Basi di Dati e
Sistemi Informativi I

Normalization

Normal form and normalization

- A **normal form** is a property of a relational database.
- When a relation is *non-normalized* (that is, does not satisfy a normal form), then it presents redundancies and produces undesirable behavior during update operations.
- This principle can be used to carry out quality analysis and constitutes a useful tool for database design.
- **Normalization** is a procedure that allows the non-normalized schemas to be transformed into new schemas for which the satisfaction of a normal form is guaranteed.

Example of a relation with anomalies

Employee	Salary	Project	Budget	Function
Brown	20	Mars	2	technician
Green	35	Jupiter	15	designer
Green	35	Venus	15	designer
Hoskins	55	Venus	15	manager
Hoskins	55	Jupiter	15	consultant
Hoskins	55	Mars	2	consultant
Moore	48	Mars	2	manager
Moore	48	Venus	15	designer
Kemp	48	Venus	15	designer
Kemp	48	Jupiter	15	manager

The key is made up of the attributes Employee and Project

Anomalies in the example relation

- The value of the salary of each employee is repeated in all the tuples relating to it: therefore there is a **redundancy**.
- If the salary of an employee changes, we have to modify the value in all the corresponding tuples. This problem is known as the **update anomaly**.
- If an employee stops working on all the projects but does not leave the company, all the corresponding tuples are deleted and so, even the basic information, name and salary is lost. This problem is known as the **deletion anomaly**.
- If we have information on a new employee, we cannot insert it until the employee is assigned to a project. This is known as the **insertion anomaly**.

Why these undesirable phenomena?

- **Intuitive explanation**: we have used a single relation to represent items of information of different types.
- In particular, the following independent real-world concepts are represented in the relation:
  - employees with their salaries,
  - projects with their budgets,
  - participation of the employees in the projects with their functions.

- To systematically study the principles introduced informally, it is necessary to use a specific notion: **the functional dependency**.

Functional dependencies in the example schema

- Employee → Salary

  the salary of each employee is unique and thus each time a certain employee appears in a tuple, the value of his or her salary always remains the same. *The value of the attribute Salary functionally depends on the vale of the Employee attribute: a function exists that associates with each Employee value in the raltion, a single value for the Salary attribute*

- Project → Budget

  the budget of each project is unique and thus each time a certain project appears in a tuple, the value of its budget always remains the same.

Functional dependencies

- Given a relation *r* on a schema *R(X)* and two non-empty subsets *Y* and *Z* of the attributes *X*, we say that there is a functional dependency on *r* between *Y* and *Z*, if, for each pair of tuples t_1 and t_2 of *r* having the same values on the attributes *Y*, t_1 and t_2 also have the same values of the *Z* attributes.

- A functional dependency between the attributes *Y* and *Z* is indicated by the notation $Y \rightarrow Z$.

Non-trivial functional dependencies

- We then say that a functional dependency $Y \rightarrow Z$ is **non-trivial** if no attribute in Z appears among the attributes of Y.

  – Employee $\rightarrow$ Salary is a non-trivial functional dependency

  – Employee Project $\rightarrow$ Project is a trivial functional dependency

Anomalies and functional dependencies

- In our example, the two properties causing anomalies correspond exactly to attributes involved in functional dependencies:
  - the property 'the salary of each employee is unique and depends only on the employee' corresponds to the functional dependency Employee $\longrightarrow$ Salary;
  - the property 'the budget of each project is unique and depends only on the project' corresponds to the functional dependency Project $\longrightarrow$ Budget.
- Moreover, the following property can be formalized by means of a functional dependency:
  - the property 'in each project, each of the employees involved can carry out only one function' corresponds to the functional dependency Employee Project $\longrightarrow$ Function.

Dependencies generating anomalies

- The first two dependencies generate undesirable redundancies and anomalies.

- The third dependency however never generates redundancies because, having Employee and Project as a key, the relation cannot contain two tuples with the same values of these attributes.

- **The difference is that Employee Project is a key of the relation.**

Boyce–Codd Normal Form (BCNF)

- A relation r is in **Boyce–Codd normal form** if for every (non-trivial) functional dependency $X \longrightarrow Y$ defined on it, X contains a key K of r. That is, X is a superkey for r.

- Anomalies and redundancies, as discussed above, do not appear in databases with relations in Boyce–Codd normal form, because the independent pieces of information are separate, one per relation.

Decomposition into Boyce–Codd normal form

- Given a relation that does not satisfy Boyce–Codd normal form, we can often replace it with one or more normalized relations using a process called **normalization**.

- We can eliminate redundancies and anomalies for the example relation if we replace it with the three relations, obtained by projections on the sets of attributes corresponding to the three functional dependencies.

- The keys of the relations we obtain are the left hand side of a functional dependency: the satisfaction of the Boyce–Codd normal form is therefore guaranteed.

Decomposition of the example relation

Employee	Salary
Brown	20
Green	35
Hoskins	55
Moore	48
Kemp	48

Project	Budget
Mars	2
Jupiter	15
Venus	15

Employee	Project	Function
Brown	Mars	technician
Green	Jupiter	designer
Green	Venus	designer
Hoskins	Venus	manager
Hoskins	Jupiter	consultant
Hoskins	Mars	consultant
Moore	Mars	manager
Moore	Venus	designer
Kemp	Venus	designer
Kemp	Jupiter	manager

A relation to be decomposed

Employee	Project	Branch
Brown	Mars	Chicago
Green	Jupiter	Birmingham
Green	Venus	Birmingham
Hoskins	Saturn	Birmingham
Hoskins	Venus	Birmingham

The relation satisfies the functional dependencies:

- Employee $\rightarrow$Branch
- Project $\rightarrow$Branch

A possible decomposition of the previous relation

Employee	Branch
Brown	Chicago
Green	Birmingham
Hoskins	Birmingham

Project	Branch
Mars	Chicago
Jupiter	Birmingham
Saturn	Birmingham
Venus	Birmingham

The join of the projections

Employee	Project	Branch
Brown	Mars	Chicago
Green	Jupiter	Birmingham
Green	Venus	Birmingham
Hoskins	Saturn	Birmingham
Hoskins	Venus	Birmingham
Green	Saturn	Birmingham
Hoskins	Jupiter	Birmingham

The result is different from the original relation: the information can not be reconstructed.

Lossless decomposition

- The decomposition of a relation *r* on X_1 and X_2 is **lossless** if the join of the projections of *r* on X_1 and X_2 is equal *to r* itself (that is, not containing *spurious* tuples).

- It is clearly desirable, or rather an indispensable requirement, that a decomposition carried out for the purpose of normalization is lossless.

A condition for the lossless decomposition

- Let r be a relation on X and let X_1 and X_2 be two subsets of X such that $X_1 \cup X_2$ = X. Furthermore, let $X_0 = X_1 \cap X_2$.

- If r satisfies the functional dependency $X_0 \rightarrow X_1$ or the functional dependency $X_0 \rightarrow X_2$, then the decomposition of r on X_1 and X_2 is lossless.

- In other words, we can say that r has a lossless decomposition on two relations if the set of attributes common to the relations is a key for at least one of the decomposed relations.

A lossless decomposition of the previous relation

Employee	Branch
Brown	Chicago
Green	Birmingham
Hoskins	Birmingham

Employee	Project
Brown	Mars
Green	Jupiter
Green	Venus
Hoskins	Saturn
Hoskins	Venus

Another problem with the new decomposition

- Assume we wish to insert a new tuple that specifies the participation of the employee named Armstrong, who works in Birmingham, on the Mars project.
- In the original relation an update of this kind would be immediately identified as illegal, because it would cause a violation of the $\rightarrow$ Project Branch dependency.
- On the decomposed relations however, it is not possible to reveal any violation of dependency since the two attributes Project and Branch have been separated: one into one relation and one into the other.

Preservation of dependencies

- A decomposition **preserves the dependencies** if each of the functional dependencies of the original schema involves attributes that appear all together in one of the decomposed schemas.

- It is clearly desirable that a decomposition preserves the dependencies since, in this way, it is possible to ensure, on the decomposed schema, the satisfaction of the same constraints as the original schema.

Qualities of decompositions

- Decompositions should always satisfy the properties of lossless decomposition and dependency preservation:

  – Lossless decomposition ensures that the information in the original relation can be accurately reconstructed based on the information represented in the decomposed relations.

  – Dependency preservation ensures that the decomposed relations have the same capacity to represent the integrity constraints as the original relations and thus to reveal illegal updates.

A relation not satisfying the BCNF

Manager	Project	Branch
Brown	Mars	Chicago
Green	Jupiter	Birmingham
Green	Mars	Birmingham
Hoskins	Saturn	Birmingham
Hoskins	Venus	Birmingham

Assume that the following dependencies are defined:

- Manager $\rightarrow$ Branch: each manager works at a particular branch;
- Project Branch $\rightarrow$ Manager: each project has more managers who are responsible for it, but in different branches, and each manager can be responsible for more than one project; however, for each branch, a project has only one manager responsible for it.

A problematic decomposition

- The relation is not in Boyce–Codd normal form because the left hand side of the first dependency is not a superkey.
- At the same time, no good decomposition of this relation is possible: the dependency Project Branch $\rightarrow$ Manager involves all the attributes and thus no decomposition is able to preserve it.
- We can therefore state that sometimes, Boyce–Codd normal form cannot be achieved.

A new normal form

- A relation *r* is in **third normal form** if, for each (non-trivial) functional dependency $X \rightarrow Y$ defined on it, at least one of the following is verified:

  - X contains a key K of *r*;
  - each attribute in Y is contained in at least one key of *r*.

BCNF and third normal form

- The previous schema does not satisfy the Boyce–Codd normal form, but it satisfies the third normal form:
  - The Project Branch → Manager dependency has as its left hand side a key for the relation, while Manager → Branch has a unique attribute for the right hand side, which is part of the Project Branch key.

- The third normal form is less restrictive than the Boyce–Codd normal form and for this reason does not offer the same guarantees of quality for a relation; it has the advantage however, of always being achievable.

The tolerated redundancy

Manager	Project	Branch
Brown	Mars	Chicago
Green	Jupiter	Birmingham
Green	Mars	Birmingham
Hoskins	Saturn	Birmingham
Hoskins	Venus	Birmingham

Decomposition into third normal form

- Decomposition into third normal form can proceed as suggested for the Boyce–Codd normal form:
  - a relation that does not satisfy the third normal form is decomposed into relations obtained by projections on the attributes corresponding to the functional dependencies.

- The only condition to guarantee in this process is of always maintaining a relation that contains a key to the original relation.

A restructuring of the previous relation

Manager	Project	Branch	Division
Brown	Mars	Chicago	1
Green	Jupiter	Birmingham	1
Green	Mars	Birmingham	1
Hoskins	Saturn	Birmingham	2
Hoskins	Venus	Birmingham	2

Functional dependencies:

- Manager $\rightarrow$ Branch Division: each manager works at one branch and manages one division;
- Branch Division $\rightarrow$ Manager: for each branch and division there is a single manager;
- Project Branch $\rightarrow$ Division: for each branch, a project is allocated to a single division and has a sole manager responsible.

A good decomposition of the restructured schema

Manager	Branch	Division
Brown	Chicago	1
Green	Birmingham	1
Hoskins	Birmingham	2

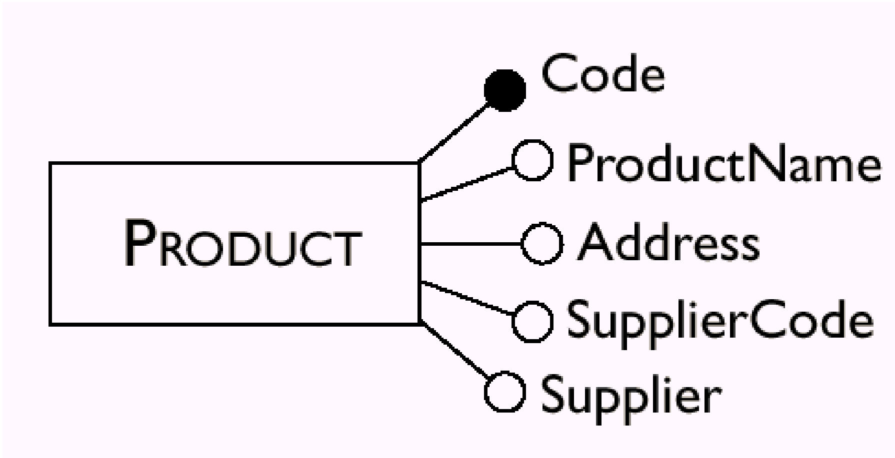
Project	Branch	Division
Mars	Chicago	1
Jupiter	Birmingham	1
Mars	Birmingham	1
Saturn	Birmingham	2
Venus	Birmingham	2

- The decomposition is lossless and the dependencies are preserved.
- This example shows that often the difficulty of achieving Boyce–Codd normal form could be due to an insufficiently accurate analysis of the application.

Database design and normalization

- The theory of normalization can be used as a basis for quality control operations on schemas, in both the conceptual and logical design phases:
  – the analysis of the relations obtained during the logical design phase can identify places where the conceptual design was inaccurate: this verification of the design is often relatively easy;
  – the ideas on which normalization is based can also be used during the conceptual design phase for the quality control of each element of the conceptual schema.

Analysis of the entity



- The attribute Code constitutes the identifier of the entity.
- The functional dependency SupplierCode $\rightarrow$ Supplier Address is verified on the attributes of the entity: all the properties of each supplier are identified by its SupplierCode.
- The entity violates the third normal form since this dependency has a left hand side that does not contain the identifier and a right hand side made up of attributes that are not part of the key.

Decomposition of the PRODUCT entity

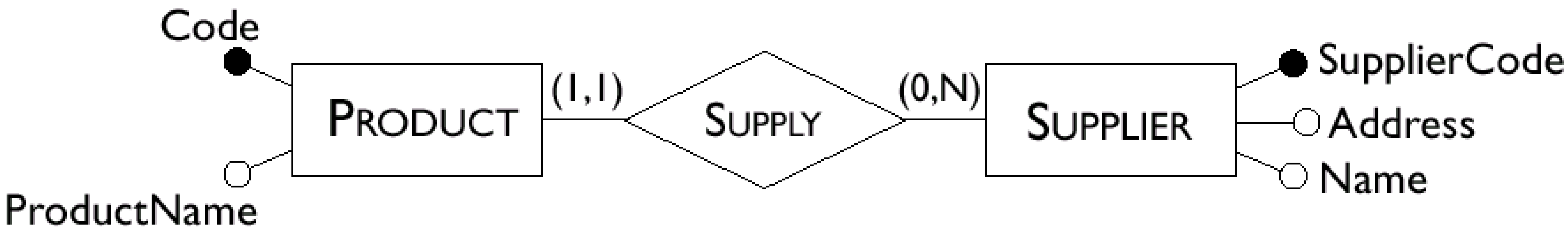
- The concept of supplier is independent of that of product and has associated properties (code, surname and address)

- It is appropriate to model the concept of supplier by means of a separate entity, where SupplierCode is the identifier, and Name and Address are further attributes.

- It is evident that if we separate product and supplier in two entities, it is appropriate that these entities be linked by a releationship.

- Since there is a functioanal dependency from Code to SupplierCode, we are sure that each product has at most one supplier (maximum cardinality 1).
- Since there is no dependency from SupplierCode to Code, we have an unlimeted maximum cardinality (N) for Supplier in the relationship.

Decomposition of the PRODUCT entity



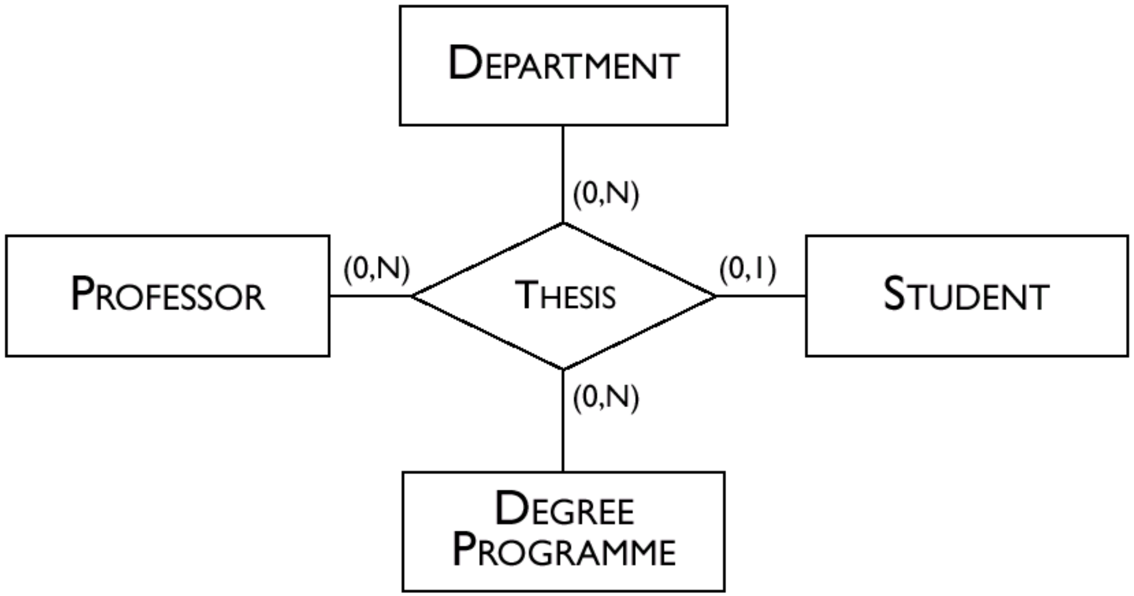
- It is possible to verify that the decomposition satisfies the two fundamental properties.

- It is lossless decomposition, becauseon the basis of the on-to-many relationship it is possible to recostruct the values of the attributes of the original entity.
- It preserves the dependencies, because each of the dependencies is embedded in one of the entities or it can be reconstructed from them.

Analysis of the relationship

- It is easy to verify that each binary relation is in the third normal form (and also in Boyce-Codd normal form), the verification of normalization is carried out only on the *n*-ary relationships, that is, on those which involve at least there entities.

- Consider



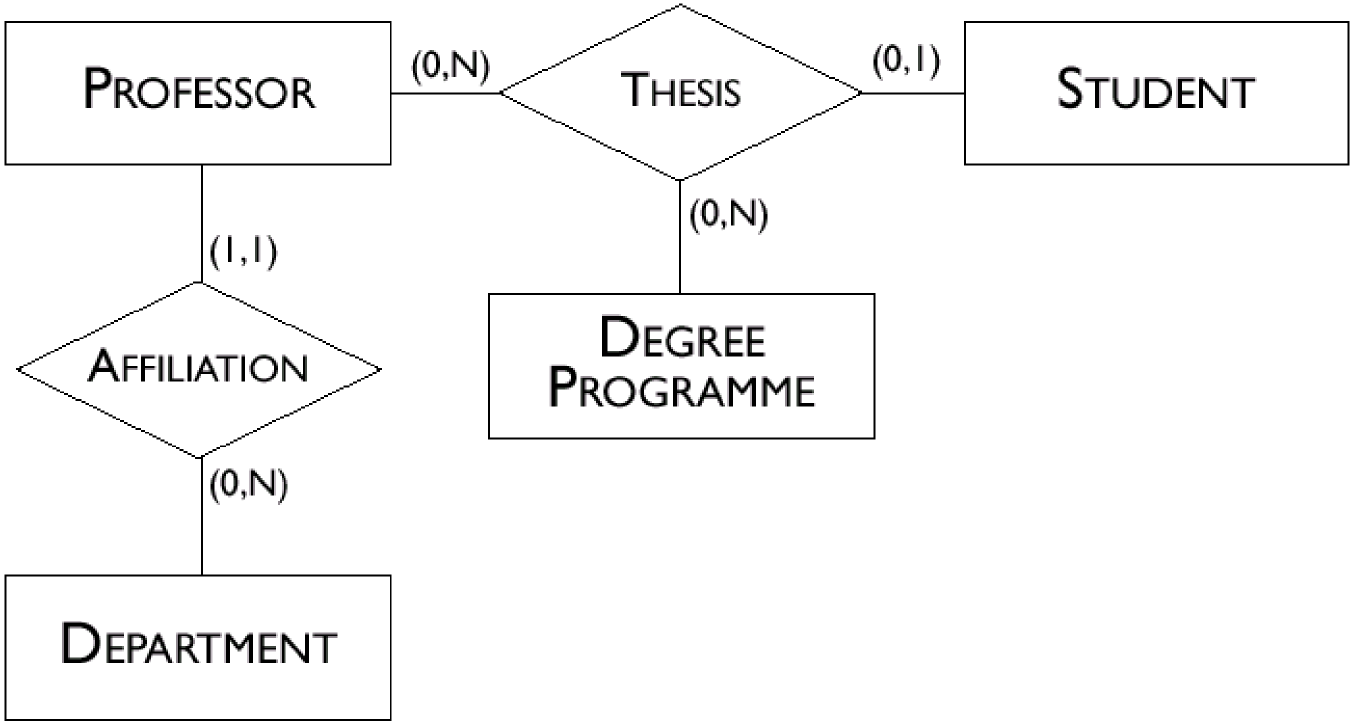
Analysis of the relationship

- The following functional dependencies can be identified:
  - STUDENT $\rightarrow$ DEGREEPROGRAMME (each student is enrolled in a degree programme)
  - STUDENT $\rightarrow$ PROFESSOR (each student writes a thesis under the supervision of a single professor)
  - PROFESSOR $\rightarrow$ DEPARTMENT (each professor belongs to a single department and the students under her supervision write their theses under the care of that department)

- The (unique) key of the relationship is STUDENT (given a student, the degree programme, the professor and the department are unambiguously identified)
- Therefore, the third functional dependency causes a violation of the third normal form.

Analysis of the relationship

- The following is a decomposition where the two relationships are both in the third normal form (and in Boyce-Codd normal form)



Further observations on the decomposed schema

- The relationship THESIS is in third normal form, because its key is made up of the STUDENT entity, and the only dependencies that exist on it are those that have this entity as left hand side.
- On the other hand, the properties described by the two dependencies are independent of each other: not all students are writing theses and so not all of them have supervisors.
- From the normalization point of view, this situation does not present problems.
- However, at the conceptual modeling level, we must distinguish among the various concepts.
- We can therefore conclude that it would be appropriate to decompose the relationship further, obtaining two relationships, one for each of the two concepts.

A further decomposition