

Chapter 4 – Relational database design

Databases lectures

Prof. Dr Kai Höfig



Quo Vadis?

TH Organization:

MatrNr	SBirthday	SName	SGrade	Course	Semester	PName	POffice
1	14.05.2001	Max	1.0	DB	WS23	HoKa	B1.18
2	17.04.2000	Miri	2.3	DB	WS23	HoKa	B1.18
2	17.04.2000	Miri	1.3	OOP	SS23	LeGr	B1.12
3	18.09.2002	Sarah	4.0	OOP	SS22	LeGr	B1.12
4	07.06.2002	Ben	3.3	DB	WS23	HoKa	B1.18

Flat-file database.
Redundancies cause
anomalies. Very bad!

Intuitive
optimizations



Sweet Spot using **Normalization**

MatrNr:	MatrNrID	MatrNr									
	1	1	Name:	PNameID	PName	Birthday:	SBirthdayID	SBirthday			
	2	2							1	HoKa	14.05.2001
	3	3							2	LeGr	17.04.2000
	4	4							3	18.09.2002	
			4			07.06.2002					
			Name:	SNameID	SName						
		1	Max	rel_MatrNr_Sbirthday:	MatrNrID	SBirthdayID					
		2	Mir				1	1			
		3	Sari				2	2			
		4	Ben				3	3			
				4			4				
rel_PName_Poffice:	PNameID	PofficeID									
	1	1									
	2	2									

Every attribute with separate
key, relationships over IDs.
Grows exponentially. Low
performance. Very bad!

Intuitive
optimizations

After this chapter, you will be able to decide whether a given relation is a good relation and to make necessary optimizations.



Anomalies

TH Organization:	MatrNr	SBirthday	SName	SGrade	Course	Semester	PName	POffice
	1	14.05.2001	Max	1.0	DB	WS23	HoKa	B1.18
	2	17.04.2000	Miri	2.3	DB	WS23	HoKa	B1.18
	2	17.04.2000	Miri	1.3	OOP	SS23	LeGr	B1.12
	3	18.09.2002	Sarah	4.0	OOP	SS22	LeGr	B1.12
	4	07.06.2002	Ben	3.3	DB	WS23	HoKa	B1.18

♦ Update or insertion anomaly

Identities are stored redundantly and on updates, not all redundancies are updated, or the inserted data contradicts the existing data.

E.g. changing the name of the lecture DB to Databases only in the first tuple. Now it appears that DB and Databases are two different lectures. Or, the new tuple inserts db instead of DB.

♦ Deletion anomaly

Deleting a tuple deletes more data than it was intended to be deleted.

E.g. when deleting the whole tuples of Miri and Sarah then LeGr and the office B1.12 are not present anymore.



Redundancies cause anomalies, that destroy data over time and consume additional memory.

We need to **remove them** using a good data model **preserving performance** and get to the sweet spot.



Normalization

1. Identify functional dependencies

Functional dependencies are special relationships between attributes. Whether there is a functional between some attributes is domain knowledge. We need to ask the client, to collect as many functional dependencies as possible.

2. Calculate the key

If we did collect enough functional dependencies, all possible candidate keys can be calculated (One of them can then be selected as the primary key).

3. Check whether those attributes together already fulfill the requirements of a normal form, preferably BCNF

By analyzing the functional dependencies, it can be decided whether putting all attributes in one relation already is a good idea in terms of freedom of anomalies and redundancies.

4. If no: Normalize

We again use the functional dependencies to create relationships that are free of anomalies.





What is a functional dependency?

TH Organization:	MatrNr	SBirthday	SName	SGrade	Course	Semester	PName	POffice
	1	14.05.2001	Max	1.0	DB	WS23	HoKa	B1.18
	2	17.04.2000	Miri	2.3	DB	WS23	HoKa	B1.18
	2	17.04.2000	Miri	1.3	OOP	SS23	LeGr	B1.12
	3	18.09.2002	Sarah	4.0	OOP	SS22	LeGr	B1.12
	4	07.06.2002	Ben	3.3	DB	WS23	HoKa	B1.18

- ◆ Best to understand by example

The name of a student is functionally dependent on its matriculation number.

or

The matriculation number determines a student's name.

- ◆ We write $MatrNr \rightarrow SName$
- ◆ Meaning: If there are two tuples in the database with the same matriculation number, they cannot have different names. (The other way around it is possible!)



Functional dependencies formally

- ◆ If the values for attributes A_1, \dots, A_n are the same for two different tuples t_1, t_2 and the values for attributes B_1, \dots, B_m must be same in all instances of the database, then B_1, \dots, B_m are functionally dependent on A_1, \dots, A_n .

$$A_1, \dots, A_n \rightarrow B_1, \dots, B_m \quad := \quad \forall t_i, t_j \in r, i \neq j, \\ \{A_1, \dots, A_n, B_1, \dots, B_m\} \subseteq R : \\ t_i(A_1, \dots, A_n) = t_j(A_1, \dots, A_n) \Rightarrow t_i(B_1, \dots, B_m) = t_j(B_1, \dots, B_m)$$

Or shorter

$$A \rightarrow B := \forall t_i, t_j \in r, i \neq j, A, B \subseteq R : t_i(A) = t_j(A) \Rightarrow t_i(B) = t_j(B)$$

- ◆ Functional dependencies (FDs) can only be determined from the context of the respective database application. They cannot automatically be retrieved from data but must come from a domain expert.

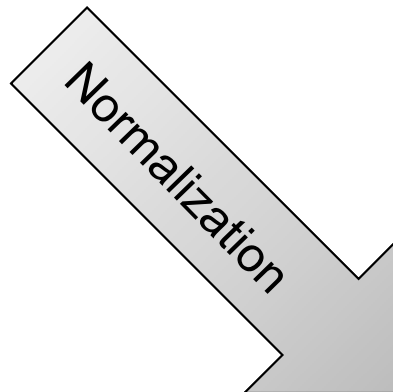




So, we know how it should look like...

TH Organization:

<u>MatrNr</u>	<u>SBirthday</u>	<u>SName</u>	<u>SGrade</u>	<u>Course</u>	<u>Semester</u>	<u>PName</u>	<u>POffice</u>
1	14.05.2001	Max	1.0	DB	WS23	HoKa	B1.18
2	17.04.2000	Miri	2.3	DB	WS23	HoKa	B1.18
2	17.04.2000	Miri	1.3	OOP	SS23	LeGr	B1.12
3	18.09.2002	Sarah	4.0	OOP	SS22	LeGr	B1.12
4	07.06.2002	Ben	3.3	DB	WS23	HoKa	B1.18



Students:

<u>MatrNr</u>	<u>SName</u>	<u>SBirthday</u>
1	Max	14.05.2001
2	Miri	17.04.2000
3	Sarah	18.09.2002
4	Ben	07.06.2002

Profs:

<u>PName</u>	<u>POffice</u>
HoKa	B1.18
LeGr	B1.12

Grades:

<u>MatrNr</u>	<u>S_Grade</u>	<u>Course</u>	<u>Semester</u>
1	1.0	DB	WS23
2	2.3	DB	WS23
2	1.3	OOP	SS23
3	4.0	OOP	SS22
4	3.3	DB	WS23

Couses:

<u>Name</u>	<u>Semester</u>	<u>PName</u>
DB	WS23	HoKa
OOP	SS22	LeGr
DB	WS23	HoKa

But why? What has
that to do with FDs?



Superkey

- ◆ When there is a relationship with attributes

$$R = \{A_1, \dots, A_n\}$$

- ◆ Then a subset of attributes

$$A = \{A_i, \dots, A_j\} \subseteq R$$

- ◆ Is called a **Superkey** if

$$A \rightarrow R$$

meaning the attribute determines the whole relation

- ◆ Examples

$$\{MatrNr\}$$

$$\{MatrNr, SName\}$$

$$\{MatrNr, SName, SBirthDay\}$$

because

$$MatrNr \rightarrow Students$$

$$MatrNr, SName \rightarrow Students$$

$$MatrNr, SName, SBirthDay \rightarrow Students$$

Students: MatrNr SName SBirthDay

1	Max	14.05.2001
2	Miri	17.04.2000
3	Sarah	18.09.2002
4	Ben	07.06.2002



Key or candidate key

- ◆ A minimal subset of a superkey that still fulfills the property of being a superkey is called a **key** or **candidate key**.

Students: MatrNr SName SBirthday

1	Max	14.05.2001
2	Miri	17.04.2000
3	Sarah	18.09.2002
4	Ben	07.06.2002

$\{MatrNr\}$

~~$\{MatrNr, SName\}$~~

~~$\{MatrNr, SName, SBirthday\}$~~

The only key, the only
prime attribute, the
primary key

there is no subset of K that does not provide the whole relation

- ◆ Formally, K is a key of R if $K \rightarrow R$, $K \supset K' \not\rightarrow R$
- ◆ An attribute that is part of a key, is called a **prime attribute**.
- ◆ A **primary key** is one of the candidate keys that is selected during the database design as primary key.



Foreign key

- ◆ When there are two relations R and S , then a subset of R fulfills the **foreign key constraint** for a subset of S when

$$\{A_1, \dots, A_n\} \subseteq R \text{ and } \{B_1, \dots, B_n\} \subseteq S$$

e.g.: Events has lecturer which is a foreign key of the name attribute in Lecturers, meaning the first is a subset of the latter

$$\{t(A_1, \dots, A_n) | t \in R\} \subseteq \{t(B_1, \dots, B_n) | t \in S\}$$

- ◆ If that property is fulfilled in all instances of the database, this subset is called a **foreign key**.

Couses: <u>Name Semester PName</u>			
DB	WS23	HoKa	
OOP	SS22	LeGr	
DB	WS23	HoKa	

$\{Course, Semester\}$ in *Grades* fulfills the foreign key constraint for $\{Name, Semester\}$ in *Courses*

Grades: <u>MatrNr S_Grade Course Semester</u>			
1	1.0	DB	WS23
2	2.3	DB	WS23
2	1.3	OOP	SS23
3	4.0	OOP	SS22
4	3.3	DB	WS23

$\{Course, Semester\}$ in *Grades* is a foreign key.



Exercise: Paniccis Pizza

- ◆ Name Functional Dependencies, Superkeys and Candidatekeys, Prime Attributes and a Key

No.	Name	Vegetarian	Vegan	Meat	Price
1	Pizza Peperoni	false	false	true	8,99
2	Pizza 4 Cheese	true	false	false	8,99
3	Spaghetti Carbonara	false	false	true	9,99
4	Cocacola	true	true	false	3,99
5	Salat	true	true	false	6,99
6	Caprese	true	false	false	7,99
7	Salat Ham	false	false	true	7,99
8	Tiramisu	true	false	false	4,99
9	Icecream	true	false	false	2,99



How does that help?

- ◆ The process of normalization has got something to do with the keys.
- ◆ How can we find the keys?
- ◆ Easy: we take all subsets of attributes and check them for superkey property.
- ◆ Once we found all superkeys, we minimize them to keys.





Answer: not so much

- ◆ Checking each subset..*

- ◆ For being a key..



2	2
3	5
7	877
13	27644437
42	35742549198872617291353508656626642567
55	359334085968622831041960188598043661065388726959079837



- ◆ We need a way to get some FDs from the domain expert and then generate new ones!

* The number of all subsets of an n-element set is referred to as the *Bell number*.



Properties of FDs: Triviality

- ◆ Functional dependencies contain trivialities

$$\begin{aligned} & \{X\} \subseteq R \\ \Rightarrow & X \rightarrow X \in FD_R \end{aligned}$$

- ◆ Proof:

$$\begin{aligned} & t_1(X) = t_2(X), t_i \in r(R) \\ \Rightarrow & t_1(X) = t_2(X) \\ \Rightarrow & X \rightarrow X \in FD_R \end{aligned}$$

R:	X	Y	Z
	1	1	1
	2	1	1
	3	2	1
	4	3	2

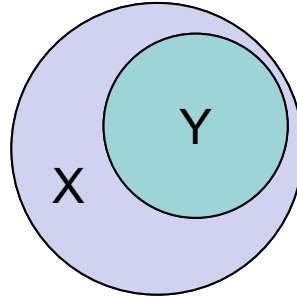
Every Attribute determines itself



Properties of FDs: Reflexivity

- ◆ Functional dependencies are reflexive

$$Y \subseteq X \subseteq R$$
$$\Rightarrow X \rightarrow Y \in FD_R$$



R:	X	Y	Z
	1	1	1
	2	1	1
	3	2	1
	4	3	2

- ◆ Proof:

$$t_1(X) = t_2(X) \text{ for } t_i \in r(R)$$
$$\wedge Y \subseteq X \subseteq R$$
$$\Rightarrow t_1(Y) = t_2(Y)$$
$$\Rightarrow X \rightarrow X \in FD_R$$

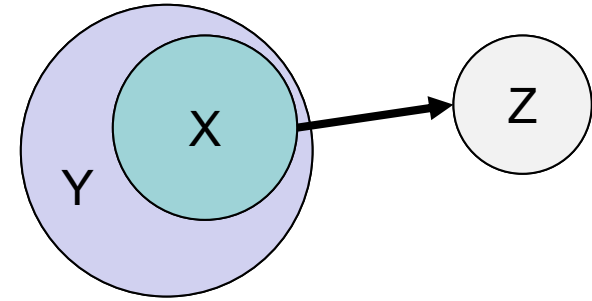
$$\{A\} \subset \{A, B\} \subset R = \{A, B, C\}$$
$$\Rightarrow A, B \rightarrow A$$

Every set of attributes determines its subset



Properties of FDs: Augmentation

$$\begin{aligned} X \rightarrow Z \in FD_R &\Rightarrow X, Y \rightarrow Z \in FD_R \\ &\Rightarrow X, Y \rightarrow Z, Y \in FD_R \end{aligned}$$



◆ Proof:

- Assumption:
$$X \rightarrow Z \in FD_R \quad (1)$$
$$X, Y \rightarrow Z, Y \notin FD_R \quad (2)$$

- Contradiction:

$$\begin{aligned} &t_1, t_2 \in r(R) \\ t_1(X) = t_2(X) &\stackrel{(1)}{\implies} t_1(Z) = t_2(Z) \\ t_1(X, Y) = t_2(X, Y) &\Rightarrow t_1(Y) = t_2(Y) \\ t_1(X, Y) = t_2(X, Y) &\stackrel{(2)}{\implies} t_1(Z, Y) \neq t_2(Z, Y) \end{aligned}$$

Every left side can be enlarged



Properties of FDs: Decomposition

- ◆ Functional dependencies can be decomposed

$$\begin{aligned} & \{X, Y, Z\} \subseteq R \\ \wedge & \quad X \rightarrow Y, Z \subseteq FD_R \\ \Rightarrow & \quad X \rightarrow Y \in FD_R \end{aligned}$$

- ◆ Proof:

- Assumption: $\{X, Y, Z\} \subseteq R \wedge X \rightarrow Y, Z \subseteq FD_R$

$$\begin{aligned} & t_1, t_2 \in FD_R \\ t_1(X) = t_2(X) & \Rightarrow t_1(Y, Z) = t_2(Y, Z) \\ & \Rightarrow t_1(Y) = t_2(Y) \\ & \Rightarrow X \rightarrow Y \in FD_R \end{aligned}$$

Right sides can be splitted



Properties of FDs: Union

- ◆ Functional dependencies can be combined, if the left side is the same

$$\begin{aligned} & \{X, Y, Z\} \subseteq R \\ \wedge & \{X \rightarrow Y, X \rightarrow Z\} \subseteq FD_R \\ \Rightarrow & X \rightarrow Y, Z \in FD_R \end{aligned}$$

- ◆ Proof:

- Assumption: $\{X, Y, Z\} \subseteq R \wedge \{X \rightarrow Y, X \rightarrow Z\} \subseteq FD_R$

$$\begin{aligned} & t_1, t_2 \in FD_R \\ t_1(X) = t_2(X) & \Rightarrow t_1(Y) = t_2(Y) \\ & \Rightarrow t_1(Z) = t_2(Z) \\ & \Rightarrow t_1(Y, Z) = t_2(Y, Z) \\ & \Rightarrow X \rightarrow Y, Z \in FD_R \end{aligned}$$

Right sides can be combined



Properties of FDs: Transitivity

- ◆ Functional dependencies are transitive

$$\begin{aligned} & \{X, Y, Z\} \subseteq R \\ \wedge & \{X \rightarrow Y, Y \rightarrow Z\} \subseteq FD_R \\ \Rightarrow & X \rightarrow Z \in FD_R \end{aligned}$$

- ◆ Proof:

- Assumption: $\{X \rightarrow Y, Y \rightarrow Z\} \subseteq FD_R$
- If $t_1(X) = t_2(X)$ for $t_i \in r(R) \Rightarrow t_1(Y) = t_2(Y)$

because

$$A \rightarrow B := \forall t_i, t_j \in r, i \neq j, A, B \subseteq R : t_i(A) = t_j(A) \Rightarrow t_i(B) = t_j(B)$$

- Therefore

$$t_1(Z) = t_2(Z) \Rightarrow X \rightarrow Z \in FD_R$$

FDs can be extended by following links



Properties of FDs: Pseudo-transitivity

- ◆ Functional dependencies are pseudo-transitive

$$\begin{aligned} & \{W, X, Y, Z\} \subseteq R \\ \wedge \quad & \{X \rightarrow Y, W, Y \rightarrow Z\} \subseteq FD_R \\ \Rightarrow \quad & W, X \rightarrow Z \in FD_R \end{aligned}$$

- ◆ Proof:

- Assumption: $\{X \rightarrow Y, W, Y \rightarrow Z\} \subseteq FD_R$

$$\begin{array}{ccc} X \rightarrow Y \in FD_R & \xRightarrow{\text{Augmentation}} & W, X \rightarrow W, Y \in FD_R \\ & \xRightarrow{\text{Transitivity}} & W, X \rightarrow Z \in FD_R \end{array}$$

FDs can be extended by following links



Summary of FD properties

		$X, Y, Z \subseteq R$
	Triviality \implies	$X \rightarrow X \in FD_R$
$Y \subseteq X \subseteq R$	Reflexivity \implies	$X \rightarrow Y \in FD_R$
$X \rightarrow Z \in FD_R$	Augmentation \implies	$X, Y \rightarrow Z \in FD_R$
	Augmentation \implies	$X, Y \rightarrow Z, Y \in FD_R$
$X \rightarrow Y, Z \subseteq FD_R$	Decomposition \implies	$X \rightarrow Y \in FD_R$
$\{X \rightarrow Y, X \rightarrow Z\} \subseteq FD_R$	Union \implies	$X \rightarrow Y, Z \in FD_R$
$\{X \rightarrow Y, Y \rightarrow Z\} \subseteq FD_R$	Transitivity \implies	$X \rightarrow Z \in FD_R$
$\{X \rightarrow Y, W, Y \rightarrow Z\} \subseteq FD_R$	Pseudo-transitivity \implies	$W, X \rightarrow Z \in FD_R$



Splitting Algorithm

- ◆ A very simple algorithm to simplify a set of functional dependencies

$MatrNr \rightarrow SBirthday$

$MatrNr \rightarrow SBirthday, SName$

$MatrNr, Course, Semester, PName \rightarrow MatrNr, SGrade, Couser, Semester$

1. We split all FDs with more than one attribute on the right side using decomposition.

$MatrNr \rightarrow SBirthday$

~~$MatrNr \rightarrow SBirthday$~~

$MatrNr \rightarrow SName$

~~$MatrNr, Course, Semester, PName \rightarrow MatrNr$~~

$MatrNr, Course, Semester, PName \rightarrow SGrade$

~~$MatrNr, Course, Semester, PName \rightarrow Course$~~

~~$MatrNr, Course, Semester, PName \rightarrow Semester$~~

2. We remove all trivial FDs and double FDs

3. We summarize again using the union property

$MatrNr \rightarrow SBirthday, SName$

$MatrNr, Course, Semester, PName \rightarrow SGrade$





Closure of FDs

- ◆ The closure FD^+ of a set of functional dependencies FD is used to refer to the set of all functional dependencies that can be derived from an existing set FD .
- ◆ The closure expresses the same as the set of FDs itself but contains all FDs that can be derived using the properties.

$$\begin{aligned} Splitting(FD_R) &\subseteq FD_R \subseteq FD_R^+ \\ Splitting(FD_R) &\equiv FD_R \equiv FD_R^+ \end{aligned}$$

Equivalence means that it expresses the same. It is $\{A \rightarrow B\} \equiv \{A \rightarrow B, A \rightarrow A\}$

$$FD_R^+ := \{A \rightarrow B \mid A, B \subseteq R, FD_R \Rightarrow A \rightarrow B\}$$

- ◆ This would help us in finding all the keys by reducing all super-keys, but generating this set is not feasible.



Closure of attributes

- ◆ The closure of attributes regarding functional dependencies FD is defined as

$$\{X_1, \dots, X_n\}_{FD}^+ := \{A \mid \{X_1, \dots, X_n\} \rightarrow A \in FD^+\}$$

- ◆ So in words, A^+ is the set of attributes that are functionally dependent on A.
- ◆ With the closure-set of attributes, the **membership problem** can be solved:

Is $X_1, \dots, X_n \rightarrow A \in FD$?

1. Calculate $\{X_1, \dots, X_n\}_{FD}^+$

2. If $A \in \{X_1, \dots, X_n\}_{FD}^+ \Rightarrow \{X_1, \dots, X_n\} \rightarrow A \in FD^+$

- ◆ With the closure-set of attributes, we can check for keys:

X is key of R if $\{X_1, \dots, X_n\} \rightarrow R \in FD$ and X cannot be shortened.



CLOSURE Algorithm

♦ Input: $FD_R, X = \{X_1, \dots, X_n\} \subseteq R$

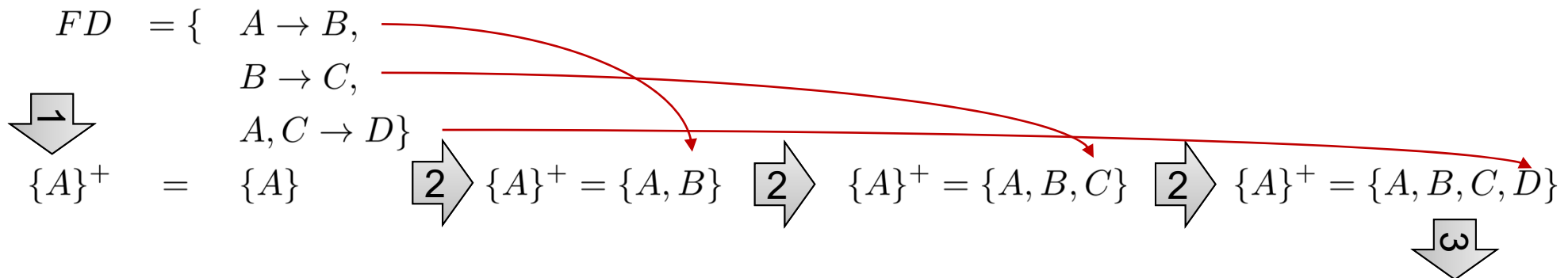
♦ Output: $\{X_1, \dots, X_n\}_{FD}^+ = X_{FD}^+$

♦ Algorithm:

$$1. \quad Closure(X) = X$$

$$2. \quad \forall \{B_1, \dots, B_i\} \rightarrow C \in FD : \{B_1, \dots, B_i\} \in Closure(X) \\ \Rightarrow Closure(X) = Closure(X) \cup C$$

$$3. \quad X_{FD}^+ = Closure(X)$$





MEMBER Algorithm

◆ Input: $FD_R, A \rightarrow B$

◆ Output: true/false

◆ Algorithm

1) $\{A\}_{FD}^+$

2) $B \in \{A\}_R^+ \Rightarrow A \rightarrow B \in FD \Rightarrow Member(FD, A \rightarrow B) = true$

$B \notin \{A\}_R^+ \Rightarrow A \rightarrow B \notin FD \Rightarrow Member(FD, A \rightarrow B) = false$

$$\begin{aligned}\{A\}_{FD}^+ = \{A, B, C, D\} &\Rightarrow A \rightarrow A \in FD \\ &\Rightarrow A \rightarrow B \in FD \\ &\Rightarrow A \rightarrow C \in FD \\ &\Rightarrow A \rightarrow A, B \in FD \\ &\Rightarrow A \rightarrow B, D \in FD\end{aligned}$$

...



Relationship between functional dependencies and keys

- ◆ A set of attributes K is a **superkey** of R if

$$K \rightarrow R \Leftrightarrow \{K\}_{FD}^+ = R$$

- ◆ A set of attributes K is a **key** of R if

$$K \rightarrow R, K \supset K' \not\rightarrow R \Leftrightarrow \{K\}_{FD}^+ = R \wedge K' \subset K : \{K'\}_{FD}^+ \neq R$$



Key determination

- ♦ Generate all possible attribute combinations (candidates), and then check whether they are keys using the MEMBER tests as just shown
 - Checking is possible in linear time → OK
 - But exponentially many attribute combinations exist → not practicable

Remember?

- ♦ Checking each subset..*

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$

- ♦ For being a key..



2	2
3	5
7	877
13	27644437
42	35742549198872617291353508656626642567
55	359334085968622831041960188598043661065388726959079837



- ♦ We need a way to get some FDs from the domain expert and then generate new ones!



Key determination in practice – Key Heuristic

1. First, Splitting(FD), to simplify the FDs
2. All attributes that do not appear on any right side must be in the key (that includes the ones that do not appear in any FD at all).
3. Check whether the set from (2) is already the key using Closure of that attributes. If so, that is the only key and we are done!
4. If not, try all combinations of the attributes from (2) and the remaining attributes.

Because of (4), the Key Heuristic Algorithm still grows exponentially, but in practice, this is manageable in nearly all cases.



Simple example of key determination in practice

$$R = \{A, B, C, D, E\}$$

$$FD = Splitting(FD) = \{A \rightarrow B, \\ B \rightarrow C, \\ C \rightarrow D, \\ A \rightarrow D\}$$

1. Done, FDs already given in Splitting Format
2. Attributes that do not appear on any right side are contained in the key (prime- attributes)

$\{A, E\}$



Why?

3. Check if is already *the* key.. Done. $\{A, E\}$ Is the only key.

Why?

$$\{A, E\}_{FD}^+ = \{A, B, C, D, E\} = R$$



Proof (1)

- ◆ Attributes that do not appear on any right side are contained in the key (prime- attributes).

If a set S of attributes is not contained on any right side of any functional dependency, it is

$$\forall K \subseteq \{R \setminus S\} : S \notin \text{Closure}(FD, K), K_{FD}^+ \neq R$$

If an attribute is not contained on any right side, how will we ever reach it using closure when using only the other attributes? So, S must contain prime attributes.

$$\begin{aligned} R &= \{A, B, C, D, E\} \\ FD = \text{Splitting}(FD) &= \{A \rightarrow B, \} \\ &\quad B \rightarrow C \\ &\quad C \rightarrow D \\ &\quad A \rightarrow D \end{aligned}$$



Proof (2)

- ◆ If the third step of the Key Heuristic Algorithm is successful, there is only one key.

S is the set of attributes, that are not contained on any right side of any FD and it is $S_{FD}^+ = R$

Then it is $\forall K \subseteq \{R \setminus s\}, s \in S : s \notin K_{FD}^+, K_{FD}^+ \neq R$


so, S cannot be shortened without losing its key property and adding attributes would make K a superkey.



Advanced example of key determination in practice

$$R = \{A, B, C, D, E\}$$

$$FD = \text{Splitting}(FD) = \{A \rightarrow B, \\ B \rightarrow A \\ C \rightarrow D\}$$

1. Done, FDs already given in Splitting Format
2. Attributes that do not appear on any right side are contained in the key (prime- attributes) $\{C, E\}$ 
3. Check if is already *the* key.. No.
 $\{C, E\}_{FD}^+ = \{C, D, E\} \neq R$
4. Check combinations..
Two keys!

$$\{C, E, A\}_{FD}^+ = \{A, B, C, D, E\} = R$$

$$\{C, E, B\}_{FD}^+ = \{A, B, C, D, E\} = R$$

$$\{C, E, D\}_{FD}^+ = \{C, D, E\} \neq R$$

~~$$\{C, E, A, B\}_{FD}^+$$~~
~~...~~

More combinations cannot be key. See exercise.



Cover Algorithm (part 1)

♦ Input: functional dependencies FD_R

♦ Output: compact functional dependencies FD_R^C

1. We initialize the set of functional dependencies H and apply Splitting Algorithm: $H = FD_R, H = \text{Splitting}(FD_R)$

2. Minimize *left* sides of all functional dependencies.
When we can reach a right side of a functional dependency also with a shortened left side, the left side is shortened.

$$\begin{aligned} & \{A_1, \dots, A_n\} \rightarrow B \in H \\ & \exists A_i : \{A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_n\} \rightarrow B \in H^+ \\ \Rightarrow & H = H \setminus \{A_1, \dots, A_n\} \rightarrow B \\ & H = H \cup \{A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_n\} \rightarrow B \end{aligned}$$



Example 2nd Step of Cover

$$H = \text{Splitting}(FD_R) = FD_R = \{A \rightarrow B, \\ A, B, C \rightarrow D\}$$

- ◆ The left side of a functional dependency shrinks

$$\begin{aligned} \{A, C\}_H^+ &= \{A, B, C, D\} \\ D &\in \{A, C\}_H^+ = \{A, B, C, D\} \\ \Rightarrow H &= \{A \rightarrow B, \\ &\quad A, C \rightarrow D\} \end{aligned}$$

- ◆ But that's it for this step of Cover, since now it is

$$\begin{aligned} \{A\}_H^+ &= \{A, B\} & \{C\}_H^+ &= \{C\} \\ D &\notin \{A\}_H^+ & D &\notin \{C\}_H^+ \end{aligned}$$

- ◆ And the left side cannot be shortened further



Cover Algorithm (part 2)

3. Remove unnecessary functional dependencies.
If a right side can also be reached from the left side without using the functional dependency, it is unnecessary.

$$\begin{aligned} & \{A_1, \dots, A_n\} \rightarrow B \in H \\ & \textcolor{red}{H}' = H \setminus \{A_1, \dots, A_n\} \rightarrow B \\ & B \in \{A_1, \dots, A_n\}^+_{\textcolor{red}{H}'} \\ \Rightarrow & H = H \setminus \{A_1, \dots, A_n\} \rightarrow B \end{aligned}$$

4. Summarize left sides by reversing Splitting, then it is

$$FD_R^C = H$$

The cover algorithm is not deterministic, depends on the order in which the rules are processed.



Example 3rd Step of Cover

$$H = \text{Splitting}(FD_R) = FD_R = \begin{aligned} &\{A \rightarrow B, \\ &\quad B \rightarrow C, \\ &\quad A \rightarrow C\} \end{aligned}$$

- ◆ We don't need the last functional dependency, since it is

$$\begin{aligned} H' &= \{A \rightarrow B, B \rightarrow C\} \\ C &\in \{A\}_{H'}^+ = \{A, B, C\} \\ \Rightarrow H &= H' \end{aligned}$$

- ◆ But that's it for this step of Cover, since now it is

$$\begin{array}{ll} H' = \{A \rightarrow B\} & H' = \{B \rightarrow C\} \\ C \notin \{B\}_{H'}^+ = \{B\} & B \notin \{A\}_{H'}^+ = \{A\} \end{array}$$

- ◆ And no other functional dependency is unnecessary



Example of computing a canonical cover

$$H = \text{Splitting}(FD_R) = FD_R = \{A \rightarrow B, \\ \textcircled{1} \quad A, B, C \rightarrow D, \\ A, C \rightarrow E, \\ D \rightarrow E\}$$

$A, B, C \rightarrow D$ becomes
 $A, C \rightarrow D$ since
 $D \in \{A, C\}^+ = \{A, B, C, D, E\}$

$\textcircled{2}$

$$\begin{aligned} \{A, C\}_H^+ &= \{A, B, C, D, E\} \\ D &\in \{A, C\}_H^+ \\ \Rightarrow H &= \{A \rightarrow B, \\ &\quad A, C \rightarrow D, \\ &\quad A, C \rightarrow E, \\ &\quad D \rightarrow E\} \end{aligned}$$

$$H' = \{A \rightarrow B, \\ A, C \rightarrow D, \\ D \rightarrow E\}$$

$\textcircled{3}$

$$\begin{aligned} E &\in \{A, C\}_{H'}^+ = \{A, B, C, D, E\} \\ \Rightarrow H &= H' \end{aligned}$$

$A, C \rightarrow E$ Is removed since
 $E \in \{A, C\}_{H'}^+ = \{A, B, C, D, E\}$

$\textcircled{4}$

$$FD_R^C = H = \{A \rightarrow B, \\ A, C \rightarrow D, \\ D \rightarrow E\}$$



Example of computing a canonical cover *compact writing*

$$H = \text{Splitting}(FD_R) = FD_R = \{A \rightarrow B, \\ A, \text{X}, C \rightarrow D, \\ \text{A, C} \rightarrow E, \\ D \rightarrow E\}$$

$A, B, C \rightarrow D$ becomes
 $A, C \rightarrow D$ since
 $D \in \{A, C\}_H^+ = \{A, B, C, D, E\}$

$A, C \rightarrow E$ Is removed since
 $E \in \{A, C\}_{H'}^+ = \{A, B, C, D, E\}$



First normal form 1NF

- ◆ A relation R corresponds to the first normal form if
 1. Every attribute has an atomic range of values
 2. R is free of repeating groups
- ◆ Advantage regarding sortability and processing

How to check it?
Well, we have to see
data examples.

Repeating groups

R:

Vehicle	Models
BMW 1 series	3ZB, 4ZB,...
BMW 3 series	R6ZB,R6ZD,...
Tata Estate	1ZG
Tata Sierra	4ZE



R:

Manufacturer	Name	Model
BMW	1 series	3ZB
BMW	1 series	4ZB
BMW	3 series	R6ZB
BMW	3 series	R6ZD
Tata	Estate	1ZG
Tata	Sierra	4ZE

Non-atomic: Brand, Name



Second normal form 2NF

- ◆ A relation R corresponds to the second normal form if
 1. R corresponds to the first normal form
 2. Every attribute that is not part of a **key** depends on the whole key, and not just on a real subset.
- ◆ Advantage: monothematic tables

R:

Brand	Name	Norm	HQ
BMW	1 series	Euro5A	Munich
BMW	3 series	Euro4	Munich
Tata	Estate	Euro2	Mumbai
Tata	Sierra	Euro1	Mumbai

What are the FDs?



R1:

Brand	Name	Norm
BMW	1 series	Euro5A
BMW	3 series	Euro4
Tata	Estate	Euro2
Tata	Sierra	Euro1

R2:

Brand	HQ
BMW	Munich
Tata	Mumbai

Clearly: the redundancy in the company headquarters (HQ) is not advantageous: it's a waste of storage space and can result in update anomalies and deletion anomalies.

But: how do we check the second criterion?



How to check it?
Key plus Cover!



2NF Check – Key plus Cover

R:

Brand	Name	Norm	HQ
BMW	1 series	Euro5A	Munich
BMW	3 series	Euro4	Munich
Tata	Estate	Euro2	Mumbai
Tata	Sierra	Euro1	Mumbai

◆ Key

(i) All Attributes that are on no right side

Brand, Name

(ii) Check for Key.. checks out, so only key!

$$\{Brand, Name\}_{FD}^+ = R$$

◆ Cover

$$FD = \{Brand, Name \rightarrow Norm, \\ Brand, \text{~~NameBrand} \rightarrow HQ\}~~$$

Brand, Name \rightarrow *HQ* becomes
Brand \rightarrow *HQ* since
 $HQ \in \{Brand\}_{FD}^+ = \{HQ\}$

Violates 2NF constraint: every attribute that is not part of a key (HQ) depends on the whole key, and not just on a real subset (Brand).



Third normal form 3NF

- ♦ A relation R corresponds to the third normal form if
 1. R corresponds to the first normal form
 2. No non-key attribute is transitively dependent on a key.
In other words: every left side is a superkey or the right side is prime.
- ♦ Advantage: monothematic tables

R:

Brand	Name	Engine	Type
BMW	1 series	Diesel	Fossil
BMW	3 series	Electric	Renewable
Tata	Estate	Otto	Fossil
Tata	Sierra	Electric	Renewable

What are the FDs?

Clearly: the redundancy in the engine type is not advantageous: it's a waste of storage space and can result in update anomalies and deletion anomalies.



R1:

Brand	Name	Engine
BMW	1 series	Diesel
BMW	3 series	Electric
Tata	Estate	Otto
Tata	Sierra	Electric

R2:

Engine	Type
Diesel	Fossil
Otto	Fossil
Electric	Renewable
H	Renewable



How to check it?
Key plus Cover!



3NF Check – Key plus Cover

R:

Brand	Name	Engine	Type
BMW	1 series	Diesel	Fossil
BMW	3 series	Electric	Renewable
Tata	Estate	Otto	Fossil
Tata	Sierra	Electric	Renewable

◆ Key

(i) All Attributes that are on no right side

Brand, Name

(ii) Check for Key.. checks out, so only key!

$$\{Brand, Name\}_{FD}^+ = R$$

◆ Cover

$$FD = \{Brand, Name \rightarrow Engine, \\ Engine \rightarrow Type\}$$

Is in 2NF: Every non prime attribute depends on the whole key and not just on a real subset



Violates 3NF constraint: every left side is superkey or right side is prime



Boyce-Codd normal form (BCNF)

- ◆ A relation R corresponds to the Boyce-Codd normal form if

1. R corresponds to the first normal form
2. Every left side is a superkey.



How to check it?
Key plus Cover!

- ◆ Advantage: monothematic tables

Is in 3NF: left side is
superkey or right
side is prime



$$FD = \{A, B \rightarrow C, \\ C \rightarrow A\}$$

Is in 2NF: Every non prime
attribute depends on the whole
key and not just on a real subset



Violates BCNF: not
every left side is
superkey



*Challenge: try to find a
sound real-world example
that is 3NF, but not BCNF
and win a price!*

*Exercise: proof, that
all relations with
only 2 attributes are
in BCNF.*



The normal forms and their constraints

1NF	Every attribute has an atomic range of values R is free of repeating groups
2NF	Every attribute that is not part of a key depends on the whole key, and not just on a real subset.
3NF	Every non-key attribute is transitively dependent on a key.
BCNF	Every attribute set that other attributes functionally depend on is a superkey.

*Every **non-key** attribute must provide a fact about the key (1NF), the whole key (2NF), and nothing but the key (3NF), so help me Codd (and Boyce for all (BCNF)).*



Which NF is the best?

- ◆ 1NF – It's a must!
Its nonsense for CRUD to have non-atomic values or lists in data fields.
- ◆ 2NF – It's a must!
The amount of redundancies and anomalies is huge when attributes depend only on a part of the key.
- ◆ 3NF – Its ok, better than nothing
Still redundancies and anomalies, but manageable.
- ◆ BCNF – That's the one we want
Free of anomalies, in practice we will achieve this in nearly all applications!





Normalization using Decomposition

1. Identify Key and apply Cover to FDs
2. Identify the first functional dependency that *violates BCNF*, let's say
 $A_1, \dots, A_n \rightarrow B_1 \dots, B_m$

3. Decompose R into

$$R_1 = \{A_1, \dots, A_n\}_{FD}^+$$

$$R_2 = R \setminus R_1 \cup \{A_1, \dots, A_n\}$$

4. Check whether all FDs of R can still be found in the decomposition. This property is called **dependency preservation**. It should be

$$FD_{R_1} \cup FD_{R_2} = FD_R$$

5. Check whether the decomposed schemas are in BCNF. If not, continue with the decomposed schemas.

Decomposition is not always dependency preserving but always lossless (later)



Example: Decomposition

$$FD = \{Brand, Name \rightarrow Engine, \\ Engine \rightarrow Type\}$$

Violates BCNF constraint:
every left side is superkey.

$$R_1 = \{Engine\}_{FD}^+ = \{Engine, Type\}$$

$$\begin{aligned} R_2 &= R \setminus R_1 \cup \{Engine\} \\ &= \{Brand, Name, Engine, Type\} \setminus \{Engine, Type\} \cup \{Engine\} \\ &= \{Brand, Name, Engine\} \end{aligned}$$

$$\begin{aligned} FD_{R_1} &= Engine \rightarrow Type \\ FD_{R_2} &= Brand, Name \rightarrow Engine \end{aligned}$$

The FDs can be expressed
using the decomposed schemas
and they are both in BCNF





Example 2: Decomposition

$$FD = \{A, B \rightarrow C, \\ C \rightarrow A\}$$

What is the key?

Violates BCNF constraint:
every left side is superkey.

$$R_1 = \{C\}_{FD}^+ = \{C, A\}$$

$$\begin{aligned} R_2 &= R \setminus R_1 \cup \{C\} \\ &= \{A, B, C\} \setminus \{A, C\} \cup \{C\} \\ &= \{B, C\} \end{aligned}$$

$$FD_{R_1} = C \rightarrow A$$

$$FD_{R_2} = \emptyset$$

Decomposition not successful. The functional dependency $A, B \rightarrow C$ cannot be expressed using the decomposed schemas

There is nothing we can do. Also changing the order of processing the FDs does not help, since we only have one FD that violates BCNF. The relation R cannot successfully be decomposed into a BCNF normal form using Decomposition. The decomposition is not **dependency preserving**.



Dependency preservation – Why is it important?

- ◆ Let's assume we have

$$FD_R = \{MatNr \rightarrow Name, \\ MatNr \rightarrow Birthday\}$$

- ◆ And we (accidentally) decompose into

$$R_1 = \{Name, Birthday\}$$

$$FD_{R_1} = \emptyset$$

$$R_2 = \{MatNr, Name\}$$

$$FD_{R_2} = \{MatNr \rightarrow Name\}$$

- ◆ Then we would have something like this

R1: Name	Birthday	R2: MatNr	Name
Maria	01.03.2000	1	Maria
Max	02.04.2002	2	Max
Maria	22.12.2002	3	Maria

- ◆ We cannot store $(3, Maria, 22.12.2002)$ in the database, since $MatNr \rightarrow Birthday$ cannot be stored in any of the decomposed tables.



Normalization using Synthesis

- ♦ Synthesis is **always dependency preserving** but has other issues. It is **not always lossless** (later) and is more likely to decompose into more tables than Decomposition.

1. Calculate the cover $FD_R = Cover(FD_R)$
2. For every functional dependency, create a relation. Summarize right sides.

$$\begin{aligned} FD_R &\ni A_1, \dots, A_n \rightarrow B_1 \\ \Rightarrow R' &= \{A_1, \dots, A_n\} \cup \{B \mid A_1, \dots, A_n \rightarrow B \in FD_R\} \end{aligned}$$

3. If none of these relations R' contains a superkey of R , create an additional relation

$$\forall R' : \{R'\}_{FD_R}^+ \neq R \Rightarrow \text{add } R^* : \{R^*\}_{FD_R}^+ = R$$

4. Remove unnecessary relations that are contained in other relations with

$$R'' \subset R'$$



Example 2: Synthesis

$FD = \{A, B \rightarrow C, C \rightarrow A\}$

Violates BCNF constraint:
every left side is superkey.

Result of Decomposition:

$R_1 = \{C, A\}$

$R_2 = \{B, C\}$

Not dependency
preserving

1. Is in cover
2. For every functional dependency, create a relation. Summarize right sides.

$R_1 = \{A, B, C\}, R_2 = \{C, A\}$

3. Remove R_2 since $R_2 \subset R_1 = R$. Again, **no normalization**.



Example 3: Synthesis

1. Canonical cover given
2. Synthesis brings
3. One of the relations contains a superkey of R. Done.

$$\begin{aligned} R &= \{A, B, C, E\} \\ FD_R &= \{A \rightarrow B, \\ &\quad B \rightarrow C, \\ &\quad B \rightarrow A, \\ &\quad C \rightarrow E\} \\ R_1 &= \{A, B, C\} \supset \{A, B\} \\ R_2 &= \{C, E\} \end{aligned}$$

Same result as Decomposition:

$$\{A\}_{FD_R}^+ = R, A \in R_1$$



Lossy join during Synthesis

- Given $R = \{A, B, C\}$
 $FD_R = \{A \rightarrow B, C \rightarrow B\}$

R:	A	B	C
	1	2	3
	4	2	5

- Synthesis will result in

$$R_1 = \{A, B\}$$

$$R_2 = \{B, C\}$$

R1:	A	B
	1	2
	4	2

R2:	B	C
	2	3
	2	5

- Not a successful decomposition, (super)key in **no** relation.

$$\{A, C\}_{FD_R}^+ = R$$
$$R_1 \not\supseteq \{A, B\} \not\subseteq R_2$$

R'=R1 join R2:	A	B	C
	1	2	3
	4	2	5
	1	2	5
	4	2	3

Join loses information

Problem: No (super)key in the resulting relations makes us loose the information which tuple originally belonged together → **lossy join**



Lossless join during Synthesis

- Given $R = \{A, B, C\}$
 $FD_R = \{A \rightarrow B, B \rightarrow C\}$

R:	A	B	C
	1	2	3
	4	2	3

- Synthesis will result in

$$R_1 = \{A, B\}$$

$$R_2 = \{B, C\}$$

R1:	A	B
	1	2
	4	2

R2:	B	C
	2	3

- Successful decomposition, (super)key in a relation.

$$\{A\}_{FD_R}^+ = R$$
$$R_1 \supseteq \{A\} \not\subseteq R_2$$

R'=R1 join R2=R:	A	B	C
	1	2	3
	4	2	3

Join keeps information



Solution: (Super)key in the resulting relations makes us keep the information which tuple originally belonged together → **lossless join**



Lossless join decomposition - formally

- ◆ Formally: the decomposition of an attribute set

$$R \text{ into } R_1, \dots, R_n \text{ with } R = \bigcup_{i=1}^n R_i$$

is a **lossless join decomposition** with regard to a set of FDs if and only if the following applies to the decomposition:

$$r(R) = R_1 \bowtie \dots \bowtie R_n$$

- Comment: we can't lose any tuples due to the join, but we could get additional tuples. So why is it called "lossless"? Because we have lost the information about which tuples were originally there!
- Decomposition is "lossy" if $R_1 \bowtie R_2 \supset R$
- Decomposition is "lossless" if $R_1 \bowtie R_2 = R$



User Cover to generate a Superkey

- ◆ Since Synthesis requires at least a superkey to be successful, we can generate one superkey instead of generating all keys using Key Heuristic. That reduces the complexity of Synthesis from exponential to polynomial.
- ◆ To do so, we add an artificial functional dependency to the input of Cover. The remaining FD pointing to delta will hold a superkey on the right side after Cover. $R \rightarrow \delta$

- ◆ Example

$$\begin{array}{lcl} A, B & \rightarrow & C \\ B & \rightarrow & C \\ C & \rightarrow & D \\ D, E & \rightarrow & F \\ A, B, D & \rightarrow & F \\ (A, B, C, D, E, F, G & \rightarrow & \delta) \end{array} \quad \xrightarrow{\text{Cover}} \quad \begin{array}{lcl} B & \rightarrow & C \\ C & \rightarrow & D \\ D, E & \rightarrow & F \\ A, B & \rightarrow & F \\ (A, B, E, G & \rightarrow & \delta) \end{array}$$

One superkey
of R



So lets apply Sythesis and Decomposition

$$R = \{A, B, C, D, E, F, G\}$$

$$B \rightarrow C$$

$$C \rightarrow D$$

$$D, E \rightarrow F$$

$$A, B \rightarrow F$$

$$(A, B, E, G \rightarrow \delta)$$

B → C violates BCNF :

$$R_1 = \{B, C, D\}$$

$$R_2 = \{A, B, E, F, G\}$$

D, E → F is lost ↯

C → D violates BCNF :

$$R_1 = \{C, D\}$$

$$R_2 = \{A, B, C, E, F, G\}$$

D, E → F is lost ↯

D, E → F violates BCNF :

$$R_1 = \{D, E, F\}$$

$$R_2 = \{A, B, C, D, E, G\}$$

A, B → F is lost ↯

A, B → F violates BCNF :

$$R_1 = \{A, B, F\}$$

$$R_2 = \{A, B, C, D, E\}$$

D, E → F is lost ↯

- ◆ So, Decomposition fails in every possible combination. Synthesis instead?

$$R_1 = \{B, C\}$$

$$R_2 = \{C, D\}$$

$$R_3 = \{D, E, F\}$$

$$R_4 = \{A, B, F\}$$

$$R_5 = \{A, B, E, G\}$$

Works every time, when it is a real decomposition, and no relation is contained in another. Always dependency preserving. Lossless when superkey is added. Always BCNF*. But many relations!



Decomposition can also be used to reach 3NF

- When Decomposition into BCNF fails and/or Sythesis has too many tables, we apply Decomposition to reach 3NF by tackling the FDs that violate 3NF:

$$R = \{A, B, C, D, E, F\} \quad Keys = \{A, E, B\}^+ = \{A, E, C\}^+ = \{A, E, D\}^+ = R \neq \{A, E, F\}^+$$

$$B \rightarrow C \quad Prime = \{A, B, C, D, E\}$$

$$C \rightarrow D$$

$$D, E \rightarrow F$$

$$D \rightarrow B$$

$D \rightarrow B$ violates BCNF :

$$R_1 = \{D, B, C\}$$

$$FD_{R_1} = \{B \rightarrow C, C \rightarrow D, D \rightarrow B\}$$

$$R_2 = \{A, D, E, F\}$$

$$FD_{R_2} = \{D, E \rightarrow F\}$$

$$R_{2.1} = \{D, E, F\}$$

$$R_{2.2} = \{A, D, E\}$$

BCNF, OK.

Not further
decomposable
into BCNF

So, either we take two relations
in 3NF using Decomposition or
we take four relations in BCNF
using Sythesis.

$D, E \rightarrow F$ violates 3NF :

$$R_1 = \{D, E, F, B, C\}$$

$$FD_{R_1} = \{B \rightarrow C, C \rightarrow D, D, E \rightarrow F, D \rightarrow B\}$$

$$R_2 = \{A, D, E\}$$

$$FD_{R_2} = \{\}$$

Now in BCNF!



Requirements for decomposition / transformation

- ◆ We would generally like to have three important properties for the decomposition (transformation) of a relational schema:

1) Elimination of the anomalies

- No more anomalies should occur in the resulting relational schemas. This is the case if the schemas are all in BCNF.

2) Lossless join decomposition

- Precisely those tuples (application data) of the original relation should be able to be derived again from the tuples of the decomposed relational schemas. This is the case after Decomposition or Synthesis with key.

3) Dependency preservation

- The functional dependencies that can be derived from the keys of the decomposed relations should be equivalent to the original FDs. This is the case after Synthesis, but not necessary after Decomposition.



Practical example of 3NF but not BCNF, not normalizable

	Postcodes:	Place	State	Street	Postcode
$FD = \{Postcode \rightarrow Place, State$ $Street, Place, State \rightarrow Postcode\}$		Frankfurt	Hesse	Göthestraße	60313
		Frankfurt	Hesse	Galgenstraße	60437
		Frankfurt	Brandenburg	Göthestraße	15234
$Keys = \{Street, Postcode\}, \{Street, Place, State\}$					
$R_1 = \{Postcode, Place, State\}$	Lost FD: $Street, Place, State \rightarrow Postcode$				
$R_2 = \{Postcode, Street\}$					

- ♦ Is in 3NF but not in BCNF and cannot be decomposed into BCNF using Decomposition. Let's try Synthesis:

$$\begin{aligned} R_1 &= \{Postcode, Place, State\} \\ R_2 &= \{Street, Place, State, Postcode\} \supset R_1 \end{aligned}$$

- ♦ Also no, not a real decomposition. Stays in 3NF.

Decomposition algorithms and properties

Algorithm	Lossless join decomposition	Dependency preservation	Solution	Time complexity	Advantages/ disadvantages
Decomposition BCNF	Yes, always	No, not always	BCNF if dependency preserving	Exp.	Few relations, no anomalies, can fail.
Synthesis	Yes, if superkey is used	Yes, always	Mostly BCNF if <i>useful</i> decomposition	Poly.	Many relations, sometimes no decomposition.
Decomposition 3NF	Yes, always	Yes, always	Can always be reached.	Exp.	Has anomalies, cannot fail.



We can do now

- ◆ We can derive FDs from experts
- ◆ We can minimize FDs using Splitting
- ◆ We can solve the member problem using Closure
- ◆ We can derive keys from FDs
- ◆ And we can compress FDs
- ◆ Using Cover and Key Heuristic, we can decide whether a schema is a good schema and in BCNF.
- ◆ Transform a schema that is in a bad normal form into a good schema that is (hopefully) in BCNF or sometimes only 3NF using Decomposition or Synthesis.