Datenbanken II A: DB-Entwurf

Chapter 14: Relational Normal Forms, Part II

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2020/21

http://www.informatik.uni-halle.de/~brass/dd20/

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 and correct violations to 4NF.
- detect normal form violations on the ER-level.
- decide about denormalization.

Contents

Splitting Relations (1)

- When a table is not in BCNF, one can split it into two tables ("decomposition").
- E.g. `COURSES(CRN, TITLE, INAME, PHONE)` is split into

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

- If the FD $A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$ violates BCNF, one creates a new relation $S(\underline{A_1}, \ldots, \underline{A_n}, B_1, \ldots, B_m)$ and removes $B_1, \ldots, B_m$ from the original relation.

  In unusual cases (multiple violations), it might be necessary to repeat the splitting step with one or both of the resulting relations. Then you have to consider also implied FDs.

Splitting Relations (2)

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

$$\texttt{COURSES} = \texttt{COURSES\_NEW} \bowtie \texttt{INSTRUCTORS}.$$

- I.e. the original relation can be defined as a view:

```
CREATE VIEW COURSES(CRN, TITLE, INAME, PHONE)
AS
SELECT C.CRN, C.TITLE, C.INAME, I.PHONE
FROM   COURSES_NEW C, INSTRUCTORS I
WHERE  C.INAME = I.INAME
```

Splitting Relations (3)

Definition:

- Let a relation $R(A_1, \ldots, A_k, B_1, \ldots, B_m, C_1, \ldots, C_n)$ be decomposed (split) into relations
  - $R_1(A_1, \ldots, A_k, B_1, \ldots, B_m)$
  - $R_2(A_1, \ldots, A_k, C_1, \ldots, C_n)$
- The decomposition is lossless if and only if

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

  for all valid database states.

  I.e. for all states that satisfy the constraints, which means in normal form theory usually the given functional dependencies. But later, also other types of constraints are studied, e.g. multivalued dependencies.

Splitting Relations (4)

Theorem (Decomposition Theorem):

- Consider again the decomposition of a relation $R(A_1, \ldots, A_k, B_1, \ldots, B_m, C_1, \ldots, C_n)$ into
  - $R_1(A_1, \ldots, A_k, B_1, \ldots, B_m)$
  - $R_2(A_1, \ldots, A_k, C_1, \ldots, C_n)$
- If the intersection of the attributes of the resulting relations, i.e. $A_1, \ldots, A_k$, is key in at least one of them, the decomposition is lossless.

  Note that is not an "if and only if"-condition. The decomposition might be lossless even when the condition is not satisfied.

Splitting Relations (5)

- In the example, the intersection of the attributes of the result of the decomposition is:

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

- Since INAME is key in INSTRUCTOR(INAME, PHONE), the decomposition is lossless.
- The above method for transforming relations into BCNF does only splits that satisfy this condition.
- It is always possible to transform a relation into BCNF by lossless splitting (if necessary repeated).

Splitting Relations (6)

- Not every lossless split is reasonable:

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

- Splitting this into STUD_FIRST(SSN,FIRST_NAME) and STUD_LAST(SSN,LAST_NAME) is lossless, but
  - is not necessary to enforce a normal form and
  - only requires costly joins in later queries.

Splitting Relations (7)

- Losslessness means that the resulting schema can represent all states which were possible before.

  We can translate states from the old schema into the new schema and back (if the FD was satisfied). The new schema supports all queries which the old schema supported: We can define the old relations as views.

- However, the new schema allows states which do not correspond to a state in the old schema: Now instructors without courses can be stored.

- Thus, the two schemas are not equivalent: The new one is more general.

Splitting Relations (8)

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

    None of the two can be specified declaratively in the `CREATE TABLE` statement. Thus, nothing is gained or lost.
  - the redundancy is avoided (update anomaly).

Splitting Relations (9)

- It should also be remarked that although computed columns (such as `AGE` from `BIRTHDATE`) lead to violations of BCNF, splitting the relation is not the right solution.

  The split would give a table `R(BIRTHDATE, AGE)` which does not have to be stored because it can be computed.

- The right solution is to eliminate `AGE` from the table, but to define a view which contains all columns of the table plus the computed column `AGE`.

Preservation of FDs (1)

- Another property, which a good decomposition of a relation should satisfy, is the preservation of FDs.
- The problem is that an FD can refer only to attributes of a single relation.

  Of course, you could still have a general constraint which states that the join of the two tables must satisfy an FD.

- When you split a relation, there might be FDs that can no longer be expressed.

  Of course, you can try to find implied FDs such that each FD refers only to the attributes of one of the resulting relations, but together imply the global FD. But even this is not always possible.

Preservation of FDs (2)

- A classical example is

  `ADDRESSES(STREET_ADR, CITY, STATE, ZIP)`

  with the FDs:

  (1) `STREET_ADR, CITY, STATE` $\longrightarrow$ `ZIP`

  (2) `ZIP` $\longrightarrow$ `STATE`

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

Preservation of FDs (3)

- Probably most database designers would not split the table (it is actually in 3NF, but violates BCNF).
- Textbooks say that it is more important to preserve FDs than to achieve BCNF.
- This is probably not the real reason: Few DB designers would actually write programs/triggers which check the FD.
- It does not often happen that there are two customers with exactly the same address.

  Only then the first FD could be potentially violated.

Preservation of FDs (4)

- If there are many addresses with the same ZIP code, there will be redundancies. If you split, you need expensive joins.
- Here the dependencies between ZIP code and other parts of the address are not considered as an independent fact, they are only interesting in the context of a given address (so insertion and deletion anomalies do not arise).

  Modifications are also relatively uncommon.
- If this were a DB for a mailing company, a table of ZIP codes might be of its own interest. Then the split should be done.
- The following table (with added customer number) is not even in 3NF, yet the same considerations apply:

  `CUSTOMER(NO, NAME, STREET_ADR, CITY, STATE, ZIP)`

Algorithm for 3NF (1)

- The following algorithm ("Synthesis Algorithm") produces a lossless decomposistion of a given relation into 3NF relations that preserve the FDs.
- First, one determines a minimal (canonical) set of FDs that is equivalent to the given FDs as follows:
  - Replace every FD $\alpha \longrightarrow B_1, \ldots, B_m$ by the corresponding FDs $\alpha \longrightarrow B_i$ $(i = 1, \ldots, m)$. Let the result be $\mathcal{F}$.
  - Continued on next slide ...

Algorithm for 3NF (2)

- Computation of canonical set of FDs, continued:
  - Minimize the left hand side of every FD:

    For each FD $A_1, \ldots, A_n \longrightarrow B$ and each $i = 1, \ldots, n$ compute the cover $\{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n\}^+$ (with respect to $\mathcal{F}$). If the result contains B, the attribute A_i is not necessary to uniquely determine B. $\mathcal{F}$ already implies $A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \longrightarrow B$. Thus, set $\mathcal{F} := (\mathcal{F} \setminus \{A_1, \ldots, A_n \longrightarrow B\}) \cup \{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \longrightarrow B\}$.
  - Remove implied FDs.

    For each FD $\alpha \longrightarrow B$ in $\mathcal{F}$, compute the cover α^+ with respect to $\mathcal{F}' := \mathcal{F} \setminus \{\alpha \longrightarrow B\}$. If the cover contains B, remove the FD $\alpha \longrightarrow B$, i.e. continue with $\mathcal{F} := \mathcal{F}'$ ($\alpha \longrightarrow B$ is implied by the other FDs).

Algorithm for 3NF (3)

- Synthesis Algorithm:
  - Compute the above minimal set of FDs $\mathcal{F}$.
  - For each left hand side α of an FD in $\mathcal{F}$, create a relation with attributes $\mathcal{A} := \alpha \cup \{B \mid \alpha \longrightarrow B \in \mathcal{F}\}$.

    Assign to this relation all FDs $\alpha' \longrightarrow B' \in \mathcal{F}$ with $\alpha' \cup \{B'\} \subseteq \mathcal{A}$.
  - If none of the constructed relations contains a key of the given relation, add one relation with all attributes of a key.
  - If one of the constructed relations has a set of attributes that is a subset of another relation, remove the relation with the subset of attributes.

Summary

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

  Such FDs indicate a redundancy created by combining pieces of information which should be stored separately.

- You can transform tables into BCNF by splitting them.
- Sometimes there is no really good solution, and not doing the split (which would give BCNF) might be the better of two bad things. But you should know what you are doing.

Contents

Introduction (1)

- The development of BCNF has been guided by a particular type of constraints: FDs.
- The goal of 3NF/BCNF is to
  - eliminate the redundant storage of data that follows from these constraints, and to
  - transform the tables such that the constraints are automatically enforced by means of keys.
- However, there are other types of constraints which are also useful for determining the table structure.

Introduction (2)

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

Introduction (3)

- Intuitively, 4NF means: Whenever it is possible to split a table (i.e. the decomposition is lossless), and this is not superfluous (see, e.g., slide 9), do it.
- Still shorter: 4NF means "Do not store unrelated information in the same relation."
- Probably, in practice it is very seldom that a relation violates 4NF, but not BCNF.

  However, I have seen students merging two binary relationships to one ternary relationship, which gives such a case. See below.
- But theoretically, it is a nice roundoff.

Multivalued Dependencies (1)

- Suppose that every employee knows a set of programming languages and a set of DBMS and that these are independent facts about the employees:

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	Access
Maria Brown	Java	Access

Multivalued Dependencies (2)

- If the sets of known DBMS and known programming languages are independent facts, 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	Access

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

Multivalued Dependencies (3)

- The table can only be decomposed if the knowledge of DBMS and programming languages is indeed independent.
- If a row means that the employee knows the interface between the language and DBMS, the split would lead to a loss of information.

  Then it would be only by chance that e.g. "`John Smith`" knows all four possible interfaces. If e.g. he would know only the interface between C and Oracle, and the interface between C++ and DB2, the contents of the two tables would be the same. One would not know exactly which interface the employee knows.

Multivalued Dependencies (4)

- The multivalued dependency

  $$\texttt{ENAME} \longrightarrow\!\!\!\!\!\rightarrow \texttt{PROG\_LANG}$$

  means that the set of values for PROG_LANG that is associated with every ENAME is independent of the other columns.

  Hidden in the table is a mapping from "ENAME" to sets of "PROG_LANG".

- Formally, "ENAME $\longrightarrow\!\!\!\!\!\rightarrow$ PROG_LANG" holds if whenever two tuples agree in the value for ENAME, one can exchange their values for PROG_LANG and get two other tuples in the table.

Multivalued Dependencies (5)

- E.g. from the two table rows

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

and the multivalued dependency, one can conclude that the table must contain also these two 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.

Multivalued Dependencies (6)

- A multivalued dependency (MVD)

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

  is satisfied in a DB state $\mathcal{I}$ if and only if for all tuples t and u that have the same values for the columns $A_1, \ldots, A_n$ (i.e. $t.A_i = u.A_i$ for $i = 1, \ldots, n$), there are two further tuples t' and u' such that

  - t' has the same values as t for all columns except that $t'.B_j = u.B_j$ for $j = 1, \ldots, m$, and
  - u' agrees with u except that $u'.B_j = t.B_j$

  (i.e. the values for the B_j are exchanged).

Multivalued Dependencies (7)

- Multivalued dependencies come always in pairs:
  If "ENAME $\longrightarrow\!\!\!\!\!\rightarrow$ PROG_LANG" holds,
  then "ENAME $\longrightarrow\!\!\!\!\!\rightarrow$ DBMS" is automatically satisfied.

  For $R(A_1, \ldots, A_k, B_1, \ldots, B_n, C_1, \ldots, C_m)$ the multivalued dependency $A_1, \ldots, A_k \longrightarrow\!\!\!\!\!\rightarrow B_1, \ldots, B_n$ is equivalent to $A_1, \ldots, A_k \longrightarrow\!\!\!\!\!\rightarrow C_1, \ldots, C_m$.
  Exchanging the B_j values in the two tuples is the same as exchanging the values for all other columns (the A_i values are identical, so exchanging them has no effect).

Multivalued Dependencies (8)

- An MVD $A_1, \ldots, A_n \longrightarrow\!\!\!\!\!\rightarrow B_1, \ldots, B_m$ for a relation R is trivial if and only if
  - $\{B_1, \ldots, B_m\} \subseteq \{A_1, \ldots, A_n\}$ or

    The precondition for exchanging the B_j-values in the two tuples is that they agree in the A_i-values. But if every B_j is also an A_i, only equal values are exchanged, which has no effect.
  - $\{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_m\}$ are all columns of R.

    In this case, exchanging the B_j values of tuples t and u gives the tuples u and t, and no new tuples.

Multivalued Dependencies (9)

- If the FD $A_1, \ldots, A_n \longrightarrow B_1, \ldots, B_m$ holds, the corresponding MVD $A_1, \ldots, A_n \longrightarrow\!\!\!\!\!\rightarrow B_1, \ldots, B_m$ is trivially satisfied.

  The functional dependency means that if t and u have the same values for the A_i, then they also have the same values for the B_j. But then exchanging their B_j values changes nothing.

- As an FD can be implied (i.e. automatically true) given a set of FDs, FDs and MVDs can also follow from a set of given FDs and MVDs.

  A constraint φ is implied by constraints $\{\varphi_1, \ldots, \varphi_n\}$ if and only if every database state which satisfies $\{\varphi_1, \ldots, \varphi_n\}$ also satisfies φ.

FD/MVD Deduction Rules

- The following deduction rules permit to derive all implied FDs/MVDs (they are sound and complete):
  - The three Armstrong axioms for FDs.
  - If $\alpha \longrightarrow\!\!\!\!\!\rightarrow \beta$ then $\alpha \longrightarrow\!\!\!\!\!\rightarrow \gamma$, where γ are all remaining columns of the relation.
  - If $\alpha_1 \longrightarrow\!\!\!\!\!\rightarrow \beta_1$ and $\alpha_2 \supseteq \beta_2$, then $\alpha_1 \cup \alpha_2 \longrightarrow\!\!\!\!\!\rightarrow \beta_1 \cup \beta_2$.
  - If $\alpha \longrightarrow\!\!\!\!\!\rightarrow \beta$ and $\beta \longrightarrow\!\!\!\!\!\rightarrow \gamma$, then $\alpha \longrightarrow\!\!\!\!\!\rightarrow (\gamma \setminus \beta)$.
  - If $\alpha \longrightarrow \beta$, then $\alpha \longrightarrow\!\!\!\!\!\rightarrow \beta$.
  - If $\alpha \longrightarrow\!\!\!\!\!\rightarrow \beta$, $\beta' \subseteq \beta$, and there is γ with $\gamma \cap \beta = \emptyset$ and $\gamma \longrightarrow \beta'$, then $\alpha \longrightarrow \beta'$.

Fourth Normal Form (1)

- A relation is in Fourth Normal Form (4NF) with respect to given FDs and MVDs if and only if no MVD $A_1, \ldots, A_n \longrightarrow\!\!\!\!\!\rightarrow B_1, \ldots, B_m$ is implied which is
  - not trivial and
  - not directly implied by a key, i.e. $A_1, \ldots, A_n$ does not functionally determine all other attributes.
- The definition of 4NF is very similar to the BCNF definition: It only looks at implied MVDs instead of given 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).

  If might first seem that an FD violating BCNF could lead to a trivial MVD: The second case for trivial MVDs has no counterpart for FDs. But if $\{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_m\}$ are all columns of R, the FD corresponds to a key and cannot violate BCNF.

- `EMP_KNOWLEDGE(ENAME, PROG_LANG, DBMS)` is an example of a relation that is in BCNF, but not in 4NF.

  It has no non-trivial FDs (the key consists of all attributes).

Fourth Normal Form (3)

- But if there are other columns besides the key of the entity (ENAME) and a multivalued attribute, even 2NF is violated:

EMPLOYEES		
ENAME	PROG_LANG	SALARY
John Smith	C	58 000
John Smith	C++	58 000
Maria Brown	Prolog	62 000
Maria Brown	Java	62 000

- It is not very common that 4NF is violated, but BCNF is not.

Fourth Normal Form (4)

- Splitting a relation

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

into relations

$$R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \quad \text{and}$$
$$R_2(A_1, \ldots, A_n, C_1, \ldots, C_k)$$

is lossless, i.e.

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

if and only if $A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots, B_m$.

Or equivalently $A_1, \ldots, A_n \twoheadrightarrow C_1, \ldots, C_k$.

Fourth Normal Form (5)

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

  It might be necessary to split it multiple times.

- So the essence of 4NF is:
  - If a decomposition into two relations is lossless (i.e. the original relation can 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.

Contents

Fifth Normal Form (1)

- Fifth normal form is very seldom used in practice.

  Many textbooks actually do not discuss it at all (more than half of those I checked). 5NF is also called projection-join normal form (PJNF).

- 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 in two tables is lossless.

  This means that the required decomposition cannot be reached by repeated splitting into two tables. Instead, one needs additional tables with overlapping columns which only serve as a filter in the join.

Fifth Normal Form (2)

- E.g. consider

  `COURSE_OFFER(TITLE, TERM, INSTRUCTOR)`

- Normally, information is lost by the split into
  - `OFFERED(TITLE, TERM)`,
  - `EMPLOYED(INSTRUCTOR, TERM)`,
  - `TEACHES(TITLE, INSTRUCTOR)`.
- But the split would be lossless if following constraint were true: "If a course C is offered in a term T, and instructor I ever taught C and teaches some course in T, then I teaches C in T."

Fifth Normal Form (3)

- A join dependency (JD) states that a split 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, every attribute of R must appear in at least one of the projections. Then "$\subseteq$" is always satisfied, only "$\supseteq$" is interesting. It states that if there are n tuples $t_1, \ldots, t_n$ in R that agree in the values for the attributes listed in more than one projection, then one can construct a tuple from them that must also appear in R.

- An MVD $A_1, \ldots, A_n \longrightarrow\!\!\!\!\!\rightarrow B_1, \ldots, B_m$ corresponds to

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

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

Fifth Normal Form (4)

- A relation R is in Fifth Normal Form (5NF) if and only if every join dependency that holds for it is already implied by a key for R.

  Note that trivial constraints are always implied, so they do not have to be treated specially.

- Of course, 5NF implies 4NF, BCNF, 3NF, 2NF.

- If a relation is in 3NF, and all its keys consist of a single column each, it is automatically in 5NF.

  So in this case, it is not necessary to consider multivalued dependencies and join dependencies.

Contents

Domain-Key Normal Form (1)

- Consider a table for possible answers in multiple choice tests:

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

Domain-Key Normal Form (2)

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

  Each question can have only one correct answer.

- Domain-Key Normal Form (DKNF) requires that every constraint on the relation is implied by the domains and keys defined for that relation.
- It is of course nice if a relation is in DKNF: One only has to enforce domains and keys.
- Many relations cannot be transformed into DKNF.

Domain-Key Normal Form (3)

- Here, a horizontal decomposition might be useful:

CORRECT_ANSWERS		
QUESTION	ANSWER	TEXT
1	A	...
2	B	...

WRONG_ANSWERS		
QUESTION	ANSWER	TEXT
1	B	...
1	C	...
2	A	...
2	C	...

Domain-Key Normal Form (4)

- Now the key of CORRECT_ANSWERS enforces that every question has only one correct answer.

  CORRECT_ANSWERS may even be merged with a table QUESTIONS that contains the text of each question. In this way, it is certain that every question has exactly one correct answer.

- Each relation (considered in isolation) is in DKNF.
- However, a new inter-relational constraint must be enforced: The same question with the same answer may not appear in both relations.
- Exercise: Discuss which schema is better.

Domain-Key Normal Form (5)

- If every domain contains at least two values, DKNF implies 4NF.

  It also implies 5NF if every domain contains at least as many values as there are components in a join dependency.

- One important goal of normalization is indeed to replace general constraints as far as possible by standard constraints.

  Domain and key constraints are very simple constraints, supported in (nearly) every DBMS. Today, also `CHECK`-constraints defined on entire tuples could be used (domain constraints are basically `CHECK`-constraints for single attributes).

Domain-Key Normal Form (6)

Summary:

- If one has only FDs, BCNF is probably the best one can do. The next step was to look at more general constraints.
- 5NF is the end of vertical decomposition, i.e. using projections for normalization.
- But, as the example showed, also horizontal decomposition or other schema transformations should be considered.
- DKNF is the ultimate normal form.

Contents

Introduction (1)

- If a good ER-schema is transformed into the relational model, the resulting tables will satisfy all normal forms.

  Otherwise it was not a good ER-schema ...

- If normal form violations are detected in the relational schema, one must go back to the ER-schema and correct them there.

  Unless one has done special tricks during the logical design, a normal form violation always means that the same problem occurs in the ER-schema. It is important that ER-schema and relational schema remain in sync, otherwise the ER-schema loses its value as a documentation for the actually implemented database.

Introduction (2)

- Of course, it is better to detect the errors directly in the ER-schema.

  The earlier one detects an error, the cheaper it is to correct it.

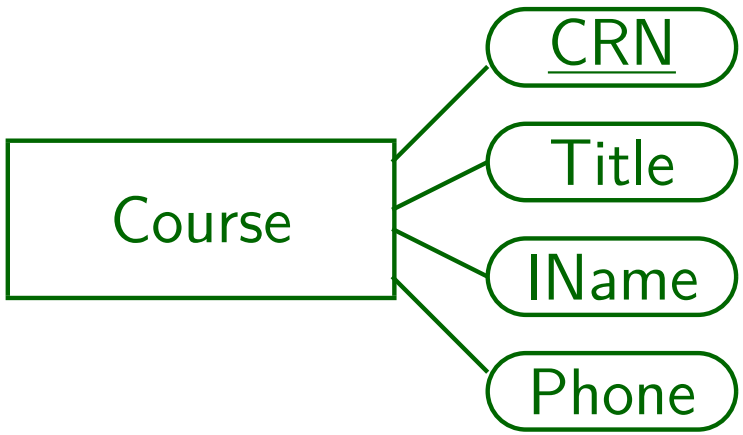
- There is no independent normalization theory for ER-schemas.

  It is possible to define normal forms like BCNF for ER-schemas, but this is significantly more complicated than for relational schemas, and does not really give something new (as far as I know).

- The definition is simply: An entity or a relationship is in BCNF if and only if the corresponding table is in BCNF.

Examples (1)

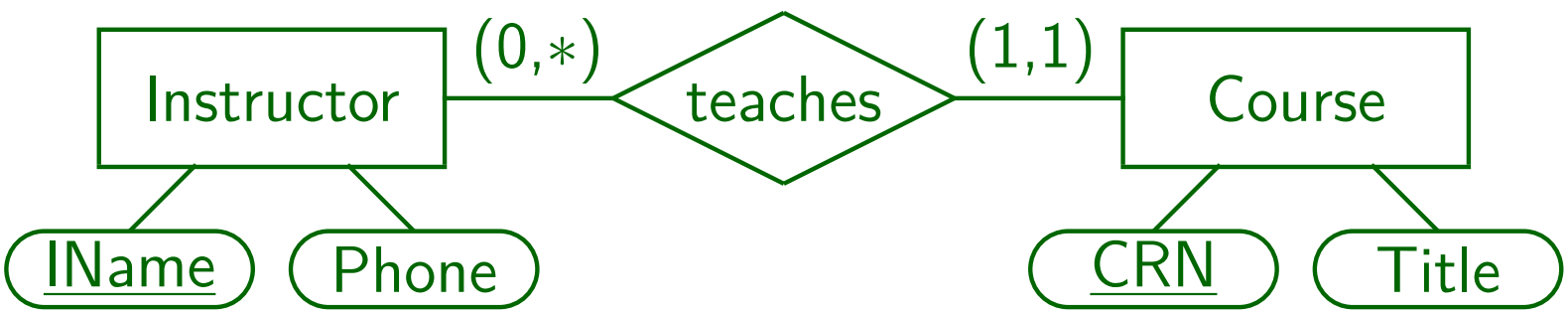
- Consider again the first example of this chapter:



- Here the functional dependency "IName $\longrightarrow$ Phone" leads to the violation of BCNF discussed above.
- Also in the ER-model, FDs between attributes of an entity should be implied by a key constraint.

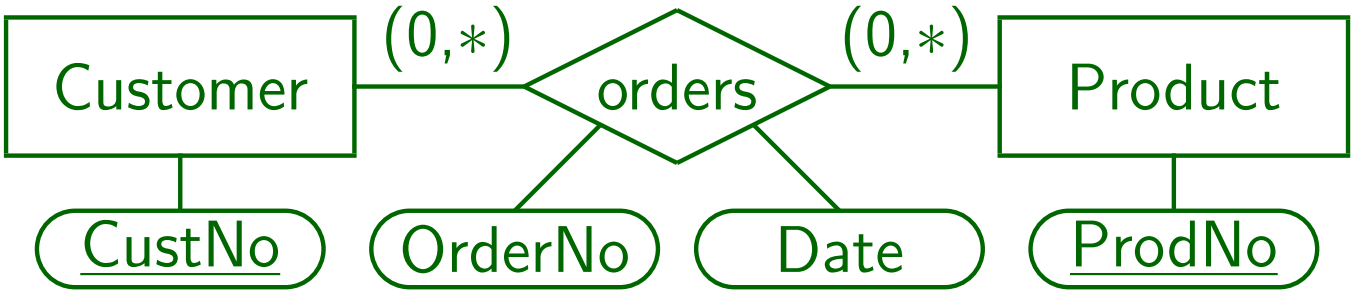
Examples (2)

- In the ER-model the solution is the same as in the relational model: We have to split the construct.
- In this case, we discover that "Instructor" is an independent entity:



Examples (3)

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



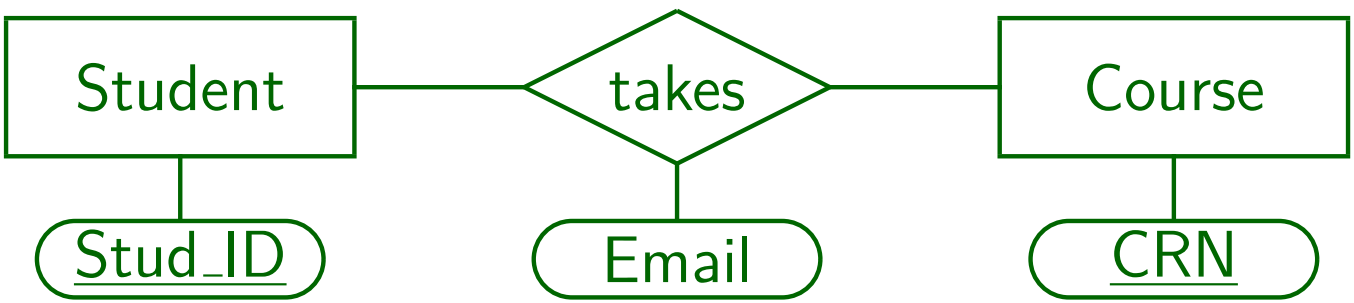
- The FD "OrderNo $\longrightarrow$ Date" violates BCNF.

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

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

Examples (4)

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



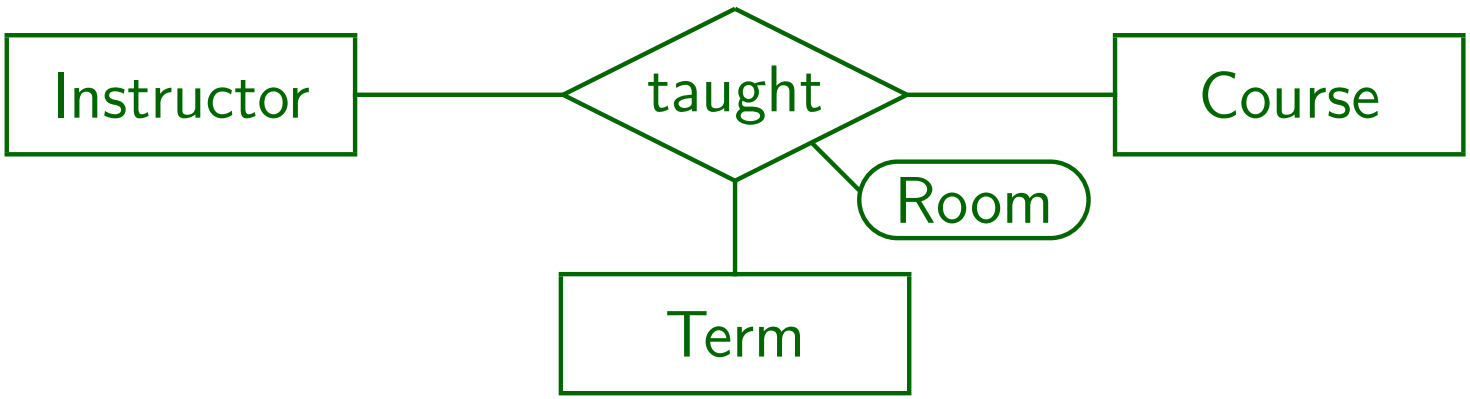
- The relationship is translated into

  Takes(Stud_ID, CRN, Email).

- Then the FD "Stud_ID $\longrightarrow$ Email" violates BCNF.
- Obviously "Email" is an attribute of "Student".

Examples (5)

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

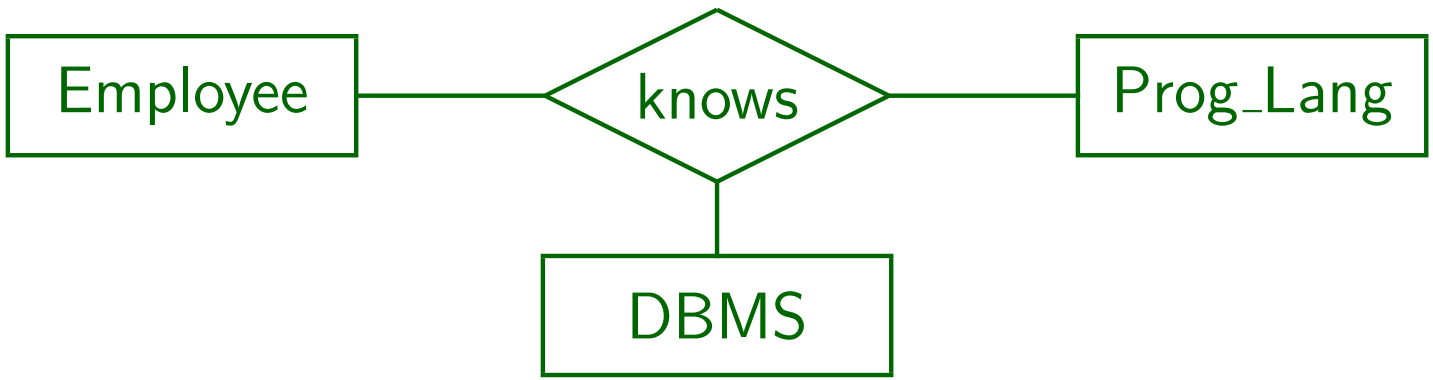


- If every course is taught only once per term, the "Room" depends only on "Course" and "Term".

  Solution hint: Create "Course_Offer" as association entity between "Course" and "Term".

Examples (6)

- If independent relationships are combined, 4NF is violated:



- If the knowledge of programming languages and DBMSs is independent, one must use two binary relationships instead.

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

Summary (1)

- Many errors in ER-design manifest themselves later as violations of relational normal forms.
- So it is a good test to think about FDs on the created tables and check for normal forms.

  One should check whether the tables make sense and not blindly trust the automatic generation from an ER-schema. Of course, the automatic translation preserves equivalence. However, the ER-schema might contain errors which have been overlooked earlier.
- Think about FDs already when developing the ER-schema, and not only after the translation!

Summary (2)

- Normalization is about:
  - Avoiding redundancy.
  - Storing separate facts separately.
  - Transforming general integrity constraints into constraints that are supported by the DBMS.
- Relational normalization theory is based mainly on FDs, but there are other types of constraints.

  The ER-model is also richer in constructs and built-in constraints.
- Instead of simply applying relational normal forms to ER-schemas, follow the above three goals!

Contents

Denormalization (1)

- Denormalization is the process of adding redundant columns and tables to the database in order to improve performance.
- E.g. if the phone number of the instructor of a course is often needed, the column `PHONE` can be added to the table `COURSES`.
- Then the join between `COURSES` and `INSTRUCTORS` can often be avoided, because the required instructor information (`PHONE`) is stored redundantly in the same row as the course information.

Denormalization (2)

- Since there is a separate table `INSTRUCTORS` (which contains the phone number, too), insertion and deletion anomalies are avoided.
- But there will be update anomalies (changing a single phone number requires updating many rows).
- Thus, one must pay for the performance gain with a more complicated application logic, the need for triggers, and some remaining insecurity (will all copies always agree?).

Denormalization (3)

- If the tables `COURSES` and `INSTRUCTORS` are small, it is certainly a bad decision to violate BCNF here, since performance is anyway no problem.
- Koletzke/Dorsey state that if your tables have less than 100 000 rows, you need no denormalization.

  Of course, it also depends on the number of queries per second.
- Too much denormalization can make a database nearly unmodifiable.

  The requirements often change, so one must anticipate that the DB application system will need changes.

Denormalization (4)

- In the above example, denormalization helped to avoid joins.
- Denormalization also includes creating tables or columns which hold aggregated values.

  In this case, formally no normal form is violated, but the information is of course redundant.

- E.g. suppose one stores invoices to a customer, and payments from a customer. One could e.g. store in the `CUSTOMERS` table the current balance.

  This is redundant, because it can be computed as the sum of all payments minus the sum of all invoices.

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.