4.Normalization: Quality of relational designs

Lit: Kemper/Eickler: chap 6; Garcia-Molina/Ullman/Widom: chap 3.4 ff.; Elmasr/Navathe: chap 14 , Kifer et al.: chap. 6

4.1.1 Design quality

Freie Universität Berlin

What is a "good" conceptual model ?

Usually many alternatives.
No clear guidelines, best practice.

Wanted: **Formal methods for comparing designs**

But…

Use common sense!
Simple problems have simple solutions!
"Design is an art but a science"

Informal guidelines (1)

Freie Universität Berlin

Avoid redundancies:

Example:

```
CREATE TABLE employee (
    p#              serial,
    name            VARCHAR(30),
    ...
    qualification   VARCHAR(20), -- typicallymore
                                 -- than one
                                 --
    PRIMARY KEY (p#, qualification));
```

More than one qualification, primitive values
⇒ `p#` is not a key. *Why?*

Dumb design!

p#	name ...	qual
22	Meyer ...	programmer
27	Müller ...	secretary
22	Meyer ...	DB admin

Informal guidelines (2)

Freie Universität Berlin

Avoid modeling more than one object from reality in one entity / relation

```
CREATE TABLE Experiment (
        id          SERIAL PRIMARY KEY,
        responsible_Scientist VARCHAR(40),
        institute           VARCHAR (30),
        phone               INT,
        purpose             VARCHAR(100),
        start               TIMESTAMP,
        endTime             TIMESTAMP,
        result              INT)
```

What is the problem with this table design?

Informal guidelines (2)

Freie Universität Berlin

Avoid modeling more than one object from reality in one entity / relation

```
CREATE TABLE Experiment (
        id          SERIAL PRIMARY KEY,
        responsible_Scientist VARCHAR(40),
        institute           VARCHAR (30),
        phone               INT,
        purpose             VARCHAR(100),
        start               TIMESTAMP,
        endTime             TIMESTAMP,
        result              INT)
```

Redundancy ....

4.1.2 Update Anomalies

Freie Universität Berlin

Redundancies may cause "anomalies"

Deletion of a row may delete all data about a different object

Update of an attribute may cause update on many rows

Insertion may be difficult / impossible, since data are missing

"update anomaly": deletion, update or insertion anomaly

Update anomalies: Examples

```
CREATE TABLE Experiment (
   id SERIAL PRIMARY KEY,
   responsible_Scientist VARCHAR(40),
   institute         VARCHAR (30),
   phone             INT,
   purpose   VARCHAR(100),
   start             TIMESTAMP,
   endTime           TIMESTAMP,
   result            INT)
```

Table definition using SQL / DDL (PostgresSQL)

```
DELETE  FROM Experiment WHERE  result < 10
```

Data on experimenter may be lost!

Update anomaly

Example

```
update Experiment set phone = 37784
  where responsible_Scientist
      = 'Müller-Lüdenscheid'
```

⇨ all those experiment tuples the experimentor of which was `'Müller-Lüdenscheid'` have to be changed

⇨ **update anomaly**

What is an insertion anomaly?

4.1.3 Functions, Functional dependencies

Important formal concept: **Functions**
Functional dependencies (FD) **generalize key concept**

FDs are used to **formalize integrity constraints** on attributes and relationships

Remember **keys** and **1:N relationships.**

If *k* is a key of relation *R(k,a,b,...c)* and *k_val* a key-value, what can be said about the attribute values?

Next to trivial: values are unique for a given key!

Example:

```
Experiment (id, responsible_Scientist,
            institute, phone,...result)
```

"The same responsible_person cannot have different affiliations (institutes)"

```
{responsible_Scientist} -> {institute}
```

is a **function** which expresses the **constraint**:

Formal notation of RDM

Relation: R ⊆ dom(a1) x dom(a2) x … x dom(an)

Attribute set: **Σ(R) = {a1, a2, …, an}** = R_A , signature

Tuple: r ∈ R

Degree of R: number of attributes

Relation Schema: R(a1, a2, …, an)

Database **schema**: set of relation schemas

different notations in use

Terminology:

Relation = table (file)
but relations are sets.
tables may have duplicate entries
Tuple = row, record
Attribute = field (component)

Functional Dependencies and keys

Property of a key : at most **one row for each value**

⇒

Given a key, i.e. values for the key attributes, then the value of each a ∈ Σ(R) is unique or all values are undefined.

⇒ **each subset of Σ(R) is functional dependent on K**

key

Student			
fName	name	email	matrNo
Tina	Müller	mueller@…	13555
Anna	Meier	mei@…	12555
Carla	Maus	piep@…	11222

For a given key value, there is a unique value for each attribute e.g.
`{matrNo}  → {fName}`

Functional Dependencies

Functional and **key dependencies** are **constraints** (invariants) of the application domain

"Functional dependency" constraints *have to be identified during requirements engineering* – like all constraints.

Ultimate goal: DBS monitors compliance with DB state.

Example:

```
Experiment (id, responsible_Scientist, institute,
            phone,...result)
```

What has to be done, when a new experiment is inserted?

Motivation for Normal Forms

Suppose we can find a relational schema which **has only key-induced functional dependencies** (FD)
(and "trivial" ones like {a,b} -> {b})

How can we **efficiently check the DB state after an update with respect to FD**? Do they still hold?

A "good" schema avoids

Update anomalies

Costly check of functional dependencies after update

Functional Dependency: Definition

Def.: **Functional Dependencies** (FDs)

Let A = $\Sigma(R)^* = \{a,b,c,...a_i,..\}$ be the attribute set of a relation Rand X, $Y \subseteq A$, r, r' $\in$ R, r $\neq$ r'

Y is functionally dependent on X (written: **$X \rightarrow Y$**)

$:\Leftrightarrow$ $(\forall xi \in X)$ r.xi = r'.xi $\Rightarrow (\forall yi \in Y)$ r.yi = r'.yi

- As we know: invariants are independent of the particular database state
- They must hold at all times, i.e. they **restrict the valid states of the database**

* Σ(R) : Attribute set of relation R

4.1.4 FD Properties

Trivial functional dependency

$X \subseteq Y \Rightarrow Y \rightarrow X$

Augmentation

$Z \subseteq A=\Sigma(R), \quad X \rightarrow Y \Rightarrow XZ \rightarrow YZ$

Transitivity

$X,Y,Z \subseteq A=\Sigma(R),\ X \rightarrow Y,\ Y \rightarrow Z \ \Rightarrow X \rightarrow Z$

Proof?

Notation XY -> Z means $X \cup Y$ -> Z

Implied and inferred FD

A functional dependency $Y \rightarrow Z$ is called **implied** by a set F= {F1, … , Fn} of functional dependencies, if $Y \rightarrow Z$ can be **proven from F.**

A functional dependency $Y \rightarrow Z$ can be **inferred** ($\vdash$)by a set of **inference rules** R = {r1,…rm} from set F = {F1, … , Fn} of functional dependencies if $Y \rightarrow Z$ can be constructed by a finite number of **syntactic transformations** of F according to rules ri

Armstrong inference rules

Given a set of FDs, find all implied FD's

A **sound, complete, minimal set** (Armstrong axioms):

$Y \subseteq X \vdash X \rightarrow Y$	(I: inclusion)
$\{X \rightarrow Y\ ,\ Y \rightarrow Z\} \vdash X \rightarrow Z$	(T: transitivity)
$\{X \rightarrow Y\} \vdash XZ \rightarrow YZ$	(A: augmentation)

Sound:
Only implied FDs are constructed by the inference rules

Complete:
Every implied FD will be produced by a finite number of inferences

Types of Functional Dependencies

Freie Universität Berlin

Given
- schema signature $\Sigma(R) = \{a1,\ldots,an\}$
- Primary key $P = \{p_1,\ldots p_e\}$
- Set of **candidate keys** $C = \{ \{k_1,..,k_f\}, \ldots,\{k'_1,\ldots,k'_g\}\}$

Def.: $K = P \cup (\cup C)$ are called **prime** (or key) **attributes**
i.e. attributes belonging to any candidate key.
S(R) \ K : **non-prime** (or non-key) attributes

Types of FDs

Freie Universität Berlin

Types of functional dependencies:
1. **Key dependencies**
2. **Partial dependencies** on one of the candidate keys
   expl.: `{p#} -> {name}`
   `// R(p#,name,qualification, ...)`
   `since key is {p#,qualification}`

3. Dependencies among **non-key attributes**
   expl.: `{responsible_Scientist} -> {institute}`
4. **Dependencies among** attributes of different candidate keys

Normalization

Freie Universität Berlin

Roadmap

Functional dependencies may cause "update anomalies" ☑

Update anomalies cause troubles
⇒ find relational schema without "anomalies" in case of update ☑

Define **"Normal forms"** for relations which do not show (all) anomalies

Given a set of functional dependencies, find **algorithm which generates a relational schema in some normal form**

4.2 Normal Forms

Freie Universität Berlin

4.2.1 Definitions

Def.: First normal form
A relation is in 1NF :⇔
all attributes are single valued and atomic

Example:
`Customer (c_id, name, …, {phone}, …)`
`(53,'Miller', …, {47653, 478992},…)`

Equivalent to *Key dependency property*:
every attribute is functionally dependent on e very candidate key

Second Normal Form (2NF)

Freie Universität Berlin

Def.: R is in **Second Normal Form (2NF)**, :⇔
$\forall X \subseteq \Sigma(R), \forall a \in \Sigma(R)$:
$a \notin X \wedge$ a is not a prime attribute $\wedge$ X -> a
⇒ X is a key or a superset of a key
but not a proper subset of any key of R

This <u>means</u>: **No non-prime** attribute **functionally depends** on only **a part of a key** ("<u>**No partial dependency**</u>")

Example: `Building(bNo, roomNo, rSeats, bAdr, bNoRooms,…)`

Third Normal Form (3NF)

Freie Universität Berlin

Def.: R is in **third normal form** (3NF) :⇔
$\forall X \subseteq \Sigma(R), \forall a \in \Sigma(R)$: $a \notin X \wedge X \rightarrow a$
⇒ X contains a key or a is prime

⇔
There is **no functional dependency between non-prime attributes** (or attribute sets). Proof?

Example:
`Experiment (id, responsible_Scientist, institute,`
`phone,...result)`

but: `{responsible_Scientist} → {institute}`
⇒ **not in 3NF**

More on 3NF

Equivalent definition: R is in 3NF :⇔ no non-prime attribute depends transitively on a key.

A non-prime attribute y is **transitively dependent** on a key K, if $K \rightarrow X$ and $X \rightarrow y$ and *not* $X \rightarrow K$
Notation: $K \rightarrow X \rightarrow y$

```
Experiment
  (id, responsible_Scientist, institute,..,result)
```

© HS-2010 04-DBS-NF-25

2NF and 3NF

R is in 3NF $\Rightarrow$ R is in 2NF

Proof:

Suppose R not in 2NF

$\Rightarrow$ exists candidate key K and $X \subset K$ and non-prime attribute a and $X \rightarrow a$ (i.e. exists partial dependency)

Since K is a key $\Rightarrow K \rightarrow X \rightarrow a$, contradiction to 3NF

© HS-2010 04-DBS-NF-26

Design quality and Normal Forms

```
Experiment
  (id, responsible_Scientist, institute,..,result)
```

Indicates the modeling of **two different 'real world entities' as one relation**

Split into two relations:

```
Experiment
  (id, purpose, start , responsible,..,result);
                                 foreign key
                primary key
Experimentor ( p_id, institute, phone,..)
```

© HS-2010 04-DBS-NF-27

DESIGN QUALITY: what do we have?

1. **Key dependencies** $\Leftarrow$ 1NF
2. **No partial dependencies** on candidate keys $\Leftarrow$ 2NF
3. **No dependencies among non-key attributes** $\Leftarrow$ 3NF
4. Dependencies among attributes of different candidate keys ??

$\Rightarrow$ there is at least one more Normal Form which excludes FDs between prime attributes.

© HS-2010 04-DBS-NF-28

Beyond third NF

Dependencies among key attributes

There are relations in 3NF with nontrivial functional dependencies!

Example (*): R(p, o, s, n)
with keys {o,s,n} and {p,s,n}, FD p -> o

R in 3NF, but transitive dependency involving key attribute o: {p,s,n} -> p -> o

(*) Interpretation e.g.: PLZ, Ortsteil, Straße, Nummer (in Germany)
Annahme: (Ort, Straße, Nr) , (PLZ, Straße, Nr) eindeutig

© HS-2010 04-DBS-NF-29

Beyond 3NF

Characterizing essential (= nontrivial and no superkey involved) FDs in 3NF relations

Proposition:
If R is in 3NF and X is a *proper subset of a candidate key K* and $X \rightarrow a$ for some attribute a
$\Rightarrow a \notin K \wedge \exists$ key K': $a \in K'$

Corollary: If $X \rightarrow a$ is an essential FD of a relation in 3NF, and X not a superkey
$\Rightarrow$ X and a belong to different candidate keys

© HS-2010 04-DBS-NF-30

Boyce Codd Normal Form (BCNF)

Def.: A relation R is in **BCNF**
$:\Leftrightarrow$
if there is a non-trivial dependencies $X \rightarrow a$
then X is a superkey of R

Equivalent to:

$X \rightarrow a \Rightarrow$
(i) trivial or
(ii) X contains a key of R
(iii) There a no essential FDs in R

Obvious consequence: **BCNF $\Rightarrow$ 3NF**

3NF and BCNF

3NF more important in practice than BCNF

Partial dependencies of candidate keys infrequent

R relation in 3NF and candidate keys have only one attribute each $\Rightarrow$ R is in BCNF

R in 3 NF and at most one candidate key has more than one attribute $\Rightarrow$ R is in BCNF

Proof?

BCNF vs 3NF

Last proposition useful in many practical situation:
If a relation R has a multi-attribute key and a unique identifier (e.g. a sequence number) then 3NF implies BCNF

e.g. `Customer ( cID, name, city, street, no, discount, …)`
has keys {`cID`} and {`name, city, street, no`}

4.2.2 Lossless property and preserving dependencies

Normalization (by decomposition)

Given relation R having schema $\Sigma(R)$ and
FD = $\{X \rightarrow Y \mid X,Y \subseteq \Sigma(R)\}$ set of FDs,
Find a set $R_1, \ldots, R_n$ of relations in 3NF / BCNF such that:

- $\Sigma(R) = \cup\, \Sigma(R_i)$
- For each $f = X \rightarrow Y \in FD$ there exists R_i such that $X \cup Y \subseteq \Sigma(R_i)$ "Dependency preserving"
- R can be reconstructed from R_i, i=1..n

"Lossless

Joining relations

When relation R has been split into relations $R_1, R_2, \ldots, R_n$, reconstruction of R from $R_1, \ldots, R_n$ by means of the **join operator**

Join operation (natural join):
concatenate those tuples of R and S which have same name and same value. Eliminate the redundant attribute.

R_1

a	b
1	2
4	2
3	3

$\bowtie$ b=b

R_2

b	c
2	3
3	5

=

a	b	~~b~~	c
1	2	~~2~~	3
4	2	~~2~~	3
3	3	~~3~~	5

"Natural join"

Lossless property

Criterion for „preserved information (losslessness)":

$R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n = R$

Example:

R =

a	b	c
1	2	3
4	2	5

a	b
1	2
4	2

$\bowtie$

b	c
2	3
2	5

=

a	b	c
1	2	3
4	2	5
1	2	5
4	2	3

$\supset$ R

Lossless joins

But

a	b	c
1	2	3
4	2	5

a	c
1	3
4	5

$\bowtie$

c	b
3	2
5	2

=

a	b	c
1	2	3
4	2	5

FD = {a -> b, c -> b}

In general:

Decomposition of R into R1 and R2 is **lossless**, if

$\Sigma(R1) \cap \Sigma(R2) \rightarrow \Sigma(R2)$ or $\Sigma(R1) \cap \Sigma(R2) \rightarrow \Sigma(R1)$

Lossless joins

Lossless decomposition and keys

$\Sigma(R1) \cap \Sigma(R2)$ -> $\Sigma(R2)$ or $\Sigma(R1) \cap \Sigma(R2)$ -> $\Sigma(R1)$

⇨

The common attribute(s) of R1 and R2 are a key **(or a superset of a key) of R1 or R2**

⇨Functional dependencies are transformed into **key dependencies**

⇨ Invariance property expressed by FDs may now be checked by checking the **primary key property** - efficiently done by every DBS

BCNF and 3NF

BCNF does **not always guarantees both** the **lossless property and dependency preservation**

Example:
R(p, o, s, n) with keys {o,s,n} and {p,s,n}, FD p -> o
Normalisation to BCNF:
R1 (p,s,n) and R2(p,o)
⇨ Dependency (o,s,n) -> p is lost

Consequence:
Normalization to 3NF is the **best to achieve** in general

4.2.5 Multivalued dependencies and 4NF

Example

Hobbies(name,affiliation,hobby)

Meier	FUB	skiing
Müller	TUB	trekking
Meier	FUB	trekking
Schulze	HU	skating
Schulze	FU	tennis
Schulze	FU	skating
Schulze	HU	tennis

Assumption: a person can have one or more affiliations, and one or more hobbies, e.g

Schulze | {HU.FU} | {tennis,skating}

These rows must exist (MVD restriction)

- Two multivalued attributes: affiliation, hobbies, both dependent on name.
- Introduce redundancy
- MVD defines which tuples must exist.

MVD

Example with MV attributes:

Schulze	{FU, HU}	{tennis,skating}

MVD are invariants on the existence of tuples if multivalued attributes are represented in 1NF

Def.: MVD (multivalued dependency)
Let R = (a, y, b),
b is multivalued dependent on a (a ->>b)
if for each value v of a
$\{v\} \times (\pi_y(\sigma_{a=v} R)) \times (\pi_b(\sigma_{a=v} R)) \subseteq R$

MVD: example

Example:
{'Meier'} × {'FU'} × {'skiing', 'trekking'} ⊆ Person
{'Müller'} × {'TU'} × {'trekking'} ⊆ Person
{'Schulze'} × {'HU',FU} × {'skating','tennis'} ⊆ Person

Meier	FUB	skiing
Müller	TUB	trekking
Meier	FUB	trekking
Schulze	HU	skating
Schulze	FU	tennis
Schulze	FU	skating
Schulze	HU	tennis

'hobby' is mv-dependent on 'name': **`name ->> hobby`**

Fourth Normal Form

Def.: Let A, B $\subseteq \Sigma(R)$; R is in **Fourth Normal Form**
if for every MVD A ->> B
(i) B $\subseteq$ A or (ii) B = $\Sigma(R) \setminus A$ or
(iii) A contains a key

Example not in 4NF, check

Normalized representation:

Müller	TUB
Meier	FUB
Schulze	HU
Schulze	FU

Müller	trekking
Meier	trekking
Meier	skiing
Schulze	skating
Schulze	tennis

Normal forms: summary

Normal forms are **quality criteria** for **database design.**
Important: **1NF – 3NF**
Exotics: **BCNF, 4NF (and higher!)**

2NF / 3NF formalize the basic design principle:
"Never mix up different real world entities into a single design object (e.g. entity)"

2NF / 3NF already defined for ERM, since FDs are given (result of requirement analysis, just like key dependencies) .

4.3 Finding Normal Forms

Invariants of application domain have to be made explicit during requirements analysis

e.g. "A scientist has at most one affiliation – her institute"
" A region-id is unique within a country"
" A person has exactly one date of birth"

Formalization ⇨ **Functional Dependencies**
Wanted: **algorithm producing "good" relational schemas from the set DEP of all FDs**

FDs and Normal Forms

Given a set of dependencies DEP there are two approaches:

- **Synthesis**
  Set up relations in such a way, that
  – All attributes are consumed
  – The relations are in normal form
- **Decomposition**
  For a given set of relations find those which are not normalized with respect to DEP and decompose them into normalized relations

Decomposition: eliminate FDs

Given $\Sigma(R) = U$ and DEP the set of FDs
Algorithm DECOMP(R):
(i) Find the set of keys K:
$K \rightarrow U \in DEP$ or $K \rightarrow U \in DEP^+$
(DEP+ set of all implied dependencies)
(ii) Eliminate all transitive dependencies by splitting recursively:
{if $K \rightarrow Y$ -> a is a transitive FD in R_k, split R_k into R_i, R_j
$\Sigma(R_i) = \Sigma(R_k) \setminus \{a\}$, $\Sigma(R_j) = Y \cup \{a\}$
}
(iii) If no more relations R_k with transitive dependency exit else for all R_k DECOMP(R_k)

Synthesis

Disadvantage of decomposition:
inefficient (e.g. determination of keys)
produces more relations than necessary

Synthesis
Given relation R and set of FDs DEP
Find a canonical set MIN of FDs which "covers" DEP and is minimal.
Construct normalized Relations R_k from MIN
with $\cup \Sigma(R_k) = \Sigma(R)$

Finding a canonical set of FDs

Given a **set of FDs DEP** and a relational schema R
- Find a minimal set **MIN** such that **DEP** $\subseteq$ **MIN$^+$**
- Find a relational schema in 3NF, from which R can be losslessly reconstructed

MIN is called minimal cover of DEP

Definitions

$X \rightarrow Y \in$ **MIN$^+$** :⇔ $X \rightarrow Y$ can be proven from the FDs $\in$ MIN

e.g. $\{a \rightarrow b, b \rightarrow c\}+ = \{a \rightarrow b, b \rightarrow c, a \rightarrow c\}$

MIN is **minimal** :⇔ for every FD f $(MIN \setminus f)^+ \neq MIN^+$

Finding a minimal cover (1)

Example:

$\{ab \rightarrow d, b \rightarrow c, dc \rightarrow e\}+ = \{ab \rightarrow d, b \rightarrow c, a \rightarrow dc, dc \rightarrow e, ab \rightarrow e\}\}$

Important first step:

Given a set of attributes X, determine all attributes (closure of X) which can be functionally determined by X?

Def.: **Closure of attribute set X** with respect to the set DEP of FDs is the largest set Y of attributes such that $X \rightarrow Y \in DEP^+$

Closure of attribute set X

```
I = 0; X[0] = X;        /* integer I, attr. set X[0] */
REPEAT                  /* loop to find larger X[I] */
 I = I + 1;      /* new I */
 X[I] = X[I-1];         /* initialize new X[I] */
 FOR ALL Z->W in DEP /* loop on all FDs Z ->W in DEP*/
  IF Z ⊆X[I]            /* if Z contained in X[I] */
  THEN X[I] = X[I]∪ W; /* add attributes in W to X[I]*/
 END FOR                /* end loop on FDs */
UNTIL X[I] = X[I-1]; /* loop till no new attributes*/
RETURN X = X[I] ;    /* return closure of X */
```

Rule used: X -> YZ and Z -> W then X -> YZW
Proof?

Finding a canonical set

Algorithm for determining a <u>minimal cover</u> in polynomial time

Step 1: Normalization

Replace each FD $X \rightarrow Y$ of DEP in which Y contains more than one attribute, by FDs with one attribute on the right hand side

Example:
DEP = {ab -> cd, a -> e} → {ab -> c, ab -> d, a -> e}

Finding a canonical set (2)

Step 2: Remove redundant FDs

f is redundant, if $(DEP \setminus \{f\})^+ = DEP^+$
For each $f = X \rightarrow Y \in DEP$:
if $Y \subseteq (X+)$ using only FDs $\in DEP \setminus\{f\}$
then f is redundant else not redundant

Example: {b → d, d → e, ef → a, c → f, bc → a}
bc+ = {b,d,e,f,a} ⊇ {a}
⇒ FD f= bc -> a is redundant

Explicit derivation not using f:
b -> d, d -> e ⇒ b -> e, c -> f ⇒ bc -> ef , ef -> a ⇒ bc -> a

Finding a canonical set (3)

Step 3: Minimize left hand side

For of each FD $f = X \rightarrow Y \in DEP$, $a \in X$
$DEP' = \{DEP \setminus f\} \cup \{X \setminus \{a\} \rightarrow Y\}$
if $\{DEP\}+ = (DEP')^+$
then replace DEP by DEP'
end_for
If a FD f has been minimized repeat step 2

Example:
$\{bcd \rightarrow a, c \rightarrow e, e \rightarrow b\}^+ = \{cd \rightarrow a, c \rightarrow e, e \rightarrow b\}^+$

Finding a canonical set (4)

Step 4: Unify left hand side

Applying the union rule

" $X \to Y \wedge X \to Z \Rightarrow X \to YZ$ "

Example: {cd -> a, cd -> e, d -> f}

unified: {cd -> ae, d -> f}

The remaining set of Functional Dependencies is the minimal cover of the original set DEP of FDs

Synthesis algorithm

Normalization problem:

Given a relation R in 1NF and a set DEP of FD Find a lossless, dependency preserving decomposition $R_1,\ldots,R_k$, all of them in 3NF

Synthesis Algorithm

```
Find minimal cover MIN of DEP;
For all X -> Y in MIN define a relation
   RX  with schema Σ(RX) = X ∪ Y
Assign all FDs X' -> Y' with X' ∪ Y' ⊆  Σ(RX) to RX
If none of the synthesized relations RX contains a
   candidate key of R
then introduce a relation Rkey which contains a
   candidate key of R
Remove relations RY where:  Σ(RY) ⊆ Σ(RX)
```


Normal Forms: Critical review

Should relations be always normalized ?

Yes : **makes invariant checking easy,** and no „update anomalies"

No: **Why** should we **normalize if** there are **no updates** ?

Example: Customer(cu_Id, name, fname, zipCode, city, street, no)
No reason to normalize if only one address per customer and updates are infrequent

Consider **cost of joins / updates**

- How expensive are selects which need joins because of normalization?
- Updates which cause anomalies?

ER modeling and Normal Forms

ER and Normal Forms:

Two different mechanisms to design a database scheme

ER more intuitive,

NF uses algorithms

ER-models often already in 3NF

Use **normalization** as a **complementary design method**

- Set up ER model
- Transform to relations
- Normalize each non normalized relation if the tradeoff of join processing and updating redundant data suggests to do so