Higher Normal Forms

http://www.dcs.fmph.uniba.sk/~plachetk

/TEACHING/DB2

http://www.dcs.fmph.uniba.sk/~sturc/databazy/rldb

Tomáš Plachetka, Ján Šturc

Faculty of mathematics, physics and informatics,
Comenius University, Bratislava

Summer 2025–2026

Multi-valued dependency

Definition: A **multi-valued dependency (MVD) X $\rightarrow\rightarrow$ Y** holds in relation r, if
r(**X**, **Y1**, **Z1**) $\wedge$ r(**X**, **Y2**, **Z2**) $\Rightarrow$ r(**X**, **Y1**, **Z2**) $\wedge$ r(**X**, **Y2**, **Z1**)

Example:

emp(Emp_name,	Proj_name,	Boss_name)
joe	x	fred
joe	y	lisa
joe	x	lisa
joe	y	fred

Emp_name $\rightarrow\rightarrow$ Proj_name, Emp_name $\rightarrow\rightarrow$ Boss_name both hold.
Emp_name $\rightarrow\rightarrow$ Emp_name also holds, but it is a trivial MVD

Definition: An MVD **X** $\rightarrow\rightarrow$ **Y** is called trivial, when either **Y** is subset of **X** or r = **X** $\cup$ **Y**

Multi-valued dependency: inference rules

IR1 (**reflexive rule for FDs**): $X \supseteq Y \Rightarrow X \rightarrow Y$

IR2 (**augmentation rule for FDs**): $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$

IR3 (**transitive rule for FDs**): $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

IR4 (**complementation rule for MVDs**):

$X \twoheadrightarrow Y \Rightarrow X \twoheadrightarrow (R - (X \cup Y))$

IR5 (**augmentation rule for MVDs**): $X \twoheadrightarrow Y \wedge W \supseteq Z \Rightarrow WX \twoheadrightarrow YZ$

IR6 (**transitive rule for MVDs**): $X \twoheadrightarrow Y \wedge Y \twoheadrightarrow Z \Rightarrow X \twoheadrightarrow (Z - Y)$

IR7 (**replication rule for FD to MVD**): $X \rightarrow Y \Rightarrow X \twoheadrightarrow Y$

IR8 (**coalescence rule for FDs and MVDs**): $X \twoheadrightarrow Y \wedge$ (exists W such that $W \cap Y = \varnothing \wedge W \rightarrow Z \wedge Y \supseteq Z) \Rightarrow X \rightarrow Z$

Sometimes the following notation is used (sharing the same left-hand side, similarly as with FDs): X $\twoheadrightarrow$ Y, Z stands for X $\twoheadrightarrow$ Y, X $\twoheadrightarrow$ Z. But Y and Z are now sets of attributes (i.e. the comma is important)!

Multi-valued dependency: 4NF

Definition: A relation schema *r* is in **4NF** with respect to a set of dependencies *F* (that includes functional dependencies and multivalued dependencies) if for every *nontrivial* multivalued dependency $\boldsymbol{X} \rightarrow\rightarrow \boldsymbol{Y}$ in F^+, *X* is a superkey in r

F^+ is a **closure** of F with respect to inference rules IR1–IR8. These inference rules are sound and complete (the soundness results from the definitions of FD and MVD, the completeness is harder to prove)

Apparently, emp(Emp_name, Proj_name, Boss_name) should be decomposed into
project(Emp_name, Proj_name), boss(Emp_name, Boss_name)

Multi-valued dependency: 4NF

Definition: Relation schemas r_1 and r_2 form a **lossless (non-additive) join decomposition** of r with respect to a set F of functional ***and*** **multivalued dependencies**, if

$(r_1 \cap r_2) \rightarrow\rightarrow (r_1 - r_2)$, or

$(r_1 \cap r_2) \rightarrow\rightarrow (r_2 - r_1)$

Multi-valued dependency: 4NF

4NF decomposition algorithm

naive, but with lossless non-additive join guarantee

Input: A universal relation r and a set of functional and multivalued dependencies F

```
decompose(r, F)
{
        D := {r};
        while (a relation schema s exists in D which is not in 4NF)
        {
                find a non-trivial MVD X ↠ Y in s which violates 4NF;
                replace s in D by relation schemas (s - Y) and (X ∪ Y);
        }
}
```

Join dependency

Definition. Let $\mathbf{R}$, $\mathbf{R}_1$, …, $\mathbf{R}_N$ denote sets of attributes. A **join dependency** is a constraint of the form $\mathbf{R} = \mathbf{R}_1 \bowtie \ldots \bowtie \mathbf{R}_N$ which is satisfied if for each valid population of the relation r over attributes R holds
$r = \Pi_{R_1}(r) \bowtie \ldots \bowtie \Pi_{R_N}(r)$

Note that this is (indeed) a definition of lossless decomposition, i.e. global consistency. However, a **join dependency is a constraint just over sets of attributes, which does not depend on the actual decomposition of the database schema**

Example: (SSN, Name, ChildSSN) = (SSN, Name) ⋈ (SSN, ChildSSN)
is implied by functional dependency SSN → Name. In fact, each functional dependency implies some join dependency

We are particularly interested in **binary join dependencies** (as it turns out, these are MVDs)**, which are of the form**
$\mathbf{R} = \mathbf{R}_1 \bowtie \mathbf{R}_2$
Again, recall that $\mathbf{R}$, $\mathbf{R}_1$, $\mathbf{R}_2$ are sets of attributes which are not necessarily actual relations of the database

Join dependency: 5NF

An MVD is a special case of a join dependency: $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ in r can be equivalently expressed as $\mathbf{R} = (\mathbf{XY}) \bowtie (\mathbf{X} \cup (\mathbf{R} - \mathbf{Y}))$

Definition. A relation schema r is in **fifth normal form (5NF**, or **Project-Join Normal Form, PJNF)** with respect to a set F of functional, multivalued, and join dependencies if
for every nontrivial join dependency JD(R_1, R_2, ..., R_n) in F^+ (i.e., implied by F) holds that every R_i is a superkey of r

Note that 5NF takes into account not only FDs and MVDs, but also all kinds of JDs

No sound and complete set of inference rules is known for JDs (some sound rules have been published, though)

Inclusion dependency

Definition: An inclusion dependency *r*.***A*** < *s*.***B*** holds if for any valid population of r and any valid population of s holds
$\Pi_{\mathbf{A}}(r) \supseteq \Pi_{\mathbf{B}}(s)$,
where **A** and **B** are sets of attributes

Informally: inclusion dependency states that relation r contains a set of attributes **A** such that when **A** attains a value in r, then the same value of **A** must exist in relation s (stored in attributes B)

Alternatively, **inclusion dependency** from relation r to relation s exists if
$(\exists \mathbf{X}\ \exists \mathbf{Y}\ r(\mathbf{X}, \mathbf{Y})) \Rightarrow (\exists \mathbf{Z}\ s(\mathbf{X}, \mathbf{Z}))$

Inclusion dependency

Example (a university database):
students(Student_ID, Name),
grades(Student_ID, Course_ID, Grade)
When a Student_ID appears in the relation grades, then the same Student_ID must exists in the relation students (as grades are assigned only to enrolled students)

In this example, Student_ID in grades is a foreign key for the relation students, i.e. it is a superkey in the relation students (and also vice versa). **Foreign key is a special case of an inclusion dependency**. Inclusion dependencies cover and generalise the concept of weak entity sets in ER and UML diagrams (presented in the winter semester)

Inclusion dependency

Example (an extended university database):
students(Student_ID, Name),
graduates(Student_ID, Name)
grades(Student_ID, Course_ID, Grade)

A graduated student must have received some grades, so there is an inclusion dependency from relation graduates to relation grades (but not in the opposite direction). But in this case, Student_ID is **not a foreign key** (a superkey) in the relation grades, because Student_ID $\rightarrow$ Course_ID, Grade does not hold

Organisational (business) rules often dictate inclusion dependencies and other **integrity constraints** which must hold **for any valid population of a database**. It is desirable that the database system guards the database against violation of any known constraint. Still, many constraints either cannot be expressed in SQL or they can only be expressed in SQL in an awkward way (subsequently, their guarding increases the cost of updates)

Inclusion dependency: inference rules

IR1 (reflexivity)
r.**X** < r.**X**

IR2 (attribute correspondence)
if r.**X** < s.**Y**,
where **X** = {A_1, A_2 ,..., A_n} and **Y** = {B_1, B_2, ..., B_n} and
A_i corresponds to B_i,
then r.A_i < s.B_i for $1 \leq i \leq n$

IR3 (transitivity)
if r.**X** < s.**Y** and s.**Y** < t.**Z**, then r.**X** < r.**Z**

More on 4NF decomposition

Naïve approach to 4NF decomposition may produce bad database design

Example (Kifer, Bernstein, Lewis):
contracts(Buyer, Vendor, Product, Currency)
Business rule 1: If a company accepts several currencies, then each contract is described in each currency
Buyer, Vendor $\rightarrow\rightarrow$ Product, Currency
or, equivalently,
contracts = (Buyer, Vendor, Product) $\bowtie$ (Buyer, Vendor, Currency)

Business rule 2: If two vendors supply a product, they accept some currency; and if a buyer has a contract with one of these vendors, then it has a contract with the other vendor as well
Product, Currency $\rightarrow\rightarrow$ Buyer, Vendor

More on 4NF decomposition

contracts(Buyer, Vendor, Product, Currency)
r1: Buyer, Vendor $\rightarrow\rightarrow$ Product, Currency
r2: Product, Currency $\rightarrow\rightarrow$ Buyer, Vendor

Decomposition using r1, (Buyer, Vendor, Product), (Buyer, Vendor, Currency) breaks r2, i.e. the second MVD is lost. And vice versa

Breaking an MVD by 4NF decomposition is much worse than breaking an FD by BCNF decomposition! The reason is that although e.g. **(Buyer, Vendor, Product), (Buyer, Vendor, Currency) is a losless 4NF decomposition**, it can still contain a lot of **redundancy** if some MVD is broken:

Buyer	Vendor	Product
B1	V1	P
B2	V2	P
B1	V2	P
B2	V1	P

Buyer	Vendor	Currency
B1	V1	C
B2	V2	C
B1	V2	C
B2	V1	C

It is better not to insist on 4NF in this case

More on 4NF decomposition

Example (Kifer, Bernstein, Lewis):
dictionary(Latin, Hungarian, German, Slovak)
Business rule: If the dictionary provides translation from any language to any other (one word can have even more translations in other languages)
Latin $\rightarrow\!\!\rightarrow$ Hungarian, German, Slovak
Hungarian $\rightarrow\!\!\rightarrow$ Latin, German, Slovak
German $\rightarrow\!\!\rightarrow$ Latin, Hungarian, Slovak
Slovak $\rightarrow\!\!\rightarrow$ Latin, Hungarian, German

Again, there is no lossless 4NF decomposition which preserves all MVDs above. A workaround is to add a concept to the dictionary, i.e. to promote the dictionary to a multilingual thesaurus:
thesaurus(Concept, Descr, Latin, Hungarian, German, Slovak)
Latin $\rightarrow$ Concept, Hungarian $\rightarrow$ Concept,
German $\rightarrow$ Concept, Slovak $\rightarrow$ Concept,
Concept $\rightarrow$ Descr,
Concept $\rightarrow\!\!\rightarrow$ Latin, Hungarian, German, Slovak

(What about homonyms? Oh well…)

More on 4NF decomposition

thesaurus(Concept, Descr, Latin, Hungarian, German, Slovak)
Latin $\rightarrow$ Concept, Hungarian $\rightarrow$ Concept,
German $\rightarrow$ Concept, Slovak $\rightarrow$ Concept,
Concept $\rightarrow$ Descr,
Concept $\rightarrow\!\!\rightarrow$ Latin, Hungarian, German, Slovak

Every FD is MVD as well. So let the decomposition begin with
Latin $\rightarrow$ Concept and continue likewise. Resulting decomposition:
(Latin, Concept), (Latin, Hungarian), (Latin, German),
(Latin, Slovak), (Latin, Descr) is clearly 4NF. But it is too leaned towards Latin, which seems unnatural (well, perhaps not to a linguist)

More natural is to derive
Concept, Descr $\rightarrow\!\!\rightarrow$ Latin, Hungarian, German, Slovak
which yields lossless 4NF decomposition:
(Concept, Descr, Latin), (Concept, Descr, Hungarian),
(Concept, Descr, German), (Concept, Descr, Slovak)

More on 4NF decomposition

thesaurus(Concept, Descr, Latin, Hungarian, German, Slovak)
Latin $\rightarrow$ Concept, Hungarian $\rightarrow$ Concept,
German $\rightarrow$ Concept, Slovak $\rightarrow$ Concept,
Concept $\rightarrow$ Descr,
Concept $\rightarrow\!\!\rightarrow$ Latin, Hungarian, German, Slovak

We have got:
(Concept, Descr, Latin), (Concept, Descr, Hungarian),
(Concept, Descr, German), (Concept, Descr, Slovak)

Now we can get back to FD Concept $\rightarrow$ Descr and decompose even more:
(Concept, Descr),
(Concept, Latin), (Concept, Hungarian),
(Concept, German), (Concept, Slovak)

This is the decomposition we were after!

More on 4NF decomposition

4NF decomposition is a problem difficult to tackle formally

[Beeri and Kifer, 1986] give a universal recipe, which does not depend on a particular schema:

1. Find splitting left-hand side of MVDs
2. Find intersection anomalies
3. Apply 5-step “design strategy”

More on 4NF decomposition: 1.splitting lefthand anomaly

Definition. We say that $X \rightarrow\rightarrow V, W$ **splits left-hand side** of $Y \rightarrow\rightarrow K, L$ when both $Y \cap V$ and $Y \cap W$ are both non-empty

If there are two MVDs in the schema where one splits left-hand side of the other, then the schema exhibits **left-hand anomaly**

A schema which suffers from a left-hand anomaly is probably wrong (the constraints are too strong)

For instance, in ***contracts example***, the MVDs (business rules) r1, r2 split each other's left-hand side. It indicates that something went wrong already with the schema itself. Indeed, r1 and r2 should be rather replaced by a single business rule which is the following JD:

contracts = (Buyer, Product) ⋈ (Vendor, Product) ⋈
(Buyer, Currency) ⋈ (Vendor, Currency)

It states that buyers buy some products, vendors sell some products, buyers accept some currencies, vendors accept some currencies

More on 4NF decomposition: 2.intersection anomaly

Definition. We say that a schema suffers from **intersection anomaly** when a pair of MVDs $X \to\to Z$, $Y \to\to Z$ holds, but MVD $X \cap Y \to\to Z$ does not hold

If a schema suffers from intersection anomaly, then the schema itself is probably wrong (incomplete)

For instance, in ***dictionary example***, Latin $\to\to$ Slovak and Hungarian $\to\to$ Slovak hold, but $\varnothing \to\to$ Slovak does not hold. This is an intersection anomaly

A solution is usually adding a new attribute or attributes

More on 4NF decomposition: 3.design strategy

We now assume no splitting left-hand side anomalies. Then a dependency-preserving, lossless join decomposition can be produced as follows:

a) Compute attribute closure, X^+ of every MVD $X \rightarrow\rightarrow Y$. Then replace the MVD $X \rightarrow\rightarrow Y$ with $X^+ \rightarrow\rightarrow Y$

b) Find minimal cover of resulting MVDs from step a)

c) Eliminate intersection anomalies by adding new attributes. (This can be done automatically! See [Beeri&Kifer, 1986].)

d) Apply naïve 4NF decomposition algorithm using MVDs only

e) Apply BCNF design algorithm using FDs only

See the ***multilingual thesaurus*** example

A “solution” to normalisation: DKNF

No matter how high a normal form is, a normalised database can still exhibit some redundancy, because not all dependencies are covered (or known). The domain-key normal form (DKNF) is an ultimate normal form: DKNF⇒5NF⇒4NF⇒BCNF⇒3NF

Definition [Fagin, 1981]: A relation is in domain-key normal form (DKNF) if every constraint on the relation is a logical consequence of keys and domains

What is a domain? A set of values which an attribute can attain
What is a constraint? In this context, it is an arbitrary rule precise enough that one can decide whether it is fulfilled (true) or not

DKNF

More precisely, DKNF allows only two kinds of constraints:

1. A **domain constraint**, which expresses a set of values which an attribute can attain, e.g. color$\in${red, green, blue}
2. A **key constraint**, which expresses that a set of attributes is a superkey of the relation in question. Hence, a key constraint for a relation R states that the functional dependency K$\rightarrow$R holds, where K$\subseteq$R

Definition (rephrased): Let **D** be a set of domain constraints, let **K** be a set of key constraints for a relation R. Let **G** be a set of general constraints (all “business rules” which hold in R). Then R is in DKNF, if (**D** $\cup$ **K**) $\Rightarrow$ **G**

DKNF

The constraints in DKNF are local, i.e. bound to a single relation. There is no concept of e.g. referential integrity, although sometimes such constraints can be expressed in terms of domains and keys

Unfortunately, there is no universal recipe which constructs a DKNF decomposition (in general, it cannot be even decided whether a DKNF decomposition exists). Once you have found one, bravo, there is no redundancy. But in the meantime, you are stuck

There is no universal recipe which verifies whether a given decomposition is in DKNF