Principles of Data- and Knowledge-based Systems

Torsten Schaub
University of Potsdam
torsten@cs.uni-potsdam.de

Relational normal form: Overview

Outline

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.

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	Brass	9404
42232	DS	Brass	9404
31822	IS	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.

Introduction (3)

- Of course, it is no problem if a column contains the same value two times (e.g. consider a `Y/N` column).
- But in this case, the following holds: If two rows have the same value in the column `INAME`, they must have the same value in the column `PHONE`.
- This is an example of a functional dependency: `INAME` $\rightarrow$ `PHONE`.
- Because of this rule, one of the two `PHONE` entries for `Brass` is redundant.

Introduction (4)

- Table entries are redundant if they can be reconstructed from other table entries and additional information (like the FD in this case).
- Redundant information in database schemas is bad:
  - Storage space is wasted.
  - If the information is updated, all redundant copies must be updated. If one is not careful, the copies become inconsistent (Update Anomaly).

Introduction (5)

- In the example, the information about the two concepts “Course” and “Instructor” are intermixed in one table.
  This is bad:
  - The phone number of a new faculty member can be stored in the table only together with a course (Insertion Anomaly).
  - When the last course of a faculty member is deleted, his/her phone number is lost (Deletion Anomaly).

Introduction (6)

- Third Normal Form (3NF) is considered part of a decent database design.
- Boyce-Codd Normal Form (BCNF) is slightly stronger, easier to define, and better matches intuition.
  Intuitively, BCNF means that all FDs are already enforced by keys.
- Only BCNF is defined here.
- If a table is in BCNF, it is automatically in 3NF.

Outline

Functional Dependencies (1)

- Functional dependencies (FDs) are generalizations of keys.
- A functional dependency specifies that an attribute (or attribute combination) uniquely determines another attribute (or other attributes).
- Functional dependencies are written in the form

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

- This means that whenever two rows have the same values in the attributes $A_1, \ldots, A_n$, then they must also agree in the attributes $B_1, \ldots, B_m$.

Functional Dependencies (2)

- As noted above, the FD "INAME $\rightarrow$ PHONE" is satisfied in the following example:

COURSES			
<u>CRN</u>	TITLE	INAME	PHONE
22268	DB	Brass	9404
42232	DS	Brass	9404
31822	IS	Spring	9429

- If two rows agree in the instructor name, they must have the same phone number.

Functional Dependencies (3)

- A key uniquely determines every attribute, i.e. the FDs "`CRN`$\rightarrow$`TITLE`", "`CRN`$\rightarrow$`INAME`", "`CRN`$\rightarrow$`PHONE`" are trivially satisfied:
  - There are no two distinct rows that have the same value for a key (`CRN` in this case).
  - Therefore, whenever rows t and u agree in the key (`CRN`), they must actually be the same row, and therefore agree in all other attributes, too.
- Instead of the three FDs above, one can also write the single FD "`CRN` $\rightarrow$ `TITLE, INAME, PHONE`".

Functional Dependencies (4)

- In the example, the FD "`INAME` $\rightarrow$ `TITLE`" is not satisfied: There are two rows with the same `INAME`, but different values for `TITLE`.
- In the example, the FD "`TITLE` $\rightarrow$ `CRN`" is satisfied.
- However, like keys, FDs are constraints: They must hold in all possible database states, not only in a single example state.

Functional Dependencies (5)

- Therefore, it is a database design task to determine which FDs should hold. This cannot be decided automatically, and the FDs are needed as input for the normalization check.
- In the example, the DB designer must find out whether it can ever happen that two courses are offered with the same title (e.g. two sessions of a course that is overbooked).
- If this can happen, the FD "TITLE $\rightarrow$ CRN" does not hold in general.

Functional Dependencies (6)

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

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

- In discussing FDs, the focus is on a single relation R. All attributes A_i, B_i are from this relation.
- The FD $A_1, \ldots, A_n \to B_1, \ldots, B_m$ is equivalent to the m FDs:

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

FDs vs. Keys

- 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 "$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$" holds.
- Given the FDs for a relation, one can compute a key by finding a set of attributes $A_1, \ldots, A_n$ that functionally determines the other attributes.

Trivial FDs

- A functional dependency $\alpha \to \beta$ such that $\beta \subseteq \alpha$ is called trivial.
- Examples are:
  - TITLE $\to$ TITLE
  - INAME, PHONE $\to$ PHONE
- Trivial functional dependencies are always satisfied (in every database state, no matter whether it satisfies other constraints or not).
- Trivial FDs are not interesting.

Implication of FDs (1)

- CRN→PHONE is nothing new when one knows already CRN→INAME and INAME→PHONE.
- A set of FDs $\{\alpha_1 \to \beta_1, \ldots, \alpha_n \to \beta_n\}$ implies an FD $\alpha \to \beta$ if and only if every DB state which satisfies the $\alpha_i \to \beta_i$ for $i = 1, \ldots, n$ also satisfies $\alpha \to \beta$.

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 $\alpha \rightarrow \beta$ trivially holds (Reflexivity).
  - If $\alpha \rightarrow \beta$, then $\alpha \cup \gamma \ \rightarrow \ \beta \cup \gamma$ (Augmentation).
  - If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (Transitivity).

Implication of FDs (3)

- A simpler way to check whether $\alpha \rightarrow \beta$ is implied by given FDs is to compute first the attribute cover α^+ of α and then to check whether $\beta \subseteq \alpha^+$.
- The attribute cover $\alpha^+_{\mathcal{F}}$ of a set of attributes α is the set of all attributes B that are uniquely determined by α (with respect to given FDs $\mathcal{F}$).

$$\alpha^+_{\mathcal{F}} := \{B \mid \text{The FDs } \mathcal{F} \text{ imply } \alpha \rightarrow B\}.$$

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

Implication of FDs (4)

- The cover is computed as follows:

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

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

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

Implication of FDs (5)

- Consider the following FDs:

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

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

Implication of FDs (6)

- The first of the given FDs, namely

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

  has a left hand side (ISBN) that is contained in the current set x (actually, $x = \{\texttt{ISBN}\}$).
- Therefore, we can extend x by the attributes on the right hand side of this FD, i.e. TITLE, and PUBLISHER:

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

Implication of FDs (7)

- Now the third of the FDs, namely

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

  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

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

  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.

$$\texttt{ISBN} \rightarrow \texttt{PUB\_URL}.$$

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

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.
- $\alpha \subseteq \mathcal{A}$ is a key if and only if $\alpha^+ = \mathcal{A}$.
- Normally, one is only interested in minimal keys.

How to Determine Keys (2)

- 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).
- 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.

Outline

Motivation (1)

- Consider again the example:

COURSES			
CRN	TITLE	INAME	PHONE
22268	DB	Brass	9404
42232	DS	Brass	9404
31822	IS	Spring	9429

- As noted above, the FD `INAME`→`PHONE` leads to problems, one of which is the redundant storage of certain facts (e.g. the phone number of "`Brass`").
- The FD `INAME`→`PHONE` leads exactly then to redundancies, if there are several lines with the same value for `INAME` (left side of the FD).
- Then these lines must also have the same value for `PHONE` (right side of the FD). Of these, all but one copy are redundant.

Motivation (2)

- Actually, any FD $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ will cause redundant storage unless $A_1, \ldots, A_n$ is a key, so that each combination of attribute values for $A_1, \ldots, A_n$ can occur only once.
- Avoid (proper) FDs by transforming them into key constraints. This is what normalization does.

Motivation (3)

- The problem in the example is also caused by the fact that information about different concepts is stored together (faculty members and courses).
- Formally, this follows also from "INAME$\rightarrow$PHONE":
  - INAME is like a key for only part of the attributes.
  - It identifies faculty members, and PHONE depends only on the faculty member, not on the course.
- Again: The left hand side of an FD should be a key.

Boyce-Codd Normal Form (BCNF)

- A Relation R is in BCNF if and only if all its FDs are already implied by key constraints.
- I.e. for every FD "$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$" one of the following conditions must 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.

Boyce-Codd Normal Form

Check

- From the given FDs $\mathcal{F}$ one can first determine the keys of the relation, and then apply the definition directly:
  For each FD $\alpha \rightarrow \beta$ from $\mathcal{F}$: If $\beta \not\subseteq \alpha$, then α contains one of the keys (plus possibly other attributes).
- However, one can also check for each FD $\alpha \rightarrow \beta$ with $\beta \not\subseteq \alpha$ whether the attribute cover α^+ is already the set of all the attributes of the relation.
  Then α or a subset of it is just a key.

Examples (1)

- `COURSES(`<u>`CRN`</u>`, TITLE, INAME, PHONE)` with the FDs
  - `CRN` → `TITLE, INAME, PHONE`
  - `INAME` → `PHONE`

  is not in BCNF because the FD "`INAME` → `PHONE`" is not implied by a key:
  - "`INAME`" is not a key of the entire relation.
  - The FD is not trivial.
- However, without the attribute `PHONE` (and its FD), the relation is in BCNF:
  - `CRN` → `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` $\rightarrow$ `TITLE, DAY, TIME, ROOM`
  - `DAY, TIME, ROOM` $\rightarrow$ `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)

- Suppose that PRODUCT(NO, NAME, PRICE) has these FDs:

  (1) NO $\rightarrow$ NAME (3) PRICE, NAME $\rightarrow$ NAME

  (2) NO $\rightarrow$ PRICE (4) NO, PRICE $\rightarrow$ NAME

- This relation is in BCNF:
  - The first two FDs show that NO is a key. Since their left hand side is a key, they are no problem.
  - 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.

Splitting Relations (1)

- A table which is not in BCNF can be split into two tables ("decomposition"), e.g. split COURSES into

  COURSES_NEW(<u>CRN</u>, TITLE, INAME→INSTRUCTORS)
  INSTRUCTORS(<u>INAME</u>, PHONE)

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

Splitting Relations (2)

- When splitting relations, it is of course important that the 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)

- The split of the relations is guaranteed to be lossless if the intersection of the attributes of the new tables is a key of at least one of them ("decomposition theorem"):

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

- 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 (4)

- 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 (5)

- Losslessness means that the resulting schema can represent all states which were possible before.
- 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 (6)

- 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
  - the redundancy is avoided (update anomaly).

Outline

Summary

- Data base design
- Redundancy causes anomalies on
  - Deletion
  - Insertion
  - Update
- Functional dependencies
  - Implication
  - Comutation of attribute cover
- BCNF eliminates all redundancies based on functional dependencies
  - Check
  - Split

Bibliography

- The following list of references is compiled from the open source bibliography available at

  `https://github.com/krr-up/bibliography`

- Feel free to submit corrections via pull requests!

[1] S. Abiteboul, R. Hull, and V. Vianu.
Foundations of Databases.
Addison-Wesley, 1995.

[2] C. Aggarwal, editor.
Data Streams — Models and Algorithms, volume 31 of *Advances in Database Systems*.
Springer-Verlag, 2007.

[3] K. Apt, H. Blair, and A. Walker.
Towards a theory of declarative knowledge.
In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann Publishers, 1987.

[4] M. Arenas, L. Bertossi, and J. Chomicki.
Consistent query answers in inconsistent databases.

In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'99)*, pages 68–79. ACM Press, 1999.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors.
The Description Logic Handbook: Theory, Implementation, and Applications.
Cambridge University Press, 2003.

[6] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom.
Models and issues in data stream systems.
In L. Popa, editor, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'02)*, pages 1–16. ACM Press, 2002.

[7] C. Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.

[8] S. Ceri, G. Gottlob, and L. Tanca.
Logic Programming and Databases.
Springer-Verlag, 1990.

[9] R. Elmasri and S. Navathe.
Fundamentals of database systems.
Addison-Wesley, 1994.

[10] R. Fagin, J. Ullman, and M. Vardi.
On the semantics of updates in databases. preliminary report.
In *Proceedings of the Second ACM Conference SIGACT-SIGMOD*, pages 352–365, 1983.

[11] H. Gallaire, J. Minker, and J. Nicolas.
Logic and databases: A deductive approach.
Computing Surveys, 16(2):153–185, 1984.

[12] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele.
Potassco User Guide.

University of Potsdam, 2 edition, 2015.

[13] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
Answer Set Solving in Practice.
Synthesis Lectures on Artificial Intelligence and Machine Learning.
Morgan and Claypool Publishers, 2012.

[14] M. Gelfond and Y. Kahl.
Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach.
Cambridge University Press, 2014.

[15] M. Gelfond and V. Lifschitz.
Classical negation in logic programs and disjunctive databases.
New Generation Computing, 9:365–385, 1991.

[16] H. Katsuno and A. Mendelzon.
On the difference between updating a knowledge database and revising it.

In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 387–394. Morgan Kaufmann Publishers, 1991.

[17] V. Lifschitz.
Closed-world databases and circumscription.
Artificial Intelligence, 27:229–235, 1985.

[18] V. Lifschitz.
Nonmonotonic databases and epistemic queries.
In J. Myopoulos and R. Reiter, editors, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 381–386. Morgan Kaufmann Publishers, 1991.

[19] V. Lifschitz.
Introduction to answer set programming.
Unpublished draft, 2004.

[20] V. Lifschitz, F. van Harmelen, and B. Porter, editors.

Handbook of Knowledge Representation.
Elsevier Science, 2008.

[21] L. Liu and M. Özsu, editors.
Encyclopedia of Database Systems.
Springer-Verlag, 2009.

[22] V. Marek and M. Truszczyński.
Nonmonotonic logic: context-dependent reasoning.
Artifical Intelligence. Springer-Verlag, 1993.

[23] J. Minker, editor.
Foundations of Deductive Databases and Logic Programming.
Morgan Kaufmann Publishers, 1988.

[24] R. Reiter.
On closed world data bases.
In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 55–76. Plenum Press, New York, 1978.

[25] R. Reiter.

Towards a logical reconstruction of relational database theory.
In M. Brodie, J. Myopoulos, and J. Schmidt, editors, *On conceptual modeling: Perspectives from Artificial Intelligence, Datbases and Programming Languages*, pages 191–233. Springer-Verlag, 1984.

[26] R. Reiter.
On asking what a database knows.
In J. Lloyd, editor, *Computational Logic*, pages 96–113. Springer-Verlag, 1990.

[27] J. Ullman.
Principles of Database Systems.
Computer Science Press, Rockville MD, 1982.

[28] J. Ullman.
Principles of Database and Knowledge-Base Systems.
Computer Science Press, 1988.