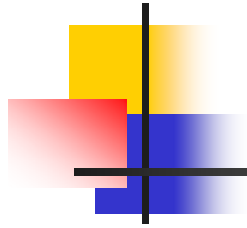


# FUNCTIONAL DEPENDENCIES

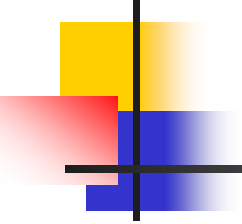


Attributes are grouped together to form relations (tables)

Tables are put together to form a relational database schema

The above process is specified by the database designer or by mapping an ER/EER diagram to a relational schema

We need to be able to measure how “**good**” a particular grouping is, w.r.t. other possible groupings



The quality of a relational database schema is evaluated at two levels:

---

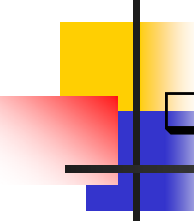
- ❑ the *logical* (conceptual) level (how clear is the meaning of the attributes → easier to formulate queries)
- ❑ the *implementation* (storage) level (how the tuples are stored and updated → more efficient query execution)

Database design can be done using 2 methodologies:

*top-down* (start with some relations and refine the decomposition until all requirements are met)

*bottom-up* (start with basic relationships between attributes and build up relations)

# GENERAL DESIGN GUIDELINES

- 
- A relation should correspond to a single entity type or a single relationship type
  - A database should be designed so as to avoid update (insert, delete, modify) anomalies
  - Limit the number of null values in tuples (problems with JOIN, COUNT, SUM etc)
  - Design relations schemas that can be JOINed with equality conditions on either primary or foreign keys (spurious tuples)



# Concept of FD

---

- *Functional Dependency*: a constraint between two sets of attributes from the database
- *Universal relation schema*: assemble all the attributes in one big relation  $R = \{A_1, \dots, A_n\}$  (theoretical concept)
- Used to define *1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup>, BCNF*

# Definition of FD

- $X \rightarrow Y$  ( $X, Y$  subsets of  $\mathbf{R}$ ) specifies a limitation on the possible tuples in a potential relation state  $\mathbf{r}$  of  $\mathbf{R}$
- For any tuples  $t_1, t_2$  with  $t_1[X] = t_2[X]$  we must also have  $t_1[Y] = t_2[Y]$
- The values of the  $Y$  component of a tuple in  $\mathbf{r}$  **are determined** by the values of the  $X$  component. The values of the  $X$  component **uniquely (functionally) determine** the values of the  $Y$  component

# FD Terminology



---

- There is an FD from  $X$  to  $Y$
- $Y$  is functionally dependent on  $X$
- $X$  is the *left-hand side* of the FD
- $Y$  is the *right-hand side* of the FD
- **$X$  functionally determines  $Y$  in  $R$  iff every two tuples that agree on their  $X$ -values, agree on their  $Y$ -values**

# Remarks on FDs



---

- If  $X$  is a candidate key for  $R$ , then  $X \rightarrow Y$  for any subset  $Y$  of  $R$
- $X \rightarrow Y$  in  $R$ , does not mean  $Y \rightarrow X$  in  $R$
- An FD is a semantic property of the attributes
- **Legal relation:** is a relation state that satisfies the specified FDs
- FDs specify constraints that must always hold

# Examples of FDs

- $\{Province, \#DriverLicence\} \rightarrow SIN$
- In the **EMPLOYEEPROJECT** relation schema
  - $SIN \rightarrow ENAME$
  - $PNUMBER \rightarrow \{PNAME, PLOCATION\}$
  - $\{SIN, PNUMBER\} \rightarrow HOURS$
- Diagrammatic Notation for FDs:  
FDs (horizontal lines)   lhs (vertical lines)  
rhs (pointing arrows)

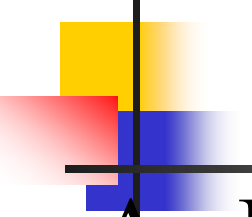


# A subtle point about FDs

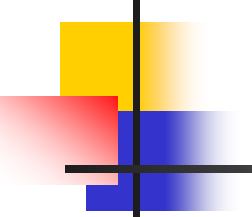
- ❑ An FD is a property of the relation schema, **not** of a particular relation state
- ❑ As a consequence, an FD cannot be inferred from a given relation state **r**
- ❑ However, it is sufficient to exhibit a counterexample to show that a certain FD cannot hold
- ❑  $TEXT \rightarrow COURSE$  ?  $COURSE \rightarrow TEXT$

# Inference Rules for FDs

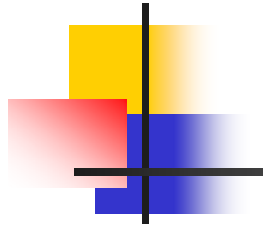
- $F$  denotes the set of all FDs in  $R$
- Other FDs may be deduced from  $F$
- $F^+$  (the closure of  $F$ ) is the set of all possible FDs deduced from  $F$
- Example:  $F = \{ SIN \rightarrow \{ENAME, ADDRESS, BDATE, DNUMBER\}, DNUMBER \rightarrow \{DNAME, DMGRSIN\} \}$
- We can infer the following  $F^+$  elements:  
 $SIN \rightarrow \{DNAME, DMGRSIN\}, SIN \rightarrow SIN, DNUMBER \rightarrow DNAME$

- 
- 
- An FD  $X \rightarrow Y$  is *inferred from* a set of FDs  $F$  on  $R$  if  $X \rightarrow Y$  holds in every legal relation state  $r$  of  $R$  (if  $r$  satisfies all the FDs in  $F$ , then  $r$  satisfies  $X \rightarrow Y$ )
  - A systematic way to establish FDs is provided by a set of *inference rules* that can be used to infer new FDs from given ones
  - **NOTATIONS:**  $F \models X \rightarrow Y \ \& \ \{X, Y\} \rightarrow Z \quad XY \rightarrow Z$

# A SET OF INFERENCE RULES

- 
- IR1 (reflexivity)  $\mathcal{Y}$  subset of  $\mathcal{X}$ ,  $\mathcal{X} \rightarrow \mathcal{Y}$
  - IR2 (augmentation)  $\{\mathcal{X} \rightarrow \mathcal{Y}\} \models \mathcal{XZ} \rightarrow \mathcal{YZ}$
  - IR3 (transitivity)  $\{\mathcal{X} \rightarrow \mathcal{Y}, \mathcal{Y} \rightarrow \mathcal{Z}\} \models \mathcal{X} \rightarrow \mathcal{Z}$
  - IR4 (decomposition)  $\{\mathcal{X} \rightarrow \mathcal{YZ}\} \models \mathcal{X} \rightarrow \mathcal{Y}$
  - IR5 (union)  $\{\mathcal{X} \rightarrow \mathcal{Y}, \mathcal{X} \rightarrow \mathcal{Z}\} \models \mathcal{X} \rightarrow \mathcal{YZ}$
  - IR6 (pseudo-transitivity)  
 $\{\mathcal{X} \rightarrow \mathcal{Y}, \mathcal{WY} \rightarrow \mathcal{Z}\} \models \mathcal{WX} \rightarrow \mathcal{Z}$

# Comments on the Inference Rules



- Applying IR4 repeatedly we can decompose an FD  $X \rightarrow \{A_1, A_2, \dots, A_n\}$  into a set of FDs:  $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$
- Applying IR5 repeatedly we do the opposite
- IR1, ..., IR6 can be proved from the definition **or** by contradiction **or** by using previously proved rules

# Armstrong's Axioms



- {IR1, IR2, IR3} Armstrong's Axioms **AA**
- **AA** are sound (any FD inferred from **F** using **AA** will hold in any relation state **r** of **R**, that satisfies a given set of FDs **F**)
- **AA** are complete (we can compute the closure  $F^+$  of a given set of FDs **F**, using exclusively **AA** )

# Usage of AA in database design



---

- Specify a set  $\mathbf{F}$  of semantically obvious FDs on the attributes of  $\mathbf{R}$
- Use **AA** to infer additional FDs (2 steps)
  - Determine each set  $\mathcal{X}$  of attributes that appear as lhs of FDs in  $\mathbf{F}$
  - Determine the set  $\mathcal{X}^+$  (closure of  $\mathcal{X}$  under  $\mathbf{F}$ ) of all attributes functionally determined by  $\mathcal{X}$  based on  $\mathbf{F}$



# Algorithm to compute $\mathcal{X}^+$

---

**INPUT:** a set of FDs  $\mathbf{F}$ , a set  $\mathcal{X}$  of lhs of elements in  $\mathbf{F}$

**OUTPUT:** the set of attributes  $\mathcal{X}^+$  (closure of  $\mathcal{X}$  under  $\mathbf{F}$ )

**STEP 1.** Assign  $\mathcal{X}^+ := \mathcal{X}$       (*justification: IR1*)

**STEP 2.** Repeat

$\text{old}\mathcal{X}^+ := \mathcal{X}^+$  (for loop justification: IR3, IR4)

    for each FD  $\mathcal{Y} \rightarrow \mathcal{Z}$  in  $\mathbf{F}$  do

        if  $\mathcal{Y}$  subset of  $\mathcal{X}^+$  then  $\mathcal{X}^+ := \mathcal{X}^+ \mathbf{U} \mathcal{Z}$

Until  $\mathcal{X}^+ = \text{old}\mathcal{X}^+$



# Example of application of the closure computation algorithm

- In the **EMPLOYEEPROJECT** relation schema

- $\mathbf{F} = \{SIN \rightarrow ENAME, PNUMBER \rightarrow \{PNAME, PLOCATION\}, \{SIN, PNUMBER\} \rightarrow HOURS\}$

- $\{SIN\}^+ = \{SIN, ENAME\}$

- $\{PNUMBER\}^+ = \{PNUMBER, PNAME, PLOCATION\}$

- $\{SIN, PNUMBER\}^+ = \{SIN, PNUMBER, ENAME, PNAME, PLOCATION, HOURS\}$

In general  $\chi^+ \cup \gamma^+$  different than  $(\chi \cup \gamma)^+$

# Equivalent sets of FDs

- Consider  $\mathbf{E}$ ,  $\mathbf{F}$  two sets of FDs
  - $\mathbf{E}$  is covered by  $\mathbf{F}$  ( $\mathbf{F}$  covers  $\mathbf{E}$ ),  $\mathbf{E}$  subset of  $\mathbf{F}^+$
  - $\mathbf{E}$ ,  $\mathbf{F}$  equivalent,  $\mathbf{E}^+ = \mathbf{F}^+$  in words:
  - Every FD in  $\mathbf{F}$  can be deduced from  $\mathbf{E}$  and vice-versa
- Determine whether  $\mathbf{F}$  covers  $\mathbf{E}$ : a) compute  $\mathcal{X}^+$  wrt  $\mathbf{F}$  for every FD  $\mathcal{X} \rightarrow \mathcal{Y}$  in  $\mathbf{E}$       b) check  $\mathcal{Y}$  subset of  $\mathcal{X}^+$   
c) if b) is true for every FD in  $\mathbf{E}$ , then  $\mathbf{F}$  covers  $\mathbf{E}$ .
- Determine whether  $\mathbf{E}$ ,  $\mathbf{F}$  are equivalent:  $\mathbf{E}$  covers  $\mathbf{F}$  and  $\mathbf{F}$  covers  $\mathbf{E}$ .

# Minimal Sets of FDs

A set of FDs  $\mathbf{F}$  is *minimal* if:

- The rhs of every FD in  $\mathbf{F}$  is a single attribute
- If we replace a FD  $X \rightarrow A$  with a FD  $Y \rightarrow A$  where  $Y$  is a proper subset of  $X$ , we will obtain a set of FDs *not equivalent* to  $\mathbf{F}$
- If we remove a FD from  $\mathbf{F}$ , we will obtain a set of FDs *not equivalent* to  $\mathbf{F}$

Minimal Set  $\rightarrow$  canonical form & no redundancies

A *minimal cover* of a set of FDs  $\mathbf{F}$ , is a minimal set of FDs  $\mathbf{G}$ , that is also equivalent to  $\mathbf{F}$

# Algorithm to compute a minimal cover

**INPUT:** a set of FDs  $F$

**OUTPUT:** a minimal cover  $G$  of  $F$

---

**STEP 1.** Assign  $G := F$

**STEP 2.** Replace each FD  $X \rightarrow \{A_1, A_2, \dots, A_n\}$  in  $G$   
by  $n$  FDs  $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$

**STEP 3.** For each FD  $X \rightarrow A$  in  $G$ ,  
For each attribute  $B$  of  $X$ ,  
if  $(G - \{X \rightarrow A\}) \cup \{(X - \{B\}) \rightarrow A\}$  equiv. to  $G$   
then replace  $X \rightarrow A$  by  $X - \{B\} \rightarrow A$  in  $G$

**STEP 4.** For each remaining FD  $X \rightarrow A$  in  $G$ ,  
if  $G - \{X \rightarrow A\}$  is equiv. to  $G$   
then remove  $X \rightarrow A$  from  $G$



# Normalization

---

- Use FDs to describe the semantics of relations schemas
- Assume that a set of FDs and a **primary key** are given for each relation
- Define 1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup> NF, BCNF (and higher NF)
- **Evaluate** each relation against each NF and **decompose** it (top-down design) in order to obtain relations that satisfy NF



---

## ■ Normalization Process

- Framework to analyze relations schemas based on their keys and FDs among attributes
- Normal Form tests carried out on relation schemas to normalize them to any desired degree

### *Normal Form of a relation:*

the *highest* normal form condition satisfied by the relation

Other properties that a good relational schema must have:

*nonadditive join* (spurious tuples), *dependency preservation*

# Surprise Quiz !



- *Superkey*
- *Key*
- *Candidate Key*
- *Primary Key*
- *Foreign Key*
- Prime attribute*
- Nonprime attribute*

Some of these definitions are useful in defining the 1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup> NF introduced by Codd in 1972

member of some candidate key of the relation



# First Normal Form 1NF

---

- **1NF** ATOMICITY OF ATTRIBUTE DOMAINS
- Attribute values allowed by **1NF** are single indivisible atomic values from the attribute domain
- In particular, **1NF** forbids
  - (a) multi-valued attributes
  - (b) nested relations
  - (c) relations as values of tuples



# Normalization into 1NF (I)



---

- When a relation is not in **1NF** there are 3 main techniques to normalize it using the attribute **A** that violates **1NF** (case **A** is a composite attribute)
  - Remove **A** and place it in a new relation together with the primary key
  - Expand the key, so that there will be a separate tuple for each atomic value of **A** (pb: introduces redundancy)
  - If a max number of values **n**, is known for **A**, replace **A**, with **n** atomic attributes (pb: introduces null values)

# Normalization into 1NF (II)



---

- When a relation is not in **1NF** there is one other technique to normalize it using the attribute **A** that violates **1NF** (case **A** is a multi-valued attribute)
  - Remove **A** into a new relation and propagate the primary key into this new relation
  - The primary key of the new relation will combine the partial key of the nested relation and the primary key of the original relation
  - This process can be applied recursively to denest relations

# Second Normal Form 2NF

- **2NF FULL FD OF ALL NONPRIME ATTRIBUTES ON PRIMARY KEY**
- Full FD:  $X \rightarrow Y$  is a Full FD if removal of any attribute of  $X$  destroys the FD
- Partial FD:  $X \rightarrow Y$  is a Partial FD if removal of some attribute of  $X$  does not destroy the FD
- If the primary key contains only one attribute then the relation satisfies the **2NF** criterion
- 2NF is concerned with FDs whose lhs attributes are parts of the primary key

# Normalization into 2NF



---

- If a relation schema is not in 2NF, it can be *normalized to 2NF*: decomposed into a number of relations in which nonprime attributes are fully functionally dependent on the primary key.
- 2NF normalization recipe:
  - Set up a new relation for each partial key with its dependent attribute(s)
  - Keep a relation with the original primary key and its fully functionally dependent attribute(s)



# Third Normal Form 3NF

---

- **3NF** NO NONPRIME ATTRIBUTE IS TRANSITIVELY DEPENDENT ON THE PRIMARY KEY OF R

- *Transitive FD:*  $X \rightarrow Y$  is a Transitive FD in R if there is a set of attributes Z (that is neither a candidate key nor a subset of any key of R) such that both  $X \rightarrow Z$  and  $Z \rightarrow Y$  are valid FDs.

- *Example:*  $SIN \rightarrow DMGRSIN$  is a transitive FD ( $Z = DNUMBER$ , not a key, neither a subset of the key)

# Normalization into 3NF



---

- If a relation schema is not in 3NF, it can be *normalized to 3NF*: decomposed into a number of relations in which no nonprime attributes are transitively dependent on the primary key.
- 3NF normalization recipe:
  - Set up a new relation for each nonprime attribute that is transitively dependent on the primary key

# General Definitions of 2NF, 3NF



---

- We want to design relation schemas that do not contain neither *partial* nor *transitive* dependencies
- 2NF normalization disallows partial dependencies
- 3NF normalization disallows transitive dependencies
- These definitions of 2NF & 3NF take into account only the *primary key* and not the candidate keys
- The more general definitions of 2NF & 3NF take into account *all* candidate keys of a relation
- *Prime attribute*: part of any candidate key



# General Definition of 2NF

---

- **R IS IN 2NF IF EVERY NONPRIME ATTRIBUTE IS FULLY FUNCTIONALLY DEPENDENT ON EVERY KEY OF R**
- *Example:* LOTS relation with 2 candidate keys PROPERTY\_ID# and {COUNTY\_NAME, LOT#}
- LOTS violates 2NF because of FD3
- To normalize LOTS in 2NF we decompose it into 2 relations by removing the problematic nonprime attribute
- Both new relations are in 2NF



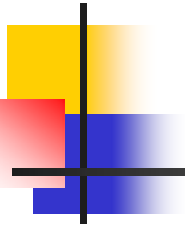
# General Definition of 3NF



---

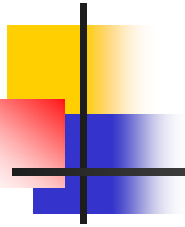
- **R IS IN 3NF IF FOR EVERY FD  $X \rightarrow A$  EITHER (A)  $X$  IS A SUPERKEY OF R OR (B)  $A$  IS A PRIME ATTRIBUTE OF R**
- Example: LOTS2 is in 3NF, LOTS1 is not in 3NF because of FD4 (which gives rise to a transitive FD)
- To normalize LOTS1 in 3NF we decompose it into 2 relations by removing the problematic nonprime attribute together with the lhs of FD4

# Boyce-Codd Normal Form BCNF



- Stronger requirement than 3NF
- Every relation in BCNF is also in 3NF, but not vice-versa
- Motivating Example: LOTS relation with FD1...FD4
  - Suppose there are 1000s of lots but from only two counties
  - (Suppose that lot sizes from county1 are 0.5, 0.6, 0.7 and lot sizes from county2 are 1.2, 1.5, 1.8, 2.1) → we have an additional FD  
FD5:  $AREA \rightarrow COUNTY\_NAME$  however, 3NF is not violated
  - Since there are only 7 possible area values, FD5 could be represented in a separate relation **R(AREA,COUNTY\_NAME)** to avoid repeating the same information the 1000s of tuples

# BCNF



- **R IS IN BCNF IF FOR EVERY FD  $X \rightarrow A$ ,  $X$  IS A SUPERKEY OF R**
- FD5 violates BCNF (AREA is not a superkey)
- To normalize LOTS1A in BCNF we decompose it onto two relations by removing the problematic FD