

Part 8: Introduction to Relational Normal Forms

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”
- Simson/Witt: Data Modeling Essentials, 2nd Edition. Coriolis, 2001. Ch. 2: “Basic Normalization”, Ch. 8: “Advanced Normalization”.
- 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.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.

Objectives

After completing this chapter, you should be able to:

- Detect bad relational database designs (that contain redundancies).
- Determine functional dependencies.
- Check whether a given table is in BCNF for given functional dependencies.
- Detect redundancy and normalization problems already during the conceptual design in the ER-model.

Overview

1. Introduction (Anomalies)

2. Functional Dependencies

3. BCNF

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 which violate the conditions of a normal form.

However, there are exceptions and tradeoffs.

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			
<u>CRN</u>	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 → **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).

E.g. if an employee table contains the date of birth, the additional column `AGE` would be redundant: The age can be computed from the date of birth (and the knowledge about today's date).

- 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)

- Redundant information might be convenient for easy query formulation.
- In relational databases, it is possible to define virtual tables (views) that are computed by a query.
- Since the contents of a view is not explicitly stored and not directly updated, redundant information is no problem for views.

The entire view is redundant, since it is computed from the stored relations.

Introduction (6)

- Sometimes, redundant information is needed for efficient query evaluation.
- There is a tradeoff: Storing redundant information is bad, but slow query evaluation is also bad.
- But adding redundant information should only be discussed during physical design. There must be really good and quantifiable reasons.
- Avoid storing redundant data whenever you can!

Many cases of redundant information can be detected by checking for normal forms.

Introduction (7)

- 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).

Null values also do not help since the course reference number is the key of the table, and the key must be not null.
 - ◇ If one deletes the last course of a faculty member, his/her phone number is lost (Deletion Anomaly).

Introduction (8)

- If one does a good Entity-Relationship design and translates it into the relational model, all normal forms will be automatically satisfied.
- However, normal forms are generally accepted. If one should have to argue about design alternatives in a team, saying that one schema violates a normal form is a strong and formal reason against it.
- Normal forms give another possibility for checking a proposed schema. However, it is much better to detect the problems already on the ER-level.

Introduction (9)

- Today, Third Normal Form (3NF) is considered part of general database education.
- 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 (so one can forget about FDs after the normalization check).
- Only BCNF is defined here.
- If a table is in BCNF, it is automatically in 3NF.

Overview

1. Introduction (Anomalies)

2. Functional Dependencies

3. BCNF

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

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

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

Functional Dependencies (2)

- As noted above, the FD “**INAME** → **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.

If two rows do not have the same value for **INAME**, the condition is void for them.

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.

Of course, if an FD does not hold in a valid example state, it is clear that it cannot hold in general. E.g. “**INAME** \rightarrow **TITLE**” does not have to be considered any further.

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

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

FDs vs. Keys

- FDs are a generalization of keys: A_1, \dots, A_n is a key of $R(A_1, \dots, A_n, B_1, \dots, B_m)$ if and only if the FD “ $A_1, \dots, A_n \rightarrow B_1, \dots, B_m$ ” 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, it is possible to compute a key by finding a set of attributes A_1, \dots, A_n that functionally determines the other attributes.

Implication of FDs

- The FD $CRN \rightarrow PHONE$ is nothing new when $CRN \rightarrow INAME$ and $INAME \rightarrow PHONE$ are already known.

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

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

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

- So only in a representative set of FDs should be specified, which implies all other FDs.

A set of FDs $\{\alpha_1 \rightarrow \beta_1, \dots, \alpha_n \rightarrow \beta_n\}$ implies an FD $\alpha \rightarrow \beta$ if and only if every database state which satisfies the $\alpha_i \rightarrow \beta_i$ for $i = 1, \dots, n$ also satisfies $\alpha \rightarrow \beta$ (α and β stand here for sets of attributes/columns).

Exercise

- The following relation is used for storing orders:

`ORDER(ORD_NO, DATE, CUST_NO, PROD_NO, QUANTITY)`

- Please list FDs which hold for this relation:

One order can be about multiple products.

- Do these FDs imply the following FD?

`ORD_NO, PROD_NO → DATE`

- Determine a key of the relation `ORDER`.

Overview

1. Introduction (Anomalies)

2. Functional Dependencies

3. BCNF

Motivation (1)

- Consider again the example:

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

- As noted above, the FD $INAME \rightarrow PHONE$ leads to problems, one of which is the redundant storage of certain facts (e.g. the phone number of “Brass”).

Motivation (2)

- Actually, any FD $A_1, \dots, A_n \rightarrow B_1, \dots, B_m$ will cause redundant storage unless A_1, \dots, A_n is a key, so that each combination of attribute values for A_1, \dots, A_n can occur only once.

Trivial constraints must be excluded here, i.e. at least one of the B_i should not appear among the A_j .

- In general, whenever one stores redundant data, one needs a constraint that ensures that the different copies of the same information remain consistent (i.e. do not contradict each other).

Motivation (3)

- In the cases of redundant data considered here, the constraints are precisely the FDs, e.g. **INAME**→**PHONE**.
- But FDs are not one of the standard constraints of the relational model. They cannot be specified in the `CREATE TABLE` statement of current DBMSs.
- Only the special case of keys is supported.
- Thus: Avoid (proper) FDs by transforming them into key constraints. This is what normalization does.

Motivation (4)

- 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**→**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.

It is not a problem if a relation has two keys: Then there are only two ways to identify the same concept.

Boyce-Codd Normal Form

- A Relation R is in BCNF if and only if all its FDs are already implied by key constraints.

Thus, a relation in BCNF does not require FD constraints, only key constraints.

- I.e. for every FD " $A_1, \dots, A_n \rightarrow B_1, \dots, B_m$ " one of the following conditions must hold:
 - ◇ The FD is trivial, i.e. $\{B_1, \dots, B_m\} \subseteq \{A_1, \dots, A_n\}$.
 - ◇ The FD follows from a key, because $\{A_1, \dots, A_n\}$ or some subset of it is already a key.

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

Examples (1)

- COURSES(CRN, TITLE, INAME, PHONE) with the FDs
 - ◇ $CRN \rightarrow TITLE, INAME, PHONE$
 - ◇ $INAME \rightarrow PHONE$is not in BCNF because the FD “ $INAME \rightarrow PHONE$ ” 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:
 - ◇ $CRN \rightarrow TITLE, INAME$ 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 → TITLE, DAY, TIME, ROOM`
- ◇ `DAY, TIME, ROOM → 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 $\text{PRODUCT}(\text{NO}, \text{NAME}, \text{PRICE})$ has these FDs:
 - (1) $\text{NO} \rightarrow \text{NAME}$
 - (2) $\text{NO} \rightarrow \text{PRICE}$
 - (3) $\text{PRICE}, \text{NAME} \rightarrow \text{NAME}$
 - (4) $\text{NO}, \text{PRICE} \rightarrow \text{NAME}$
- 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.

Exercises

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

(1) `STUD_ID, EX_NO → POINTS`

(2) `EX_NO → MAX_POINTS`

First determine a key.

- Is the relation

`ORDER(ORD_NO, DATE, CUST_NO, PROD_NO, QUANTITY),`

for which you determined FDs above, in BCNF?

Splitting Relations

- A table which is not in BCNF can be split into two tables (“decomposition”), e.g. split COURSES into
COURSES_NEW(CRN, TITLE, INAME → INSTRUCTORS)
INSTRUCTORS(INAME, PHONE)
- General case: If $A_1, \dots, A_n \rightarrow B_1, \dots, B_m$ violates BCNF, create a relation $S(\underline{A_1}, \dots, \underline{A_n}, B_1, \dots, B_m)$ and remove B_1, \dots, B_m from the original relation.

B_1, \dots, B_m should be all attributes that are functionally determined by A_1, \dots, A_n . No B_i should appear among the A_j . A_1, \dots, A_n become a foreign key in the original relation. In unusual cases (multiple violations), it is necessary to repeat the splitting step with one or both of the resulting relations. Then also implied FDs must be considered.