

# SD202: Databases

## Functional dependencies and normal forms

---

Antoine Amarilli

Télécom Paris

Schema normalization

Functional dependencies

Boyce-Codd Normal Form

Conclusion

# Schema normalization

---

# Connection to Entity-Relationship

- We know how to **design a logical schema** via **entity-relationship diagrams...**
- ... and how to **implement it as a physical schema**
- The goal of **normalization** is to **check for remaining problems** and **fix the physical schema**
  - Intuitively, we will look for **additional constraints** in data, called **functional dependencies**
  - These dependencies mean that tables should be **subdivided further**

- The theory of functional dependencies and normal forms is **complicated** and could fill an entire class!
- We will only see **basic insights** here

## First normal form

A schema satisfies the **first normal form** if the data of every cell is an **atomic type**.  
For instance, avoid:

Student		
id	name	classes
42	John Student	SD202
43	Jane Student	SD202,INF280

→ This should already be the case at the **logical schema** level, e.g., these attributes should have been **composite attributes** or **multi-valued attributes**

Schema normalization

Functional dependencies

Boyce-Codd Normal Form

Conclusion

# Functional dependencies

---



## Definition of a functional dependency

- A **functional dependency** on a relation  $R$  is an assertion of the form  $A_1 \dots A_n \rightarrow B_1 \dots B_m$ , where the  $A_i$  and  $B_j$  are **attributes** of  $R$
- **Semantics:** for any two tuples in  $R$ , if they agree on **all** of  $A_1 \dots A_n$  then they agree on **all** of  $B_1 \dots B_m$

Student		
id	name	grade
42	John Student	14
43	Jane Student	16

- The functional dependency **id, name**  $\rightarrow$  **grade** holds

## FDs on the data vs FDs on the schema

- An FD is part of the **schema**: it is a constraint that should **always** hold  
→ “In HotelBookings, the **date** and **room** determine the **reservation\_id**”
- The FD will be satisfied on **every relation instance of the schema**
- However, a relation instance **may** satisfy some FDs “by chance”

Student		
id	name	grade
101	Jean Student	14
102	Jamie Student	14

This data satisfies **name** → **grade**, but **the schema does not!**

## FD violations

An **violation** of an FD  $A_1 \dots A_n \rightarrow B_1 \dots B_m$  is two tuples that:

- Agree on (all) the attributes  $A_1 \dots A_n$
- Disagree on (some of) the attributes  $B_1 \dots B_m$

Student		
id	name	grade
42	John Student	14
43	Jane Student	14

**Example:** This demonstrates that the FDs **grade**  $\rightarrow$  **name, id**, and the FD **grade**  $\rightarrow$  **name**, do not hold in the data, hence in the schema

## Examples and properties of FDs

- The FD  $A_1 \dots A_n \rightarrow B_1 \dots B_m$  **always holds** if  $\{B_1 \dots B_m\} \subseteq \{A_1 \dots A_n\}$ 
  - For instance,  $A \rightarrow A$ , or  $AA' \rightarrow A$ , always hold
  - FDs of this kind are called **trivial FDs**
- If attributes  $A_1 \dots A_n$  are a **key** for the relation then any FD with (at least)  $A_1 \dots A_n$  in the left-hand side will hold
- An FD  $A_1 \dots A_n \rightarrow B_1 \dots B_m$  is true iff the FDs  $A_1 \dots A_n \rightarrow B_j$  are true for each  $B_j$ 
  - It **suffices to consider** the FDs of the form  $A_1 \dots A_n \rightarrow B$
  - The general form can still be **useful as a shorter notation**

## Finding FDs

Which FDs hold, and which FDs do not hold, in that instance?

<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
1	5	10	4
2	5	11	5
2	5	10	6

## Finding FDs

Which FDs hold, and which FDs do not hold, in that instance?

<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
1	5	10	4
2	5	11	5
2	5	10	6

The FDs  $A \rightarrow B$ ,  $C \rightarrow B$ ,  $AC \rightarrow D$ , and  $D \rightarrow ABC$  hold

## Finding FDs

Which FDs hold, and which FDs do not hold, in that instance?

<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
1	5	10	4
2	5	11	5
2	5	10	6

The FDs  $A \rightarrow B$ ,  $C \rightarrow B$ ,  $AC \rightarrow D$ , and  $D \rightarrow ABC$  hold

The FDs  $B \rightarrow A$ ,  $B \rightarrow C$ ,  $A \rightarrow C$ ,  $C \rightarrow B$ ,  $A \rightarrow D$ , etc., do not hold

## FDs and anomalies

- First **find FDs**, by analysing the business needs
- Some of these FDs are “good”, e.g., the ones from the **relation key**
- Others are “bad” and illustrate a **problem in schema modeling**

<b>Student</b>			
<b>id</b>	<b>name</b>	<b>supervisor</b>	<b>supervisor_email</b>
42	John Student	Patricia Professor	pprof@telecom-paris.fr
43	Jane Student	Patricia Professor	pprof@telecom-paris.fr
44	Jean Student	Leonard Lecturer	llect@telecom-paris.fr

- Can you find the **bad FD**?



## FDs and anomalies

- First **find FDs**, by analysing the business needs
- Some of these FDs are “good”, e.g., the ones from the **relation key**
- Others are “bad” and illustrate a **problem in schema modeling**

<b>Student</b>			
<b>id</b>	<b>name</b>	<b>supervisor</b>	<b>supervisor_email</b>
42	John Student	Patricia Professor	pprof@telecom-paris.fr
43	Jane Student	Patricia Professor	pprof@telecom-paris.fr
44	Jean Student	Leonard Lecturer	llect@telecom-paris.fr

- Can you find the **bad FD**? Yes, it is **supervisor\_email** → **supervisor**

## FDs and anomalies

- First **find FDs**, by analysing the business needs
- Some of these FDs are “good”, e.g., the ones from the **relation key**
- Others are “bad” and illustrate a **problem in schema modeling**

Student			
id	name	supervisor	supervisor_email
42	John Student	Patricia Professor	pprof@telecom-paris.fr
43	Jane Student	Patricia Professor	pprof@telecom-paris.fr
44	Jean Student	Leonard Lecturer	llect@telecom-paris.fr

- Can you find the **bad FD**? Yes, it is **supervisor\_email** → **supervisor**
- Can you understand why it will lead to **insert/update/delete anomalies**?

Schema normalization

Functional dependencies

**Boyce-Codd Normal Form**

Conclusion

# Boyce-Codd Normal Form

---

# Boyce-Codd Normal Form

- A set of attributes  $A_1 \dots A_n$  is a **superkey** if it determines the entire relation, i.e., the FDs  $A_1 \dots A_n \rightarrow B$  hold for every attribute  $B$ 
  - To simplify, we assume **only one key**, then the superkeys are its supersets
- A relation is in **Boyce-Codd Normal Form** (BCNF) if for every **non-trivial** FD  $A_1 \dots A_n \rightarrow B_1 \dots B_m$  that it satisfies, then  $A_1 \dots A_n$  is a **superkey**
- BCNF disallows, for instance:
  - FDs between **non-key attributes** (attributes outside the key)
  - FDs from a **strict subset of the key attributes**

## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a...

## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a... **many-to-many relationship** between classes and students

## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a... **many-to-many relationship** between classes and students
- The **key** is...



## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a... **many-to-many relationship** between classes and students
- The **key** is... student, class

## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a... **many-to-many relationship** between classes and students
- The **key** is... student, class
- The **teacher** is...

## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a... **many-to-many relationship** between classes and students
- The **key** is... student, class
- The **teacher** is... an **attribute** of the relationship

## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a... **many-to-many relationship** between classes and students
- The **key** is... student, class
- The **teacher** is... an **attribute** of the relationship
- But: the **teacher** in fact...

## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a... **many-to-many relationship** between classes and students
- The **key** is... **student, class**
- The **teacher** is... an **attribute** of the relationship
- But: the **teacher** in fact... **only depends on the class!**

## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a... **many-to-many relationship** between classes and students
- The **key** is... student, class
- The **teacher** is... an **attribute** of the relationship
- But: the **teacher** in fact... **only depends on the class!**
- The FD **class** → **teacher** holds but...

## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a... **many-to-many relationship** between classes and students
- The **key** is... student, class
- The **teacher** is... an **attribute** of the relationship
- But: the **teacher** in fact... **only depends on the class!**
- The FD **class** → **teacher** holds but... **class** is not a superkey (it is a strict subset of the key)

## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a... **many-to-many relationship** between classes and students
- The **key** is... student, class
- The **teacher** is... an **attribute** of the relationship
- But: the **teacher** in fact... **only depends on the class!**
- The FD **class** → **teacher** holds but... **class** is not a superkey (it is a strict subset of the key)
- Hence, the relation is...



## Non-BNCF example (1)

Registration		
<u>student</u>	<u>class</u>	teacher
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

- This represents a... **many-to-many relationship** between classes and students
- The **key** is... student, class
- The **teacher** is... an **attribute** of the relationship
- But: the **teacher** in fact... **only depends on the class!**
- The FD **class** → **teacher** holds but... **class** is not a superkey (it is a strict subset of the key)
- Hence, the relation is... **not in BCNF**

## Non-BNCF example (2)

Candidates		
<u>candidate_id</u>	prepa_of_origin	city_of_origin
1	Lycée Kléber	Strasbourg
2	Louis-Le-Grand	Paris

- This table describes the prépa and city of origin of candidates to a competitive exam
- The **key** is **candidate\_id**
- The prépa and city and origin are **attributes** of the entity
- But: the prépa determines the city!
- The FD **prepa\_of\_origin** → **city\_of\_origin** holds but **prepa\_of\_origin** is not a superkey

## How to fix BCNF violations? (example)

Take the FD class → **teacher**, and find all attributes determined by class:

<u>student</u>	<u>class</u>	<b>teacher</b>
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

Make two relations:

- One with class and **with** the attributes it determines
- The other with class but **without** the attributes that it determines

## How to fix BCNF violations? (example)

Take the FD class → **teacher**, and find all attributes determined by class:

<u>student</u>	<u>class</u>	<b>teacher</b>
John Doe	SD202	Antoine Amarilli
Jane Doe	SD202	Antoine Amarilli

Make two relations:

- One with class and **with** the attributes it determines
- The other with class but **without** the attributes that it determines

<u>student</u>	<u>class</u>
John Doe	SD202
Jane Doe	SD202

<u>class</u>	<b>teacher</b>
SD202	Antoine Amarilli

## How to fix BCNF violations? (theory)

- Consider a relation  $R$  which is **not in BCNF**
- Consider a **counterexample FD** (non-trivial)  $A_1 \dots A_n \rightarrow B_1 \dots B_m$
- Find the **closure** of  $A_1 \dots A_n$ :
  - All attributes  $B$  such that  $A_1 \dots A_n \rightarrow B$  holds
  - Call this  $B'_1 \dots B'_p$ : it contains in particular  $B_1 \dots B_m$

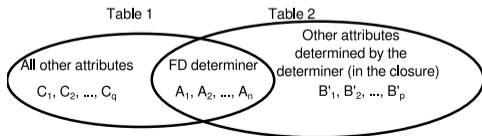
## How to fix BCNF violations? (theory)

- Consider a relation  $R$  which is **not in BCNF**
  - Consider a **counterexample FD** (non-trivial)  $A_1 \dots A_n \rightarrow B_1 \dots B_m$
  - Find the **closure** of  $A_1 \dots A_n$ :
    - All attributes  $B$  such that  $A_1 \dots A_n \rightarrow B$  holds
    - Call this  $B'_1 \dots B'_p$ : it contains in particular  $B_1 \dots B_m$
  - **Split** the attributes between:
    - The FD determiner  $A_1 \dots A_n$
    - The closure  $B'_1 \dots B'_p$  without the FD determiner  $A_1 \dots A_n$
    - The other attributes  $C_1 \dots C_q$
- Neither of these sets are **empty!** (why?)

## How to fix BCNF violations? (theory)

- Consider a relation  $R$  which is **not in BCNF**
- Consider a **counterexample FD** (non-trivial)  $A_1 \dots A_n \rightarrow B_1 \dots B_m$
- Find the **closure** of  $A_1 \dots A_n$ :
  - All attributes  $B$  such that  $A_1 \dots A_n \rightarrow B$  holds
  - Call this  $B'_1 \dots B'_p$ : it contains in particular  $B_1 \dots B_m$
- **Split** the attributes between:
  - The FD determiner  $A_1 \dots A_n$
  - The closure  $B'_1 \dots B'_p$  without the FD determiner  $A_1 \dots A_n$
  - The other attributes  $C_1 \dots C_q$

→ Neither of these sets are **empty!** (why?)
- **Build** two tables:
  - The projection on  $A_1 \dots A_n$  and  $B'_1 \dots B'_p$
  - The projection on  $A_1 \dots A_n$  and  $C_1 \dots C_q$



## Why it works?

- Splitting in two tables **reduces the redundancy**
- Fundamental property: the **join** of the two tables (on the common attributes  $A_1 \dots A_n$ ) is **equal to the original table**
  - Clearly it contains **at least the same tuples**
  - It cannot contain **more tuples** (why?)



## Why it works?

- Splitting in two tables **reduces the redundancy**
- Fundamental property: the **join** of the two tables (on the common attributes  $A_1 \dots A_n$ ) is **equal to the original table**
  - Clearly it contains **at least the same tuples**
  - It cannot contain **more tuples** (why?)
  - The first table  $A_1 \dots A_n, B'_1 \dots B'_p$  **satisfies the FD**  $A_1 \dots A_n \rightarrow B'_1 \dots B'_p$
  - The rows of the second table will join with **exactly one** row of the first table
- We say that this decomposition is a **lossless decomposition**, as opposed to a **lossy decomposition**

## How to compute the closure?

**Closure:** Given a set of attributes  $A_1 \dots A_n$ , how do we compute all attributes  $B$  such that the FD  $A_1 \dots A_n \rightarrow B$  holds?

Very simple algorithm:

- Consider all the FDs that you **know** (when defining the schema)
- Initialize a **set**  $X = \{A_1 \dots A_n\}$
- Repeatedly go over all FDs **until convergence**:
  - If an FD  $L_1 \dots L_p \rightarrow R_1 \dots R_q$  is such that  $\{L_1 \dots L_p\} \subseteq X$
  - Then add  $R_1 \dots R_q$  to  $X$
- At the end, the set  $X$  is the **closure** (why?)

## Example of closure computation

Consider the following **attributes**:

- **item**
- **power**
- **category**
- **color**
- **design\_grade**
- **functionality\_grade**
- **final\_grade**

Consider the FDs:

- **item** is a **key**
- **category**  $\rightarrow$  **color**
- **color**  $\rightarrow$  **design\_grade**
- **functionality\_grade, design\_grade**  
 $\rightarrow$  **final\_grade**

What is the closure of **category, functionality\_grade**?

Schema normalization

Functional dependencies

Boyce-Codd Normal Form

Conclusion

# Conclusion

---

- The theory of normalization is **very rich**, we only saw the basics to repair violations in a schema
- Other topics:
  - BCNF is not **dependency preserving**, i.e., sometimes some FDs of the original table are **lost** and cannot be expressed on the BCNF decomposition
  - There is an algorithm to decide which FDs are **implied** by the known FDs (Armstrong's axioms – similar to closure)
  - There are **many other** normal forms!

## Sources:

- <https://pierre.senellart.com/enseignement/2016-2017/bd/6-normalisation.pdf>
- <https://sites.google.com/site/bahrimarouaa/teaching/inf725>  
“Functional dependencies and normalization”