Normalization of a Relational Model: Preliminaries

- Normalization is a process of decomposition of a set of relations into a new set of relations
- The new set of relations is in a predetermined Normal Form
- There are several Normal Forms: 1NF, 2NF, 3NF, BCNF, ... (for first, second, third and Boyce-Codd)
- Each of them has certain defining properties

  Properties that have to do with preventing issues such as data redundancy, and update anomalies
- The process and the properties of a Normal Form heavily depend on explicit and implicit semantic constraints

  Generically called just (data) "dependencies"

- In principle, we could start with the (single) wide universal relation; or with a given set of relations

  Plus a set of dependencies

- One can check if a given relational schema (i.e. a set of relation schemas) is in the desired normal form

  If not, it can be further refined accordingly ...

- Particularly prominent are sets of Functional Dependencies (FDs)

  They generalize the notion of a key for a relation

- We will concentrate first on normal forms -and the methodology for reaching them- that rely on FDs

- We need some notation: $R(\mathbf{A})$ denotes a relation schema, i.e. one relational predicate with the list of its attributes $\mathbf{A}$

  We will use capital letter in boldface for sets of attributes

  E.g. $\text{Wine}(\text{Wine\#}, \text{Grape}, \text{Vintage}, \text{Percentage})$

- Given an extension $R[D]$ for R in a database D, a tuple $\mathbf{t}$ is an element of $R[D]$

Wine	Wine#	Grape	Vintage	Percentage
	003	cabernet	2018	13
	011	chardonay	2020	12
	333	chardonay	2019	12

← three tuples

- For $\mathbf{B} \subseteq \mathbf{A}$, $\mathbf{t}[\mathbf{B}]$ is the restriction of $\mathbf{t}$ to $\mathbf{B}$

  For the first tuple $\mathbf{t}_1$ above, with $\mathbf{B} = \{\text{Grape}, \text{Percentage}\}$, $\mathbf{t}_1[\mathbf{B}] = \langle \text{cabernet}, 13 \rangle$

- As always, two tuples (or subtuples) are equal if they are equal componentwise

  Above: $\mathbf{t}_2[\text{Grape}, \text{Percentage}] = \mathbf{t}_3[\text{Grape}, \text{Percentage}]$

- Definition: $R(\mathbf{A})$ a relation schema; $\mathbf{B}, \mathbf{C} \subseteq \mathbf{A}$

  (subsets or sublists of different attributes)

  Example:

Drinking	Drinker#	Surname	Fname	Type	Wine#	Grape	Vintage	Percentage	Date	Quantity

  B = Wine#; C = Grape, Vintage, Percentage

- **C** functionally depends on **B**, denoted $R\colon \mathbf{B} \longrightarrow \mathbf{C}$
  iff, for every instance of (extension for) R and tuples $\mathbf{t}_1$, $\mathbf{t}_2$ in it:

$$\mathbf{t}_1[\mathbf{B}] = \mathbf{t}_2[\mathbf{B}] \;\Rightarrow\; \mathbf{t}_1[\mathbf{C}] = \mathbf{t}_2[\mathbf{C}]$$

- Example: $\text{Drinking}\colon \text{Wine\#} \rightarrow \text{Grape}, \text{Vintage}, \text{Percentage}$

  It cannot be the case that two tuples with the same wine number have different values for any of the three attributes on the RHS

  This portion of the wide table violates the FD

Wine#	Grape	Vintage	Percentage
003	cabernet	2018	13
011	chardonay	2020	12
003	chardonay	2018	13

Drinking	Drinker#	Surname	Fname	Type	Wine#	Grape	Vintage	Percentage	Date	Quantity

- Also: Drinking : Drinker# $\rightarrow$ Surname, Fname, Type
- Drinking : Drinker#, Wine# $\rightarrow$ Surname, Fname, Type, Grape, Vintage, Percentage
- Drinking : Drinker#, Wine# $\overset{?}{\rightarrow}$ Date, Quantity

  It does not make much sense!

  A drinker can drink a wine on multiple dates and different quantities Drinking : Drinker#, Wine# $\not\rightarrow$ Date, Quantity
- This is application dependent
- Relation schema $R(\mathbf{A})$ and $\mathbf{B} \subseteq \mathbf{A}$:

  $\mathbf{B}$ is a key for R if $R: \mathbf{B} \rightarrow \mathbf{A}$ (equivalently, $R: \mathbf{B} \rightarrow (\mathbf{A} \setminus \mathbf{B})$)

  All the attributes of the relation functionally depend on $\mathbf{B}$

Drinking	Drinker#	Surname	Fname	Type	Wine#	Grape	Vintage	Percentage	Date	Quantity

- None of these are a key for Drinking:

  {Drinker#}, {Wine#}, {Drinker#, Wine#}

- {Drinker#, Wine#, Date, Quantity} is a key for Drinking (the only one)

- Example:

Students	St#	RUT	Name	LName	Address	Study	Year

- Both St# and RUT are keys for Students

- But not {St#, RUT}

- By definition, a key is minimal: No subset of a key is a key

- When a relation (schema) has several keys, we can choose one a primary key, and the others become candidate keys

Some Remarks about FDs:

- Every attribute A in a relation schema R functionally determines itself:
$$R:\ A \rightarrow A \tag{1}$$

- For example: $St\# \rightarrow St\#$

  Obvious: If two tuples coincide on A, then they coincide on A

- (1) is easy to prove by checking the definition on page 55:

  For any tuples $\mathbf{t}_1, \mathbf{t}_2 \in R[D]$: $\mathbf{t}_1[A] = \mathbf{t}_2[A] \ \Rightarrow\ \mathbf{t}_1[A] = \mathbf{t}_2[A]$

  (a propositional tautology of the form $p \Rightarrow p$)

- We accept (1) as a "basic axiom" of FDs

  (1) is true in (is satisfied by) every instance of a relation that has attribute A

- Notation: The union (or juxtaposition) of two sets of attributes **B** and **C** is denoted with **BC**

- With relation schema $R(\ldots, A, \ldots, B, \ldots)$, what about this?

$$R\colon\ AB \rightarrow A \qquad (2)$$

- For example: $Students\colon\ Name, Study \rightarrow Study$

- It is easy to verify that every instance of R makes the FD (2) true (satisfies it)

  If two tuples coincide on A and B, then they coincide on A

- (1) and (2) are like logical axioms or valid logical formulas, i.e. always true

- More precisely: FDs of the forms (1) and (2) are true in (satisfied by) every instance of a relational predicate (with those attributes)

- (1) and (2) are particular cases of FDs of the form:

  If $\mathbf{B} \subseteq \mathbf{C}$, then the FD $\mathbf{C} \rightarrow \mathbf{B}$ holds

  Equivalently: $\mathbf{BX} \rightarrow \mathbf{X}$ (3)

- As with logical implication, a set of FDs may imply other FDs that have not been explicitly stated
- Those implicit FDs have to be taken into account, e.g. for normalization
- For example, consider relation schema $R(A, B, C)$ with declared FDs $A \rightarrow B$ and $B \rightarrow C$

  The FD $A \rightarrow C$ is implicit
- It "follows" from -or is "implied" by- the first two
- Intuitively acceptable and natural, but in what sense?
  What does this mean?
- We have to prove this transitivity property of FDs
- How does the proof go?

- We have to reason as usual in Math

  The idea is as follows ...

- Consider an arbitrary instance $R[D]$ for relation predicate $R(A, B, C, \cdots)$ in database instance D
- Assume that: $R\colon\ A \to B,\quad B \to C$ (*)
- Prove that the following holds in $R[D]$: $A \to C$ (!)
- Appeal to definition of FD on page 55

  Consider arbitrary tuples $\mathbf{t}_1, \mathbf{t}_2 \in R[D]$

  Assume that: $\mathbf{t}_1[A] = \mathbf{t}_2[A]$ (**)

- To prove: $\mathbf{t}_1[C] = \mathbf{t}_2[C]$ (this establishes (!))
- From (*) and (**):

  First: $\mathbf{t}_1[B] = \mathbf{t}_2[B]$

  Next: $\mathbf{t}_1[C] = \mathbf{t}_2[C]$

- What did we do?

- What kind of proof is this?

- We proved that whenever an instance $R[D]$ satisfies the FDs $A \to B, \quad B \to C$, then the instance also satisfies the FD: $A \to C$

- Reasoning at the *semantic level* (metalevel, "human" level)

  As usual in Mathematics, appealing to arbitrary structures (here, database instances)

- Actually, we proved that the FD $A \to C$ is implied (entailed) by $\mathcal{F} = \{A \to B, \; B \to C\}$

  Because it is satisfied by every instance of R that satisfies the FDs in $\mathcal{F}$

- Actually, this is the general definition of implication of FDs:

  For a set $\mathcal{F}$ of FDs and an FD ψ on relation schema R, ψ is a consequence of $\mathcal{F}$ iff ψ is satisfied by every instance $R[D]$ that satisfies $\mathcal{F}$

  Notation: $\mathcal{F} \models \psi$

- Above we established: $\{A \rightarrow B,\ B \rightarrow C\} \models A \rightarrow C$

- Generalizing the previous example, an important problem in DBs is the following:

  If a certain set of FDs $\mathcal{F}$ holds for a relation schema, what other FDs hold for that schema?

  Equivalently, what FDs are implied by $\mathcal{F}$?

  Equivalently, for what FDs ψ it holds $\mathcal{F} \models \psi$?

- Obtaining implied FDs is important

  Normalization depends on explicit and implicit FDs

- Proving implication of FDs can be a cumbersome process
- Implication of FDs appeals to all possible instances of a relation schema; it is a semantic notion
- Is there a purely symbolic alternative?

  That symbolically processes the FDs, e.g. $A \rightarrow B, \quad B \rightarrow C$ (as symbolic expressions)

  In such a way that the entailed FD is symbolically derived, here $A \rightarrow C$
- Something that a computer could do?
- The same problem appears for other classes of dependencies ....