Relational Normal Forms

- After completing this chapter, you should be able to
  - ▷ work with **functional dependencies** (FDs),
    define them, detect them in DB schemas, decide implication, determine keys
  - ▷ explain insert, update, and delete **anomalies,**
  - ▷ explain **BCNF,** test a given relation for BCNF, and transform a relation into BCNF,
  - ▷ detect and correct violations of **4NF,**
  - ▷ detect **normal form violations** on the level of ER,
  - ▷ decide if and how to **denormalize** a DB schema.

Relational Normal Forms

Overview

1. Functional Dependencies (FDs)
2. Anomalies, FD-based Normal Forms
3. Multivalued Dependencies (MVDs) and 4NF
4. Normal Forms and ER Design
5. Denormalization

Introduction (1)

- **Relational database design theory** is mainly based on a class of constraints called **Functional Dependencies (FDs).**

  FDs are a **generalization of keys.**

- This theory defines when a relation is in **normal form** (*e.g.*, in Third Normal Form or 3NF) for a given set of FDs.

- It is usually a sign of **bad DB design** if a schema contains relations that **violate the normal form** requirements.

  There are exceptions and tradeoffs, however.

Introduction (2)

- If a normal form is violated, data is stored **redundantly,** and information about different concepts is **intermixed.**

Redundant data storage

COURSES

CRN	TITLE	INAME	PHONE
22268	Databases I	Grust	75477
42232	Query Compilers	Grust	75477
31822	Functional Programming	Klaeren	78955

▷ The phone number for each instructor is stored multiple times (once for every course the instructor is reading).

Introduction (3)

- Today, the **Third Normal Form (3NF)** is considered the standard relational normal form used in practice (and education).
- **Boyce-Codd Normal Form (BCNF)** is a bit more restrictive, easier to define, and will be a better match for our intuition of good DB design.

  In rare circumstances, a relation might not have an equivalent BCNF form while preserving all its FDs (this is always guaranteed for 3NF).
- In a nutshell, BNCF requires that **all FDs are already enforced (represented) by keys.**

Introduction (4)

- **Normalization algorithms** can construct good relation schemas from a set of attributes and a set of FDs alone.
- In practice, relations are derived from ER models and normalization is used as an additional check (for BCNF) only.
- When an ER model is well designed, the resulting derived relational tables will **automatically be in in BCNF** (even 4NF).
- Awareness of normal forms can help to detect design errors already in the ER design phase.

First Normal Form (1)

First Normal Form (1NF)

First Normal Form (1NF) requires that all **table entries are atomic** (*not* lists, sets, records, relations).

- The relational model is already defined this way. All further normal forms assume that tables are in 1NF.
- A few modern DBMS allow table entries to be non-atomic (structured). Such systems are commonly referred to as **NF^2 DBMS.**

  (NF^2 = NFNF = Non-First Normal Form).

First Normal Form (2)

NF^2 relation ($\neg$ 1NF):

COURSES

CRN	TITLE	TAUGHT_BY	STUDENTS
22332	Databases I	Grust	FNAME: John, Ann; LNAME: Smith, Miller
31822	Functional Programming	Klaeren	FNAME: Ann; LNAME: Miller

- Note: A discussion of 1NF does not really belong here (1NF is not based on functional dependencies at all).

First Normal Form (3)

- It is **not** a violation of 1NF if
  1. a table entry contains values of a domain which exhibit an internal structure (*e.g.*, all `CRN` attented by a student collected in a `CHAR(100)` attribute separated by spaces),
  2. related information is represented by several columns for a single row (*e.g.*, columns `DEGREE1`, `DEGREE2`, `DEGREE3` to represent the degrees a person has obtained).

These are signs of bad design anyway.

Why?

Notes

Functional Dependencies (1)

- An instance of a **functional dependency (FD)** in our latest example tables is

$$\texttt{INAME} \rightarrow \texttt{PHONE} \ .$$

  ▷ Whenever two rows of a relation agree in the instructor name `INAME`, they **must** also agree in the `PHONE` column values:

COURSES

CRN	TITLE	INAME	PHONE
22268	Databases I	Grust	75477
42232	Query Compilers	Grust	75477
31822	Functional Programming	Klaeren	78955

Functional Dependencies (2)

- Intuitively, the reason for the validity of $\texttt{INAME} \rightarrow \texttt{PHONE}$ is that the contact phone number **only depends on the instructor**, not on other course data.
- This FD is read as

  "`INAME` {functionally, uniquely} determines `PHONE`."

- Also: *"`INAME` is a determinant for `PHONE`."*

  Saying that A is a **determinant** for B is actually stronger than the FD $A \rightarrow B$ since A must be minimal and $A \neq B$ (see below).

- A determinant is like a **partial key:** it uniquely determines some attributes, but not all in general.

Functional Dependencies (3)

- A **key** uniquely determines **all** attributes of its relation. In the example, the FDs `CRN` $\rightarrow$ `TITLE`, `CRN` $\rightarrow$ `INAME`, `CRN` $\rightarrow$ `PHONE` hold, too.

  There will be never be two distinct rows with the same `CRN`, so the FD condition is trivially satisfied.

- The FD `INAME` $\rightarrow$ `TITLE` is **not** satisfied, for example. At least two rows agreeing on `INAME` disagree on `TITLE`:

COURSES

CRN	TITLE	INAME	PHONE
22268	Databases I	Grust	75477
42232	Query Compilers	Grust	75477
31822	Functional Programming	Klaeren	78955

Functional Dependencies (4)

- In general, an FD takes the form

$$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \ .$$

- Sequence and multiplicity of attributes in an FD are unimportant, since both sides formally are sets of attributes:

$$\{A_1, \ldots, A_n\} \rightarrow \{B_1, \ldots, B_m\} \ .$$

- In discussing FDs, the focus is on a **single relation** R. All A_i, B_j are attributes of the same relation R.

Functional Dependencies (5)

Functional Dependency

The **functional dependency (FD)**

$$A_1, \ldots A_n \rightarrow B_1, \ldots B_m$$

holds for a relation R in a database state I if and only if for all tuples $t, u \in I(R)$:

$$\begin{array}{rl} & t.A_1 = u.A_1 \wedge \cdots \wedge t.A_n = u.A_n \\ \Rightarrow & t.B_1 = u.B_1 \wedge \cdots \wedge t.B_m = u.B_m \ . \end{array}$$

Functional Dependencies (6)

- An FD with m attributes on the right-hand side (rhs)

  $$A_1, \ldots A_n \rightarrow B_1, \ldots B_m$$

  is **equivalent** to the m FDs:

  $$\begin{array}{ccc} A_1, \ldots, A_n & \rightarrow & B_1 \\ \vdots & & \vdots \\ A_1, \ldots, A_n & \rightarrow & B_m \end{array}$$

- Thus, in the following discussion it suffices to consider FDs with a single column name on the rhs.

FDs are Constraints (1)

- For the DB design process, the only interesting FDs are those that **hold for all possible database states.**
  - ▷ FDs are **constraints** (like keys).

Non-interesting FDs

COURSES

CRN	TITLE	INAME	PHONE
22268	Databases I	Grust	75477
42232	Query Compilers	Grust	75477
31822	Functional Programming	Klaeren	78955

In this example state, the FD TITLE $\rightarrow$ CRN holds. But this is probably not true in general (it is a task of DB design to verify whether this is mere coincidence).

FDs are Constraints (2)

- There are tools for analyzing example database states for possible FDs, and then asking the DB designer whether these FDs hold in general.
- If an FD (or any other constraint) does not hold in the example state, it certainly cannot hold in general.
- Database design based on normal forms requires the DB designer to **collect all FDs that generally hold in the mini-world.** This is a design task that cannot be automated.

  Actually, it is sufficient to collect a representative subset of FDs which **implies** all valid FDs (see below).

FDs vs. Keys (1)

- **FDs are a generalization of keys:**

$$A_1, \ldots, A_n \text{ is a key of relation } R(A_1, \ldots, A_n, B_1, \ldots, B_m)$$
$$\Leftrightarrow$$
$$A_1, \ldots, A_n \rightarrow B_1, \ldots B_m \text{ holds.}$$

- Given the FDs for a relation R, one can **derive a key for R,** *i.e.*, a set of attributes $A_1, \ldots, A_n$ that functionally determines the other attributes of R (see below).

FDs vs. Keys (2)

- Conversely, FDs can be explained with keys. FDs are "keys" for some columns, *e.g.*, the FD `INAME` $\rightarrow$ `PHONE` means that `INAME` is key of

$$\pi_{\texttt{INAME,PHONE}}(\texttt{COURSES}) \quad \equiv$$

INAME	PHONE
Grust	75477
Klaeren	78955

An FD $A \rightarrow B$ holds in the original relation R (here: `COURSES`) if the projection result $\pi_{A,B}(R)$ contains no row with duplicate A values.

FDs are *partial keys.* It is the **goal of DB normalization to turn all FDs into real keys[a].**

[a] The DBMS is then able to enforce the FDs for the user.

FDs Describe Functions (1)

- Any relation R describes a partial **function** (given in terms of a lookup table):
  - ▷ function input are the values of the key attributes of R,
  - ▷ function output are the values of the remaining attributes.
- In the example table `COURSES`, **another function** is represented in the very same table: this function maps instructor names to phone numbers.

  This "embedded" function is partial, too: its domain is the active domain of the `INAME` attribute.

FDs Describe Functions (2)

- FDs indicate the presence of a function, but sometimes we even know **how** to compute the indicated function.
  - ▷ Example: `BIRTHDAY` and `AGE` are represented in the same table. Then, of course, `BIRTHDAY` $\rightarrow$ `AGE` holds.
  - ▷ We even know how to compute `AGE` given `BIRTHDAY`. Clearly, `AGE` is redundant.
  - ▷ In such cases, database normalization theory will point out the potential redundancy but will not be able to actually remove it.

Example (1)

- The following tables holds information about "DB textbook classics" and their respective authors:

BOOKS

AUTHOR	NO	TITLE	PUBLISHER	ISBN
Elmasri	1	Fund. of DBS	Addison-W.	0805317554
Navathe	2	Fund. of DBS	Addison-W.	0805317554
Silberschatz	1	DBS Concepts	Mc-Graw H.	0471365084
Korth	2	DBS Concepts	Mc-Graw H.	0471365084
Sudarshan	3	DBS Concepts	Mc-Graw H.	0471365084

A book may have multiple authors, one author per row. Attribute NO is used to track authorship sequence (which may not be alphabetical).

Example (2)

- The ISBN uniquely identifies a single book. Thus, the following FD holds

$$\texttt{ISBN} \rightarrow \texttt{TITLE}, \texttt{PUBLISHER} \ .$$

  ▷ Equivalently, we could use two FDs:

$$\texttt{ISBN} \rightarrow \texttt{TITLE}$$
$$\texttt{ISBN} \rightarrow \texttt{PUBLISHER} \ .$$

- Since a book may have several authors, the following FD does **not** hold:

$$\texttt{ISBN} \rightarrow \texttt{AUTHOR}$$

Example (3)

- One author can write many books, thus the following FD may **not** be assumed although it happens to hold in the given example DB state:

$$\texttt{AUTHOR} \rightarrow \texttt{TITLE} \qquad \text{(not true in general)}$$

- It is possible that there are books with the same title but different authors and different publishers. So, TITLE determines no other attribute.

Example (4)

- For every book (we now know: books are identified by ISBN), there can only be one first (second, third, . . .) author:

$$\texttt{ISBN}, \texttt{NO} \rightarrow \texttt{AUTHOR} \ .$$

- At first glance, the author of any given book is also uniquely assigned a position in the authorship sequence:

$$\texttt{ISBN}, \texttt{AUTHOR} \rightarrow \texttt{NO} \qquad \text{(questionable)}$$

  However, this is violated by an author list like *Smith & Smith*.

Example (5)

- We might also be tempted to assume the FD

  $\texttt{PUBLISHER}, \texttt{TITLE}, \texttt{NO} \rightarrow \texttt{AUTHOR}$  (questionable)

  If, in a new book edition, the authorship sequence changes, we are in trouble.

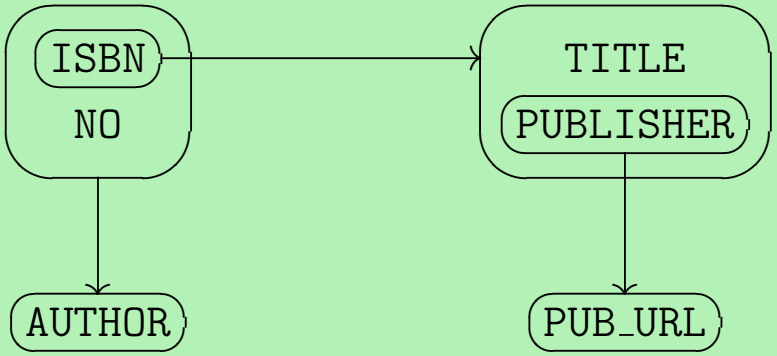
- During DB design, only unquestionable conditions should be used as FDs.

  DB normalization alters the **table structure** depending on the specified FDs. If it turns out later that an FD indeed does not hold, it is *not* sufficient to simply remove a constraint (*e.g.*, a key) from the schema with an `ALTER TABLE` SQL statement. Instead, tables will be created or deleted, data will be copied, and application programs may have to be changed.

Example (6)

- A set of FDs can be visualized as a *hypergraph*[7] (`PUB_URL` has been added to make the example more interesting):

FD hypergraph



[7] In a hypergraph, edges connect sets of nodes.

FD Quiz

Homework grades DB

① Which FDs should hold for this table in general?

② Identify an FD which holds in this state but not in general?

HOMEWORK_RESULTS

STUD_ID	FIRST	LAST	EX_NO	POINTS	MAX_POINTS
100	Andrew	Smith	1	9	10
101	Dave	Jones	1	8	10
102	Maria	Brown	1	10	10
101	Dave	Jones	2	11	12
102	Maria	Brown	2	10	12

Notes

Implication of FDs (1)

- Note: the FD CRN $\rightarrow$ PHONE is a consequence since we already knew CRN $\rightarrow$ INAME and INAME $\rightarrow$ PHONE.

  Whenever $A \rightarrow B$ and $B \rightarrow$C hold, $A \rightarrow C$ is automatically satisfied. (If f, g are functions, then $g \circ f$ is a function.)

- PHONE $\rightarrow$ PHONE holds, but is not interesting.

  FDs of the form $A \rightarrow A$ always hold (for every DB state).

Implication of FDs

A set of FDs $\{\alpha_1 \rightarrow \beta_1, \ldots, \alpha_n \rightarrow \beta_n\}$ **implies** an FD $\alpha \rightarrow \beta$ if and only if every DB state which satisfies $\alpha_i \rightarrow \beta_i, 1 \leqslant i \leqslant n$, also satisfies $\alpha \rightarrow \beta$.

Implication of FDs (2)

- The DB designer is normally not interested in all FDs which hold, but only in a **representative FD set** that implies all other FDs.

Armstrong Axioms

- **Reflexivity:**
  If $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$.
- **Augmentation:**
  If $\alpha \rightarrow \beta$, then $\alpha \cup \gamma \rightarrow \beta \cup \gamma$.
- **Transitivity:**
  If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$.

Implication of FDs (3)

- However, a simpler way to **check whether** $a \to \beta$ **is implied by a given FD set** is to compute the **cover** α^+ of α and then to check if $\beta \subseteq \alpha^+$.

Cover

The **cover** α^+ of a set of attributes α is the set of all attributes B that are uniquely determined by the attributes α (with respect to a given FD set $\mathcal{F}$):

$$\alpha^+ := \{B \mid \mathcal{F} \text{ implies } \alpha \to B\}$$

If necessary, write $\alpha^+_{\mathcal{F}}$ to make the dependence on $\mathcal{F}$ explicit.

- A set of FDs $\mathcal{F}$ implies $\alpha \to \beta$ if and only if $\beta \subseteq \alpha^+_{\mathcal{F}}$.

Implication of FDs (4)

Cover computation

Input: α (set of attributes)
$\alpha_1 \to \beta_1, \ldots, \alpha_n \to \beta_n$ (set of FDs $\mathcal{F}$)
Output: α^+ (set of attributes)

```
x ← α;
while x did change do
  foreach given FD α_i → β_i do
    if α_i ⊆ x then
      x ← x ∪ β_i;
    fi
  od
od
return x;
```

Implication of FDs (5)

Compute $\{\texttt{ISBN}\}^+$.

- Consider the following FDs:

$$\begin{aligned} \texttt{ISBN} &\rightarrow \texttt{TITLE}, \texttt{PUBLISHER} \\ \texttt{ISBN}, \texttt{NO} &\rightarrow \texttt{AUTHOR} \\ \texttt{PUBLISHER} &\rightarrow \texttt{PUB\_URL} \end{aligned}$$

① We start with $x = \{\texttt{ISBN}\}$.

② The FD $\texttt{ISBN} \rightarrow \texttt{TITLE}, \texttt{PUBLISHER}$ has a lhs which is completely contained in the current attribute set x.

③ Extend x by the FD's rhs attribute set, giving $x = \{\texttt{ISBN}, \texttt{TITLE}, \texttt{PUBLISHER}\}$.

Implication of FDs (6)

Compute $\{\texttt{ISBN}\}^+$ (cont'd).

④ Now the FD $\texttt{PUBLISHER} \rightarrow \texttt{PUB\_URL}$ becomes applicable.

⑤ Add rhs attribute set of the FD to the current attribute set x, giving $x = \{\texttt{ISBN}, \texttt{TITLE}, \texttt{PUBLISHER}, \texttt{PUB\_URL}\}$.

⑥ The FD $\texttt{ISBN}, \texttt{NO} \rightarrow \texttt{AUTHOR}$ is still not applicable (attribute $\texttt{NO} \notin x$).

⑦ No further way to extend set x, the algorithm returns

$$\{\texttt{ISBN}\}^+ = \{\texttt{ISBN}, \texttt{TITLE}, \texttt{PUBLISHER}, \texttt{PUB\_URL}\}$$

⑧ We may now conclude, *e.g.*, $\texttt{ISBN} \rightarrow \texttt{PUB\_URL}$.

How to Determine Keys (1)

- Given a set of FDs and the set of all attributes $\mathcal{A}$ of a relation R, one can determine all possible keys of R:

$$\alpha \subseteq \mathcal{A} \text{ is key of } R \Leftrightarrow \alpha^+ = \mathcal{A} \ .$$

- Remember: Normally, we are interested in **minimal keys** only.

  A superset of a key is a key again ($\{\texttt{ISBN}, \texttt{NO}\}$ is key for relation `BOOKS` and so is $\{\texttt{ISBN}, \texttt{NO}, \texttt{TITLE}\}$). We thus usually require that every $A \in \alpha$ is vital, *i.e.*, $(\alpha - \{A\})^+ \neq \mathcal{A}$.

How to Determine Keys (2)

① Start to construct a key x iteratively with the empty attribute set ($x = \emptyset$).

② In order to avoid non-minimal keys, make sure that the set x never contains the rhs B of an FD $\alpha \rightarrow B$ when x already contains the lhs, *i.e.*, $\alpha \subseteq x$.

  Wlog, we assume the rhs of all FDs to consist of a single attribute only.

How to Determine Keys (3)

③ As long as x does not yet determine all attributes of R (*i.e.*, $x^+ \neq \mathcal{A}$), choose any of the remaining attributes $X \in \mathcal{A} - x^+$.

④ Now the process splits (need to follow both alternatives):

- ⓐ Simply add X to x.
- ⓑ For each FD $\alpha \rightarrow X$, add α to x.

The search for the key branches. We need to backtrack to such a branch if

- we run into a dead-end (determine a non-minimal key), or
- we have found a key and want to look for an alternative minimal key.

How to Determine Keys (4)

Key Computation (call with $x = \emptyset$)

```
procedure key(x, A, F)
  foreach α → B ∈ F do
    if α ⊆ x and B ∈ x and B ∉ α then
      return;                                  /* x not minimal */
    fi
  od
  if x⁺ = A then
    print x;                                   /* found a minimal key x */
  else
    X ← any element of A − x⁺;
    key(x ∪ {X}, A, F);
    foreach α → X ∈ F do
      key(x ∪ α, A, F);
    od
  fi
```

FD Quiz (1)

Determine keys for a relation

RESULTS

STUD_ID	EX_NO	POINTS	MAX_POINTS
100	1	9	10
101	1	8	10
102	1	10	10
101	2	11	12
102	2	10	12

$$\mathcal{F} = \left\{ \begin{array}{rcl} \texttt{STUD\_ID}, \texttt{EX\_NO} & \rightarrow & \texttt{POINTS} \\ \texttt{EX\_NO} & \rightarrow & \texttt{MAX\_POINTS} \end{array} \right\}$$

① Does $\mathcal{F}$ imply $\texttt{STUD\_ID}, \texttt{EX\_NO} \rightarrow \texttt{MAX\_POINTS}$?

② Determine a key for relation RESULTS.

Notes

Determinants

Determinants (Non-trivial FDs)

The attribute set $A_1, \ldots, A_n$ is called a **determinant** for attribute set $B_1, \ldots, B_m$ if and only if

① the FD $A_1, \ldots, A_n \rightarrow B_1, \ldots B_m$ holds, and

② the lhs is minimal, *i.e.*, whenever any A_i is removed then $A_1, \ldots, A_{i-1}, A_{i+1}, A_n \rightarrow B_1, \ldots B_m$ does not hold, and

③ the lhs and rhs are distinct, *i.e.*, $\{A_1, \ldots, A_n\} \neq \{B_1, \ldots, B_m\}$.

Relational Normal Forms

Overview

1. Functional Dependencies (FDs)
2. Anomalies, FD-based Normal Forms
3. Multivalued Dependencies (MVDs) and 4NF
4. Normal Forms and ER Design
5. Denormalization

Consequences of Bad DB Design (1)

- It is normally a severe sign of **bad DB design** if a table contains an FD (encodes a partial function) that is **not implied by a key** (*e.g.*, INAME → PHONE).
- Among other problems, this **leads to the redundant storage of certains facts** (here: phone numbers):

Redundant data storage

COURSES

CRN	TITLE	INAME	PHONE
22268	Databases I	Grust	75477
42232	Query Compilers	Grust	75477
31822	Functional Programming	Klaeren	78955

Consequences of Bad DB Design (2)

- If a schema permits redundant data, additional **constraints** need to be specified to ensure that the redundant copies indeed agree.
- In the example, this constraint is precisely the FD INAME → PHONE.
- **General FDs**, however, are *not* standard constraints of the relational model and thus **unsupported by RDBMS.**
- The solution is to transform FDs into **key constraints**. This is what **DB normalization** tries to do.

Consequences of Bad DB Design (3)

- **Update anomalies:**
  - ▷ When a **single mini-world fact** needs to be changed (*e.g.*, a change in phone number for `Grust`), **multiple tuples** must be updated. This complicates application programs.
  - ▷ Redundant copies potentially get out of sync and it is impossible/hard to identify the correct information.
  - ▷ Application programs unexpectedly receive multiple answers when a single answer was expected:

`SELECT INTO` breaks due to unexpected result

```
SELECT DISTINCT PHONE
  INTO :phone
  FROM COURSES
 WHERE INAME = 'Grust'
```


Consequences of Bad DB Design (4)

- **Insertion anomalies:**
  - ▷ The phone number of a new instructor cannot be inserted into the database until it is known what course(s) she/he will teach.
  - ▷ Since `CRN` is a key, it cannot be set to `NULL`.

  - ▷ Such insertion anomalies arise when **unrelated concepts** (here: courses and instructors' phone numbers) are **stored together in a single table.**

    Remember: a relational table should encode a *single* (partial) function.

Consequences of Bad DB Design (5)

- **Deletion anomalies:**
  - ▷ When the last course of an instructor is deleted, his/her phone number is lost.

    Even if `CRN` could be set to `NULL`: it would be strange that all courses held by an instructor may be deleted but the last.
  - ▷ Again: such deletion anomalies arise when **unrelated concepts** (here: courses and instructors' phone numbers) are **stored together in a single table.**

Boyce-Codd Normal Form (1)

Boyce-Codd Normal Form

A relation R is in **Boyce-Codd Normal Form (BCNF)** if and only if all its FDs are already implied by its key constraints.

For relation R in BCNF and any FD $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ of R one of the following statements hold:

① the FD is trivial, *i.e.*, $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$, or

② the FD follows from a key, because $\{A_1, \ldots, A_n\}$ or a subset of it is already a key of R.

Boyce-Codd Normal Form (2)

- BCNF $\Longleftrightarrow$ Every determinant is a possible key of R.
- If a relation R is in BCNF, ensuring its key constraints automatically satisfies all of R's FDs.
- The **three anomalies** (udpate/insertion/deletion) **do not occur.**

BCNF Examples (1)

BCNF counter example

The relation

COURSES (CRN, TITLE, INAME, PHONE)

with the FDs

CRN → TITLE, INAME, PHONE

INAME → PHONE

is **not in BCNF** because the FD INAME → PHONE is not implied by a key:

- INAME is not a key of the entire relation,
- the FD is not trivial.

However, without the attribute PHONE (and the second FD) the relation COURSES (CRN, TITLE, INAME) is in BCNF.

BCNF Examples (2)

BCNF example

Each course meets once per week in a dedicated room:

`CLASS (CRN, TITLE, WEEKDAY, TIME, ROOM)`

The relation thus satisfies the following FDs (plus implied ones):

$$\texttt{CRN} \rightarrow \texttt{TITLE}, \texttt{WEEKDAY}, \texttt{TIME}, \texttt{ROOM}$$
$$\texttt{WEEKDAY}, \texttt{TIME}, \texttt{ROOM} \rightarrow \texttt{CRN}$$

- The keys of `CLASS` are
  - ▷ `CRN`, and
  - ▷ `WEEKDAY`, `TIME`, `ROOM`.
- Both FDs are implied by keys (in this case, their lhs even conincide with the keys), thus `CLASS` **is in BCNF.**

BCNF Examples (3)

BCNF example

Consider the relation `PRODUCT (NO, NAME, PRICE)` and the following FDs:

$$\texttt{NO} \rightarrow \texttt{NAME} \qquad \texttt{PRICE}, \texttt{NAME} \rightarrow \texttt{NAME}$$
$$\texttt{NO} \rightarrow \texttt{PRICE} \qquad \texttt{NO}, \texttt{PRICE} \rightarrow \texttt{NAME}$$

- This relation **is in BCNF:**
  - ▷ The two left FDs indicate that `NO` is a key. Both FDs are thus implied by a key.
  - ▷ The third FD is trivial (and may be ignored).
  - ▷ The lhs of the last FD contains a key and is thus also implied by a key.

BCNF Quiz

BCNF Quiz

① Is `RESULTS (STUD_ID, EX_NO, POINTS, MAX_POINTS)` with the following FDs in BCNF?

$$\begin{aligned} \texttt{STUD\_ID, EX\_NO} &\rightarrow \texttt{POINTS} \\ \texttt{EX\_NO} &\rightarrow \texttt{MAX\_POINTS} \end{aligned}$$

② Is `INVOICE (INV_NO, DATE, AMOUNT, CUST_NO, CUST_NAME)` with the following FDs in BCNF?

$$\begin{aligned} \texttt{INV\_NO} &\rightarrow \texttt{DATE, AMOUNT, CUST\_NO} \\ \texttt{INV\_NO, DATE} &\rightarrow \texttt{CUST\_NAME} \\ \texttt{CUST\_NO} &\rightarrow \texttt{CUST\_NAME} \\ \texttt{DATE, AMOUNT} &\rightarrow \texttt{DATE} \end{aligned}$$

Notes

Third Normal Form (1)

- **Third Normal Form (3NF)** is slightly weaker than BCNF: if a relation is in BCNF, it is automatically in 3NF.

  The differences are small. For most practical applications, the two can be considered as equivalent.

- BCNF is a clear concept: we want no non-trivial FDs except those which are implied by a key constraint.

- In some rare scenarios, this **preservation of FDs** (see below) is lost when a relation is transformed into[8] BCNF. This never happens with 3NF.

[8] "normalized into"

Third Normal Form (2)

Third Normal Form

A relation R is in **Third Normal Form (3NF)** if and only if every FD $A_1, \ldots, A_n \rightarrow B$[a] satisfies at least one of the following conditions:

1. the FD is trivial, *i.e.*, $B \in \{A_1, \ldots, A_n\}$,or
2. the FD follows from a key, because $\{A_1, \ldots, A_n\}$ or some subset of it is already a key of R, or
3. B is a **key attribute**, *i.e.*, element of a key of R.

[a] Assume that FDs with multiple attributes on rhs have been expanded.

Third Normal Form (3)

- The only difference to BCNF is the additional clause ③.
- An attribute is called a **non-key attribute** (of R), if it does not appear in any minimal key of R.

  Note: the minimality requirement is important here. Otherwise all attributes of R would qualify as key attributes.

- 3NF means that **every determinant of a non-key attribute is a key** (see clauses ②, ③).

Second Normal Form (1)

- The **Second Normal Form (2NF)** is only interesting for historical reasons.

Second Normal Form (2NF)

A relation is in **Second Normal Form (2NF)** if and only if every non-key attribute depends on the **complete** key.

- If a relation is in 3NF or BCNF, it is automatically in 2NF.
- 2NF is "too weak", *e.g.*, the `COURSES` example actually satisfies 2NF (although it exhibits anomalies).

Second Normal Form (2)

- A relation is in 2NF is and only if the given FDs do not imply an FD $A_1, \ldots, A_n \to B$ such that
  1. $A_1, \ldots, A_n$ is a **strict subset** of a minimal key, and
  2. B is not contained in any minimal key (*i.e.*, is a non-key attribute).

2NF counter example

The homework results table

`RESULTS (STUD_ID, EX_NO, POINTS, MAX_POINTS)`

is **not in 2NF:** `MAX_POINTS` is already uniquely determined by `EX_NO`, *i.e.*, a part of a key.

Splitting Relations (1)

- If a table R is not in BCNF, we can **split** it into two tables (**decomposition**). The violating FD determines how to split.

Splitting "along an FD"

Remember: the FD `INAME` $\to$ `PHONE` is the reason why table

`COURSES (CRN, TITLE, INAME, PHONE)`

violates BCNF.

Split the table into

`INSTRUCTORS (CRN, TITLE, INAME)`

`PHONEBOOK (INAME, PHONE)`

- If the FD $A_1, \ldots, A_n \to B_1, \ldots B_m$ violates BCNF, create a new relation $S(\underline{A_1}, \ldots, \underline{A_n}, B_1, \ldots, B_m)$ and remove $B_1, \ldots, B_m$ from the original relation R.

Splitting Relations (2)

- It is important that this splitting transformation is **lossless**, *i.e.*, that the original relation can be reconstructed by means of a (natural) join:

Reconstruction after split

$$\texttt{COURSES} = \texttt{INSTRUCTORS} \bowtie \texttt{PHONEBOOK}$$

```
CREATE VIEW COURSES (CRN, TITLE, INAME, PHONE)
    AS
SELECT I.CRN, I.TITLE, I.INAME, P.PHONE
  FROM INTSTRUCTORS I, PHONEBOOK P
 WHERE I.INAME = P.INAME
```


Splitting Relations (3)

Decomposition Theorem

The split of relations is **guaranteed to be lossless** if the **intersection of the attributes of the new tables is a key of at least one of them.**

The join connects tuples depending on the attribute (values) in the intersection. If these values uniquely identify tuples in the other relation we do not lose information.

"Lossy" decomposition

Original table (key A, B, C)

A	B	C
a_{11}	b_{11}	c_{11}
a_{11}	b_{11}	c_{12}
a_{11}	b_{12}	c_{11}

Decomposition

R_1

A	B
a_{11}	b_{11}
a_{11}	b_{12}

R_2

A	C
a_{11}	c_{11}
a_{11}	c_{12}

"Reconstruction" $R_1 \bowtie R_2$

A	B	C
a_{11}	b_{11}	c_{11}
a_{11}	b_{11}	c_{12}
a_{11}	b_{12}	c_{11}
a_{11}	b_{12}	c_{12}

Splitting Relations (4)

Lossless split condition satisfied

$$\{\texttt{CRN}, \texttt{TITLE}, \texttt{INAME}\} \cap \{\texttt{INAME}, \texttt{PHONE}\} = \{\texttt{INAME}\}$$

- All splits iniated by the above method for transforming relations into BCNF satisfy the condition of the decomposition theorem.
- It is **always possible** to transform a relation into BCNF by lossless splitting (if necessary, split repeatedly).

Splitting Relations (5)

- However, not every lossless split is reasonable:

STUDENTS

SSN	FIRST_NAME	LAST_NAME
111-22-3333	John	Smith
123-45-6789	Maria	Brown

- Splitting STUDENTS into STUD_FIRST (SSN, FIRST_NAME) and STUD_LAST (SSN, LAST_NAME) is lossless, but
  - ▷ the split is *not* necessary to enforce a normal form,
  - ▷ only requires costly joins in subsequent queries.

  Full splitting down to **binary tables** on the physical DB level, however, leads to high-performance DBMS implementations.

Splitting Relations (6)

- Losslessness means that the resulting schema (after splitting) can represent all DB states which were possible before.

  We can translate states from the old schema into the new schema and back. The new schema supports all queries supported by the old schema (may "simulate" old schema via views).

- However, the new schema allows states which do not correspond to the state in the old schema (we may now store instructors and phone numbers without any affiliation to courses).

  The new schema is more **general.**

Splitting Relations (7)

- If instructors without courses are possible in the real world, the decomposition removes a fault in the old schema (insertion and update anomalies).
- If they are not,
  - ▷ a new constraint is needed that is not necessarily easier to enforce than the FD, but
  - ▷ at least the redundancy is avoided (update anomaly).

Splitting Relations (8)

- Finally, note that although **computable columns** (such as `AGE` which is derivable from `BIRTDATE`) lead to violations of BCNF, splitting the relation is *not* the right solution.

  The split would yield a relation `R(BIRTHDAY, AGE)` which would try to materialize the computable function.

- The correct solution is to eliminate `AGE` from the table and to define a view which contains all columns plus the **computed** column `AGE` (invoking a SQL stored procedure).

Preservation of FDs (1)

- Besides losslessness, a property which a good decomposition of a relation should guarantee is the **preservation of FDs.**
- The problem is that an FD can refer only to attributes of a single relation.
- When you split a relation into two, there might be FDs that can no longer be expressed (these FDs are not preserved).

Preservation of FDs (2)

FD gets lost during decomposition

Consider `ADRESSES (STREET_ADDR, CITY, STATE, ZIP)` with FDs

$$\texttt{STREET\_ADDR}, \texttt{CITY}, \texttt{STATE} \rightarrow \texttt{ZIP}$$
$$\texttt{ZIP} \rightarrow \texttt{STATE}$$

- The second FD violates BCNF and would lead us to split the relation into
  - ▷ `ADDRESSES1 (STREET_ADDR, CITY, ZIP)` and
  - ▷ `ADDRESSES2 (ZIP, STATE)`.
- But now the first FD can no longer be expressed.

Preservation of FDs (3)

- Probably, most designers would not split the table (the table is in 3NF but violates BCNF).
- If there are many addresses with the same ZIP code, there will be significant redundancy. If you split, queries will involve more joins.
- If this were a DB for a mailing company, a separate table of ZIP codes might be of interest on its own. Then the split looks feasible.

3NF Synthesis Algorithm (1)

- The following algorithm, the **synthesis algorithm**, produces a lossless decomposition of a given relation into 3NF relations that preserve the FDs.

- Determine a **minimal** (canonical) set of FDs that is equivalent to the given FDs as follows:

  ⓐ Replace every FD $\alpha \to B_1, \ldots, B_m$ by the corresponding FDs $\alpha \to B_i$, $1 \leqslant i \leqslant m$. Let the result be $\mathcal{F}$.

  ⓑ Minimize the lhs of every FD in $\mathcal{F}$.
  For each FD $A_1, \ldots, A_n \to B$ and each $i = 1, \ldots, n$ compute $\{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n\}^+_{\mathcal{F}}$. If the result contains B, replace $\mathcal{F}$ by $(\mathcal{F} - \{A_1, \ldots, A_n \to B\}) \cup \{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \to B\}$, and repeat.

continued. . .

3NF Synthesis Algorithm (2)

. . . continued from last slide

ⓒ Remove implied FDs.
For each FD $\alpha \to B$, compute the cover $\alpha^+_{\mathcal{F}'}$ where $\mathcal{F}' = \mathcal{F} - \{\alpha \to B\}$. If the cover contains B (*i.e.*, the FD is already implied by the FDs in $\mathcal{F}$), continue with $\mathcal{F} \leftarrow \mathcal{F}'$.

3NF Synthesis Algorithm (3)

① Compute a canonical (minimal) set of FDs $\mathcal{F}$ (see above).

② For each lhs α of an FD in $\mathcal{F}$ create a relation with attributes $\mathcal{A} = \alpha \cup \{B \mid \alpha \to B \in \mathcal{F}\}$.

③ If none of the relations constructed in step ② contains a key of the original relation R, add one relation containing the attributes of a key of R.

④ For any two relations $R_{1,2}$ constructed in steps ②,③, if the schema of R_1 is contained in the schema of R_2, discard R_1.

Summary

- Tables should *not* contain FDs other than those implied by the keys (*i.e.*, all tables should be in **BCNF**).

  Such violating FDs indicate the combination of pieces of information which should be stored separately (presence of an embedded function). This leads to redundancy.

- A relation may be **normalized** into BCNF by splitting it.

- Sometimes it may make sense to avoid a split (and thus to violate BCNF). The DB designer has to carefully resolve such scenarios, incorporating application or domain knowledge.

Relational Normal Forms

Overview

1. Functional Dependencies (FDs)
2. Anomalies, FD-based Normal Forms
3. Multivalued Dependencies (MVDs) and 4NF
4. Normal Forms and ER Design
5. Denormalization

Introduction (1)

- The development of BCNF has been guided by a particular type of constraint: FDs.
- The goal of normalization into BCNF/3NF is to
  - ▷ elimminate the redundant storage of data that follows from these constraints, and to
  - ▷ transform tables such that the constraints are automatically enforced by means of keys.
- However, there are **further types of constraints** which are also useful to during DB design.

Introduction (2)

- The condition in the decomposition theorem is only
  - ▷ **sufficient** (it guarantees losslessness),
  - ▷ but **not necessary** (a decomposition might be lossless even if the condition is not satisfied).
- **Multivalued dependencies (MVDs)** are constraints which give a **necessary and sufficient** condition for lossless decomposition.
- MVDs lead to the **Fourth Normal Form (4NF).**

Introduction (3)

- 4NF intuitively:

  Whenever it is possible to split a table (*i.e.*, the decomposition is lossless) and the split is not superfluous (cf. slide 386), then split!

- Still shorter:

  "Do not store unrelated information in the same relation."

- In practice it is very seldom that a relation violates 4NF but not BCNF.

  Such cases occur, however (*e.g.*, when merging two binary relationships into a ternary relationship, see below).

Multivalued Dependencies (1)

MVD example

Suppose that every employee knows a set of programming languages as well as a set of DBMSs. Such knowledge typically is **independent.**

EMP_KNOWLEDGE

ENAME	PROG_LANG	DBMS
John Smith	C	Oracle
John Smith	C	DB2
John Smith	C++	Oracle
John Smith	C++	DB2
Maria Brown	Prolog	PostgreSQL
Maria Brown	Java	PostgreSQL

Multivalued Dependencies (2)

- If the sets of known programming languages and DBMSs are indeed **independent** (in particular PROG_LANG $\not\to$ DBMS and DBMS $\not\to$ PROG_LANG), the table contains redundant data, and must be split:

EMP_LANG

ENAME	PROG_LANG
John Smith	C
John Smith	C++
Maria Brown	Prolog
Maria Brown	Java

EMP_DBMS

ENAME	DBMS
John Smith	Oracle
John Smith	DB2
Maria Brown	PostgreSQL

- The original table is in BCNF (no non-trivial FDs).

Multivalued Dependencies (3)

- Note: table EMP_KNOWLEDGE may only be decomposed if the knowledge of DBMS and programming languages is **indeed independent.**
- Otherwise, decomposition leads to **loss of information.**

  If the semantics of EMP_KNOWLEDGE is that the employee knows the interface between the programming language and the DBMS (*i.e.*, the knowledge is *not* independent), we cannot deduce from the split tables if John Smith indeed knows the C and C++ interfaces of both DBMS, Oracle and DB2.

Multivalued Dependencies (4)

- The **multivalued dependency (MVD)**

  $$\texttt{ENAME} \twoheadrightarrow \texttt{PROG\_LANG}$$

  indicates that the **set of values** in column PROG_LANG associated with every ENAME is **independent of all other columns.**

  The tables contains an embedded function from ENAME to **sets of** PROG_LANG.

- Formally, ENAME $\twoheadrightarrow$ PROG_LANG holds if whenever two tuples agree on ENAME, one can exchange their PROG_LANG values and get two other tuples of the same table.

Multivalued Dependencies (5)

- From the two table rows

ENAME	PROG_LANG	DBMS
John Smith	C	Oracle
John Smith	C++	DB2

and the MVD ENAME $\twoheadrightarrow$ PROG_LANG, we can conclude that the table must also contain the following rows:

ENAME	PROG_LANG	DBMS
John Smith	C++	Oracle
John Smith	C	DB2

- This expresses the **independence** of PROG_LANG for a given ENAME from the rest of the table columns.

Multivalued Dependencies (6)

Multivalued Dependency

A **multivalued dependency (MVD)**

$$A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$$

is satisfied in a DB state I if and only if for all tuples t, u with $t.A_i = u.A_i, 1 \leqslant i \leqslant n$, there are two further tuples t', u' such that

① t' agrees with t except that $t'.B_j = u.B_j, 1 \leqslant j \leqslant m$, and

② u' agrees with u except that $u'.B_j = t.B_j, 1 \leqslant j \leqslant m$.

(*i.e.*, the values of the B_j are swapped).

Multivalued Dependencies (7)

- Note: MVDs always **come in pairs:** if `ENAME` $\twoheadrightarrow$ `PROG_LANG` holds, then `ENAME` $\twoheadrightarrow$ `DBMS` is automatically satisfied.

  ▷ More general:

  For $R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_k)$ the MVD $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$ is equivalent to the MVD $A_1 \ldots, A_n \twoheadrightarrow C_1, \ldots, C_k$: swapping the B_j values in two tuples is the same as swapping the values for all other columns (the A_i values are identical, so swapping them has no effect).

Multivalued Dependencies (8)

Trivial MVD

An MVD $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$ for a relation R is **trivial** if and only if

① $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$, or

For two tuples agreeing on the A_i (and thus also on the B_j) swapping has no effect.

② $\{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_m\}$ are all columns of R.

In this case, swapping tuple t, u gives the tuples u, t, *i.e.*, the same tuples again.

Multivalued Dependencies (9)

- If the FD $A_1, \ldots A_n \to B_1, \ldots B_m$ holds, the corresponding MVD $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$ is trivially satisfied.

  The FD means that if tuples t, u agree on the A_i then they also agree on the B_j. Swapping thus has no effect (yields t, u again).

- Just like an FD may be implied (*i.e.*, automatically follow from) by a given set of FDs, FDs and MVDs can also follow from a set of given FDs and MVDs.

FD/MVD Deduction Rules

- These deduction rules permit to derive **all** implied FDs/MVDs[9]

  ▷ The three Armstrong Axioms for FDs.
  ▷ If $\alpha \twoheadrightarrow \beta$ then $\alpha \twoheadrightarrow \gamma$, where γ are all remaining columns of the relation.
  ▷ If $\alpha_1 \twoheadrightarrow \beta_1$ and $\alpha_2 \supseteq \beta_2$ then $\alpha_1 \cup \alpha_2 \twoheadrightarrow \beta_1 \cup \beta_2$.
  ▷ If $\alpha \twoheadrightarrow \beta$ and $\beta \twoheadrightarrow \gamma$ then $\alpha \twoheadrightarrow (\gamma - \beta)$.
  ▷ If $\alpha \to \beta$, then $\alpha \twoheadrightarrow \beta$.
  ▷ If $\alpha \twoheadrightarrow \beta$ and $\beta' \subseteq \beta$ and there is γ with $\gamma \cap \beta = \emptyset$ and $\gamma \to \beta'$, then $\alpha \to \beta'$.

[9] The rules are *sound* (only derive implied FDs/MVDs) and *complete* (derive all such dependencies).

Fourth Normal Form (1)

Fourth Normal Form (4NF)

A relation is in **Fourth Normal Form (4NF)** with respect to given FDs and MVDs if and only if no MVD $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$ is implied which is

① not trivial and

② not directly implied by a key, *i.e.*, $A_1, \ldots, A_n$ does not uniquely determine all other attributes.

- Note: this definition of 4NF is very similar to BCNF but with a focus on implied MVDs (not FDs).

Fourth Normal Form (2)

- Since every FD is also an MVD, 4NF is stronger than BCNF (*i.e.*, if a relation is in 4NF, it is automatically in BCNF).
- The relation

  `EMP_KNOWLEDGE (ENAME, PROG_LANG, DBMS)`

  is an example of a relation that is in BCNF, but not in 4NF.

  The relation has no non-trivial FDs (the key consists of all attributes).
- However, it is not very common that 4NF is violated, but BCNF is not.

Fourth Normal Form (3)

- **Splitting** a relation

$$R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_k)$$

into relations

$$R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \qquad R_2(A_1, \ldots, A_n, C_1, \ldots, C_k)$$

is lossless, *i.e.*

$$R = \underbrace{\pi_{A_1,\ldots,A_n,B_1,\ldots,B_m}(R)}_{=R_1} \bowtie \underbrace{\pi_{A_1,\ldots,A_m,C_1,\ldots,C_k}(R)}_{=R_2}$$

if and only if $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$ (or $A_1, \ldots, A_n \twoheadrightarrow C_1, \ldots, C_k$).

Fourth Normal Form (4)

- Any relation can be transformed into 4NF by splitting as shown above.

  To reach 4NF, it might be necessary to split multiple times.

- So the **essence of 4NF** is:

  ▷ If a **decomposition** into two relations **is lossless** (*i.e.*, the original relation may always be reconstructed by a join),

  ▷ and the two resulting relations do *not* have identical keys (then the split would be superfluous),

  ▷ then this decomposition must be done.

Fifth Normal Form

- **Fifth Normal Form (5NF)** is very seldomly used in practice. We only sketch some issues here.[10]
- 4NF handles all cases where a decomposition into **two tables** is needed.
- However, it is theoretically possible that only a decomposition into **three or more tables is lossless**, but no decomposition into two tables is lossless.

  The required decomposition thus cannot be obtained by repeated splitting into two tables. Instead, one needs additional tables with overlapping join columns which only serve as a filter in the reconstructing join.

[10] See *Relational Database Theory*, Atzeni, DeAntonellis; Benjamin Cummings, 1993; ISBN 0-8053-0249-2.

Join Dependencies (1)

Lossy split into multiple tables

Consider `COURSE_OFFER(COURSE, TERM, INSTRUCTOR)`
Normally, information is lost by the split into

`OFFER(COURSE, TERM)`
`EMP(INSTRUCTOR, TERM)`
`TEACHES(COURSE, INSTRUCTOR)`

- But the split would be lossless if the following constraint were true: "*If a course C is offered in term T, and instructor I ever taught C, and I teaches some course in T, then I teaches C in T.*"

Join Dependencies (2)

- There are relations which have a lossless decomposition **without satisfying any non-trivial FD.**

  Note: the decomposition theorem gives a sufficient conditon only.

Lossless decomposition without FDs

$R =$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2
a_1	b_1	c_2	d_2
a_1	b_2	c_2	d_2
a_2	b_2	c_2	d_2

Relation R does not satisfy any non-trivial FD but has a lossless decomposition into $R_1(A, B)$, $R_2(B, C)$, and $R_3(C, D)$.

Join Dependencies (3)

- A **join dependency (JD)** states that a split of R into n relations is lossless:

$$R = \pi_{A_{i_{1,1}},\ldots,A_{i_{1,k_1}}}(R) \bowtie \cdots \bowtie \pi_{A_{i_{n,1}},\ldots,A_{i_{n,k_n}}}(R)$$

  Of course, each attribute of R must appear in at least one of the projections. Then "$\subseteq$" is always satisfied (as we have seen already). The equality states that if there are n tuples $t_1, \ldots, t_n$ found in the split relations which satisfy the join predicates, then this corresponding result tuple also appears in R.

- An MVD $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots B_m$ is a special JD:

$$R = \pi_{A_1,\ldots,A_n,B_1,\ldots,B_m}(R) \bowtie \pi_{A_1,\ldots,A_n,C_1,\ldots,C_k}(R)$$

  where $C_1, \ldots, C_k$ are the remaining columns of R.

Other Constraints (1)

Multiple choice test

The following relation encodes the correct solution to a typical multiple choice test:

ANSWERS

QUESTION	ANSWER	TEXT	CORRECT
1	A	...	Y
1	B	...	N
1	C	...	N
2	A	...	N
2	B	...	Y
2	C	...	N

Other Constraints (2)

Using keys to enforce other constraints

The following is an example of a constraint for table ANSWERS that is not an FD, MVD, or JD:

"*Each question can only have one correct answer.*"

① Can you suggest a transformation of table ANSWERS such that the above constraint is already implied by a key?

Notes

Relational Normal Forms

Overview

1. Functional Dependencies (FDs)
2. Anomalies, FD-based Normal Forms
3. Multivalued Dependencies (MVDs) and 4NF
4. Normal Forms and ER Design
5. Denormalization

Introduction (1)

- If a "good" ER schema is transformed into the relational model, the resulting tables will **satisfy all normal forms** (4NF, BCNF, 3NF, 2NF).
- A normal form violation detected in the generated relational schema indicates a flaw in the input ER schema. This needs to be corrected on the ER level.
- There is no independent normalization theory for the ER model.

Examples (1)

FDs in the ER model

The ER equivalent of the very first example in this chapter:

- Obviously, the FD IName $\rightarrow$ Phone leads to a violation of BCNF in the resulting table for entity Course.
- Also in the ER model, FDs between attributes of an entity should be implied by a key constraint.

Examples (2)

- In the ER model, the solution is the "same" as in the relational model: we have to **split** the entity.

ER entity split

In this case, we discover that Instructor is an idependent entity:

Examples (3)

- Functional dependencies between **attributes of a relationship** always violate BCNF:

Violation of BCNF on the ER level

The FD OrderNo $\rightarrow$ Date violates BCNF.

The key of the table corresponding to the relationship "orders" consists of the attributes CustNo, ProdNo.

- This shows that the concept "Order" is an independent entity.

Examples (4)

- Violations of BCNF might also be due to the **wrong placement of an attribute:**

Questionable attribute placement

▷ The relationship is translated into

TAKES (STUD_ID, CRN, EMAIL)

▷ Then the FD STUD_ID → EMAIL violates BCNF.

▷ Obviously, Email should be an attribute of Student.

Examples (5)

- If an attribute of a ternary relationship depends only on two of the entities, this violates BCNF (2NF):

Ternary relationship

▷ If every course is taught only once per term, then attribute Room depends only on Term and Course (but not Instructor).

Examples (6)

- If **independent entities** are combined, 4NF is violated:

Independent entities combined

▷ If the knowledge of programming languages and DBMSs is independent, use **two binary relationships** instead.

One between Employee and DBMS, one between Employee and Prog_Lang.

Normalization: Summary

- **Relational normalization** is about:
  - ▷ **Avoiding redundancy.**
  - ▷ **Stroting separate facts (functions) separately.**
  - ▷ **Transforming general integrity constaints** into constraints that are supported by the DBMS: **keys**
- Relational normalization theory is mainly based on FDs, but there are other types of constraints (*e.g.*, MVDs, JDs).

Relational Normal Forms

Overview

1. Functional Dependencies (FDs)
2. Anomalies, FD-based Normal Forms
3. Multivalued Dependencies (MVDs) and 4NF
4. Normal Forms and ER Design
5. Denormalization

Denormalization (1)

- **Denormalization** is the process of **adding redundant columns** to the database in order to **improve performance.**
- For example, if an application extensively access the phone number of instructors, performance-wise it may make sense to add column `PHONE` to table `COURSES`.

Redundant data storage

COURSES			
CRN	TITLE	INAME	PHONE

▷ This **avoids the otherwise required joins** (on attribute `INAME`) between tables `COURSES` and `PHONEBOOK`.

Denormalization (2)

- Since there is still the separate table `PHONEBOOK`, **insertion and deletion anomalies are avoided.**
- But there will be **update anomalies** (changing a single phone number requires the update of many rows).
- The performance gain is thus paid for with
  - ▷ a more complicated application logic (*e.g.*, the need for triggers),
  - ▷ and the risk that a faulty application will turn the DB inconsistent.

Denormalization (3)

- If the table `COURSES` and `PHONEBOOK` contain few tuples, it is probably a bad decision to violate BCNF here. Performance will be no problem anyway.
- Denormalization may not only be used to avoid joins.
  - ▷ Complete **separate, redundant tables** may be created (increasing the potential for parallel DBMS operation, especially updates).
  - ▷ Columns may be added which **aggregate** information in other columns/rows (*e.g.*, store the current `BALANCE` in a `CUSTOMER` table that also records `PAYMENT`s and `INVOICE`s).

    In this case, BCNF might not be violated but the added information will be redundant nevertheless.