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

21

21

Functional dependencies and keys

- Given a key K of a relation r

  K → any other attribute of r

  (or set of attributes)
- Examples
  - StudentID CodCourse → Residence
  - StudentID CodCourse → NameCourse
  - StudentID CodCourse → Vote

22

22

Functional dependencies and anomalies

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

23

23

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

24

24

Functional dependencies and anomalies

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

25

25

Functional dependencies and anomalies

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

26

26

Boyce Codd normal form (BCNF)

- BCNF = Boyce Codd Normal Form
- A relation r is in BCNF if, for every (non-trivial) functional dependency X → 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

27

27

Normalization

Normal form decomposition

28

BCNF decomposition

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

29

29

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

30

30

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

31

31

Example

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

32

32

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

33

33

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

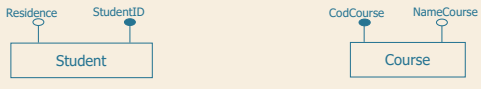
R_3

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

34

34

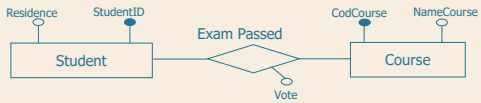
Example: corresponding ER scheme



35

35

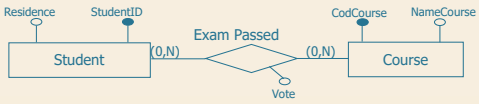
Example: corresponding ER scheme



36

36

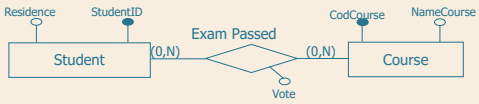
Example: corresponding ER scheme



37

37

Example: relational logic scheme



Student (StudentID, Residence)
Course (CodCourse, NameCourse)
Exam Passed (StudentID, CodCourse, Vote)

38

38

Normalization

Decomposition properties

39

Decomposition properties

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

40

40

Example

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

R (Employee, Category, Salary)

41

41

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

42

42

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

43

43

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

44

44

Normalization

Lossless Decomposition

45

Example: decomposition (n.1)

R (Employee, Category, Salary)

- Decomposition based on functional dependencies

Employee→ Salary
Category→ Salary

46

46

Example: decomposition (n.1)

R (Employee, Category, Salary)

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

47

47

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

48

48

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

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

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

49

49

Example: recomposition (n.1)

- Recomposing

$R_1 \bowtie R_2$

50

50

Example: recomposition (n.1)

- Recomposing

$R_1 \bowtie R_2$

Employee	Category	Salary
Rossi	2	1800

51

51

Example: recomposition (n.1)

- Recomposing

$R_1 \bowtie R_2$

Employee	Category	Salary
Rossi	2	1800
Rossi	3	1800

52

52

Example: recomposition (n.1)

- Recomposing

$R_1 \bowtie R_2$

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

53

53

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

54

54

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
...	...	...

55

55

Example: recomposition (n.1)

- Recomposing

$R_1 \bowtie R_2$

Employee	Category	Salary	
Rossi	2	1800	
Rossi	*3*	*1800*	"spurious" tuples
Verdi	*2*	*1800*	"spurious" tuples
Verdi	3	1800	
Bianchi	4	2500	
...	...	...	

56

56

Example: recomposition (n.1)

- Recomposing

$R_1 \bowtie R_2$

Employee	Category	Salary	
Rossi	2	1800	
Rossi	*3*	*1800*	"spurious" tuples
Verdi	*2*	*1800*	"spurious" tuples
Verdi	3	1800	
Bianchi	4	2500	
...	...	...	

- Reconstruction with loss of information

57

57

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

58

58

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

  $X_0 \rightarrow X_1$ 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

59

59

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

60

60

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

61

61

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

62

62

Example: decomposition (n.2)

R (Employee, Category, Salary)

- Decomposition based on functional dependencies

  Employee→ Category
  Employee→ Salary

63

63

Example: decomposition (n.2)

R (Employee, Category, Salary)

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

64

64

Example: decomposition (n.2)

R (Employee, Category, Salary)

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

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

65

65

Example: decomposition (n.2)

R (Employee, Category, Salary)

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

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

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

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

66

66

Example: decomposition without loss?

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

$R_1 \bowtie R_2$

- Is decomposition without loss?

67

67

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

68

68

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

69

69

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

70

70

Normalization

Conservation of dependencies

71

Example: inserting a new tuple

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

- Inserting the tuple
  Employee: Gialli – Category: 3 – Salary: 3500

72

72

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

73

73

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

74

74

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

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

Gialli	3

75

75

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

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

Gialli	3

76

76

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

77

77

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

78

78

Example: decomposition (n.3)

R (Employee, Category, Salary)

- Decomposition based on functional dependencies

Employee → Category
Category → Salary

79

79

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

80

80

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

81

81

Example

- Recomposing

$R_1 \bowtie R_2$

82

82

Example: decomposition without loss

- Recomposing

$R_1 \bowtie R_2$

- Condition check for lossless decomposition

X_1 = Employee, Category
X_2 = Category, Salary

83

83

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

84

84

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

85

85

Example: Conservation of functional dependencies

- Recomposing

  $R_1 \bowtie R_2$

- Conserved functional dependencies

  Employee → Category
  Category → Salary

86

86

Example: Conservation of functional dependencies

- Recomposing

  $R_1 \bowtie R_2$

- Conserved functional dependencies

  Employee → Category
  Category → Salary

- Functional dependency

  Employee → Salary

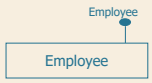
  can be reconstructed from

  Employee → Category
  Category → Salary

87

87

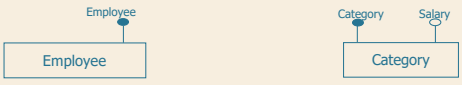
Example: corresponding ER scheme



88

88

Example: corresponding ER scheme



89

89

Example: corresponding ER scheme

90

90

Example: corresponding ER scheme

91

91

Example: ER schema

Employee (Employee, Category)
Category (Category, Salary)

92

92

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

93

93