

Databases

Jörg Endrullis

VU University Amsterdam

2015

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

- **Functional Dependencies (FDs)**
 - are a **generalization of keys**
 - central part of **relational database design theory**
- This theory defines when a relation is in **normal form**.
(with respect to a given set of functional dependencies)
- It is usually a sign of **bad database design** if a schema contains relations that **violate the normal form**.
- If a normal form is violated
 - data is stored **redundantly** and
 - information about different concepts is **intermixed**

| COURSES | | | |
|---------|------------------------|-------|-------|
| CRN | TITLE | INAME | PHONE |
| 22268 | Databases I | Grust | 7111 |
| 42232 | Functional Programming | Grust | 7111 |
| 31822 | Graph Theory | Klotz | 2418 |

The phone number for each instructor is stored multiple times!

Introduction

There are different normal forms. The main ones are:

- **Third Normal Form (3NF):** the standard relational normal form used in practice (and education).
- **Boyce-Codd Normal Form (BCNF):**
 - a bit more restrictive
 - easier to define
 - better for our intuition of good database design

Roughly speaking, BCNF requires that **all FDs are keys**.

- In rare circumstances, a relation might not have an equivalent BCNF form while preserving all its FDs.
The 3NF normal form always exists (and preserves the FDs).

Introduction

Normalization algorithms can construct good relation schemas from a set of attributes and a set of functional dependencies.

In practice:

- relations are derived from ER models
- normalization is used as an additional check only

When an ER model is **well designed**, the resulting derived relational tables will **automatically be in BCNF**.

- Awareness of normal forms can help to detect design errors already in the conceptual design phase.

First Normal Form

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

- The relational model all table entries are already atomic.
- All further normal forms assume that tables are in 1NF.

The following are **not violations of 1NF**:

- A table entry contains values with internal structure.
 - e.g. a CHAR(100) containing a comma separated list
- List represented by several columns.
 - e.g. columns value1, value2, value3

Nevertheless, these are **bad design**.

Functional Dependencies

| COURSES | | | |
|------------|------------------------|-------|-------|
| <u>CRN</u> | TITLE | INAME | PHONE |
| 22268 | Databases I | Grust | 7111 |
| 42232 | Functional Programming | Grust | 7111 |
| 31822 | Graph Theory | Klotz | 2418 |

A **functional dependency (FD)** in this table is

$INAME \rightarrow PHONE$

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

Functional Dependencies

Intuitively, there is a functional dependency

$$\text{INAME} \rightarrow \text{PHONE}$$

since the phone number **only depends on the instructor**, not on other course data.

This functional dependency read as

INAME {functionally, uniquely} determines PHONE

Also **INAME is a determinant for PHONE**.

A **determinant** is a 'minimal' functional dependency.

A determinant is like a **partial key**:

- uniquely determines some attributes, but not all in general

E.g. **INAME \rightarrow TITLE is not satisfied.**

Functional Dependencies

In general, an **functional dependencies** take the form

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

- sequence of attributes is unimportant
- both sides formally are sets of attributes

$$\{A_1, \dots, A_n\} \rightarrow \{B_1, \dots, B_m\}$$

The **functional dependency (FD)**

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

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

$$\begin{aligned} & t.A_1 = u.A_1 \wedge \dots \wedge t.A_n = u.A_n \\ \Rightarrow & t.B_1 = u.B_1 \wedge \dots \wedge t.B_m = u.B_m \end{aligned}$$

Functional Dependencies

A **key** uniquely determines **all** attributes of its relation.

- There are never two distinct rows with the same key, so the **functional dependency condition is trivially satisfied**.

| COURSES | | | |
|---------|------------------------|-------|-------|
| CRN | TITLE | INAME | PHONE |
| 22268 | Databases I | Grust | 7111 |
| 42232 | Functional Programming | Grust | 7111 |
| 31822 | Graph Theory | Klotz | 2418 |

We have the following functional dependencies:

- $CRN \rightarrow TITLE, INAME, PHONE$

or equivalently:

- $CRN \rightarrow TITLE$
- $CRN \rightarrow INAME$
- $CRN \rightarrow PHONE$

Functional Dependencies

An functional dependency with m attributes on the right

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

is **equivalent** to the m FDs:

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

Thus, in the following it suffices to consider FDs with a single column name on the right-hand side.

Functional Dependencies are Keys

Functional dependencies are **constraints** (like keys).

| COURSES | | | |
|---------|------------------------|-------|-------|
| CRN | TITLE | INAME | PHONE |
| 22268 | Databases I | Grust | 7111 |
| 42232 | Functional Programming | Grust | 7111 |
| 31822 | Graph Theory | Klotz | 2418 |

In this example state, the functional dependency

$\text{TITLE} \rightarrow \text{CRN}$

holds. But this is probably **not true in general!**

It is a task of DB design to verify if this is mere coincidence.

For the database design process, the only interesting functional dependencies are those that **hold for all possible states**.

Functional Dependencies are Keys

Functional dependencies are a **generalisation of keys**.

A_1, \dots, A_n is a key of relation $R(A_1, \dots, A_n, B_1, \dots, B_m)$



the functional dependency $A_1, \dots, A_n \rightarrow B_1, \dots, B_m$ holds.

| COURSES | | | |
|---------|------------------------|-------|-------|
| CRN | TITLE | INAME | PHONE |
| 22268 | Databases I | Grust | 7111 |
| 42232 | Functional Programming | Grust | 7111 |
| 31822 | Graph Theory | Klotz | 2418 |

Here $CRN \rightarrow TITLE, INAME, PHONE$.

Functional Dependencies are Keys

Functional dependencies are partial keys.

The functional dependency

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

holds for a relation R if $\{A_1, \dots, A_n\}$ is a key for the relation obtained by restricting R to the columns $\{A_1, \dots, A_n, B_1, \dots, B_m\}$.

The restriction of the table COURSES to $\{ \text{INAME}, \text{PHONE} \}$ is:

| COURSES | |
|---------|-------|
| INAME | PHONE |
| Grust | 7111 |
| Klotz | 2418 |

The attribute INAME is a key of this table.

The **goal of database normalization is to turn FDs into keys.**

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

Example

The following tables contains books and their 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 indicate the order of the authors

- The ISBN uniquely identifies a book. Thus

ISBN \rightarrow TITLE, PUBLISHER

Equivalently

- ISBN \rightarrow TITLE, and
- ISBN \rightarrow PUBLISHER

Example

The following tables contains books and their 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 indicate the order of the authors

- A book may have many authors. Thus

ISBN \rightarrow AUTHOR

does not hold!

Example

The following tables contains books and their 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 indicate the order of the authors

- One author can write many books, thus

AUTHOR \rightarrow TITLE

does not hold in general.

Although it happens to hold in the above database state.

Example

The following tables contains books and their 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 indicate the order of the authors
- There may be books with the same title but different authors and different publishers. So

TITLE

determines no other attributes.

Example

The following tables contains books and their 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 indicate the order of the authors
- Every book has only one first (second, third, ...) author.
Thus

ISBN, NO \rightarrow AUTHOR

Example

The following tables contains books and their 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 indicate the order of the authors
- At first glance, the author of any given book is also uniquely assigned a position in the authorship sequence.

ISBN, AUTHOR \rightarrow NO ? **questionable**

However, violated by an author list like Smith & Smith.

Example

The following tables contains books and their 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 indicate the order of the authors
- What about the functional dependency

PUBLISHER, TITLE, NO \rightarrow AUTHOR ? **questionable**

Authorship sequence might change in a new edition of a book!

Example

The following tables contains books and their 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 indicate the order of the authors

During database design, **only unquestionable conditions should be used as functional dependencies.**

- Database normalization **alters the table structure** depending on the specified functional dependencies.
Later hard to change: needs creation/deletion of tables!

Quiz

A table with homework grades:

| 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 |

- Which FDs should hold for this table in general?
- Identify FDs that hold in this table but not in general.

Implication of Functional Dependencies

Whenever $A \rightarrow B$ and $B \rightarrow C$ hold, then $A \rightarrow C$ is automatically satisfied.

Note that $CRN \rightarrow PHONE$ is a consequence of

$$CRN \rightarrow INAME \quad \text{and} \quad INAME \rightarrow PHONE$$

FDs of the form $A \rightarrow A$ always hold.

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

Implication of Functional Dependencies

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

Implication of Functional Dependencies

The DB designer is normally not interested in all FDs, 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 $\alpha \rightarrow \gamma$.

Use the Armstrong axioms to show that

ISBN \rightarrow TITLE, PUBLISHER

ISBN, NO \rightarrow AUTHOR

PUBLISHER \rightarrow PUB_URL

implies ISBN \rightarrow PUB_URL.

Implication of Functional Dependencies

Simpler way to **check whether $a \rightarrow \beta$ is implied by an FD set:**

- compute the **cover** α^+ of α , and
- then check if $\beta \subseteq \alpha^+$.

Cover

The **cover** $\alpha_{\mathcal{F}}^+$ of

- a set of attributes α
- with respect to an FD set \mathcal{F}

is the set of all attributes B that are uniquely determined by α :

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

Implication Check

A set of FDs \mathcal{F} implies an FD $\alpha \rightarrow \beta$ if and only if $\beta \subseteq \alpha_{\mathcal{F}}^+$.

Implication of Functional Dependencies

Cover computation

Input: α (set of attributes)
 $\alpha_1 \rightarrow \beta_1, \dots, \alpha_n \rightarrow \beta_n$ (set of FDs \mathcal{F})

Output: α^+ (the cover of α)

$x = \alpha;$

while x did change **do**

for all given FD $\alpha_j \rightarrow \beta_j$ **do**

if $\alpha_j \subseteq x$ **then**

$x = x \cup \beta_j;$ *(add attributes in β_j to x)*

end if

end for

end while

return $x;$

Implication of Functional Dependencies

Compute the cover $\{\text{ISBN}\}^+$ for the following FDs:

$\text{ISBN} \rightarrow \text{TITLE, PUBLISHER}$

$\text{ISBN, NO} \rightarrow \text{AUTHOR}$

$\text{PUBLISHER} \rightarrow \text{PUB_URL}$

1. We start with $x = \{\text{ISBN}\}$.
2. The FD $\text{ISBN} \rightarrow \text{TITLE, PUBLISHER}$ is applicable since the left-hand side of is completely contained in x .

We get $x = \{\text{ISBN, TITLE, PUBLISHER}\}$.

3. Now the FD $\text{PUBLISHER} \rightarrow \text{PUB_URL}$ is applicable.

We get $x = \{\text{ISBN, TITLE, PUBLISHER, PUB_URL}\}$.

4. No further way to extend set x , the algorithm returns

$$\{\text{ISBN}\}^+ = \{\text{ISBN, TITLE, PUBLISHER, PUB_URL}\}$$

5. We may now conclude, e.g., $\text{ISBN} \rightarrow \text{PUB_URL}$.

How to Determine Keys

Given a set of FDs and the set of all attributes \mathcal{A} of a relation R :

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

That is α is a key if the cover α^+ contains all attributes.

We can use FDs to determine all possible keys of R .

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

A key α is **minimal** if every $A \in \alpha$ is **vital**, that is

$$(\alpha - \{A\})^+ \neq \mathcal{A}$$

How to Determine Keys

Finding a Minimal Key

Input: \mathcal{A} (set of all attributes of R)
 $\alpha_1 \rightarrow \beta_1, \dots, \alpha_n \rightarrow \beta_n$ (set of FDs \mathcal{F})

Output: α (a minimal key of R)

$x = \mathcal{A}$;

for all attributes $A \in X$ **do**

if $A \in \{x - A\}_{\mathcal{F}}^{\pm}$ **then**

$x = x - A$; (remove A from x)

end if

end for

return x ;

We might get different keys depending on the order in **for all**.

How to Determine Keys

Finding all Minimal Keys

Input: A_1, A_2, \dots, A_n (all attributes of R) and \mathcal{F} (set of FDs)

$Results = \emptyset$;

$Candidates = \{\{A_1\}, \{A_2\}, \dots, \{A_n\}\}$;

while $Candidates \neq \emptyset$ **do**

 choose and remove a smallest $\kappa \in Candidates$;

if $\kappa_{\mathcal{F}}^{\pm} = \{A_1, A_2, \dots, A_n\}$ **then**

if κ contains no key in $Results$ **then**

$Results = Results \cup \{\kappa\}$;

end if

else

for all $A_i \notin \kappa_{\mathcal{F}}^{\pm}$ **do**

$\kappa_i = \kappa \cup \{A_i\}$;

$Candidates = Candidates \cup \{\kappa_i\}$;

end for

end if

end while

return $Results$;

How to Determine Keys

Finding all minimal keys

Find **all** minimal keys the relation R

| R | | | | |
|---|---|---|---|---|
| A | B | C | D | E |

with the functional dependencies

$$A, D \rightarrow B, D$$

$$B, D \rightarrow C$$

$$A \rightarrow E$$

$$C, D, E \rightarrow A$$

using the algorithm on the previous slide.

Determinants

Determinants (Non-trivial, minimal FDs)

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

- the FD $A_1, \dots, A_n \rightarrow B_1, \dots, B_m$ holds, and
- the **lhs is minimal**, i.e., whenever any A_i is removed then $A_1, \dots, A_{i-1}, A_{i+1}, A_n \rightarrow B_1, \dots, B_m$ does *not* hold, and
- the lhs and rhs are distinct, i.e., $\{A_1, \dots, A_n\} \neq \{B_1, \dots, B_m\}$.

$$\mathcal{F} = \left\{ \begin{array}{ll} \text{STUD_ID, EX_NO} & \rightarrow \text{POINTS} \\ & \text{EX_NO} \rightarrow \text{MAX_POINTS} \end{array} \right\}$$

Are the following determinants?

- POINTS, MAX_POINTS for POINTS, MAX_POINTS ? No
- EX_NO for POINTS, MAX_POINTS ? No
- STUD_ID, EX_NO for POINTS, MAX_POINTS ? Yes
- EX_NO, POINTS for POINTS, MAX_POINTS ? Yes

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

Usually a severe sign of **bad DB design** if a table contains an FD (encodes a partial function) that is **not implied by a key**.

INAME \rightarrow PHONE

| COURSES | | | |
|------------|------------------------|-------|-------|
| <u>CRN</u> | TITLE | INAME | PHONE |
| 22268 | Databases I | Grust | 7111 |
| 42232 | Functional Programming | Grust | 7111 |
| 31822 | Graph Theory | Klotz | 2418 |

This leads to

- **redundant storage of certain facts**
(here, phone numbers)
- **insert, update, deletion anomalies**

Consequences of Bad DB Design

Redundant storage is bad for several reasons:

- it **wastes storage space**
- difficult to ensure **integrity** when updating the database
 - all redundant copies need to be updated
 - **wastes time**, inefficient
- need for **additional constraints** to guarantee integrity
 - ensure that the redundant copies indeed agree
 - e.g. the constraint INAME → PHONE

Problem

General FDs are not supported by relational databases.

The solution is to transform FDs into **key constraints**.
This is what **DB normalization** tries to do.

Consequences of Bad DB Design

Update anomalies

- When a single value needs to be changed (e.g., a phone number), **multiple tuples** must be updated. This **complicates programs and updates takes longer**.
- Redundant copies potentially get **out of sync** and it is impossible/hard to identify the correct information.

Insertion anomalies

- The phone number of a new instructor cannot be inserted into the DB until it is known what course she/he will teach.
- Insertion anomalies arise when **unrelated concepts** are **stored together in a single table**.

Deletion anomalies

- When the last course of an instructor is deleted, his/her phone number is lost.

Boyce-Codd Normal Form

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

That is, for any FD $A_1, \dots, A_n \rightarrow B_1, \dots, B_m$ of R we have:

- $\{B_1, \dots, B_m\} \subseteq \{A_1, \dots, A_n\}$ (the FD is trivial), or
- $\{A_1, \dots, A_n\}$ contains a key of R .

The relation

COURSES (CRN, TITLE, INAME, PHONE)

with the FDs

CRN \rightarrow TITLE, INAME, PHONE

INAME \rightarrow PHONE

is **not in BCNF** because of the FD $\text{INAME} \rightarrow \text{PHONE}$:

- the FD is not trivial, and
- INAME is not a key

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

Boyce-Codd Normal Form: Examples

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):

CRN \rightarrow TITLE, WEEKDAY, TIME, ROOM
WEEKDAY, TIME, ROOM \rightarrow CRN

The keys of CLASS are

- { CRN }
- { WEEKDAY, TIME, ROOM }

Is the relation in BCNF?

- both FDs are implied by keys
(their left-hand sides even coincide with the keys)

Thus CLASS **is in BCNF**.

Boyce-Codd Normal Form: Examples

Consider the relation

PRODUCT (NO, NAME, PRICE)

and the following FDs:

| | | | | | |
|----|---|-------|-------------|---|------|
| NO | → | NAME | PRICE, NAME | → | NAME |
| NO | → | PRICE | NO, PRICE | → | NAME |

Is this relation 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 left-hand side of the last FD contains a key.

Thus the relation PRODUCT **is in BCNF**.

Boyce-Codd Normal Form

Alternative characterisation of Boyce-Codd Normal Form:

BCNF \iff every determinant is key

Advantages of Boyce-Codd Normal Form

If a relation R is in BCNF, then. . .

- Ensuring its key constraints automatically satisfies all FDs. Hence, no additional constraints are needed!
- The **anomalies** (update/insertion/deletion) **do not occur**.

Boyce-Codd Normal Form: Quiz

BCNF Quiz

1. Is the relation

RESULTS (STUD_ID, EX_NO, POINTS, MAX_POINTS)

with the following FDs in BCNF?

STUD_ID, EX_NO \rightarrow POINTS

EX_NO \rightarrow MAX_POINTS

2. Is the relation

INVOICE (INV_NO, DATE, AMOUNT, CUST_NO, CUST_NAME)

with the following FDs in BCNF?

INV_NO \rightarrow DATE, AMOUNT, CUST_NO

INV_NO, DATE \rightarrow CUST_NAME

CUST_NO \rightarrow CUST_NAME

DATE, AMOUNT \rightarrow DATE

Third Normal Form

A **key attribute** is an attribute that appears in a minimal key.

Minimality is important, otherwise all attributes are key attributes.

Assume that FDs with multiple attributes on rhs have been expanded. That is, every FD has a single attribute on the right-hand side.

Third Normal Form (3NF)

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

- $B \in \{A_1, \dots, A_n\}$ (the FD is trivial), or
- $\{A_1, \dots, A_n\}$ contains a key of R , or
- B is a **key attribute** of R .

The only difference with BCNF is the last condition.

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

Third Normal Form

In short, we can say:

BCNF \iff for every non-trivial FD:

- the left-hand side contains a key

3NF \iff for every non-trivial FD:

- the left-hand side contains a key, or
- the right-hand side is an attribute of a minimal key

Alternative characterisation of 3NF:

3NF \iff **every determinant of a non-key attribute is a key**

Third Normal Form Quiz

3NF vs BCNF

| BOOKINGS | | | |
|----------|------------|----------|------------|
| COURT | START_TIME | END_TIME | RATE |
| 1 | 9:30 | 11:00 | SAVER |
| 2 | 9:30 | 12:00 | PREMIMUM-A |
| 1 | 12:00 | 14:00 | STANDARD |

The table contains bookings for one day at a tennis club:

- there are courts 1 (hard court) and 2 (grass court)
- the rates are
 - SAVER for member bookings of court 1
 - STANDARD for non-member bookings of court 1
 - PREMIMUM-A for member bookings of court 2
 - PREMIMUM-B for non-member bookings of court 2

Quiz:

- Find a representative FDs set.
- Is the table in BCNF? Is the table in 3NF?

Splitting Relations

If a table R is not in BCNF, we can **split** it into two tables.

- the violating FD determines how to split

Table Decomposition

If the FD $A_1, \dots, A_n \rightarrow B_1, \dots, B_m$ violates BCNF:

- create a new relation $S(\underline{A_1}, \dots, \underline{A_n}, B_1, \dots, B_m)$ and
- remove B_1, \dots, B_m from the original relation R .

Splitting “along an FD”

The FD $\text{INAME} \rightarrow \text{PHONE}$ is the reason why table

COURSES (CRN, TITLE, INAME, PHONE)

violates BCNF because of $\text{INAME} \rightarrow \text{PHONE}$. We split into:

INSTRUCTORS (CRN, TITLE, INAME)

PHONEBOOK (INAME, PHONE)

Splitting Relations

It is important that this splitting transformation is **lossless**, i.e., that the original relation can be reconstructed by a join.

Reconstruction after split

Recall that we have split

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

into tables

```
INSTRUCTORS (CRN, TITLE, INAME)
```

```
PHONEBOOK (INAME, PHONE)
```

We can reconstruct the original table as follows:

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

Splitting Relations

When is a split lossless?

Decomposition Theorem

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

The join \bowtie 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

Lossless split condition satisfied

Recall that we have split

COURSES (CRN, TITLE, INAME, PHONE)

into tables

INSTRUCTORS (CRN, TITLE, INAME)

PHONEBOOK (INAME, PHONE)

The lossless split condition is satisfied since

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

and INAME is a key of the table PHONEBOOK.

All splits initiated by the **table decomposition 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

Not every lossless split is reasonable!

| STUDENTS | | |
|-------------|------------|-----------|
| <u>SSN</u> | FIRST_NAME | LAST_NAME |
| 111-22-3333 | John | Smith |
| 123-45-6789 | Maria | Brown |

Splitting STUDENTS into

STUD_FIRST (SSN, FIRST_NAME)

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

Splitting Relations

Lossless split guarantees that the resulting schema (after splitting) can represent all DB states that were possible before.

- we can translate states from the old into the new schema
- we may “simulate” the old schema via views

Lossless splits can lead to **more general schemas!**

- the new schema allows states which do not correspond to the state in the old schema

Recall that we have split

COURSES (CRN, TITLE, INAME, PHONE)

into tables

INSTRUCTORS (CRN, TITLE, INAME)

PHONEBOOK (INAME, PHONE)

We may now store instructors and phone numbers without any affiliation to courses.

Splitting Relations: Computable Columns

Although **computable columns** lead to violations of BCNF, splitting the relation is **not** the right solution.

E.g. AGE which is derivable from BIRTHDATE.

As a consequence we have a functional dependency:

$$\text{BIRTHDATE} \rightarrow \text{AGE}$$

A split would yield a relation:

$$R(\text{BIRTHDAY}, \text{AGE})$$

which would try to materialise 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 Functional Dependencies

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).

FD gets lost during decomposition

ADRESSES (STREET_ADDR, CITY, STATE, ZIP)

with functional dependencies

STREE_ADDR, CITY, STATE \rightarrow ZIP

ZIP \rightarrow STATE

The second FD violates BCNF and would lead to the split:

- ADDRESSES1 (STREET_ADDR, CITY, ZIP) and
- ADDRESSES2 (ZIP, STATE).

But now the first FD can no longer be expressed.



Preservation of Functional Dependencies

ADRESSES (STREET_ADDR, CITY, STATE, ZIP)

with functional dependencies

STREE_ADDR, CITY, STATE \rightarrow ZIP

ZIP \rightarrow STATE

Is the table in 3NF? Yes

- Most designers would not split the table since it is in 3NF.
- **Pro split:** if there are many addresses with the same ZIP code, there will be significant redundancy.
- **Contra split:** queries will involve more joins.

Whether or not to split depends on the intended application:

- A table of ZIP codes might be of interest on its own.

E.g. it this were a database for a mailing company.

3NF Synthesis Algorithm

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 \mathcal{F} as follows:

1. Replace every FD $\alpha \rightarrow B_1, \dots, B_m$ by $\alpha \rightarrow B_i, 1 \leq i \leq m$.
2. **Minimise:** For each $A_1, \dots, A_n \rightarrow B$ and each $i = 1, \dots, n$
 - Compute the cover $\{A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_n\}_{\mathcal{F}}^+$.
If the result contains B , replace \mathcal{F} by

$$\mathcal{F}' = (\mathcal{F} - \{A_1, \dots, A_n \rightarrow B\}) \\ \cup \{A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_n \rightarrow B\}$$

Keep repeating until all left-hand sides are minimal.

3. **Remove implied FDs:** For each FD $\alpha \rightarrow B$
 - Compute the cover $\alpha_{\mathcal{F}'}^+$, where $\mathcal{F}' = \mathcal{F} - \{\alpha \rightarrow B\}$.
If the cover contains B continue with $\mathcal{F} \leftarrow \mathcal{F}'$.

3NF Synthesis Algorithm

Compute the canonical set of FDs for

$A, B, C \rightarrow D, E$

$B \rightarrow C$

$B \rightarrow E$

$C \rightarrow E$

$C, D \rightarrow D, F$

3NF Synthesis Algorithm

3NF Synthesis Algorithm

Input: relation R and a set of FDs for R .

1. Compute a canonical (minimal) set of FDs \mathcal{F} .
2. For each left-hand side α of an FD in \mathcal{F} create a relation with attributes $\mathcal{A} = \alpha \cup \{B \mid \alpha \rightarrow B \in \mathcal{F}\}$.
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 minimal key of R .
4. For any two relations R_1, R_2 constructed in steps 2,3, if the schema of R_1 is contained in the schema R_2 , discard R_1 .

3NF Synthesis Algorithm: Example

Use the 3NF synthesis algorithm to normalise the relation

$R (A, B, C, D, E, F)$

with the following canonical functional dependencies:

$A \rightarrow D$

$B \rightarrow C$

$B \rightarrow D$

$D \rightarrow E$

Efficiency Considerations: BCNF vs 3NF

BCNF does not retain all FDs, therefore 3NF is popular.

- Database systems are good at checking **key** constraints, because they create an index on the key columns.

If we leave a table in 3NF (and not BCNF), we have non-key constraints. Namely those FDs that are not implied by keys.

Sometimes we can enforce non-key constraints as follows:

- create a **materialised view** that contains the non-key FD (a selection of the columns of the FD)
- define the key constraint on the materialised view
 - updates to the table will cause updates to the view through
 - constraint checking is index-based, hence efficient

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** by splitting it.
 - Normalization to BCNF might not preserve FDs.
 - Normalization to 3NF preserves FDs.

- 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.

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

The development of BCNF/3NF has been guided by a particular type of constraint: **functional dependencies**.

The goal of normalization into BCNF/3NF is to

- eliminate 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

Recall the Decomposition Theorem

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

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 that give a **necessary and sufficient** condition for lossless decomposition

MVDs lead to the **Fourth Normal Form (4NF)**.

Multivalued Dependencies

The following table shows for each employee:

- knowledge of programming languages
- knowledge of programming DBMSs

| EMP_KNOWLEDGE | | |
|---------------|------------------|-------------|
| <u>ENAME</u> | <u>PROG_LANG</u> | <u>DBMS</u> |
| John Smith | C | Oracle |
| John Smith | C | DB2 |
| John Smith | C++ | Oracle |
| John Smith | C++ | DB2 |
| Maria Brown | Prolog | PostgreSQL |
| Maria Brown | Java | PostgreSQL |

- There are no non-trivial functional dependencies.
- The table is in **BCNF**.

Nevertheless, there is **redundant information**.

Multivalued Dependencies

The table contains redundant data & must be split.

| EMP_LANG | |
|--------------|------------------|
| <u>ENAME</u> | <u>PROG_LANG</u> |
| John Smith | C |
| John Smith | C++ |
| Maria Brown | Prolog |
| Maria Brown | Java |

| EMP_DBMS | |
|--------------|-------------|
| <u>ENAME</u> | <u>DBMS</u> |
| John Smith | Oracle |
| John Smith | DB2 |
| Maria Brown | PostgreSQL |

Note: table may only be decomposed if PROG_LANG and DBMS are **independent**; otherwise **loss of information**.

E.g. it may not be decomposed if the semantics of the table is that the employee knows the interface between the language and the database.

Multivalued Dependencies

The **multivalued dependency (MVD)**

ENAME \twoheadrightarrow PROG_LANG

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

| 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 |

That is, the table contains an

embedded function from ENAME to **sets of** PROG_LANG

Multivalued Dependencies

Formally, $ENAME \twoheadrightarrow PROG_LANG$ holds if: whenever two tuples agree on $ENAME$, one can exchange their $PROG_LANG$ values and the resulting tuples are in the same table.

From the two table rows

| <u>ENAME</u> | <u>PROG_LANG</u> | <u>DBMS</u> |
|--------------|------------------|-------------|
| 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:

| <u>ENAME</u> | <u>PROG_LANG</u> | <u>DBMS</u> |
|--------------|------------------|-------------|
| 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

Multivalued Dependency

A multivalued dependency (MVD)

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

is satisfied in a DB state I if and only if

- for all tuples t, u in $I(R)$ with $t.A_i = u.A_i$, $1 \leq i \leq n$, there are two further tuples t', u' in $I(R)$ such that
 1. t' agrees with t except that $t'.B_i = u.B_i$, $1 \leq i \leq m$, and
 2. u' agrees with u except that $u'.B_i = t.B_i$, $1 \leq i \leq m$.

The condition means that the values of the B_i are swapped:

| | | | |
|-----|---|------|---|
| t | $a_1, \dots, a_n, b_1, \dots, b_m, c_1, \dots, c_k$ | t' | $a_1, \dots, a_n, b'_1, \dots, b'_m, c_1, \dots, c_k$ |
| u | $a_1, \dots, a_n, b'_1, \dots, b'_m, c'_1, \dots, c'_k$ | u' | $a_1, \dots, a_n, b_1, \dots, b_m, c'_1, \dots, c'_k$ |

Multivalued Dependencies

Multivalued dependencies always **come in pairs!**

If $\text{ENAME} \twoheadrightarrow \text{PROG_LANG}$ holds, then $\text{ENAME} \twoheadrightarrow \text{DBMS}$ is automatically satisfied.

More general:

For a relation $R(A_1, \dots, A_n, B_1, \dots, B_m, C_1, \dots, C_k)$, the following multivalued dependencies are equivalent

- $A_1, \dots, A_n \twoheadrightarrow B_1, \dots, B_m$
- $A_1, \dots, A_n \twoheadrightarrow C_1, \dots, C_k$

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

If the FD $A_1, \dots, A_n \rightarrow B_1, \dots, B_m$ holds, the corresponding MVD

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

is trivially satisfied.

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

Deduction rules to derive all implied FDs/MVDs

- The three Armstrong Axioms for FDs.
- If $\alpha \twoheadrightarrow \beta$ then $\alpha \twoheadrightarrow \gamma$, where γ are all remaining columns.
- 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 $\alpha \rightarrow \beta$, then $\alpha \twoheadrightarrow \beta$.
- If $\alpha \twoheadrightarrow \beta$ and $\beta' \subseteq \beta$ and there is γ with $\gamma \cap \beta = \emptyset$ and $\gamma \rightarrow \beta'$, then $\alpha \rightarrow \beta'$.

Fourth Normal Form

Fourth Normal Form (4NF)

A relation is in **Fourth Normal Form (4NF)** if every MVD

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

is

- either trivial, or
- implied by a key.

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

Since every FD is also an MVD, 4NF is stronger than BCNF.

That is, if a relation is in 4NF, it is automatically in BCNF.

However, it is not very common that 4NF is violated, but BCNF is not.

Fourth Normal Form

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.

Other Constraints

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 |

Using keys to enforce other constraints

The constraint is not an FD, MVD, or JD:

“Each question can only have one correct answer.”

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

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

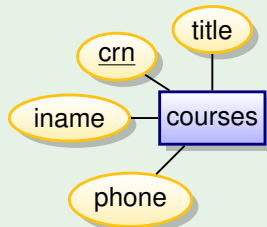
If a “good” ER schema is transformed into the relational model, the result will **satisfy all normal forms** (4NF, BCNF, 3NF).

- 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.

FDs in the ER model

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



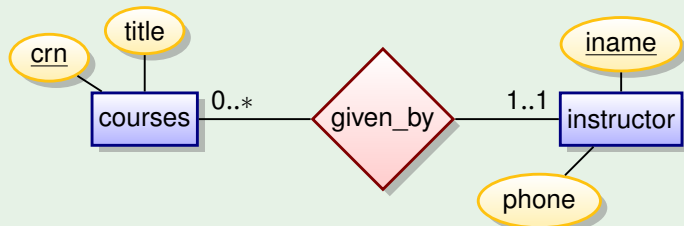
- Obviously, the FD $\text{iname} \rightarrow \text{phone}$ leads to a violation of BCNF in the resulting table for entity Course.
- **Also in the ER model, FDs between attributes of an entity should be implied by a key constraint.**

Examples

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

ER entity split

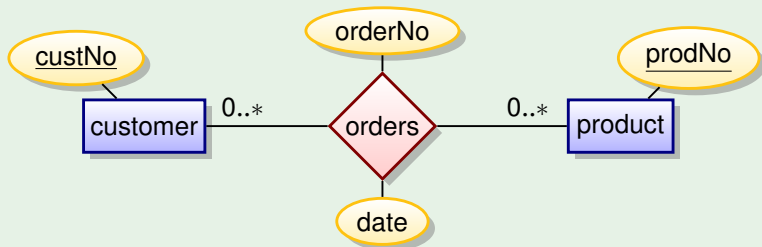
In this case, the instructor is an independent entity:



Examples

Functional dependencies between **attributes of a relationship** always violate BCNF.

Violation of BCNF on the ER level



The FD $\text{orderNo} \rightarrow \text{date}$ violates BCNF.

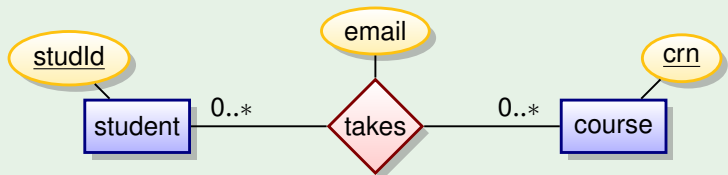
- The key of the table corresponding to the relationship “orders” consists of the attributes CustNo, ProdNo.

This shows that the concept “order” is an independent entity.

Examples

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

Questionable attribute placement

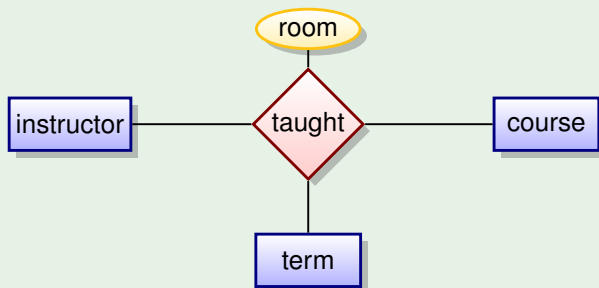


- The relationship is translated into
TAKES (STUD_ID, CRN, EMAIL)
- Then the FD $STUD_ID \rightarrow EMAIL$ violates BCNF.
- Obviously, email should be an attribute of Student.

Examples

If an attribute of a ternary relationship depends only on two of the entities, this violates BCNF.

Ternary relationship



If every course is taught only once per term, then attribute room depends only on term and course (but not instructor).

Then the FD $TERM, COURSE \rightarrow ROOM$ violates BCNF.

Normalization: Summary

Relational normalization is about:

- **Avoiding redundancy.**
 - **Storing separate facts (functions) separately.**
 - **Transforming general integrity constraints** 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).

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

Denormalization is the process of **adding redundant columns** to the database in order to **improve performance**.

Redundant data storage

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.

| COURSES | | | |
|------------|-------|-------|-------|
| <u>CRN</u> | TITLE | INAME | PHONE |

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

Denormalization

- 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 may not only be used to avoid joins:
 - Complete **separate, redundant tables** may be created (increasing the potential for parallel operations).
 - Columns may be added which **aggregate** information in other columns/rows.

Relational Normal Forms: Objectives

After completing this chapter, you should be able to

- work with **functional dependencies** (FDs),
 - define what they are
 - detect them in database schemas
 - decide implication, determine keys
- explain insert, update, and delete **anomalies**,
- understand, explain and use **BCNF**
 - test a given relation for BCNF, and
 - transform a relation into BCNF
- understand, explain and use **3NF**
 - test a given relation for 3NF, and
 - transform a relation into 3NF
- understand, explain **MVDs** and **4NF**
- detect **normal form violations** on the level of ER,
- explain when and how to **denormalize** a DB schema