Databases

Normalization

Normal Forms

- "Normal form" is a property of a relational database, that guarantees its quality, i.e. the absence of certain flaws
- When a relation is not in normal form:
  - it has redundancies,
    - it can have undesired behaviours during updates
- Normal forms are usually defined on the relational model, but they make sense also in other contexts, for example in the E-R model

Normalization

- Such task allows to transform non-normalized schema into schema that satisfy the normal form

- Normalization has to be used as a **verification technique** to test the result of the database design

- It is not a methodology for database design

A relation with anomalies

Employee	Wage	Project	Budget	Role
Jones	20	Mars	2	Technician
Smith	35	Jupiter	15	Designer
Smith	35	Venus	15	Designer
Williams	55	Venus	15	Chief
Williams	55	Jupiter	15	Consultant
Williams	55	Mars	2	Consultant
Brown	48	Mars	2	Chief
Brown	48	Venus	15	Designer
White	48	Venus	15	Designer
White	48	Jupiter	15	Director

Anomalies

- **Redundancy**: each employee's wage is repeated throughout the relation
- **Update Anomaly**: when an employee's wage changes, then we have to change all its occurrences
- **Deletion Anomaly**: when an employee stops participating in all its project, it is completely removed from the database
- **Insertion Anomaly**: we cannot create an employee without an associated project

Why is this situation undesirable?

- Because different pieces of information are represented within the same relation
  - Employees and their wages
  - Projects and their budgets
  - The role of each employee within a project they're working on

Functional Dependencies

In order to study in a systematic way the concept we just introduced, we need to use the functional dependency, which is a specific integrity constraint for the relational model that describes functional bounds among the attributes of a relation

Functional Dependencies: definition

Given a relation **r** having a schema **R(X)**, and two non-empty subsets of attributes **Y** and **Z** of **X**, we say that it exists a **functional dependency** (FD) between **Y** and **Z** if, for each couple of tuples $\mathbf{t_1}$ and $\mathbf{t_2}$ of **r** having the same values on the attributes **Y**, it results that $\mathbf{t_1}$ and $\mathbf{t_2}$ have the same values also on the attributes **Z**

A relation with anomalies

Employee	Wage	Project	Budget	Role
Jones	20	Mars	2	Technician
Smith	35	Jupiter	15	Designer
Smith	35	Venus	15	Designer
Williams	55	Venus	15	Chief
Williams	55	Jupiter	15	Consultant
Williams	55	Mars	2	Consultant
Brown	48	Mars	2	Chief
Brown	48	Venus	15	Designer
White	48	Venus	15	Designer
White	48	Jupiter	15	Director

Functional Dependencies: notation

$$Y \rightarrow Z$$

Examples:

Employee → Wage

Project → Budget

Employee Project → Role

Functional Dependencies: example

Employee	**Wage**	**Project**	**Budget**	**Role**
Jones	20	Mars	2	Technician
Smith	35	Jupiter	15	Designer
Smith	35	Venus	15	Designer
Williams	55	Venus	15	Chief
Williams	55	Jupiter	15	Consultant
Williams	55	Mars	2	Consultant
Brown	48	Mars	2	Chief
Brown	48	Venus	15	Designer
White	48	Venus	15	Designer
White	48	Jupiter	15	Director

Employee → Wage

Project → Budget

Employee Project → Role

FD, some properties

- Employee Project → Project

- Such FD is "trivial" (it is always true)
- A FD Y → A is **nontrivial** if one of the following conditions are met:
  - A is an attribute and doesn't belong to Y
  - A is a set and no attributes in A belongs to Y

Anomalies depend on some FDs

- Employees must have only one Wage

$$\text{Employee} \rightarrow \text{Wage}$$

- Projects must have only one Budget

$$\text{Project} \rightarrow \text{Budget}$$

Not all the FD provoke anomalies

- In each Project, an Employee has only one Role

$$\text{Employee Project} \rightarrow \text{Role}$$

- The constraint is “trivially” satisfied because Employee Project is a key

FD and anomalies

- The third FD **is a key** (Employee Project) and **does not cause anomalies**
- The first two FD **are not keys** and **cause anomalies**
- The relation contains some informations linked to the key and other informations linked to attributes that do not compose a key.
- Hence, **anomalies are caused by heterogeneous** informations:
  - Employee's properties (the Wage)
  - Projects' properties (the Budget)
  - Properties for the key Employee Project (the Role)

Boyce-Codd Normal Form (BCNF)

Normal forms are (useful) properties that are satisfied only in absence of anomalies, by defining constraints on functional dependencies. The most important normal form the the one named after Boyce and Codd (BCNF)

Definition:

A relation **r** is in **BCNF** if, for each functional dependency (non trivial) **X→Y** defined on **r**, **X** has a key **K** of **r**, that is, **X** is superkey for **r**

When a relation does not satisfy BCNF

- In most cases we can replace it with two or more normalized relations satisfying the BCNF. Such process is called *normalization*
- This process is based on a simple criteria:
  - If a relation represent more than one dependent concept, then it must be decomposed in smaller relations, one for each concept

Decomposition example

Employee	Wage
Jones	20
Smith	35
Williams	55
Brown	48
White	48

Project	Budget
Mars	2
Jupiter	15
Venus	15

Employee	**Project**	**Role**
Jones	Mars	Technician
Smith	Jupiter	Designer
Smith	Venus	Designer
Williams	Venus	Chief
Williams	Jupiter	Consultant
Williams	Mars	Consultant
Brown	Mars	Chief
Brown	Venus	Designer
White	Venus	Designer
White	Jupiter	Director

Another decomposition (but with loss)

Employee	Project	Office
Jones	Mars	Rome
Smith	Jupiter	Milan
Smith	Venus	Milan
White	Saturn	Milan
White	Venus	Milan

Employee → Office

Employee	Office
Jones	Rome
Smith	Milan
White	Milan

Project → Office

Project	Office
Mars	Rome
Jupiter	Milan
Venus	Milan
Saturn	Milan

We try to rebuild

Employee	Office
Jones	Rome
Smith	Milan
White	Milan

⋈

Office	Project
Rome	Mars
Milan	Jupiter
Milan	Venus
Milan	Saturn

=

Employee	Office	Project
Jones	Rome	Mars
Smith	Milan	Jupiter
Smith	Milan	Venus
Smith	**Milan**	**Saturn**
White	**Milan**	**Jupiter**
White	Milan	Saturn
White	Milan	Venus

≠

Employee	Project	Office
Jones	Mars	Rome
Smith	Jupiter	Milan
Smith	Venus	Milan
White	Saturn	Milan
White	Venus	Milan

DIFFERENT FROM THE ORIGINAL RELATION!

Decomposition: Lossless Join Property

Definition:

A relation r can be **decomposed lossless** in two relations X,Y if the join of the projection of r on X and Y is the same as r: $\boldsymbol{\pi_X(r) \bowtie \pi_Y(r) = r}$

This property is verified if the common attributes contains a key for at least one of the decomposed relations

Example: insert a tuple (1)

- Suppose that a new tuple (White,Mars,Milan) is inserted.

Employee	Project	Office
Jones	Mars	Rome
Smith	Jupiter	Milan
Smith	Venus	Milan
White	Saturn	Milan
White	Venus	Milan
White	**Mars**	**Milan**

Example: insert a tuple (2)

Employee	Project	Office
Jones	Mars	Rome
Smith	Jupiter	Milan
Smith	Venus	Milan
White	Saturn	Milan
White	Venus	Milan
White	**Mars**	**Milan**

Employee → Office

Employee	Office
Jones	Rome
Smith	Milan
White	Milan

Project → Office

Project	Office
Mars	Rome
Jupiter	Milan
Venus	Milan
Saturn	Milan
Mars	**Milan**

Dependency-preserving decompositions

We say that a decomposition **preserves the dependencies** if each functional dependency of the original schema involves attributes that appear all together in one of the decomposed schemas

Decompositions: properties

- The process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should have:
  - the lossless join property, that assures the rebuilding of the original information
  - the dependency preservation, that assures the keeping of the original integrity constraints

A relation not in normal form

Chief	Project	Office
Smith	Mars	Rome
Johnson	Jupiter	Milan
Johnson	Mars	Milan
White	Saturn	Milan
White	Venus	Milan

Project Office → Chief
Chief → Office

Decomposition has some problems

- **Project Office → Chief**
- Involves all the attributes, so no decomposition could preserve the dependency
- In some cases BCNF “cannot be reached”

Third normal form (3NF)

Definition:

A relation r is in third normal form if, for each non-trivial FD $X \rightarrow Y$ on r, at least one of the following conditions is met:

- $K \subset X$, X is key in r
- Each attribute Y is in at least one key of r

BCNF and 3NF

- BCNF is stricter than 3NF (3NF admits relations with some anomalies)
- 3NF can always be reached (there is a theorem)
- If a relation has only one key, it is in BCNF if and only if it is in 3NF

3NF decomposition

- Create a relation for each set of attributes involved in a functional dependency
- Check that in the end a relation has a key of the original relation

- It depends on the found dependencies

A possible solution

- If the relation is not normalized, decompose it to 3NF
- then, check if the resulting schema is in BCNF

A relation that cannot be put in BCNF

Chief	Project	Office
Smith	Mars	Rome
Johnson	Jupiter	Milan
Johnson	Mars	Milan
White	Saturn	Milan
White	Venus	Milan

Project Office → Chief
Chief → Office

A possible reorganization

Chief	Project	Office	Dept.
Smith	Mars	Rome	1
Johnson	Jupiter	Milan	1
Johnson	Mars	Milan	1
White	Saturn	Milan	2
White	Venus	Milan	2

Dept. Office → Chief
Chief → Office Dept.
Project Office → Dept.

Decomposition in BCNF

Chief	Office	Dept.
Smith	Rome	1
Johnson	Milan	1
White	Milan	2

Project	Office	Dept.
Mars	Rome	1
Jupiter	Milan	1
Mars	Milan	1
Saturn	Milan	2
Venus	Milan	2

A Theory for dependency

The previous concepts could be automated in an algorithmic process. That is:

- Given a relation and a set of functional dependences over it, generate a decomposition of such relation containing only relations in normal form that also satisfy the aforementioned decomposition properties (lossless and dependencies preservation)

Functional dependences, implications

From the valid functional dependencies, we can determine other dependencies, we say that the first imply the seconds:

A set of functional dependencies *F implies f* if each relation satisfying all the dependencies in *F* satisfies also *f*

Example

Employee	Type	Wage
Smith	3	30.000
Johnson	3	30.000
White	4	50.000
Williams	4	50.000
Brown	5	72.000

Employee → Type and **Type → Wage** *imply* **Employee → Wage**

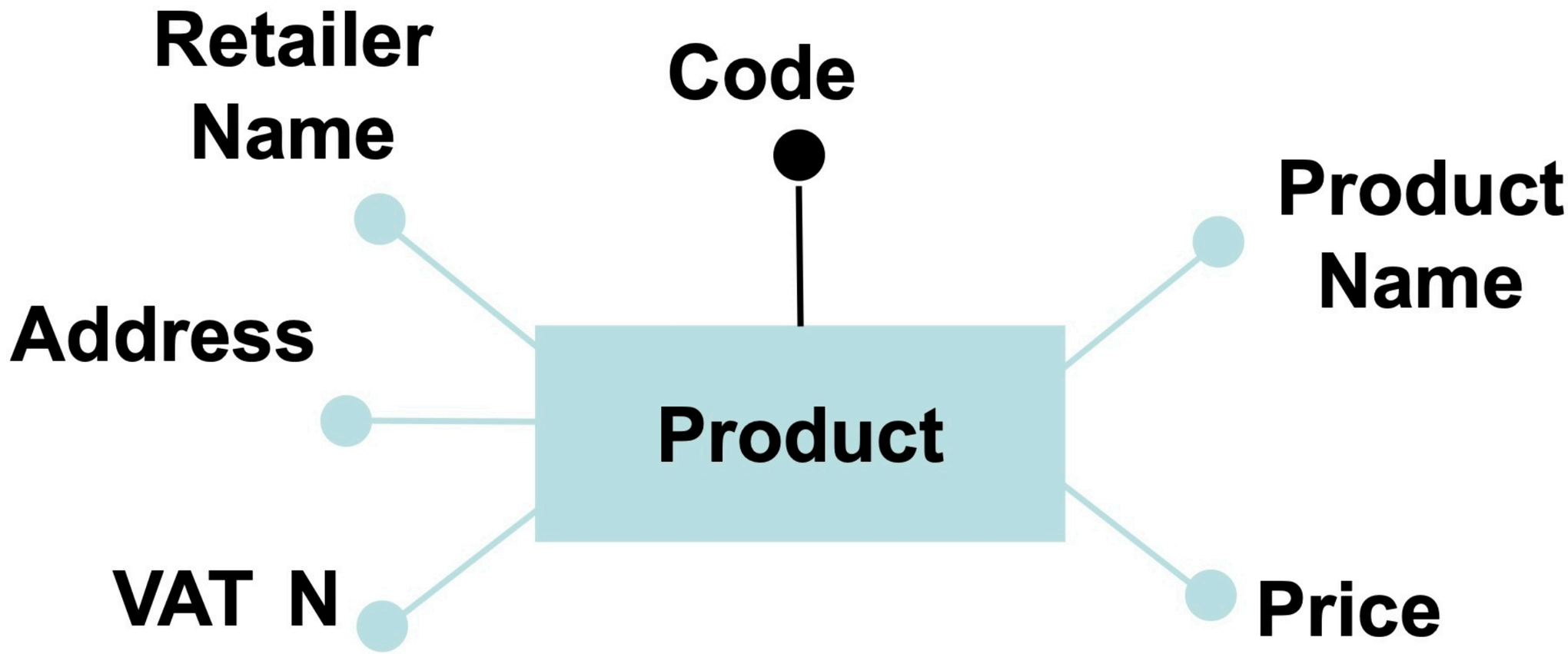
Each relation satisfying the first two dependencies, satisfies also the third one

Design and normalization

- Normalization theory can be used within the logical design to check the schema of the final relation

- It could be also used during the conceptual design phase to verify the quality of the conceptual schema

Normalization over entities



VAT N → RetailerName Address

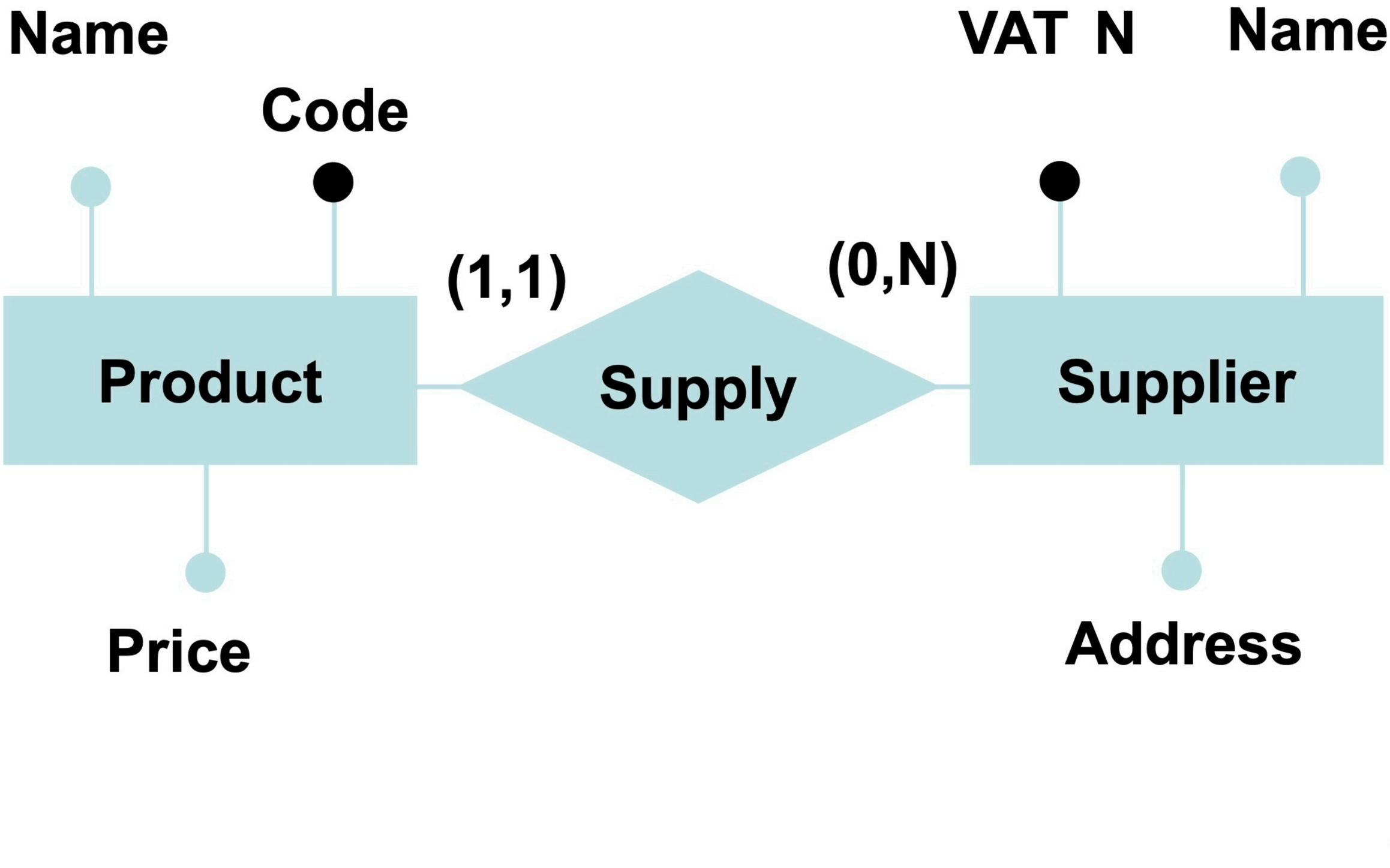
Check

- It violates the normal form due to the dependency:

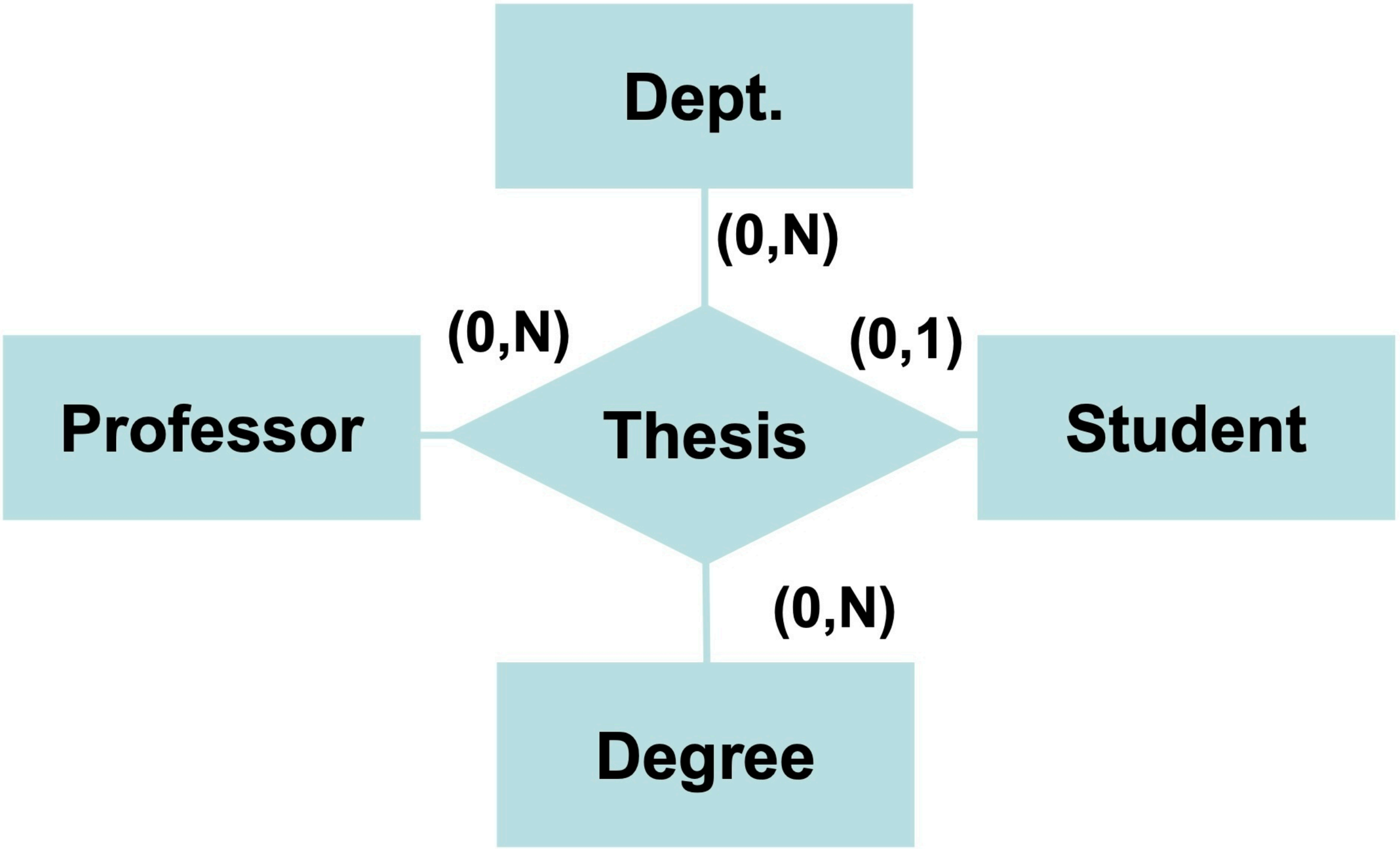
$$VAT\ N \rightarrow RetailerName\ Address$$

- We can decompose the entity using this dependency

Entity decomposition



Normalization over relationships



Student → Degree
Student → Professor
Professor → Dept.

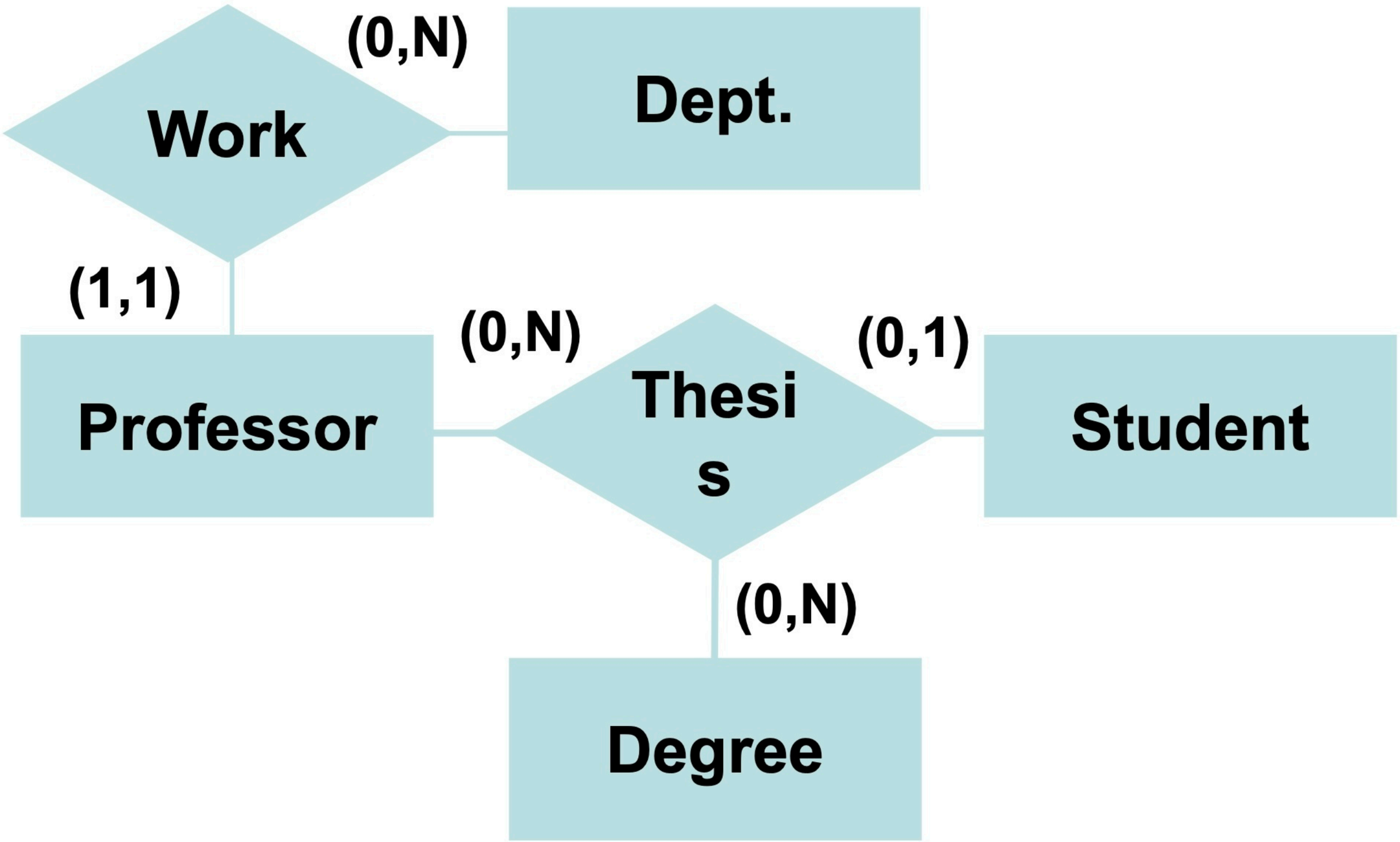
Checking the relationship

- It has no 3NF due to the following dependency:

$$\text{Professor} \rightarrow \text{Dept.}$$

- We can decompose using this dependency

Decomposing the relationship



Yet another dependency analysis

- Thesis is in BCNF based on the dependencies:

$$\text{Student} \rightarrow \text{Degree}$$

$$\text{Student} \rightarrow \text{Professor}$$

- The two properties are independent
- We can perform a further decomposition

Decomposing the relationship