

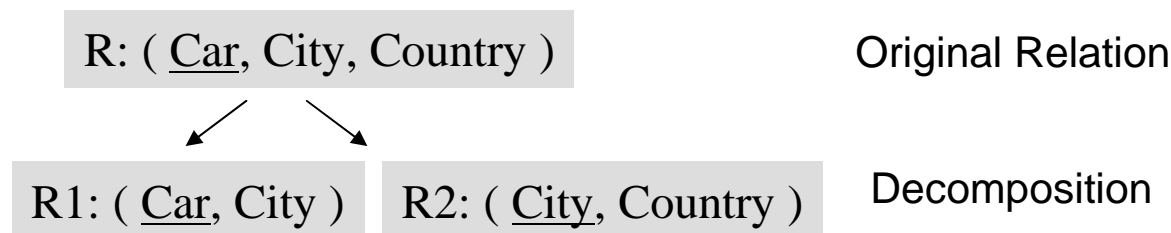
Schema Refinement

Book: Chapter 19



Schema Refinement and Normal Forms

- ER model design yields conceptual schema
- Refinement of conceptual schema: Integrity Constraints
 - So far: Key constraints, Participation Constraints
- Here: **Functional Dependencies (FDs)**
- Problem: Relations with Redundancy
- Solution: Decomposition ... and their properties
- Normal Forms (1NF, 2NF, 3NF, BCNF)
- Example:



Part I: Schema Decompositions and Redundancy Avoidance



Introduction: Redundancy Problems

- Redundant Storage (nowadays not very important)
- Update Anomalies
 - Update of one copy of repeated data creates inconsistency unless all copies are similarly updated as well
- Insertion Anomalies
 - It may be impossible to add a tuple to the database unless some other tuple is created or updated as well.
- Deletion Anomalies
 - It may be impossible to delete a tuple without losing some information.



Redundancy Problems: Example

- Hourly_Emps(ssn, name, lot, rating, hourly_wages, hours_worked)

<i>ssn</i>	<i>name</i>	<i>lot</i>	<i>rating</i>	<i>hourly_wages</i>	<i>hours_worked</i>
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

↑
Key

- FD: *hourly_wages* determined uniquely by *rating*
- Short schema notation: (SNLRWH)
- FD notation: $R \rightarrow W$ (“R determines W” OR: “W is FD on R”)



Decomposition: Example

- In a table one attribute (or several attributes) is a function of one (or several) other attributes: this often implies redundancy
- *Decomposition* of relation schema R:

- Replacing R by two (or more) relation schemas that each contain a subset of the attributes of R and
- Together include all attributes of R.

- Hourly_Emps2(ssn, name, lot, rating, hours_worked)
Wages(rating, hourly_wages)

<i>ssn</i>	<i>name</i>	<i>lot</i>	<i>rating</i>	<i>hours_worked</i>
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

<i>Rating</i>	<i>Hourly_wages</i>
8	10
5	7



Decomposition into Normal Forms: What's the point?

- Do we need to decompose a relation ?
 - *Normal forms* have been proposed for relations.
 - Certain kinds of problems cannot arise, if relation is in one of these NF's
- . What properties should a decomposition have?
 - Lossless-join property: Possible to recover any instance of the decomposed relation from instances of the smaller relations.
 - Dependency-preservation property: Possible to enforce any constraint on the original relation by enforcing some constraints on each of the smaller relations.
 - Functional dependencies should be limited to dependencies on key attributes, if possible



Decomposition: Pro's and Con's

- What are the benefits of decompositions into normal forms?
 - Redundancy is avoided: Storage, constraint enforcement
 - Guaranteed properties can be easier exploited by DBMS
- What can be disadvantages of these decompositions?
 - Decompositions can decrease performance (joins !).
 - SQL Queries over many tables can be difficult to read
 - If too many tables are created, overall transparency of conceptual design may be bad.

Trade-off situation: not always a 'best solution' available. However, decomposition into normal form appears often to be a better database design, if the number of tables is not too big.



Part II: Functional Dependencies: a formal approach



Functional Dependencies (FD's)

- Definition: Functional Dependency

Let R denote a relation schema, X, Y nonempty sets of attributes of R.

An instance r of R satisfies FD: $X \rightarrow Y$ iff:

For all tuples t_1, t_2 in r: $t_1.X = t_2.X$ implies $t_1.Y = t_2.Y$

- Example: $AB \rightarrow C$

- AB is NOT a key (FD is not a key constraint) !
- Adding $\langle a1, b1, c2, d1 \rangle$ would violate FD.

- Legal instance of R must satisfy all specified ICs (includes all FDs) !

- FD is statement about all possible legal instances !

- FD is kind of an IC that generalizes the concept of a key.

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1



Keys, Superkeys, and FDs

- **Key:** Is a special case of FD $X \rightarrow Y$
 - Attributes in key corresponds to X
 - X is minimal, i.e. for each $S \subset X$, the FD $S \rightarrow Y$ does not hold
 - Set of **all** attributes corresponds to Y
- **Superkey:** If $X \rightarrow Y$ holds, where
 - set of all attributes corresponds to Y ,
 - there is some subset $V \subseteq X$ such that $V \rightarrow Y$ holds,
 - then X is a superkey.

Intuitively: a key \subseteq a superkey

- A key is always a superkey, but a superkey not always a key! **Why?**



Reasoning about FD's: Example

- Example: $Workers(\underline{ssn}, name, parking-lot, did, since)$
- $ssn \rightarrow did$ holds, as ssn is the key.
- $did \rightarrow parking-lot$ is a given FD.
- This means: Identical ssn value implies identical did value.
Identical did value implies identical $parking-lot$ value.
- Conclusion: $ssn \rightarrow parking-lot$ also holds.
- This FD is *implied* by the other two FD's.
- Interesting questions:
 - (1) How can we find all implied FD's?
=> *closure* of a set of FD's
 - (2) How can we find a minimal set of FD's that implies all others?
=> *minimal cover*



Closure of a Set of FDs

- **Armstrong's Axioms:**

Let X, Y, Z denote sets of attributes over a relation schema R

Reflexivity:

$$X \supseteq Y \Rightarrow X \rightarrow Y$$

Augmentation:

$$X \rightarrow Y \Rightarrow \forall Z : XZ \rightarrow YZ$$

Transitivity:

$$X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z$$

- **Closure** F^+ of F: Set of all FD's implied by a given set F of FD's.



Closure of a Set of FDs

- **Theorem:**

Armstrong's Axioms are **sound**: They generate only FD's in F^+ when applied to a set F of FD's.

Armstrong's Axioms are **complete**: Repeated application of them will generate all FD's in F^+ .

- **Additional (derived) rules:**

Union:

$$X \rightarrow Y \wedge X \rightarrow Z \Rightarrow X \rightarrow YZ$$

Proof: using Armstrong's axioms!

Decomposition:

$$X \rightarrow YZ \Rightarrow X \rightarrow Y \wedge X \rightarrow Z$$

Proof: using Formal Logic!



Reasoning about FD's: Example II

- Example: $CSJDPQV$

Contracts(contractid, supplierid, projectid, deptid, partid, qty, value)

- Known ICs:

- C is a key: $C \rightarrow CSJDPQV$
- Projects purchase a given part using a single contract: $JP \rightarrow C$
- Department purchases at most one part from a supplier: $SD \rightarrow P$

- Derived FD's:

- $JP \rightarrow C, C \rightarrow CSJDPQV$ implies $JP \rightarrow CSJDPQV$ (Why???)
- $SD \rightarrow P$ implies $SDJ \rightarrow JP$ (augmentation)
- $SDJ \rightarrow JP, JP \rightarrow CSJDPQV$ implies $SDJ \rightarrow CSJDPQV$ (transitivity)
- $C \rightarrow C, C \rightarrow S, C \rightarrow J, C \rightarrow D, \dots$ (decomposition)

Note that c jp and sdj are keys



Part III: Normal Forms



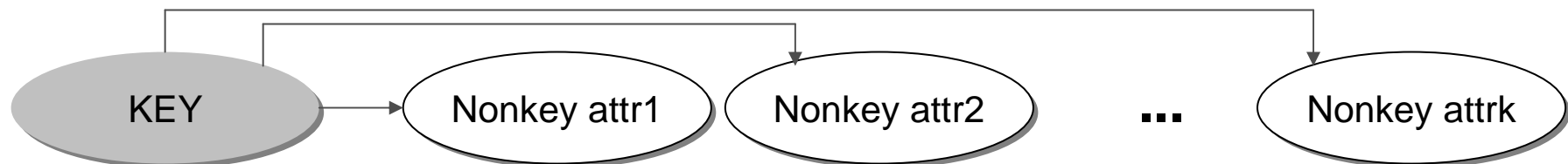
Normal Forms

- Relation schema: Good design? Or decompose it ?
- **Normal forms** provide such guidance.
- First normal form (1NF) - attributes contain only atomic values
- Second normal form (2NF) - 1NF and no attribute is FD on a part of a key
- Third normal form (3NF) - 2NF and no transitive FD's via a non-key attribute
- Boyce-Codd normal form (BCNF) - FD's only on keys
- **Note that:**
 - R is in BCNF implies 3NF implies 2NF implies 1NF.



Boyce-Codd Normal Form

- Let R be a relation schema, F set of FD's over R , X subset of attributes of R , A an attribute of R .
- R is in **Boyce-Codd normal form (BCNF)** if, for every FD $X \rightarrow A$ in F , one of the following statements is true:
 - $A \in X$; i.e., it is a trivial FD, or
 - X is a superkey.
- This means:
 - The only nontrivial dependencies are those in which a key determines some attribute(s).
 - No redundancy can be detected using FD information alone.



Third Normal Form

- Let R be a relation schema, F set of FDs over R , X subset of attributes of R , A an attribute of R .
- R is in **third normal form (3NF)** if, for every FD $X \rightarrow A$ in F , one of the following statements is true:
 - $A \in X$; i.e., it is a trivial FD, or
 - X is a superkey, or
 - A is part of some key for R .
- Note:
 - A key is a *minimal* set of attributes that uniquely determines all other attributes.
 - It is not enough for A to be part of a superkey (that would be satisfied by every attribute!).
 - There is no transitive FD via a non-key attribute
- Motivation for 3NF: See later (definition of decomposition properties needed)



Second Normal Form

- Let R be a relation schema, F set of FDs over R , X subset of attributes of R , A an attribute of R .
- R is in second **normal form (2NF)** if, for every FD $X \rightarrow A$ in F , one of the following statements is true:
 - $A \in X$; i.e., it is a trivial FD, or
 - X is a superkey, or
 - A is part of some key for R , or
 - X and A are both non-key attributes (then it is a transitive FD)
- This means: there is no FD $X \rightarrow A$ where X is a part of a key.



Summary of all important NF's

- **1NF**: no multivalued attributes
- **2NF**: 1NF and: no FD's of non-key attributes on a part of a key
- **3NF**: 2NF and: no transitive FD's via a non-key attribute
- **BCNF**: each key attribute is only FD on all keys
(so no other non-trivial FD's are present)

Note: in 3 NF there might be a FD between key-attributes that are parts of keys, in BCNF not.



NF Example 1

Given the relation:

order (onr, cnr, date, cname, pnr, pname, qty).

Note that onr is a key so all attributes are FD on onr.

Besides this the following FD's are given:

$(\text{cnr}, \text{date}) \rightarrow \text{onr}$; $\text{cnr} \rightarrow \text{cname}$; $\text{pnr} \rightarrow \text{pname}$.

Question: In which NF is this relation? Why?



NF Example 2

Given the relation:

order (onr, cnr, date, pnr, pname, qty).

Note that onr is a key so all attributes are FD on onr.

Besides this the following FD's are given:

(cnr, date) \rightarrow onr; pnr \rightarrow pname.

Question: In which NF is this relation? Why?



NF Example 3

Given the relation:

order (onr, cnr, date, cname, pnr, qty).

Note that onr is a key so all attributes are FD on onr.

Besides this the following FD's are given:

$(\text{cnr}, \text{date}) \rightarrow \text{onr}$; $(\text{cname}, \text{date}) \rightarrow \text{onr}$; $\text{cnr} \rightarrow \text{cname}$.

Question: In which NF is this relation? Why?



Decomposition Properties: Lossless-Join

- Let R be a relation schema, F a set of FD's over R.
- Decomposition of R into 2 schemas with attribute sets X, Y is called a **lossless-join decomposition** with respect to F, iff for every instance r of R that satisfies F:

$$\pi_X(r) \bowtie \pi_Y(r) = r$$

- I.e., it is possible to recover the original relation from the decomposed ones.

- Example:

Lossy decomp.

S	P	D
s1	p1	d1
s2	p2	d2
s3	p1	d3

S	P
s1	p1
s2	p2
s3	p1

$$\pi_{SP}(r)$$

P	D
p1	d1
p2	d2
p1	d3

$$\pi_{PD}(r)$$

S	P	D
s1	p1	d1
s2	p2	d2
s3	p1	d3
s1	p1	d3
s3	p1	d1

$$\pi_{SP}(r) \bowtie \pi_{PD}(r) \neq r$$



Decomposition Properties: Lossless-Join

- All decompositions used to eliminate redundancies must be lossless-join !

- Theorem:

The decomposition of R into relations with attribute sets R_1 and R_2 is lossless-join if and only if F^+ contains either one of these FDs:

$$R_1 \cap R_2 \rightarrow R_1$$

$$R_1 \cap R_2 \rightarrow R_2$$

Common
attribute
set

This means: the attributes of R_1 and R_2 must contain a key in R_1 or R_2

- Example: Hourly_Emps (*SNLRWH*), FD: $R \rightarrow W$

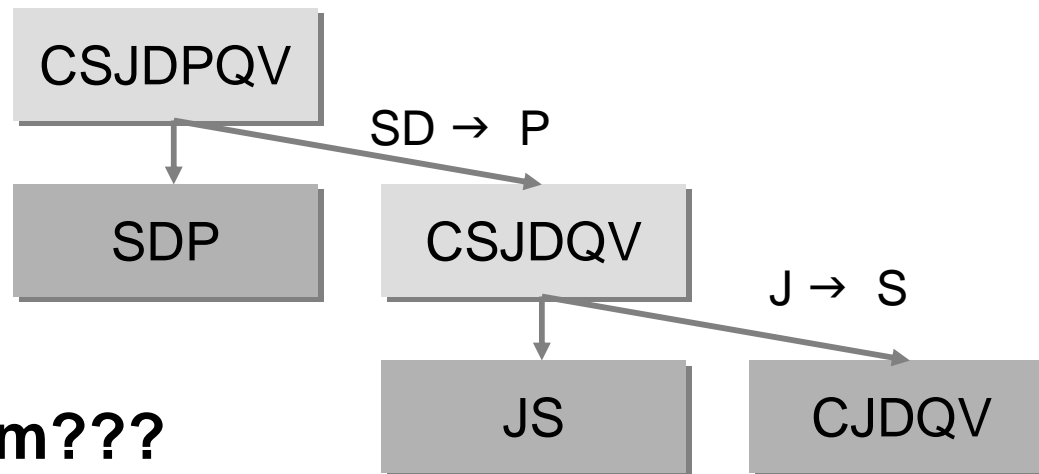
Decomposition *SNLRH*, *RW* is lossless-join:

- R is common to both decomposed relations
- $R \rightarrow W$ holds



Decomposition into BCNF

- Algorithm:
 - Suppose R is not in BCNF. Let $X \subset R$, A a single attribute of R , and $X \rightarrow A$ an FD that causes a violation of BCNF. Decompose R into $R - A$ and XA .
 - If either $R - A$ or XA is not in BCNF, decompose them further by a recursive application of this algorithm.
- Example: Contracts relation, $\underline{C}SJD PQV$, $J \rightarrow S$, $SD \rightarrow P$, $JP \rightarrow C$



Problem???



Decomposition Properties: Dependency-Preserving

- Definition: A **dependency-preserving decomposition** allows us to enforce all FD's by examining a single relation instance on each insertion / modification of a tuple.

- Theorem:

The decomposition of R into relations with attribute sets X and Y is dependency-preserving if $(F_X \cup F_Y)^+ = F^+$

- F_X is the set of all FD's in F^+ that involve only attributes in X.
- Meaning:
 - Taking the dependencies in F_X and F_Y and computing the closure of their union gets us all dependencies in the closure of F back.
 - Only dependencies in F_X and F_Y need to be enforced; all FD's in F^+ are then surely satisfied.



3rd Normal form: What's the point?

- There is not always a lossless join decomposition in BCNF which is also dependency preserving.
- The 3NF weakens the condition of the BCNF only slightly.
- There is always a lossless join decomposition into 3NF which is also ***dependency preserving*** w.r.t. F.

- There is a simple algorithm to determine the 3NF (without the BCNF problem namely the non-dependency preserving):
 1. begin with 1NF (no multivalued attributes)
 2. Then 2NF (no non-key attribute is FD on a part of a key)
 3. Then 3NF (no transitive FD's)



Decomposition into 3NF: Minimal Cover

- Algorithm for BCNF gives 3NF as well, but: does not ensure dependency-preservation.
- **Minimal Cover** for a set F of FD's is a set G of FD's such that:
 - Every FD in G is of the form $X \rightarrow A$ where A is a single attribute.
 - $F^+ = G^+$
 - For any set of FD's H obtained from G by deleting one or more dependencies or by deleting attributes from a dependency, then $F^+ \neq H^+$
- Intuitively: Minimal in two respects
 - Every dependency as small as possible (i.e., every attribute on the left side is necessary, and the right side is always a single attribute).
 - Every dependency in it is required for the closure to equal F^+ .



Decomposition into 3NF: Minimal Cover Algorithm

- Algorithm:
 - **Put the FD's in standard form**
Obtain a collection G of equivalent non-trivial FD's with a single attribute on the right side (decomposition axiom).
 - **Minimize the left side of each FD**
For each FD in G , check each left-side attribute to see if it can be deleted (preserving equivalence to F^+).
 - **Delete redundant FD's**
Check each remaining FD in G to see if it can be deleted (preserving equivalence to F^+).
- Steps need to be executed in this order.



Decomposition into 3NF: Minimal Cover Example

- Example: $A \rightarrow B, ABCD \rightarrow E, EF \rightarrow G, EF \rightarrow H, ACDF \rightarrow EG$
- Rewriting $ACDF \rightarrow EG$: $ACDF \rightarrow E, ACDF \rightarrow G$
- $ACDF \rightarrow G$ is implied by $A \rightarrow B, ABCD \rightarrow E, EF \rightarrow G$
Therefore: Delete it !
- Similarly, delete $ACDF \rightarrow E$
- Replace $ABCD \rightarrow E$ by $ACD \rightarrow E$ (because $A \rightarrow B$).
- Final minimal cover:
 $A \rightarrow B$
 $ACD \rightarrow E$
 $EF \rightarrow G$
 $EF \rightarrow H$



Dependency-Preserving Decomposition into 3NF

- Let R be a relation, set F of FDs that is a minimal cover, R_1, \dots, R_n a lossless-join decomposition of R (each R_i in 3NF).
- Let F_i denote the projection of F onto the attributes of R_i . Then:
 - Identify the set N of dependencies in F that is not preserved (i.e., not included in the closure of the union of F_i s).
 - For each FD $X \rightarrow A$ in N , create a relation schema XA and add it to the decomposition of R .

- Example: $\underline{C}SJDPQV, SD \rightarrow P, JP \rightarrow C, J \rightarrow S$
Decomposition: $\underline{S}DP, \underline{J}S, \underline{C}JDQV$ is lossless-join, all in BCNF.
Dependency $JP \rightarrow C$ is not preserved!
Solution: Add relation schema $\underline{J}PC$.



Summary

- Main purpose of schema normalization: avoidance of redundancies that can be problematic.
 - Reducing the number of functional dependencies within one table is main idea to avoid redundancy; but we may lose efficiency and transparency.
 - Armstrong's axioms provide complete basis for reasoning with functional dependencies.
 - Normal forms guarantee that functional dependencies only involve key attributes.
 - Decompositions can be lossless join and dependency preserving w.r.t. a set of functional dependencies.
 - A lossless join decomposition into BCNF can always be obtained using the decomposition algorithm.
 - A lossless join and dependency preserving decomposition into 3NF can be obtained by following a systematic procedure:
 - first a minimal cover is obtained and
 - then tables are added until all dependencies are preserved
- Or by decomposing via 1NF, 2NF and 3NF, like in slide 29 of this ppt.

