

---

# DATABASTEKNIK - 1DL116

Fall 2003

An introductory course on database systems

<http://user.it.uu.se/~udbl/dbt-ht2003/>

Kjell Orsborn  
Uppsala Database Laboratory  
Department of Information Technology, Uppsala University,  
Uppsala, Sweden

---

# Introduction to Normalization

Elmasri/Navathe ch 10

Kjell Orsborn

Department of Information Technology  
Uppsala University, Uppsala, Sweden

# Normalization

## (Elmasri/Navathe ch. 10)

- Relational database design - normalization of relational schemas or “we want to have good relations . . .” (ch. 10)
  - Problems with bad schema designs
  - Functional dependencies
  - Normal forms

# Problems in schema design

- Unclear semantics
- Redundancy
- Null values
- Modification problems
  - updates
  - insertion
  - deletion

# Design of relational data bases

- To design a relational database we start from an E-R model and design a number of relational schemas as e.g.:
  - *emps(ename, salary, dept, dname, dept#, mgr)*
  - *supp\_info(sname, iname, saddr, price)*
  - *items(iname, item#, dept)*
  - *customers(cname, addr, balance, o#, date)*
  
  - *includes(o#, iname, quantity)*

# Bad design causes problems

- We have chosen here to combine the schemas for *SUPPLIES* and *SUPPLIERS* and exchange them with one schema that contain all information about suppliers:

*SUPPLIERS(SNAME, SADDR)*

*SUPPLIES(SNAME, INAME, PRICE)*

*SUPP\_INFO(SNAME, INAME, SADDR, PRICE)*

- Due to this decision we will run into several problems with:
  - Redundancy
  - Update anomalies
  - Insertion anomalies
  - Deletion anomalies

# Redundancy

*SUPP\_INFO(SNAME, INAME, SADDR, PRICE)*

- **Redundancy:**
  - When each supplier supplies several products, the address will be repeated for each product.

## Update anomaly - inconsistency

*SUPP\_INFO(SNAME, INAME, SADDR, PRICE)*

- When you have redundant data you risk, for example, when *updating* the address in the table, to miss updating all occurrences of the redundant values (here several occurrences of address values for the same supplier).
- The result will be that we get *inconsistencies* among data in the database.



# Insertion anomaly

*SUPP\_INFO(SNAME, INAME, SADDR, PRICE)*

- We can not insert the address for a supplier that do not currently supply any product.
  - One can of course assign *null* values *INAME* and *PRICE*, but the one must remember to remove these null values later when additional information exist. Hence, no tractable solution!
- *Furthermore*, since *INAME* is part of the key for this relation this is not a good solution. It is probably not even allowed to assign null values to key attributes.

# Deletion anomaly

*SUPP\_INFO(SNAME, INAME, SADDR, PRICE)*

- If all *tuples are removed*, that corresponds to items that a specific supplier provide, then *all* information about that supplier will disappear.

# How to avoid these problems?

- In order to avoid all these problems, we divide the relation (table) in the following manner:

*SUPPLIERS(SNAME, SADDR)*

*SUPPLIES(SNAME, INAME, PRICE)*

- Disadvantages:
  - To retrieve addresses for suppliers that provide bananas, we must perform an expensive join operation, *SUPPLIERS \* SUPPLIES*, that we could get away with if we instead had the relation:  
*SUPP\_INFO(SNAME, INAME, SADDR, PRICE)*
  - The operations *SELECT* and *PROJECT* that is much cheaper will do the job in the latter case.

# Normalization

- What principles do we have to follow to systematically end up with a good schema design.
  - In 1972 Codd defined a set of conditions that put various requirements on a relation.
  - To fulfill these conditions, the relation schema is in several steps divided into smaller schemas.
- This process is called **normalization**.
- We need to study the following concepts for performing the normalization:
  - Functional dependencies
  - Normal forms for relations

# Functional dependency (FD)

- Let  $R$  be a relation schema with attributes  $A_1, \dots, A_n$  and let  $X$  and  $Y$  be subsets of  $\{A_1, \dots, A_n\}$ .
- Let  $r(R)$  determine a relation (instance) of the schema  $R$ .

**We say that  $X$  functionally determines  $Y$ , and we write**

$$X \twoheadrightarrow Y$$

**if for every pair of tuples  $t_1, t_2 \in r(R)$  and for all  $r(R)$  the following hold:**

**If  $t_1[X] = t_2[X]$  it holds that  $t_1[Y] = t_2[Y]$**

- The attribute set  $Y$  is said to be dependent of the attribute set  $X$ .
- There is a functional dependency between  $X$  and  $Y$ .

# Functional dependency - properties

- If  $X$  is a candidate key for  $R$  we have:

$X \twoheadrightarrow Y$  for all  $Y \subseteq \{A_1, \dots, A_n\}$ .

$X \twoheadrightarrow Y$  do not need to be the empty set.

Assume that  $R$  represents an N:1 relation between the entity types  $E1$  and  $E2$ .

It is further assumed that  $X \subseteq \{A_1, \dots, A_n\}$  constitutes a *key* for  $E1$  and  $Y$  constitutes a *key* for  $E2$ ,

Then we have the dependency  $X \twoheadrightarrow Y$  but not  $Y \twoheadrightarrow X$ .

Om  $R$  skulle vara en 1:1 relation då gäller även  $Y \twoheadrightarrow X$ .

- A functional dependency is a property of  $R$  and not of any relation  $r(R)$ . The semantic for the participating attributes in a relation decides if there exists a FD or not.

# Example

- The simplest form of FD is between the key attributes and the other attributes in a schema:

$\{\underline{SNAME}\} \twoheadrightarrow \{SADDR\}$	<i>SUPPLIERS</i>
$\{\underline{SNAME}, \underline{INAME}\} \twoheadrightarrow \{PRICE\}$	<i>SUPPLIES</i>
$\{\underline{CNAME}\} \twoheadrightarrow \{CADDR, BALANCE\}$	<i>CUSTOMERS</i>
$\{\underline{SNAME}\} \twoheadrightarrow \{SNAME\}$	

- A slightly more complex FD:

$$\{\underline{SNAME}, \underline{INAME}\} \twoheadrightarrow \{SADDR, PRICE\}$$

- This dependency is the combination of the first two above. We are able to combine these two because we understand the concepts supplier, item, address and price and the relationships among them.

# Full functional dependency - FFD

- A functional dependency  $X \twoheadrightarrow Y$  is termed FFD if there is no attribute  $A \in X$  such that  $(X - \{A\}) \twoheadrightarrow Y$  holds.
- In other words, a FFD is a dependency that do not contain any unnecessary attributes in its *determinant* (i.e. the left-hand side of the dependency).

*SUPP\_INFO*(SNAME, INAME, SADDR, PRICE)

$\{\underline{SNAME}, \underline{INAME}\} \twoheadrightarrow \{PRICE\}$	FFD
$\{\underline{SNAME}, \underline{INAME}\} \twoheadrightarrow \{SADDR\}$	Not FFD
$\{\underline{SNAME}\} \twoheadrightarrow \{SADDR\}$	FFD
compare: $f(x,y) = 3x + 2$	Not FFD
$\llbracket f \rrbracket(x) = 3x + 2$	FFD



# Prime attribute

- Definition: an attribute that is a member in any of the candidate keys is called a **prime** attribute of R.
- For example:
  - In the schema  $R(CITY, STREET, ZIPCODE)$  all attributes are prime attribute, since we have the dependencies  $CS \twoheadrightarrow Z$  and  $Z \twoheadrightarrow C$  and both  $CS$  and  $SZ$  are keys.
  - In the schema  $R(A, B, C, D)$  with the dependencies  $AB \twoheadrightarrow C$ ,  $B \twoheadrightarrow D$  och  $BC \twoheadrightarrow A$ , are both  $AB$  and  $BC$  candidate keys. Therefore are  $A$ ,  $B$  and  $C$  prime attribute but not  $D$ .
- Attributes that is not part of any candidate key is called **non-prime** or **non-key** attributes.



# First normal form - 1NF

- Only atomic values are allowed as attribute values in the relational model.
  - The relations that fulfill this condition are said to be in 1NF

# Second normal form - 2NF

- A relation schema  $R$  is in 2NF if:
  - It is in 1NF
  - Every non-key attribute  $A$  in  $R$  is FFD of each candidate key in  $R$ .

# Third normal form - 3NF

- A relation schema  $R$  is in 3NF if:
  - It is in 2NF
  - No non-key attribute  $A$  in  $R$  is allowed to be FFD of any other non-key attribute.

# Boyce-Codd normal form - BCNF

- A relation schema  $R$  is in BCNF if:
  - It is in 1NF
  - Every determinant  $X$  is a candidate key.
- The difference between BCNF and 3NF is that in BCNF, a prime attribute  $A$  in  $R$  can not be FFD of a non-key (or non-prime) determinant  $X$ . Therefore BCNF is a more strict condition.

# Additional normal forms

- There are a number of other normal forms available
- These explores additional data dependencies such as:
  - multi-valued dependencies (4NF)
  - join dependencies (5NF)

# Multi-valued dependencies

- Consider the relation *classes(course, teacher, book)*

<u>course</u>	<u>teacher</u>	<u>book</u>
database	Tore	Elmasri
database	Tore	Ullman
database	Martin	Elmasri
database	Martin	Ullman
operating system	Martin	Silberschatz
operating system	Martin	Shaw

- The *book* attribute specifies required literature independent of teacher.
- The relation fulfills BCNF, but still every teacher must be supplied two times.

# Lossless join

- We decompose a schema  $R$  with functional dependencies  $D$  into schemas  $R1$  and  $R2$
- $R1$  and  $R2$  is called a lossless-join decomposition (in relation to  $D$ ) if  $R1 * R2$  contain the same information as  $R$ .
  - no spurious tuples are generated when we perform the join between the decomposed relations.
  - no information is lost when the relation is decomposed.



# 3NF/BCNF

- BCNF is a strong condition. It is not always possible to transform (through decomposition) a schema to BCNF and keep the dependencies.
- 3NF has the most of BCNF's advantages and can still be fulfilled without giving up dependencies or the lossless-join property for the relation schema.
- Observe that 3NF admits some sort of redundancy that BCNF do not allow. BCNF requires that one eliminates redundancy caused by functional dependencies.