Datenbanken II A: DB-Entwurf

Chapter 8: Relational Normal Forms, Part I

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2024/25

http://www.informatik.uni-halle.de/~brass/dd24/

Objectives

After completing this chapter, you should be able to:

- work with functional dependencies (FDs),

  Define them, detect them in applications, decide whether an FD is implied by other FDs, determine a key based on FDs.

- explain insert, update, and delete anomalies.
- explain BCNF, check a given relation for BCNF, and transform a relation into BCNF.
- detect normal form violations on the ER-level.

Contents

Introduction (1)

- Relational database design theory is based mainly on a class of constraints called "Functional Dependencies" (FDs). FDs are a generalization of keys.
- This theory defines when a relation is in a certain normal form (e.g. Third Normal Form, 3NF) for a given set of FDs.
- It is usually bad if a schema contains relations that violate the conditions of a normal form.
- However, there are exceptions and tradeoffs.

  In databases courses for beginners, one usually requires that all relations must be in 3NF. In more advanced database courses (like this one), it is possible to discuss whether the negative consequences of violating a normal form really occur in the specific application.

Introduction (2)

- If a normal form is violated, data is stored redundantly, and information about different concepts is intermixed. E.g. consider the following table:

COURSES			
CRN	TITLE	INAME	PHONE
22268	DB Management	Brass	9404
42232	Data Structures	Brass	9404
31822	Client-Server	Spring	9429

- The phone number of "Brass" is stored two times. In general, the phone number of an instructor will be stored once for every course he/she teaches.

  The concepts "course" and "instructor" are different entities in the real world and should be stored in separate tables.

Introduction (3)

- Third Normal Form (3NF) is today considered part of the general database education.
- Boyce-Codd Normal Form (BCNF) is a little bit stronger, easier to define, and better matches our intuition.

  BCNF should really replace 3NF. The only problem is that in rare circumstances, a relation cannot be transformed into BCNF with the FDs preserved. However, every relation can be transformed into 3NF with the FDs preserved. However, in my view, the preservation of FDs has no practical advantage as long as the FDs are not keys (see below). In these problematic examples, there is simply no good solution.
- In short, BCNF means that all functional dependencies are already enforced by keys.

Introduction (4)

- Normalization algorithms can construct tables from a set of attributes and a set of functional dependencies.

  So in theory, database design can be done by only collecting attributes and FDs. No ER-design is needed.

- In practice, normalization is only used as an additional check.

  E.g. one does the ER-design, translates the ER-schema into the relational model, and check the resulting tables for BCNF.

- In discussions about database designs, the violation of a normal form is a clear and objective problem.

  If one really wants to keep this violation, one is automatically in a "defense" position and must have good arguments why this is one of the rare cases where the violation is really the better solution.

Introduction (5)

- When an Entity-Relationship design is done well, the resulting tables will automatically be in BCNF (even 4NF).

  If the resulting tables are not in BCNF, one must go back to the ER-design, and correct the normal form violation there.

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

  There is a normal form theory for the ER-model, too, but it is quite complicated. It is easier to understand relational normal forms and combine them with the ER-to-relational translation.

Theory vs. Intuition

- Once one understood normal forms, the intuition should be sufficient in 97% of the cases.

  It will all seem very obvious. But in order to develop the intuition, one needs the theory. Good students showed me non-normalized designs.

- But in the remaining difficult 3% of the cases, it might be necessary to apply the formal definitions.

  In order to convince other people, it is also better if one can argue with generally accepted formal definitions.

- Even Codd needed three tries to get the normal form definition right (2NF, 3NF, BCNF).

  To be fair, 2NF and 3NF were defined in the same paper.

Contents

First Normal Form (1)

- First Normal Form only requires that all table entries are atomic (not lists, sets, records, relations).
- Today, the relational model is already defined in this way. All further normal forms assume that the tables are in 1NF (First Normal Form).
- Some modern database management systems allow structured attributes. Such systems are call NF^2 systems ("Non First Normal Form").

  Structured attributes are also usually considered a requirement for an object-relational DBMS.

First Normal Form (2)

- Example of an NF^2-relation (not 1NF):

COURSES				
CRN	TITLE	TAUGHT_BY	STUDENTS	
			FNAME	LNAME
22332	DB Management	Brass	John	Smith
			Ann	Miller
31864	Client-Server	Spring	Ann	Miller

- 1NF doesn't really belong in this chapter.

  It has nothing to do with functional dependencies.

First Normal Form (3)

- Some authors feel that 1NF is already violated if there are string-valued attributes that have an inner structure (i.e. which could be further decomposed).
- Simple example: Last name and first name are put together in one attribute, separated by a comma.

  This means that one will have to use string operations in some queries.
- Really bad example: The CRNs of all courses a student is registered for are put into an attribute of the students table (separated by spaces).

  Some interesting queries will need real programming now.

First Normal Form (4)

- Some practical DB designers argue that 1NF is already violated if there are repeated attributes like `DEGREE1`, `DEGREE2`, `DEGREE3` in in the instructors table.
- Normally, such attributes make queries and updates more difficult, and should be avoided.

  And is there any guarantee that there cannot be an instructor with four degrees? It is better to have a separate degree table with one row for each degree (together with the key of the instructor).
- However, formally, this is no violation of First Normal Form.

Contents

Functional Dependencies (1)

- An example of a functional dependency (FD) is

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

- It means that whenever two rows agree in the instructor name column INAME, they must also have the same value in the column PHONE:

COURSES			
CRN	TITLE	INAME	PHONE
22268	DB Management	Brass	9404
42232	Data Structures	Brass	9404
31822	Client-Server	Spring	9429

Functional Dependencies (2)

- The reason for the validity of INAME $\longrightarrow$ PHONE is that the contact phone number for the course depends only on the instructor, not on the other course data.
- The FD is read as: "INAME (functionally, uniquely) determines PHONE".
- One says also that INAME is a determinant for PHONE.

  Saying that A is a determinant for B is slightly stronger than the FD $A \longrightarrow B$, see below (A must be minimal and $\neq B$).

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

Functional Dependencies (3)

- A key uniquely determines every attribute, i.e. the FDs CRN $\longrightarrow$ TITLE, CRN $\longrightarrow$ INAME, CRN $\longrightarrow$ PHONE hold, too.

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

- E.g. the FD "INAME $\longrightarrow$ TITLE" is not satisfied. There are rows with the same INAME, but different TITLE:

COURSES			
CRN	TITLE	INAME	PHONE
22268	DB Management	Brass	9404
42232	Data Structures	Brass	9404
31822	Client-Server	Spring	9429

Functional Dependencies (4)

- In general, an FD has the form

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

- Sequence and multiplicity of attributes in an FD are unimportant, since both sides are formally sets of attributes: $\{A_1, \ldots, A_n\} \longrightarrow \{B_1, \ldots, B_m\}$.
- In discussing FDs, the focus is on a single relation R. All attributes A_i, B_i are from this relation.

Functional Dependencies (5)

- The FD $A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$ holds for a relation R in a database state $\mathcal{I}$ if and only if for all tuples $t, u \in \mathcal{I}(R)$:

  If $t.A_1 = u.A_1$ and $\ldots$ and $t.A_n = u.A_n$,
  then $t.B_1 = u.B_1$ and $\ldots$ and $t.B_m = u.B_m$.

- I.e. $A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$ holds if there no two rows contain the same values in all columns A_i, but different values in one of the columns B_j.

Functional Dependencies (6)

- An FD with m attributes on the right hand side

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

is equivalent to the m FDs:

$$\begin{array}{lcl} A_1, \ldots, A_n & \longrightarrow & B_1 \\ \vdots & \vdots & \vdots \\ A_1, \ldots, A_n & \longrightarrow & B_m. \end{array}$$

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

  However, sometimes it is a useful abbreviation to put multiple columns on the right hand side.

FDs are Constraints (1)

- For database design, only FDs are interesting that must hold in all possible database states.
- I.e. FDs are constraints (like keys).
- E.g. in the example state for "COURSES", also the FD "TITLE $\longrightarrow$ CRN" holds, because no two courses have the same title.
- But probably this is not true in general, only in this small example state.

  It would be important for the design to find out whether this holds in general, i.e. there can never be two sessions of the same course.

FDs are Constraints (2)

- There are tools for analyzing example data for possible FDs, and then asking the designer whether these FDs hold in general.
- If an FD (or any constraint) does not hold in the example state, it certainly cannot hold in general.
- If one wants to use normal forms, one needs to collect all FDs that hold in general. This is a design task, it cannot be done automatically.

  Actually, only a representative subset is needed (that implies the remaining ones, see below). Of course, such tools could help.

FDs vs. Keys (1)

- FDs are a generalization of keys: $A_1, \ldots, A_n$ is a key of $R(A_1, \ldots, A_n, B_1, \ldots, B_m)$ if and only if the FD "$A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$" holds.

  Under the assumption that there are no duplicate rows. Two distinct rows that are identical in every attribute would not violate the FD, but they would violate the key. In theory, this cannot happen, because relations are sets of tuples, and tuples are defined only by their attribute values. In practice, SQL permits two identical rows in a table as long as one did not define a key (therefore, always define a key).

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

FDs vs. Keys (2)

- Conversely, FDs can be explained with keys. FDs are keys for some columns, e.g. "INAME $\longrightarrow$ PHONE" means that INAME is a key of $\pi_{\text{INAME, PHONE}}(\text{COURSES})$:

INAME	PHONE
Brass	9404
Spring	9429

I.e. one removes all columns from the table that do not appear in the FD and then eliminates duplicate rows. $A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$ is satisfied in the original table if the projection result has no two rows with the same A_i-values (these would have to differ in a B_j).

- So an FD describes a key for part of the attributes. The goal of normalization is to make it a real key.

FDs Describe Functions

- A key means that a relation (a table) really describes a partial function (with values for the key attributes as inputs and values for the other attributes as outputs).
- In the example, another function is represented in the table: It maps instructors' names to phone numbers.

  Partial function, because it is not defined for arbitrary strings.
- Sometimes there are functions which can be computed: E.g. if the date of birth and the age are stored in the same table. Then of course the FD "$\texttt{BIRTHDATE} \longrightarrow \texttt{AGE}$" holds.
- But we actually know much more than what the FD expresses: we know the exact formula.
- Therefore, normalization theory does not help in this case. AGE is simply redundant.

Example (1)

- The following table is used to store information about 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
Date	1	SQL Standard	Addison-W.	0201964260
Darwen	2	SQL Standard	Addison-W.	0201964260
Meier	1	Theory of RDB	Comp.Sc.P.	0914894420

A book can have multiple authors. There is one row for every author of a book. "NO" is used to keep track of their sequence (it is not necessarily alphabetic, e.g. Date/Darwen).

Example (2)

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

$$\texttt{ISBN} \longrightarrow \texttt{TITLE, PUBLISHER}$$

- Equivalently, one can use the two FDs:

$$\texttt{ISBN} \longrightarrow \texttt{TITLE}$$
$$\texttt{ISBN} \longrightarrow \texttt{PUBLISHER}$$

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

$$\texttt{ISBN} \longrightarrow \texttt{AUTHOR} \qquad \textbf{[Not Satisfied]}$$

  In the same way, ISBN does not determine NO.

Example (3)

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

  $$\texttt{AUTHOR} \longrightarrow \texttt{TITLE}$$ **[Not in general true]**

- It is possible that there are books with the same title but different authors and different publishers.

  E.g. there are several unrelated books called "Database Management". So TITLE determines none of the other attributes.

- Books can have the same author and title, but different ISBNs (paperback and hardcover edition).

Example (4)

- At every position, there can be only one author:

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

- At first, it seems impossible that the same author appears in two different positions in the author list of the same book:

$$\texttt{ISBN, AUTHOR} \longrightarrow \texttt{NO}$$ **[Questionable]**

This would be violated if there is a book from Smith & Smith.

Example (5)

- Only unquestionable conditions should be used as constraints.

  The table structure depends on the FDs. Therefore, if it should turn out later that an FD is too restrictive, it will normally not suffice to simply remove a constraint (e.g. a key) with an `ALTER TABLE` statement. Instead, one has to create new tables, copy data, and change application programs. If conversely, new FDs are discovered later, the old tables can still be used, but they violate a normal form.

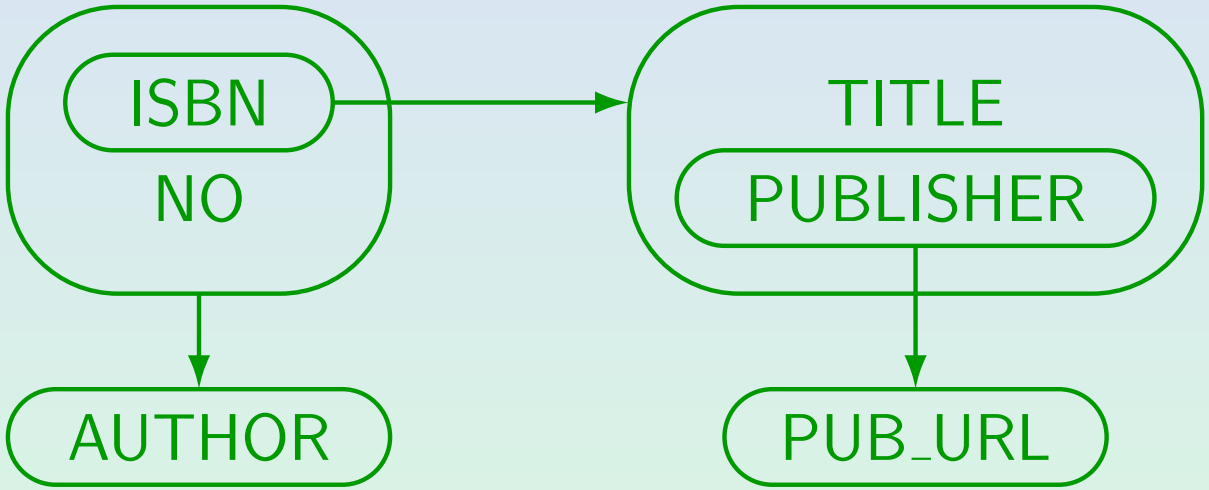
- One might also be tempted to assume this FD:

  $\texttt{PUBLISHER, TITLE, NO} \longrightarrow \texttt{AUTHOR}$ **[Questionable]**

  It is probably unlikely, but if e.g. in a new edition the author sequence changes, one is in trouble.

Example (6)

- A set of FDs can be displayed as a "hypergraph":



In a hypergraph the edges are between sets of nodes, not only between two nodes as in a standard graph.

A publisher URL was added to make the example more interesting.

Exercise (1)

- Consider a one-relation version of the homework grades DB:

HOMEWORK_RESULTS					
STUD_ID	FIRST	LAST	EX_NO	POINTS	MAX_PT
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)?

Exercise (2)

- What does the FD "LAST $\longrightarrow$ FIRST" mean?
  - ☐ If students have the same first name, they must have the same last name.
  - ☐ There cannot be siblings or other students with the same last name, different first name.
  - ☐ There cannot be different students with the same name (first and last).
- Name one FD which holds in this state, but not in general.
- Draw the hypergraph displaying the FDs.

Contents

Implication of FDs (1)

- CRN$\longrightarrow$PHONE is nothing new when one knows already CRN$\longrightarrow$INAME and INAME$\longrightarrow$PHONE.

  Whenever $A \longrightarrow B$ and $B \longrightarrow C$ are satisfied, $A \longrightarrow C$ automatically holds.

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

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

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

  α and β stand here for sets of attributes/columns. Note that this notion of implication is not specific to FDs, the same definition is used for general constraints.

Implication of FDs (2)

- One is normally not interested in all FDs which hold, but only in a representative set that implies all other FDs.
- Implied dependencies can be computed by applying the Armstrong Axioms:
  - If $\beta \subseteq \alpha$, then $\alpha \longrightarrow \beta$ trivially holds (Reflexivity).
  - If $\alpha \longrightarrow \beta$, then $\alpha \cup \gamma \longrightarrow \beta \cup \gamma$ (Augmentation).
  - If $\alpha \longrightarrow \beta$ and $\beta \longrightarrow \gamma$, then $\alpha \longrightarrow \gamma$ (Transitivity).

Implication of FDs (3)

- However, a simpler way to check whether $\alpha \longrightarrow \beta$ is implied by given FDs is to compute first the cover α^+ of α and then to check whether $\beta \subseteq \alpha^+$.
- 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 given FDs).

$$\alpha^+ := \{B \mid \text{The given FDs imply } \alpha \longrightarrow B\}.$$

The cover α^+ depends on the given FDs, although the set of FDs is not explicitly shown in the usual notation α^+. If necessary, write $\alpha^+_{\mathcal{F}}$.

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

Implication of FDs (4)

- The cover is computed as follows:

Input: α (Set of attributes)
$\alpha_1 \longrightarrow \beta_1, \ldots, \alpha_n \longrightarrow \beta_n$ (Set of FDs)

Output: α^+ (Set of attributes, Cover of α)

Method:
$x := \alpha$;
while x did change **do**
  for each given FD $\alpha_i \longrightarrow \beta_i$ **do**
    if $\alpha_i \subseteq x$ **then**
      $x := x \cup \beta_i$;
output x;

Implication of FDs (5)

- Consider the following FDs:

$$\texttt{ISBN} \longrightarrow \texttt{TITLE, PUBLISHER}$$
$$\texttt{ISBN, NO} \longrightarrow \texttt{AUTHOR}$$
$$\texttt{PUBLISHER} \longrightarrow \texttt{PUB\_URL}$$

- Suppose we want to compute $\{\texttt{ISBN}\}^+$.
- We start with $x = \{\texttt{ISBN}\}$.

  x is the set of attributes for which we know that there can be only a single value. We start with the assumption that for the given attributes in α, i.e. ISBN, there is only one value. Then the cover α^+ is the set of attributes for which we can derive under this assumption that their value is uniquely determined (using the given FDs).

Implication of FDs (6)

- The first of the given FDs, namely

  $$\texttt{ISBN} \longrightarrow \texttt{TITLE, PUBLISHER}$$

  has a left hand side that is entirely contained in the current set x (actually, it happens to be the same).

  I.e. there is a unique value for these attributes. Then the FD means that also for the attributes on the right hand side have a unique value.

- Therefore, we can extend x by the attributes on the right hand side of this FD, i.e. TITLE, and PUBLISHER:

  $$x = \{\texttt{ISBN, TITLE, PUBLISHER}\}.$$

Implication of FDs (7)

- Now the third of the FDs, namely

  $$\texttt{PUBLISHER} \longrightarrow \texttt{PUB\_URL}$$

  is applicable: Its left hand side is contained in x.
- Therefore, we can add the right hand side of this FD to x and get

  $$x = \{\texttt{ISBN}, \texttt{TITLE}, \texttt{PUBLISHER}, \texttt{PUB\_URL}\}.$$
- The last FD, namely

  $$\texttt{ISBN, NO} \longrightarrow \texttt{AUTHOR}$$

  is still not applicable, because NO is missing in x.

Implication of FDs (8)

- After checking again that there is no way to extend the set x any further with the given FDs, the algorithm terminates and prints

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

- From this, we can conclude that the given FDs imply e.g. $\texttt{ISBN} \longrightarrow \texttt{PUB\_URL}$.

- In the same way, one can compute e.g. the cover of $\{\texttt{ISBN}, \texttt{NO}\}$. It is the entire set of attributes.

  This means that $\{\texttt{ISBN}, \texttt{NO}\}$ is a key of the relation, see next slide.

Contents

How to Determine Keys (1)

- Given a set of FDs (and the set of all attributes $\mathcal{A}$ of a relation), one can determine all possible keys for that relation.

  Again, one must assume that duplicate rows are excluded.

- $\mathcal{K} \subseteq \mathcal{A}$ is a key if and only if $\mathcal{K}^+ = \mathcal{A}$.

- Normally, one is only interested in minimal keys.

  The superset of a key is again a key, e.g. if $\{\texttt{ISBN}, \texttt{NO}\}$ uniquely identifies all other attributes, this automatically holds also for $\{\texttt{ISBN}, \texttt{NO}, \texttt{TITLE}\}$. Therefore, one usually requires in addition that every $A \in \mathcal{K}$ must be necessary, i.e. $(\mathcal{K} \setminus \{A\})^+ \neq \mathcal{A}$. Most authors make the minimality requirement part of the key definition. But then a key is not only a constraint, it also says that stronger constraints do not hold.

How to Determine Keys (2)

Algorithm to compute one minimal key:

```
(1)  K := A;    /* Certainly a key (non-minimal) */
(2)  foreach A ∈ A do
(3)      if (K \ {A})+ = A then
(4)          K := K \ {A};  /* A is not necessary */
(5)  print K;
```

- One starts with the set of all attributes and tries to remove each attribute in succession.

  The sequence, in which the attributes are checked, determines which of several keys is computed.

How to Determine Keys (3)

Correctness:

- The construction obviously ensures $\mathcal{K}^+ = \mathcal{A}$.

  This holds at the beginning, and $\mathcal{K}$ is only changed when this condition holds for the new value of $\mathcal{K}$.

- An important property of the attribute closure is

$$\mathcal{X} \subseteq \mathcal{Y} \implies \mathcal{X}^+ \subseteq \mathcal{Y}^+.$$

- Thus, if $(\mathcal{K} \setminus \{A\})^+ \neq \mathcal{A}$ at some point in the computation, this also holds for any later value of $\mathcal{K}$ (which will be a subset of the current value).

How to Determine Keys (4)

Algorithm `minimize`($\mathcal{S}$): (global Variables $\mathcal{A}$ and $\mathcal{F}$)

```
(1)  K := S;
(2)  assert S+_F = A;  /* S must be a key */
(3)  foreach A ∈ A do
(4)      if (K \ {A})+_F = A then
(5)          K := K \ {A};  /* A is not necessary */
(6)  return K;  /* Minimal Key */
```

- Slight generalization of the algorithm from Slide 46: Instead of starting with all attributes, we can start with any (not necessarily minimal) key $\mathcal{S}$.

  The algorithm then returns a subset $\mathcal{K} \subseteq \mathcal{S}$ which is a minimal key.

How to Determine Keys (5)

Theorem:

- Let $\mathcal{A}$ be the set of all attributes, $\mathcal{F}$ be a set of functional dependencies, and $\mathcal{K}_1, \ldots, \mathcal{K}_n$, $n \geq 1$ be a set of minimal keys with respect to $\mathcal{A}$ and $\mathcal{F}$.
- There is an additional minimal key $\mathcal{K}'$ (wrt $\mathcal{A}$, $\mathcal{F}$), not contained in $\{\mathcal{K}_1, \ldots, \mathcal{K}_n\}$ if and only if there is an $i \in \{1, \ldots, n\}$ and an FD $\alpha \longrightarrow \beta \in \mathcal{F}$ such that $\alpha \cup (\mathcal{K}_i \setminus \beta)$ does not include any key in $\{\mathcal{K}_1, \ldots, \mathcal{K}_n\}$.

  Furthermore, the additional key $\mathcal{K}'$ is a subset of $\alpha \cup (\mathcal{K}_i \setminus \beta)$.

  This theorem is a slightly rephrased version of Lemma 4 in:
  Cláudio L. Lucchesi, Sylvia L. Osborn: Candidate keys for relations. Journal of Computer and System Sciences 17:2, Oct. 1978, 270–279.

How to Determine Keys (6)

Explanation / Proof Sketch:

- Given a key $\mathcal{K}_i$, and a functional dependency $\alpha \longrightarrow \beta$, the set $\alpha \cup (\mathcal{K}_i \setminus \beta)$ is a key, too (not necessarily minimal).

  The first step of the iteration of the computation of the attribute closure adds β, then we have a superset of $\mathcal{K}_i$ (not necessarily proper).

- By applying the minimization algorithm (see Slide 48), we can construct a subset, which is a minimal key.

- If before the minimization, none of the known keys was a subset, this obviously holds also after the minimization, i.e. the constructed key is new.

How to Determine Keys (7)

Explanation / Proof Sketch (continued):

- The important point is that the converse holds, too:
  If this construction does not give any new key, then there is no further key (completeness).

  Let $\mathcal{K}'$ be a key such that $\mathcal{K}_i \not\subseteq \mathcal{K}'$ for all $i \in \{1, \ldots, n\}$. Let $\mathcal{M}$ be a maximal superset of $\mathcal{K}'$ of which no $\mathcal{K}_i$ is a subset (i.e. one adds attributes as long as no $\mathcal{K}_i$ is fully contained, formally: for every attribute $A \notin \mathcal{M}$ there is a $\mathcal{K}_i$ such that $\mathcal{K}_i \subseteq \mathcal{M} \cup \{A\}$).

  $\mathcal{M} \neq \mathcal{A}$, because it contains no $\mathcal{K}_i$, and there is at least one. Because $\mathcal{M}$ is a key, $\mathcal{M}^+ = \mathcal{A}$. Thus, there must be $\alpha \longrightarrow \beta$ with $\alpha \subseteq \mathcal{M}$ and $\beta \not\subseteq \mathcal{M}$. Because of the maximality of $\mathcal{M}$ there is $i_0 \in \{1, \ldots, n\}$ with $\mathcal{K}_{i_0} \subseteq \mathcal{M} \cup \beta$. Then $\mathcal{K}_{i_0} \setminus \beta \subseteq \mathcal{M}$, and since $\alpha \subseteq \mathcal{M}$, it follows that $\alpha \cup (\mathcal{K}_{i_0} \setminus \beta) \subseteq \mathcal{M}$. But since $\mathcal{K}_i \not\subseteq \mathcal{M}$ for all $i \in \{1, \ldots, n\}$, this holds all the more for the subset: $\mathcal{K}_i \not\subseteq \alpha \cup (\mathcal{K}_{i_0} \setminus \beta)$.

How to Determine Keys (8)

Algorithm to compute all minimal keys:

```
(1)  K_1 := Key computed with Alg. on Slide 46;
(2)  n := 1; /* Number of Keys known so far */
(3)  i := 0; /* Number of keys processed so far */
(4)  while i < n do
(5)      foreach α ⟶ β ∈ F do
(6)          S := α ∪ (K_i \ β);
(7)          found := false;
(8)          for j := 1 to n do
(9)              if K_j ⊆ S then found := true;
(10)         if not found then
(11)             n := n + 1;
(12)             K_n := minimize(S);
```

How to Determine Keys (9)

- In general, there can be exponentially many keys.

  Consider $\mathcal{A} = \{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_n\}$
  and $\{\mathcal{F} = \{A_i \longrightarrow B_i \mid i = 1, \ldots, n\} \cup \{B_i \longrightarrow A_i \mid i = 1, \ldots, n\}\}$.

- The question whether a given attribute is contained in any minimal key, is NP-complete.

  Such attributes are called “prime attributes” (or “key attributes”).

- The question whether there is a key consisting of at most i attributes is NP-complete.

How to Determine Keys (10)

- One can construct a key also in a less formal way.
- So one starts with the set of required attributes (that do not appear on any right side).

  In the above example one is already done: `ISBN` and `NO` appear at no right side, but their cover is the set of all attributes.

- If the required attributes do not already form a key, one adds attributes: The left hand side of an FD or directly one of the missing attributes.

  Only make sure at the end that the set is minimal. If it contains attributes that are functionally determined by other attributes in the set, remove them.

Exercise (1)

- The following relation is used for storing homework results:

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

Exercise (2)

- It is known that these FDs hold:

  $\texttt{STUD\_ID}, \texttt{EX\_NO} \longrightarrow \texttt{POINTS}$

  $\texttt{EX\_NO} \longrightarrow \texttt{MAX\_POINTS}$

- Do these FDs imply the following FD?

  $\texttt{STUD\_ID}, \texttt{EX\_NO} \longrightarrow \texttt{MAX\_POINTS}$

- Does this FD imply "$\texttt{EX\_NO} \longrightarrow \texttt{MAX\_POINTS}$"?

  So which of the two is the stronger restriction?

- Determine a key of the relation RESULTS.

Determinants

- $A_1, \ldots, A_n$ is called a determinant for a set of attributes $B_1, \ldots, B_m$ if and only if
  - The FD $A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$ holds.
  - The left hand side is minimal, i.e. whenever an attribute A_i is removed from the left hand side
    $$A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \longrightarrow B_1, \ldots, B_m$$
    does not hold.
  - Left and right hand side are distinct, i.e. $\{A_1, \ldots, A_n\} \neq \{B_1, \ldots, B_m\}$.

    Together with the minimality requirement this excludes trivial FDs.

Contents

Problems (1)

- It is usually bad if a tables contains an FD that is not implied by a key (e.g. INAME$\longrightarrow$PHONE).
- Among other problems, this leads to the redundant storage of certain facts. E.g. in the example table, the phone number of "Brass" is stored two times:

COURSES			
CRN	TITLE	INAME	PHONE
22268	DB Management	Brass	9404
42232	Data Structures	Brass	9404
31822	Client-Server	Spring	9429

Problems (2)

- If a schema contains redundant data, a constraint must be specified to ensure that the copies of the information agree.
- In the example, this constraint is precisely the FD "$\texttt{INAME} \longrightarrow \texttt{PHONE}$".
- But FDs are not supported as one of the standard constraints of the relational model.
- Therefore, one should try to avoid FDs and transform them into key constraints. This is what normalization does.

Problems (3)

- Redundant data leads to the Update Anomalies: When a single fact needs to be changed, e.g. the phone number of "Brass", multiple tuples must be updated. This complicates application programs.
- If one forgets to change one of the tuples, the redundant copies get out of sync, and it is not clear which is the correct information.

  Application programs may break if the data does not satisfy the assumptions of the programmer. E.g. querying the phone number of an instructor should normally yield a unique value, so "`SELECT INTO` " may be used. This gives an error if the query returns more than one value.

Problems (4)

- In the example, information about two different entities (Course and Instructor) is stored together in one table.
- This leads to insertion and deletion anomalies.
- Insertion Anomalies: The phone number of a new faculty member cannot be inserted until it is known what course he/she will teach.

  Since CRN is a key, it cannot be filled with "null".

- Deletion Anomalies: When the last course of a faculty member is deleted, his/her phone number is lost.

  Even if null values were possible, it would be strange that all courses by an instructor can be deleted except the last. Then the course data must be replaced by null values instead. This complicates the logic of application programs.

Contents

Boyce-Codd Normal Form (1)

- A Relation is in Boyce-Codd Normal Form (BCNF) if and only if all its FDs are already implied by the key constraints.
- So for every FD "$A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$" for R one of the following conditions hold:
  - The FD is trivial, i.e. $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$.
  - The FD follows from a key, because $\{A_1, \ldots, A_n\}$ or some subset of it is already a key of R.

    It can be any key, not necessarily the primary key.

Boyce-Codd Normal Form (2)

- BCNF $\iff$ Every determinant is a candidate key.
- So if a relation is in BCNF, FD constraints are not needed, only key constraints.
- The anomalies do not occur.

  At least not because of FDs: Since every (non-trivial) FD has a key on the left-hand side, and each key value can appear only once in the table, no redundancies occur because of the FDs. Again because every FD has a key on the left-hand side, there is no indication that information about different entities was merged.

Examples (1)

- COURSES(CRN, TITLE, INAME, PHONE) with the FDs
  - CRN $\longrightarrow$ TITLE, INAME, PHONE
  - INAME $\longrightarrow$ PHONE

  is not in BCNF because the FD "INAME $\longrightarrow$ 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 its FD), the relation COURSE(CRN, TITLE, INAME) is in BCNF:
  - CRN $\longrightarrow$ TITLE, INAME corresponds to the key.

Examples (2)

- Suppose that each course meets only once per week and that there are no cross-listed courses. Then

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

  satisfies the following FDs (plus implied ones):

  - `CRN` $\longrightarrow$ `TITLE, DAY, TIME, ROOM`
  - `DAY, TIME, ROOM` $\longrightarrow$ `CRN`

- The keys are `CRN` and `DAY, TIME, ROOM`.
- Both FDs have a key on the left hand side, so the relation is in BCNF.

Examples (3)

- Consider the relation PRODUCT(NO, NAME, PRICE) with the following functional dependencies:

  (1) NO $\longrightarrow$ NAME
  (2) NO $\longrightarrow$ PRICE
  (3) PRICE, NAME $\longrightarrow$ NAME
  (4) NO, PRICE $\longrightarrow$ NAME

- This relation is in BCNF:
  - The first two FDs show that NO is a key. Thus, they have a key on the left hand side.
  - The third FD is trivial and can be ignored.
  - The fourth FD has a superset of the key on the left hand side, which is also no problem.

Exercises

- Is the relation

  RESULTS(STUD_ID, EX_NO, POINTS, MAX_POINTS)

  with the following FDs in BCNF?

  (1) STUD_ID, EX_NO $\longrightarrow$ POINTS
  (2) EX_NO $\longrightarrow$ MAX_POINTS

- Is the relation

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

  with the following FDs in BCNF?

  (1) INV_NO $\longrightarrow$ DATE, AMOUNT, CUST_NO
  (2) INV_NO, DATE $\longrightarrow$ CUST_NAME
  (3) CUST_NO $\longrightarrow$ CUST_NAME
  (4) DATE, AMOUNT $\longrightarrow$ AMOUNT

Third Normal Form (1)

- Third Normal Form is a bit weaker than BCNF.
  If a relation is in BCNF, it is automatically in 3NF.

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

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

- In some rare circumstances, the "preservation of FDs" (see below) is lost when a relation is transformed into BCNF, whereas 3NF can always be reached without this problem.

Third Normal Form (2)

- A Relation R is in Third Normal Form (3NF) if and only if every FD "$A_1, \ldots, A_n \longrightarrow B$" satisfies at least one of the following conditions:

  An FD $A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$ with multiple attributes on the right hand side is treated as abbreviation for the FDs $A_1, \ldots, A_n \longrightarrow B_i$ $(i = 1, \ldots, m)$.

  - The FD is trivial, i.e. $B \in \{A_1, \ldots, A_n\}$.
  - The FD follows from a key, because $\{A_1, \ldots, A_n\}$ or some subset of it is already a key of R.
  - B is a key attribute, i.e. element of a minimal key of R.

Third Normal Form (3)

- The only difference to BCNF is the additional third possibility for showing that an FD does not violate the normal form.
- An attribute is called a non-key attribute if it does not appear in any minimal key.

  Not necessarily the primary key, but any candidate key. The minimality requirement is important here because otherwise the entire set of attributes of a relation would always qualify as a key.
- 3NF means that every determinant of a non-key attribute is a key.

  Again, not necessarily the primary key, but any candidate key.

Transitive Dependencies (1)

- BCNF and 3NF can also be defined via transitive dependencies.
- A relation is in BCNF iff there are no attribute sets α, β, γ such that
  - $\alpha \longrightarrow \beta$ and $\beta \longrightarrow \gamma$ are implied by the given FDs,
  - $\beta \not\longrightarrow \alpha$ (i.e. $\beta \longrightarrow \alpha$ is not implied),
  - $\gamma \not\subseteq \beta$.
- It suffices to consider sets γ that consist of a single attribute C.

Transitive Dependencies (2)

- The charactization of 3NF is again the same, except that now γ must consist of non-key attributes.

  Or alternatively, its single element C must be a non-key attribute.

- In the literature, transitive dependencies are often mentioned in connection with 3NF. But they apply equally well to BCNF.

  The reason is probably that transitive dependencies motivate why 2NF is not enough, and most textbooks use the sequence (1NF), 2NF, 3NF, BCNF. It is quite obvious that a definition with transitive dependencies is more complicated than the definition given above. Note especially that here implied FDs must be taken into account.

Second Normal Form (1)

- 2NF is only interesting for historical reasons. It is too weak, e.g. the "COURSES" example actually satisfies 2NF.
- If a relation is in 3NF or BCNF, it is automatically in 2NF.
- A relation is in Second Normal Form (2NF) if and only if every non-key attribute depends on the complete key.

Second Normal Form (2)

- I.e. a relation is in 2NF if and only if the given FDs do not imply an FD $A_1, \ldots, A_n \longrightarrow B$ such that:
  - $A_1, \ldots, A_n$ is a strict subset of a minimal key, and
  - B is not contained in any minimal key.

    I.e. a non-key attribute.
- E.g., the homework results table is not in 2NF:

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

  `MAX_POINTS` depends only on part of the key.

References

- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Ed.,
  Ch. 14, "Functional Dependencies and Normalization for Relational Databases"
  Ch. 15, "Relational Database Design Algorithms and Further Dependencies"
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3rd Ed.,
  Ch. 7, "Relational Database Design"
- Ramakrishnan/Gehrke: Database Management Systems, 2nd Ed., Mc-Graw Hill, 2000.
  Ch. 15, "Schema Refinement and Normal Forms"
- Simsion/Witt: Data Modeling Essentials, 2nd Ed.. Coriolis, 2001.
  Ch. 2: "Basic Normalization", Ch. 8: "Advanced Normalization".
- Batini/Ceri/Navathe: Conceptual Database Design, An Entity-Relationship Approach. Benjamin/Cummings, 1992.
- Kemper/Eickler: Datenbanksysteme (in German), Oldenbourg, 1997.
  Ch. 6, "Relationale Entwurfstheorie"
- Rauh/Stickel: Konzeptuelle Datenmodellierung (in German). Teubner, 1997.
- Kent: A Simple Guide to Five Normal Forms in Relational Database Theory. Communications of the ACM 26(2), 120–125, 1983.
- Thalheim: Dependencies in Relational Databases. Teubner, 1991, ISBN 3-8154-2020-2.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.