Introduction to Database Systems - Normalization

Radim Bača

Department of Computer Science, FEECS

radim.baca@vsb.cz

dbedu.cs.vsb.cz

Content

- 1NF
- 2NF
- 3NF
- BCNF
    - BCNF problem
- Database scheme decomposition
- Algorithm of a scheme decomposition
- Summary

Database Scheme Design

- There is a lot of ways how to design a database scheme corresponding to a particular assignment
- Some solutions are comparably good, others are considerably worse
- There exists an elegant theory for the database design

First Normal Form

A table must satisfy the following to be in a first normal form:

- The values in each attribute has to be atomic

eID	eName	eSkill
1	Cayley	C++ Python
2	Laplace	C++ Java

First Normal Form

A table must satisfy the following to be in a first normal form:

- The values in each attribute has to be atomic

eID	eName	eSkill
1	Cayley	C++ Python
2	Laplace	C++ Java

First Normal Form

A table must satisfy the following to be in a first normal form:

- The values in each attribute has to be atomic

eID	eName	eSkill
1	Cayley	C++ Python
2	Laplace	C++ Java

→

eID	eName	eSkill
1	Cayley	C++
1	Cayley	Python
2	Laplace	C++
2	Laplace	Java

First Normal Form

A table must satisfy the following to be in a first normal form:

- The values in each attribute has to be atomic

eID	eName	eSkill
1	Cayley	C++ Python
2	Laplace	C++ Java

$\rightarrow$

eID	eName	eSkill
1	Cayley	C++
1	Cayley	Python
2	Laplace	C++
2	Laplace	Java

Such table is considered to be in the **first normal form**

Second Normal Form

A table must satisfy the following to be in a second normal form:

- Table is in 1NF and
- there is no partial functional dependencies

Second Normal Form

A table must satisfy the following to be in a second normal form:

- Table is in 1NF and
- there is no partial functional dependencies

But what is a partial functional dependency?

Second Normal Form

A table must satisfy the following to be in a second normal form:

- Table is in 1NF and
- there is no partial functional dependencies

But what is a partial functional dependency?

It is closely related to the candidate keys ...

Candidate for a Key

- There can exist more keys for one table
- Candidates for a key of a scheme are all attributes $\overline{K}$ for which there is no $\overline{K'} \subset \overline{K}$ representing also a key
- In other words, we speak about shortest keys

Candidate for a Key - Example

Let us slightly extend the previous employee table

eID	eEmail	eName	eSkill
1	cayley@vsb.cz	Cayley	C++
1	cayley@vsb.cz	Cayley	Python
2	laplace@vsb.cz	Laplace	C++
2	laplace@vsb.cz	Laplace	Java

Functional dependencies:

- eID $\rightarrow$ eName eEmail
- eEmail $\rightarrow$ eID

Candidate for a Key - Example

Let us slightly extend the previous employee table

eID	eEmail	eName	eSkill
1	cayley@vsb.cz	Cayley	C++
1	cayley@vsb.cz	Cayley	Python
2	laplace@vsb.cz	Laplace	C++
2	laplace@vsb.cz	Laplace	Java

Functional dependencies:

- eID $\rightarrow$ eName eEmail
- eEmail $\rightarrow$ eID

What are the candidate keys?

Candidate for a Key - Example

Let us slightly extend the previous employee table

eID	eEmail	eName	eSkill
1	cayley@vsb.cz	Cayley	C++
1	cayley@vsb.cz	Cayley	Python
2	laplace@vsb.cz	Laplace	C++
2	laplace@vsb.cz	Laplace	Java

Functional dependencies:

- eID $\rightarrow$ eName eEmail
- eEmail $\rightarrow$ eID

$\{eID\}+ = \{eID, eName, eEmail\}$
$\{eEmail\}+ = \{eID, eName, eEmail\}$

Candidate for a Key - Example

Let us slightly extend the previous employee table

eID	eEmail	eName	eSkill
1	cayley@vsb.cz	Cayley	C++
1	cayley@vsb.cz	Cayley	Python
2	laplace@vsb.cz	Laplace	C++
2	laplace@vsb.cz	Laplace	Java

Functional dependencies:

- eID $\rightarrow$ eName eEmail
- eEmail $\rightarrow$ eID

$\{eID, eSkill\}+ = \{eID, eName, eEmail, eSkill\}$
$\{eEmail, eSkill\}+ = \{eID, eName, eEmail, eSkill\}$

Candidate for a Key - Example

Let us slightly extend the previous employee table

eID	eEmail	eName	eSkill
1	cayley@vsb.cz	Cayley	C++
1	cayley@vsb.cz	Cayley	Python
2	laplace@vsb.cz	Laplace	C++
2	laplace@vsb.cz	Laplace	Java

Functional dependencies:

- eID $\rightarrow$ eName eEmail
- eEmail $\rightarrow$ eID

$\{eID, eSkill\}+ = \{eID, eName, eEmail, eSkill\}$
$\{eEmail, eSkill\}+ = \{eID, eName, eEmail, eSkill\}$

Candidate for a Key - Example

Functional dependencies:

- eID $\rightarrow$ eName eEmail
- eEmail $\rightarrow$ eID

$\{eID, eSkill\}+ = \{eID, eName, eEmail, eSkill\}$
$\{eEmail, eSkill\}+ = \{eID, eName, eEmail, eSkill\}$

Ok, but what is a partial functional dependency?

Candidate for a Key - Example

Functional dependencies:

- eID $\rightarrow$ eName eEmail
- eEmail $\rightarrow$ eID

$\{eID, eSkill\}+ = \{eID, eName, eEmail, eSkill\}$
$\{eEmail, eSkill\}+ = \{eID, eName, eEmail, eSkill\}$

In this case the first *functional dependency is partial* because has a proper subset of a candidate key on the left side and non-key attribute on the right side.

Candidate for a Key - Example

Functional dependencies:

- eID $\rightarrow$ eName eEmail
- eEmail $\rightarrow$ eID

$\{eID, eSkill\}+ = \{eID, eName, eEmail, eSkill\}$
$\{eEmail, eSkill\}+ = \{eID, eName, eEmail, eSkill\}$

In this case the first *functional dependency is partial* because has a proper subset of a candidate key on the left side and non-key attribute on the right side.

$\downarrow$

We need to decompose the relation in order to have a relation in 2NF.

Second Normal Form

A table must satisfy the following to be in a second normal form:

- Table is in 1NF and
- there is no partial functional dependencies

mID	mName	eID	eName	eSkill
1	Newton	1	Cayley	C++
1	Newton	1	Cayley	Python
1	Newton	2	Laplace	C++
1	Newton	2	Laplace	Java

mID $\rightarrow$ mName
eID $\rightarrow$ eName mId

Second Normal Form

A table must satisfy the following to be in a second normal form:

- Table is in 1NF and
- there is no partial functional dependencies

mID	mName	eID	eName	eSkill
1	Newton	1	Cayley	C++
1	Newton	1	Cayley	Python
1	Newton	2	Laplace	C++
1	Newton	2	Laplace	Java

mID → mName
eID → eName mId

→

mID	mName	eID	eName
1	Newton	1	Cayley
1	Newton	2	Laplace

eID	eSkill
1	C++
1	Python
2	C++
2	Java

Third Normal Form

A table must satisfy the following to be in a third normal form:

- Table is in 2NF and
- No non-key attributes are transitively dependent

Third Normal Form

A table must satisfy the following to be in a third normal form:

- Table is in 2NF and
- No non-key attributes are transitively dependent

mID	mName	eID	eName
1	Newton	1	Cayley
1	Newton	2	Laplace

mID $\rightarrow$ mName
eID $\rightarrow$ eName mId

Is this relation in 3NF?

Third Normal Form

A table must satisfy the following to be in a third normal form:

- Table is in 2NF and
- No non-key attributes are transitively dependent

mID	mName	eID	eName
1	Newton	1	Cayley
1	Newton	2	Laplace

mID $\rightarrow$ mName
eID $\rightarrow$ eName mId

No, there is a transitive relationship, we need to decompose it

Third Normal Form

A table must satisfy the following to be in a third normal form:

- Table is in 2NF and
- No non-key attributes are transitively dependent

mID	mName	eID	eName
1	Newton	1	Cayley
1	Newton	2	Laplace

$\rightarrow$

mID	mName
1	Newton

eID	eName	mID
1	Cayley	1
2	Laplace	1

Boyce-Codd Normall Form

A table must satisfy the following to be in a Boyce-Codd normal form (BCNF):

- Table is in 3NF and
- every functional dependency must have a key on the left side

Sometimes it is not possible to achieve the BCNF and we are satisfied with 3NF

Boyce-Codd Normall Form

A table must satisfy the following to be in a Boyce-Codd normal form (BCNF):

- Table is in 3NF and
- every functional dependency must have a key on the left side

Sometimes it is not possible to achieve the BCNF and we are satisfied with 3NF

BCNF Problem

- A relational scheme R with a set of FDs is in BCNF if for every FD $\overline{A} \to \overline{B}$ it holds that $\overline{A}$ is a key
- However, there are cases when the BCNF condition cannot be satisfied
- *Consider* $R(J, K, L)$ *and FDs* $= \{JK \to L, L \to K\}$
  - There are two candidates for a key: JK and JL
  - R is not in BCNF and none of its decompositions will satisfy the $JK \to L$ dependency!
  - There is no decomposition which would be in BCNF and all the dependencies would be satisfied

One decomposition step

- One decomposition step decomposes an original table into two
- Every decomposition of a relational scheme $R(\overline{A})$ into two relational schemes $R_1(\overline{B})$ and $R_2(\overline{C})$ with a minimal cover F has to satisfy the following rules:
  - $\overline{B} \cup \overline{C} = \overline{A}$
  - We try to keep all functional dependencies, i.e. it is not necessary to join R_1 and R_2 to check a FD.
  - $R_1 \bowtie R_2 = R$ (lossless join)

One decomposition step

- One decomposition step decomposes an original table into two
- Every decomposition of a relational scheme $R(\overline{A})$ into two relational schemes $R_1(\overline{B})$ and $R_2(\overline{C})$ with a minimal cover F has to satisfy the following rules:
  - $\overline{B} \cup \overline{C} = \overline{A}$
  - We try to keep all functional dependencies, i.e. it is not necessary to join R_1 and R_2 to check a FD.
  - $R_1 \bowtie R_2 = R$ (lossless join)

One decomposition step

- One decomposition step decomposes an original table into two
- Every decomposition of a relational scheme $R(\overline{A})$ into two relational schemes $R_1(\overline{B})$ and $R_2(\overline{C})$ with a minimal cover F has to satisfy the following rules:
  - $\overline{B} \cup \overline{C} = \overline{A}$
  - We try to keep all functional dependencies, i.e. it is not necessary to join R_1 and R_2 to check a FD.
  - $R_1 \bowtie R_2 = R$ (lossless join)

One decomposition step

- One decomposition step decomposes an original table into two
- Every decomposition of a relational scheme $R(\overline{A})$ into two relational schemes $R_1(\overline{B})$ and $R_2(\overline{C})$ with a minimal cover F has to satisfy the following rules:
  - $\overline{B} \cup \overline{C} = \overline{A}$
  - We try to keep all functional dependencies, i.e. it is not necessary to join R_1 and R_2 to check a FD.
  - $R_1 \bowtie R_2 = R$ (lossless join)

Example - Lossless decomposition

- `Purchase(cName, email, pID, pCathegory, pLabel, when, price)`

Purchase

zName	email	pID	pCath.	pLabel	when	price
Radim	Radim.B@vsb.cz	1	hairdryer	Electrolux	1.8.2012	520
Radim	R_Moskva@a.cz	2	vacuum	Hoover	3.9.2012	3500
Li	Jet.li@email.hk	5	toothpick	WoodOva	1.10.2012	5

- We perform the scheme decomposition

Customer

zName	email
Radim	Radim.B@vsb.cz
Radim	R_Moskva@a.cz
Li	Jet.li@email.hk

ProductPurchase

email	pID	pCath.	pLabel	when	price
Radim.B@vsb.cz	1	hairdryer	Electrolux	1.8.2012	520
R_Moskva@a.cz	2	vacuum	Hoover	3.9.2012	3500
Jet.li@email.hk	5	toothpick	WoodOva	1.10.2012	5

Example - Lossless decomposition

- `Purchase(cName, email, pID, pCathegory, pLabel, when, price)`

Purchase

zName	email	pID	pCath.	pLabel	when	price
Radim	Radim.B@vsb.cz	1	hairdryer	Electrolux	1.8.2012	520
Radim	R_Moskva@a.cz	2	vacuum	Hoover	3.9.2012	3500
Li	Jet.li@email.hk	5	toothpick	WoodOva	1.10.2012	5

- We perform the scheme decomposition

Customer

zName	email
Radim	Radim.B@vsb.cz
Radim	R_Moskva@a.cz
Li	Jet.li@email.hk

ProductPurchase

email	pID	pCath.	pLabel	when	price
Radim.B@vsb.cz	1	hairdryer	Electrolux	1.8.2012	520
R_Moskva@a.cz	2	vacuum	Hoover	3.9.2012	3500
Jet.li@email.hk	5	toothpick	WoodOva	1.10.2012	5

Example - Lossless decomposition

- `Purchase(cName, email, pID, pCathegory, pLabel, when, price)`

Purchase

zName	email	pID	pCath.	pLabel	when	price
Radim	Radim.B@vsb.cz	1	hairdryer	Electrolux	1.8.2012	520
Radim	R_Moskva@a.cz	2	vacuum	Hoover	3.9.2012	3500
Li	Jet.li@email.hk	5	toothpick	WoodOva	1.10.2012	5

- We perform the scheme decomposition

Customer

zName	email
Radim	Radim.B@vsb.cz
Radim	R_Moskva@a.cz
Li	Jet.li@email.hk

ProductPurchase

email	pID	pCath.	pLabel	when	
Radim.B@vsb.cz	1	hairdryer	Electrolux	1.8.2012	520
R_Moskva@a.cz	2	vacuum	Hoover	3.9.2012	3500
Jet.li@email.hk	5	toothpick	WoodOva	1.10.2012	5

- After joining the `Customer` and `ProductPurchase` relations, we get the original relation `Purchase`

Example - Lossy decomposition

- `Purchase(cName, email, pID, pCathegory, pLabel, when, price)`

Purchase

zName	email	pID	pCath.	pLabel	when	price
Radim	Radim.B@vsb.cz	1	hairdryer	Electrolux	1.8.2012	520
Radim	R_Moskva@a.cz	2	vacuum	Hoover	3.9.2012	3500
Li	Jet.li@email.hk	5	toothpick	WoodOva	1.10.2012	5

- We perform another decomposition of the scheme

Customer

zName	email
Radim	Radim.B@vsb.cz
Radim	R_Moskva@a.cz
Li	Jet.li@email.hk

ProductPurchase

zName	pID	pCath.	pLabel	when	price
Radim	1	hairdryer	Electrolux	1.8.2012	520
Radim	2	vacuum	Hoover	3.9.2012	3500
Li	5	toothpick	WoodOva	1.10.2012	5

Example - Lossy decomposition

- `Purchase(cName, email, pID, pCathegory, pLabel, when, price)`

Purchase

zName	email	pID	pCath.	pLabel	when	price
Radim	Radim.B@vsb.cz	1	hairdryer	Electrolux	1.8.2012	520
Radim	R_Moskva@a.cz	2	vacuum	Hoover	3.9.2012	3500
Li	Jet.li@email.hk	5	toothpick	WoodOva	1.10.2012	5

- We may test a different decomposition of the scheme

Customer

zName	email
Radim	Radim.B@vsb.cz
Radim	R_Moskva@a.cz
Li	Jet.li@email.hk

ProductPurchase

zName	pID	pCath.	pLabel	when	price
Radim	1	hairdryer	Electrolux	1.8.2012	520
Radim	2	vacuum	Hoover	3.9.2012	3500
Li	5	toothpick	WoodOva	1.10.2012	5

- After joining the `Customer` and `ProductPurchase` relations, we do NOT get the original `Purchase` relation!

Lossless join

- There exists a simple criterion $R(\overline{A})$ to $R_1(\overline{B})$ and $R_2(\overline{C})$ is lossless if one of the following FD is satisfied:
  - $\overline{B} \cap \overline{C} \to \overline{B}$
  - $\overline{B} \cap \overline{C} \to \overline{C}$

Decomposition

- A correct decomposition is based on functional dependencies (FDs) above a relational scheme
- Having an FD picked, one from the resulting relational schemes has to have only this FD's attributes and common attributes of the resulting schemes have to be on the left hand side of this FD

Decomposition - Flashback Example

- *For the* `Purchase` *scheme, consider this set of FDs:*
  $email \rightarrow cName$
  $pID \rightarrow pCathegory, pLabel$
  $email, pID, when \rightarrow price$
  Why has the first decomposition been correct?
  - In the first example, the decomposition was done with respect to the rule `email` → `cName`
  - The `Customer` scheme consists of the attributes {`email`, `cName`}
  - The common attribute is `email`

Decomposition - Flashback Example

- *For the* `Purchase` *scheme, consider this set of FDs:*
  email → cName
  pID → pCathegory, pLabel
  email, pID, when → price
  Why has the first decomposition been correct?
  - In the first example, the decomposition was done with respect to the rule `email` → `cName`
  - The `Customer` scheme consists of the attributes {`email`, `cName`}
  - The common attribute is `email`

Algorithm of a Scheme Decomposition into BCNF

We find keys for an original relational scheme R;
$D = \{R\}$;
while *there exists a scheme* $R' \in D$ *not being in BCNF* **do**
  We choose $\overline{A} \to \overline{B}$ for R' not satisfying the BCNF condition;
  We decompose R' into $R_1(\overline{A}, \overline{B})$ and $R_2(\overline{A}, \textit{remaining attributes})$;
  We assign FDs to the new schemes R_1 and R_2;
  We find keys for R_1 and R_2;
  We remove R' from the D set and add R_1 and R_2 there;
end

Algorithm of a Scheme Decomposition into BCNF

We find keys for an original relational scheme R;
$D = \{R\}$;
while *there exists a scheme* $R' \in D$ *not being in BCNF* **do**
    We choose $\overline{A} \rightarrow \overline{B}$ for R' not satisfying the BCNF condition;
    We decompose R' into $R_1(\overline{A}, \overline{B})$ and $R_2(\overline{A},$ *remaining attributes*$)$;
    We assign FDs to the new schemes R_1 and R_2;
    We find keys for R_1 and R_2;
    We remove R' from the D set and add R_1 and R_2 there;
end

Algorithm of a Scheme Decomposition into BCNF

We find keys for an original relational scheme R;
$D = \{R\}$;
while *there exists a scheme* $R' \in D$ *not being in BCNF* **do**

- We choose $\overline{A} \to \overline{B}$ for R' not satisfying the BCNF condition;
- We decompose R' into $R_1(\overline{A}, \overline{B})$ and $R_2(\overline{A}$, *remaining attributes*);
- We assign FDs to the new schemes R_1 and R_2;
- We find keys for R_1 and R_2;
- We remove R' from the D set and add R_1 and R_2 there;

end

Algorithm of a Scheme Decomposition into BCNF

We find keys for an original relational scheme R;
$D = \{R\}$;
while *there exists a scheme* $R' \in D$ *not being in BCNF* **do**
  We choose $\overline{A} \rightarrow \overline{B}$ for R' not satisfying the BCNF condition;
  We decompose R' into $R_1(\overline{A}, \overline{B})$ and $R_2(\overline{A},$ *remaining attributes*);
  We assign FDs to the new schemes R_1 and R_2;
  We find keys for R_1 and R_2;
  We remove R' from the D set and add R_1 and R_2 there;
end

Algorithm of a Scheme Decomposition into BCNF

We find keys for an original relational scheme R;
$D = \{R\}$;
while *there exists a scheme* $R' \in D$ *not being in BCNF* **do**
  We choose $\overline{A} \rightarrow \overline{B}$ for R' not satisfying the BCNF condition;
  We decompose R' into $R_1(\overline{A}, \overline{B})$ and $R_2(\overline{A}$, *remaining attributes*);
  We assign FDs to the new schemes R_1 and R_2;
  We find keys for R_1 and R_2;
  We remove R' from the D set and add R_1 and R_2 there;
end

Algorithm of a Scheme Decomposition into BCNF

We find keys for an original relational scheme R;
$D = \{R\}$;
while *there exists a scheme* $R' \in D$ *not being in BCNF* **do**
    We choose $\overline{A} \to \overline{B}$ for R' not satisfying the BCNF condition;
    We decompose R' into $R_1(\overline{A}, \overline{B})$ and $R_2(\overline{A}$, *remaining attributes*);
    We assign FDs to the new schemes R_1 and R_2;
    We find keys for R_1 and R_2;
    We remove R' from the D set and add R_1 and R_2 there;
end

Decomposition into BCNF - Example

- $\mathtt{Purchase(cName,\ email,\ pID,\ pCathegory,\ pLabel,\ when,\ price)}$

  $\{\mathtt{email} \rightarrow \mathtt{cName}; \mathtt{pID} \rightarrow \mathtt{pCathegory}, \mathtt{pLabel}; \mathtt{email}, \mathtt{pID}, \mathtt{when} \rightarrow \mathtt{price}\}$
- $\{\mathtt{email}, \mathtt{pID}, \mathtt{when}\}$ is the key

Decomposition into BCNF - Example

- *Purchase(cName, email, pID, pCathegory, pLabel, when, price)*

  $\{email \rightarrow cName; pID \rightarrow pCathegory, pLabel; email, pID, when \rightarrow price\}$
- {email, pID, when} is the key

Decomposition into BCNF - Example

- `Purchase(cName, email, pID, pCathegory, pLabel, when, price)`

  $\{email \rightarrow cName; pID \rightarrow pCathegory, pLabel; email, pID, when \rightarrow price\}$
- {`email`, `pID`, `when`} is the key
- There are two FDs violationg the BCNF condition;

  let us choose, e.g., `email` → `cName`
- We decompose the original `Purchase` scheme into two:
  - `Customer(email, cName)`
  - `Purchase(email, pID, pCathegory, pLabel, when, price)`

Decomposition into BCNF - Example

- Purchase(cName, email, pID, pCathegory, pLabel, when, price)

  $\{email \rightarrow cName; pID \rightarrow pCathegory, pLabel; email, pID, when \rightarrow price\}$
- {email, pID, when} is the key
- There are two FDs violationg the BCNF condition;

  let us choose, e.g., email → cName
- We decompose the original Purchase scheme into two:
  - Customer(email, cName)
  - Purchase(email, pID, pCathegory, pLabel, when, price)

Decomposition into BCNF - Example

- We decompose the original `Purchase` scheme into two:
  - `Customer(email, cName)`
  - `Purchase(email, pID, pCathegory, pLabel, when, price)`
- The attributes {`email`, `pID`, `when`} are still the key of the `Purchase` scheme

Decomposition into BCNF - Example

- We decompose the original `Purchase` scheme into two:
  - `Customer(email, cName)`
  - `Purchase(email, pID, pCathegory, pLabel, when, price)`
- The attributes {`email`, `pID`, `when`} are still the key of the `Purchase` scheme
- `pID` → `pCathegory`, `pLabel` violates the BCNF condition
- We docompose the new `Purchase` scheme:
  - `Product(pID, pCathegory, pLabel)`
  - `Purchase(email, pID, when, price)`

Decomposition into BCNF - Example

- We decompose the original `Purchase` scheme into two:
  - `Customer(email, cName)`
  - `Purchase(email, pID, pCathegory, pLabel, when, price)`
- The attributes $\{$`email`, `pID`, `when`$\}$ are still the key of the `Purchase` scheme
- `pID` $\rightarrow$ `pCathegory`, `pLabel` violates the BCNF condition
- We docompose the new `Purchase` scheme:
  - `Product(pID, pCathegory, pLabel)`
  - `Purchase(email, pID, when, price)`

Decomposition into BCNF - Example

- We decompose the original `Purchase` scheme into two:
  - `Customer(email, cName)`
  - ~~`Purchase(email, pID, pCathegory, pLabel, when, price)`~~
- The attributes $\{\texttt{email}, \texttt{pID}, \texttt{when}\}$ are still the key of the `Purchase` scheme
- $\texttt{pID} \rightarrow \texttt{pCathegory}, \texttt{pLabel}$ violates the BCNF condition
- We docompose the new `Purchase` scheme:
  - `Product(pID, pCathegory, pLabel)`
  - `Purchase(email, pID, when, price)`

Decomposition into BCNF - Properties

- Please recall that there are two requirements:
  - The decomposition satisfies the requirement on lossless join (i.e., $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n = R$)
  - All functional dependecies has to be satisfied
- The first requirement is always satisfied if we use the decomposition to BCNF
- The resulting scheme produced by the algorithm is dependent on an order in which we pick FDs
- We start with FD where attributes on the right side never occur on the left side in any other FD
- The problem is that sometimes we can not perform a decomposition to BCNF and keep all FDs

Decomposition into BCNF - Properties

- Please recall that there are two requirements:
  - The decomposition satisfies the requirement on lossless join (i.e., $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n = R$)
  - All functional dependecies has to be satisfied
- The first requirement is always satisfied if we use the decomposition to BCNF
- The resulting scheme produced by the algorithm is dependent on an order in which we pick FDs
- We start with FD where attributes on the right side never occur on the left side in any other FD
- The problem is that sometimes we can not perform a decomposition to BCNF and keep all FDs

Decomposition into BCNF - Properties

- Please recall that there are two requirements:
  - The decomposition satisfies the requirement on lossless join (i.e., $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n = R$)
  - All functional dependecies has to be satisfied
- The first requirement is always satisfied if we use the decomposition to BCNF
- The resulting scheme produced by the algorithm is dependent on an order in which we pick FDs
- We start with FD where attributes on the right side never occur on the left side in any other FD
- The problem is that sometimes we can not perform a decomposition to BCNF and keep all FDs

Decomposition into BCNF - Properties

- Please recall that there are two requirements:
  - The decomposition satisfies the requirement on lossless join (i.e., $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n = R$)
  - All functional dependecies has to be satisfied
- The first requirement is always satisfied if we use the decomposition to BCNF
- The resulting scheme produced by the algorithm is dependent on an order in which we pick FDs
- We start with FD where attributes on the right side never occur on the left side in any other FD
- The problem is that sometimes we can not perform a decomposition to BCNF and keep all FDs

Decomposition into BCNF - Properties

- Please recall that there are two requirements:
  - The decomposition satisfies the requirement on lossless join (i.e., $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n = R$)
  - All functional dependecies has to be satisfied
- The first requirement is always satisfied if we use the decomposition to BCNF
- The resulting scheme produced by the algorithm is dependent on an order in which we pick FDs
- We start with FD where attributes on the right side never occur on the left side in any other FD
- The problem is that sometimes we can not perform a decomposition to BCNF and keep all FDs

Summary

- Functional dependencies (FDs)
    - represent a relationship among attributes, which can cause redundancy in data
    - By using FDs, we find a closure for some attributes and also a key of a scheme
- Optimization of a set of FDs
- 1NF, 2NF, 3NF, BCNF
- Relational scheme decomposition
- BCNF:
    - form of relational scheme that is lossless
    - decomposition into BCNF with respect to FDs

Summary

- Functional dependencies (FDs)
  - represent a relationship among attributes, which can cause redundancy in data
  - By using FDs, we find a closure for some attributes and also a key of a scheme
- Optimization of a set of FDs
- 1NF, 2NF, 3NF, BCNF
- Relational scheme decomposition
- BCNF:
  - form of relational scheme that is lossless
  - decomposition into BCNF with respect to FDs

Summary

- Functional dependencies (FDs)
  - represent a relationship among attributes, which can cause redundancy in data
  - By using FDs, we find a closure for some attributes and also a key of a scheme
- Optimization of a set of FDs
- 1NF, 2NF, 3NF, BCNF
- Relational scheme decomposition
- BCNF:
  - form of relational scheme that is lossless
  - decomposition into BCNF with respect to FDs

Summary

- Functional dependencies (FDs)
  - represent a relationship among attributes, which can cause redundancy in data
  - By using FDs, we find a closure for some attributes and also a key of a scheme
- Optimization of a set of FDs
- 1NF, 2NF, 3NF, BCNF
- Relational scheme decomposition
- BCNF:
  - form of relational scheme that is lossless
  - decomposition into BCNF with respect to FDs

Summary

- Functional dependencies (FDs)
  - represent a relationship among attributes, which can cause redundancy in data
  - By using FDs, we find a closure for some attributes and also a key of a scheme
- Optimization of a set of FDs
- 1NF, 2NF, 3NF, BCNF
- Relational scheme decomposition
- BCNF:
  - form of relational scheme that is lossless
  - decomposition into BCNF with respect to FDs

Summary

- Functional dependencies (FDs)
  - represent a relationship among attributes, which can cause redundancy in data
  - By using FDs, we find a closure for some attributes and also a key of a scheme
- Optimization of a set of FDs
- 1NF, 2NF, 3NF, BCNF
- Relational scheme decomposition
- BCNF:
  - form of relational scheme that is lossless
  - decomposition into BCNF with respect to FDs

References

- Jennifer Widom. Introduction to Databases. https://www.coursera.org/course/db
- Course home pages http://dbedu.cs.vsb.cz